Principles of Communication Systems Simulation with Wireless Applications

William H. Tranter
K. Sam Shanmugan
Theodore S. Rappaport
Kurt L. Kosbar

PRENTICE HALL
Professional Technical Reference
Upper Saddle River, New Jersey 07458
www.phptr.com

Library of Congress Cataloging-in-Publication Data
Principles of communication systems simulation with wireless applications / William H. Tranter ...[et al.]
p. cm. – (Prentice Hall communications engineering and emerging technologies series ; 16)
Includes bibliographical references and index.
ISBN 0-13-494790-8
1. Telecommunication systems–Computer simulation. I. Tranter, William H. II. Series.
TK\5102.5.P673 2003
621.382'01'1–dc22
2003063403

Editorial/production supervision: *Kerry Reardon*
Composition: *Lori Hughes and TIPS Technical Publishing, Inc.*
Cover design director: *Jerry Votta*
Cover design: *Nina Scuderi*
Art director: *Gail Cocker-Bogusz*
Manufacturing manager: *Alexis Heydt-Long*
Manufacturing buyer: *Maura Zaldivar*
Publisher: *Bernard Goodwin*
Editorial assistant: *Michelle Vincenti*
Marketing manager: *Dan DePasquale*
Full-service production manager: *Anne R. Garcia*

Prentice Hall Professional Technical Reference
Upper Saddle River, NJ 07458

Prentice Hall PTR offers excellent discounts on this book when ordered in quantity for bulk purchases of special sales. For more information, please contact: U.S. Corporate and Government Sales, 1-800-382-3419, corpsales@pearsontechgroup.com. For sales outside of the U.S., please contact: International Sales, 1-317-581-3793, international@pearsontechgroup.com

Printed in the United States of America

First printing

ISBN 0-13-494790-8

Pearson Education LTD.
Pearson Education Australia PTY, Limited
Pearson Education Singapore, Pte. Ltd.
Pearson Education North Asia Ltd.
Pearson Education Canada, Ltd.
Pearson Education de Mexico, S.A. de C.V.
Pearson Education-Japan
Pearson Education Malaysia, Pte. Ltd.

To my loving and supportive wife Judy.
William H. Tranter

To my loving wife Radha.
K. Sam Shanmugan

To my loving wife, our children, and my former students.
Theodore S. Rappaport

To my wife and children.
Kurt L. Kosbar

CONTENTS

PREFACE

This book is a result of the recent rapid advances in two related technologies: communications and computers. Over the past few decades, communication systems have increased in complexity to the point where system design and performance analysis can no longer be conducted without a significant level of computer support. Many of the communication systems of fifty years ago were either power or noise limited. A significant degrading effect in many of these systems was thermal noise, which was modeled using the additive Gaussian noise channel. Many modern communication systems, however, such as the wireless cellular system, operate in environments that are interference and bandwidth limited. In addition, the desire for wideband channels and miniature components pushes transmission frequencies into the gigahertz range, where propagation characteristics are more complicated and multipath-induced fading is a common problem. In order to combat these effects, complex receiver structures, such as those using complicated synchronization structures, demodulators and symbol estimators, and RAKE processors, are often used. Many of these systems are not analytically tractable using non-computer based techniques, and simulation is often necessary for the design and analysis of these systems.

The same advances in technology that made modern communication systems possible, namely microprocessors and DSP techniques, also provided us with high-speed digital computers. The modern workstation and personal computer (PC) have computational capabilities greatly exceeding the mainframe computers used just a few years ago. In addition, modern workstations and PCs are inexpensive and therefore available at the desktop of design engineers. As a result, simulation-based design and analysis techniques are practical tools widely used throughout the communications industry.

As a result, graduate-level courses dealing with simulation-based design and analysis of communication systems are becoming more common. Students derive a number of benefits from these courses. Through the use of simulation, students in communications courses can study the operating characteristics of systems that are more complex and more real world than those studied in traditional communications courses since, in traditional courses, complexity must be constrained to ensure that analyses can be conducted. Simulation allows system parameters to be easily changed, and the impact of these changes can be rapidly evaluated by

using interactive and visual displays of simulation results. In addition, an understanding of simulation techniques supports the research programs of many graduate students working in the communications area. Finally, students going into the communications industry upon graduation have skills needed by industry. This book is intended to support these courses.

A number of the applications and examples discussed in this book are targeted to wireless communication systems. This was done for several reasons. First, many students studying communications will eventually work in the wireless industry. Also, a significant number of graduate students pursuing university-based research are working on problems related to wireless communications. Finally, as a result of the high level of interest in wireless communications, many graduate programs contain courses in wireless communications. This book is designed to support, at least in part, these courses, as well as the self-study needs of the working engineer.

This book is divided into three major sections. The first section, Introduction, consists of two chapters. The first of these introductory chapters discusses the motivation for using simulation in both the analysis and the design process. The theory of simulation is shown to draw on several classic fields of study such as number theory, probability theory, stochastic processes, and digital signal processing, to name only a few. We hope that students will appreciate that the study of simulation ties together, or unifies, material from a number of separate areas of study. Different types of simulations are discussed, as well as software packages used for simulation. The development of appropriate simulation models and simulation methodology is a basic theme of this book, and the basic concepts of model development are introduced in Chapters 1 and 2. Chapter 2 focuses on methodology at a very high level. Many of the basic concepts used throughout the book are introduced here. Students are encouraged to revisit this material frequently as the remainder of the book is studied. Revisiting this material will help ensure that the big picture remains in focus as specific concepts are explored in detail.

The second section, Fundamental Techniques, consists of nine chapters (Chapters 3-11). These nine chapters present basic material encountered in almost all simulations of communication systems. The sampling theorem, and the role of the sampling theorem in simulation, is the subject of Chapter 3. Also covered are quantization, pulse shaping, and the effect of pulse shaping on the required sampling frequency. The representation of bandpass signals by quadrature lowpass signals, which is a fundamental tool of simulation methodology, is the subject of Chapter 4. This is a key chapter, in that the techniques presented here will be used repeatedly throughout the book. Filter models and simulation techniques for digital filters are the subject of Chapter 5. Filters, of course, have memory, and more computation is required to simulate filters than most other functional blocks in a system. As a result, filters must be efficiently simulated if reasonable run times are to be achieved. The simulation of a phase-locked loop is presented as a case study in Chapter 6. The student should realize that, even though this material is presented early in the study of simulation, important problems can be investigated with the tools developed to this point. This case study focuses on the acquisition behavior of the phase-locked loop. Acquisition studies require the use of nonlinear models and,

as a result, analysis is very difficult using traditional techniques. The methodology used to develop the simulation is presented in detail, and serves as a guide to the simulations developed later in the book. The simulation techniques for generating random numbers are the subject of Chapter 7. Initially, the focus is on the generation of a pseudo-random sequence having a uniform probability density function (pdf). Both linear conguential methods and techniques based on pseudo-noise (PN) sequences are included. A number of methods for shaping the pdf and PSD of a random sequence are presented. Postprocessing, which is the manipulation of the data generated by a simulation into desired forms for visualization and analysis, is the subject of Chapter 8. Monte Carlo simulation techniques are introduced in Chapter 9 as a general tool for estimating the value of a parameter. The concept of unbiased and consistent estimators is introduced, and the convergence properties of estimators is investigated. The concepts developed in Chapter 9 are applied to communications systems in Chapter 10, which is devoted to Monte Carlo and semianalytic simulation of communication systems. Several simple examples are presented in this chapter. Chapter 11 discusses in detail the methodology used for simulating a wireless communications system in a slowly-varying environment. The calculation of the outage probability is emphasized, and a number of semianalytic techniques are presented for reducing the simulation run time.

The third section of this book, Advanced Models and Simulation Techniques, is devoted to a number of specialized topics encountered when developing more advanced simulations. Chapter 12 is devoted to the simulation of nonlinear systems. Model development based on measurements is emphasized, and a number of models that have found widespread use are presented. Chapter 13 deals with time-varying systems. The important subject of modeling time-varying channels is introduced. Chapter 14 presents a number of models for waveform channels. Drawing on the material presented in the preceding chapter, models for multipath fading channels are developed. Chapter 15 continues the study of channel models, and presents techniques for replacing waveform channel models with discrete channel models at the symbol level. The motivation is a significant reduction in the required simulation run time. The principal tools used are the Baum-Welch algorithm and the hidden Markov model. System models based on the hidden Markov model are presented. Chapter 16 deals with various strategies for reducing the variance of a bit error rate estimator. Several strategies are presented, but the emphasis is on importance sampling. Chapter 17 is devoted to the simulation of wireless cellular communication systems. It is shown that cellular systems tend to be interference limited rather than noise limited. In many systems, co-channel interference is a major degrading effect. Chapter 18 concludes the book with two example simulations. The first of these considers a CDMA system, and presents a simulation in which the bit error rate is computed as a function of the spread-spectrum processing gain, the number of interferers, the power-delay profile, and the signal-to-noise ratio at the receiver input. The data collected by the simulation is used to construct a discrete channel model based on the hidden Markov model. The hidden Markov model is then used to statistically reconstruct the error events on the channel. The BER is then computed using the discrete channel model, and the results are compared with

the results obtained using a waveform-level channel model. The second example is an FDM system operating over a nonlinear channel. The effect of intermodulation distortion on bit error rate is investigated using semianalytic techniques.

From the preceding discussion, it is clear that this book covers a very wide range of topics. A completely rigorous treatment of all of the topics considered here would require a volume many times the size of this book, and the result would not be suitable as a course textbook. A compromise between completeness and rigor must always be made in developing a textbook. We have, in developing this book, attempted to provide sufficient rigor to make the results both understandable and believable. A large number of references are given for those wishing additional study.

Although this book is targeted to a one-semester course in communications, there is more material included here than is typically covered in a one-semester course. In the view of the authors, all courses using this book should cover the first two sections (Chapters 1-11). The instructor can then complete the course with selected material from the third section (Chapters 12-18), assuming that time is available.

A number of computer programs, written in MATLAB, are included in the text. The decision to include computer code within the body of the book was made for a number of reasons. First, the programs illustrate the methodology used to develop simulations, and illustrate the algorithms used to perform a number of important DSP operations. In addition, many code segments included in the MATLAB examples can be used by the student to aid the development of their own simulations. In order not to break the flow of the material, only short programs, those requiring no more than a single page of text, are included within the body of the chapters. Programs that are too long to fit on a single page are placed in appendices at the end of the chapter. The MATLAB code included here is designed to be easily followed by the student. For that reason, a number of the MATLAB programs are not written in the most efficient manner possible, in that for-loops are often used when the loop could be replaced by a matrix operation. It is not suggested that the student type the computer code from the text. A web page is maintained by Prentice Hall containing all of the computer code included in the text, and code can be downloaded from this site. The URL is

http://authors.phptr.com/tranter/

The MATLAB code on this site will be updated periodically in order to ensure that errors and omissions are corrected.

The choice of MATLAB requires some explanation. There are a number of reasons for this choice, and these are discussed in detail in Chapter 1. The main motivations are compactness (complex algorithms can be expressed with a very few lines of code), graphics support, and the installed base. Since MATLAB is used extensively in engineering curricula, most students will already have the resources required to execute the MATLAB programs contained herein. Many simulation programs involve extensive computational burden, and reasonable execution run times require the use of a compiled language such as C or C++. This is especially

true of Monte Carlo simulations used to estimate the bit error rate when the signal-to-noise ratio is high. Many symbols must be processed through the channel in order to achieve a quality (low variance) estimator. MATLAB, however, is a powerful tool even in this situation, since a prototype simulation can be developed in MATLAB to design and test the individual signal-processing algorithms, as well as the entire simulation. The resulting MATLAB code can then be mapped to C or C++ code for more efficient execution, and the results obtained can be compared against the results achieved with the prototype MATLAB simulation. Using MATLAB for prototyping allows conceptual errors to be quickly identified, which often speeds the development of the final software product. SIMULINK, although designed for simulation, was not used in this book, so that the details of the algorithms used in simulation programs, and the methodology used to develop the simulation code, would be clear to the students.

ACKNOWLEDGMENTS

A number of colleagues, research sponsors, and organizations have contributed significantly to this book. Early in this project a CRCD (Combined Research Curriculum Development) grant was awarded to Virginia Tech by the National Science Foundation. Much of the material in Chapters 3-10 and Chapter 17 was developed as a part of this effort. The NSF program manager Mary Poats, encouraged us to develop simulation-based courses within the communications curriculum, and we thank her for the encouragement and support. The authors thank Cyndy Graham of Virginia Tech for her LaTeX skills, and for managing the development of such a large manuscript. In addition, the individual authors have the following specific acknowledgements:

William H. Tranter thanks the many students who took the simulation of communications systems course at the University of Missouri–Rolla, Canterbury University (Christchurch, New Zealand), and at Virginia Tech from the notes that formed the basis of much of this book. These students provided many valuable suggestions. Specific thanks are due to Jing Jiang, who helped with the semianalytic estimators in Chapter 10; Ihsan Akbar, who did much of the coding of the Markov and semi-Markov model estimators in Chapter 15 (especially the code contained in Appendices B, C, and D); and Bob Boyle, who developed the CDMA estimator upon which the CDMA case study in Chapter 18 is based. He also thanks Sam Shanmugan, who provided friendship, support, encouragement, and above all patience, through the years that it took to bring this material together. Also to be thanked are Des Taylor and Richard Duke, who provided support through an Erskine Fellowship at Canterbury University, and Theodore Rappaport at Virginia Tech, who provided support during a sabbatical year. It was during this sabbatical that much of the material in the early chapters of this book were originally drafted.

Sam Shanmugan would like to thank his colleagues and students at the University of Kansas, who have in many ways contributed to this book, and also the University of Canterbury, Christchurch, New Zealand for the Erskine Professorship during his sabbatical when much of this book was written. He would also like to thank his wife for her patience, understanding, and support while he was working

on this and on many other writing projects. Dr. Shanmugan would like to add a special note of thanks to his co-author Professor William Tranter, for his friendship and the extra effort he put in towards pulling together all the material for this book.

Ted Rappaport wishes to thank his many graduate students who provided insights and support through their teaching and research activites in wireless communications simulation and analysis. In particular, Prof. Paulo Cardieri, University of Campinas—UNICAMP, Brazil; Hao Xu of QUALCOMM Incorporated; and Prof. Gregory Durgin of the Georgia Institute of Technology, all contributed suggestions for the text. In particular, Dr. Cardieri's experiences as a graduate student researcher formed the basis of Chapter 17.

Kurt Kosbar thanks the students who screened early versions of this material, and the reviewers who provided valuable comments, including Douglas Bell, Harry Nichols, and David Cunningham.

William H. Tranter
K. Sam Shanmugan
Theodore S. Rappaport
Kurt L. Kosbar

PART I

Introduction

Chapter 1

THE ROLE OF SIMULATION

The complexity of modern communication systems is a driving force behind the widespread use of simulation. This complexity results both from the architecture of modern communication systems and from the environments in which these systems are deployed. Modern communication systems are required to operate at high data rates with constrained power and bandwidth. These conflicting requirements lead to complex modulation and pulse shaping along with error control coding and an increased level of signal processing at the receiver. Synchronization requirements also become more stringent at high data rates and, as a result, receivers become more complex. While the analysis of linear communication systems operating in the presence of additive, white, Gaussian noise is usually quite simple, most modern systems operate in much more hostile environments. Multihop systems often require nonlinear amplifiers for efficiency. Wireless cellular systems often operate in the presence of heavy interference along with multipath and shadowing that leads to signal fading at the receiver site. This combination of complex systems and hostile environments leads to design and analysis problems that are no longer analytically tractable using traditional (nonsimulation-based) techniques.

Fortunately, the past two decades have seen the development of digital computers that are both powerful and inexpensive. Thus, modern computers are suitable for use at the desktop and can therefore be dedicated to the solution of problems taking many hours of computer time without interfering with the work of others. Computers have become easy to use, and the cost of computer resources is no longer

a significant factor in many efforts. As a result, computer-aided design and analysis techniques are available to almost all who need them. The development of powerful software packages targeted to communication systems has accelerated the use of simulation in the communications area. Thus, the increase in system complexity has been accompanied by an increase in computing power. In many cases, the availability of appropriate computational power has directly led to many of the complex signal-processing structures that constitute the building blocks of modern communication systems. Thus, it is not just good luck that computational tools appeared at the time they were needed. Rather, practical computational power, in the form of the microprocessor, is the enabling technology for modern communication systems and is also the enabling technology for powerful simulation engines.

The growth in computer technology has also been accompanied by a rapid growth in what we loosely refer to as simulation theory. As a result, the tools and methodologies required for the successful application of simulation to design and analysis problems are more accessible and better understood than was the case a few decades ago. A large number of technical papers and several books are now available that illustrate the application of these tools to the design and analysis of communication systems.

An important motivation for the use of simulation is that simulation is a valuable tool for gaining insight into system behavior. A properly developed simulation is much like a laboratory implementation of a system. Measurements can easily be made at various points in the system under study. Parametric studies are easily conducted, since parameter values, such as filter bandwidths and signal-to-noise ratios (SNRs), can be changed at will and the effects of these changes on system performance can quickly be observed. Time-domain waveforms, signal spectra, eye diagrams, signal constellations, histograms, and many other graphical displays can easily be generated and, if desired, a comparison can be made between these graphical products and the equivalent displays generated by system hardware. We will see that the process of comparing simulation results with hardware-generated results is an important part of the design process. Most importantly, perhaps, one can perform "what if" studies more easily and economically using a simulation than with actual system hardware. Although we often perform a simulation to obtain a number, such as a bit error rate (BER), the main role of simulation, as noted by R. W. Hamming, is not to obtain numbers but to gain insight.

1.1 Examples of Complexity

The complexity of communication systems varies widely. We now consider three communications systems of increasing complexity. We will see that for the first system, simulation is not necessary. For the second system, simulation, while not necessary, may be useful. For the third system, simulation is necessary in order to conduct detailed performance studies. Even the most complicated of the three systems considered here is still simple by today's standard.

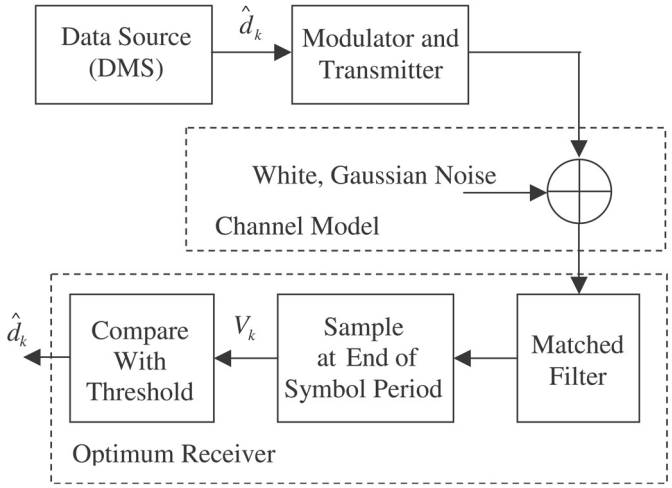


Figure 1.1 Analytically tractable communications system.

1.1.1 The Analytically Tractable System

A very simple communications system is shown in Figure 1.1. This system should remind us of the basic communications system studied in a first course on communications theory. The data source generates a sequence of symbols, d_k. The symbols are assumed to be discrete. The source symbols are assumed to be elements from a finite symbol library. For a binary communication system, the source alphabet consists of two symbols, which are usually denoted $\{0, 1\}$. In addition, the source is assumed to be memoryless, which means that the k^{th} symbol generated by the source is independent from all other symbols generated by the source. A data source satisfying these properties is referred to as a discrete memoryless source (DMS). The role of the modulator is to map the source symbols onto waveforms, with a different waveform representing each of the source symbols. For a binary system, we have two possible waveforms generated by the modulator. This set of waveforms may be denoted $\{s_1(t), s_2(t)\}$. The transmitter, in this case, is simply assumed to amplify the modulator output so that the signals generated by the modulator are radiated with the desired energy per bit.

The next part of the system is the channel. In general, the channel is the most difficult part of the system to model accurately. Here, however, we will assume that the channel simply adds noise to the transmitted signal. This noise is assumed to have a power spectral density (PSD) that is constant for all frequency. Noise satisfying this constant PSD property is referred to as white noise. The noise amplitude is also assumed to have a Gaussian probability density function. Channels in which the noise is additive, white, and Gaussian are referred to as AWGN channels.

The function of the receiver is to observe the signal at the receiver input and from this observation form an estimate, denoted $\widehat{d}_k$, of the original data signal,

d_k. The receiver illustrated in Figure 1.1 is referred to as an optimum receiver because the estimate of the data symbol is made so that the probability of error, P_E, is minimized. We know from basic digital communication theory that the optimum receiver for the system described in the preceding paragraphs (binary signaling in an AWGN environment) consists of a matched filter (or, equivalently, a correlation receiver), which observes the signal over a symbol period. The output of the matched filter is sampled at the end of a symbol period to generate a statistic, V_k, which is a random variable because of the addition of noise to the transmitted signal in the channel. The statistic, V_k, is compared to a threshold, T. If $V_k > T$ the decision, $\widehat{d}_k$, is made in favor in one of the data symbols. If $V_k < T$ the decision is made in favor of the other data signal.

We refer to this system as an analytically tractable because, with a knowledge of basic communication theory, analysis of the system is carried out with ease. For example, the probability of error is found to be

$$P_E = Q\left(\sqrt{k\frac{E_s}{N_0}}\right) \tag{1.1}$$

where E_s represents the average energy, calculated over a symbol period, associated with the set of waveforms $\{s_1(t), s_2(t)\}$, and N_0 represents the single-sided power spectral density of the additive channel noise. The parameter, k, is determined by the correlation of the waveforms $\{s_1(t), s_2(t)\}$. As an example, for FSK (frequency-shift keying) transmission, the waveforms $\{s_1(t), s_2(t)\}$ are sinusoids having different frequencies and equal power. Assuming that the frequencies are chosen correctly, the signals are uncorrelated and $k = 1$. For the PSK case (phase-shift keying), the signals used for data transmission are assumed to be sinusoids having the same frequencies and equal power but different initial phases. If the phase difference is π radians, so that $s_2(t) = -s_1(t)$, the signals are anticorrelated and $k = 2$.

The performance of the system illustrated in Figure 1.1 is easily determined using traditional analysis techniques, and we are therefore able to classify the system as analytically tractable. Why is this system analytically tractable? The first and most obvious reason deals with the AWGN channel and the fact that the receiver is linear. Since the noise is Gaussian and the matched filter is a linear system, the decision statistic, V_k, is a Gaussian random variable. We are therefore able to calculate the bit error rate (BER) analytically as a function of the parameters of the receiver filter and determine the values of those parameters that result in a minimum BER.

There are a number of other factors leading to the fact that the system shown in Figure 1.1 is analytically tractable. These relate to the simplicity of the system model, which results from a number of assumptions. The data source was assumed memoryless, which may or may not be true in practice. In addition, perfect symbol synchronization was assumed, so that we have exact knowledge of the beginning and ending times of the data symbols. This assumption allows the decision statistic, V_k, to be correctly extracted.

Would simulation ever play a role in an analytically tractable system? The answer is yes, since the system shown in Figure 1.1 may well be the basic building

block of a more complex system. The simulation code can be developed for the system. The resulting simulation can be validated with ease, since analysis of the system is straightforward. At this point the data source, modulator, channel, or receiver can be modified as required to model the system under study. In addition, other subsystems as needed can be added to the simulation model. As we proceed with the task of developing a simulation model of the system of interest, we can be confident that the starting point was correct.

1.1.2 The Analytically Tedious System

We now turn attention to a somewhat more complex system. The system illustrated in Figure 1.2, which is identical to the previously investigated system except for the addition of the nonlinear high-power amplifier (HPA) and filter in the transmitter. Consider first the nonlinear amplifier. Nonlinear amplifiers exhibit much higher power efficiency than linear amplifiers and, as a result, are often preferred over linear amplifiers for use in environments where power is limited. Examples include space applications and mobile cellular systems, where battery power must be conserved. Unlike linear amplifiers, which preserve the spectrum of the input signal, the nonlinear amplifier will generate harmonic and intermodulation distortion. As a result, the spectrum of the amplifier output will be spread over a much larger bandwidth than that occupied by the spectrum of the modulator output. The filter following the amplifier will in most cases be a bandpass filter with a center frequency equal to the desired carrier frequency. The role of the filter is to attenuate the harmonic and intermodulation distortion resulting from the nonlinearity.

The filter following the modulator and HPA leads to time dispersion of the data signal so that the filtered signals are no longer time limited to the symbol period.

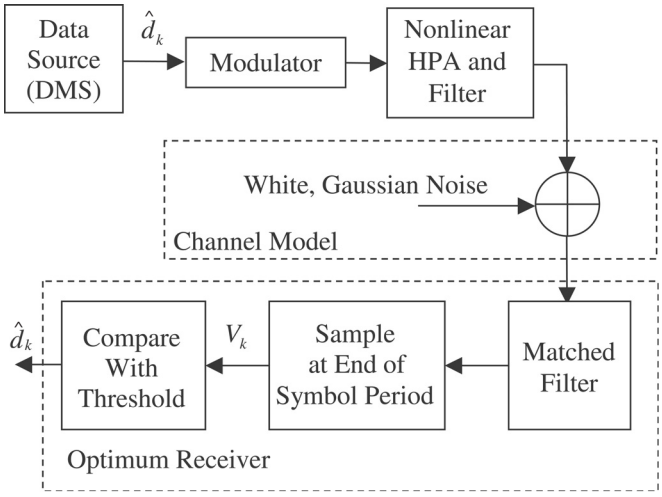


Figure 1.2 Analytically tedious communications system.

This leads to intersymbol interference (ISI). As a result of ISI, the probability of error of the i^{th} symbol is dependent upon one or more of the symbols previous to the symbol upon which the decision is being made. The number of previous symbols that must be considered in the demodulation of the i^{th} symbol depends upon the memory associated with the signal at the filter output. If the probability of error for the i^{th} symbol depends on the k previous symbols we compute the quantity

$$\Pr\{E_i|d_{i-1}d_{i-2}\cdots d_{i-k}\}$$

For the binary case there are 2^k different sequences of length k. Assuming that each data symbol is equally likely to be a binary 0 or 1, the error probability of the i^{th} symbol is given by

$$P_E = \frac{1}{2^k}\sum_{d_{i-1}=0}^{1}\sum_{d_{i-2}=0}^{1}\cdots\sum_{d_{i-k}=0}^{1}\Pr\{E_i|d_{i-1}d_{i-2}\cdots d_{i-k}\} \tag{1.2}$$

In other words, one must compute 2^k different error probabilities, with each error probability dependent upon one of the 2^k preceding sequences of length k, and average the k results. Since the channel is assumed AWGN, each of the 2^k error probabilities is a Gaussian Q-function. It is a straightforward, but tedious procedure to calculate the argument of each Q-function and, therefore, simulation is often used.

The system illustrated in Figure 1.2 has an important property that makes analysis straightforward. Note that the system is linear from the point at which the noise is injected to the point at which the statistic V_k appears. The statistic V_k often takes the form

$$V_k = S_k + I_k + N_k \tag{1.3}$$

where S_k and I_k are the components of V_k due to signal and intersymbol interference, respectively, and N_k is the component of V_k due to the channel noise. Thus, if the channel noise is Gaussian, N_k will be a Gaussian random variable, since it is a linear transformation of a Gaussian random variable. In addition, the decision statistic V_k will be a Gaussian random variable having the same variance as N_k but with mean $S_k + I_k$, both of which are deterministic. The mean of V_k can be computed in a straightforward manner. The variance of V_k is determined from knowledge of the power spectral density of the channel noise and the equivalent noise bandwidth the system from the channel to the point where V_k appears. The pdf of V_k is therefore known and the error probability is easily determined. To summarize, the reason that we can easily determine the pdf of V_k, even though the system has a nonlinearity, is because the noise does not pass through the system nonlinearity.

The fact that the noise passes only through the linear portion of the system has a significant impact on the simulation methodology. Because the noise does not pass through a nonlinearity, the mean of V_k can quickly be determined using a noise-free simulation. The variance of V_k can be determined analytically and, as a result, the pdf of V_k is known and the error probably is easily determined. These concepts are combined in a simulation technique that is both simple and fast. The

result is the semi-analytic method in which analysis and simulation is combined in a way that leads to very fast simulations. Semi-analytic simulation is an important tool and will be the subject of a later chapter.

1.1.3 The Analytically Intractable System

The system illustrated in Figure 1.3 is referred to as an analytically intractable system and is a simple model of a two-hop satellite communications system. The satellite transponder is modeled as a nonlinear HPA and a filter to remove the out-of-band harmonic distortion caused by the nonlinearity. Comparison of Figure 1.3 with Figure 1.2 shows that they are quite similar. A satellite channel model has been added and consists of two noise sources rather than one. One noise source represents the uplink (transmitter-to-satellite) noise, and the other noise source represents the downlink (satellite-to-receiver) noise. The problem lies in the fact that the noise at the receiver consists of two components; the downlink noise and the uplink noise that was passed through the nonlinear HPA. Even assuming that both the uplink and the downlink noise are Gaussian, the pdf of the noise at the receiver is very

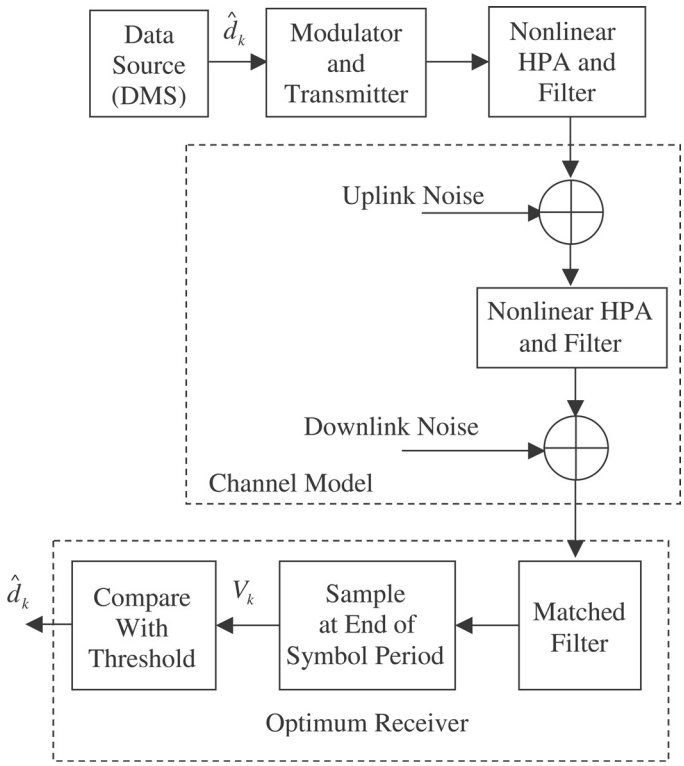


Figure 1.3 Analytically intractable communications system.

difficult to determine. The downlink noise is easy to model, since the downlink noise passes only through the linear portion of the system. The uplink noise, however, leads to difficulties. The reason for the difficulty lies in the fact that the uplink noise passes through the nonlinear HPA. Even if the uplink noise is Gaussian, the pdf of the uplink noise at receiver input is no longer Gaussian. Determination of the pdf of the decision statistic, V_k, is a very difficult, if not impossible, undertaking. Without exact knowledge of the pdf of the decision statistic, the probability of error cannot be determined. Simulation is an essential tool for these types of systems.

The range of communication systems considered in this section has been very narrow. The systems were chosen simply to illustrate how increasing complexity gives rise to the need for simulation. Many systems of current interest fall into the analytically intractable category. Consider, for example, a wireless cellular radio link operating in a high interference and multipath environment. Simulation is almost always necessary for the detailed analysis of such systems.

1.2 Multidisciplinary Aspects of Simulation

Prior to the 1970s simulation problems were often solved in a somewhat ad hoc manner. The methodologies for developing simulations, and the error sources present in all simulation programs, were not understood by many. Over the past 20 years, the research community has produced a body of knowledge that provides a methodology for simulation development and a theoretical framework for solving many of the problems that arise in the development of simulation programs. This body of knowledge provides those using simulation as an analytical tool the insights and understanding necessary to develop reliable simulations that execute in reasonable computer run times. Building this body of knowledge has required the integration of material from a variety of fields. Although not exhaustive, nine important areas of study that impact our study of simulation are depicted in Figure 1.4. We will now briefly look at these nine areas in order to better understand their relationship to the art and science of simulation.

The concepts of linear system theory give us the techniques for determining the input-output relationships of linear systems. This body of knowledge allows us to represent system models in both the time domain (the system impulse response) and in the frequency domain (the system transfer function). The basic concepts of linear system theory builds the foundation for much of what follows.

An understanding of communication theory is obviously important to our study. The architecture of systems, the operational characteristics of various subsystems such as modulators and equalizers, and the details of channel models must be understood prior to the development of a simulation. While simulation can be used to determine appropriate values for system parameters, the practical range of parameter values must usually be known before the simulation is developed. Some insight into proper system behavior is necessary in order to ensure that the simulation is working properly and that the results are reasonable.

The tools of digital signal processing (DSP) are used to develop the algorithms that constitute the simulation model of a communication system. This simulation

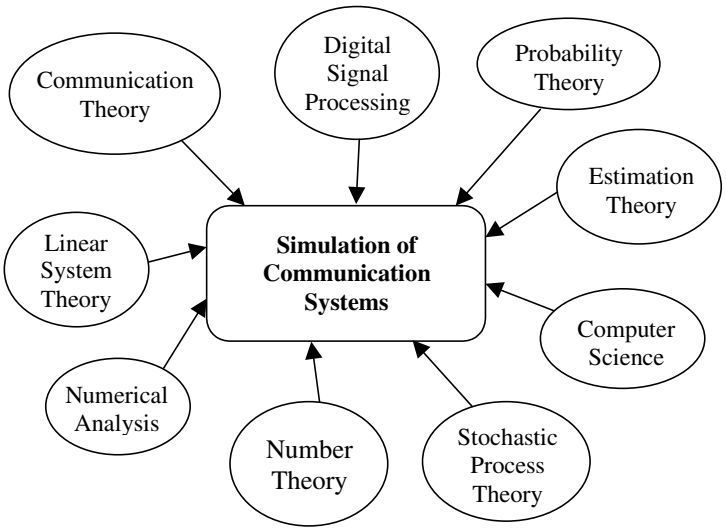


Figure 1.4 Areas impacting the study of the simulation of communication systems.

model usually consists of several discrete-time approximations of continuous-time system components, such as filters, and a knowledge of DSP techniques is necessary to understand and appreciate the nature of these approximations. As a matter of fact, each functional block in a simulation model is a DSP operation and, therefore, the tools of digital signal processing provide the techniques for implementing simulations.

Numerical analysis is closely related to DSP but is mentioned separately, since it is an older discipline. Many classical techniques, such as the suite of tools for numerical integration, polynomial interpolation, and curve fitting have their origins in numerical analysis.

The concepts of probability are also fundamental to our study. The performance measures of communication systems are often expressed in probabilistic terms. As examples, we often have interest in the probability of bit error or symbol error in a digital communication system. In synchronization systems we have interest in the probability that a phase error will exceed a given level. Basic probability theory provides us with the concept of random variables and the probability density function. Knowledge of the underlying probability density function allows us to compute the quantities previously discussed. We will see later that the result of many simulations (called stochastic simulations) is typically a random variable, and the variance of that random variable is often a measure of the usefulness and statistical accuracy of the simulation.

The signal and noise waveforms that are processed by our simulations will, in many cases, be assumed to be sample functions of a stochastic process. Development of the algorithms to produce waveforms having the appropriate statistical properties

will require knowledge of the underlying stochastic process. This is especially true for developing simulation models for channels. Stochastic process theory gives us the tools to describe these processes in the time domain (the autocorrelation function) and in the frequency domain (the power spectral density). Many other applications of stochastic process theory will appear in the course of our work.

A few of the very basic concepts of number theory provide us with the tools used to develop random number generators. These random number generators are the basic building blocks of the waveform generators used to represent digital sequences, noise waveforms, signal fading, and random interference, to name only a few applications.

Some of the basic concepts of computer science will be useful in the course of our study. As examples, the word length, and the format of words, used to represent samples of signals will impact simulation accuracy, although this is often of minimal importance in floating-point processors. The choice of language is important in the development of commercial simulators. Available memory, and the organization of that memory, will impact the manner in which data and instructions are passed from one part of the simulation to another. Graphics requirements and capabilities will determine how waveforms are displayed and will impact the transportability of the simulation code from one computer platform to another.

The tools and concepts of estimation theory will allow us to evaluate the effectiveness of a given simulation result. As mentioned earlier, the result of a stochastic simulation is a random variable. Each execution of the simulation will produce a value of that random variable, and this random variable will constitute an *estimator* of a desired quantity. Typically, all values produced by replications of the simulation will be different. Simulations are most useful when the estimator produced by a simulation is *unbiased* and *consistent*. Unbiased estimators are those for which the average value of the estimate is the quantity being measured. This is another way of saying that *on the average* the estimates produced by the simulation are correct. This is clearly a desired attribute. A consistent estimate is one for which the variance of the estimate decreases as the simulation run length increases. In other words, if 100 independent measurements of the height of a person are made, and the results averaged, we would expect a more accurate estimate of the height than would result from a single measurement. Estimation theory provides us with the analytical tools necessary to explore questions of this type and, in general, to access the reliability of simulation results.

The previous paragraphs are not intended to make a study of simulation appear to be a daunting task. The goal is simply to point out that simulation is a field of study in its own right. It draws from many other fields just as electrical engineering draws from physics, mathematics, and chemistry, to name only a few. It is expected that those embarking on this study have a grasp of linear system theory, communications, and probability theory. Much of the remaining material will be treated in the following chapters of this text.

1.3 Models

The first step in developing a simulation of a communication system is the development of a simulation model for the system of interest. We are all familiar with models and should understand that models describe the input-out relationship of physical systems or devices. These models are typically expressed in mathematical form. The art of modeling is to develop behavioral models (we use this term since the model captures the input-output behavior of the device under specific conditions) that are sufficiently detailed to maintain the essential features of the system being modeled and yet are not overly complex so that the models can be used with reasonable expenditures of computational resources. Tradeoffs between accuracy, complexity, and computational requirements are therefore usually required.

It is useful to consider two different types of models in the work to follow: analytical models and simulation models. Both analytical models and simulation models are abstractions of a physical device or system as illustrated in Figure 1.5. The physical device illustrated in Figure 1.5 may be a single circuit element such as a resistor or a subsystem such as a single chip implementation of a phase-locked loop (PLL) used as a bit synchronizer. It may be a complete communications system. The first and most important step in the modeling process is to identify those attributes and operational characteristics of the physical device that are to be represented in the model. The identification of these essential features often requires considerable engineering judgment and *always requires a thorough understanding of the application for which the model is being developed*. The accuracy required of any mathematical analysis or any computer simulation based on the model is limited by the accuracy of the model. Once these questions have been answered, an analytical model is developed that captures the essential features of the physical device. Analytical models typically take the form of equations, or systems of equations, that define the input-output relationship of the physical device. These

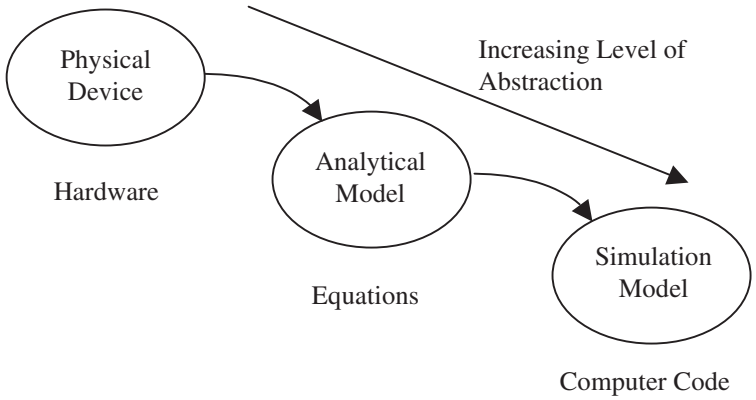


Figure 1.5 Devices and models.

equations are, at best, only a partial description of the device being modeled, since only certain aspects of the device are modeled. In addition, the equations that define the device are typically accurate only over a limited range of voltages, currents, and frequencies. The simulation model is usually a collection of algorithms that implement a numerical solution of the equations defining the analytical model. The techniques of numerical analysis and digital signal processing are the tools used in the development of these algorithms.

We also see from Figure 1.5 that the level of abstraction increases as one moves from the physical device to the analytical model and finally to the simulation model. The increase in abstraction results, in part, from the assumptions and approximations made in moving from the physical device to the analytical model to the simulation model. Every assumption and approximation moves us farther from the physical device and its operating characteristics. In addition, the level of abstraction present at any step in the process is due, in large part, to the representation used for the analytical model. As an example, assume that the physical device being considered is a phase-locked loop. The analytical model for a PLL can take many forms, with each form corresponding to a different level of abstraction. An analytical model having a low level of abstraction could consist of a system of equations, with each equation corresponding a single functional operation within the PLL. Each of these functional, or signal-processing, operations within the PLL (phase detector, loop filter, and voltage-controlled oscillator) is represented by a separately identifiable equation within the system of equations defining the overall PLL. The process and assumptions used in moving from the hardware device to the analytical model are often clear from observation of these equations. In addition, simulations developed from such a system of equations may allow individual signals of interest within the PLL to be observed and compared to corresponding signals in the hardware device. We will see that such comparisons are often an essential part of the design process. On the other hand, the individual equations representing separate signal-processing operations may be combined into a single nonlinear (and perhaps time-varying) differential equation relating the input-output relationship of the PLL, which leads to a much more abstract model. The individual signal-processing operations that take place within the PLL, and the waveforms associated with these operations, are no longer separately identifiable. It might seem logical to consider only analytical models having a low level of abstraction. This, however, is not the case.

Models having different levels of abstraction will be frequently encountered throughout our studies. As another example, we will see that channels may be modeled using a waveform-level approach, in which sample values of waveforms are processed by the model. On the other hand, channels may be represented by a discrete Markov process based on symbols rather than on samples of waveforms. In addition, Markov channel models usually absorb the modulator, transmitter, and receiver into the channel. These models are highly abstract and are difficult to parameterize accurately but, once found, result in numerically efficient simulations that execute rapidly. This efficiency is a principal reason for having interest in the more abstract modeling approaches.

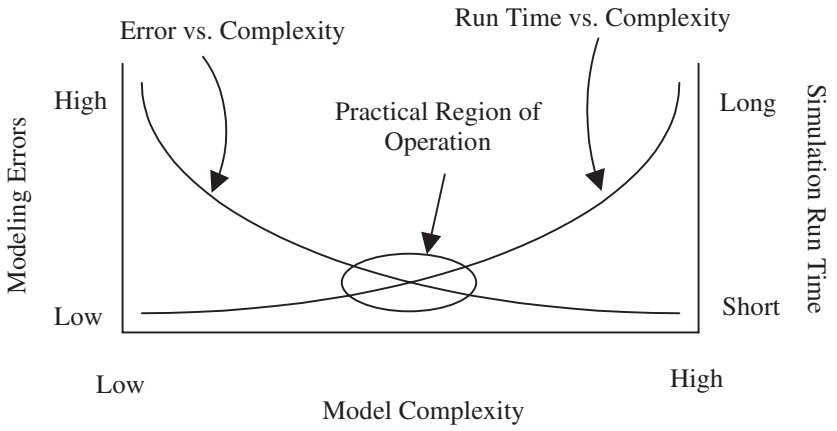


Figure 1.6 Effects of model complexity.

Figure 1.6 also tells us much about the modeling process. It is intuitively obvious that a desirable attribute of a simulation is fast execution of the simulation code. Simple models will execute faster than more complex models, since fewer lines of computer code need to be processed each time the model is invoked by the simulation. Simple models may not, however, fully characterize the important attributes of a device, and therefore the simulation may yield inaccurate results. In such a case, more complex models are necessary. While more complex models may yield more accurate simulation results, the increased accuracy usually comes at the cost of increased simulation run time.

Figure 1.6 makes it clear that the desirable attributes of simulation accuracy and execution speed are in competition. A well-designed simulation is one that provides reasonable accuracy along with reasonable execution speeds. Of course, when the specifications for a simulation demand a high level of accuracy, the ability to trade off accuracy and execution speed becomes severely constrained. In this case the model complexity must be sufficient to guarantee the required accuracy, and long simulation run times become, perhaps, unavoidable.

Figure 1.6 tells only part of the story. More complex models often require that extensive measurements be made before accurate simulation models can be developed. The development of simulation models for a nonlinear amplifier is one example. Another, and even more complex example is the development of a simulation model of a wireless communication channel when multiple interference sources and severe frequency selective fading is present. There are many other cases we could mention in which extensive measurements are required. It should be kept in mind that these measurements require resources (both equipment and engineering time) and therefore a relationship exists between the cost of model development and model complexity. It should also be kept in mind that complex models are more error prone than simple models.

When we move from an analytical model to a discrete-time (digital) simulation model, additional assumptions and approximations are involved. At this point we mention only a few of the most obvious. The voltages and currents present in both the physical device and in the analytical model are usually considered to be continuous functions of the continuous variable time. In moving from the analytical model to the simulation model, we move from the continuous domain to the discrete domain. This process involves quantizing the amplitudes of the voltages and currents and time sampling these quantities. The process of time sampling leads to aliasing errors, and quantizing amplitudes leads to quantizing errors. While quantizing errors are often negligible in simulations performed on floating-point processors, aliasing errors require our attention if the sampling frequency for the simulation is to be selected appropriately. Aliasing errors are reduced by increasing the sampling frequency, but an increased sampling frequency results in more samples being required to represent a given segment of data. The result is that more samples must be processed in order to execute the simulation, and the time necessary to execute the simulation is thereby increased. Hence, a tradeoff therefore exists between sampling frequency and simulation run time. One therefore should not attempt to eliminate aliasing errors, or most other errors for that matter, but rather should seek a simulation having the required accuracy with reasonable run times.

The modeling concepts briefly touched on here will be revisited in more detail in the following chapter and will be encountered many times throughout this book. The purpose of this brief introduction is simply to remind the reader that we deal not with physical devices but with models in performing any engineering analysis. Analytical models (equations) are abstractions of physical devices and involve many assumptions and approximations. Simulation models are based on analytical models and involve additional assumptions and approximations. Great care must be exercised at each step in this process to ensure a valid simulation model and to ensure that the simulation results reflect reality.

1.4 Deterministic and Stochastic Simulations

There are basically two types of simulation: deterministic simulation and stochastic simulation. Deterministic simulation is probably familiar to most of us from previous experiences. An example might be a SPICE simulation of a fixed electrical circuit in which the response to certain deterministic input signals are of interest. A software program is developed that represents the components of the circuit and the input applied to the circuit. The simulation generates the currents present in each branch of the network and, consequently, generates the voltage across each circuit element. The voltages and currents are typically expressed as waveforms. The desired time duration of these waveforms is specified prior to executing the simulation program. Since the circuit is fixed and the input signal is deterministic, identical results will be obtained each time the simulation is executed. In addition, these same waveforms will be obtained if the network is solved using traditional (pencil and paper) techniques. Simulation is used in order to save time and to avoid the mathematical errors that result from performing long and tedious calculations.

Now assume that the input to the network is a random waveform. (In more precise terminology we would say that the input to the network is a sample function of a stochastic process.) Equivalently the system model might require that the resistance of a resistor is a random variable defined by a certain probability density function. The result of this simulation will no longer be a deterministic waveform, and samples of this waveform will yield a set of random variables. Simulations in which random quantities are present are referred to as *stochastic simulations*.

As an example assume that the voltage across a certain circuit element is denoted $e(t)$ and the simulation is performed to generate the value of $e(t)$ at 1 millisecond. In other words we desire $e(0.001)$. In a deterministic simulation $e(0.001)$ is fixed and we get the same result each time we perform the simulation. We also get this same number using traditional analysis techniques. In a stochastic simulation $e(0.001)$ is a random variable and each time we perform the simulation we get a different value of this random variable.

Another example might be a digital communication system in which the received signal consists of the transmitted signal plus random noise. Suppose that it is our task to compute the probability of symbol error at the receiver output. We know from a basic course in digital communications that if the modulation format is BPSK (binary phase-shift keying) and if the channel is AWGN (additive, white, Gaussian noise), the probability of symbol error is given by

$$P_E = Q\left(\sqrt{\frac{2E_b}{N_o}}\right) \tag{1.4}$$

where E_b is the symbol energy, N_0 is the single-sided noise power spectral density, and $Q(x)$ is the Gaussian Q-function defined by

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty \exp\left(-\frac{y^2}{2}\right)\, dy \tag{1.5}$$

Note that P_E is a number and not a random variable, even though there is a random quantity (noise) present at the receiver input. The number P_E is an average over an infinite number of trials, in which a trial consists of passing a digital symbol through the system and observing the result. The result, of course, will be that either a correct decision or an error is observed at the receiver output. For *ergodic* processes we can determine the probability of error in two different ways. We can view a single bit being transmitted and calculate P_E as an *ensemble average* in which we have an infinite ensemble of noise waveforms all having the same statistical properties. Alternately, we can determine P_E as a time average by transmitting infinitely many binary symbols and using a single *sample function* of the noise. The key is that we calculate P_E using an infinite number of transmitted binary symbols. If instead of determining P_E based on an infinite number of transmitted symbols, we *estimate* P_E using a finite number of transmitted binary symbols, we will find that the estimate of P_E is indeed a random variable, since each finite-duration sample function will yield a different (hopefully not much different) value for the error probability. This will be demonstrated in a following paragraph when we take a brief look at the Monte Carlo technique.

It is very important to note that both analysis and deterministic simulations result in a number. Each time the analysis is performed, the same number will result. Each time a deterministic simulation is performed, the same result will be obtained. Stochastic simulations, however, result in random variables, and the statistical behavior of these random variables is very important in determining the quality of the simulation result.

1.4.1 An Example of a Deterministic Simulation

Although the main purpose of this book is to present and explore the techniques used in stochastic simulations, one should not lose sight of the fact that completely deterministic simulations are important tools for gaining insights into the operational behavior of communication systems. One can execute a simulation that determines the waveforms present at points of interest in a system. System parameters can be changed and the effects of changing parameters can be readily observed. Very simple models can often be used and still important results can be obtained.

As a simple example consider a phase-locked loop, such as would be used for synchronization or demodulation. A block diagram is illustrated in Figure 1.7. The system appears quite simple. However, due to the nonlinear characteristics of the phase detector, analysis of phase-locked loops in the acquisition mode is quite complex. As a simple example, an important performance parameter of a PLL is the time required to acquire a signal, given various loop parameters and the specification of the input signal. To solve this problem analytically requires the solution of a nonlinear differential equation. We therefore turn our attention to simulation.

Suppose that a PLL is designed with a natural frequency of 5 Hz and a damping factor of 0.707. Also assume that the PLL is operating in lock and that the input frequency changes instantaneously by 20 Hz at $t = 0.1$ second. Given the large ratio of the step change in the input frequency to the natural frequency of the PLL, the PLL will lose phase lock and must reacquire the input signal. The nonlinear behavior of the loop leads to a phenomenon called "cycle slipping," and

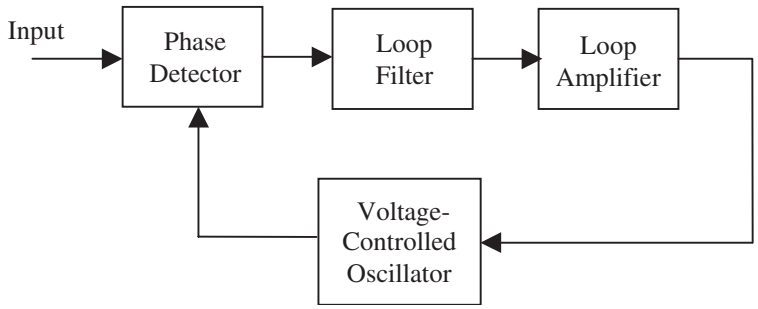


Figure 1.7 PLL model.

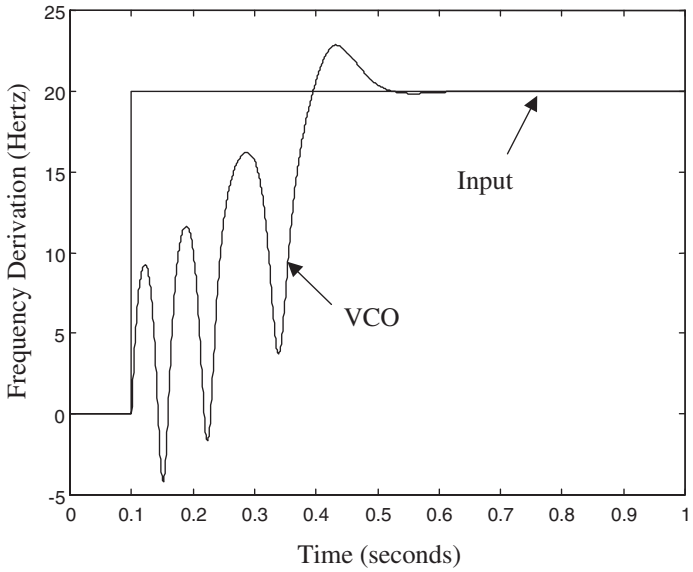


Figure 1.8 PLL acquisition behavior.

the acquisition time will be largely dependant upon the number of cycles slipped in the acquisition process.

The result of a simple simulation is illustrated in Figure 1.8, in which the step in the input frequency occurs at $t = 0.1$. We see that the PLL slips three cycles and then reacquires approximately 0.6 s after application of the frequency step. The simulation is completely deterministic, and performing multiple simulations using the same PLL parameters and signal model will result in identical results. This problem will be explored in greater depth in a later chapter in order to examine techniques for developing system simulations without the complications imposed by the presence of random perturbations.

1.4.2 An Example of a Stochastic Simulation

We now consider a completely different situation. Consider the simple digital communication system illustrated in Figure 1.1 and assume that we wish to determine the bit error rate (BER). The most basic simulation technique for determining this important performance measure is to pass a large number of digital symbols through the system and count errors at the receiver output. This is known as the Monte Carlo technique. If N symbols are processed by the system and N_e errors are observed at the system output, the Monte Carlo estimate of the error probability is

$$\widehat{P}_E = \frac{N_e}{N} \tag{1.6}$$

This is known as the BER based on N symbols, and the value of the BER is that it provides an estimate of the symbol error probability, which, using the relative frequency definition of probability, is

$$P_E = \lim_{N\to\infty} \frac{N_e}{N} \tag{1.7}$$

Since a simulation of necessity can process only a finite number of symbols, the symbol error probability can only be approximated.

Since the terms *bit error rate* and *probability of bit error* are often taken to mean the same thing, it might appear confusing to distinguish between the two. These two terms, however, are actually quite different. The BER is an estimate of the probability of bit error. One should keep in mind that "rate" is formed as a fraction, such as miles per hour. BER is indeed a rate, since it means N_e errors per N transmitted symbols. Replicating the random experiment of transmitting N symbols through a noisy, or random, channel K times will usually result in K different error counts, N_e. The probability of bit error, however, is based on passing an infinite number of symbols through the system. The probability of bit error, rather than being a random variable, is a number. For example, the probability of bit error for a binary PSK (phase-shift keying) system in an AWGN (additive, white, Gaussian noise) is $Q(\sqrt{2E_b/N_0})$ where E_b is the energy per bit and N_0 is the single-sided power spectral density of the channel noise. This number remains fixed as long as E_b and N_0 are held constant.

Suppose we perform $K = 7$ independent Monte Carlo simulations of a binary PSK communications system in which we have adjusted E_b/N_0 so that the probability of symbol (or bit) error is 0.1. Each simulation is based on $N = 1,000$ transmitted symbols. The result of replicating the random experiment of passing 1,000 symbols through the random channel seven times is shown in Figure 1.9. The randomness is evident in that the BER based on any number of transmissions $N \leq 1,000$ gives a spread of results. This spread is related to the variance of the estimate and in general, in order for simulation results to be useful, the spread should be small. Note that, for the results shown in Figure 1.9, the variance grows smaller as N grows larger. This is typical behavior for a correctly developed estimator. Also note that for large N, the results cluster about the true probability or error, and we tend to believe that, for a correctly developed simulation, the estimator, $\widehat{P}_E$, will converge to the probability of error, P_E, consistent with the relative frequency definition of probability. This is also typical of correctly developed estimators. These two desired conditions are well-defined concepts in estimation theory. If the variance of the estimate tends to zero as N grows arbitrarily large, we say that the estimate is consistent. Also, if $E\left\{\widehat{P}_E\right\} = P_E$, we say that the estimate is unbiased. We will have much more to say about the properties of estimators in later chapters, and we will also learn how to develop the simulation upon which Figure 1.9 is based.

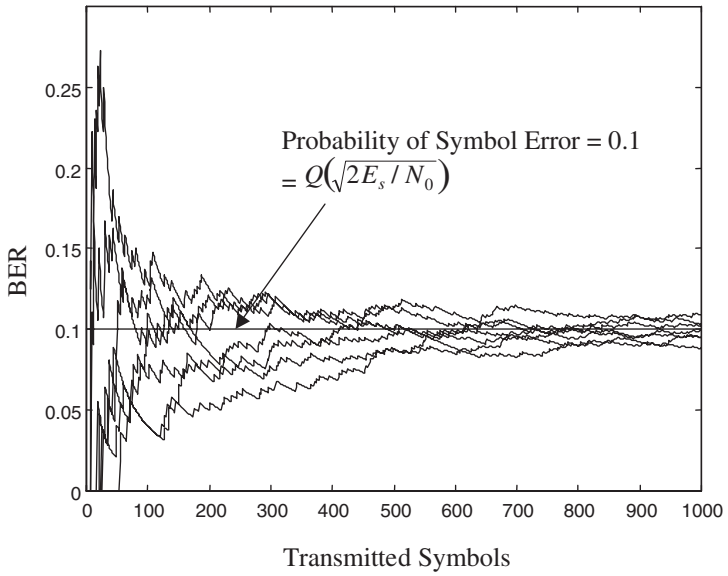


Figure 1.9 Monte Carlo simulation results.

1.5 The Role of Simulation

Simulation is used extensively during the many phases of the system design process and deployment process of modern communication systems. While simulation is used primarily for performance evaluation and design tradeoff studies (parameter optimization), simulation can also be used for establishing test procedures and benchmarks, end-of-life predictions, and anomaly investigations after the system has been deployed in the field. Both the simulation methodology and the simulation model used to represent the system will depend on the various phases of the design, implementation, and the lifecycle of the system. The simulation methodology will also be governed or guided by the overall design flow used. We illustrate the design flow and the use of simulation during various phases of the design and lifecycle of a communication system.

The design of a complex communication system is done from the "top down," whereas hardware implementation usually proceeds from the "bottom up." By this we mean that, in designing a system, we start at the system level (the highest level of abstraction) and start filling in the details of the design from system level and proceed down to subsystem level and finally to the component level. We then reach the bottom level at which the details of component assembly can be identified. When building a system, the components are first fabricated. These are then assembled into subsystems, and finally the entire system is constructed from the subsystems. Simulation development follows the top-down approach. We start with a system-level simulation, having a high level of abstraction, followed by more

and more detailed models and simulations of subsystems and components. As the implementation begins, the measured characteristics of components and subsystems are included in the simulation model.

We now describe the various phases of the design process and how simulations are used during various phases of the design process.

1.5.1 Link Budget and System-Level Specification Process

The design process for a communications system begins with the statement and analysis of user requirements and performance expectations including throughput, error rate, outage probability, and constraints on bandwidth, power, weight, complexity/cost, channel over which the system is expected to operate, and the life expectancy of the system. Based on the user requirements, the "systems engineer" arrives at an initial concept for the system such as the modulation schemes to be used, the coding and equalization techniques if necessary, and so on. A set of parameter values called A-level specifications such as power levels, bandwidths, and modulation index are also established during this initial stage of the design.

The overall goal at this point in the design process is to determine a system topology and the parameter values that will meet performance objectives and also meet the design constraints. As stated earlier, the system performance will be a function of the signal-to-noise ratio (SNR, or equivalently the value of E_b/N_0) and the total distortion introduced by all the components in the communication link. The signal-to-noise ratio is established though a process called link budgeting, which for the most part is a power calculation that takes into account such factors as the transmitted power, antenna gains, path losses, power gains, and noise figures of amplifiers and filters.[1] While the link budget is not the primary quantity of interest in simulations, it does establish a range of values of S/N or E_b/N_0 over which simulations for performance estimations have to be carried out.

Since it is impossible to build ideal components, practical implementation of components like amplifiers and filters will produce nonideal behavior. As a result, signal distortion will be induced, which will impact system performance. This is taken into account in the link budget by calculating the performance of the system with ideal components and then including "implementation losses" that account for performance degradation due to the signal distortion induced by nonideal components. The implementation loss is a measure (often an estimate based on prior experience) of how much the E_b/N_0 must be increased in order to overcome the effect of the distortion induced by nonideal components. Sometimes the implementation losses are also referred to as communication or distortion parameters. Note that some parameters, such as filter bandwidths, might affect the noise power at

[1] A link budget typically takes the form of a spreadsheet in which all of the system gains and losses (both signal and noise) including propagation loss, antenna gains, amplifier noise, cabling losses and other effects are identified and numerically defined. These are usually expressed in dB. After the spreadsheet is complete, the SNR necessary at the receiver input for the required level of performance is determinied. Using the spreadsheet, one can then work backward and determine the transmitter power required to achieve the required performance.

various points in the system and this in turn will impact the link budget as well as the distortion.

The system designer starts with an initial configuration for the system, the A-level specs and the link budget. The link budget is expressed in a spreadsheet-like format, and the bottom line in the link budget is the net E_b/N_0 at a critical point in the system after all the implementation losses have been taken into account. This "critical point" is often the receiver input. The link budget is said to be "closed" or "balanced" if the link has sufficient E_b/N_0, with a safe margin, to produce acceptable system performance. There are many different measures of system performance. As examples, analog systems often use mean-square error as a performance measure while a typical performance measure for digital systems is the bit error rate. At this point in the design process, the performance metric is computed from approximate formulas and not simulated. Since all of the implementation losses have been accounted in the net E_b/N_0, the BER, for example, can be computed using the formula for an ideal system.

If the link budget does not close or balance, then the A-level specifications, the implementation losses, and even the system configuration are changed and the link budget is recomputed. For example, the bandwidth of one or more filters may be changed, the antenna size (gain) may be increased, and the specification of the noise figure of an amplifier might be lowered. This process is continued until the budget is balanced or closed with an adequate margin.

Based on the initial system configuration, the A-level specifications and the link budget, which is now assumed to be closed, it should be possible to construct a simulation model that can be used to verify the link budget and refine the design. Performance measures can be estimated accurately and performance degradations due to nonideal implementations can be verified through detailed simulations. If the allocations in the link budget are verified through simulation and the link budget is still closed, the design process then proceeds to the next stage, which involves the detailed design and implementation of subsystems and components. If the link budget does not close, then some of the distortion allocations are changed and the system topology and A-level specs might have to be changed. For example, the coding gain might have to be increased and the specifications on the linearity requirements of an amplifier might be changed. Also if the simulation indicates that the distortion due to a component is less than what was allocated to that component in the link budget, the resulting savings can be applied to relax the requirements for some other component (i.e., more distortion can be tolerated elsewhere in the system). This iterative process continues until the link budget is balanced. A balanced link budget provides the initial specifications for hardware (and software) development.

This initial phase of the design involves a considerable amount of "art," and it is usually done by someone who has a considerable level of experience in designing communication systems. In most cases the initial design will be based on previous designs for similar systems with minor modifications. In other words, new designs are often evolutionary or incremental in nature.

1.5.2 Implementation and Testing of Key Components

The design for a new communication system will almost always contain some new signal-processing algorithms, and new hardware (and software) technologies. With any new technology there is always some risk or uncertainty about performance. If the new technology is introduced in a critical element of a communication system, that component must first to be built and tested under realistic operating conditions in order to verify the performance and minimize the risk. Since only a few key components will have been built at this early stage in the design process, it is impossible to test the entire system in hardware. Simulation provides an excellent test environment in this situation and the use of simulation is much less costly than hardware prototyping an entire system. All components and signals, up to the input to the component being tested and after its output, are simulated with measured characteristics of the component being tested inserted into the simulation model for the component. For example, if the component being tested is a new amplifier, its AM to AM and AM to PM transfer characteristics are measured and the measured characteristics are inserted into a nonlinear model for the amplifier. The entire system is then simulated to verify the resulting performance and the link budget. Once again, if the measured characteristics inserted into the simulation indicate better-than-expected distortion, then the savings are applied elsewhere in the system.

If the link budget closes, then the hardware development proceeds to the next critical component. Otherwise, either the component is redesigned, rebuilt, and tested again, or the link budget is modified to take into account additional degradation introduced by the component (beyond what was allocated in the original link budget for the component). This procedure is repeated for other key components.

1.5.3 Completion of the Hardware Prototype and Validation of the Simulation Model

As the procedure described above advances, a hardware prototype of the entire system begins to emerge along with an accompanying simulation model. The simulation model now includes measured characteristics for most of the components in the model. Many of the performance metrics for the entire system can now be measured on the hardware prototype. Parallel simulations are also conducted. Measured performance characteristics can be compared with the simulation results, and vice versa. Simulations provide benchmarks for testing, and test results validate the simulations. The end result of this phase of the design process is a complete prototype of the system, which serves as the basis for developing the production version of the system. In addition, we have a validated simulation model that can be used for end-of-life (EOL) predictions with a high degree of confidence.

1.5.4 End-of-Life Predictions

While the preceding procedure leads to a design that guarantees a given level of performance when the system is deployed, there is another important requirement that must be satisfied for most systems. This is the end-of-life performance. Many

communication systems such as communication satellites and under-sea cable systems are expected to have a long lifespan (usually 10 years or more) over which performance must be guaranteed. It is of course impossible to subject a hardware prototype to an actual lifecycle test, since such a test might, if executed in real time, last many years! While procedures for so called accelerated life testing have been developed, it is a common practice to use simulations as a complementary approach to accelerated life testing.

EOL performance predictions using simulations are accomplished though the use of aging models for the major components in the system. If we have a validated simulation model for the entire system at the beginning of life (BOL) and also have good models for the behavior of components as a function of age, which are somewhat easier to obtain, then the aging models for the components can be substituted in the validated BOL model to arrive at EOL performance metrics for the system.

If the predicted EOL performance is satisfactory, and the final EOL link budget is closed with adequate margin, the system design and implementation is complete. Otherwise, the process has to be iterated until convergence is achieved.

A summary of the key steps in the design flow and the role of simulation in communication systems engineering is shown in Figure 1.10.

1.6 Software Packages for Simulation

Over the past decade a variety of software packages have been developed, and are being widely used, to simulate communication systems at the waveform level. The essential components of a simulation framework for communication systems include a model builder, a model library, a simulation kernel, and a postprocessor. Individual simulation packages differ in the way these components are implemented and in the scope and focus of the model libraries that are provided.

Irrespective of the specific simulation package used, the first step in simulating a communication system consists of building simulation models of the various subsystems that make up the overall system and configuring these subsystems into an end-to-end simulation of the system of interest. Simulation models can be built using a general-purpose programming language and writing the appropriate code or by using a graphical model builder. With graphical model builders, simulation models of subsystems and of the overall communication system are developed using building blocks taken from the model libraries provided with the simulation environment. Icons representing functional blocks such as information sources, encoders, modulators, multiplexers, channel models, noise and interference sources, filters, demodulators, decoders, and demultiplexers are selected from various model libraries. These subsystem icons are then placed on the screen of a PC or workstation, moved to appropriate locations, and "wired" together to create a simulation model in a hierarchical block diagram form. SIMULINK is a relatively simple simulation package using the graphical model builder approach.

Models are built either from the top down or from the bottom up with the top-down view being preferred by systems engineers and the bottom-up approach being the choice of hardware engineers. At the "leaf level," which is the lowest level in the

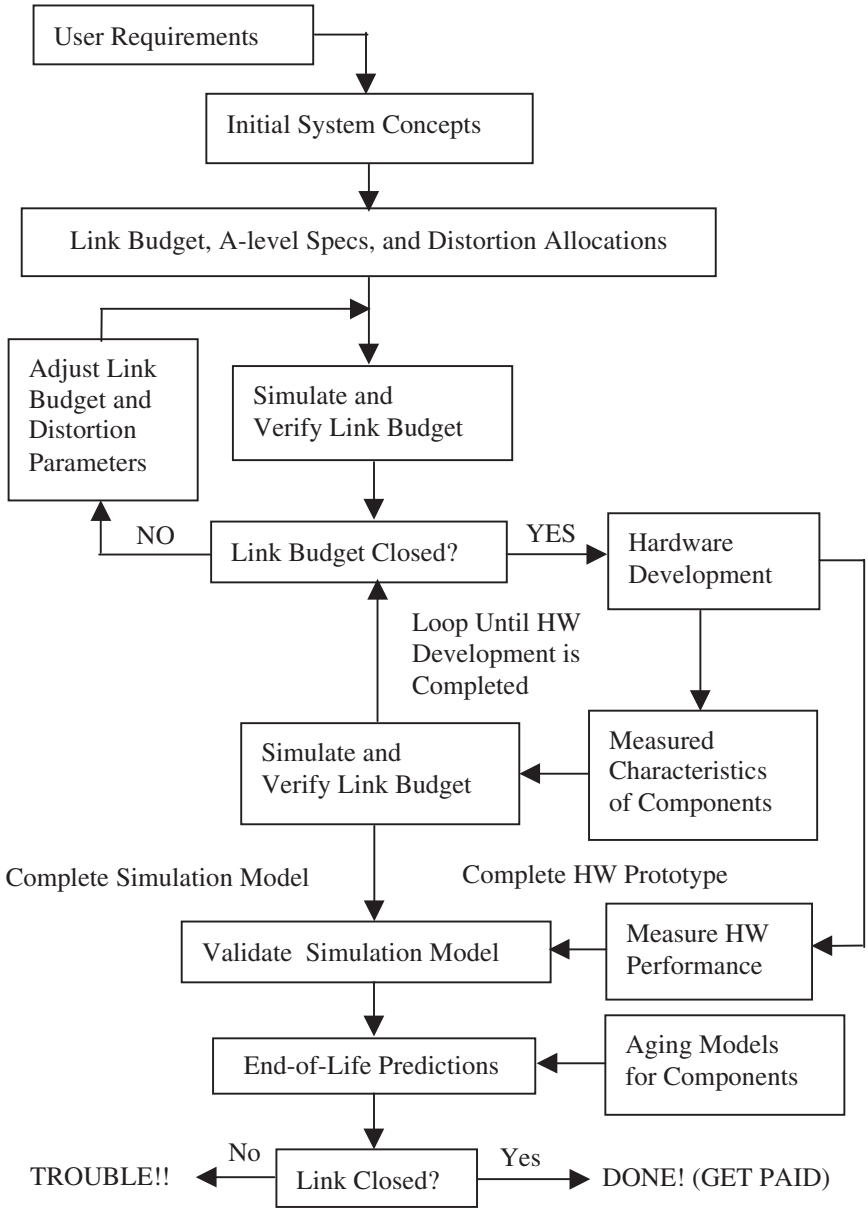


Figure 1.10 Systems engineering and design flow.

hierarchy, models can have a number of representations ranging from floating-point subroutines or procedures in a programming language, such as FORTRAN, C, or C++, to bit-level implementations of subsystem models in VHDL.

As an alternate approach to using a graphical block diagram editor for model building, one could use an intermediate (pseudo) language such as the MATLAB command language. Those producing simulations to guide the development of complex and expensive communication systems generally prefer the block diagram approach and graphical model builders. This preference results because the block diagram approach is a natural representation of communication systems and provides the systems engineer with a user-friendly environment. Despite the advantages of the block diagram approach, we will use MATLAB extensively in the work to follow for the reasons discussed in Section 1.8.

The level of effort expended in building a simulation model of a system is greatly reduced by the availability of model libraries that contain an extensive set of well-documented and well-tested building blocks. Many of the commercial simulation packages for communications systems available today have extensive model libraries available.

After the simulation model is developed, *simulation parameters* (such as sampling rates, seed numbers for random number generators and simulation length) and *design parameters* (such as filter bandwidths, code rates, and signal-to-noise ratios) are specified. The simulation is then executed. Linking all the models together, generating executable code, starting the simulation, saving sampled values of waveforms generated by the simulation, and monitoring the completion of simulation are functions that are usually performed by the simulation kernel/manager.

After the completion of the simulation, performance measures such as bit error rates and signal-to-noise ratios are computed from the waveforms generated by the simulation and the results are displayed as a function of design parameters using a simulation "postprocessor." Spectral plots, waveform plots, scatter diagrams, and eye diagrams are some of the commonly used visual aids for both viewing the simulation results and for debugging the simulation. As an additional aid in viewing results and debugging simulations, some simulation environments also provide the ability to view simulation results inactively as they are generated during a simulation rather than viewing results only after a simulation has been completed. This is especially helpful in simulations requiring lengthy run times.

Just as a well-stocked model library reduces the effort involved in creating a simulation model, a well-developed postprocessor with good interactive graphical capabilities can significantly reduce the effort involved in analyzing and displaying the simulation results. A rich set of analysis and estimation algorithms for error rates, power spectral densities, probability density functions, statistical parameters, and a flexible set of display routines are essential components of a good postprocessor.

Different simulation kernels or frameworks provide different sampling and simulation techniques. These techniques can be classified as time driven (single-rate, multirate, or variable-rate sampling), stream driven, event driven, or mixed. In the simplest case of a time-driven simulation there is a single simulation clock, and each functional block in the simulation model is executed once every "tick" of the

simulation clock. The simulation clock is then advanced by a fixed (constant) increment equal, to the reciprocal of the sampling frequency. All functional blocks in the model are then invoked again so that each model can update the model state to correspond to the new value of the simulation clock. Simulations of this type are structured as a single "do loop" or "for loop" in which each tick of the simulation clock increments the loop index by one.

Event-driven simulations, on the other hand, advance the clock by an arbitrary amount to the scheduled time of the next event of interest, and each functional block in the system updates its state corresponding to the value of the new simulation time. Typically, only a few blocks need to be activated to update their internal states, and no processing takes place during the "inter-event" time. Simulations of queueing systems are typically developed in this manner.

Event-driven and variable-step size simulations are computationally more efficient than time-driven simulations. However, they might require interpolation and resampling in some cases and they do carry an overhead associated with scheduling. For simulations of communication systems, time-driven simulation with either single-rate or multirate sampling is most commonly used. Multirate sampling is called for in simulations of systems having signals with widely varying bandwidths. A spread spectrum system is therefore an example of a system in which the use of multirate sampling can greatly reduce the required simulation run time.

Digital signal-processing algorithms play an important role in both simulation and in the implementation of communication systems. Simulation algorithms used for functions such as filtering and equalization can actually be used for implementing these functions in hardware or in software using DSP processors. Hence it is often of interest to include implementation issues such as bit widths and resource sharing in the simulation model and to move seamlessly from simulation to implementation. For hardware implementation, this is accomplished using hardware description languages such as VHDL to provide the interface between the system-level simulation framework and the hardware design tools. For software implementation of system components the simulation framework can translate the simulation algorithm to the assembly language code required for a target DSP processor. These links to implementation are becoming increasingly important as more and more functions in communication receivers are implemented in digital hardware or as embedded software.

1.7 A Word of Warning

We should never think of simulation as a replacement for traditional analysis or hardware measurements. Simulation is most powerful when used hand in hand with analysis and measurement. Quite often, the insights gained through repeated simulations allow the critical parameters in a system to be identified and for the system model to be simplified. The resulting simplifications often allow additional analysis to be performed.

"TranterBook" — 2003/11/18 — 11:36 — page 27 — #45

Section 1.9. Outline of the Book 27

Some level of analysis is always required for solving system-level problems. As an example, one must understand the basic dependence of performance parameters, such as the bit error rate, mean-square error at the demodulator output, or the signal-to-noise ratio at the receiver input, on system parameters, such as transmitted power and bandwidth, modulation format, or code rate, in order to ensure that the system is performing properly and that the simulation results are reasonable. In other words, as parameters are varied within a simulation, one must ensure that the observed results of these changes are reasonable and consistent with known theory. These "sanity checks" are important for validating the simulation and almost always require some level of analytical effort.

1.8 The Use of MATLAB

MATLAB will be used throughout this book for demonstrating concepts, for problem solving, and for performing example simulations. As mentioned in the preface, there are a number of reasons for the choice of MATLAB. First, MATLAB is widely used in the engineering community. MATLAB combines excellent computational capabilities with excellent and easy-to-use graphical capabilities. MATLAB contains a rich library of preprogrammed functions (m-files) for generating, analyzing, processing and displaying signals. Add-on libraries (toolboxes) allow the basic MATLAB library to be supplemented with m-files important to specific application areas. It is easy for the MATLAB user to generate new m-files for user-dependent applications. In addition, MATLAB code is very concise, making it possible to express complex signal-processing (simulation) algorithms using a very few lines of code.

Most of the examples, demonstrations, and problems used in this book can be solved using the Student Edition of MATLAB. Occasionally, a restriction present in the MATLAB Student Edition may make it necessary to use the professional version of MATLAB.

1.9 Outline of the Book

This book is divided into three parts. The first part, "Introduction," consists of two chapters that explore simulation and modeling philosophy in a very broad context. The second part, "Fundamental Concepts and Techniques," covers the basic techniques used in the simulation of almost all communication systems. These include the fundamental concepts of sampling and discrete system theory, filters and filter models, the representation of signals and systems in simulations, noise generation and modeling, the development of graphical displays, and Monte Carlo simulation techniques. A number of simple case studies are included in Part II. The first case study, devoted to the acquisition behavior of phase-locked loops, allows us to illustrate simple simulation techniques and to identify the sources of error in simulations without having to consider the complicating effects of noise. The second case study considers the simulation of a wireless communications system. This case study comes after our study of noise, and therefore the effects of noise on the

communications system are considered. After a careful study of Part II, one should be able to simulate a moderately complex digital communications system in operating in a Gaussian noise environment. While simulation, strictly speaking, may not always be necessary for determining the performance of these systems, important insights can often be gained by observing the waveforms present at various points in the system. Simulation of simpler systems, in which the simulation results can easily be understood and verified, often provides a starting point for developing simulations of more complex systems.

The third part of this book, "Advanced Modeling and Simulation Techniques," treats many of the concepts required for the development of simulations of modern systems. In Part III, simulation strategies for nonlinear and time-varying systems are explored. Attention is then turned to the important problem of modeling time-varying channels, such as those encountered in mobile wireless communication systems. Both waveform-based models and discrete channel models based on Markov processes are considered. Finally, variance reduction techniques are briefly considered. The general term *variance reduction techniques* encompasses a number of strategies that allow knowledge of system details to be used in a way that reduces the time required to execute a simulation with a given level of accuracy.

1.10 Further Reading

Very few books have been written that focus specifically on the simulation of communication systems. Two books falling into this category are

M. C. Jeruchim, P. Balaban, and K. S. Shanmugan, *Simulation of Communication Systems*, 2nd ed., New York: Kluwer Academic/Plenum Publishers, 2000.

F. M. Gardner and J. D. Baker, *Simulation Techniques*, New York: Wiley, 1997.

However, a number of books cover general topics relevant to our study. Several that are cited from time-to-time in this book include

R. Y. Rubinstein, *Simulation and the Monte Carlo Method*, New York: Wiley, 1981.

B. D. Ripley, *Stochastic Simulation*, New York: Wiley, 1987.

S. M. Ross, *A Course in Simulation*, New York: Macmillan, 1990.

P. Bratley, B. L. Fox, and L. E. Schrage, *A Guide to Simulation*, 2nd ed., New York: Springer-Verlag, 1987.

As mentioned earlier in this chapter, MATLAB is used throughout this book to illustrate methodology and algorithms. A number of complete simulations are also included. It is therefore important to have at least a basic familiarity with MATLAB. While the MATLAB manuals, together with the online help, provide descriptions of the techniques and routines used in this book, the following two references have been useful to the authors:

D. Hanselman and B. Littlefield, *Mastering MATLAB 5: A Comprehensive Tutorial and Reference*, Upper Saddle River, NJ: Prentice-Hall, 1998.

A. Biran and M. Breiner, *MATLAB for Engineers*, Reading, MA: Addison-Wesley, 1995.

G. J. Borse, *Numerical Methods With MATLAB*, Boston, MA: PWS Publishing Company, 1997.

The first of these books, as the title implies, is a good tutorial on MATLAB and is a useful reference for the beginning MATLAB user. The second two books are more oriented toward applications and algorithms. The last cited book (Borse) is more advanced and contains a number of DSP applications and techniques that are useful in the development of simulations.

Chapter 2

SIMULATION METHODOLOGY

As discussed in the preceding chapter, simulation plays an important role in the design of communication systems. Simulation is used for the detailed design of various components in a communication system as well as for system-level performance evaluation. This chapter is, in many ways, a continuation of the material presented in Chapter 1. In this chapter we will consider the modeling and simulation process in more detail and will see that there are both qualitative and quantitative aspects of simulation. Stated another way, simulation is both an art and a science.

Some steps used for creating and executing a simulation model are theoretically based and therefore quantitative in nature. Modeling of individual system components and the generation of random numbers fall in this category. On the other hand, many steps in simulation involve approaches and considerations that are not clearly quantifiable and are heuristic in nature. These are lumped into what is loosely called the "methodology" of simulation. The emphasis of this chapter is on the methodology or the "art" of simulation, especially as it applies to system-level performance evaluation. The "science" of simulation, the part that deals with the quantitative aspects of modeling, estimation, etc., are dealt with in later chapters. The quantitative and qualitative parts of simulation are not totally dichotomous but are closely interrelated. All steps used in simulation, including the modeling of

specific components, involve some "methodology." Furthermore, the executation of a simulation requires a set of algorithms.

For discussion and presentation purposes we will treat these two topics, the quantitative and qualitative aspects of simulation, as if they are somewhat disjoint. However, some familiarity with each topic will aid in the understanding of the other. This chapter should therefore be read before proceeding to the remainder of the book. The material in this chapter will have greatest value, however, if it is periodically revisited as the remaining chapters are studied.

2.1 Introduction

All but the most simple simulation problems involve the following fundamental steps:

- Mapping a given problem into a simulation model
- Decomposing the overall problem into a set of smaller problems
- Selecting an appropriate set of modeling, simulation, and estimation techniques and applying them to solve these subproblems
- Combining the results of the subproblems to provide a solution to the overall problem

Usually, specific techniques for solving the smaller subproblems (the third bullet) will be well defined and rigorous, and are algorithmic or quantitative in nature. For example, the technique for simulating a linear filter represented by a transfer function using the finite impulse response (FIR) method is the well-defined convolutional sum. On the other hand, the overall "methodology" used for mapping a design or performance estimation problem to a suitable simulation model, and selecting a set of consistent and compatible techniques for applying that model, will involve heuristic procedures and "tricks of the trade."

The basic purpose of a communication system is to process waveforms and symbols, and hence the simulation of communication systems is an attempt to emulate this process by generating and processing sampled values of those waveforms. This involves modeling the signal-processing operations performed by the various functional "blocks" in a communication system and generating the required input waveforms that enter the communication system at various points. The process of "running" or "executing" a simulation consists of driving the models with appropriate input waveforms to produce output waveforms (which might serve as inputs to other functional blocks), and analyzing these waveforms to optimize design parameters or to obtain performance measures such as error rates in a digital communication system.

To illustrate the various aspects of methodology, we will use the example of a digital communication system operating over a time-varying or "fading" mobile communication channel. This channel introduces linear distortion that can be minimized by an equalizer in the receiver. The approaches to the detailed design of

the equalizer will be used to illustrate some aspects of methodology. The channel will also be time-varying because of mobility, causing the received signal to change randomly as a function of time. This random change in the received signal level is referred to as fading. When the received signal power falls below some threshold, the performance of the system as measured by the probability of error will become unacceptable and the system will be declared to be out of service. The outage probability of a communication system is defined as the percentage of time the communication system is "not available" due to bad channel conditions, which cause the error rate to exceed some specified threshold value. Estimation of the outage probability will require the simulation of the system under a large number of channel conditions and hence is a computationally intensive task. Methods for minimizing the overall computational burden associated with executing the simulation will be discussed.

The overall approach to waveform-level simulation of communication system is straightforward. We start with a description of the portion of the system that is to be simulated in a block diagram form in which each functional block in the block diagram performs a specific signal-processing operation. The appropriate simulation model for each functional block is chosen from a library of available models, and the block diagram model is created by interconnecting the set of chosen blocks (i.e., their models). Specific values, or a permissible range of values, for the parameters of each block, such as the bandwidths of filters, are specified prior to the execution of simulation. The block diagram is simplified if possible, and partitioned if necessary. The mapping of the design, and/or the performance estimation problem, into a simulation model is one of the most difficult steps in methodology. The computer time required for executing the simulation, and the accuracy of the simulation results, will depend upon how well this is done.

The next step involves the generation of sampled values of all the input waveforms or the stimuli to drive the simulation model. Signals, noise, and interference are represented by random processes, and sampled values of random processes are generated using random number generators. During simulation, the outputs of the random number generators are applied as inputs of the appropriate blocks to "drive" the simulation model and produce sampled values at the outputs of various functional blocks. Some of the output samples are recorded and are analyzed either while the simulation is executing ("in-line estimation") or at the end of simulation ("off-line estimation" or postprocessing), and various performance measures such as signal-to-noise ratios (SNRs), mean-square error, and probability of error are estimated.

The final step, and a very important step, in the simulation is the validation of the simulation results using analytical approximations and bounds, or measured results when available. Measured results are typically available only toward the end of a design cycle after prototypes have been built. Even when a prototype system is available, only a limited number of measurements are typically made. Measurements are inherently expensive and the reason we rely on simulation is to avoid the time and expense of taking a large number of measurements. (If everything can be measured at the time the measurement is required, then there is no need for

simulation!) Nevertheless, some validation against measured results has to be done to verify the models and the methodology used, and to establish the credibility of the simulation results.

A real communication system will usually be far too complex to model and simulate exactly in its entirety even if unlimited computational resources are available. A variety of techniques are used to reduce the overall complexity of the simulation problem to something that is within the scope of the available computer resources, the time available, and the accuracy desired. These techniques or tricks of the trade are loosely referred to as methodology and are described in the following sections with examples.

2.2 Aspects of Methodology

The overall approach, or methodology, used to solve a design or performance estimation problem depends on the nature of the specific problem. While it is difficult to present methodology as an independent set of rules or algorithms, there are certain generic aspects of methodology that can be applied to a wide variety of simulation problems. We describe these first and then present a set of specific methods for solving a set of individual problems.

2.2.1 Mapping a Problem into a Simulation Model

The starting point of a simulation is a clear statement of the problem and the objectives of the simulation. To illustrate various aspects of methodology, we will use the mobile communication system example, and consider the following two problems:

- Equalizer Design: Determine the number of taps, the tap spacing, and the number of bits used to perform the arithmetic operations in the equalizer to be used in the receiver.

- System Performance Evaluation: Determine the E_b/N_0 required to maintain an acceptable level of performance. (A more detailed description of the system and its performance specifications will be provided in later sections of this chapter.)

The first problem deals with the detailed design of a component in the receiver, whereas the second problem is one of system-level performance estimation. These two problems require different approaches in terms of what portion of the system to model, the level of detail to be included in the model, as well as the modeling techniques, the simulation techniques, and the estimation procedures to be used. Also, we must assume that the first problem has been solved before we can approach the second problem.

Irrespective of whether we are dealing with a detailed design problem, or a high-level performance estimation problem, the starting point is usually a detailed block diagram that represents the portion of the system that needs to be simulated. This block diagram representation, in its initial form, will often include more detail

than what might ultimately be necessary and also might have a lot of detail dealing with aspects of the system that might have no bearing on the design or performance issues being addressed. Nevertheless, it is customary and useful, as a starting point, to include "everything that one can think of" in the initial overall block diagram.

The final simulation model is created from the initial block diagram by simplifying it. There are three classes of generic techniques that are applied in this step of creating a simulation model:

- Hierarchical representation
- Partitioning and conditioning
- Simplifications (approximations, assumptions, and various simplifications)

Hierarchical Representation

Hierarchy is a commonly used approach for reducing the complexity in modeling, software design and other applications. In the context of a communication system, hierarchy is used to manage and reduce the complexity of the simulation model and also reduce the computational load associated with simulating the model. The hierarchical representation is done in various "layers" starting with a "system"-level model and progressing down through various other layers, which are usually referred to as the subsystem layer, the component layer, and the physical (gate-level or circuit) layer. Example "layers" for a particular communications system are shown in Figures 2.1, 2.2, and 2.3. The number of layers and the terminology used to define a given layer are not unique. There could be an arbitrary number of layers in the hierarchy, and what might be viewed as a subsystem in one context may be considered as a system in a different context. Nevertheless, we will use the term *system* to refer to the entire entity of interest. In terms of a hardware analogy, a system usually may be viewed as what is contained in a rack, cabinet, or box. A system contains subsystems that are often implemented at the board level. Boards are assembled from components (discrete components and ICs) and what is inside ICs are transistors and other physical devices.

In a hierarchical representation, or model of a system, the building blocks used in lower layers in the hierarchy will contain more detail, whereas the blocks at the higher layers are more abstract and deal with the overall function of the block. The decomposition into lower layers is done until no further meaningful decomposition is possible or necessary. The lowest level is often based on components such as resistors, capacitors, and ICs.

In the context of a communication system, the system-level model, shown in Figure 2.1, will consist of functional blocks such as information sources, encoders, decoders, modulators, demodulators, filters, and the channel. Each of these functional blocks can be considered a subsystem and decomposed, or expanded further, in order to show more detail. For example, the timing recovery subsystem can be decomposed into a fourth-order nonlinearity, two bandpass filters, and a phase-locked loop (PLL) as shown in Figure 2.2. Additional decomposition yields a "component"-level model. As examples, the bandpass filters shown in Figure 2.2 may be discrete

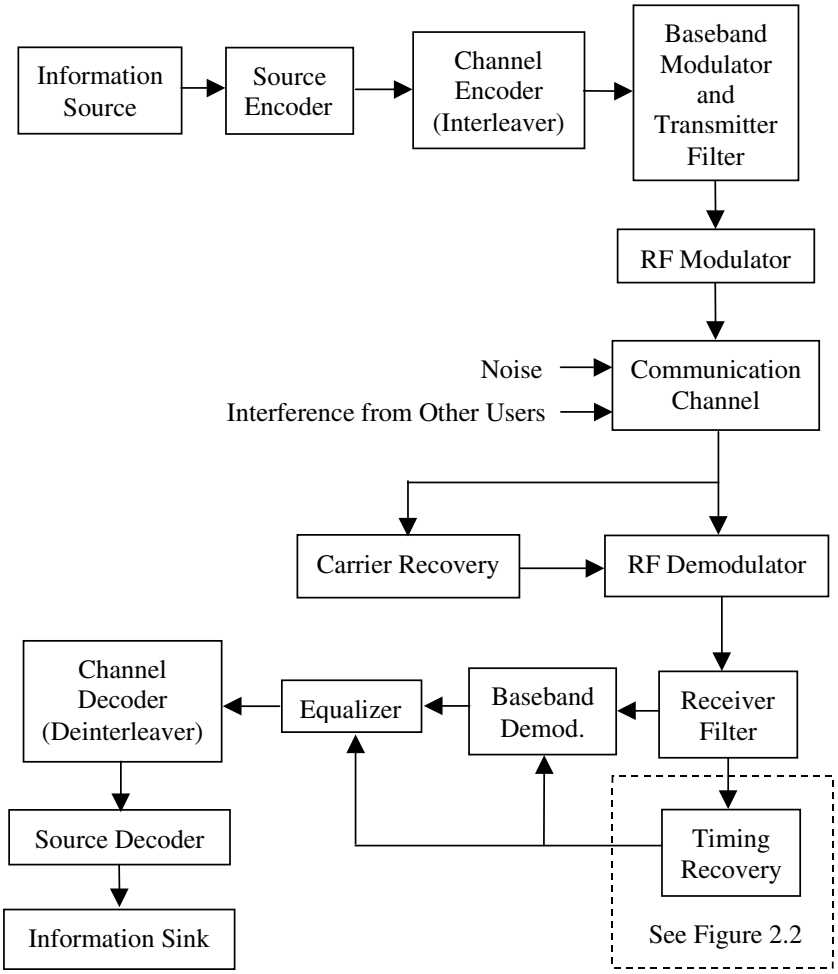


Figure 2.1 System-level model for communication system.

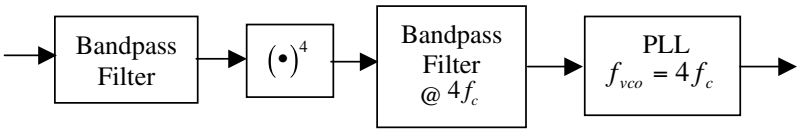


Figure 2.2 System-level model for timing recovery subsystem.

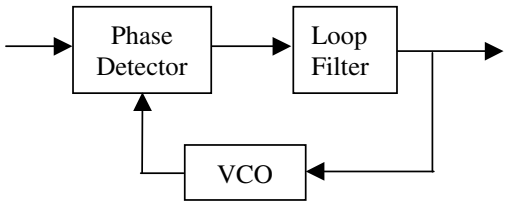


Figure 2.3 Component-level model for PLL.

component analog filters, microwave filters, or digital filters. In the case of analog filters, it might be possible to expand the filters into "circuit"-level models. For a digital filter, this decomposition will be down to bit-level adders, multipliers, and accumulators. A layer below this will involve individual transistors and gates. However, we very seldom go down to this level of detail in the context of waveform-level simulation of communication systems. The "component"-layer model for the PLL, which was briefly discussed in Chapter 1, is shown in Figure 2.3. We will consider the simulation of the PLL in detail later in this book. As with the filter, additional decomposition to a circuit-level model may be necessary for certain applications.

The main reason for using the concept of hierarchy is to manage the complexity of the simulation model and also to reduce the computational burden associated with simulating the model. In general, one should perform the simulation at the highest possible level of abstraction, consistent with the goals of the simulation, since higher levels of abstraction imply fewer paramerers and more efficient simulations. In the equalizer design example, the equalizer itself might be simulated at the bit level, whereas the channel might be simulated at a much higher level of abstraction. For example, a transfer function might be used to represent a channel. Similarly, a digital baseband filter in the receiver need not be simulated at the bit level if the goal of the simulation is system performance evaluation. The manner in which the filter is implemented will, of course, not affect the performance of the overall system so long as the transfer function of the filter is preserved.

In addition to reducing complexity and the required simulation time, higher-level models will have fewer parameters and also might be easier to validate. Fewer parameters imply that the model can be characterized by fewer measurements. For example, a circuit-level model of a Butterworth filter might involve a dozen or more component values. However, a high-level transfer function of the same Butterworth filter is characterized by just two parameters, the filter order and the filter bandwidth. Both of these parameters are easy to measure. In addition, validation of simulation results is simpler, with fewer measurements required, when the simulation model is at a higher level of abstraction.

At the system level, simulation is done at a highest level of abstraction using "behavioral" models such as transfer functions rather than physical models. The functional forms of the behavioral models are usually assumed or obtained from separate, but not concurrent, simulations of the lower-level blocks, or from

measurements. A digital filter, for example, might be simulated at the bit level and an analog filter might be simulated at the circuit level. A higher-level model for both filters can be derived from the bit-level or circuit-level simulations in the form of a transfer function. The only model for the filter that will be used at the higher level will be the transfer function model, which is computationally more efficient than a bit-level or circuit-level model. The details of the lower-level model, whether it is an analog filter or digital filter, is completely hidden from the higher layer. This approach of creating a higher-level model from the details of a lower-level model, and substituting back at the higher level is called "back annotation."

During the early phase of the design cycle, the filter transfer function is assumed or "specified" (e.g., a fifth-order Elliptic filter) and the actual characterization of the transfer function will be obtained later when the filter has been designed and simulated. Later in the design cycle, when the filter is actually built, its transfer function can be measured and the measured transfer function can be used in higher-level simulations. Simulation is very flexible in this context and a hierarchical approach adds to the flexibility of including multiple versions of models for a subsystem or a component with different levels of abstraction, but with the same external interfaces and parameters. In addition, hierarchy also reduces overall model complexity and the resulting computational burden.

Like the modeling process, the actual design of a communication system also flows top down through various layers. In the design process, specifications flow down through the layers of the hierarchy, and characterization (measured or simulated at the lower levels) flows back up through the layers of hierarchy. In some applications, it might be necessary to use different levels of detail in a single simulation model. For example, in the equalizer design, it may be necessary to estimate the system probability of error as a function of number of bits used for arithmetic in the equalizer. In this case all parts of the system surrounding the equalizer will be simulated at a very high level of abstraction, whereas the equalizer itself might be simulated in great detail using, sometimes, a different simulator. This approach is often referred to as "co-simulation."

Partitioning and Conditioning

Partitioning of a complex problem into a set of interrelated but independent problems, which can be solved separately and whose solutions can be combined later, is another technique that is useful for reducing the complexity and also the computational burden. Whereas hierarchy deals with different levels of abstraction, partitioning usually deals with the same level of abstraction but with various aspects of the problem that can be simulated separately and the results combined. Thus, for partitioning, we view and inspect the block diagram "horizontally" while hierarchy may be viewed as a "vertical" separation. In the context of the example shown in Figure 2.1 it might be possible to separate synchronization and coding from the rest of the problem and simulate them separately.

Conditioning is another technique that is very similar to partitioning—we simply fix the condition or state of a portion of the system and simulate the rest of the system under various values of the conditioned variables or states. The conditioned

part of the system is simulated separately and the results obtained in the first part are averaged with respect to the distribution of the conditioning variable obtained in the second part. This process is best illustrated with an example.

Suppose we wish to estimate the probability of error in the system shown in Figure 2.1 with nonideal synchronization (timing and carrier recovery). We can use partitioning and conditioning to simplify the problem by estimating the conditional probability of error in the system for various values of timing and carrier phase errors, and then simulating the synchronization system to obtain the distribution of the timing errors. We then average the conditional probability of error with respect to the distribution of timing errors and phase errors. What we are doing here is a well-known operation in statistics involving conditional expected values. In general:

$$\begin{aligned} E_{XY}\{g(X,Y)\} &= \int\int g(x,y) f_{XY}(x,y)\,dx\,dy \\ &= \int \left\{ \int g(x,y) f_{X|Y}(x\,|\,y)\,dx \right\} f_Y(y)\,dy \end{aligned} \tag{2.1}$$

which, in terms of conditional expectation, is

$$E_{XY}\{g(X,Y)\} = E_Y\left\{E_{X|Y}\{g(X,Y)\}\right\} \tag{2.2}$$

Returning to our example of determining the bit error rate (BER) in the presence of timing errors and phase errors, this principle can be applied to give

$$\widehat{P}_E = \int\int \widehat{\Pr}\{\text{Error}\,|\,\tau,\theta\}\,\widehat{f}_{\mathrm{T}\Theta}(\tau,\theta)\,d\tau\,d\theta \tag{2.3}$$

where $\widehat{\Pr}\{\text{Error}\,|\,\tau,\theta\}$ is the simulation-based estimate of the conditioned probability of error in the system given that the phase error is θ and the timing error is τ. The result of averaging, $\widehat{P}_E$, is the unconditioned (overall) probability of error and $f_{\mathrm{T}\Theta}(\tau,\theta)$ is the estimated (simulated) distribution of the phase error and timing error produced by the synchronization system. Note that the synchronization system is simulated by itself, apart from the rest of the system, and the results are averaged. This leads to the simulation of two simpler systems and should result in less simulation time.

If we can assume that the timing and phase recovery systems produce independent timing and phase errors, then these parts can be partitioned and simulated separately to obtain estimates of the distributions of the timing error, $\widehat{f}_{\mathrm{T}}(\tau)$, and the phase error, $\widehat{f}_{\Theta}(\theta)$. The joint distribution of the timing and phase error can be obtained as

$$\widehat{f}_{\mathrm{T}\Theta}(\tau,\theta) = \widehat{f}_{\mathrm{T}}(\tau)\widehat{f}_{\Theta}(\theta) \tag{2.4}$$

which can then be substituted in (2.3) to do the averaging.

It should be noted that partitioning deals with separating the problem into parts and conditioning guides partitioning and, more importantly, helps integrate

the results. The independence assumption, where appropriate, also aids in bringing simulation results together. The latter will be case in which the simulated parts produce statistically independent phenomena and processes that need to be combined.

Simplifications and Approximations

It was stated earlier that, as a starting point, it is common practice to include as much detail in the initial block diagram model as possible. The complexities of the overall model and the subsystem models are then reduced using a number of techniques including the omission of those blocks that do not have a significant impact on the problem being addressed, the use of approximations, and simplification by combining blocks.

As an example of how portions of the block diagram may be omitted, consider the system-level performance evaluation problem. If we can assume that the channel is very slowly time varying, and that the system is operating at a high signal-to-noise ratio, S/N, it is reasonable to expect that synchronization errors will be very small and hence the effects of synchronization can be ignored during performance estimation. In this situation, the timing recovery and the carrier recovery portions need not be simulated and can be deleted from the block diagram.

Approximations and assumptions are used extensively to simplify the simulation model. The most commonly used assumptions and approximations involve time invariance (stationarity), and linearization. While most practical systems, when observed over a long time and over a wide dynamic range of the input signal, might exhibit time-varying and nonlinear behavior, they can be well approximated by linear and time-invariant models over short time periods and for low signal levels.

Time invariance implies that over the simulation interval, the properties of the signals and system components being modeled do not change. In practice, the concept of time invariance is applied as an approximation in a relative and not in an absolute sense. If a system parameter is changing slowly, there are situations in which it may be assumed fixed over the simulation interval. As an example, consider the problem of BER estimation over a radio channel in which the transmit and receive antennas are stationary. If the changes in channel characteristics are due to changes in the atmospheric conditions, which have a time constant of several minutes to hours, and if the symbol rate of transmission is millions of symbols per second, then the channel can be assumed to be "quasi-static." That is to say that the channel remains in nearly the same condition while several hundred millions of symbols flow through it and it is meaningful to talk about instantaneous probability of error for a given channel condition. If the BER being estimated is of the order of 10^{-3}, then we need to simulate only a few thousand symbols to estimate this BER. This represents a simulation time interval of milliseconds, whereas the time constant of the channel is of the order of several minutes and hence it is reasonable to assume the channel to be static during the simulation interval. This quasi-static approximation plays a very important role in simplifying simulation models.

The quasi-static assumption, and the resulting simplifications, may be applied in any system in which we have phenomena and processes that have significantly

different bandwidths. In such systems, it might be possible to simulate the effects of the faster process while assuming the slow process to be in a fixed state. We can therefore view the quasi-static assumption as a requirement for partitioning and conditioning.

In a similar vein, we use linear approximations for nonlinear components. Nonlinear models in general are very complex to analyze, and while they are somewhat easier to simulate, they still pose problems. Whenever possible we try to approximate the behavior of these components by linear models.

Finally, we use many of the principles of linear systems to simplify the block diagram. We can combine several cascaded and parallel blocks into one block by multiplying or adding the transfer functions of the cascaded and parallel blocks, respectively. In the case of linear time-invariant blocks we can also interchange the order of the blocks when doing so leads to a simpler model. This type of simplification is desirable, especially when using simulation for performance estimation. First of all, performance estimation simulations are usually very long, and hence the effort in simplifying the model is justifiable. Second, unlike the situation in which simulation is used to support a detailed design, we are not interested in observing the evolution and progression of waveform through each functional block of the system. When simulation is used for performance estimation we are usually interested in simply comparing the input and output waveforms and counting errors. In this situation the intermediate waveforms are of very little interest or use. The entire system can be reduced to a very small number of blocks during performance estimation, and this would lead to considerable reduction in simulation time. If the blocks have similar complexity, combining the transfer function of n blocks will lead to computational savings of the order of (but less than) n.

To state the obvious, it must be clear that every attempt should be made to reduce the simulation model to the smallest number of functional blocks possible with the highest-level abstraction consistent with the goals of the simulation exercise. High-level models and lower-level models are relative terms. While high-level system descriptions with a smaller number of subsystem models will require less computation time, more detailed models will in general lead to more accurate simulation results. This improved accuracy, however, comes at the expense of increased computational time.

2.2.2 Modeling of Individual Blocks

The role of each functional block in a communication system is to perform a specific signal-processing function and hence its simulation model should mirror this function with varying degrees of abstraction. Irrespective of the internal details, the simulation model should accept a sequence of time-ordered samples of the input waveform and produce a time-ordered set of output samples according to some well-defined transfer characteristic. A number of choices and considerations must be taken into account in building the model, and we describe some of the methodological issues associated with modeling in the following sections. (Even though we now focus our attention on models at the system or subsystem level, some of the

methodological issues described in the preceding section apply here. Also, many of the concepts described here at the subsystem level also apply at the system level.)

The simulation model of a subsystem or a component (block) is a transformation of the form

$$\begin{aligned}&\{y[k], y[k-1], \cdots, y[k-m]\}\\&= F\{x[k-j], x[k-j-1], \cdots, x[k-j-n]; k; p_1, p_2, \cdots p_q\}\end{aligned} \tag{2.5}$$

where $x[k]$ represents input samples, $y[k]$ represents output samples, $p_1, p_2, \cdots, p_q$ represent parameters of the block, and $k = m, 2m, 3m, \cdots$ is a time index. The model uses n samples of the input to produce m samples of the output per "invocation" of the model according to the transformation F, which will be defined explicitly in terms of the input samples, the parameters of the block, and the time index k. If the transformation F does not depend upon the index k, the model is time invariant. If $m > 0$, the model is considered a block input-output model, and when $m = 0$ we have a sample-by-sample model. If $n = 0$ the model is memoryless.

In constructing the model for a functional block, and executing the model within a simulation, a number of considerations must be taken into account. These considerations are related to each other even though they are presented in an arbitrary order.

Lowpass Equivalent Representation

Communication systems contain components and signals that are either bandpass or lowpass in nature. From a simulation perspective, it is computationally advantageous to represent all signals and system elements by the complex lowpass equivalent representation. For signals and linear systems, the lowpass equivalent is obtained by shifting the bandpass spectra from the carrier frequency to $f = 0$ and the linear model of the block can be implemented using the lowpass equivalent representation of the input and output signals and the signal transformation. Details of this representation are presented in Chapter 4.

The lowpass equivalent of a deterministic signal is obtained via frequency translation of its Fourier transform, whereas the power spectral densities are used for random signals. If the bandpass spectrum is nonsymmetric around the carrier, the lowpass equivalent representation in the time domain will be complex valued. Furthermore, the components of the lowpass equivalent random process will be correlated in this case.

For certain types of nonlinear systems it is also possible to use the lowpass equivalent representation. Details of the lowpass equivalent models for nonlinear systems are described in Chapter 12.

Sampling

When signals and systems are lowpass or represented by their lowpass equivalent in the bandpass case, they can be sampled and represented by uniformly time spaced samples. The minimum sampling rate required is twice the bandwidth of the signal (or the system) for the ideal lowpass case. However, in practical cases in which

frequency functions may not be confined to a finite bandwidth, the sampling rate is often taken to be 8 to 16 times some measure of bandwidth, such as the 3-dB bandwidth. In the case of digital systems the sampling rate is usually set at 8 to 16 times the symbol rate. Factors such as aliasing error, frequency warping in filters implemented using bilinear transforms, and bandwidth expansion due to nonlinearities need to be considered in establishing a suitable sampling rate. These effects can be minimized by increasing the sampling rate, but higher sampling rates will increase the computational load and therefore one has to trade off accuracy versus simulation time. Multirate sampling, variable step size, and/or partitioning fast and slower parts of the system, when possible, are techniques that can be applied to reduce the computational burden.

Linear versus Nonlinear Models

While most of the blocks in communication systems will be linear, a significant portion of a communication system may involve nonlinear processing. Some of the nonlinear processing is intentional whereas some is unintended. Examples of the former are operations such as decision feedback equalizers, nonlinear operations in synchronization subsystems, and intentional limiting of impulse noise. Examples of the latter are the nonlinear behavior of power amplifiers near their maximum operating power.

As a first approximation, most of the nonlinearities can be modeled as having linear effects on communication signals, particularly if the signal is a constant envelope signal such as phase-shift keying (PSK). However, in multicarrier systems, or in single-carrier systems with higher-order quadrature amplitude modulation (QAM), unintended nonlinear behavior might impact system performance significantly and hence it is necessary to include nonlinear simulation models. Fortunately, most of these nonlinearities in communication systems can be modeled efficiently using complex lowpass equivalent representation.

There are various approaches to modeling systems with nonlinearities. These include, in order of increasing complexity, memoryless power series nonlinear models, frequency-selective nonlinear models with memory, and nonlinear differential equations. The mathematical analysis of nonlinear systems, and the evaluation of the effects of nonlinearities, is in general difficult. However, simulation is rather straightforward even in the case of frequency-selective nonlinear models.

Nonlinear models fall into two broad categories: input-output block models and nonlinear differential equations. The first set of models are usually based on measurements whereas the second class of models are often derived from modeling the physical behavior of the device. Solutions of nonlinear differential equation models implemented using variable time-step integration models are computationally most efficient even though they might involve more setup time. It is also possible to decompose a nonlinear subsystem in a block diagram form and simulate in block diagram form using simpler building blocks such as memoryless nonlinearities and filters. This approach, although easier to set up, will not be the most computationally efficient approach.

One important factor that must be considered while simulating nonlinear elements is that the nonlinearity will produce bandwidth expansion and the sampling rate has to be set high enough to capture the effects of the bandwidth expansion.

Time Invariance

As stated earlier, all systems, components, and processes will exhibit time-varying behavior to some extent when observed over a long time period. Whether or not to use a time-varying model is guided by a number of factors.

In many applications such as modeling and simulating an optical fiber, the fiber characteristics may change very little over the lifespan of the communication system, and hence a time-invariant model will be adequate. In other cases, the time variations might be significant but the rate of change of the time variations may be very slow compared to the bandwidth of the time-invariant parts of the system. The quasi-static approximation is valid in this situation and the simulation can be carried out using fixed snapshots of the time-varying parts and the results can be averaged (i.e., partitioning and averaging). In these two cases, the performance measure of interest is some long-term average rather than the dynamic behavior.

The third approach that is warranted sometimes is the dynamic simulation of the time variations. This approach is used when the time variations are "rapid" and the performance measure of interest is based on the dynamic or transient behavior of the system is of interest. An example of this is the acquisition and tracking behavior of the synchronization subsystem time in a burst mode communication system operating over a fast fading channel. The simulation model for this case will be a tapped delay line with time-varying tap gains, which are often modeled as filtered random processes.

While the tapped delay line simulation model for time-varying systems is straightforward to derive and implement, two factors must be taken into account. First, there might be significant spectral spreading due to time variations. The result of this spectral spreading will lead to a requirement for higher sampling rates. In addition, the order of time-varying blocks cannot be changed, since the commutative property does not hold for time-varying systems.

Memory

If the instantaneous output $y[k]$ of a component depends on the instantaneous input $x[k]$ (or $x[k-j]$), the component is memoryless; otherwise the component has memory. Filters, due to their frequency-selective behavior, have memory (frequency-selective behavior and memory are synonymous). Also, some types of nonlinearities have memory and there are many models available for simulating such components. Care must be exercised in implementing models with memory with respect to storing the internal states of the model such that the model can be reentrant. For example, when a generic filter model is used in several instances in a block diagram, the internal state of each instance of the filter must be stored separately so that when a filter model is invoked several times during a simulation it is always entered with its previous state intact.

Time-Domain and Frequency-Domain Simulations

The input-output relationship of a functional block, or system component, can be modeled and simulated in either the time domain or in the frequency domain. The computational burden of the two approaches for linear blocks is often approximately equivalent,[1] and the preferred domain of implementation will depend on the domain in which the specifications are initially provided. For example, if a filter is specified in terms of measured frequency response, it is natural and convenient to implement the filtering operation within the model in frequency domain. For nonlinear components, the specifications and implementation are almost always performed in the time domain.

While the implementation of a model can be in either the time or frequency domain, it is common practice to use time domain samples to represent the input and output signals. Frequency domain models, such as filters simulated using the fast Fourier transform (FFT), will require internal buffering of the time domain input samples, taking the transform of the input vector stored in the buffer followed by frequency domain processing, inverse FFT, and buffering at the output. Buffering is required, since taking the transform is a block-processing operation based on a set of samples rather than on a single sample. During simulation, samples of the input and output may be transferred in and out of the buffers one sample at a time or in blocks of N samples.

Block Processing

A model may be implemented to accept and process one time domain sample at a time or a block of N time domain samples per invocation of the model. The computational efficiency of the two methods will depend on the relative complexity of the model and the overhead associated with the invocation of the model. If the model is simple with a small number of internal states and parameters, or if the overhead of calling or invoking a model is small compared to the computations performed inside the model, then it is convenient and efficient to invoke the model on a sample-by-sample basis. When the invocation overhead is large, it is computationally more efficient to use a block or vector-processing approach where the model is called with an input vector of size N.

Block processing requires buffering and a careful interface to the blocks preceding and following the block in question. Block processing introduces a time delay of $N * T_s$ seconds, since the output cannot be computed until all N samples of the input are accumulated and transferred into the model. Inclusion of a block input-output model in a feedback loop will produce incorrect results because of the large processing delay. Also, if block-processing models are intermixed with nonlinear models, it may be necessary to revert to sample-by-sample processing for the non-

[1] At first reading this statement may not seem obvious. The computational burden associated with the convolutation required for simulaion of systems with memory in the time domain can be reduced by truncating the impulse response and using an infinite impulse response (IIR) filter structure. Some preprocessing must be done to establish the filter structure but this need only be done once. Overhead is also associated with the frequency domain approach because of the required buffering.

linear elements, since blocking the input and processing on a block-by-block basis assumes that superposition holds, which is not the case with nonlinear blocks. An example of this situation occurs when a nonlinearity appears between two overlap and add type FFT filters. The overlap and add FFT filter is based on the linearity principle. The technique is to compute the filter response to nonoverlapping blocks of input samples and add the responses at the output. With a nonlinearity after the filter, this block processing cannot be carried through the nonlinearity, since superposition does not apply across the nonlinearity. To do this processing correctly, the response due to each block of input samples has to be added at the output of the first filter, and the nonlinearity model then processes the added output of the first filter on a sample-by-sample basis. The output of the nonlinearity can then be processed by the second filter using the overlap and add method of block processing.

Another factor that has to be taken into account with block processing is scheduling. If different models in a system use different input and output block sizes, then the simulation framework should be capable of scheduling the order and frequency of invocations properly. Otherwise, the user has to take care of the scheduling.

Variable Step-Size Processing

Multirate sampling is used in simulations if a system model includes processes and phenomena of widely differing bandwidths. With multirate sampling, each signal is sampled and processed at a rate consistent with its bandwidth, and this leads to significant improvements in computational efficiency. When multirate sampling is used, interpolation and or decimation might be necessary to interface the sample streams with different sampling rates.

Variable step-size processing is also used often to improve computational efficiency. This approach is commonly used in numerical integration routines for solving linear and nonlinear differential equations. If the underlying differential equations and their solutions are well behaved, then this approach will reduce the computational load significantly. When a variable step size is used in a model, the output has to be buffered and resampled if the following blocks use a uniform step size.

Parameterization

One of the primary uses of simulation is design optimization, which in most cases reduces to finding the optimum value of critical parameters such as the bandwidth of a receiver filter, the operating point of an amplifier, and the number of quantization levels to be used in the receiver. In order to do this, models have to parameterized properly and the key design parameters should be made visible externally; that is, models should have "external knobs" that can be used to adjust the deign parameter iteratively during simulations. One consideration that must be taken into account is the number of parameters for a given model. In general this number should be kept to a minimum, since a complex communication system will involve a large number of components. If each component has a large number of external parameters, the overall parameter space becomes very large. In this case it will be extremely difficult

to optimize the design using simulation. Also, measurement of parameter values and validation becomes easier with a smaller number of parameters.

Interface to Other Blocks

While the modeling and simulation approach used for each component depends on the nature of the component being modeled and simulated, considerable attention must be paid to the interfacing of a given model to the models of other blocks. Since the block diagram of a system might consist of an arbitrary set of blocks that are interconnected in an arbitrary manner, consistency and compatibility must be ensured either by the simulation framework and/or by the user. This task can be made easier if the models of individual blocks are constructed with well-defined and well-documented interfaces. Inconsistencies might result for a number of reasons. These include different domains of processing, signal types, block size, step size, multirate sampling, inconsistent parameter specification within different blocks, and many other reasons outlined in the preceding sections.

Many of the difficulties in simulating a complex system-level model arise because of these inconsistencies. Hence it is necessary to exercise care in formulating the overall simulation model and the selection of the individual blocks and their parameters in the context of not only the individual models but also in the context of the overall model.

2.2.3 Random Process Modeling and Simulation

Assuming that we have a system model with the highest level of abstraction and least complexity, we now turn our attention to aspects of methodology that apply to modeling and generating the input waveforms (signals, noise, and interference) that drive the simulation model. Since the basic goal of waveform-level simulation of communication systems is to emulate the waveforms in the system and compute some measures of waveform fidelity, it is important that close attention be paid to the modeling and simulation of the input waveforms or stimuli.

In a communication systems, information-bearing waveforms as well as unwanted noise and interference are random in nature and they are modeled by random processes. Stationarity is almost universally assumed, since it can be justified in many cases based on the nature of the signals or sequences being modeled. For example, the statistics of the symbol sequence in an English-language text have not changed over hundreds of years and hence a stationary model can be justified.

Stationary random processes are characterized by multidimensional probability distributions, which are difficult to specify, and it is also very difficult in the general case to generate sampled values of a stationary process with an arbitrary n-dimensional distribution. One notable exception is the stationary Gaussian process, which is completely specified by the second-order distribution (whose parameters are the mean and the autocorrelation function). For non-Gaussian processes it is a common practice to limit the specification to second-order distributions.

Sampled values of random processes that are used to drive the simulations are sequences of random numbers that are generated using random number generators.

Algorithms for generating random sequences with arbitrary distributions (first and second order) and correlation functions are presented elsewhere in the book. We discuss below some aspects of methodology that apply the modeling and generation of sampled values of random process.

Gaussian Approximation

Two concepts simplify the stimuli generation considerably. The first one is the Gaussian approximation, which can be invoked via the central limit theorem, which states that the summed effect due to a large number of independent causes tends to a Gaussian process. In other words, $Y = X_1 + X_2 + \cdots + X_n$ approaches a Gaussian distribution for large n assuming that the component variables X_i are independent. Thus, the noise picked up by the antenna of a receiver, contributed by a large number of sources, can be approximated by a Gaussian process. Similarly, interference from a large number of users can also be approximated by a Gaussian process, and hence it is not necessary to individually generate the signals from a large number of users and sum them. The net result can be duplicated by the output of a single Gaussian random number generator.

Equivalent Process Representation

The second concept deals with the notion of equivalent process representation, which can be stated as follows. Suppose an input random process $X(t)$ goes through n blocks and appears at the output of the n^{th} block as a process $Y(t)$. If by some means (through a rigorous analysis or by approximation or through simulation itself) we can deduce the properties of the process $Y(t)$, then for all subsequent processing that follows the n^{th} block we can simply inject a sequence that represents the sampled values of $Y(t)$, thus eliminating the need to generate and process sampled values of $X(t)$ through n blocks. When $X(t)$ is Gaussian and the blocks are linear it can be shown analytically that $Y(t)$ will also be Gaussian. The parameters of the process $Y(t)$ can be derived analytically or through a simulation of $X(t)$ passing through the n blocks. Unfortunately, it is difficult to derive the properties of $Y(t)$ analytically when $X(t)$ is arbitrary and or the blocks are nonlinear. In such cases simulation can be used to estimate the properties of $Y(t)$ and the estimated properties can be used to generate the equivalent process.

This approach is used to represent phase noise in communication systems as well as timing and phase jitter produced by synchronization subsystems. The most commonly used assumption here is that these processes are stationary Gaussian. For the front-end noise process, the power spectral density (PSD) that will be assumed to be white. For other processes, the PSD is assumed to be given in closed form as a ratio of polynomials in f^2, in which case the process can be generated by filtering a white Gaussian process with a filter whose transfer function can be obtained using spectral factorization methods. In the case of arbitrary PSD functions, such as the case for the doppler PSD of fading channels, we can either approximate the spectrum by a ratio of polynomials in f^2 and apply the spectral factoring method to obtain the transfer function of a filter or fit an autoregressive moving average

(ARMA) model directly to the PSD to obtain the coefficients of a recursive filter that will produce the desired PSD.

Non-Gaussian processes with arbitrary PSDs are harder to synthesize and simulate. A method for handling this case, though very difficult to apply, may be found Chapter 7.

Slow versus Fast Processes

It is not uncommon to have in a communication system many different random phenomena with widely different bandwidths or "time constants." If the bandwidth of one process is much different from the other, say, a difference of several orders of magnitude, then one of the following two approaches should be taken in order to reduce simulation time. In the first approach, which is applicable when the bandwidths differ by several orders of magnitude, the problem should be partitioned and conditioned on the slow process and the simulation should be executed separately if possible with the value of the slow process held constant while the portion of the system dealing with the faster process is simulated. There is no need to generate sampled values of the slow process during the simulation, since its value will change very little over the duration of a large number of samples of the faster process. This approach is commonly used to simulate the performance of communication systems operating over slowly fading channels.

A second approach that can be considered is multirate sampling, which is applicable when the bandwidths of the processes differ, say, by one or two orders of magnitude. Here the processes are sampled at different rates consistent with their bandwidths so that the number of samples generated during the simulation interval is proportional to the bandwidths of the respective processes. Interpolation and decimation can be used when necessary to mix these signals together at some point in the system. (If the bandwidths differ by less than an order of magnitude, then the overhead associated with multirate sampling will offset any computational savings that might result.)

2.3 Performance Estimation

One of the main objectives of simulation is performance estimation. For communication systems, the primary measure of performance is the output signal-to-noise ratio $(S/N)_o$ for analog communication systems, and bit error rate (BER) or frame error rate (FER) for digital communication systems. Signal-to-noise ratio is also a secondary measure of performance in digital communication systems. Performance measures are estimated using Monte Carlo techniques. To illustrate the methodology aspects of Monte Carlo simulation and performance estimation, let us consider the problem of estimating the probability of error in a digital communication system. The simulation model for the candidate system in shown in Figure 2.4. Note that the simulation model in this figure is a simplified model of the system shown in Figure 2.1. Some blocks such as the synchronization and coding are left out of this block diagram. Synchronization is either assumed to be ideal, or the effects of imperfect synchronization are handled though conditioning and partitioning as

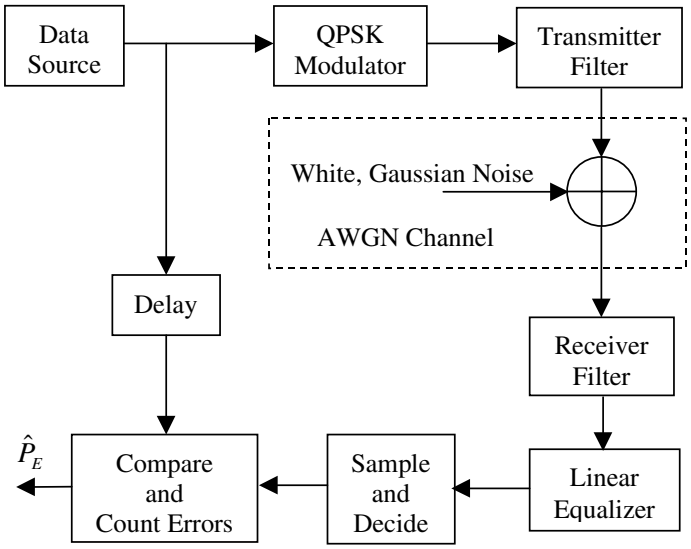


Figure 2.4 Simulation model for BER determination.

explained in the preceding section. Coding is also omitted, since our focus here is on computing the uncoded probability of error in the system; the effects of coding are handled separately as outlined in Chapter 8. Also, the channel is assumed to be slowly varying or quasi-static, and the equalizer weights are "frozen" in place after they have converged to steady-state values.

The bit error rate performance of the system can be simulated using a random bit sequence as the modulator input and it is not necessary to include an actual data source, source decoder, error control coder, and interleaver in the overall simulation model—the net effect of these blocks is to produce a random binary sequence, and therefore these functional blocks can be omitted from the block diagram and be replaced by a block that produces as its output a random binary sequence.

These simplifications are typical of what is usually done prior to executing a simulation for performance estimation. The primary motivation for the simplification is the reduction of simulation time, which could be very long in the case of performance estimation involving low error rates. Hence, only those components that might have a significant impact on performance are included in the block diagram, which is reduced to as minimal a form as possible.

The BER is determined using the Monte Carlo method. As mentioned in the previous chapter, the bit (or symbol) error probability cannot be determined but rather is estimated by passing N symbols through the system and counting errors. Assuming that N_e errors are counted in passing N symbols through the system, the BER is

$$\widehat{P}_E = \frac{N_e}{N} \tag{2.6}$$

which is an estimate of the error probability

$$P_E = \lim_{N\to\infty} \frac{N_e}{N} \tag{2.7}$$

In general, the Monte Carlo estimate is unbiased. Small values of N give error estimates with large variance and large values of N give error estimates with small variance. The estimate $\widehat{P}_E$ converges to P_E, the true value of the error probability, as $N \to \infty$, and we therefore typically use the largest practical value of N. A natural tradeoff exists between simulation accuracy and the simulation run time. In a later chapter we consider techniques for reducing the variance of the error estimate for a fixed value of N. These techniques, collectively known as variance reduction techniques, require a combination of analysis and simulation and must be applied with considerable care.

Two functional blocks appear in the simulation model, Figure 2.4, that are not part of the physical system being analyzed. These are the blocks labeled "Delay" and "Compare and Count Errors." The "Compare and Count Errors" block has a clear function. The received symbols are compared to the original data symbols so that the error count, N_e, can be determined. A moment's thought shows the need for the block labeled "Delay." A number of the functional blocks in the communication system have a nonzero phase response, and therefore a signal passing through these functional blocks incurs a time delay. As a result, the signal at the output of the data source must be delayed so that a given symbol at the output of the receiver is compared to the corresponding symbol at the output of the data source. The determination of this delay must be done with care. If the delay is not exactly correct, the resulting BER estimate will no longer be unbiased and, on average, the estimated BER will exceed the true probability of error. The determination of the appropriate value of the "delay" is part of the important part of a procedure known as calibration. Calibration of a simulation is a procedure performed to ensure that signal levels, noise levels, delays, and other important system attributes in the simulation of a system match the corresponding attributes of the system being simulated. This important aspect of simulation is examined in detail in Chapter 10 when we consider Monte Carlo methods in detail.

The delay block is realized as a delay line of variable length. The length for a specific application is selected to give the proper alignment of demodulated symbols at the receiver output with the symbols at the data source output. The delay is usually quantized to be an integer number of sampling periods. Having very fine control over the delay requires having very short sampling periods or, equivalently, having very large sampling frequency for the simulation. Increasing the sampling frequency increases the required time to execute the simulation. This is a typical result, and we will see in our future studies that minimizing the error sources in a simulation has the negative effect of increasing the simulation run time. Effectively controlling the many tradeoffs involved in developing an effective simulation is, as previously noted, part of the "art" of simulation.

2.4 Summary

This chapter has presented the basic methodology that is used for simulation development. Whether time driven or event driven, simulations must be properly developed and organized if reliable, and verifiable, results are to be obtained. The concepts presented in this chapter outline the main considerations that play a significant role in this process. The organizational structure of a simulation often mirrors the approach used for actual system design. However, many tricks of the trade have been discussed that can be applied to a simulation in order to ensure that the simulation results accurately reflect the operation of the system under design or evaluation. Throughout the remainder of this book, we will make use of the techniques outlined in this chapter. As pointed out early in this chapter, the student is encouraged to revisit this material from time to time as the study of simulation progresses.

2.5 Further Reading

The references given in Chapter 1 are also appropriate for the material given here.

2.6 Problems

2.1 Read the table of contents of the two books, *Simulation of Communication Systems* by Jeruchim, Balaban, and Shanmugan, and *Simulation Techniques* by Gardner and Baker (citations to these two books are given in the Further Reading section of Chapter 1). Compare the topics covered in these two books with the ones covered in this text.

2.2 Read and summarize the following tutorial articles dealing with simulation:

(a) W. H. Tranter and K. L. Kosbar, "Simulation of Communication Systems," *IEEE Communications Magazine*, July 1994, pp. 26–36.

(b) K. Sam Shanmugan, "Simulation and Implementation Tools for Communication and Signal Processing Systems," *IEEE Communications Magazine*, July 1994, pp. 36–41.

(c) B. D. Woerner, J. H. Reed, and T. S. Rappaport, "Simulation Issues for Future Wireless Modems," *IEEE Communications Magazine*, July 1994, pp. 42–53.

(d) K. Sam Shanmugan, "Simulation of Communication Systems," *Wiley Encyclopedia of Communications*, ed. John Proakis, New York: Wiley, 2001.

2.3 The *IEEE Journal on Selected Areas in Communications* periodically publishes issues on computer-aided modeling and analysis of communication systems. Starting with the first issue on this topic, which was published in January 1984, locate all subsequent issues on this topic. Scan the articles in these issues and write a brief paper covering:

(a) Types of analysis and design problems addressed using simulation

(b) Evolution of various simulation techniques and methodologies over the past 15 years

(c) Evolution of simulation frameworks over the past 15 years

2.4 Find the Websites for the following software packages: MATLAB, Labview, SPW, ADS, OPNET, and others. From these Websites download general information about these packages. Collect and summarize information about the following aspects of these packages:

(a) Simulation engines for each framework (time driven, event driven, data-flow, etc.)

(b) Model libraries and toolboxes available to support the development of simulation programs

(c) Model-building framework and debugging

(d) Interactive simulation capabilities

(e) Signal analysis and display capabilities

(f) Availability of online tutorials and demos (if there are online demos, exercise them and comment about what you learned from the tutorial)

2.5 Consider a passive single-pole RC lowpass filter. Describe the various hierarchical representations of the filter in terms of "layers" as discussed earlier in this chapter.

2.6 Describe how conditioning could be used to simulate the impact of timing errors in a matched-filter data detector.

PART II

Fundamental Concepts and Techniques

Chapter 3

SAMPLING AND QUANTIZING

Our main purpose in this book is to study the basic techniques required to accurately simulate communication systems using digital computers. In most communications applications, waveforms are generated and processed through the system under study. The computer, of course, can only process numbers representing samples of the waveforms of interest. In addition, since the computer has finite word length, the sample values have finite precision. In other words, the sample values are quantized. Thus, sampling and quantizing are underlying operations in all digital simulations, and each of these operations give rise to errors in the simulation results. The complete elimination of these error sources is not possible and tradeoffs are often required. We will see that the best we can do is to minimize the effects of sampling and quantizing on simulation accuracy. It is worth noting that many physical systems make use of digital signal-processing (DSP) techniques and also suffer from the effects of sampling and quantizing errors.

3.1 Sampling

As illustrated in Figure 3.1, a digital signal is formed from an analog signal by the operations of sampling, quantizing, and encoding. The analog signal, denoted $x(t)$, is continuous in both time and amplitude. The result of the sampling operation is a signal that is still continuous in amplitude but discrete in time. Such signals are often referred to as sampled-data signals. A digital signal is formed from a sampled data-signal by encoding the time-sampled values onto a finite set of values. As we will see, errors are usually induced at each step of this process.

3.1.1 The Lowpass Sampling Theorem

The first step in forming a digital signal from a continuous-time signal, $x(t)$, is to sample $x(t)$ at a uniformly spaced series of points in time to produce the sample values, $x_s(t) = x(kT_s) = x[k]$.[1] The parameter T_s is known as the sampling period and is the inverse of the sampling frequency, f_s.

A model for the sampling operation is illustrated in Figure 3.2. The signal $x(t)$ is multiplied by a periodic pulse $p(t)$ to form the sampled signal $x_s(t)$. In other words

$$x_s(t) = x(t)\,p(t) \tag{3.1}$$

The signal $p(t)$ is referred to as the sampling function. The sampling function is assumed to be a narrow pulse, which is either zero or one. Thus $x_s(t) = x(t)$ when $p(t) = 1$, and $x_s(t) = 0$ when $p(t) = 0$. We will see shortly that only the period of the sampling function $p(t)$ is significant and the waveshape of $p(t)$ is arbitrary. The pulse type function illustrated in Figure 3.2 simply provides us with the intuitively pleasing notion of a switch periodically closing at the sampling instants.

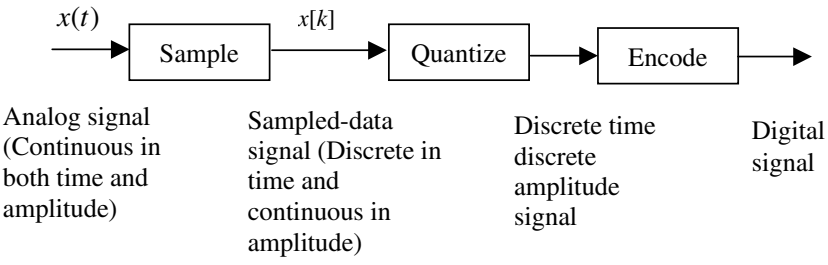
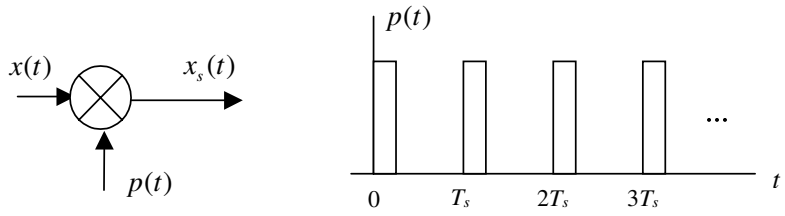


Figure 3.1 Sampling, quantizing, and encoding.

[1] Once a signal is sampled, the sample values are a function of the index k and the notation $x[k]$ is used. This notation, made popular by Oppenheim and Schafer [1], is commonly used in the DSP literature. Since the square brackets implies a sampling operation the subscript s is not needed to denote sampling. The value of $x[\cdot]$ is defined only for integer arguments.



(a) Sampling operation (b) Sampling function

Figure 3.2 Sampling operation and sampling function.

Since $p\,(t)$ is a periodic signal, it can be represented by the Fourier series

$$p\,(t) = \sum_{n=-\infty}^{\infty} C_n \exp(j2\pi n f_s t) \tag{3.2}$$

in which the Fourier coefficients are given by

$$C_n = \frac{1}{T_s} \int_{-T_s/2}^{T_s/2} p\,(t) \exp(-j2\pi n f_s t)\, dt \tag{3.3}$$

Substituting (3.2) into (3.1) gives

$$x_s\,(t) = x(t) \sum_{n=-\infty}^{\infty} C_n \exp(j2\pi n f_s t) \tag{3.4}$$

for the sampled signal.

In order to derive the sampling theorem and thereby show that under appropriate conditions $x\,(t)$ is completely represented by the samples $x\,(kT_s)$, we must derive the spectrum of $x_s\,(t)$ and show that $x(t)$ can indeed be reconstructed from $x_s(t)$. The Fourier transform of the sampled signal is

$$X_s\,(f) = \int_{-\infty}^{\infty} x(t) \sum_{n=-\infty}^{\infty} C_n \exp(j2\pi n f_s t) \exp(-j2\pi f t)\, dt \tag{3.5}$$

which, upon interchanging integration and summation, becomes

$$X_s\,(f) = \sum_{n=-\infty}^{\infty} C_n \int_{-\infty}^{\infty} x\,(t) \exp\left[-j2\pi(f - n f_s)t\right]\, dt \tag{3.6}$$

Since the Fourier transform of the continuous-time signal $x(t)$ is

$$X\,(f) = \int_{-\infty}^{\infty} x\,(t) \exp(-j2\pi f t)\, dt \tag{3.7}$$

it follows from (3.6) that the Fourier transform of the sampled signal can be written

$$X_s(f) = \sum_{n=-\infty}^{\infty} C_n X(f - nf_s) \tag{3.8}$$

We therefore see that the effect of sampling a continuous-time signal is to reproduce the spectrum of the signal being sampled about dc ($f = 0$) and all harmonics of the sampling frequency ($f = nf_s$). The translated spectra are weighted by the corresponding Fourier coefficient in the series expansion of the sampling pulse $p(t)$.

The next, and final, step in the development of the sampling theorem is to define $p(t)$. Since the samples are assumed to be taken instantaneously, a suitable definition of $p(t)$ is

$$p(t) = \sum_{k=-\infty}^{\infty} \delta(t - kT_s) \tag{3.9}$$

This is known as impulse function sampling in which the sample values are represented by the weights of the impulse functions. Substitution of (3.9) into (3.3) gives

$$C_n = \frac{1}{T_s} \int_{-T_s/2}^{T_s/2} \delta(t) \exp(-j2\pi n f_s t)\, dt \tag{3.10}$$

Applying the sifting property of the delta function gives

$$C_n = \frac{1}{T_s} = f_s \tag{3.11}$$

Using this result in (3.2) shows that the Fourier transform of $p(t)$ can be represented by

$$P(f) = f_s \sum_{n=-\infty}^{\infty} \delta(f - nf_s) \tag{3.12}$$

For impulse function sampling $C_n = f_s$ for all n. Thus, using (3.8) the spectrum of the sampled signal becomes

$$X_s(f) = f_s \sum_{n=-\infty}^{\infty} X(f - nf_s) \tag{3.13}$$

Note that this result could have also been obtained from the expression

$$X_s(f) = X(f) \circledast P(f) \tag{3.14}$$

where $\circledast$ denotes convolution. The generation of $X_s(f)$ using (3.14) is illustrated in Figure 3.3 for the case of a bandlimited signal.

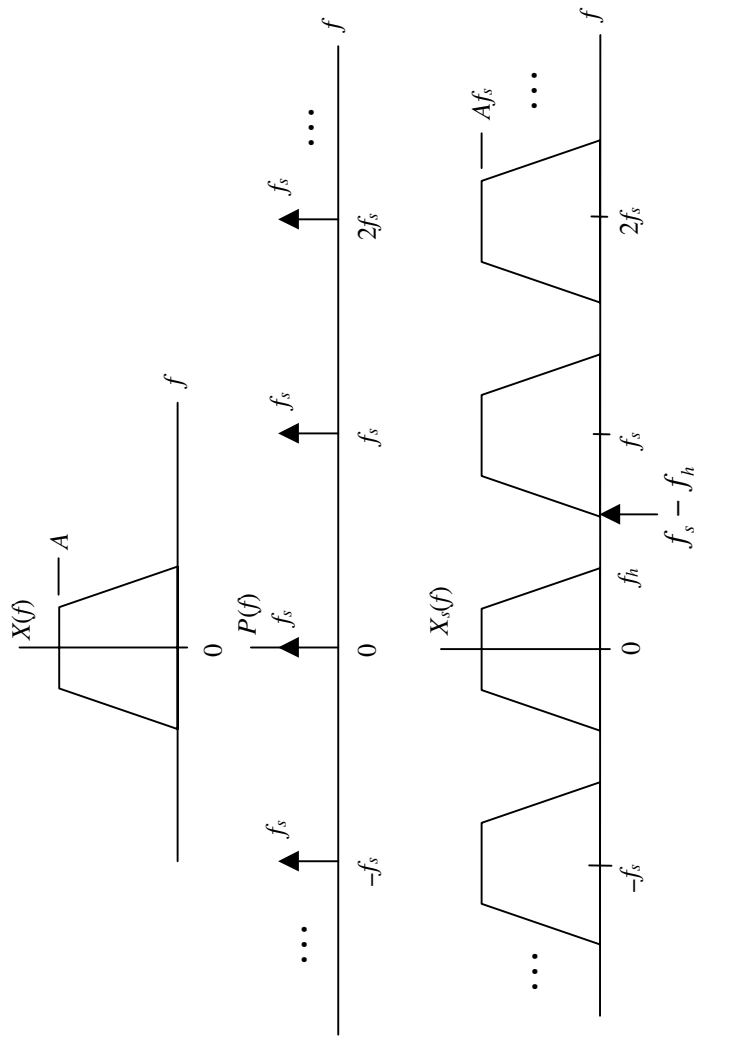


Figure 3.3 Sampling viewed in the frequency domain.

The sampling theorem can be developed from observation of Figure 3.3. In order for the samples $x(nT_s)$ to contain all of the information in the continuous-time signal $x(t)$, so that no information is lost in the sampling process, the sampling must be performed so that $x(t)$ can be reconstructed without error from the samples $x(nT_s)$. We will see that reconstruction of $x(t)$ from $x_s(t)$ is accomplished by extracting the $n = 0$ term from $X_s(f)$ by lowpass filtering. Accomplishing reconstruction without error therefore requires that the portion of the spectrum of $X_s(f)$ about $f = \pm f_s$ [the $n = \pm 1$ terms in (3.13)] not overlap the portion of the spectrum about $f = 0$ [the $n = 0$ term in (3.13)]. In other words, all translated spectra in (3.13) must be disjoint. This requires that $f_s - f_h > f_h$ or $f_s > 2f_h$, which proves the sampling theorem for lowpass signals.

Theorem 1 *A bandlimited signal may be reconstructed without error from samples of the signal if the sampling frequency f_s exceeds $2f_h$, where f_h is the highest frequency present in the signal being sampled.*

While this theorem is usually referred to as the lowpass sampling theorem, it also works for bandpass signals. However, applying the lowpass sampling theorem to bandpass signals usually results in excessively high sampling frequencies. Sampling bandpass signals is the topic of a later section.

If $f_s < 2f_h$ the spectra centered on $f = \pm f_s$ overlap the spectrum centered on $f = 0$ and the output of the reconstruction filter, as illustrated in Figure 3.4, will be a distorted version of $x(t)$. This distortion is referred to as aliasing. The effect of aliasing is also illustrated in Figure 3.4, assuming that the spectrum of $x(t)$ is real.

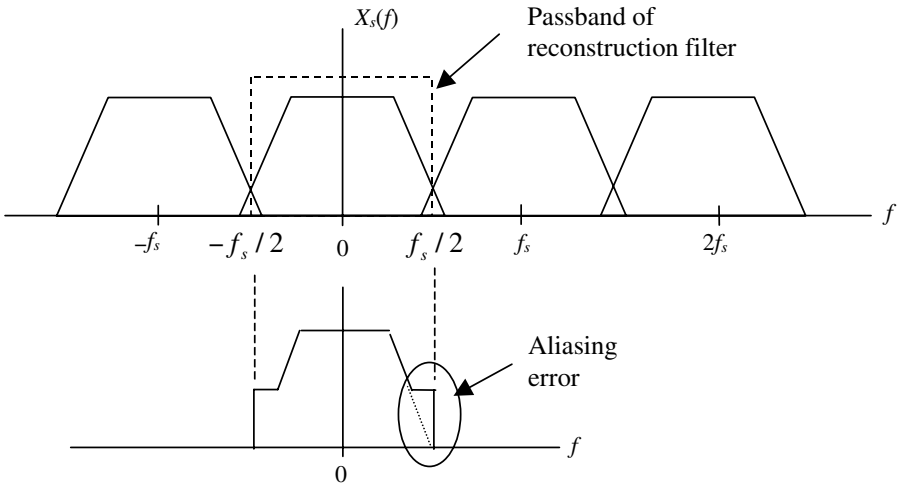


Figure 3.4 Illustration of undersampling leading to aliasing error.

3.1.2 Sampling Lowpass Random Signals

The waveform $x(t)$ in the preceding discussion was assumed to be a deterministic finite-energy signal. As a result of these assumptions, the Fourier transform exists and the sampling theorem could be based on the spectrum (the Fourier transform) of the signal. In most of our applications throughout this book it will be more natural to assume that the simulation processes sample functions of a random process. Therefore, instead of selecting a sampling frequency based on the Fourier transform of the signal to be sampled, the selection of an appropriate sampling frequency must be based on the power spectral density (PSD) of the sampling frequency.

For the case of random signals we write

$$X_s(t) = X(t)P(t) \tag{3.15}$$

where the sampling function $P(t)$ is written

$$P(t) = \sum_{k=-\infty}^{\infty} \delta(t - kT_s - D) \tag{3.16}$$

in which D is a random variable independent of $X(t)$ and uniformly distributed in $(0, T_s)$. Note the similarity of (3.15) and (3.1) and the similarity of (3.16) and (3.9). There are only two essential differences. First, uppercase letters are used in the time functions $X(t)$, $P(t)$, and $X_s(t)$ to remind us that they represent random processes. The other difference is the use of the random variable D in (3.16). The effect of D is to ensure that $X_s(t)$ is a *stationary* random process. Without the inclusion of D the sampled signal is *cyclostationary*. The effect of D is to make the time origin of $P(t)$ random but fixed.

The power spectral density of $X_s(t)$ is found by first determining the autocorrelation function of

$$X_s(t) = X(t) \sum_{k=-\infty}^{\infty} \delta(t - kT_s - D) \tag{3.17}$$

The Fourier transform of the resulting autocorrelation function gives the PSD of $X_s(t)$, which is [2]

$$S_{X_s}(f) = f_s^2 \sum_{n=-\infty}^{\infty} S_X\left(f - nf_s\right) \tag{3.18}$$

where $S_X(f)$ denotes the PSD of $X(t)$. Note the similarity of (3.18) and (3.13). Also note that Figures 3.3 and 3.4 apply if the spectra are PSDs corresponding to $X(t)$ and if the axes are labeled accordingly. Note that the sampling theorem as previously derived still holds, and therefore the signal must be sampled at a frequency exceeding twice the sampling frequency if aliasing is to be avoided.

3.1.3 Bandpass Sampling

We now consider the problem of sampling bandpass signals. There are a number of strategies that can be used for representing bandpass signals by a set of samples. In the following sections we consider the two most common methods.

The Bandpass Sampling Theorem

The bandpass sampling theorem for real bandpass signals is stated as follows [2]:

Theorem 2 *If a bandpass signal has bandwidth B and highest frequency f_h, the signal can be sampled and reconstructed using a sampling frequency of $f_s = 2f_h/m$, where m is the largest integer not exceeding f_h/B. All higher sampling frequencies are not necessarily usable unless they exceed $2f_h$, which is the value of f_s dictated by the lowpass sampling theorem.*

A plot of the normalized sampling frequency f_s/B as a function of the normalized center frequency f_0/B is illustrated in Figure 3.5, where f_0 and f_h are related by $f_h = f_0 + B/2$. We see that the allowable sampling frequency always lies in the range $2B \leq f_s \leq 4B$. However, for $f_0 \gg B$, which is typically the case, the sampling frequency dictated by the bandpass sampling theorem is approximately equal to, but is lower bounded by, $2B$.

Sampling Direct/Quadrature Signals

Suppose we have a bandpass signal expressed in the form

$$x(t) = A(t)\cos\left[2\pi f_c t + \phi(t)\right] \tag{3.19}$$

The function $A(t)$ is referred to as the envelope of the bandpass signal and the function $\phi(t)$ is referred to as the phase deviation of the bandpass signal. In most communications applications both $A(t)$ and $\phi(t)$ are lowpass signal and have bandwidths roughly on the order of the bandwidth of the information-bearing signal. Using standard trigonometric identities, the bandpass signal can be written

$$x(t) = A(t)\cos\phi(t)\cos 2\pi f_c t - A(t)\sin\phi(t)\sin 2\pi f_c t \tag{3.20}$$

or

$$x(t) = x_d(t)\cos 2\pi f_c t - x_q(t)\sin 2\pi f_c t \tag{3.21}$$

In this representation

$$x_d(t) = A(t)\cos\phi(t) \tag{3.22}$$

is called the direct (or in-phase) component and

$$x_q(t) = A(t)\sin\phi(t) \tag{3.23}$$

is the quadrature component. Since $A(t)$ and $\phi(t)$ are lowpass signals, it follows that $x_d(t)$ and $x_q(t)$ are lowpass signals and therefore must be sampled in accordance with the lowpass sampling theorem. Note from (3.21) that if $x_d(t)$, $x_q(t)$ and the carrier frequency f_c are known, the bandpass signal can be reconstructed without error. The representation of bandpass signals using direct and quadature components will be covered in detail in Chapter 4.

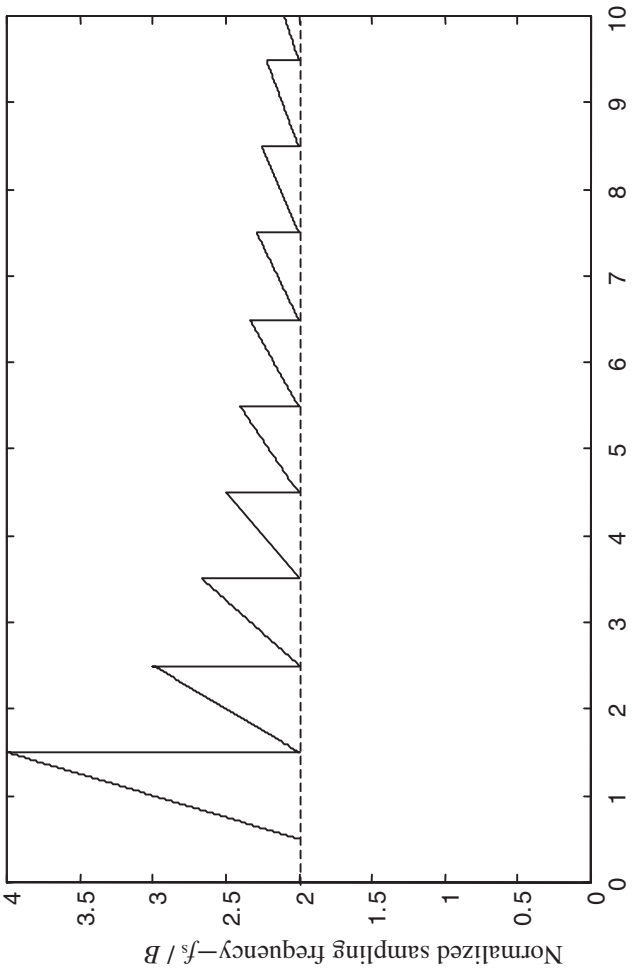


Figure 3.5 The required sampling frequency for bandpass sampling.

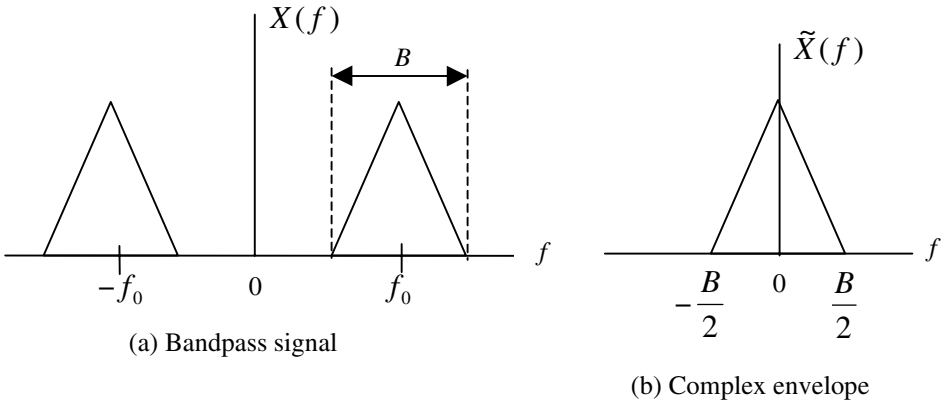


Figure 3.6 Bandpass signal and the corresponding complex envelope.

The frequency-domain representation of a bandpass signal is given in Figure 3.6(a). The complex envelope corresponding to this signal is defined by

$$\widetilde{x}(t) = x_d(t) + jx_q(t) \tag{3.24}$$

Since both $x_d(t)$ and $x_q(t)$ are lowpass signals:

$$\widetilde{X}(f) = X_d(f) + jX_q(f) \tag{3.25}$$

is lowpass as illustrated in Figure 3.6(b). In Figure 3.6 we see that $\widetilde{X}(f)$, and consequently $x_d(t)$ and $x_q(t)$ are lowpass signals. Thus $x_d(t)$ and $x_q(t)$ must be sampled according to the lowpass sampling theorem. Since the highest frequency present in $x_d(t)$ and $x_q(t)$ is $B/2$, the minimum sampling frequency for each is B. However, two lowpass signals [$x_d(t)$ and $x_q(t)$] must be sampled rather than one. As a result, a sampling rate exceeding $2B$ must be used. We therefore see that sampling the complex envelope using the lowpass sampling theorem yields the same required sampling frequency as sampling the real bandpass signal using the bandpass sampling theorem for the typical case in which $f_0 \gg B$.

Example 3.1. It follows from the preceding discussion that the bandpass signal $x(t)$ can be reconstructed without error if $x_d(t)$ and $x_q(t)$ are sampled appropriately in accordance with the lowpass sampling theorem. The advantage of representing bandpass signals by lowpass signals is obvious. Consider, for example, that we are to represent 1 second of an FM signal by a set of samples. Assume that the carrier frequency is 100 MHz (typical for the FM broadcast band) and that the highest frequency present in the modulation or information-bearing signal is 15 kHz. The bandwidth B of the modulated signal is usually approximated by Carson's rule [2], which is

$$B = 2(D+1)W \tag{3.26}$$

Assuming a deviation ratio D of 5 gives

$$B = 2(5+1)15 \quad \text{kHz} \tag{3.27}$$

which is 180 kHz (90 kHz each side of the carrier). The highest frequency present in the modulated signal is therefore 100,090 kHz. Thus, 1 second of signal requires a minimum of 200,180,000 samples according to the lowpass sampling theorem.

Now suppose we elect to represent the FM signal in direct/quadrature form. The bandwidth of both $x_d(t)$ and $x_q(t)$ is $B/2$, or 90 kHz. Thus $x_d(t)$ and $x_q(t)$ will each require at least 180,000 samples to represent 1 second of signal. This gives a total of 360,000 lowpass samples to represent 1 second of data. The savings is

$$\frac{200,180,000}{360,000} \approx 556 \tag{3.28}$$

and translates directly into a corresponding reduction in computer run time. ■

3.2 Quantizing

The quantizing process and a simple fixed-point encoding process is illustrated in Figure 3.7, which shows a continuous-time waveform and a number of samples of that waveform. The sample values are represented by the heavy dots. Each sample falls into a quantizing level. Assuming that there are n quantizing levels and that each quantizing level is represented by a b-bit binary word, it follows that

$$n = 2^b \tag{3.29}$$

In Figure 3.7 each quantizing level is mapped to three-bit ($b = 3$ and $n = 8$) digital word. After quantizing, sample values are represented by the digital word

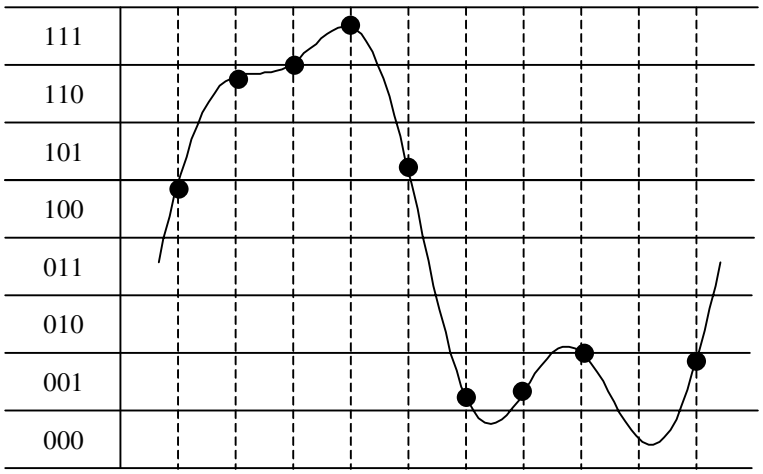


Figure 3.7 Quantizing and encoding.

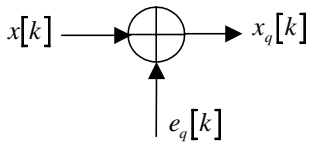


Figure 3.8 Model for quantizing error.

corresponding to the quantizing level into which the sample value falls, and digital processing of the waveform is accomplished by processing these digital words. For example, the first three sample values (from left to right) in Figure 3.7 are represented by the binary sequence 100110111.

From sampling theory we know that a continuous-time bandlimited signal, sampled at a frequency exceeding the Nyquist frequency, can be reconstructed without error from the samples. Therefore, under these conditions the sampling operation is reversible. The quantizing operation, however, is not reversible. Once sample values are quantized, only the quantizing level is maintained and therefore a random error is induced. As before, the value of the waveform at the sampling instant $t = kT_s$ is denoted $x[k]$, and the corresponding quantized value is denoted $x_q[k]$, which is

$$x_q[k] = x[k] + e_q[k] \tag{3.30}$$

where $e_q[k]$ is the error induced by the quantizing process. The quantizer model implied by (3.30) is illustrated in Figure 3.8. If the original signal is not bandlimited, the resulting digital signal contains both aliasing and quantizing errors.

The quantity of interest is the signal-to-noise ratio (SNR), where the noise is interpreted as the noise resulting from the quantizing process. The SNR due to quantizing, denoted $(SNR)_q$, is

$$(SNR)_q = \frac{S}{N_q} = \frac{E\left\{x^2[k]\right\}}{E\left\{e_q^2[k]\right\}} \tag{3.31}$$

where $E\{\cdot\}$ denotes statistical expectation and N_q is the noise power resulting from the quantizing process. In order to determine $(SNR)_q$ the probability density function of the error $e_q[k]$ must be known. The pdf of the quantizing error is a function of the format used to represent numbers in the computer. There are a wide variety of formats that can be used. The broad categories are fixed point and floating point.

Fixed-Point Arithmetic

Even though we are, for the most part, considering simulation using general-purpose computers in which numbers are represented in a floating-point format, we pause to consider quantizing errors resulting from fixed-point number representations.

There are several reasons for considering fixed-point arithmetic (quantizing). First, by considering fixed-point arithmetic, the basic mechanism by which quantizing errors arise is illustrated. Also, special-purpose simulators have been developed that use fixed-point arithmetic because fixed-point calculations can be executed much faster than floating-point calculations. In addition, power consumption is usually lower with fixed-point processors. Perhaps the most important reason for considering fixed-point arithmetic is that devices using fixed-point arithmetic must often be simulated. For example, software-based communications systems are becoming popular, since they can easily be reconfigured for different applications by simply downloading appropriate programs to the device. In order to be commercially attractive in a competitive environment, these systems must be available at the lowest possible cost. Cost is typically minimized by using fixed-point arithmetic and, in addition, fixed-point algorithms execute much faster than floating-point algorithms. We should point out that the design of these software-based devices usually starts with a simulation and, when the simulation shows that the device is properly designed and meets specifications, the simulation code is downloaded to the device.[2] In such applications, the simulation of the device and the physical device merge to a great extent. As previously mentioned, speed, cost, and power consumption requirements usually dictate that many commercial devices utilize fixed-point arithmetic, and simulation is an important tool for the design and performance evaluation of these devices.

Assume that the width of a quantizing level, as illustrated in Figure 3.7, is denoted Δ. Also assume that a sample value corresponding to a given quantizing level is assumed to be the value at the center of the quantizing level.[3] In this case the maximum value of $|e_q[k]|$ is $\Delta/2$. If the number of quantizing levels is large, corresponding to long digital wordlengths, and if the signal varies significantly from sample to sample, a given sample is equally likely to fall at any point in the quantizing level. For this case the errors due to quantizing can be assumed to be uniformly distributed and independent. The pdf (probability density function) of the quantizing error is therefore uniform over the range $[-\Delta/2, \Delta/2]$ as illustrated in Figure 3.9. Denoting the quantizing error of the k^{th} sample by $e_q[k]$, we have

$$E\{e_q[k]\} = \int_{-\Delta/2}^{\Delta/2} x \frac{1}{\Delta} dx = 0 \tag{3.32}$$

so that the quantizing error is zero mean. The variance of $e_q[k]$ is

$$E\left\{e_q^2[k]\right\} = \int_{-\Delta/2}^{\Delta/2} x^2 \frac{1}{\Delta} dx = \frac{\Delta^2}{12} \tag{3.33}$$

We now compute the signal-to-noise ratio due to quantizing.

[2] Recall the design cycle discussed in Chapter 1.

[3] In order to demonstrate basic principles, the pdf is assumed to be for a simple zero-mean process. In practice the pdf will depend on the manner in which fixed-point numbers are represented in the computer. The most common representations are sign-magnitude, ones-complement, and twos-complement [1].

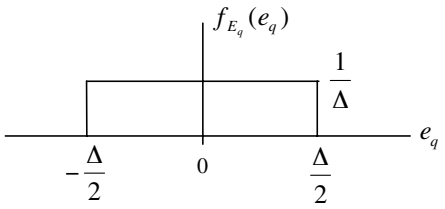


Figure 3.9 Assumed pdf of quantizing error.

Assume that a quantizer has a dynamic range D and that the word length is b. Assuming binary arithmetic, there are 2^b possible quantizing levels and the dynamic range is given by

$$D = 2^b \Delta \tag{3.34}$$

Thus

$$\Delta = D2^{-b} \tag{3.35}$$

and the noise power due to quantizing is

$$N_q = E\left\{e_q^2[k]\right\} = \frac{D^2}{12} 2^{-2b} \tag{3.36}$$

The dynamic range is determined by the peak-to-peak value of the input signal to the quantizer. If the signal power is S, the signal-to-noise ratio due to quantizing, $(SNR)_q$, is

$$(SNR)_q = \frac{S}{\left(\frac{D^2}{12}\right) 2^{-2b}} = \frac{12S}{D^2} 2^{2b} \tag{3.37}$$

Assuming the signal to be zero mean, the values of S and D are related by the *crest factor* of the underlying signal. The crest factor is defined as the ratio of the RMS, or standard deviation, of a signal to the peak value of the signal. To illustrate this relationship, assume that the underlying signal, having dynamic range (peak-to-peak value) D, lies in the range $\pm D$. Since the signal power is S, the standard deviation is $\sqrt{S}$. Therefore, the crest factor is

$$F_c = \frac{\sqrt{S}}{D/2} = \frac{2\sqrt{S}}{D} \tag{3.38}$$

Substitution of (3.38) into (3.37) gives

$$(SNR)_q = 3F_c^2 2^{2b} \tag{3.39}$$

which, in dB units, is

$$(SNR)_q = 4.7712 + 20\log_{10} F_c + 6.0206b \quad \text{dB} \tag{3.40}$$

Note that signals with a high crest factor are more immune to quantizing error than signals with a small crest factor. This result is logical, since signals with a high crest factor have a large standard deviation, which means that they are more spread out through the quantizing levels. It is, however, the word length b that has the most significant impact on $(SNR)_q$. Note that $(SNR)_q$ improves by 6 dB for each bit added to the word length.

Floating-Point Arithmetic

As mentioned previously, throughout most of this book our concern will be simulations for execution on general-purpose computers that utilize floating-point number representations. The form of a floating-point number is $\pm M * (\pm 10\hat{}E)$, where M and E are referred to as the mantissa and exponent, respectively. Where accuracy is required, 64-bit (double-precision) digital words are used and these 64 bits must be allocated between the mantissa and exponent. This allocation can have a significant effect on the result of a given computation. Fortunately, this assignment has been standardized and most, but not all, computers adhere to the standard. The ANSI/IEEE standard for floating-point arithmetic specifies that 53 bits are assigned to the mantissa and 11 bits are assigned to the exponent [3]. Fortunately, MATLAB provides a simple way to determine whether or not the IEEE standard is implemented on a given computer. One simply enters `isieee` at the MATLAB prompt, and a 1 is returned if the standard is implemented.

Since we will be using MATLAB throughout this book for developing and demonstrating simulations, it is important to consider the accuracy that can be expected. For our purposes, the most important parameters resulting from the floating-point format are the resolution (the difference between 1 and the next largest floating-point number), which is the MATLAB variable `eps`, the largest number that can be represented (`realmax` in MATLAB) and the smallest positive number that can be represented (`realmin` in MATLAB). Executing the simple MATLAB script `mparameters` tests for compliance with the IEEE floating-point standard and returns the values of each of these three important parameters. The script `mparameters` follows.

```
% File: c3_mparameters.m
format long       % display full precision
a = ['The value of isieee is ',num2str(isieee),'.'];
b = ['The value of eps is ',num2str(eps,15),'.'];
c = ['The value of realmax is ',num2str(realmax,15),'.'];
d = ['The value of realmin is ',num2str(realmin,15),'.'];
disp(a)           % display isieee
disp(b)           % display eps
disp(c)           % display realmax
disp(d)           % display realmin
```

```
format short      % restore default format
% End script file.
```

Executing the file `mparameters` on a computer that implements the IEEE floating-point standard provides the following results:

```
>> mparameters
The value of isieee is 1.
The value of eps is 2.22044604925031e-016.
The value of realmax is 1.79769313486232e+308.
The value of realmin is 2.2250738585072e-308.
```

The first result displayed (`isiee = 1`) indicates that the computer does indeed conform to the ANSI/IEEE standard for floating-point arithmetic. The next result is `eps`, which is essentially the smallest resolvable difference between two numbers. Note that `eps` is 2^{-52} (the extra bit associated with the mantissa accounts for the sign bit), which illustrates the relationship between `eps` and the word length. We see that more than 15 significant figures of accuracy are achieved. It is the value of `eps` that ties most closely to the width of the quantization level Δ that was discussed in connection with fixed-point arithmetic. Note that $\pm$`realmax` defines the dynamic range, which, in this case exceeds 600 orders of magnitude.

Example 3.2. Suppose that we use floating-point arithmetic, consistent with the ANSI/IEEE standard, to compute the value of

$$A = 1 - 0.4 - 0.3 - 0.2 - 0.1 \tag{3.41}$$

which is obviously zero. However, performing this computation in MATLAB gives the following:

```
>> a = 1-0.4-0.3-0.2-0.1
a =
 -2.7756e-017
```

We see that the error induced by floating-point arithmetic is certainly small and is probably negligible in most applications. The error is not zero, however, and the user should always keep in mind that computed results are not usually exact. ■

From this point forward we will assume that the quantizing errors resulting from floating-point calculations are negligible. While this is an appropriate (and necessary) assumption for the material contained in the remainder of this book, one should be aware that even small errors can accumulate, in certain types of calculations, to the point where the results are useless. DSP calculations in which the signal of interest is a small difference of two very large numbers are a classical example. Very large block length FFTs can give problems because of the large number of butterfly calculations that are cascaded. There are many other examples. In developing DSP algorithms care must be used to ensure that finite word length effects are minimized.

3.3 Reconstruction and Interpolation

We now consider the reconstruction of a continuous-time signal from a sequence of samples. Since a digital simulation processes only sample values, a continuous-time signal is never reconstructed from a set of samples in a simulation environment. Consideration of the reconstruction process, however, leads to the subject of interpolation, which is an important operation in a simulation environment.

The general reconstruction technique is to pass the samples through a linear filter having an impulse response $h(t)$. Thus the reconstructed waveform is given by $x_r(t) = x_s(t) \circledast h(t)$, where, as before, $\circledast$ denotes convolution. From (3.1) and (3.9) we can write

$$x_s(t) = x(t) \sum_{k=-\infty}^{\infty} \delta(t - kT_s) = \sum_{k=-\infty}^{\infty} x(kT_s)\delta(t - kT_s) \tag{3.42}$$

Thus, the reconstructed signal is given by

$$x_r(t) = \left[\sum_{k=-\infty}^{\infty} x(kT_s)\delta(t - kT_s) \right] \circledast h(t) \tag{3.43}$$

which is

$$x_r(t) = \sum_{k=-\infty}^{\infty} x(kT_s)h\left(t - kT_s\right) \tag{3.44}$$

The problem is to choose a $h(t)$ that gives satisfactory results with a reasonable level of computational burden.

3.3.1 Ideal Reconstruction

Assuming that a bandlimited signal is sampled at a rate exceeding $2f_h$, the signal may be reconstructed by passing the samples through an ideal lowpass filter having a bandwidth of $f_s/2$. This can be seen in Figure 3.10. If $f_s > 2f_h$ the spectra centered on $f = \pm f_s$ do not overlap the spectrum centered on $f = 0$. The output of the resconstruction filter is $f_s X(f)$ or, in the time domain, $f_s x(t)$. Amplitude scaling by $1/f_s = T_s$ yields $x(t)$.

It follows from Figure 3.10 that the impulse response of the reconstruction filter is

$$h(t) = T_s \int_{-f_s/2}^{f_s/2} \exp(j2\pi ft)df \tag{3.45}$$

where the scale factor of T_s has been included. Thus:

$$h(t) = T_s \frac{1}{j2\pi t} \left[\exp\left(j\pi f_s t\right) - \exp\left(j\pi f_s t\right)\right] \tag{3.46}$$

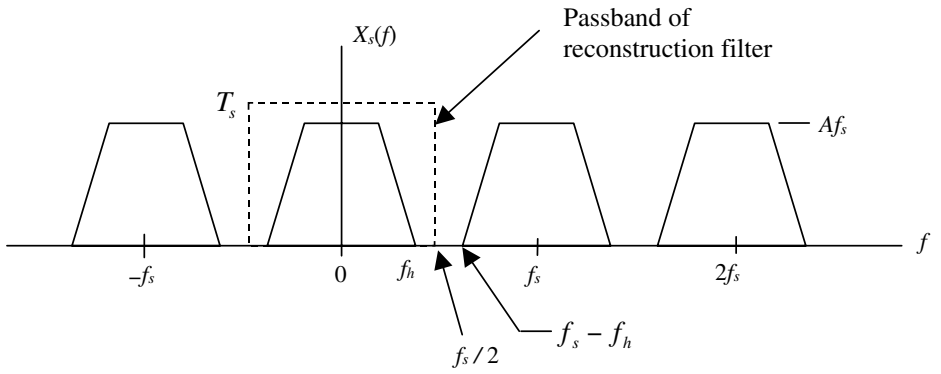


Figure 3.10 Reconstruction filter.

or

$$h(t) = T_s \frac{1}{\pi t} \sin(\pi f_s t) = \text{sinc}(f_s t) \tag{3.47}$$

Substitution into (3.44) gives

$$x_r(t) = \sum_{k=-\infty}^{\infty} x(kT_s)\,\text{sinc}\,[f_s(t - kT_s)] \tag{3.48}$$

or, in more convenient form:

$$x_r(t) = \sum_{k=-\infty}^{\infty} x(kT_s)\,\text{sinc}\left(\frac{t}{T_s} - k\right) \tag{3.49}$$

Note that since the signal $x(t)$ is assumed bandlimited and the sampling frequency is sufficiently high to ensure that aliasing errors are avoided, $x_r(t) = x(t)$. Thus, perfect reconstruction is achieved, at least in theory. Note, however, that (3.49) can never be used in practice, since the sinc(·) function is infinite in extent. Equation (3.49) will be used, however, as the building block for a practical interpolation technique in the following section.

3.3.2 Upsampling and Downsampling

Upsampling and downsampling are used in the simulation of many systems. The need for these operations is illustrated by an example. Consider the direct sequence spread-spectrum system illustrated in Figure 3.11. The data source generates a data signal having a narrowband spectrum of bandwidth W.[4] The data signal is multiplied by a wideband spreading code $c(t)$, which is represented by a binary sequence

[4]The terms *narrowband* and *wideband* are used in a relative sense.

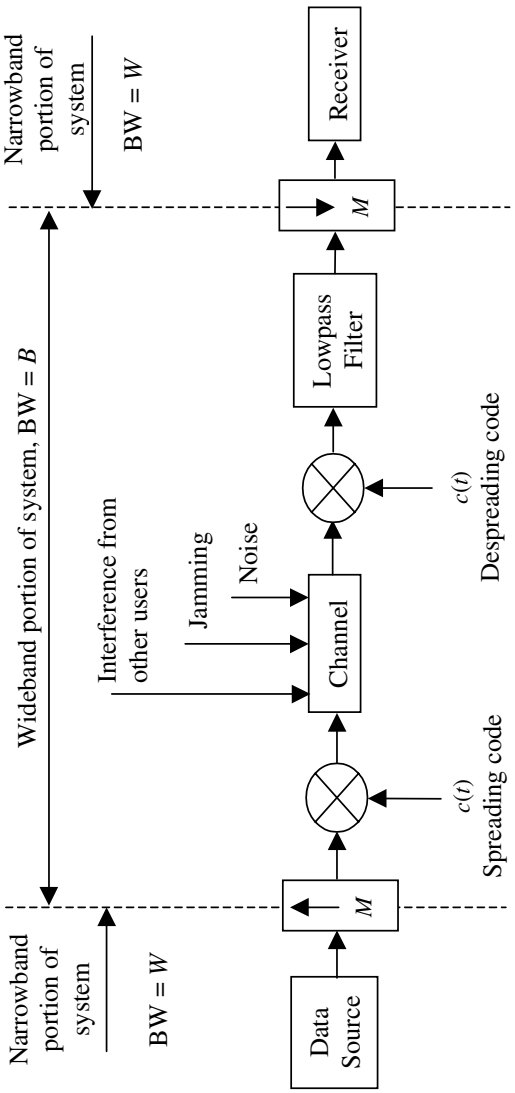


Figure 3.11 System in which upsampling and downsampling is useful.

with a symbol rate much greater than the data rate. The ratio of the spreading code rate to the data rate is called the processing gain of the system. Multiplication by the spreading code $c(t)$ generates a wideband signal, having bandwidth B. The channel imperfections may consist of interference from other users, jamming signals in military communication systems, noise, and perhaps other degradations not accounted for in Figure 3.11. The waveform at the output of the channel is multiplied by the despreading code. The spreading code is assumed to take on values ± 1 and if the spreading code and the despreading code are identical and properly synchronized, multiplying the spreading code and the despreading code gives $c^2(t) = 1$ so that the spreading and despreading codes have no effect on the channel of interest. Note that the data signal is multiplied by $c(t)$ twice and the channel impairments are multiplied by $c(t)$ only once. Thus, at the input to the lowpass filter following multiplication by the despreading code the data signal is again narrowband and all other components are wideband. The lowpass filter extracts the narrowband data signal and passes it to the receiver.

The important attribute of the system illustrated in Figure 3.11 is that both narrowband signals and wideband signals are present. If $B \gg W$, which is typically the case, sampling the narrowband signal at the sampling rate required for the wideband signal will be inefficient and will result in excessive simulation run times. Ideally, each signal should be sampled with a sampling rate appropriate for that signal.

Since signals having two different bandwidths are present in the example system, it is appropriate to use two different sampling rates. Thus the sampling rate must be increased at the boundary between the narrowband and wideband portions of the system (left-hand dashed line in Figure 3.11) and decreased at the boundary between the wideband and narrowband portions of the system (right-hand dashed line in Figure 3.11). Increasing the sampling rate is accomplished by upsampling followed by interpolation, in which new sample values are interpolated from old sample values. Reducing the sampling rate is accomplished by decimation in which unneeded samples are discarded. Upsampling is represented by a block with an upward-pointing arrow and downsampling is represented by a block with a downward-pointing arrow. The parameter M represents the factor by which the sampling period is reduced (upsampling) or increased (downsampling) by the process.

In the material to follow we will use T_s to represent the sampling period prior to the upsampling or downampling process. After upsampling or downsampling, the sampling period will be represented by T_u or T_d, respectively. The signal prior to upsampling or downsampling is denoted $x(t)$ (no subscript on x) and the signal after upsampling or downsampling will be denoted using the appropriate subscript; for example, $x_u(t)$ and $x_d(t)$.

Upsampling and Interpolation

Upsampling is the first operation illustrated in Figure 3.11 and is the process through which the sampling frequency is increased. Since upsampling reduces the sampling period by a factor of M the new sampling period T_u and the old

sampling period T_s are related by $T_u = T_s/M$. Thus, in terms of an underlying continuous-time signal $x(t)$, the upsampling process generates new sample values $x(kT_u) = x(kT_s/M)$ from the old sample values $x(kT_s)$. As an example, suppose that we construct a new set of samples by interpolating the reconstructed signal $x_r(t)$, given by (3.49) at points $t = nT_s/M$. Performing this operations gives

$$x_i(nT_u) = x(nT_s/M) = \sum_{k=-\infty}^{\infty} x(kT_s)\,\mathrm{sinc}\left(\frac{n}{M} - k\right) \tag{3.50}$$

This is not a practical interpolator, since the sinc(·) function is infinite in extent. Truncating the sinc(·) function yields

$$x_i(nT_u) = x(nT_s/M) \cong \sum_{k=-L}^{L} x(kT_s)\,\mathrm{sinc}\left(\frac{n}{M} - k\right) \tag{3.51}$$

a more practical, although not perfect, interpolator. Making L large clearly reduces the interpolation error. However, since each interpolated sample requires $2L + 1$ samples, the computational burden is often unacceptable for large L. Thus, there is a tradeoff between computational burden and accuracy. This tradeoff will be seen many times in our study of simulation. Note also that, since a causal function must be used for interpolation, a delay of LT_s is induced. This delay does not present a problem in simulation, but we must be aware of its presence.

A more practical interpolator, requiring much less computation than the sinc(·) function interpolator, is the linear interpolator. The linear interpolator, although much simpler than the sinc(·) function interpolator, can be used when the underlying signal is significantly oversampled. The impulse response of the linear interpolator is defined by

$$h[k] = \begin{cases} (M - |k|)/M, & k = 0, \pm 1, \pm 2, \cdots, \pm(M-1) \\ 0, & \text{otherwise} \end{cases} \tag{3.52}$$

Note that there are $2M - 1$ nonzero values of $h[k]$. A MATLAB program for developing $h[k]$ follows:

```
% File: c3_lininterp.m
function h=c3_lininterp(M)
h1 = zeros(1,(M-1));
for j=1:(M-1)
    h1(j) = j/M;
end
h = [0,h1,1,fliplr(h1),0];
% End of function file.
```

The upsampling operation is implemented on a discrete set of samples as a two-step process as illustrated in Figure 3.12. We first form $x_u[k]$ from $x[k]$ according to

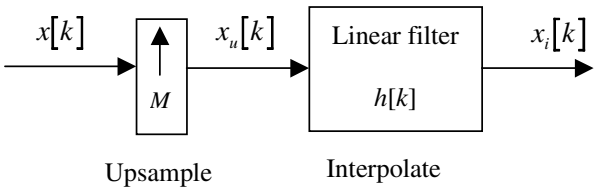


Figure 3.12 Upsampling and interpolation.

$$x_u[k] = \begin{cases} x[k/M], & k = 0, \pm M, \pm 2M, \cdots, \\ 0, & \text{otherwise} \end{cases} \tag{3.53}$$

which can be implemented with the MATLAB code

```
% File: c3_upsample.m
function out=c3_upsamp(in,M)
L = length(in);
out = zeros(1,(L-1)*M+1);
for j=1:L
 out(M*(j-1)+1)=in(j);
end
% End of function file.
```

The result of this operation is to place $M-1$ zero value samples between each sample in the original sequence $x[k]$. Interpolation is then accomplished by convolving $x_u[k]$ with $h[k]$, the impulse response of the linear interpolator. The process of linear interpolation with $M = 3$ is illustrated in Figure 3.13. Note that only two samples are used in the upsampling operation. The necessary delay is then T_s. As illustrated in Figure 3.13, the interpolated value is found by summing the contributions from

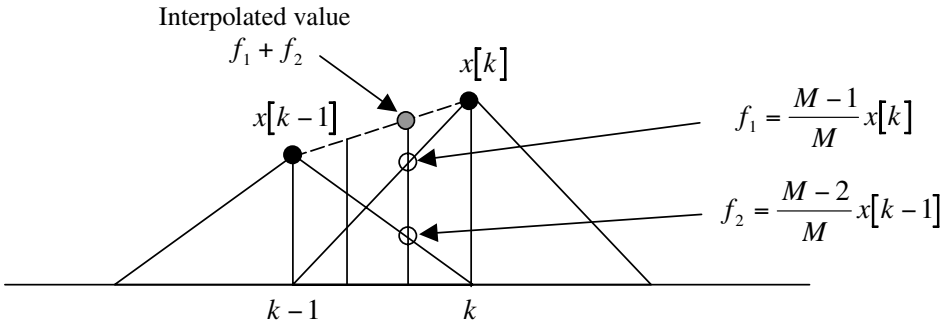


Figure 3.13 Illustration of interpolation process.

$x[k]$ and $x[k-1]$, which are $((M-1)/M)x[k]$ and $((M-2)/M)x[k-1]$, respectively. Thus, with $M = 3$, the interpolated value is

$$\frac{2}{3}x[k] + \frac{1}{3}x[k-1]$$

Since only two samples are used in the interpolation process, linear interpolation is very fast.

Example 3.3. As an illustration of upsampling and interpolation we consider interpolating the samples of a sinewave. The basic samples are illustrated in the top segment of Figure 3.14 as $x[k]$. Upsampling with $M = 6$ yields the sample values $x_u[k]$. Linear interpolation with $M = 6$ gives the sequence of samples $x_i[k]$. Note the delay of T_s. The MATLAB program used to generate Figure 3.14 follows:

```
% File: c3_upsampex.m
M = 6;                      % upsample factor
h = c3_lininterp(M);        % imp response of linear interpolator
t = 0:10;                   % time vector
tu = 0:60;                  % upsampled time vector
x = sin(2*pi*t/10);         % original samples
xu = c3_upsamp(x,M);        % upsampled sequence
subplot(3,1,1)
```

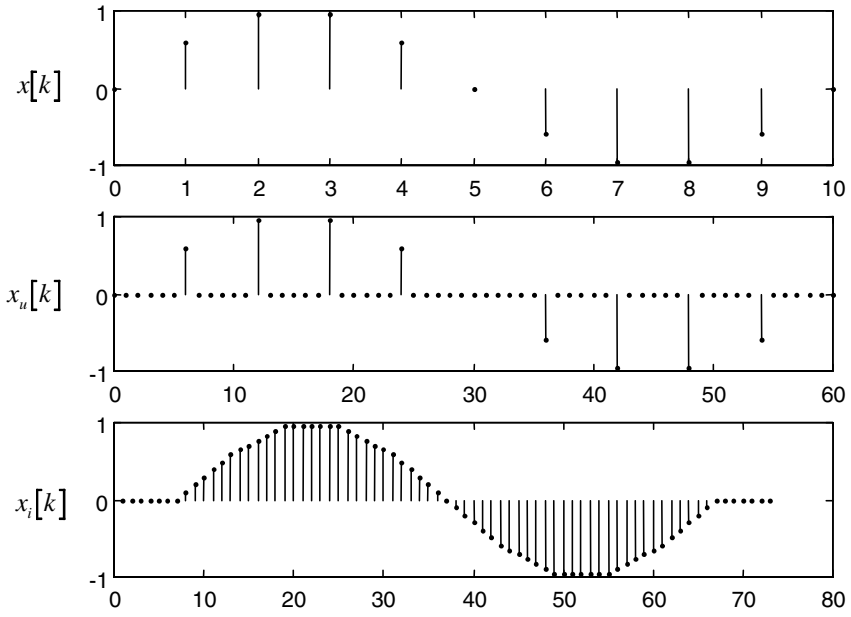


Figure 3.14 Upsampling and interpolation operations used in Example 3.3.

```
stem(t,x,'k.')
ylabel('x')
subplot(3,1,2)
stem(tu,xu,'k.')
ylabel('xu')
xi = conv(h,xu);
subplot(3,1,3)
stem(xi,'k.')
ylabel('xi')
```

■

It is clear that upsampling and downsampling involve a significant amount of overhead. If the upsampling factor M is modest, say, 2 or 3, it is usually best to develop the simulation using a single sampling frequency and therefore oversample the narrowband signals present in the system. If, however, the difference in B and W exceeds an order of magnitude, it is usually most efficient to utilize multiple sampling frequencies in the simulation and sample each signal at an appropriate sampling frequency.

Downsampling (Decimation)

Downsampling is the second operation illustrated in Figure 3.11 and is the process through which the sampling frequency is reduced. The process is accomplished by replacing a block of M samples by a single sample. Downsampling is therefore much simpler than upsampling. The functional representation for the samples at the output of a downsampler is obtained by recognizing that the downsampling process increases the sampling period by a factor of M. Thus the samples at the output of a downsampler, denoted $x_d(kT_d)$, are given by $x_d(kT_d) = x(kMT_s)$. The sample values are given by

$$x_d[k] = x[kM] \tag{3.54}$$

We need to be careful, however, to ensure that the downsampled signal does not exhibit aliasing.

3.4 The Simulation Sampling Frequency

A fundamental decision that must be made in the development of a simulation is the selection of the sampling frequency. For linear systems without feedback, the necessary sampling frequency is dictated by the allowable aliasing error, which in turn is dependent on the power spectral density of the underlying pulse shape.[5] We therefore pause to consider a common model for representing baseband signals used

[5]It will be shown in later chapters that, in addition to signal bandwidth, a number of other factors affect the required sampling frequency. For example, the presence of nonlinearities result in a requirement for higher sampling frequencies. The same is often true for systems containing feedback. In addition, multipath channels place requirements on the sampling frequency so that the multipath delays can be resolved. All of these topics will be considered in detail in later chapters.

for data transmission and to develop a technique for calculating the power spectral density of the signal corresponding to the pulse shape. Since the pulse shape plays such an important role in the selection of an appropriate sampling frequency, we consider the problem in some detail.

We know from our study of sampling that the complete elimination of aliasing errors requires an infinite sampling frequency. This is clearly a situation that cannot be achieved in practice. In addition, as the sampling frequency increases, more samples must be processed for each data symbol passed through the system. This increases the time required for executing the simulation. Since aliasing errors cannot be eliminated in practice, a natural strategy is to choose a sampling frequency for the simulation that achieves an acceptable tradeoff between aliasing errors and simulation run time. Of course, a sampling frequency must be selected so that the errors due to aliasing are negligible compared to the system degradations being investigated by the simulation.

3.4.1 General Development

A common model for the transmitted signal in a digital communication system is

$$x(t) = A\sum_{k=0}^{\infty} a_k p(t - kT - \Delta) \tag{3.55}$$

where

$$\cdots, a_{-2}, a_{-1}, a_0, a_1, a_2, \cdots, a_k, \cdots$$

is a sequence of a random variables representing the data. The values of a_k are typically denoted +1 or -1 in a binary digital system, $p(t)$ is the pulse shape function, T is the symbol period (bit period for binary transmission), and Δ is a random variable uniformly distributed over the sampling period.[6] The parameter A is a scaling constant used to establish the power in the transmitted signal. By incorporating this parameter, we can scale the pulse shape function so that the peak value is unity. We assume that $E\{a_k\} = 0$ and $E\{a_k a_{k+m}\} = R_m$ represent the mean and the autocorrelation of the data sequence, respectively.

It is easily shown [2] that the autocorrelation function of the transmitted signal is given by

$$R_{XX}(\tau) = A^2 \sum_{m=-\infty}^{\infty} R_m r(\tau - mT) \tag{3.56}$$

in which

$$r(\tau) = \frac{1}{T}\int_{-\infty}^{\infty} p(t)p(t+\tau)dt \tag{3.57}$$

[6]Note that we are now using $p(t)$ for the pulse shape rather than for the sampling function as in the preceding section. The meaning of $p(t)$ will be clear from the context in which it is used.

The required sampling frequency is determined from the PSD of the transmitted signal. Applying the Weiner-Khintchine theorem to (3.56) gives

$$S_X(f) = A^2 \int_{-\infty}^{\infty} \left(\sum_{m=-\infty}^{\infty} R_m r(\tau - mT) \right) \exp\left(-j2\pi f\tau\right)\, d\tau \tag{3.58}$$

or

$$S_X(f) = A^2 \sum_{m=-\infty}^{\infty} R_m \int_{-\infty}^{\infty} r(\tau - mT) \exp\left(-j2\pi f\tau\right)\, d\tau \tag{3.59}$$

We now put this in a form more useful for computation.

The first step is to apply the change of variables $\alpha = \tau - mT$ to (3.59). This gives

$$S_X(f) = A^2 \sum_{m=-\infty}^{\infty} R_m \int_{-\infty}^{\infty} r(\alpha) \exp\left[-j2\pi f(\alpha + mT)\right]\, d\alpha \tag{3.60}$$

Denoting

$$S_r(f) = \int_{-\infty}^{\infty} r(\alpha) \exp(-j2\pi f\alpha)\, d\alpha \tag{3.61}$$

gives

$$S_X(f) = A^2 \sum_{m=-\infty}^{\infty} R_m S_r(f) \exp(-j2\pi fmT) \tag{3.62}$$

We now determine $S_r(f)$.

Fourier transforming (3.57) gives

$$S_r(f) = \int_{-\infty}^{\infty} \left(\frac{1}{T} \int_{-\infty}^{\infty} p(t)p(t+\alpha)\, dt \right) \exp(-j2\pi f\alpha)\, d\alpha \tag{3.63}$$

Applying the change of variables $\beta = t + \alpha$ allows (3.63) to be expressed in the form

$$S_r(f) = \frac{1}{T} \left(\int_{-\infty}^{\infty} p(t) \exp(j2\pi ft)\, dt \right) \left(\int_{-\infty}^{\infty} p(\beta) \exp(-j2\pi f\beta)\, d\beta \right) \tag{3.64}$$

The second term in (3.64) is the Fourier transform of the pulse shape function $p(t)$ and the first term is the complex conjugate of the first term. This gives

$$S_r(f) = \frac{|P(f)|^2}{T} = \frac{G(f)}{T} \tag{3.65}$$

where $G(f)$ is the energy spectral density of the pulse shape function $p(t)$. Substitution of (3.65) into (3.62) gives the general result

$$S_X(f) = A^2 \frac{G(f)}{T} \sum_{m=-\infty}^{\infty} R_m \exp(-j2\pi fmT) \tag{3.66}$$

In many applications the data symbols can be assumed independent. This assumption results in significant simplifications.

3.4.2 Independent Data Symbols

If the data symbols $\{a_k\}$ are independent the autocorrelation becomes

$$R_m = E\{a_k a_{k+m}\} = E\{a_k\} E\{a_{k+m}\} = a_k^2 \delta[m] \tag{3.67}$$

so that $R_m = a_k^2$ for $m = 0$ and $R_m = 0$ otherwise. The PSD of $x(t)$ as defined by (3.66) takes a very simple form for this case:

$$S_X(f) = A^2 \frac{a_k^2 G(f)}{T} \tag{3.68}$$

If the data symbols are assumed to be $a_k = \pm 1$ for all k, $a_k^2 = 1$ and

$$S_X(f) = A^2 \frac{G(f)}{T} \tag{3.69}$$

which is independent of the data.[7] For the case in which the data symbols are not independent, the underlying autocorrelation function R_m must be determined and (3.66) must be evaluated term by term. See [2] for an example.

Example 3.4. Consider the rectangular pulse shape illustrated in Figure 3.15. It follows from Figure 3.15 that

$$P(f) = \int_0^T \exp(-j2\pi ft)\, dt = \frac{1}{j2\pi f}\left[1 - \exp(-j2\pi fT)\right] \tag{3.70}$$

This can be placed in the form

$$\begin{aligned} P(f) &= \frac{1}{j2\pi f}\left[\exp(j\pi fT) - \exp(-j\pi fT)\right]\exp(-j\pi fT) \\ &= \frac{\sin(\pi fT)}{\pi f}\exp(-j\pi fT) \end{aligned} \tag{3.71}$$

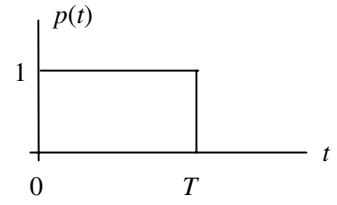


Figure 3.15 Rectangular pulse shape.

[7] We make the assumption that the data samples are +1 and −1 rather than +1 and 0 to be consistent with the assumption that $E\{a_k\} = 0$.

or, in terms of the sinc(·) function

$$P(f) = T\,\mathrm{sinc}(fT)\exp(-j\pi fT) \tag{3.72}$$

Therefore

$$G(f) = |P(f)|^2 = T^2\,\mathrm{sinc}^2(fT)$$

Substitution into (3.69) gives

$$S_X(f) = A^2T\,\mathrm{sinc}^2(fT) \tag{3.73}$$

for the power spectral density of $x(t)$.

The transmitted power is, from (3.69)

$$P = \int_{-\infty}^{\infty} S_X(f)\,df = A^2\frac{1}{T}\int_{-\infty}^{\infty} G(f)\,df \tag{3.74}$$

From Parseval's theorem and Figure 3.15 we know that

$$\int_{-\infty}^{\infty} G(f)df = \int_{-\infty}^{\infty} |P(f)|^2\,df = \int_{-\infty}^{\infty} p^2(t)\,dt = T \tag{3.75}$$

Substitution into (3.74) gives

$$P = A^2 \tag{3.76}$$

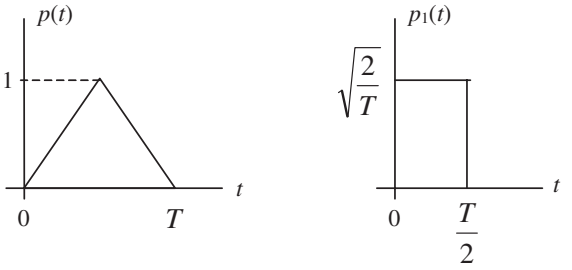
as expected. This simple result arises from the fact that $p(t)$ is a unit amplitude pulse. Thus $\sum a_k p(t - kT - \Delta)$ has unit power. Multiplication by A as shown in (3.55) simply scales the power by A^2. For other pulse shapes the relationship between power and A must be computed using the technique just illustrated. Remember also the assumed data sequence $\{a_k\}$ is a unit power (variance) process. ■

Example 3.5. An interesting pulse shape, which will be needed later, is illustrated in Figure 3.16(a). The basic pulse shape $p(t)$ can be expressed $p_1(t) \circledast p_1(t)$ where $\circledast$ denotes convolution and $p_1(t)$ is illustrated in Figure 3.16(b). Taking the Fourier transform of $p_1(t)$ gives

$$P_1(f) = \sqrt{\frac{T}{2}}\,\mathrm{sinc}\left(\frac{T}{2}f\right)\exp(-j\pi fT/2) \tag{3.77}$$

Since convolution in the time domain is equivalent to multiplication in the frequency domain we have

$$|P(f)| = |P_1(f)|^2 = \frac{T}{2}\,\mathrm{sinc}^2\left(\frac{T}{2}f\right) \tag{3.78}$$



(a) Triangular pulse shape (b) Basic shape for convolution

Figure 3.16 Triangular pulse shape.

Thus

$$G(f) = |P(f)|^2 = \frac{T^2}{4}\,\mathrm{sinc}^4\left(\frac{T}{2}f\right) \tag{3.79}$$

Substitution into (3.69) gives

$$S_X(f) = \frac{A^2T}{4}\,\mathrm{sinc}^4\left(\frac{T}{2}f\right) \tag{3.80}$$

This result will be used later in this chapter and in the problems. ■

3.4.3 Simulation Sampling Frequency

We now return to the problem of relating the simulation sampling frequency to a given pulse shape. This is accomplished by considering the signal-to-noise ratio of the sampling process where the noise power arises from aliasing. The goal is to select a sampling frequency so that the errors due to aliasing are negligible compared to the system degradations being investigated by the simulation. It will be shown that the required sampling frequency is dependent upon the waveshapes present in the simulation model.

Consider a waveform defined by (3.55), having the rectangular pulse shape illustrated in Figure 3.15, to be sampled as illustrated in Figure 3.17. In drawing

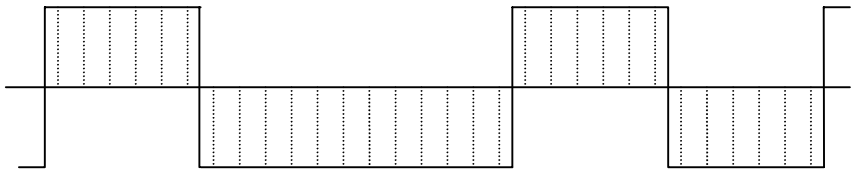


Figure 3.17 Sequence of rectangular pulses sampled at six samples per symbol.

Figure 3.17 a sampling frequency of six times the symbol frequency was assumed. The power spectral density of the data sequence is given by (3.73). Combining this result with (3.18) gives

$$S_{X_s}(f) = f_s^2 \sum_{n=-\infty}^{\infty} A^2 T \operatorname{sinc}^2[(f - nf_s)T] \tag{3.81}$$

Sampling the data sequence at m samples per symbol ($f_s = m/T$) gives

$$S_{X_s}(f) = f_s^2 \sum_{n=-\infty}^{\infty} A^2 T \operatorname{sinc}^2 \left[(f - \frac{nm}{T})T\right] \tag{3.82}$$

which is

$$S_{X_s}(f) = f_s^2 \sum_{n=-\infty}^{\infty} A^2 T \operatorname{sinc}^2 (fT - nm) \tag{3.83}$$

The next task is to compute the signal-to-noise ratio due to aliasing.

The signal-to-noise ratio due to aliasing can be expressed $(SNR)_a = S/N_a$ where the signal power is

$$S = \int_{-f_s/2}^{f_s/2} f_s^2 A^2 T \operatorname{sinc}^2(fT)\, df = 2 f_s^2 A^2 T \int_0^{f_s/2} \operatorname{sinc}^2(fT)\, df \tag{3.84}$$

and the noise power due to aliasing is

$$\begin{aligned} N_a &= \int_{-f_s/2}^{f_s/2} f_s^2 \sum_{\substack{n=-\infty \\ n\neq 0}}^{\infty} A^2 T \operatorname{sinc}^2 (fT - nm)\ df \\ &= 2 f_s^2 A^2 T \sum_{\substack{n=-\infty \\ n\neq 0}}^{\infty} \int_0^{f_s/2} \operatorname{sinc}^2 (fT - nm)\ df \end{aligned} \tag{3.85}$$

Note that we have made use of the fact that PSD is an even function of frequency. The signal power is determined by integrating the $n = 0$ term in (3.83) over the simulation bandwidth $|f| < f_s/2$. The noise power due to aliasing is the power from all of the frequency translated terms ($n \neq 0$) that fall in the simulation bandwidth. Thus, the noise due to aliasing is found by integrating over all terms in (3.83) with the $n = 0$ term excepted. This is made clear by Figure 3.18, which is drawn (not to scale) for $m = 6$. Figure 3.18 illustrates the positive frequency portion of the $n = 0$ term of (3.83) in the range $0 < f < f_s$. The translated spectra for $n = \pm 1$ and $n = 2$ are also shown.

The next step in the determination of $(SNR)_a$ is to show that both S and N_a can be determined using only the $n = 0$ term in (3.83). The lobes of the sinc$(\cdot)$ function, each having width $1/T$, are illustrated and numbered in Figure 3.18. Note that for $m = 6$ and $n = 0$ lobes 1, 2, and 3 fall in the range $0 < f < f_s/2$ and

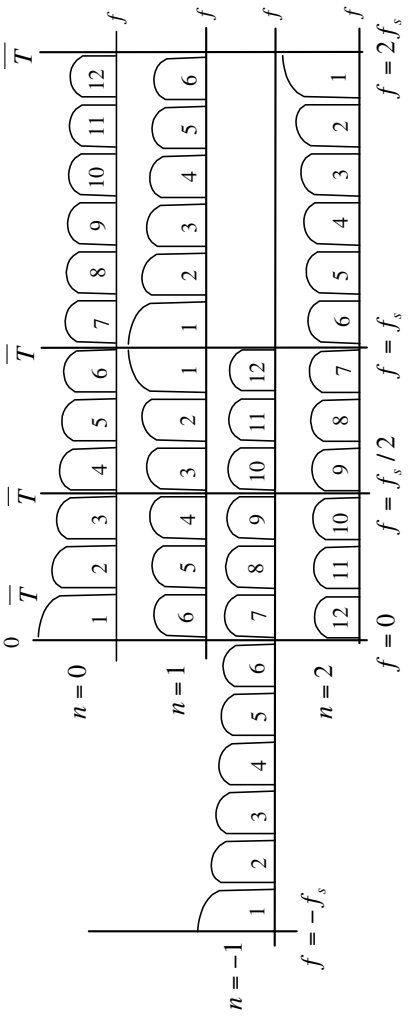


Figure 3.18 Spectra and translated spectra for $n = 0$, $n = \pm 1$, and $n = 2$.

therefore represent signal power. Lobes 4, 5, and 6 fall into the range $0 < f < f_s/2$ for the $n = 1$ term and therefore represents aliasing noise. In a similar manner lobes 7, 8, and 9 result from the $n = -1$ term in (3.83) and lobes 10, 11, and 12 result from the $n = 2$ term in (3.83). Continuing this line of thought shows that

$$\sum_{\substack{n=-\infty \\ n\neq 0}}^{\infty} \int_0^{f_s/2} \text{sinc}^2\left(fT - nm\right)\, df = \int_{f_s/2}^{\infty} \text{sinc}^2\left(fT - nm\right)\, df \tag{3.86}$$

Therefore

$$(SNR)_a = \frac{S}{N_a} = \frac{\int_0^{f_s/2} \text{sinc}^2(fT)\, df}{\int_{f_s/2}^{\infty} \text{sinc}^2(fT)\, df} \tag{3.87}$$

As can be seen by comparing Examples 3.4 and 3.5, the form of the integrand will be different depending on the pulse shape.

We will frequently find it necessary to use numerical integration in order to evaluate $(SNR)_a$. In order to accomplish this a second sampling operation is introduced in which the continuous frequency variable f is sampled at points $f = jf_1$. For accuracy we clearly require $f_1 \ll 1/T$ so that many samples are taken in each lobe of the sinc$(\cdot)$ function. Frequency sampling in this way allows the integrals in (3.87) to be replaced by sums. In order to satisfy $f_1 \ll 1/T$ let $f_1 = 1/(kT)$ where k is large so that the error induced by the numerical integration is small. With $f = jf_1$ and $f_1 = 1/(kT)$ we have

$$fT = \frac{j}{k} \tag{3.88}$$

The next step is to compute the folding frequency $f_s/2$ in terms of the discrete parameters k and m. From Figure 3.18 we see that, for m samples per symbol, the folding frequency $f_s/2$ is $m/(2T)$. Since k samples are taken for every frequency interval of width $1/T$, the folding frequency corresponds to the index $km/2$. Using (3.88) and the fact that $f_s/2$ corresponds to $km/2$ in (3.87) gives

$$(SNR)_a \cong \frac{\sum_{j=0}^{km/2} \text{sinc}^2(j/k)}{\sum_{j=km/2}^{\infty} \text{sinc}^2(j/k)} \tag{3.89}$$

The preceding is an approximation because numerical integration is used to approximate the true value of the integral.

The MATLAB program to evaluate (3.89) follows.

```
% File: c3_sna.m
k = 50;                              % samples per lobe
nsamp = 50000;                       % total frequency samples
snrdb = zeros(1,17);                 % initialize memory
x = 4:20;                            % vector for plotting
for m = 4:20                         % iterate samples per symbol
```

```
    signal = 0; noise = 0;          % initialize sum values
    f_fold = k*m/2;                 % folding frequency
    for j = 1:f_fold
        term = (sin(pi*j/k)/(pi*j/k))^2;
        signal = signal+term;
    end
    for j = (f_fold+1):nsamp
        term = (sin(pi*j/k)/(pi*j/k))^2;
        noise = noise+term;
    end
    snrdb(m-3) = 10*log10(signal/noise);
end
plot(x,snrdb)                       % plot results}
xlabel('Samples per symbol')}
ylabel('Signal-to-aliasing noise ratio')}
% End script file.
```

Note that 50 frequency samples are taken in each lobe of the sinc(·) function and that a total of 50,000 frequency samples are taken. Thus, the summation in the denominator of (3.89) spans 1,000 lobes of the sinc(·) function, after which the PSD is assumed negligible. This assumption may be verified by experimenting with the parameter `nsamp`.

Executing the preceding program yields the result illustrated in Figure 3.19. Note that $(SNR)_a$ is slightly less than 17 dB for $m = 10$ samples per symbol and that $(SNR)_a$ continues to increase as m increases. However, the impact on $(SNR)_a$ decreases for increasing m. Also note that the PSD of the sampled signal decreases as $1/f^2$ for a rectangular pulse shape. Example 3.5 shows that the PSD of the sampled signal decreases as $1/f^4$ for a triangular pulse shape. Thus, for a given value of m, the value of $(SNR)_a$ will be greater for the triangular pulse shape than for the rectangular pulse shape. The rectangular pulse shape represents is a worst-case situation. Other pulse shapes are considered in the Problems.

In a practical communications system the pulse shape $p(t)$ is chosen to give a required bandwidth efficiency [2]. High bandwidth efficiency implies that the spectrum of $x(t)$ as defined by (3.55) is compact about $f = 0$.[8] Thus, signals that exhibit high bandwidth efficiency require a smaller value of m for a given $(SNR)_a$.

3.5 Summary

The purpose of this chapter was to cover a number of topics related to sampling and the representation of sample values in communication system simulations. Two fundamental sampling theorems were considered the lowpass sampling theorem and the bandpass sampling theorem. Since bandpass signals are usually represented by lowpass signals in system simulations, the lowpass sampling theorem is the most

[8]The reference is $f = 0$ rather than $f = f_c$, in which f_c is a nonzero carrier frequency, since (3.55) represents a lowpass model of a bandpass process.

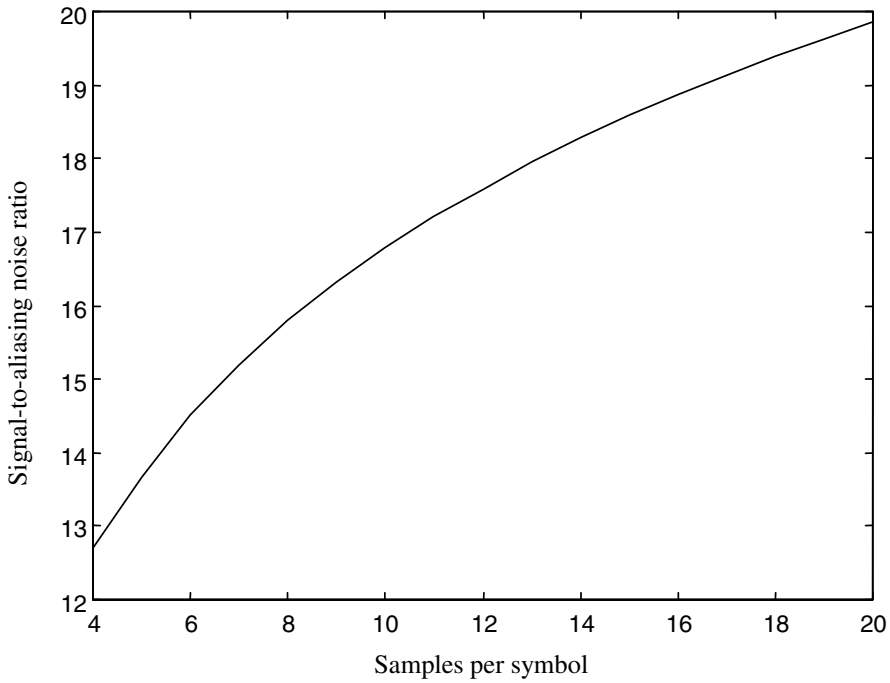


Figure 3.19 Signal-to-aliasing-noise ratio for the rectangular pulse shape.

important of these to theorems for our application. We saw that a bandlimited lowpass signal may be sampled and that the underlying bandpass signal may be reconstructed from the sample values if the sampling frequency exceeds twice the bandwidth of the bandlimited lowpass signal. The bandpass sampling theorem, although less useful in the simulation context than the lowpass sampling theorem, gave a somewhat similar result. Bandpass signals could be sampled and reconstructed if the sampling frequency is between $2B$ and $4B$ where B is the bandwidth of the bandpass signal being sampled.

Next quantizing was considered. Quantizing errors are present in all simulations, since sample values must be represented by digital words of finite length. Two types of quantizing errors were considered; errors resulting from fixed-point number representations and errors resulting from floating-point number representations. When fixed-point number representations are used, the signal-to-quantizing-noise ratio increases 6 dB for each bit added to the word length. When simulations are preformed on general-purpose computers, which use floating-point number representations, the noise resulting from quantizing errors is usually negligible. This noise, however, is never zero and there are situations in which errors can accumulate and significantly degrade the accuracy of the simulation result. The simulation user must therefore be aware of this potential error source.

The third section of this chapter treated reconstruction and interpolation. We saw that if a lowpass bandlimited signal is sampled with a sampling frequency exceeding twice the signal bandwidth, the underlying continuous-time signal can be reconstructed without error by weighting each sample with a $\sin(x)/x$ waveform, which is equivalent to passing the samples through an ideal lowpass filter. The result is a waveform defined for all values of time and, by extracting "new" samples between the original samples, interpolated samples can be generated. This operation, known as upsampling, increases the effective sampling frequency. The inverse operation, downsampling, can be accomplished by extracting every M^{th} sample from the original set of samples. Using the operations of upsampling and downsampling, one can develop a simulation in which multiple sampling frequencies are present. This is useful when the system being simulated contains signals having widely differing bandwidths. A spread-spectrum communications system is an example of such a system.

The final topic treated in this chapter was the important problem of relating the sampling frequency to the pulse shape used for waveform transmission. The pulse shape was assumed time limited and therefore cannot be bandlimited. Therefore aliasing errors occur. The criterion used for selecting the sampling frequency was to determine the required signal-to-noise ratio, where aliasing error constituted the noise source. A general method was developed for determining the PSD of the modulated signal and numerical integration of this PSD determined the signal-to-aliasing-noise ratio.

3.6 Further Reading

Most textbooks on basic communication theory consider several of the topics presented in this chapter. Included are the sampling theorem and models for transmitted signals using various pulse shape functions. Examples are:

R. E. Ziemer and W. H. Tranter, *Principles of Communications: Systems, Modulation and Noise*, 5th ed., New York: Wiley, 2001.

R. E. Ziemer and R. L. Peterson, *Introduction to Digital Communication*, 2nd ed., Upper Saddle River, NJ: Prentice Hall, 2001.

The topics of quantizing, interpolation, and decimation are typically covered in textbooks on digital signal processing. Although a wide variety of books are available in this category, the following is recommended:

A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, Upper Saddle River, NJ: Prentice Hall, 1989.

The following textbook is an excellent reference on multirate signal processing and sampling rate conversion:

R. E. Crochiere and L. R. Rabiner, *Multirate Digital Signal Processing*, Upper Saddle River, NJ: Prentice Hall, 1983.

Simulation applications of the topics presented in this chapter can be found in:

M. C. Jeruchim, P. Balaban, and K. S. Shanmugan, *Simulation of Communication Systems*, 2nd ed., New York: Kluwer Academic/Plenum Publishers, 2000.

3.7 References

1. A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, Upper Saddle River, NJ: Prentice-Hall, 1989.

2. R. E. Ziemer and W. H. Tranter, *Principles of Communications: Systems, Modulation and Noise*, 5th ed., New York: Wiley, 2002.

3. The ANSI/IEEE Standard 754-1985 (available from IEEE).

3.8 Problems

3.1 A signal $x(t)$, given by

$$x(t) = 5\cos(6\pi t) + 3\sin(8\pi t)$$

is sampled using a sampling frequency f_s of 10 samples per second. Plot $X(f)$ and $X_s(f)$. Plot the output of the reconstruction filter assuming that the reconstruction filter is an ideal lowpass filter with a bandwidth of $f_s/2$. The passband gain of the reconstruction filter is $T_s = 1/f_s$.

3.2 Repeat the preceding problem using a sampling frequency of 7 samples per second.

3.3 Develop a MATLAB program that produces, and therefore verifies, Figure 3.5.

3.4 A bandpass signal has a center frequency of 15 MHz and a bandwidth of 750 kHz (375 kHz each side of the carrier).

(a) Using the bandpass sampling theorem determine the minimum sampling frequency at which the bandpass signal can be sampled and reconstructed without error.

(b) By drawing the spectrum of the sampled signal and defining all frequencies of interest, show that the signal can be reconstructed without error if the sampling frequency found in (a) is used.

(c) Starting with the sampling frequency found in (a) consider the effect of increasing the sampling frequency. By how much can the sampling frequency be increased without incurring aliasing errors?

3.5 Assume that a signal defined by $5\sin(10\pi t)$ is sampled and quantized using a fixed-point number representation.

(a) Determine the dynamic range of the signal.

(b) Determine the crest factor of the signal.

(c) Determine the signal-to-noise ratio $(SNR)_a$ for $b = 4, 8, 16$, and 32 bits.

3.6 Repeat the preceding problem for the signal illustrated Figure 3.20.

3.7 In evaluating the effect of a fixed-point quantizing process, the assumption was made that the error induced by the quantizing process can be represented by a uniformly distributed random value. In this problem we investigate the validity of this assumption.

(a) Use $\sin(6t)$ as a signal. Using a sampling frequency of 20 Hz, generate, using MATLAB, a vector of 10,000 samples of this waveform. Note that the signal frequency and the sampling frequency are not harmonically related. Why was this done?

(b) Develop a MATLAB model for a fixed-point quantizer that contains 16 quantizing levels ($b = 4$). Using this model quantize the sample values generated in (a). Generate a vector representing the 10,000 values of quantizing error.

(c) Compute the values of $E\{e[k]\}$ and $E\{e^2[k]\}$. Compare with the theoretical values and explain the results.

(d) Using the MATLAB function `hist`, generate a histogram of the quantizing errors. What do you conclude?

3.8 The value of `realmax` is the largest number that can be represented on a computer that adheres to the ANSI/IEEE standard for representing floating-point numbers. Anything larger results in an overflow, which in MATLAB is represented by `Inf`. Using a computer that adheres to the ANSI/IEEE standard make the following computations and answer the accompanying questions:

(a) Compute `realmax + 1`. Note that no overflow occurs. Explain this apparent contradiction.

(b) Compute `realmax + 1.0e291` and `realmax + 1.0e292`. Explain the results.

3.9 Using MATLAB, compute

$$A = 1 - 0.5 - 0.25 - 0.125 - 0.125$$

Compare the result of this calculation with the result of Example 3.2. Explain the difference.

3.10 Fill in the steps to derive (3.56) and (3.57).

3.11 Data is transmitted as modeled by (3.55) in which the pulse shape $p(t)$ is the triangular pulse illustrated in Figure 3.16(a). Develop a MATLAB program to plot the signal to aliasing noise ratio $(SNR)_a$ as the number of samples per symbol varies from 4 to 20. Compare the result with that of the rectangular pulse shape by plotting both on the same set of axes. Explain the results.

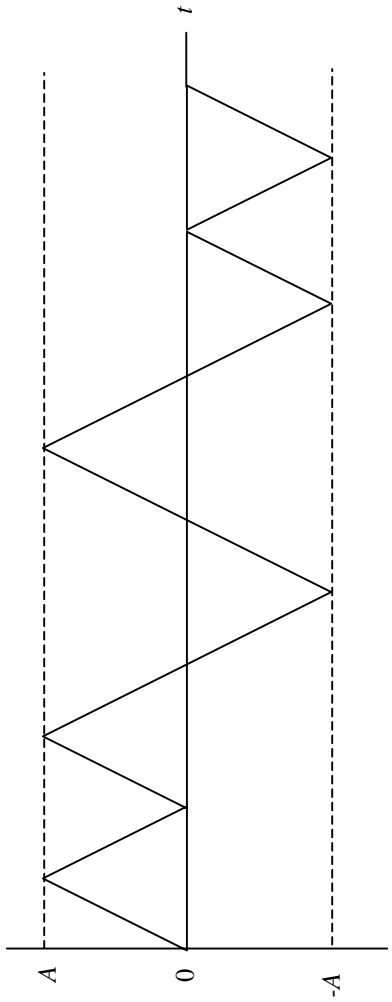


Figure 3.20 Figure for Problem 3.6.

3.12 The energy spectral density of an MSK (minimum shift keyed) signal is defined by

$$G_{MSK}(f) = \frac{16T_b \cos^2(2\pi T_b f)}{\pi^2 \left[1 - (4T_b f)^2\right]^2}$$

where T_b is the bit time [2]. Develop a MATLAB program to plot the signal to aliasing noise ratio $(SNR)_a$ as the number of samples per symbol varies from 4 to 20. Compare the result with that of the rectangular pulse shape by plotting both on the same set of axes. Explain the results.

3.13 Repeat the preceding problem for the QPSK signal for which

$$G(f) = 2T_b \operatorname{sinc}^2(2T_b f)$$

Chapter 4

LOWPASS SIMULATION MODELS FOR BANDPASS SIGNALS AND SYSTEMS

As we saw in the previous chapter, a large number of samples are required to generate an RF (radio frequency) bandpass (BP) signal, and for that reason RF bandpass signals are not typically used in waveform-level simulations. RF signals are bandpass signals in which the carrier frequency f_c usually exceeds the bandwidth B by several orders of magnitude. Using lowpass (LP) representations for bandpass signals typically results in simulations that execute much faster and have greatly reduced data storage and signal-processing requirements. The use of lowpass models for both signals and systems for use in simulation is the focus of this chapter.

4.1 The Lowpass Complex Envelope for Bandpass Signals

We now look at the lowpass complex envelope representation of bandpass signals in detail. The representation is viewed in both the time domain and in the frequency domain. Both deterministic and random signals are considered.

4.1.1 The Complex Envelope: The Time-Domain View

A general bandpass signal, such as would be found at the output of a modulator, can always be written in the form[1]

$$x(t) = A(t)\cos\left[2\pi f_0 t + \phi(t)\right] \tag{4.1}$$

where $A(t)$ is the amplitude, or *real envelope*, of the signal and $\phi(t)$ is the *phase deviation* from the phase $2\pi f_0 t$. For the case in which (4.1) represents the output of a modulator, f_0 is the carrier frequency and $2\pi f_0 t$ is the instantaneous phase of the unmodulated carrier. It follows from Euler's formula[2] that (4.1) can be written

$$x(t) = \mathrm{Re}\left\{A(t)\exp\left[j\phi(t)\right]\exp\left(j2\pi f_0 t\right)\right\} \tag{4.2}$$

which is

$$x(t) = \mathrm{Re}\left\{\widetilde{x}(t)\exp\left(j2\pi f_0 t\right)\right\} \tag{4.3}$$

The quantity

$$\widetilde{x}(t) = A(t)\exp\left[j\phi(t)\right] \tag{4.4}$$

is known as the *complex envelope* of the real signal $x(t)$. Equation (4.4) is the polar, or exponential, form of the complex envelope.

It is often convenient to express the complex envelope in rectangular form, which is

$$\widetilde{x}(t) = x_d(t) + jx_q(t) \tag{4.5}$$

The real and imaginary parts of the complex envelope are referred to as the *direct component* and *quadrature component of* $x(t)$, respectively. Applying Euler's formula to (4.4) gives

$$\widetilde{x}(t) = A(t)\cos\phi(t) + jA(t)\sin\phi(t) \tag{4.6}$$

Thus:

$$x_d(t) = A(t)\cos\phi(t) \tag{4.7}$$

and

$$x_q(t) = A(t)\sin\phi(t) \tag{4.8}$$

In applications of practical interest $A(t)$ and $\phi(t)$ are lowpass functions with bandwidths much less than f_0. Thus, $x_d(t)$ and $x_q(t)$ are also lowpass signals. It follows

[1] Throughout this chapter we will usually write mathematical expressions in terms of continuous time t in order to simplify notation. Keep in mind, however, that digital simulations actually process sample values defined by sampling at times $t = nT$, where T is the sampling period.

[2] $\exp(j\theta) = \cos\theta + j\sin\theta$.

from the definition of complex numbers that $A(t)$ and $\phi(t)$ are related to $x_d(t)$ and $x_q(t)$ by

$$A(t) = |\widetilde{x}(t)| = \sqrt{x_d^2(t) + x_q^2(t)} \tag{4.9}$$

and

$$\phi(t) = \arctan \frac{x_q(t)}{x_d(t)} \tag{4.10}$$

Using (4.3) and (4.5), the time-domain signal $x(t)$ can be written

$$x(t) = \mathrm{Re}\{[x_d(t) + jx_q(t)][\cos 2\pi f_0 t + j \sin 2\pi f_0 t]\} \tag{4.11}$$

which is

$$x(t) = x_d(t)\cos(2\pi f_0 t) - x_q(t)\sin(2\pi f_0 t) \tag{4.12}$$

Note that (4.12) could have been written by applying the trigonometric identity

$$\cos(a+b) = \cos(a)\cos(b) - \sin(a)\sin(b) \tag{4.13}$$

to (4.1) and defining $x_d(t)$ and $x_q(t)$ using (4.7) and (4.8).

Although f_0 is typically chosen as the center frequency of the bandpass signal, f_0 is arbitrary and can be chosen for convenience. However, as will be illustrated in Example 4.1, $x_d(t)$ and $x_q(t)$ are dependent upon the selection of f_0. Example 4.2 illustrates the lowpass representation of an analog FM signal. Examples 4.3, 4.4 and 4.5 illustrate the application to digital signals. These last three examples illustrate the development of simulation models for digital modulators.

Example 4.1. Consider the bandpass signal

$$x(t) = A \sin(2\pi f_m t) \cos(2\pi f_c t + \phi) \tag{4.14}$$

where f_c is the carrier frequency and ϕ is the carrier phase deviation. We assume that $f_c \gg f_m$ and desire $x_d(t)$ and $x_q(t)$ as defined by (4.3) and (4.5). The first step is to choose the frequency f_0 defined in (4.3). In order not to assume that $f_0 = f_c$, let $f_c = f_0 + f_\Delta$. This gives

$$x(t) = A \sin(2\pi f_m t) \cos(2\pi f_0 t + 2\pi f_\Delta t + \phi) \tag{4.15}$$

which can be written

$$x(t) = \mathrm{Re}\{A \sin(2\pi f_m t) \exp(j2\pi f_\Delta t) \exp(j\phi) \exp(j2\pi f_0 t)\} \tag{4.16}$$

By inspection, the complex envelope is

$$\widetilde{x}(t) = A \sin(2\pi f_m t) \exp[j(2\pi f_\Delta t + \phi)] \tag{4.17}$$

Thus:

$$\widetilde{x}(t) = A\sin(2\pi f_m t)\left[\cos(2\pi f_\Delta t + \phi) + j\sin(2\pi f_\Delta t + \phi)\right] \tag{4.18}$$

from which

$$x_d(t) = A\sin(2\pi f_m t)\cos(2\pi f_\Delta t + \phi) \tag{4.19}$$

and

$$x_q(t) = A\sin(2\pi f_m t)\sin(2\pi f_\Delta t + \phi) \tag{4.20}$$

follow directly by equating real and imaginary parts. Note that both $x_d(t)$ and $x_q(t)$ depend upon the relationship between f_c and f_0. The simplest expressions result if $f_0 = f_c$ ($f_\Delta = 0$) but, as previously mentioned, the choice of f_0 is arbitrary. In simulation problems we typically choose f_0 so that the computational burden is minimized. ■

Example 4.2. An analog FM modulator is defined by the expression

$$x_c(t) = A_c\cos\left(2\pi f_c t + k_f\int_{t_0}^{t} m(\alpha)\,d\alpha + \phi(t_0)\right) \tag{4.21}$$

where A_c and f_c represent the amplitude and frequency of the unmodulated carrier, respectively, $m(t)$ is the message or information carrying signal, k_f is the modulation index, t_0 is an arbitrary reference time, and $\phi(t_0)$ is the phase deviation at time t_0. Assuming that the time reference $t_0 = 0$ is selected and that $\phi(t_0) = 0$, it follows that $x_c(t)$ can be represented

$$x_c(t) = \mathrm{Re}\left\{A_c\exp\left(jk_f\int_0^t m(\alpha)\,d\alpha\right)\exp(j2\pi f_c t)\right\} \tag{4.22}$$

Thus, the complex envelope is

$$\widetilde{x}(t) = A_c\exp\left(jk_f\int_0^t m(\alpha)\,d\alpha\right) \tag{4.23}$$

from which

$$x_d(t) = A_c\cos\left(k_f\int_0^t m(\alpha)\,d\alpha\right) \tag{4.24}$$

and

$$x_q(t) = A_c\sin\left(k_f\int_0^t m(\alpha)\,d\alpha\right) \tag{4.25}$$

Thus, in order to represent $x(t)$ in a simulation, it is necessary only to generate the two lowpass signals given by (4.24) and (4.25).

A suite of models for an FM modulator are illustrated in Figure 4.1. Figure 4.1(a) shows the continuous-time bandpass model. The continuous-time lowpass model, in which the output is the lowpass complex envelope representation of $x_c(t)$, is shown in Figure 4.1(b). The discrete-time equivalent is illustrated in Figure 4.1(c), in which the sampling period is T and n indexes the samples. Note that, in the discrete-time model, the integrator is modeled as an accumulator (summation operator). This is equivalent to rectangular integration in which the area accumulated in the k^{th} time slot is $Tm(kT)$. Other integrator models can be used, as will be discussed in later chapters. ■

We now consider a few examples involving digital modulation. In order to do this a simulation model for the modulator is needed. The basic model is illustrated in Figure 4.2. [Note: Figure 4.2 illustrates a bandpass model for which the output is the bandpass signal $x_{ck}(t)$. The lowpass model used for simulation has the outputs $x_{dk}(t)$ and $x_{qk}(t)$, which define the lowpass complex envelope of $x_{ck}(t)$.] We assume that a_k represents a binary data stream. In addition, M-ary signaling, in which b information bits are grouped together to form a data symbol so that the transmitted signal in the k^{th} signaling interval can carry more than 1 bit of information. Typically $M = 2^b$, but this is not a necessary assumption. The binary to M-ary symbol mapping shown in Figure 4.2 performs the function of grouping together b bits to form the M-ary symbol. The output of the mapper are the direct and quadrature components of the k-symbol. These are denoted d_k and q_k. (The k^{th} symbol itself can be viewed as complex valued, $s_k = d_k + jq_k$.)

The symbols d_k and q_k can be considered impulse functions having weights determined by the bit-to-symbol mapping. The impulse response of the pulse-shaping filter is denoted $p(t)$ so that the direct and quadrature signals for the k^{th} signaling interval, $kT < t < (k+1)T$, are $x_{dk}(t)$ and $x_{qk}(t)$ as shown in Figure 4.2. The transmitter output for the k^{th} signaling interval is

$$x_{ck}(t) = x_{dk}(t)\cos 2\pi f_0 t - x_{qk}(t)\sin 2\pi f_0 t \tag{4.26}$$

The corresponding discrete-time signal model is

$$x_{ck}(nT) = x_{dk}(nT)\cos 2\pi f_0 nT - x_{qk}(nT)\sin 2\pi f_0 nT \tag{4.27}$$

which is simply the sampled version of the continuous-time signal model.

Before leaving this topic, we pause to review a few terms, to briefly discuss scattergrams, and to point out the difference between scattergrams, as used in the simulation context, and signal constellations, which are familiar to us from our study of digital communication theory. A signal space is defined as a K-dimensional space generated by K orthonormal basis functions $\phi_i(t)$, $i = 1, 2, \cdots, K$ and signals are represented as vectors in this space. For example, assume an M-ary communications system in which one of m signals is transmitted in the k^{th} signaling interval. In terms of the basis functions, the signal transmitted in the k^{th} signaling interval is expressed

$$x_c(t) = \sum_{i=1}^{K} x_{im}\phi_i(t), \quad kT < t < (k+1)T, \quad m = 1, 2, \cdots, M \tag{4.28}$$

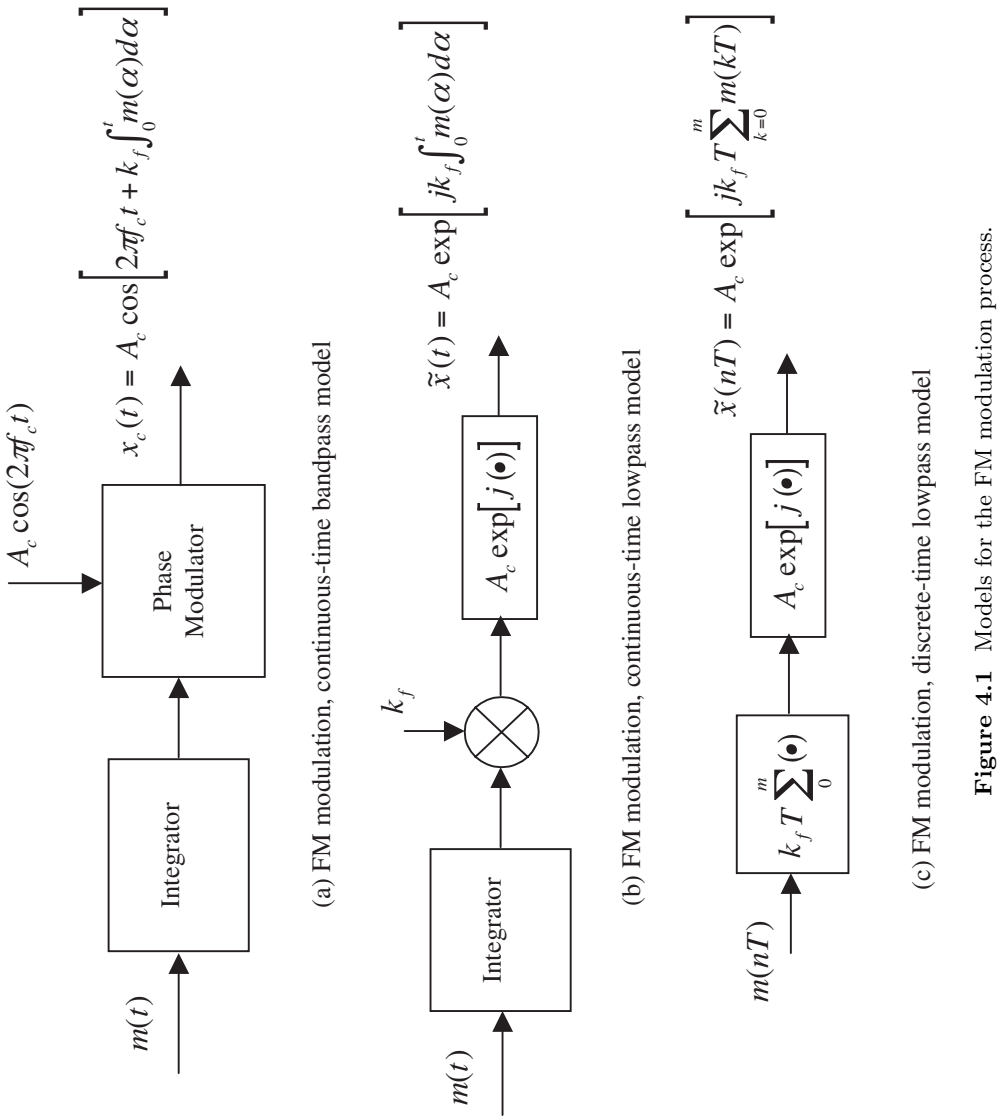


Figure 4.1 Models for the FM modulation process.

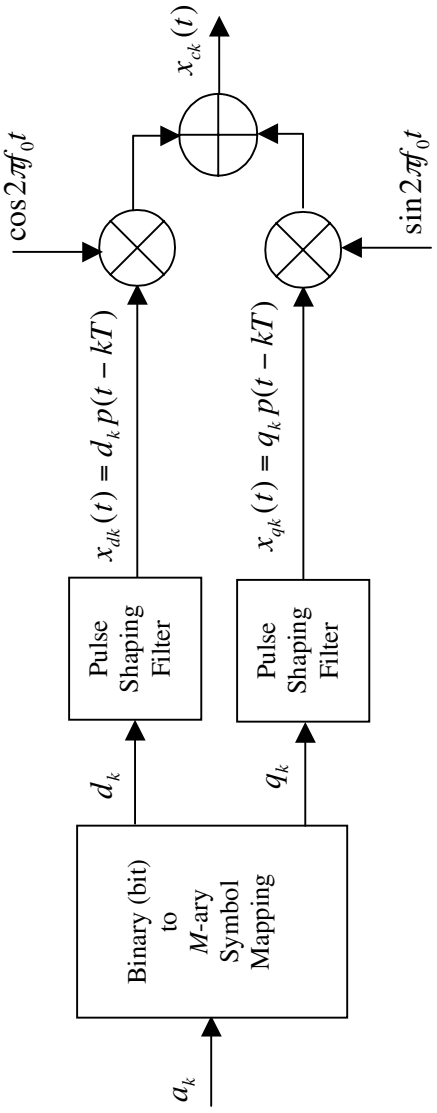


Figure 4.2 Simulation model for digital modulator (bandpass model).

In signal space the m^{th} signal is represented by the vector

$$\mathbf{X}_m = [x_{1m} \quad x_{2m} \quad \cdots \quad x_{Km}] \tag{4.29}$$

Signal space is scaled so that $|\mathbf{X}_m|^2$ represents the energy associated with the m^{th} signal in the set of possible transmitter outputs. The K-dimensional plot of the points $\mathbf{X}_m$, $m = 1, 2, \cdots, M$ defines the signal constellation.

The scattergram corresponding to a signal is a plot of $x_q(t)$ versus $x_q(t)$ and is therefore defined in terms of the lowpass complex envelope of a real bandpass signal. The dimensionality of the scattergram is one if $x_d(t) = 0$ or $x_q(t) = 0$. Otherwise, the dimensionality of the scattergram is two. Examples 4.3 and 4.4 illustrate that the scattergram and the signal constellation are closely related when $K = 2$, which is the case for QPSK and QAM. We will see in Example 4.5 that this relationship is lost when the dimensionality of the signal space exceeds two.

Example 4.3. (QPSK) In order to form a QPSK signal, the data symbols a_k are formed by taking binary symbols two at a time. Thus each data symbol consists of one of the binary pairs 00, 01, 10, or 11. The bandpass QPSK signal for the k^{th} signaling interval has the form

$$x_{ck}(t) = A_c \cos(2\pi f_0 t + \phi_k), \quad kT < t < (k+1)T \tag{4.30}$$

where ϕ_k takes on one of the four values $+\pi/4$, $-\pi/4$, $+3\pi/4$, or $-3\pi/4$. The complex envelope corresponding to $x_{ck}(t)$ can be written

$$\widetilde{x}_{ck}(t) = \text{Re}\{A_c \exp(j\phi_k)\} \tag{4.31}$$

from which the direct and quadrature components are

$$x_{dk}(t) = A_c \cos\phi_k \tag{4.32}$$

and

$$x_{qk}(t) = A_c \sin\phi_k \tag{4.33}$$

Note that if $p(t)$ is constant over a signaling interval, both $x_{dk}(t)$ and $x_{qk}(t)$ are constant over a signaling interval. In this case both $x_{dk}(t)$ and $x_{qk}(t)$ take on only one of two values and switch instantaneously from one value to the other. If $x_{qk}(t)$ is plotted as a function of $x_{qdk}(t)$, the scattergram, illustrated in Figure 4.3, results. Note that Figure 4.3, when properly scaled, also represents the signal constellation.

Each of the points in the QPSK signal constellation correspond to a pair of binary symbols as determined by the bit-to-symbol mapping illustrated in Figure 4.2. Although the mechanism by which pairs of binary symbols are mapped to QPSK symbols is arbitrary, the mapping shown in Figure 4.3, which is known as Gray code mapping, is frequently used. In Gray code mapping nearest neighbors, in signal space, differ in only one binary symbol. This strategy is justified, since the error probability is typically an inverse monotonic function of the Euclidean

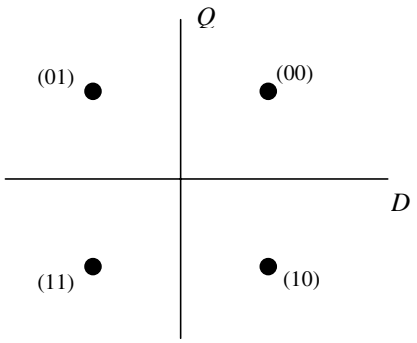


Figure 4.3 QPSK scattergram for constant $p(t)$.

distance between points in signal space.[3] For the mapping illustrated in Figure 4.3, assume that the bit sequence is b_1b_2, where b_1 is the most significant bit and b_2 is the least significant bit. If $b_2 = 0$, $d_k = 1$, and if $b_2 = 1$, $d_k = -1$ so that the least significant bit determines whether the QPSK symbol is in the left half or in the right half of the D-Q plane. In a similar fashion, if $b_1 = 0$, $q_k = 1$, and if $b_1 = 1$, $q_k = -1$ so that the most significant bit determines whether the QPSK symbol is in the upper half or in the lower half of the D-Q plane.

In order to look at QPSK scattergrams for a nonconstant $p(t)$ and the corresponding time-domain waveforms a simple MATLAB program is developed. The program is given in Appendix A and is somewhat general in that the number of levels on the D (direct) axis and the Q (quadrature) axis can be assigned independently. For QPSK both of these parameters are equal to two. The pulse-shaping filter is a sixth-order Butterworth filter.[4] The filter bandwidth used in the simula-

[3]For example, for an AWGN channel, the pairwise error probability $\Pr\{x_i \rightarrow x_j\}$, which is the probability of transmitting $x_i(t)$ and deciding (incorrectly) at the receiver that $x_j(t)$ was transmitted, $j \neq i$, is

$$\Pr\{x_i \rightarrow x_j\} = Q\left(\frac{\|Y_i - Y_j\|}{\sqrt{2N_0}}\right)$$

In this expression Y_i and Y_j are vectors, in signal space, corresponding to $y_i(t)$ and $y_j(t)$, which are $x_i(t)$ and $x_j(t)$ referenced to the receiver input and $\|Y_i - Y_j\|$ represents the Euclidean distance separating $y_i(t)$ and $y_j(t)$ [1]. Thus as signal vectors move closer together due to channel conditions or system inperfections, the error probability increases.

[4]Normally we would use a zero-ISI filter, such as a root-raised cosine filter, for these applications to ensure that the error probability does not increase due to the memory induced by the filter. However, high-order Butterworth filters approximate zero-ISI filters, since they are nearly ideal and therefore closely approximate a $\sin(x)/x$ impulse response. In addition, they are quickly implemented in MATLAB using built-in functions.

tion, denoted `bw`, is equal to the symbol rate.[5] The input to the MATLAB program follows.

```
>>c4_qamdemo
Number of D levels > 2
Number of Q levels > 2
Number of symbols > 100
Number of samples per symbol > 20
Filter bandwidth, 0<bw<1 > 0.1
```

Executing the program given in Appendix A with these parameters yields the results illustrated in Figure 4.4. The top row shows scattergrams formed by plotting $x_q(t)$ as a function of $x_d(t)$. The scattergram at the top left results for $p(t) = 1, 0 \leq t < T_{sym}$, where T_{sym} is the symbol period (twice the bit period). Note the relationship of this scattergram to the signal constellation shown in Figure 4.3. The scattergram at the top right is formed by passing this signal through a sixth-order Butterworth filter. The corresponding time-domain signals are illustrated on the bottom row of Figure 4.4, with $x_d(t)$ at the bottom left and $x_q(t)$ at the bottom right.

It is also worth noting from the simulation program given in Appendix A that the bit pattern is never explicitly formed. Rather, symbols are formed that represent pairs of bits. The receiver model demodulates symbols and, using simulation, the symbol error rate can be estimated. The mapping of symbol error rate to bit error rate is deterministic and can be accomplished analytically. ■

Example 4.4. (16-QAM) The block diagram of a QAM transmitter can also be represented as shown in Figure 4.2. The bit-to-symbol mapper maps each group of four input binary symbols a_k to a single 16-QAM symbol. The signal constellation is illustrated in Figure 4.5 along with the binary sequence corresponding to each 16-QAM symbol. As was the case with QPSK, the mapping of binary symbols to 16-QAM symbols is arbitrary but the mapping is usually defined such that a Gray code results. Note that Figure 4.5 is a Gray code mapping [2].

In QPSK each of the symbols d_k and q_k could take on one of two values, which in the previous example were defined to be $+1$ and -1. We see from Figure 4.5 that both d_k and q_k in 16-QAM can take on one of four values in each symbol period.

[5] Keep in mind that MATLAB normalizes filter bandwidths to the Nyquist frequency, f_N, which is half the sampling frequency f_s. In other words, the MATLAB parameter `bw` is f_{bw}/f_N, where f_{bw} is the filter bandwidth in Hertz. Assume that the sampling frequency is k times the symbol rate f_{sym} (recall the discussion of the simulation sampling frequency from the preceding chapter). Also assume that the ratio of the filter bandwidth to the symbol rate, f_{bw}/f_{sym} is denoted λ. Using these definitions the ratio of the filter bandwidth to the Nyquist frequency (the normalized bandwidth used in MATLAB), is

$$\frac{f_{bw}}{f_N} = \frac{2f_{bw}}{f_s} = \frac{2}{k}\frac{f_{bw}}{f_{sym}} = \frac{2\lambda}{k}$$

Thus, if the filter bandwidth is equal to the symbol rate $\lambda = 1$. Also, if the simulation sampling frequency is 20 times the symbol rate, $k = 20$. This leads to a normalized filter bandwidth of 0.1 as used in the simulation.

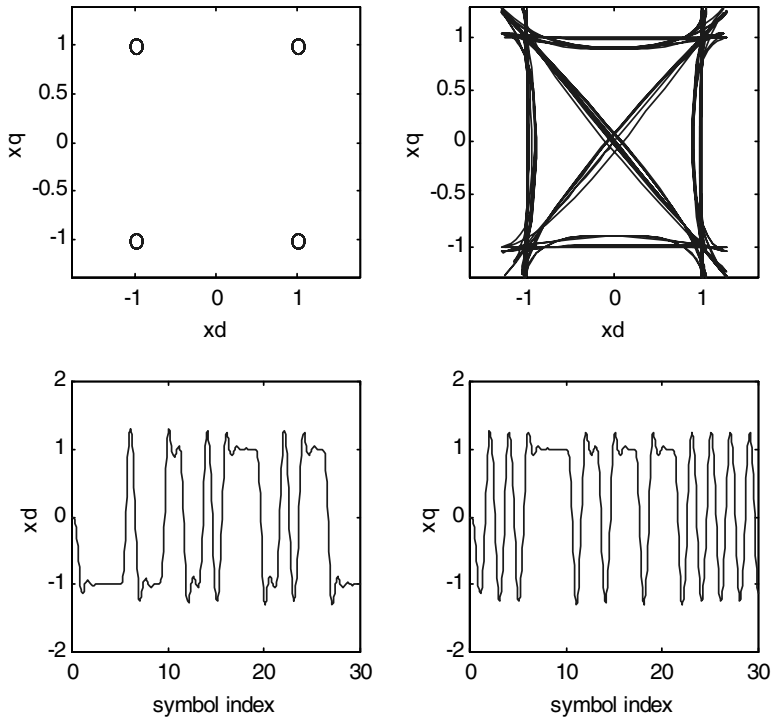


Figure 4.4 QPSK simulation results.

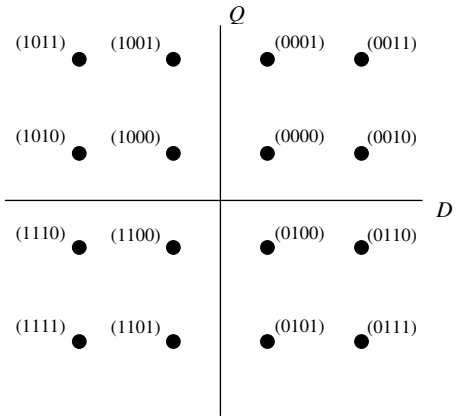


Figure 4.5 16-QAM signal constellation.

For convenience in the simulation to follow these four values are defined to be +3, +1, −1, and −3.

The same program used for the QPSK example can be used for the 16-QAM example. For 16-QAM the number of the number of levels on the D (direct) axis and the Q (quadrature) axis are both equal to four. The pulse-shaping filter is, once again, a sixth-order Butterworth filter. For this example the filter bandwidth used in the simulation is twice the symbol rate. The input to the MATLAB program follows.

```
>> c4_qamdemo
Number of D levels > 4
Number of Q levels > 4
Number of symbols > 500
Number of samples per symbol > 20
Filter bandwidth, 0<bw<1 > 0.2
```

Executing the program given in Appendix A with the preceding parameters yields the results illustrated in Figure 4.6. The top row shows scattergrams formed by plotting $x_q(t)$ as a function of $x_d(t)$. The scattergram at the top left results for $p(t) = 1$, $0 \leq t < T_{sym}$, where T_{sym} is the symbol period (four times the bit period). Note the relationship of this scattergram to the signal constellation shown in Figure 4.5. The scattergram at the top right is formed by passing this signal through a sixth-order Butterworth filter. The corresponding time-domain signals are illustrated on the bottom row of Figure 4.6, with $x_d(t)$ at the bottom left and $x_q(t)$ at the bottom right. The four (steady-state) values can clearly be seen.

As in the previous example the bit pattern is never explicitly formed in the simulation. Rather, symbols are formed that represent 16-QAM symbols The receiver model demodulates symbols and, using simulation, the symbol error rate can be estimated. The mapping of symbol error rate to bit error rate is deterministic and can be accomplished analytically. ■

Example 4.5. (4-FSK) In the two preceding examples the dimensionality of the signal space of the transmitted signals was two so that the mapping from signal space to the D-Q plane was trivial. We now consider an example in which binary symbols are grouped together two at a time as in QPSK but the dimensionality of the signal space is four. Thus, the signal-space is generated using four orthogonal basis functions rather than two basis functions as in QPSK. For this example the basis functions are chosen to be sinusoids having different frequencies (thus the name 4-FSK). Letting the pulse shaping function $p(t)$ be the constant A over the k^{th} signaling interval we have

$$s_m(t) = A\cos(2\pi f_m t), \quad kT < t < (k+1)T, \quad m = 1, 2, 3, 4 \tag{4.34}$$

Thus, in each signaling interval one of four sinusoids are transmitted. Note that each $s_m(t)$ must pass through an integer number of cycles in the T-second signaling interval to ensure that

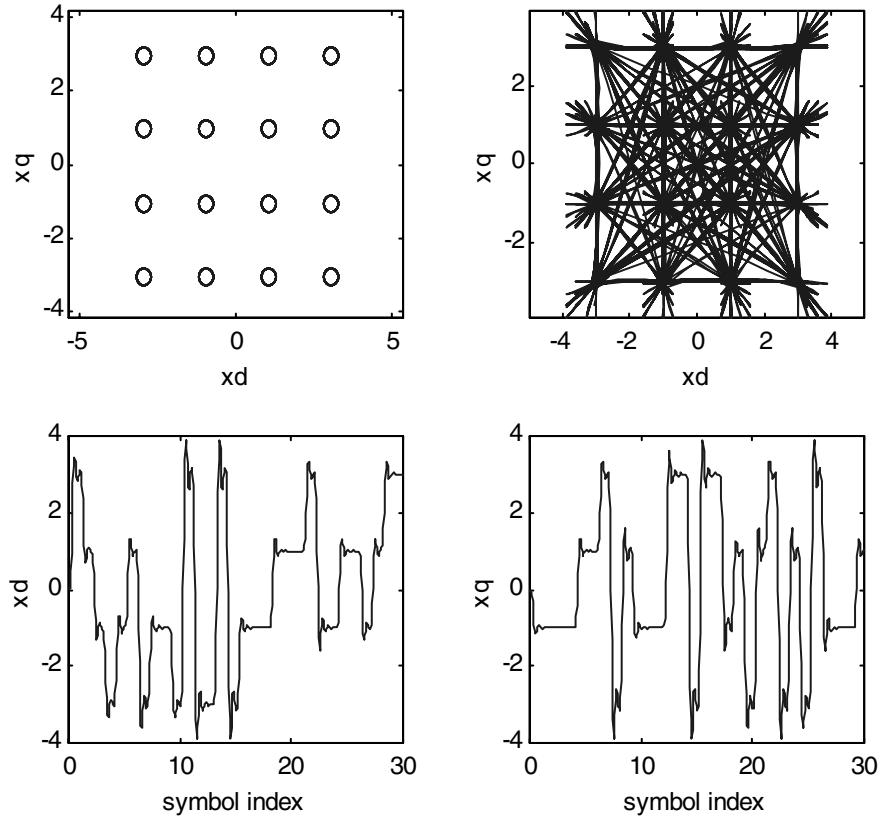


Figure 4.6 16-QAM simulation results.

$$\int_{(k-1)T}^{kT} s_i(t)s_j(t)\,dt = 0, \quad i,j = 1,2,3,4, \quad i \neq j \tag{4.35}$$

so that the signals are orthogonal.

Writing $s_m(t)$ in the form

$$s_m(t) = A\cos[2\pi(f_m - f_0)t + 2\pi f_0 t], \quad kT < t < (k+1)T, \quad m = 1,2,3,4 \tag{4.36}$$

yields

$$s_m(t) = \mathrm{Re}\left\{A\exp[j2\pi(f_m - f_0)t]\exp(j2\pi f_0 t)\right\}, \quad kT < t < (k+1)T, \quad m = 1,2,3,4 \tag{4.37}$$

Thus, the complex envelope corresponding to the m^{th} signal in the set of transmitted signals is

$$\widetilde{s}_m(t) = A\exp[j2\pi(f_m - f_0)t], \quad kT < t < (k+1)T, \quad m = 1,2,3,4 \tag{4.38}$$

from which

$$x_{dk}(t) = A\cos[2\pi(f_m - f_0)t], \quad m = 1, 2, 3, 4 \tag{4.39}$$

and

$$x_{qk}(t) = A\sin[2\pi(f_m - f_0)t], \quad m = 1, 2, 3, 4 \tag{4.40}$$

Note that, even though $p(t)$ is a constant, both $x_{dk}(t)$ and $x_{qk}(t)$ are time-varying over a signaling interval. This is in contrast to the results of the preceding two examples and is a consequence of the fact that the dimensionality of the underlying signal space K exceeds 2, the dimensionality of the D-Q plane. ■

4.1.2 The Complex Envelope: The Frequency-Domain View

From (4.3) we can write

$$x(t) = \frac{1}{2}\widetilde{x}(t)\exp(j2\pi f_0 t) + \frac{1}{2}\widetilde{x}^*(t)\exp(-j2\pi f_0 t) \tag{4.41}$$

Multiplying by $\exp(-j2\pi f_0 t)$ gives

$$2x(t)\exp(-j2\pi f_0 t) = \widetilde{x}(t) + \widetilde{x}^*(t)\exp(-j4\pi f_0 t) \tag{4.42}$$

or

$$\widetilde{x}(t) = 2x(t)\exp(-j2\pi f_0 t) - \widetilde{x}^*(t)\exp(-j4\pi f_0 t) \tag{4.43}$$

As we saw in the previous section, $\widetilde{x}(t)$ is a lowpass signal (a signal having a spectrum that is nonzero only in the neighborhood of $f = 0$). Taking the lowpass portion of (4.43) yields

$$\widetilde{x}(t) = \mathrm{Lp}\{2x(t)\exp(-j2\pi f_0 t) - \widetilde{x}^*(t)\exp(-j4\pi f_0 t)\} \tag{4.44}$$

where $\mathrm{Lp}\{\cdot\}$ denotes the lowpass portion of the argument.

The Fourier transform of (4.43) is given by

$$\widetilde{X}(f) = 2X(f + f_0) - \widetilde{X}^*(f + 2f_0) \tag{4.45}$$

As illustrated in Figure 4.7, $X(f)$ is nonzero except in the neighborhood of $f = \pm f_0$, with $X_+(f)$ representing the positive frequency portion of $X(f)$ and $X_-(f)$ representing the negative frequency portion of $X(f)$. Thus, $\widetilde{X}^*(f + 2f_0)$ is nonzero except in the neighborhood of $f = -f_0$ ($X_+(f)$ translated to the left by $2f_0$) and $f = -3f_0$ ($X_-(f)$ translated to the left by $2f_0$). This term does not contribute to $\widetilde{X}(f)$ or, equivalently, to $\widetilde{x}(t)$, since $\widetilde{X}(f)$ is nonzero only in the neighborhood of $f = 0$. The manner in which $X(f + f_0)$ contributes to $\widetilde{x}(t)$ is illustrated in Figure 4.7. Note that $X(f + f_0)$ is nonzero except in the neighborhood of $f = -2f_0$

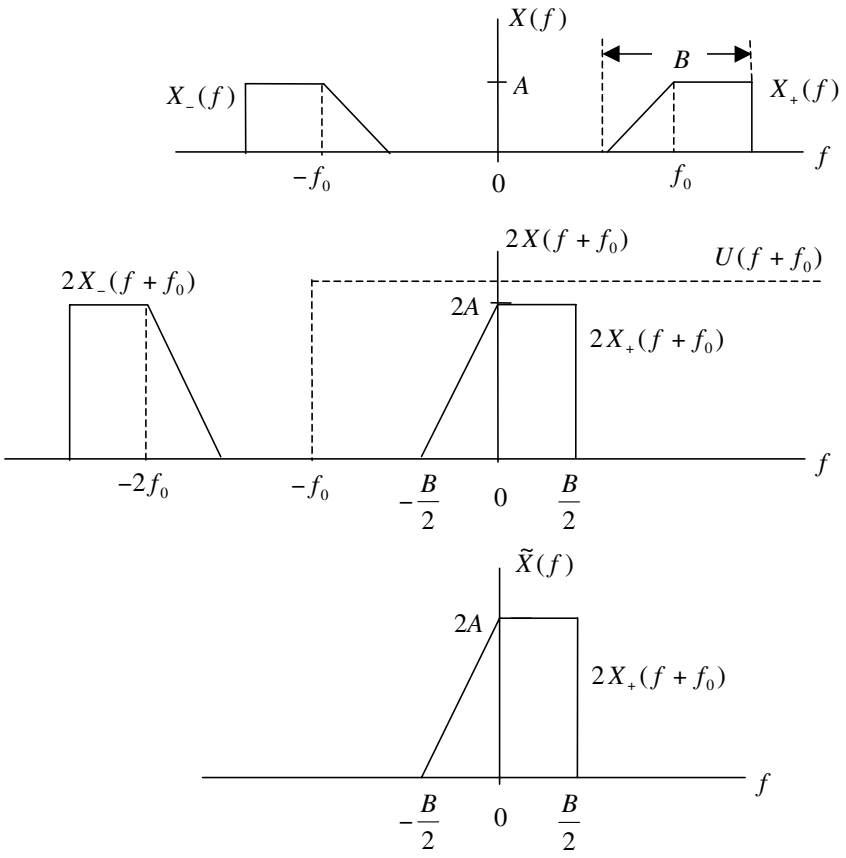


Figure 4.7 Derivation of $\widetilde{X}(f)$ from $X(f)$.

($X_-(f)$ translated to the left by f_0) and $f = 0$ ($X_+(f)$ translated to the left by f_0). It therefore follows that only $X_+(f + f_0)$ contributes to $\widetilde{x}(t)$ and

$$\widetilde{X}(f) = 2X_+(f + f_0) \tag{4.46}$$

which is equivalent to

$$\widetilde{x}(t) = 2\,\mathrm{Lp}\left\{x(t)\exp\left(-j2\pi f_0 t\right)\right\} \tag{4.47}$$

As illustrated in Figure 4.7, the lowpass filtering operation can be implemented by using a filter having the transfer function $H(f) = U(f + f_0)$. Thus, (4.46) can be expressed

$$\widetilde{X}(f) = 2X(f + f_0)U(f + f_0) \tag{4.48}$$

4.1.3 Derivation of $X_d(f)$ and $X_q(f)$ from $\widetilde{X}(f)$

Expressions for $X_d(f)$ and $X_q(f)$ are easily derived. Fourier transforming (4.5) yields

$$\widetilde{X}(f) = X_d(f) + jX_q(f) \tag{4.49}$$

Replacing f by $-f$ gives

$$\widetilde{X}(-f) = X_d(-f) + jX_q(-f) \tag{4.50}$$

Since $x_d(t)$ and $x_q(t)$ are real, $X_d(-f) = X_d^*(f)$ and $X_d(-f) = X_d^*(f)$. Thus, (4.50) can be written

$$\widetilde{X}(-f) = X_d^*(f) + jX_q^*(f) \tag{4.51}$$

Complex conjugating (4.51) gives

$$\widetilde{X}^*(-f) = X_d(f) - jX_q(f) \tag{4.52}$$

Adding (4.49) and (4.52) gives

$$X_d(f) = \frac{1}{2}\left(\widetilde{X}(f) + \widetilde{X}^*(-f)\right) \tag{4.53}$$

Multiplying (4.52) by -1 and adding the result to (4.49) gives

$$X_q(f) = \frac{1}{2j}\left(\widetilde{X}(f) - \widetilde{X}^*(-f)\right) \tag{4.54}$$

Inverse Fourier transforming $X_d(f)$ and $X_q(f)$, as defined by (4.53) and (4.54), gives $x_d(t)$ and $x_q(t)$, respectively.

The spectra $X_d(f)$ and $X_q(f)$ corresponding to the lowpass complex envelope $\widetilde{X}(f)$ derived in Figure 4.7 are illustrated in Figure 4.8. These spectra follow directly

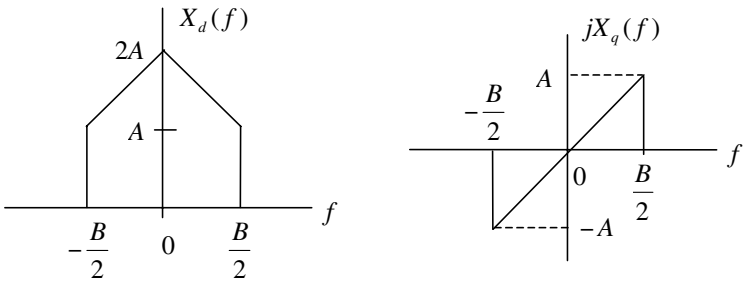


Figure 4.8 Spectrum of direct and quadrature components.

from (4.53) and (4.54). Also note that in plotting the quarature component it is more natural to plot $jX_q(f)$. Note that $X_d(f)$ is real and even and that $X_q(f)$ is imaginary and odd.

From Figure 4.7 we see that the spectrum $X(f)$ of the real bandpass signal $x(t)$ is nonsymmetric about f_0. As a result, samples of the lowpass complex envelope $\widetilde{x}(t)$ will be complex valued. Both the real and imaginary part of $\widetilde{x}(t)$, $x_d(t)$, and $x_q(t)$ will have bandwidth $B/2$, which is half the bandwidth of the real bandpass signal $x(t)$. Thus, as discussed in the previous chapter, both $x_d(t)$ and $x_q(t)$ must be sampled at a rate exceeding $2(B/2) = B$ samples per second. The result of this sampling operation will produce at least $2B$ samples per second. Conversely, if $X(f)$ has conjugate symmetry about f_0, $\widetilde{X}(f)$ will have conjugate symmetry about $f = 0$. For this case $\widetilde{x}(t)$ will be real ($x_q(t) = 0$) and sampling the quadrature component will not be required.

4.1.4 Energy and Power

As we know from linear system theory, Parseval's theorem tells us that the Fourier transform preserves energy and power. Unfortunately, however, the energy (or power) in the complex envelope is not equal to the corresponding energy (or power) in the corresponding bandpass signal. Using (4.41) gives

$$|x(t)|^2 = \frac{1}{4}\left|\widetilde{x}(t)\exp(j2\pi f_0 t) + \widetilde{x}^*(t)\exp(-j2\pi f_0 t)\right|^2 \tag{4.55}$$

for the instantaneous power in $x(t)$. The preceding expression gives

$$\begin{aligned}|x(t)|^2 = \frac{1}{4}&\left[\widetilde{x}(t)\exp(j2\pi f_0 t) + \widetilde{x}^*(t)\exp(-j2\pi f_0 t)\right]\\ &\cdot\left[\widetilde{x}^*(t)\exp(-j2\pi f_0 t) + \widetilde{x}(t)\exp(j2\pi f_0 t)\right]\end{aligned} \tag{4.56}$$

Carrying out the indicated multiplication yields

$$\begin{aligned}|x(t)|^2 = \frac{1}{4}&\left[|\widetilde{x}(t)|^2 + [\widetilde{x}(t)]^2\exp(j4\pi f_0 t)\right.\\ &\left.+ [\widetilde{x}^*(t)]^2\exp(-j4\pi f_0 t) + |\widetilde{x}(t)|^2\right]\end{aligned} \tag{4.57}$$

Since the terms $[\widetilde{x}(t)]^2\exp(j4\pi f_0 t)$ and $[\widetilde{x}^*(t)]^2\exp(-j4\pi f_0 t)$ represent bandpass signals, and therefore have zero average value, taking the expectation yields

$$E\left\{|x(t)|^2\right\} = \frac{1}{2}E\left\{|\widetilde{x}(t)|^2\right\} \tag{4.58}$$

By definition, the average power in the real bandpass signal $x(t)$ is

$$\mathcal{E}_x = E\left\{|x(t)|^2\right\} \tag{4.59}$$

This is sometimes referred to as real power. The power in the lowpass complex envelope $\widetilde{x}(t)$, which is sometimes referred to as complex power, is

$$\mathcal{E}_{\widetilde{x}} = E\left\{|\widetilde{x}(t)|^2\right\} \tag{4.60}$$

Substitution of (4.59) and (4.60) into (4.58) yields

$$\mathcal{E}_{\widetilde{x}} = 2\mathcal{E}_x \tag{4.61}$$

Thus, the power in the complex envelope of a signal is twice the power in the real bandpass signal from which it is derived. Fortunately, as we will see in a section soon to follow, a similar result holds for random signals and noise. Therefore, a number of important quantities, most notably the signal-to-noise ratio, are preserved when real bandpass signals are represented by their corresponding lowpass complex envelopes.

4.1.5 Quadrature Models for Random Bandpass Signals

The representation of signals in the frequency domain, through the use of the Fourier transform, implies that the signals are deterministic energy signals. Bandpass random signals also have a lowpass representation in terms of direct and quadrature components. The underlying mathematics for generating lowpass signal models for random bandpass signals is quite a bit different from that used in the previous two sections. We pause a minute to review the underlying theory.

Consider, for example, a narrowband random process defined by the equation

$$n(t) = n_d(t)\cos(2\pi f_0 t + \theta) - n_q(t)\sin(2\pi f_0 t + \theta) \tag{4.62}$$

where θ is an arbitrary phase angle uniformly distributed in $[-\pi, \pi)$.[6] The process in (4.62) can be written

$$n(t) = R(t)\cos[2\pi f_0 t + \phi(t) + \theta] \tag{4.63}$$

or equivalently

$$n(t) = \mathrm{Re}\left\{R(t)\exp\left[j\phi(t)\right]\exp\left[j(2\pi f_0 t + \theta)\right]\right\} \tag{4.64}$$

The complex envelope corresponding to $n(t)$ is defined as

$$\widetilde{n}(t) = R(t)\exp\left[j\phi(t)\right] \tag{4.65}$$

In rectangular coordinates

$$\widetilde{n}(t) = n_d(t) + jn_q(t) \tag{4.66}$$

The real envelope $R(t)$ is

$$R(t) = |\widetilde{n}(t)| = \sqrt{n_d^2(t) + n_q^2(t)} \tag{4.67}$$

and

$$\phi(t) = \arctan\frac{n_q(t)}{n_d(t)} \tag{4.68}$$

[6]The random phase is required for the process to be stationary. Without the random phase, which is equivalent to an arbitrary time reference, the process is cyclostationary.

We assume that we know the power spectral density (PSD) of $n(t)$. The problem is to determine the PSD $n_d(t)$, $n_q(t)$, and $\widetilde{n}(t)$.

The problem of determining quadrature models for bandpass random processes is commonly covered in basic courses on communication theory [3]. We cite only the most important results here. All of these results will be useful in the work to follow.

- (Means) Since $n(t)$ is a bandpass process, it is zero mean. Thus, the expectation of the right-hand side of (4.62) is zero mean and, consequently, $n_d(t)$ and $n_q(t)$ are also zero mean. Therefore:

$$E\{n(t)\} = E\{n_d(t)\} = E\{n_q(t)\} = 0 \tag{4.69}$$

  where $E\{\cdot\}$ denotes statistical expectation.

- (Variances) It also follows that $n_d(t)$ and $n_q(t)$ have the same variance (or power, since the process is assumed zero mean) and this power is equal to the total power in the bandpass process. In other words:

$$E\left\{n^2(t)\right\} = E\left\{n_d^2(t)\right\} = E\left\{n_q^2(t)\right\} = N \tag{4.70}$$

  where N is the total power in the underlying bandpass process.

- (PSD of $n_d(t)$ and $n_q(t)$) The PSD of $n_d(t)$ and $n_q(t)$ are equal and are determined from the PSD of $n(t)$, denoted $S_n(f)$, by the expression [3]

$$S_{n_d}(f) = S_{n_q}(f) = \mathrm{Lp}\left[S_n(f - f_0) + S_n(f + f_0)\right] \tag{4.71}$$

- (Autocorrelation of $n_d(t)$ and $n_q(t)$) It follows from the Weiner-Khintchine theorem [3] that

$$R_{n_d}(\tau) \leftrightarrow S_{n_d}(f) \tag{4.72}$$

  and

$$R_{n_q}(\tau) \leftrightarrow S_{n_q}(f) \tag{4.73}$$

  where $R_{n_d}(\tau)$ and $R_{n_q}(\tau)$ are the autocorrelation functions of $n_d(t)$ and $n_q(t)$ and $\longleftrightarrow$ denotes a Fourier transform pair.

- (Cross-PSD) The cross PSD of $n_d(t)$ and $n_q(t)$ is given by

$$S_{n_d n_q}(f) = j\,\mathrm{Lp}[S_n(f - f_0) - S_n(f + f_0)] \tag{4.74}$$

  Note that the cross-PSD is imaginary.

- (Cross-correlation of $n_d(t)$ and $n_q(t)$) We again invoke the Weiner-Khintchine theorem to define the cross-correlation of $n_d(t)$ and $n_q(t)$. The result is

$$R_{n_d n_q}(\tau) \leftrightarrow S_{n_d n_q}(f) \tag{4.75}$$

where $R_{n_d n_q}(\tau)$ is the cross-correlation of $n_d(t)$ and $n_q(t)$. Note that since $S_{n_d n_q}(f)$ is imaginary, $R_{n_d n_q}(\tau)$ is odd. For bandlimited processes the cross-correlation, as well as either autocorrelation, must be continuous. Thus, for a bandlimited process, $n_d(t)$ and $n_q(t)$ are uncorrelated. However, $n_d(t)$ and $n_q(t+\tau)$ may be correlated for $\tau \neq 0$.

- (Mean of the complex envelope $\widetilde{n}(t)$). The mean of $\widetilde{n}(t)$ is

$$E\{\widetilde{n}(t)\} = E\{n_d(t) + jn_q(t)\} \tag{4.76}$$

Since the expectation of the sum is the sum of the expectations

$$E\{\widetilde{n}(t)\} = E\{n_d(t)\} + jE\{n_q(t)\} = 0 + j0 = 0 \tag{4.77}$$

- (Variance of the complex envelope $\widetilde{n}(t)$). The power in $\widetilde{n}(t)$ is

$$P_{\widetilde{n}} = E\left\{|\widetilde{n}(t)|^2\right\} = E\left\{[n_d(t) + jn_q(t)]\left[n_d^*(t) - jn_q^*(t)\right]\right\} \tag{4.78}$$

Since $\widetilde{n}(t)$ is zero mean, $E\left\{|\widetilde{n}(t)|^2\right\}$ is also the variance. Carrying out the indicated multiplication, (4.78) can be written

$$\begin{aligned} P_{\widetilde{n}} = {} & E\left\{|n_d(t)|^2\right\} + E\left\{|n_q(t)|^2\right\} \\ & + jE\left\{n_d^*(t)n_q(t)\right\} - jE\left\{n_d(t)n_q^*(t)\right\} \end{aligned} \tag{4.79}$$

Since $n_d(t)$ and $n_q(t)$ are uncorrelated for bandlimited processes

$$E\{n_d^*(t)n_q(t)\} = E\{n_d^*(t)\}E\{n_q(t)\} = 0 \tag{4.80}$$

and, in a similar fashion

$$E\left\{n_d(t)n_q^*(t)\right\} = E\{n_d(t)\}E\left\{n_q^*(t)\right\} = 0 \tag{4.81}$$

Thus

$$P_{\widetilde{n}} = E\left\{|n_d(t)|^2\right\} + E\left\{|n_q(t)|^2\right\} = P_{n_d} + P_{n_q} \tag{4.82}$$

where P_{n_d} and P_{n_q} represent the power in $n_d(t)$ and $n_q(t)$, respectively. It follows from (4.70) that

$$P_{\widetilde{n}} = 2E\left\{n^2(t)\right\} = 2N \tag{4.83}$$

which means that the power in the lowpass complex envelope representation of a bandpass signal is double the power of the real bandpass signal from which the lowpass complex envelope is derived.

- (PSD of the complex envelope $\widetilde{n}(t)$). From (4.71) we know that the PSD of $n_d(t)$ and $n_q(t)$ are equal. Thus

$$S_{\widetilde{n}}(f) = S_{n_d}(f) + S_{n_q}(f) = 2S_{n_d}(f) \tag{4.84}$$

We now illustrate why the preceding relationships are important in the simulation context.

4.1.6 Signal-to-Noise Ratios

As we know from basic communication theory, the signal-to-noise ratio (SNR) at the input of a receiver is usually a major factor in determining the performance of the system. At a receiver input, both the signal and the noise are bandpass. Assuming that the signal and noise are additive, the receiver input is

$$z(t) = x(t) + n(t) \tag{4.85}$$

where $x(t)$ is the signal and $n(t)$ represents the noise. The signal-to-noise ratio, in terms of the real bandpass signals, is defined as

$$(SNR)_{bp} = \frac{E\left\{x^2(t)\right\}}{E\left\{n^2(t)\right\}} \tag{4.86}$$

It follows from (4.58) and (4.83) that

$$(SNR)_{bp} = \frac{\frac{1}{2}E\left\{\left|\widetilde{x}(t)\right|^2\right\}}{\frac{1}{2}E\left\{\left|\widetilde{n}(t)\right|^2\right\}} = \frac{E\left\{\left|\widetilde{x}(t)\right|^2\right\}}{E\left\{\left|\widetilde{n}(t)\right|^2\right\}} = (SNR)_{lp} \tag{4.87}$$

where $(SNR)_{bp}$ and $(SNR)_{lp}$ refer to the signal-to-noise ratio based on the real bandpass signals and the corresponding lowpass complex envelopes, respectively. This very important result shows that representing bandpass functions (both signal and noise) by their respective lowpass equivalents, which is standard simulation methodology, preserves the signal-to-noise ratios. A simple example further illustrates this important fact by viewing the underlying signals in the frequency domain.

Example 4.6. (SNR Transformations) As a simple example, assume that a bandpass signal is represented by

$$z(t) = x(t) + n(t) = x(t) + n_d(t)\cos(2\pi f_0 t + \theta) - n_q(t)\sin(2\pi f_0 t + \theta) \tag{4.88}$$

where $x(t)$ is the sinusoid

$$x(t) = A\cos(2\pi f_0 t) \tag{4.89}$$

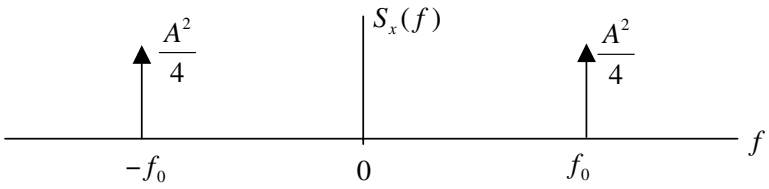
The bandpass signal can be represented

$$x(t) = \frac{A}{2}\exp\left(j2\pi f_0 t\right) + \frac{A}{2}\exp\left(-j2\pi f_0 t\right) \tag{4.90}$$

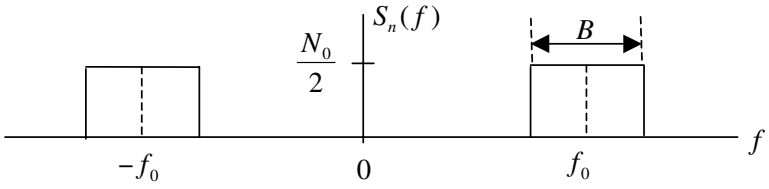
This PSD of $x(t)$ is, therefore:

$$S_x(f) = \frac{A^2}{4}\delta(f - f_0) + \frac{A^2}{4}\delta(f + f_0) \tag{4.91}$$

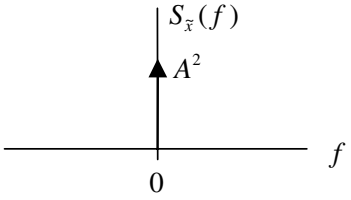
as illustrated in Figure 4.9(a). Thus, the total power in the real bandpass signal is



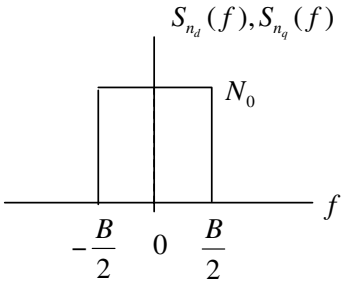
(a) PSD of bandpass signal model



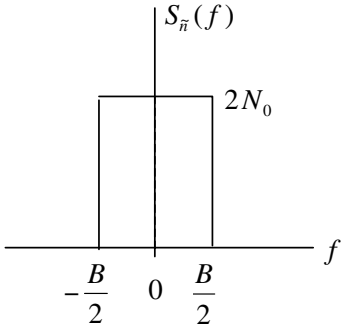
(b) PSD of bandpass noise model



(c) PSD of lowpass signal model



(d) PSD of direct and quadrature noise components



(e) PSD of lowpass noise model

Figure 4.9 Calculation of signal-to-noise ratio.

$$P_x = \int_{-\infty}^{\infty} S_x(f)\, df = \frac{A^2}{2} \tag{4.92}$$

as expected from (4.89). The PSD of the assumed noise is illustrated in Figure 4.9(b). Thus, the total noise power is

$$P_n = \int_{-\infty}^{\infty} S_n(f)\, df = 2\left(\frac{N_0}{2}B\right) = N_0 B \tag{4.93}$$

This gives, from (4.92) and (4.93)

$$(SNR)_{bp} = \frac{A^2}{2N_0 B} \tag{4.94}$$

for the bandpass signal-to-noise ratio.

We now turn our attention to the lowpass equivalents of $x(t)$ and $n(t)$. Equation (4.89) can be written

$$x(t) = \mathrm{Re}\left\{A \exp\left(j2\pi f_0 t\right)\right\} \tag{4.95}$$

from which the complex envelope is

$$\widetilde{x}(t) = A \tag{4.96}$$

The power in the complex envelope is

$$P_{\widetilde{x}} = E\left\{\left|\widetilde{x}(t)\right|^2\right\} = A^2 \tag{4.97}$$

which gives the PSD

$$P_{\widetilde{x}}(f) = A^2\delta(f) \tag{4.98}$$

as shown in Figure 4.9(c). The PSD of $n_d(t)$ and $n_q(t)$ given by (4.71) and is illustrated in Figure 4.9(d). From (4.84) the PSD of $\widetilde{n}(t)$ is found by multiplying the PSD illustrated in Figure 4.9(d) by two. The result is illustrated in Figure 4.9(e). From Figure 4.9(e) the power in the complex lowpass representation of the noise is

$$P_{\widetilde{n}} = \int_{-\infty}^{\infty} S_{\widetilde{n}}(f)\, df = 2N_0 B \tag{4.99}$$

Combining (4.97) and (4.99) yields

$$(SNR)_{lp} = \frac{P_{\widetilde{x}}}{P_{\widetilde{n}}} = \frac{A^2}{2N_0 B} \tag{4.100}$$

Comparing (4.94) and (4.100) we see that the signal-to-noise ratio is preserved when real bandpass signals are replaced by their complex lowpass equivalents. ■

4.2 Linear Bandpass Systems

We now turn our attention from signals to systems. The basic problem is to determine the time-domain input-output relationship for a linear system given that the input to the system and the unit impulse response of the system are both bandpass signals expressed in terms of lowpass complex envelopes. The result will provide us with a methodology for developing waveform-level simulations of linear systems based on lowpass models.

4.2.1 Linear Time-Invariant Systems

Assuming that a system is linear, we know that convolution may be used to determine the output, $y(t)$, given the input, $x(t)$. For the time-invariant case the convolution takes the simple form

$$y(t) = \int_{-\infty}^{\infty} x(\lambda)h(t-\lambda)\,d\lambda \triangleq x(t) \circledast h(t) \tag{4.101}$$

where $h(t)$ is the unit impulse response of the time-invariant system and the symbol $\circledast$ is used to denote convolution. By definition, the complex envelopes of the linear time-invariant (LTIV) system input, $\widetilde{x}(t)$, and output, $\widetilde{y}(t)$, are defined by

$$x(t) = \mathrm{Re}\left\{\widetilde{x}(t)\exp(j2\pi f_0 t)\right\} \tag{4.102}$$

and

$$y(t) = \mathrm{Re}\left\{\widetilde{y}(t)\exp(j2\pi f_0 t)\right\} \tag{4.103}$$

respectively. If we require that the relationship between $\widetilde{x}(t)$ and $\widetilde{y}(t)$ satisfy

$$\widetilde{y}(t) = \int_{-\infty}^{\infty} \widetilde{x}(\lambda)\widetilde{h}(t-\lambda)\,d\lambda \triangleq \widetilde{x}(t) \circledast \widetilde{h}(t) \tag{4.104}$$

so that (4.101) and (4.104) have exactly the same form, the unit impulse response of the bandpass system $h(t)$ and the corresponding complex envelope, $\widetilde{h}(t)$ must be related by

$$h(t) = \mathrm{Re}\left\{2\widetilde{h}(t)\exp\left(j2\pi f_0 t\right)\right\} \tag{4.105}$$

A formal proof of the preceding statements appear in Appendix B. A consequence of the factor of 2 is that a unity gain bandpass filter corresponds to a complex envelope representation of a unity gain lowpass filter.

Equation (4.105) is easily justified by showing that the factor of 2 in (4.105) results in the transformation of a unity gain bandpass filter into the unity gain lowpass filter so that the factor of 2 preserves the filter passband gain. Consider the ideal bandpass filter characteristic illustrated in Figure 4.10(a). The transfer function of the ideal bandpass filter is represented by

$$H(f) = H_+(f) + H_-(f) \tag{4.106}$$

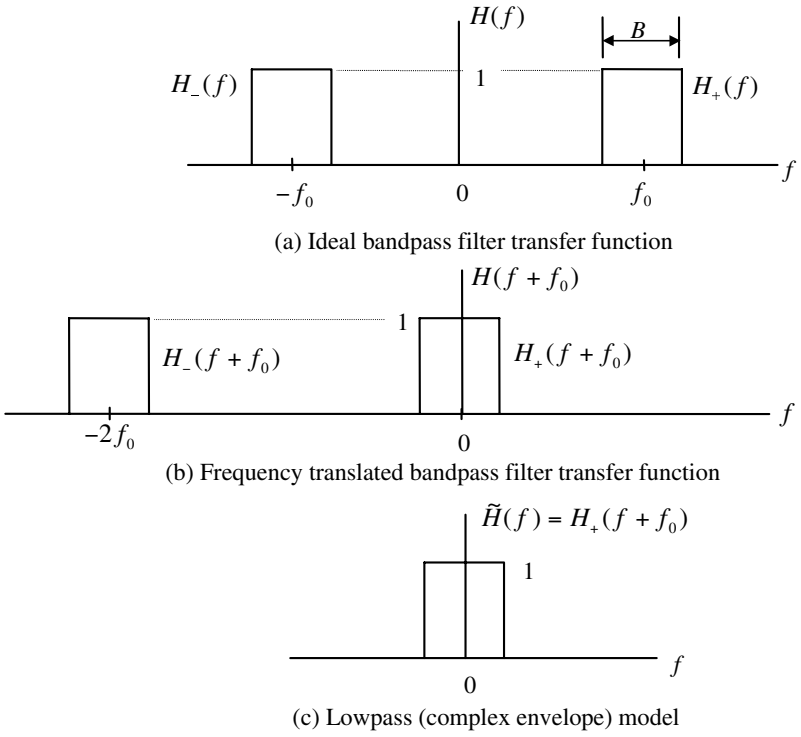


Figure 4.10 Ideal bandpass filter and lowpass model.

where $H_+(f)$ and $H_-(f)$ are the positive frequency and negative frequency portions of the bandpass filter transfer function, respectively. Replacing f by $f + f_0$ gives

$$H(f + f_0) = H_+(f + f_0) + H_-(f + f_0) \tag{4.107}$$

as illustrated in Figure 4.10(b). Clearly $H_+(f + f_0)$ is a lowpass function and is therefore defined as $\widetilde{H}(f)$. Thus:

$$\widetilde{H}(f) = H_+(f + f_0) \tag{4.108}$$

as shown in Figure 4.10(c). Note that (4.108) can also be written

$$\widetilde{H}(f) = H(f + f_0)U(f + f_0) \tag{4.109}$$

We see that a unity gain bandpass filter maps to a unity gain lowpass filter, since $\widetilde{H}(f)$ is derived from the positive frequency portion of the transfer function for the bandpass filter by a simple frequency translation, and that no amplitude scaling is involved.

It is important to understand the difference between (4.48) and (4.109). Representing bandpass signals by their lowpass complex envelope results in

$$\widetilde{X}(f) = 2X(f + f_0)U(f + f_0) \tag{4.110}$$

while representing linear bandpass systems by their lowpass equivalent results in

$$\widetilde{H}(f) = H(f + f_0)U(f + f_0) \tag{4.111}$$

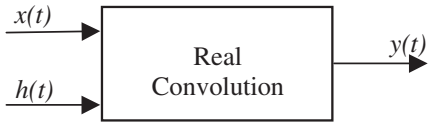
Except for the factor of 2, the two expressions are equivalent.

From (4.101) and (4.104) we see that there are two techniques that can be used to compute, in the time domain, the output of an LTIV system given the bandpass input signal and the unit impulse response of the network. The two techniques are illustrated in Figure 4.11. The first technique is to simply convolve $x(t)$ and $h(t)$ as defined by (4.101) and illustrated in Figure 4.11(a). The second technique is to determine the complex envelopes of $x(t)$ and $h(t)$, convolve the complex envelopes as defined by (4.104), and then determine the bandpass output signal, $y(t)$, using (4.103). This technique is shown in Figure 4.11(b).

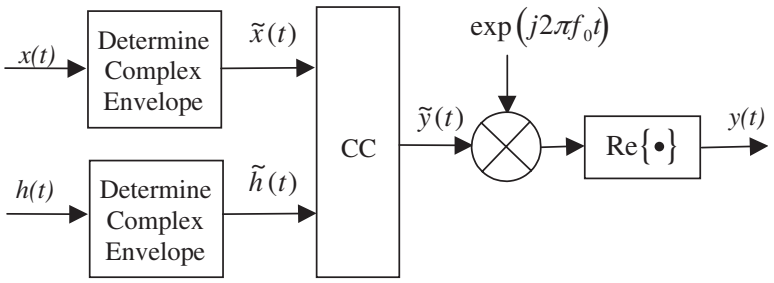
Using (4.104) we can write

$$\widetilde{y}(t) = y_d(t) + jy_q(t) = [x_d(t) + jx_q(t)] \circledast [h_d(t) + jh_q(t)] \tag{4.112}$$

Since the convolution of a sum is the sum of convolutions (linear operations again) we have



(a) System analysis using bandpass signals



(b) System analysis using complex envelope signals
(CC denotes complex convolution)

Figure 4.11 Time-domain signal analysis techniques.

$$y_d(t) + jy_q(t) = [x_d(t) \circledast h_d(t) - x_q(t) \circledast h_q(t)] + j\,[x_d(t) \circledast h_q(t) + x_q(t) \circledast h_d(t)] \tag{4.113}$$

Thus, the direct component of a linear system output is given by

$$y_d(t) = x_d(t) \circledast h_d(t) - x_q(t) \circledast h_q(t) \tag{4.114}$$

and the quadrature component of a linear system output is given by

$$y_q(t) = x_d(t) \circledast h_q(t) + x_q(t) \circledast h_d(t) \tag{4.115}$$

The convolution of two complex functions is therefore equivalent to four real convolutions just as the multiplication of two complex numbers is equivalent to four real multiplications. The operations for deriving the direct and quadrature outputs of a linear bandpass system are defined by the operations shown in Figure 4.12.

Example 4.7. In this example we determine the values of $h_d(t)$ and $h_q(t)$ for a bandpass phase shifter. Assume that the input to the system is

$$x(t) = A\cos(2\pi f_0 t + \theta) \tag{4.116}$$

and that the output of the phase shifter is

$$y(t) = A\cos(2\pi f_0 t + \theta + \phi) \tag{4.117}$$

so that the system shifts the input phase by ϕ. This model could be used to represent synchronization errors in a demodulator. In order to simulate this device using complex lowpass models $h_d(t)$ and $h_q(t)$ must be derived.

The complex envelope of $x(t)$ and $y(t)$ are given by

$$\widetilde{x}(t) = A\exp(j\theta) \tag{4.118}$$

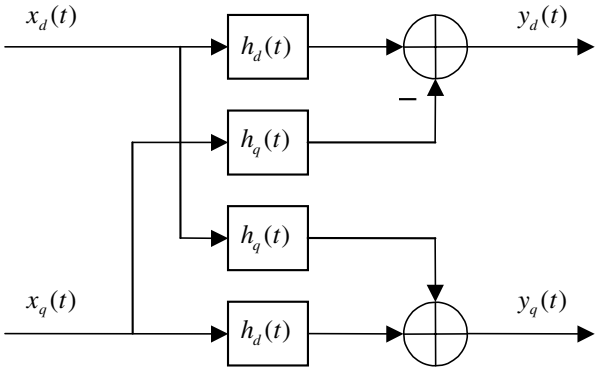


Figure 4.12 Model for linear bandpass system.

and

$$\widetilde{y}(t) = A\exp\left[j\left(\theta+\phi\right)\right] = A\exp(j\theta)\exp(j\phi) \tag{4.119}$$

respectively. It follows that

$$\widetilde{y}(t) = \widetilde{x}(t)\exp(j\phi) \tag{4.120}$$

Writing this in rectangular coordinates yields

$$\left[y_d(t) + jy_q(t)\right] = \left[x_d(t) + jx_q(t)\right]\left[\cos\phi + j\sin\phi\right] \tag{4.121}$$

Equating real parts gives

$$y_d(t) = x_d(t)\cos\phi - x_q(t)\sin\phi \tag{4.122}$$

and equating imaginary parts gives

$$y_q(t) = x_d(t)\sin\phi + x_q(t)\cos\phi \tag{4.123}$$

In this example $x_d(t)$ and $x_q(t)$ are multiplied by the $\cos\phi$ and $\sin\phi$, respectively. Thus:

$$h_d(t) = (\cos\phi)\delta(t) \tag{4.124}$$

and

$$h_q(t) = (\sin\phi)\delta(t) \tag{4.125}$$

Note that the delta function is present because the system is memoryless.

The lowpass model of a phase-shift network is illustrated in Figure 4.12 with $h_d(t)$ and $h_q(t)$ given by (4.124) and (4.125), respectively. Of course, ϕ could be time varying. ■

4.2.2 Derivation of $h_d(t)$ and $h_q(t)$ from $H(f)$

In order to simulate a system using bandpass components, such as bandpass filters, we typically know the transfer function, $H(f)$. The lowpass simulation model for the filter is that illustrated in Figure 4.12. In order to develop the simulation model for the filter, which is based on the complex envelope of $h(t)$, it is necessary to determine $h_d(t)$ and $h_q(t)$ from $H(f)$, the transfer function of the bandpass filter. There are two fundamental methods for finding $h_d(t)$ and $h_q(t)$. The first method is to determine $H_d(f)$ and $H_q(f)$ from $H(f)$ and inverse transform $H_d(f)$ and $H_q(f)$ in order to establish $h_d(t)$ and $h_q(t)$. The second method is to determine $\widetilde{H}(f)$ from $H(f)$, inverse transform $\widetilde{H}(f)$ to find $\widetilde{h}(t)$, and take the real and imaginary parts of $\widetilde{h}(t)$ to determine $h_d(t)$ and $h_q(t)$.

The first step in either of the two methods of finding $h_d(t)$ and $h_q(t)$ from $H(f)$ is to determine $\widetilde{H}(f)$. By definition

$$\widetilde{H}(f) = H_+(f+f_0) = H_d(f) + jH_q(f) \tag{4.126}$$

Taking the inverse Fourier transform yields

$$\widetilde{h}(t) = h_d(t) + jh_q(t) \tag{4.127}$$

The real and imaginary parts of $\widetilde{h}(t)$ give $h_d(t)$ and $h_q(t)$, respectively.

Replacing f by $-f$ in (4.126) gives

$$\widetilde{H}(-f) = H_d(-f) + jH_q(-f) \tag{4.128}$$

Since $h_d(t)$ and $h_q(t)$ are both real functions of time, basic Fourier transform theory tells us that $H_d(-f)$ is the complex conjugate of $H_d(f)$ and that $H_q(-f)$ is the complex conjugate of $H_q(f)$. Thus, (4.128) can be written

$$\widetilde{H}(-f) = H_d^*(f) + jH_q^*(f) \tag{4.129}$$

Note that $\widetilde{H}(f)$ and $\widetilde{H}(-f)$ are not complex conjugate pairs, because $\widetilde{h}(t)$ is not, in general, a real function of time. Taking the complex conjugate of (4.129) gives

$$\widetilde{H}^*(-f) = H_d(f) - jH_q(f) \tag{4.130}$$

Adding (4.126) and (4.130) gives

$$H_d(f) = \frac{1}{2}\left(\widetilde{H}(f) + \widetilde{H}^*(-f)\right) \tag{4.131}$$

Multiplying (4.130) by -1 and adding the result to (4.126) gives

$$H_q(f) = \frac{1}{j2}\left(\widetilde{H}(f) - \widetilde{H}^*(-f)\right) \tag{4.132}$$

The functions $h_d(t)$ and $h_q(t)$ are then obtained from the inverse Fourier transforms of $H_d(f)$ and $H_q(f)$, respectively. If $\widetilde{H}(f) = \widetilde{H}^*(-f)$, so that $\widetilde{H}_q(f)$ and $h_q(t)$ are both zero, $H(f)$ is said to exhibit conjugate symmetry about f_0. Figure 4.13 illustrates $\widetilde{H}(f)$, $\widetilde{H}^*(-f)$, $H_d(f)$ and $H_q(f)$ for the case in which $H(f)$ is an ideal bandpass filter.

Example 4.8. We now consider the determination of $h_d(t)$ and $h_q(t)$ directly from $H(f)$. From $H(f)$, $\widetilde{H}(f)$ is written using (4.108) as illustrated in Figure 4.13. The inverse transform of $\widetilde{H}(f)$ is

$$\widetilde{h}(t) = \int_{-\infty}^{\infty} \widetilde{H}(f)\exp\left(j2\pi ft\right)\,df = \int_{f_l - f_0}^{f_u - f_0} \exp\left(j2\pi ft\right)\,df \tag{4.133}$$

since $\widetilde{H}(f) = 1$ over the range of integration. This integrates to

$$\widetilde{h}(t) = \frac{1}{j2\pi t}\left[\exp\left(j2\pi(f_u - f_0)t\right) - \exp\left(j2\pi(f_l - f_0)t\right)\right] \tag{4.134}$$

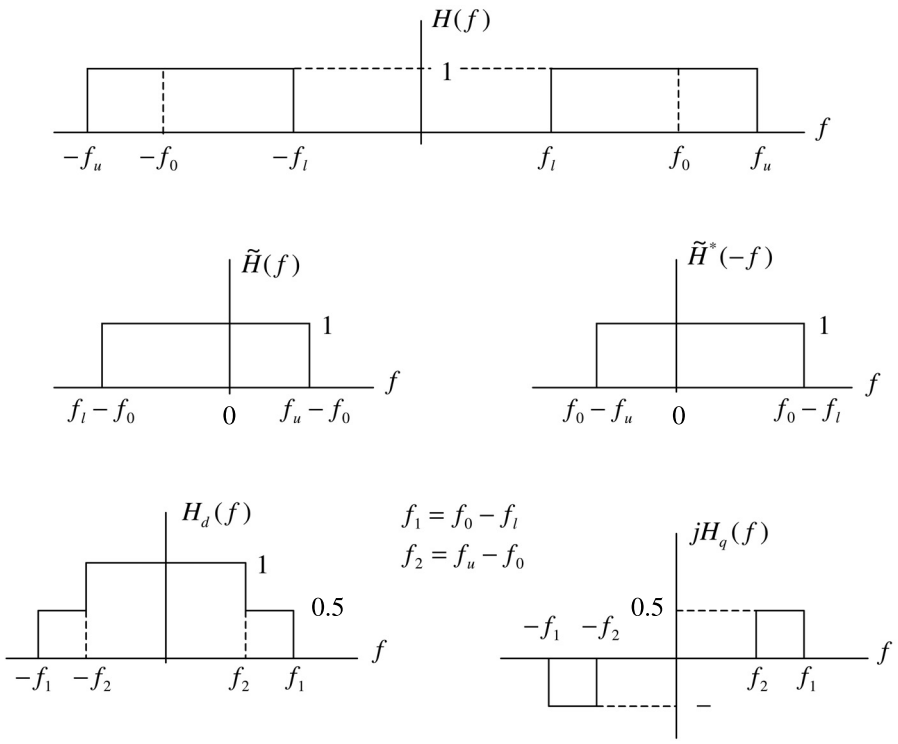


Figure 4.13 $\widetilde{H}(f)$, $\widetilde{H}^*(-f)$, $H_d(f)$, and $H_q(f)$ for an assumed $H(f)$.

The preceding expression can be written

$$\widetilde{h}(t) = \frac{1}{j2\pi t}\left[\exp\left(j\pi(f_u - f_l)\,t\right) - \exp(j\pi\,(f_l - f_u)\,t)\right] \cdot \exp\left(-j2\pi\left(f_0 - \frac{f_u + f_l}{2}\right)t\right) \tag{4.135}$$

which is

$$\widetilde{h}(t) = \frac{1}{\pi t}\sin\left(\pi(f_u - f_l)t\right)\exp\left(-j2\pi\left(f_0 - \frac{f_u + f_l}{2}\right)t\right) \tag{4.136}$$

Taking the real and imaginary parts yields

$$h_d(t) = \frac{1}{\pi t}\sin\left(\pi(f_u - f_l)t\right)\cos\left(2\pi\left(f_0 - \frac{f_u + f_l}{2}\right)t\right) \tag{4.137}$$

and

$$h_q(t) = -\frac{1}{\pi t}\sin\left(\pi(f_u - f_l)t\right)\sin\left(2\pi\left(f_0 - \frac{f_u + f_l}{2}\right)t\right) \tag{4.138}$$

Note that if f_0 is selected to be the algebraic center frequency $(f_u + f_l)/2$, $h_q(t) = 0$ for all t. This obviously simplifies the lowpass simulation model illustrated in Figure 4.12. The important consequence is that the computational burden associated with finding the system output, given the system input, is reduced by a factor of 2. ■

As implied in the preceding example, in many cases of practical interest f_0 can be selected so that $\widetilde{h}_q(t) \ll \widetilde{h}_d(t)$ for all t. In such cases $h_q(t)$ can often be neglected so that the complex envelope of the impulse response can be approximated as a real function without significant loss of accuracy. As pointed out in the preceding example, it is important to take advantage of this approximation when applicable, since elimination of $h_q(t)$ reduces the computational burden of the filtering operation by a factor of 2. It follows from basic Fourier transform theory that $\widetilde{h}(t)$ is real if $\widetilde{H}(f)$ exhibits conjugate symmetry (even amplitude spectrum and odd phase spectrum) about $f = 0$. This will be the case if $H(f)$, the transfer function of the bandpass filter, exhibits conjugate symmetry about $f = f_0$. Most filter designs closely approximate this property if the bandwidth of the filter is small compared to the center frequency of the filter. The quadrature component can be viewed as a measure of the conjugate asymmetry of $H(f)$ about f_0.

4.3 Multicarrier Signals

Consider the frequency division multiplex (FDM) of M signals

$$y(t) = \sum_{i=1}^{M} a_i(t) \cos\left[2\pi f_i t + \phi_i(t)\right] \tag{4.139}$$

where $a_i(t)$ and $\phi_i(t)$ represent the amplitude and phase modulation on the i^{th} carrier, respectively, and f_i is the i^{th} carrier frequency. Since the terms $a_i(t)$ are real, we may write

$$y(t) = \text{Re}\left\{\sum_{i=1}^{M} a_i(t) \exp\left[j\phi_i(t)\right] \exp\left(j2\pi f_i t\right)\right\} \tag{4.140}$$

Defining

$$\widetilde{x}_i(t) = a_i(t) \exp\left[j\phi_i(t)\right] \tag{4.141}$$

gives

$$y(t) = \text{Re}\left\{\sum_{i=1}^{M} \widetilde{x}_i(t) \exp\left(j2\pi f_i t\right)\right\} \tag{4.142}$$

We can define the complex envelope of $y(t)$ as

$$y(t) = \sum_{i=1}^{M} \widetilde{y}(t) \exp\left(j2\pi f_0 t\right) \tag{4.143}$$

where, for the moment, f_0 remains arbitrary. With this definition $y(t)$ can be written

$$y(t) = \text{Re}\left\{\sum_{i=1}^{M} \widetilde{x}_i(t) \exp\left[j2\pi(f_i - f_0)t\right] \exp(j2\pi f_0 t)\right\} \tag{4.144}$$

Thus, the complex envelope of $y(t)$ is

$$\begin{aligned}\widetilde{y}(t) &= \sum_{i=1}^{M} \widetilde{x}_i(t) \exp\left[j2\pi(f_i - f_0)t\right] \\ &= \sum_{i=1}^{M} a_i(t) \exp\left[j\phi_i(t)\right] \exp\left[j2\pi(f_i - f_0)t\right]\end{aligned} \tag{4.145}$$

The direct and quadrature components of the FDM signal are therefore given by

$$y_d(t) = \sum_{i=1}^{M} a_i(t) \cos\left[2\pi(f_i - f_0)t + \phi_i(t)\right] \tag{4.146}$$

and

$$y_q(t) = \sum_{i=1}^{M} a_i(t) \sin\left[2\pi(f_i - f_0)t + \phi_i(t)\right] \tag{4.147}$$

respectively.

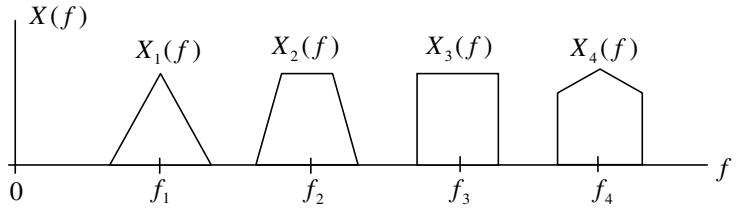
Example 4.9. Consider the frequency division multiplex (FDM) signal consisting of four channel signals as shown in Figure 4.14(a). Suppose that the signal of interest is $x_2(t)$ and that a simulation is being performed in order to examine the effects of adjacent channel interference and intermodulation distortion resulting from a nonlinear amplifier in the system. Since we have interest in $x_2(t)$, we shall let $f_0 = f_2$ so that $x_2(t)$ is translated to $f = 0$ when the complex envelope of the composite signal $y(t)$ is formed. With $f_0 = f_2$ the complex envelope of $y(t)$ is

$$\widetilde{y}(t) = \sum_{i=1}^{4} \widetilde{x}_i(t) \exp\left[j2\pi(f_i - f_2)t\right] \tag{4.148}$$

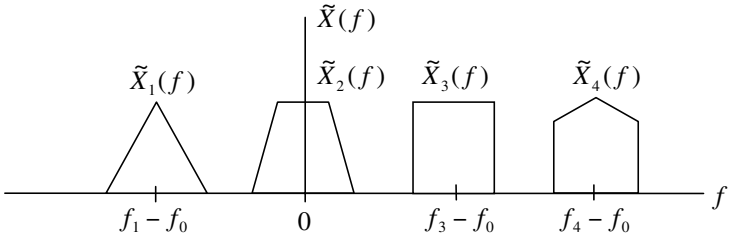
This is illustrated in Figure 4.14(b) for $f_0 = f_2$. Note that the minimum sampling frequency for $\widetilde{y}(t)$ is dependent on which bandpass signal is translated to $f = 0$. For the case shown, $f_0 = f_2$, the sampling frequency must satisfy

$$f_s > 2\left[(f_4 - f_2) + \frac{B}{2}\right] \tag{4.149}$$

where B is the bandwidth of $X_4(f)$. The operations involved in forming $\widetilde{y}(t)$ are illustrated in Figure 4.15. ■



(a) Spectrum of FDM signal



(b) Spectrum of complex envelope with $f_0 = f_2$

Figure 4.14 Complex envelope of an FDM signal.

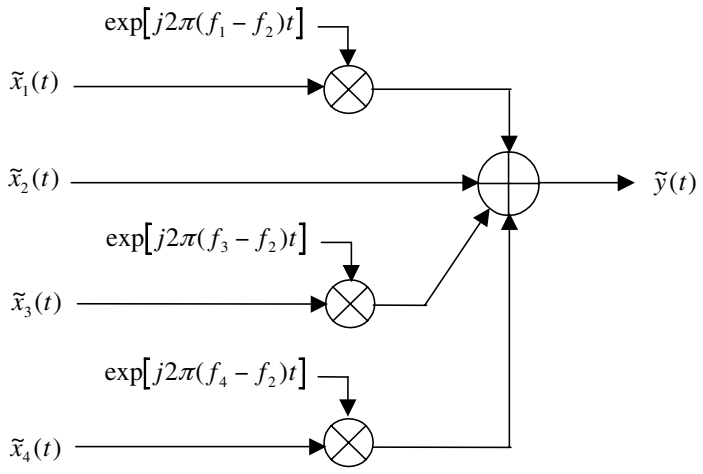


Figure 4.15 Signal processing for the complex envelope of an FDM signal.

4.4 Nonlinear and Time-Varying Systems

Throughout this chapter, the focus has been on fixed (time-invariant) linear systems. Many systems of practical interest involve time-varying components, such as the wireless radio channel, or nonlinear components, such as high-power amplifiers operating near the point of saturation. Design and analysis of systems that are nonlinear, or time-varying, or both nonlinear and time-varying, using traditional mathematical tools are usually very difficult or even impossible. As a result, simulation is frequently used as a design and analysis tool for these systems. Chapter 12 focuses on nonlinear systems and Chapter 13 is devoted to time-varying systems. However, for completeness we very briefly consider nonlinear and time-varying systems here.

4.4.1 Nonlinear Systems

The basic concept of a transfer function is not defined for a nonlinear system. Even though an impulse response can be measured for a nonlinear system, it does not in general relate the system output to the system input through convolution. The familiar convolution integral is based on the concept of superposition, which does not hold for nonlinear systems. Simulation models for nonlinear systems can certainly be developed but they are typically based on measurements obtained from physical systems. Analysis can sometimes be used to develop simulation models for nonlinear systems, but the techniques are usually ad hoc and cannot be generalized. Several important simulation models for nonlinear systems are explored in Chapter 12.

We now pause to consider a simulation model for a simple nonlinear system. The model developed in this example will be useful in our later work.

Example 4.10. Assume that the input to a system has the form

$$x(t) = A(t)\cos\left[2\pi f_0 t + \theta(t)\right] \tag{4.150}$$

Measurements made at the system output for various choices of $A(t)$ and $\theta(t)$ show that the envelope of the system output is a constant independent of $A(t)$ but that the zero crossings of the input are preserved and match the zero crossings of the output. Thus, the measurements suggest that the system can be accurately modeled by a bandpass hard limiter. Our task is to develop a simulation model for the device.

The complex envelope of the system input is

$$\widetilde{x}(t) = A(t)\exp\left[j\theta(t)\right] = x_d(t) + x_q(t) \tag{4.151}$$

The output of the bandpass hard limiter is defined as a sinusoid having a constant amplitude and a phase deviation equal to the phase deviation of the input. Thus

$$y(t) = B\cos\left[2\pi f_0 t + \theta(t)\right] \tag{4.152}$$

for which the complex envelope is

$$\widetilde{y}(t) = B\exp\left[j\theta(t)\right] \tag{4.153}$$

where B is an assumed positive constant. We see that, for a bandpass hard limiter, it is the envelope of the signal that is hard limited rather than the signal itself.

Note that by definition of magnitude

$$\sqrt{x_d^2(t) + x_q^2(t)} = |A(t)| \tag{4.154}$$

The complex envelope of the output signal can be written

$$\widetilde{y}(t) = \frac{B}{|A(t)|}\widetilde{x}(t) \tag{4.155}$$

Thus

$$\widetilde{y}(t) = \frac{B\left[x_d(t) + jx_q(t)\right]}{\sqrt{x_d^2(t) + x_q^2(t)}} \tag{4.156}$$

from which

$$y_d(t) = \frac{Bx_d(t)}{\sqrt{x_d^2(t) + x_q^2(t)}} \tag{4.157}$$

and

$$y_q(t) = \frac{Bx_q(t)}{\sqrt{x_d^2(t) + x_q^2(t)}} \tag{4.158}$$

The device defined by (4.157) and (4.158) is referred to as a bandpass hard limiter. It removes all variations on the envelope while preserving the zero-crossing locations. ■

A number of demodulators are based on nonlinear operations. For example, assume that $r(t)$, defined by

$$r(t) = A(t)\cos\left[2\pi f_c t + \phi(t)\right] \tag{4.159}$$

represents a received signal at the input of a demodulator. Also assume that the purpose of the demodulator is to remove the positive portion of envelope as is the case for AM [3].[7] The envelope detector, which is usually used for envelope recovery (AM demodulation), has the output $z(t)$ defined by

$$z(t) = |\widetilde{r}(t)| = |A(t)\exp\left[j\phi(t)\right]| \tag{4.160}$$

[7]Recall that an AM (amplitude modulated) signal is defined by

$$x(t) = A\left[1 + am(t)\right]\cos\left[2\pi f_c t + \phi(t)\right]$$

where a is the modulation index and $m(t)$ is the message signal normalized so that $am(t) \leq 1$ for all t. For this signal the positive portion of the envelope, with the dc term removed, is the message signal to be recovered by the demodulator.

which is

$$z(t) = |A(t)| \tag{4.161}$$

Also useful for carrier recovery is the square-law demodulator defined by

$$z(t) = |\widetilde{r}(t)|^2 = |A(t)|^2 \tag{4.162}$$

The envelope and square-law demodulators are examples of noncoherent nonlinear demodulators and are used when recovery of the carrier phase deviation $\phi(t)$ is not required.

A number of situations exist in which recovery of the carrier phase deviation is required. Examples are demodulators for analog FM and PM signals and demodulators for PSK and QPSK digital signals. The basic building block for demodulators requiring phase recovery is the phase-locked loop (PLL). The phase-locked loop is a nonlinear system and is covered in detail in Chapter 6.

4.4.2 Time-Varying Systems

In contrast to the nonlinear case, if a system is linear but time-varying, many of the tools developed in this chapter can be used for analysis and system-modeling purposes. This is true because, as long as a system is linear, convolution can still be used to relate the system input and output in the time domain and transfer functions can be used to relate the system input and output in the frequency domain. The system impulse response and the system transfer function are defined for linear time-varying systems. However, both the impulse response and the transfer function must be modified from their time-invariant definitions to account for the time-varying nature of the system.

For example, if a linear system is time-varying, the system input $x(t)$ and output $y(t)$ are related by the convolution of complex envelopes

$$\widetilde{y}(t) = \int_{-\infty}^{\infty} \widetilde{h}(\tau, t)\widetilde{x}(t-\tau)\, d\tau \tag{4.163}$$

where $\widetilde{h}(\tau, t)$ is the time-varying impulse response of the system. The impulse response $\widetilde{h}(\tau, t)$ is defined as the response of the system, measured at time t, to an impulse applied at the input τ seconds earlier. In other words, an impulse is applied to the system input at time $t - \tau$ and the response is measured at time t, after an "elapsed time" of τ. For a time-invariant system, the impulse response is a function only of the time difference $t - \tau$. The impulse is assumed to be applied at $t - \tau = 0$ and the resulting impulse response is the familiar $h(\tau)$.

Since the impulse response of a time-varying system is a function of two time-domain variables, t and τ, the transfer function of a time-varying system is also a function of two frequency-domain variables. It is defined as

$$\widetilde{H}(f_1, f_2) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \widetilde{h}(\tau, t) \exp(-j2\pi f_1 \tau - j2\pi f_2 t)\, d\tau\, dt \tag{4.164}$$

It follows that $\widetilde{h}(\tau, t)$ and $\widetilde{H}(f_1, f_2)$ are a Fourier transform pair.

These concepts will be expanded upon in Chapter 13 and the results will be a theoretical framework for the design, analysis, and simulation of linear time-varying systems. Thus, even though the focus of this chapter is linear time-invariant systems, many of the concepts discussed in this chapter serve as a foundation for developing a methodology for simulating time-varying systems.

Example 4.11. Consider the situation depicted in Figure 4.16. A receiver in a moving automobile (mobile) receives a signal from a single transmitter that has propagated along two paths. One propagation path is a direct path from the transmitter to the mobile. The second path is due to a reflection off a building. This is often referred to as the two-ray model. The automobile is assumed to be moving as shown. As a result of this movement, the lengths of both paths change with time. Consequently both the signal attenuation and the propagation delay for each path are time-varying. Assume that the attenuation and delay associated with the n^{th} signal path are denoted $a_n(t)$ and $\tau_n(t)$, respectively. The received signal can then be defined by the simple channel model

$$y(t) = a_1(t)x(t - \tau_1(t)) + a_2(t)x(t - \tau_2(t)) = \sum_{n=1}^{2} a_n(t)x(t - \tau_n(t)) \tag{4.165}$$

We assume that the input to the channel input is general modulated signal [see (4.1)]

$$x(t) = A(t)\cos(2\pi f_c t + \phi(t)) \tag{4.166}$$

Substitution of (4.166) into (4.165) yields

$$y(t) = \sum_{n=1}^{2} a_n(t)A(t - \tau_n(t))\cos\left[2\pi f_c(t - \tau_n(t)) + \phi(t - \tau_n(t))\right] \tag{4.167}$$

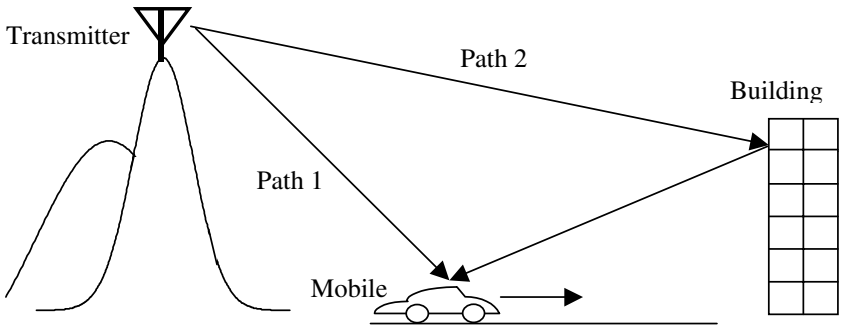


Figure 4.16 Two-ray mobile communications propagation equipment.

Since waveform-level simulation is usually accomplished using complex envelope signals, we now determine the complex envelope for both $x(t)$ and $y(t)$.

The complex envelope of the transmitted signal is, by inspection:

$$\widetilde{x}(t) = A(t)\exp\left[\phi(t)\right] \tag{4.168}$$

Determining the complex envelope of the received signal defined by (4.167) takes a little more effort. Since $a_n(t)$ and $A(t)$ are both real, (4.167) can be written

$$y(t) = \mathrm{Re}\left\{\sum_{n=1}^{2} a_n(t)A(t-\tau_n(t))\exp\left[j\phi(t-\tau_n(t))\right] \cdot \exp\left[-j2\pi f_c\tau_n(t)\right]\exp(j2\pi f_c t)\right\} \tag{4.169}$$

From (4.168)

$$\widetilde{x}(t-\tau_n(t)) = A(t-\tau_n(t))\exp\left[j\phi(t-\tau_n(t))\right] \tag{4.170}$$

so that

$$y(t) = \mathrm{Re}\left\{\sum_{n=1}^{2} a_n(t)\widetilde{x}(t-\tau_n(t))\exp\left[-j2\pi f_c\tau_n(t)\right]\exp(j2\pi f_c t)\right\} \tag{4.171}$$

The complex path attenuation is defined as

$$\widetilde{a}_n(t) = a_n(t)\exp\left[-j2\pi f_c\tau_n(t)\right] \tag{4.172}$$

so that

$$y(t) = \mathrm{Re}\left\{\sum_{n=1}^{2} \widetilde{a}_n(t)\widetilde{x}(t-\tau_n(t))\exp(j2\pi f_c t)\right\} \tag{4.173}$$

Thus, the complex envelope of the receiver input is

$$\widetilde{y}(t) = \sum_{n=1}^{2} \widetilde{a}_n(t)\widetilde{x}(t-\tau_n(t)) \tag{4.174}$$

This defines the complex lowpass channel model. This model will be revisited in Chapter 14 when we consider waveform channel models in detail. ■

4.5 Summary

This chapter dealt with signal and system theory based on lowpass complex envelope representations of bandpass signals and systems. The motivation for using complex envelope representations for bandpass signals and systems in simulations is computational efficiency. Through the use of the complex envelope, the number

of samples required to represent a bandpass signal is significantly reduced. The use of these techniques directly leads to a significant reduction in the time required to execute a given simulation. Therefore, in the work to follow we will attempt to model all bandpass signals and systems using lowpass complex models.

Two fundamental concepts were addressed in this chapter. The first concept was the development of models for signals and systems based on the lowpass complex envelope representation of bandpass signals and the lowpass equivalent impulse response of bandpass systems. The second concept dealt with the development of techniques for calculating the lowpass complex envelope for the system output given the lowpass complex envelope of the system input and the lowpass model for the system. We saw that complex envelope models for bandpass signals are typically specified in terms of $x_d(t)$ and $x_q(t)$, which are the real and imaginary components of the lowpass complex envelope $\widetilde{x}(t)$. Processing the complex envelope of the system input through the system model, defined by $h_d(t)$ and $h_q(t)$, usually involves four real convolutions. The computational burden associated with this operation can be reduced by a factor of 2 if the transfer function of the bandpass system exhibits conjugate symmetry about the reference frequency f_0. This reduction in computational burden arises from the fact that $h_q(t) = 0$ for the conjugate symmetry case. In many cases of practical interest $h_q(t)$, while not exactly equal to zero, is negligible compared to $h_d(t)$. A filter having a bandwidth much less than the center frequency is an example of a case in which $h_q(t)$ can be neglected.

While the emphasis in this chapter was on fixed (time-invariant) linear systems, many practical communication systems involve operations that are nonlinear, time-varying, or both. While these systems are much more complicated than the fixed linear systems considered here, as illustrated by the two examples presented in the preceding section, it is possible to develop simulation models, based on complex envelope representations, for these systems also. These more complicated systems are covered in Chapters 12 and 13.

4.6 Further Reading

The topic of signal and system analysis, based on complex envelope representations with an emphasis on applications to communications, can be found in a variety of books. The following are examples:

S. Haykin, *Communication Systems*, 3rd ed., New York: Wiley, 1994.

R. E. Ziemer and W. H. Tranter, *Principles of Communications: Systems, Modulation and Noise*, 5th ed., New York: Wiley, 2002.

The following two books present the material in this chapter from a simulation point of view:

M. C. Jeruchim, P. Balaban and K. S. Shanmugan, *Simulation of Communication Systems*, 2nd ed., New York: Plenum, 2000.

F. M. Gardner and J. D. Baker, *Simulation Techniques*, New York: Wiley, 1997.

4.7 References

1. A. J. Viterbi and J. K. Omura, *Principles of Digital Communication and Coding*, New York: McGraw-Hill, 1979.

2. W. T. Webb and L. Hanzo, *Modern Quadrature Amplitude Modulation*, New York: IEEE Press and London: Pentech Press, 1994.

3. R. E. Ziemer and W. H. Tranter, *Principles of Communications: Systems, Modulation and Noise*, 5th ed., New York: Wiley, 2002.

4.8 Problems

4.1 Another method for defining the lowpass complex envelope is through the use of the Hilbert transform. The Hilbert transform of a signal $x(t)$ is denoted $\widehat{x}(t)$ and is computed by passing $x(t)$ through a linear filter having the transfer function

$$H(f) = -j\,\mathrm{sgn}(f)$$

where

$$\mathrm{sgn}(f) = \begin{cases} 1, & f > 0 \\ -1, & f < 0 \end{cases}$$

so that

$$\widehat{X}(f) = -j\,\mathrm{sgn}(f)X(f)$$

The analytic signal $x_A(t)$ corresponding to the real signal $x(t)$ is the complex signal defined by

$$x_A(t) = x(t) + j\widehat{x}(t)$$

The lowpass complex envelope is then written

$$\widetilde{x}(t) = x_A(t)\exp(-j2\pi f_0 t)$$

where f_0 is usually taken as the center frequency of the bandpass signal $x(t)$.

(a) Assuming that a real bandpass signal has the spectrum defined by $X(f)$ in Figure 4.7, determine and accurately sketch the magnitude spectrum of the analytic signal. Label all amplitudes and frequencies of interest.

(b) Using the analytic signal found in (a), determine and accurately sketch the spectrum of the complex envelope. Compare this result with $\widetilde{X}(f)$ as shown in Figure 4.7.

4.2 A bandpass signal is defined by

$$x(t) = 6m(t)\cos[2\pi(100)t]$$

where

$$m(t) = 3\cos[2\pi(10)t] + 4\sin[2\pi(20)t]$$

(a) Plot the spectrum (both amplitude and phase) of $x(t)$.

(b) Using the technique outlined in Problem 1, determine and plot (both amplitude and phase) the spectrum of the analytic signal $x_A(t)$.

(c) Using the result of (b), determine and plot (both amplitude and phase) the spectrum of the analytic signal $\widetilde{x}(t)$.

4.3 An angle modulated signal is defined by

$$x(t) = 10\cos[2\pi(100)t + 2\sin(2\pi(10)t)]$$

(a) Determine $x_d(t)$ and $x_q(t)$ analytically and plot both $x_d(t)$ and $x_q(t)$.

(b) Using MATLAB and the fast Fourier transform, determine and plot $X(f)$. Plot both magnitude and phase.

(c) Using MATLAB, determine $\widetilde{X}(f)$ and plot the result (both magnitude and phase).

(d) Using MATLAB, determine $x_d(t)$ and $x_q(t)$ and plot the results.

(e) Compare the results of (d) with (a).

4.4 An FM signal is represented by

$$x_c(t) = A_c\sin\left(2\pi f_c t + k_f\int_0^t m(t)\,dt + \frac{\pi}{6}\right)$$

(a) Determine the expressions for $x_d(t)$ and $x_q(t)$.

(b) Assuming that the complex envelope is represented in the form

$$\widetilde{x}_c(t) = A(t)\exp(j\phi(t))$$

determine expressions for $A(t)$ and $\phi(t)$.

4.5 A single sideband (SSB) signal can be represented by

$$x_c(t) = A_c\left[m(t)\cos(2\pi f_c t) \pm \widehat{m}(t)\sin(2\pi f_c t)\right]$$

where the plus sign is used for lower sideband SSB, the minus sign is used for upper sideband SSB, and $\widehat{m}(t)$ is the Hilbert transform of the message signal $m(t)$.

(a) Determine the expressions for $x_d(t)$ and $x_q(t)$, in terms of $m(t)$, for both upper sideband SSB and lower sideband SSB.

(b) Determine expressions for $A(t)$ and $\phi(t)$ in terms of $m(t)$.

(c) Assuming that

$$m(t) = 2\cos(2\pi t) - \sin(4\pi t)$$

determine and plot $x_d(t)$, $x_q(t)$, $A(t)$, and $\phi(t)$. (Hint: A MATLAB program may be useful for developing the plots.)

4.6 In frequency-shift keyed (FSK) signaling, transmission of a binary 0 (space) or a binary 1 (mark) is accomplished using signals of two different frequencies. For example, assume that an FSK signal is given by

$$x(t) = A\sin\left[2\pi f_0 t - (-1)^k \pi f_\Delta t\right], \qquad 0 < t < T_b \tag{4.175}$$

where T_b is the bit period and f_Δ is the difference between the two frequencies in the FSK signal set. It follows that the two frequencies in the FSK signaling set are $f_0 + f_\Delta/2$ and $f_0 - f_\Delta/2$. Determine the complex envelope of $x(t)$ in both polar and rectangular form.

4.7 Binary phase-shift keying (PSK) signals can be defined by

$$x(t) = A\sin\left[2\pi f_c t - (-1)^k \phi\right], \quad 0 < t < T_b \tag{4.176}$$

where the signal defined by $k = 0$ is used for transmission of a binary 0 and the signal defined by $k = 1$ corresponds to transmission of a binary 1. As in the previous problem, T_b is the bit period. Determine the complex envelope of $x(t)$ in both polar and rectangular form.

4.8 The Fourier transform of a signal $x(t)$ is illustrated in Figure 4.17. Assume that $X(f)$ is real and positive for all f.

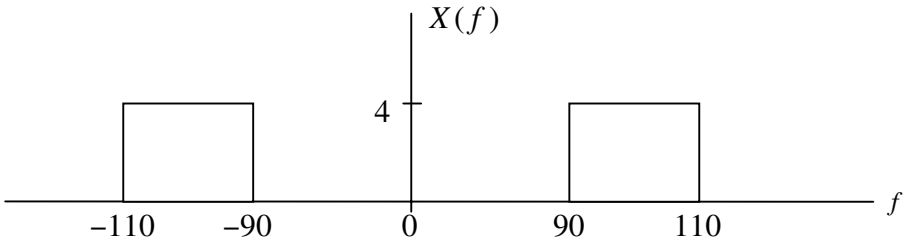


Figure 4.17 Signal spectrum for Problem 4.8.

(a) Determine and plot $X_d(f)$ and $x_d(t)$ for $f_0 = 100$.

(b) Determine and plot $X_d(f)$ and $x_d(t)$ for $f_0 = 95$.

(c) Determine and plot $X_d(f)$ and $x_d(t)$ for $f_0 = 90$.

(d) Compare and discuss the results.

4.9 Develop a MATLAB program to compute and plot $h_d(t)$ and $h_q(t)$ as defined by (4.137) and (4.138). Assume that $f_l = 180$, $f_u = 220$ and that f_0 takes on the following four values:

(a) $f_0 = 200$ Hz

(b) $f_0 = 190$ Hz

(c) $f_0 = 180$ Hz

(d) $f_0 = 160$ Hz

Compare the results. What do you observe from this comparison?

4.10 The Fourier transform of the complex envelope of a signal $x(t)$ is shown in Figure 4.18. Assume that $\widetilde{X}(f)$ is real and positive for all f. Plot accurately $X_d(f)$ and $X_q(f)$. Be sure to label all frequencies of interest and the amplitudes corresponding to these frequencies.

4.11 A second-order bandpass filter is defined by

$$H(s) = \frac{s\omega_b}{s^2 + s\omega_b + \omega_0^2}$$

where ω_0 is the geometric center frequency of the filter in radians/second and ω_b is the filter bandwidth in radians/second. Assume that the impulse response of the filter is defined by

$$h(t) = \mathrm{Re}\{2\widetilde{h}(t)\exp(j\omega_0 t)\}$$

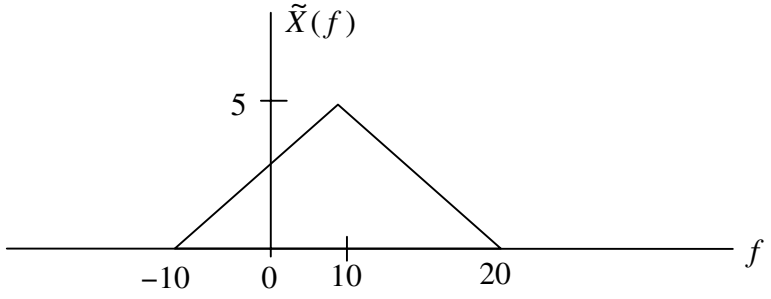


Figure 4.18 Complex envelope for Problem 4.10.

(a) Determine $h_d(t)$ and $h_q(t)$.

(b) Let $\omega_0 = 1$ and $\omega_b = 0.2$. Using the results of (a) plot $h_d(t)$ and $h_q(t)$.

(c) Repeat (b) for $\omega_b = 0.05$.

(d) Compare the results of (b) and (c). What do you conclude?

4.12 The input to a bandpass hard limiter is the AM signal

$$x(t) = 5(2 + \sin 2\pi t)\cos\left(20\pi t + \frac{\pi}{6}\right)$$

The output of the hard limiter is to be

$$y(t) = 7\cos\left(20\pi t + \frac{\pi}{6}\right)$$

(a) Determine $\widetilde{x}(t)$ and, using a suitable MATLAB program, plot $x_d(t)$ and $x_q(t)$.

(b) Develop a MATLAB program to implement the simulation model of the bandpass hard limiter.

(c) Using $\widetilde{x}(t)$ as the input to the simulation model, generate and plot $y_d(t)$, $y_q(t)$, and $|\widetilde{y}(t)|$.

(d) Using $y_d(t)$ and $y_q(t)$ as generated by the simulation, generate and plot $y(t)$. Compare the result with the analytical expression for $y(t)$.

(e) Develop analytical expressions for $x_d(t)$, $x_q(t)$, $y_d(t)$, and $y_q(t)$. Plot each of these and compare with the results obtained in (a) and (c).

4.9 Appendix A: MATLAB Program QAMDEMO

4.9.1 Main Program: c4_qamdemo.m

```
% File: c4_qamdemo.m
levelx = input('Number of D levels > ');
levely = input('Number of Q levels > ');
m = input('Number of symbols > ');
n = input('Number of samples per symbol > ');
bw = input('Filter bandwidth, 0<bw<1 > ');%
%
[xd,xq] = qam(levelx,levely,m,n);
%
[b,a] = butter(6,bw);               % determine filter coefficients
yd = filter(b,a,xd);                % filter direct coefficient
yq = filter(b,a,xq);                % filter quadrature coefficient
%
subplot(2,2,1)                      % first pane
plot(xd,xq,'o')                     % unfiltered scatterplot
a = 1.4;
maxd = max(xd); maxq = max(xq);
mind = min(xd); minq = min(xq);
axis([a*mind a*maxd a*minq a*maxq])
axis equal
xlabel('xd'); ylabel('xq')
%
subplot(2,2,2)                      % second pane
plot(yd,yq)                         % filtered scatterplot
axis equal;
xlabel('xd'); ylabel('xq');
%
sym = 30;                           % number of symbols in time plot
nsym = (0:sym*n)/n;                 % x axis vector for time plots
subplot(2,2,3)                      % third pane
plot(nsym(1:sym*n),yd(1:sym*n))     % filtered direct component
xlabel('symbol index');
ylabel('xd');
%
subplot(2,2,4)                      % fourth pane
plot(nsym(1:sym*n),yq(1:sym*n))     % filtered quadrature component
xlabel('symbol index');
ylabel('xq');
% End of script file.
```

4.9.2 Supporting Routines

qam.m

```
function [xd,xq] = qam(levelx,levely,m,n)
xd = mary(levelx,m,n);
xq = mary(levely,m,n);
% End of function file.
```

mary.m

```
function y= mary(levels,m,n)
% m = number of symbols
% n = samples per symbol
l = m*n;                              % Total sequence length
y = zeros(1,l-n+1);                   % Initalize output vector
lm1 = levels-1;
x=2*fix(levels*rand(1,m))-lm1;
for i = 1:m                           % Loop to generate info symbols
    k = (i-1)*n+1;
    y(k) = x(i);
end
y = conv(y,ones(1,n));                % Make each symbol n samples
% End of function file.
```

4.10 Appendix B: Proof of Input-Output Relationship

We now formally show that if $x(t)$ and $h(t)$ are defined as

$$x(t) = \mathrm{Re}\{\widetilde{x}(t)\exp(j2\pi f_0 t)\} \tag{4.177}$$

and

$$h(t) = \mathrm{Re}\{2\widetilde{h}(t)\exp(j2\pi f_0 t)\} \tag{4.178}$$

then

$$y(t) = \int_{-\infty}^{\infty} x(\lambda)h(t-\lambda)\,dy = \mathrm{Re}\{\widetilde{y}(t)\exp(j2\pi f_0 t)\} \tag{4.179}$$

where

$$\widetilde{y}(t) = \widetilde{x}(t) \circledast \widetilde{h}(t)$$

The proof of (4.179) is accomplished by substituting $x(t)$ and $h(t)$ in the integral and evaluating the result. Recognizing that the sum of a function and its complex conjugate is twice the real part of the function allows us to express $x(t)$ and $h(t)$ in the form

$$x(t) = \frac{1}{2}\widetilde{x}(t)\exp(j2\pi f_0 t) + \frac{1}{2}\widetilde{x}^*(t)\exp(-j2\pi f_0 t) \tag{4.180}$$

and

$$h(t) = \widetilde{h}(t)\exp(j2\pi f_0 t) + \widetilde{h}^*(t)\exp(-j2\pi f_0 t) \tag{4.181}$$

respectively. Substituting $x(t)$ and $h(t)$ into the convolution integral yields $y(t)$ as the sum of four integrals. We therefore write

$$y(t) = I_1 + I_2 + I_3 + I_4 \tag{4.182}$$

where

$$\begin{aligned} I_1 &= \frac{1}{2}\int_{-\infty}^{\infty} \widetilde{x}(\lambda)\exp(j2\pi f_0\lambda)\ \widetilde{h}^*(t-\lambda)\exp(-j2\pi f_0(t-\lambda))\ d\lambda \\ &= \frac{1}{2}\exp(-j2\pi f_0 t)\int_{-\infty}^{\infty} \widetilde{x}(\lambda)\widetilde{h}^*(t-\lambda)\exp(j4\pi f_0\lambda)\ d\lambda \end{aligned} \tag{4.183}$$

$$\begin{aligned} I_2 &= \frac{1}{2}\int_{-\infty}^{\infty} \widetilde{x}^*(\lambda)\exp(-j2\pi f_0\lambda)\ \widetilde{h}(t-\lambda)\exp(j2\pi f_0(t-\lambda))\ d\lambda \\ &= \frac{1}{2}\exp(j2\pi f_0 t)\int_{-\infty}^{\infty} \widetilde{x}^*(\lambda)\widetilde{h}(t-\lambda)\exp(-j4\pi f_0\lambda)\ d\lambda \end{aligned} \tag{4.184}$$

$$I_3 = \frac{1}{2}\int_{-\infty}^{\infty} \widetilde{x}(\lambda)\exp\left(j2\pi f_0\lambda\right)\ \widetilde{h}(t-\lambda)\exp\left(j2\pi f_0(t-\lambda)\right)\ d\lambda$$
$$= \frac{1}{2}\exp\left(j2\pi f_0 t\right)\int_{-\infty}^{\infty} \widetilde{x}(\lambda)\widetilde{h}(t-\lambda)\,d\lambda \tag{4.185}$$

and

$$I_4 = \frac{1}{2}\int_{-\infty}^{\infty} \widetilde{x}^*(\lambda)\exp\left(-j2\pi f_0\lambda\right)\ \widetilde{h}^*(t-\lambda)\exp\left(-j2\pi f_0(t-\lambda)\right)\ d\lambda$$
$$= \frac{1}{2}\exp\left(-j2\pi f_0 t\right)\int_{-\infty}^{\infty} \widetilde{x}^*(\lambda)\widetilde{h}^*(t-\lambda)\,d\lambda \tag{4.186}$$

Note that the integrands in both I_1 and I_2 are complex bandpass signals having a center frequency of $2f_0$. The envelope of these functions is slowly varying with respect to $2f_0$, since the bandwidth of $\widetilde{x}(t)$ and $\widetilde{h}(t)$ is assumed to be much less than $2f_0$. The integral therefore approximately cancels half-cycle by half-cycle. The approximation that I_1 and I_2 are negligible improves as f_0 increases. Thus, $\lim_{f_0\to\infty}(I_1 \text{and } I_2) = 0$.

Note also that I_3 and I_4 are complex conjugates. Thus:

$$y(t) = I_3 + I_4 = I_3 + I_3^* = 2\,\mathrm{Re}\{I_3\} \tag{4.187}$$

Substitution of (4.185) into (4.187) yields

$$y(t) = \mathrm{Re}\left\{\left[\int_{-\infty}^{\infty} \widetilde{x}(\lambda)\widetilde{h}(t-\lambda)\,d\lambda\right]\exp\left(j2\pi f_0 t\right)\right\} \tag{4.188}$$

which is equivalent to

$$y(t) = \mathrm{Re}\left\{\left[\widetilde{x}(t)\circledast\widetilde{y}(t)\right]\exp\left(j2\pi f_0 t\right)\right\} \tag{4.189}$$

Since by definition

$$y(t) = \mathrm{Re}\left\{\widetilde{y}(t)\exp\left(j2\pi f_0 t\right)\right\} \tag{4.190}$$

it follows that

$$\widetilde{y}(t) = \widetilde{x}(t)\circledast\widetilde{h}(t) \tag{4.191}$$

The preceding development shows that two bandpass signals may be convolved by convolving their complex envelopes and using (4.190) to generate the desired bandpass signal from its complex envelope. Note the asumption that f_0 is large.

Chapter 5

FILTER MODELS AND SIMULATION TECHNIQUES

This chapter focuses on the development of simulation models for filters. Filters are an important part of many of the subsystems that make up a communication system. Many of these filters are analog and must be mapped to a suitable digital equivalent for simulation purposes. A number of techniques are available for performing this mapping, all of which involve approximations and all of which induce errors in the simulation results. The technique used for generating a given filter for use in a given application will depend on a number of factors. Many of the most useful techniques for synthesizing and simulating filters will be explored in this chapter. Of particular interest are the limitations and the error sources inherent in these techniques.

Filters are by definition frequency selective and have impulse responses that may be either finite or infinite in duration. Since filters are frequency selective they induce memory and, as a consequence of this memory, computation of the filter output at a given time will require the use of past filter inputs and/or past filter outputs. Thus, filters require storage, and storing and retrieving sample values add significantly to the computational burden in a simulation program. This, in turn, adds to the simulation run time. Consequently, we seek architectures leading to algorithms that reduce the computational burden.

The purpose of this chapter is not to provide a detailed account of digital filter design techniques. A large number of textbooks exist on the subject of digital

filter design, and the basic techniques have been in use for many years. A partial list of these books appear in the Further Reading section at the end of this chapter. Our goal here is to review the most useful of these techniques and to present a variety of simple examples. The results of these examples provide us with a feel for the approximation errors that typically occur in simulation applications.

5.1 Introduction

A number of the techniques for developing digital filters are illustrated in Figure 5.1. The classical synthesis techniques for digital filters are often based on analog prototypes. In this case, development of the digital filter required in the simulation model starts with the s (Laplace) domain transfer function of the analog filter for which we seek the digital equivalent. The problem is then reduced to finding a digital filter that is equivalent, in some appropriate sense, to the analog prototype. There are many ways to define this equivalence. The fundamental methods are to base the measurement of equivalence on either a time-domain criterion or on a frequency-domain criterion. Although we usually think of filtering in terms of frequency selectivity characteristics, which are defined in the frequency domain, time-domain criteria are frequently used and result in efficient and useful filter

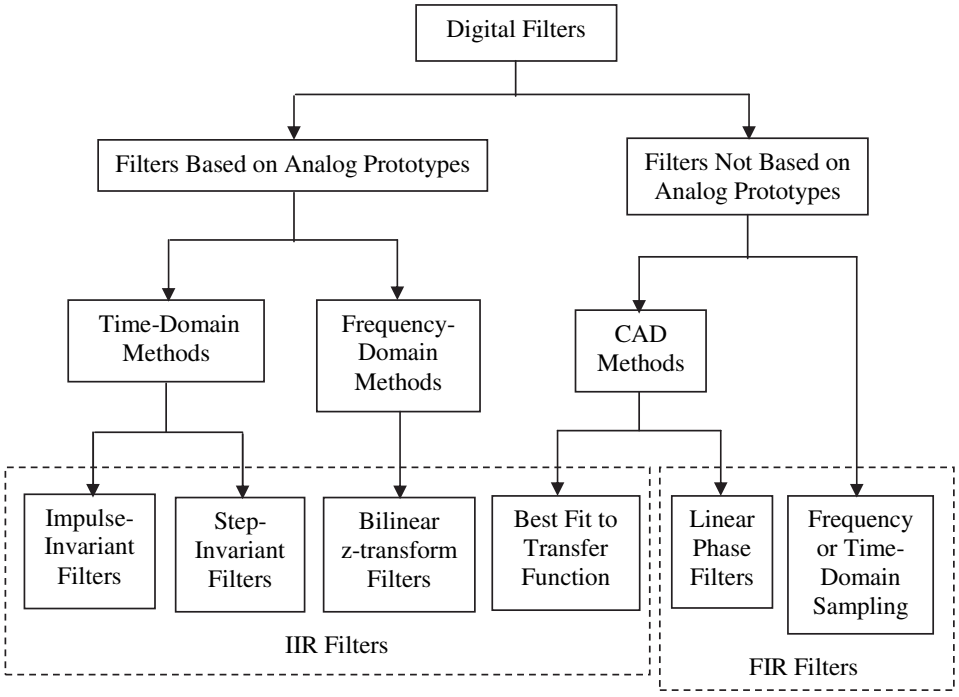


Figure 5.1 Digital filter classifications.

designs. A time-domain criterion is applied by requiring that the output of the digital filter match the sampled output of the analog prototype upon which the design is based. The two fundamental synthesis techniques based on time-domain criteria are the impulse-invariant digital filter and the step-invariant digital filter. The impulse-invariant digital filter is a design in which the impulse response of the digital filter matches the sampled impulse response of the analog prototype. For the step-invariant digital filter, the step response of the digital filter matches the sampled step response of the underlying analog prototype. Many other designs are possible using "test signals" other than impulses and unit steps. We will see that if the analog prototype and the resulting digital filters are equivalent in the time domain they are also *approximately* equivalent in the frequency domain, at least for values of frequency that are small compared to the sampling frequency.[1] This will be illustrated in the examples that follow.

Perhaps the most popular method for mapping an analog prototype to a digital equivalent is through application of the bilinear z-transform. The bilinear z-transform synthesis method, which is strictly an algebraic technique, allows one to match points in the frequency response so that both the analog prototype and the resulting digital filter have identical frequency responses (amplitude and phase) at certain designated values of frequency. In addition, the bilinear z-transform method eliminates aliasing errors at the expense of a nonlinear frequency warping. This technique will be illustrated in an example to follow. The bilinear z-transform filter is widely used in simulation applications.

Synthesis techniques based on analog prototypes result in IIR (infinite-duration impulse response) digital filters. This is to be expected, since analog filters have impulse responses that asymptotically approach zero but, strictly speaking, have an impulse response that is infinite in duration. The impulse response produced by one of the standard IIR design methods may be truncated in order to generate an FIR (finite-duration impulse response) digital filter for use in a simulation program. The error produced by this truncation can be reduced to an acceptable level by including a sufficient number of terms in the resulting FIR impulse response.

An important attribute of digital filters is that digital filters can be developed that have no analog counterpart. The most important filters falling into this category are filters that allow a given amplitude response to be approximated while maintaining a perfectly linear phase response. These are FIR filters and are implemented with transversal delay line structures. A number of design techniques exist for these filters. The most fundamental method is to expand the desired amplitude response, which is periodic in the sampling frequency, in a Fourier series. The resulting Fourier coefficients define the impulse response of the digital filter [1]. The FFT can be used to perform this operation. This is an example of frequency sampling, since the desired frequency response is "sampled" at various points in

[1] This statement might at first seem rather strange. Because of the one-to-one correspondence between a continuous-time signal and its Fourier transform, equivalence in the time domain implies equivalence in the frequency domain. In digital filters another parameter, namely, the sampling frequency, is present. The underlying sampling process gives rise to aliasing errors. The result is that sampled impulse or step responses can yield filters with different frequency responses depending on the sampling frequency.

frequency. The inverse FFT of these frequency samples gives the impulse response of the filter. Convolving the filter input with the impulse response implements the simulation model of the filter and generates the filter output.

A number of computer-aided design (CAD) techniques exist for designing digital filters. Two of these techniques are explored later in this chapter. The first technique leads to an IIR filter, and the second technique leads to a linear-phase FIR filter.

5.2 IIR and FIR Filters

As implied in the preceding discussion, digital filters are usually classified according to the impulse response duration (IIR or FIR). Closely tied to the impulse response classification is the implementation or structure, which will be discussed in the following section. We now take a look at various filter models.

5.2.1 IIR Filters

A linear digital signal processor (digital filter) computes the current output sample $y[n]$ as a weighted sum of N past output samples, $y[n-k]$, $1 \leq k \leq N$, the current input sample, $x[n]$, and N past input samples, $x[n-k]$, $1 \leq k \leq N$. In other words, the algorithm for computing the current output in terms of previous inputs and outputs is

$$y[n] = \sum_{k=0}^{N} b_k x[n-k] - \sum_{k=1}^{N} a_k y[n-k] \tag{5.1}$$

We will see in the following section that efficient algorithms exist for implementing the computation defined by (5.1) within a simulation program.

If any of the weights in (5.1), b_k or a_k, $k \geq 1$, have nonzero value, the processor has memory and is therefore frequency selective and we refer to it as a filter. For a time-varying system, one or more of the weights will be a function of the index n.[2] The transfer function $H(z)$ results by taking the z-transform of both sides of (5.1). This is accomplished by recalling that the z-transform is a linear operator, so that the transform of a sum is equal to the sum of transforms, and also recalling that a delay of k sample periods is equivalent to multiplication by z^{-k}. This yields

$$Y(z)\left[1 + \sum_{k=1}^{N} a_k z^{-k}\right] = X(z) \sum_{k=0}^{N} b_k z^{-k} \tag{5.2}$$

which gives the transfer function

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{N} b_k z^{-k}}{1 + \sum_{k=1}^{N} a_k z^{-k}} \tag{5.3}$$

[2]For now we will assume time invariance. Time-varying systems will be the subject of Chapter 13.

This is the general form of the transfer function for a linear, time-invariant, filter.

In the applications considered here we will usually have interest in the impulse response or the frequency response of the filter under study. The impulse response, denoted $h[n]$, is the inverse z-transform of the transfer function $H(z)$. The frequency response is found by substituting $\exp(j2\pi fT)$ for z in the transfer function. In other words, the frequency response is

$$H(z)|_{z=\exp(j2\pi fT)} = H(\exp(j2\pi fT)) \tag{5.4}$$

The impulse response of the digital filter, denoted $h[n]$, is found by letting $x[n] = \delta[n]$, where

$$\delta[n] = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases} \tag{5.5}$$

in (5.1). Due to the recursive nature of (5.1) ($y[n]$ is a function of $y[n-1]$) the impulse response $h[n]$ will, in general, have infinite duration and we therefore refer to the filter as an IIR filter. Note that the impulse response, $h[n]$, is a discrete function of the index n while the frequency response is a continuous function of the continuous variable f.

5.2.2 FIR Filters

An FIR digital filter results if $a_k = 0$, $k \geq 1$ in (5.1) or, equivalently, in (5.3). Thus, the transfer function of an FIR digital filter is

$$H(z) = \sum_{k=0}^{N} b_k z^{-k} \tag{5.6}$$

and the corresponding impulse response is

$$h[n] = \sum_{k=0}^{N} b_k \delta[n-k] \tag{5.7}$$

which is nonzero only for n in the range $0 \leq n \leq N$. Thus, the impulse response has at most $N+1$ nonzero terms and is therefore finite in duration. The algorithm for generating the filter output sequence $y[n]$ from the input sequence $x[n]$ is the discrete convolution

$$y[n] = \sum_{k=0}^{N} b_k x[n-k] = \sum_{k=0}^{N} h[k]x[n-k] \tag{5.8}$$

which follows directly from (5.1) with $a_k = 0$ for $k > 0$.

5.2.3 Synthesis and Simulation

It should be pointed out that use of a filter in a simulation program involves two very distinct operations. The first operation is *synthesis*. In the synthesis operation

the filtering requirements are specified and the filter transfer function, $H(z)$, that meets these requirements is determined. This establishes the simulation model. The result of the synthesis operation is usually expressed as two vectors, one vector containing the denominator coefficients, a_k, and the other vector containing the numerator coefficients, b_k. Together, these two vectors define the transfer function (5.3) and the algorithm for generating the filter output given the filter input. The computational burden associated with the synthesis operation is not usually significant, even if complex algorithms are used, since filter synthesis is only performed once and, therefore, takes place outside of the main simulation loop. The second operation involves the computation of filter outputs at each simulation time step, that is, at each tick of the simulation clock. This operation may have to be repeated millions, or even billions, of times in a Monte Carlo simulation program. Therefore, the computational burden associated with this operation must be minimized if reasonable simulation run times are to result. The transposed filter structure considered in the following section addresses this important concern.

5.3 IIR and FIR Filter Implementations

We now briefly examine the manner in which digital filters are realized in a simulation program. As discussed at the conclusion of the preceding section, the goal is to minimize the computational burden so that the time required to execute a simulation is minimized.

5.3.1 Direct Form II and Transposed Direct Form II Implementations

An efficient technique for implementing IIR digital filters in a simulation program is the transposed Direct Form II architecture.[3] The signal flow graphs of the transposed Direct Form II architecture, and the Direct Form II architecture from which it is derived, are both illustrated in Figure 5.2. We start with the Direct Form II structure because it is a straightforward implementation of the defining difference equation expressed by (5.1). The student should take time to verify that both structures illustrated in Figure 5.2 satisfy (5.1) and (5.3).

The transposed Direct Form II structure is the most commonly used structure in the simulation of filters because of execution speed. It is easily derived from the Direct Form II structure. The rules for generating a transposed filter structure from a given filter structure are simple:

1. Redraw the original (Direct Form II) signal flow graph maintaining the architecture (all links maintain their relative positions).

2. Reverse the direction of signal flow in each link.

[3]We assume that the simulation will be executed on a floating-point machine so that coefficient quantization and other finite word length effects are not important. For fixed-point applications, it is often necessary to decompose an N^{th} order filter into first-order or second-order parallel or cascade structures in order to reduce the effects of coefficient quantization [2].

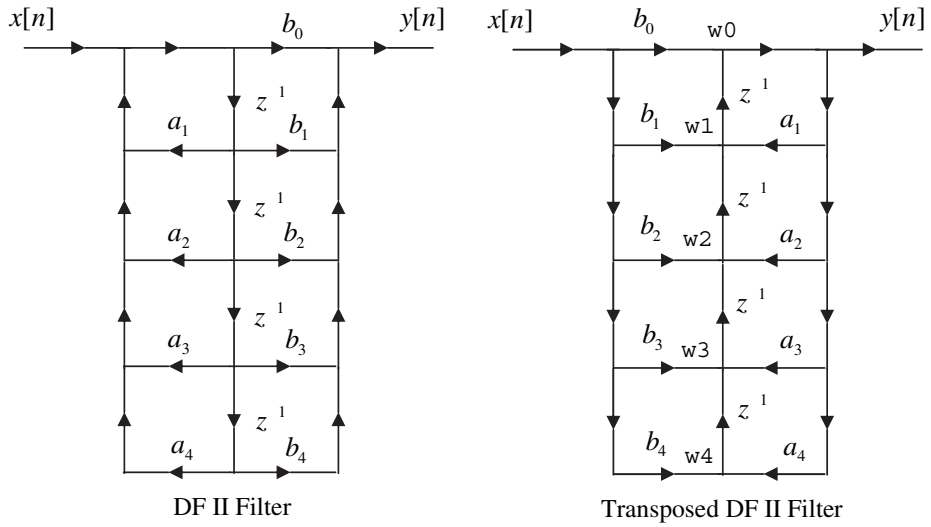


Figure 5.2 Implementation structures for IIR filters.

3. Assign to the new link the same operation (multiplication by a constant, delay, etc.) that was assigned to the original link.

4. If desired, flip (left to right) the newly constructed signal flow graph so that the direction of the input/output signal flow is consistent with the original signal flow graph. (Note: Signal flow is usually from left to right.)

The new signal flow graph, known as the transposed Direct Form II (DF II) structure, will have the same transfer function as the original signal flow graph [2].

To see the attraction of the transposed DF II implementation, the fourth-order transposed DF II filter shown in Figure 5.2 is considered. (The extension to higher-order filters is obvious.) We are given the input sample $x[n]$ and are required to compute the output $y[n]$. The first step is to compute the state variables $w_j[n]$ for $j = 0, 1, \cdots 4$. Note that for a fourth-order filter there are five state variables in our formulation. The five state variables are given by the computation

$$w_0[n] = w_1[n-1] + b_0 x[n] \tag{5.9}$$
$$w_1[n] = -a_1 w_0[n] + w_2[n-1] + b_1 x[n] \tag{5.10}$$
$$w_2[n] = -a_2 w_0[n] + w_3[n-1] + b_2 x[n] \tag{5.11}$$
$$w_3[n] = -a_3 w_0[n] + w_4[n-1] + b_3 x[n] \tag{5.12}$$
$$w_4[n] = -a_4 w_0[n] + b_4 x[n] \tag{5.13}$$

We now consider the computational advantage of the transposed structure defined by the preceding equations.

The state variables as expressed by (5.9) through (5.13) may be computed "in sequence." For example note that, given the input $x[n]$, $w_0[n]$ can be computed, since $w_1[n-1]$ is known from the previous pass through the simulation loop. Once $w_0[n]$ is known, $w_1[n]$ can be computed. Continuing the computation we see that $w_j[n]$ depends only on $w_k[n]$ where $k < j$. Thus the computation of each state variable only requires knowledge of previously computed quantities. As a matter of fact, the MATLAB code for implementing (5.9) through (5.13) within a simulation loop is

```
w1 = 0; w2 = 0; w3 = 0; w4 = 0;        % initialize state variables
for k = 1:npts                         % beginning of simulation loop

    ⋮

w0 = w1 + b0*x;
w1 = -a1*w0 + w2 + b1*x;
w2 = -a2*w0 + w3 + b2*x;
w3 = -a3*w0 + w4 + b3*x;
w4 = -a4*w0 + b4*x;
y = w0;

    ⋮

end                                    % end of simulation loop}
```

In the preceding code `x` and `y` represent the current filter input, $x[k]$, and output, $y[k]$, respectively. Note that the one-sample delay operations illustrated in (5.9) through (5.13) are implemented through the order in which the state variables are computed. No storage and retrieval is necessary. As a result, the algorithm is fast compared to algorithms based on other structures, which is why the MATLAB routine `filter` is based on the transposed DF II structure. Also note that the state variables `w1`, `w2`, `w3`, and `w4` must be initialized prior to entering the simulation loop the first time. This initialization induces a transient response into the filter output. Typically the simulation loop must be executed a number of times before useful data can be collected from the simulation. This time is often referred to as the "settle time" and is several times the reciprocal of the filter bandwidth. A vectorized form of these calculations will be discussed shortly (see Example 5.1).

State equations for a digital filter are usually expressed in matrix form. State variable matrix formulations are most valuable when the filter has multiple inputs and outputs. Here, however, we will have interest only in filters having a single input $x[n]$ and a single output $y[n]$. The general expression for a single-input filter is [2]

$$\mathbf{W}[n] = \mathbf{F}_c\mathbf{W}[n] + \mathbf{F}_d\mathbf{W}[n-1] + \mathbf{B}x[n] \tag{5.14}$$

where $\mathbf{W}[n]$ and $\mathbf{W}[n-1]$ are $k \times 1$ column vectors representing the current and previous state variables, respectively, $\mathbf{F}_c$ and $\mathbf{F}_d$ are $k \times k$ coefficient matrices and $\mathbf{B}$

is a $k \times 1$ column vector coupling the input $x[n]$ to the state variables. The output equation for a single output $y[n]$ is

$$y[n] = \mathbf{C}\mathbf{W}[n] \tag{5.15}$$

where $\mathbf{C}$ is a $1 \times k$ row vector.

Equations (5.9) through (5.13) can be placed in the matrix form

$$\begin{bmatrix} w_0[n] \\ w_1[n] \\ w_2[n] \\ w_3[n] \\ w_4[n] \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ -a_1 & 0 & 0 & 0 & 0 \\ -a_2 & 0 & 0 & 0 & 0 \\ -a_3 & 0 & 0 & 0 & 0 \\ -a_4 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} w_0[n] \\ w_1[n] \\ w_2[n] \\ w_3[n] \\ w_4[n] \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} w_0[n-1] \\ w_1[n-1] \\ w_2[n-1] \\ w_3[n-1] \\ w_4[n-1] \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} x[n] \tag{5.16}$$

A moments reflection shows that the states can be computed "in sequence" because the matrix $\mathbf{F}_c$ relating $w_j[n]$ and $w_k[n]$ in (5.16) has all zeros on and above the main diagonal. Signal-flow graphs having this property are known as computable graphs [2]. The term *computable* does not mean that this form is required in order for one to be able to calculate the output given the input. This is easily seen, since the upper triangular structure can be destroyed by simply relabeling the states. Rather, the term *computable* simply means that the states may be computed in turn as shown in (5.9) through (5.13). It can be shown that a necessary and sufficient condition for a computable form to exist is that there be no closed paths without at least one delay, that is, z^{-1}, element in the path [2].

One of the conveniences of MATLAB is that the filter coefficients a_k and b_k can be computed with ease for a vast number of different analog prototypes. Although these filter coefficients are typically generated for use in the MATLAB routine, `filter`, which is intended for block processing, IIR filters may be simulated on a sample-by-sample basis using the simple three-line MATLAB program

```
out = b(1)*in + sreg(1,1);          % compute filter output
sreg = in*b - out*a + sreg;         % update shift register contents
sreg = [sreg(1,2:(order+1)),0];     % cycle shift register
```

where `a` and `b` are determined outside of the simulation loop. The parameter `sreg` in the preceding code represents the shift register of length `order+1` where `order` is the order of the filter. An advantage of the approach illustrated here is that the MATLAB filter synthesis routines, such as `butter`, `cheby1`, and `elliptic`, can be used to compute the filter coefficient vectors **a** and **b**. Care must be used to ensure that the vectors holding the numerator and denominator coefficients have

the same length. If the two vectors are not of equal length, the shorter vector can be zero-padded to the length of the longer sequence.

Example 5.1. In order to illustrate the preceding technique, consider the following MATLAB program, which determines the impulse response of a fourth-order Butterworth filter using both block and serial (sample-by-sample) processing:

```
% File:  c5_filterex1.m
n = 40;                         % number of samples
order = 4;                      % filter order
[b,a] = butter(order,0.1);      % prototype
%
% The following segment is the block processing implementation.
%
in1 = [1,zeros(1,n-1)];         % input vector
out1 = filter(b,a,in1);         % output vector
%
% The following segment is the sample-by-sample implementation.
%
sreg = zeros(1,order+1);        % initialize shift register
for k=1:n
    if k==1
        in=1;             % impulse input
    else
        in=0;
    end
    out = b(1)*in + sreg(1,1);        % determine output
    sreg = in*b - out*a + sreg;       % update register
    sreg = [sreg(1,2:(order+1)),0];   % shift
    out2(k) = out;                    % create output vector
end
%
subplot(2,1,1)
index = 0:n-1;
stem(index,out1)
xlabel('Sample Index')
ylabel('Block Processing')
subplot(2,1,2)
stem(index,out2)
xlabel('Sample Index')
ylabel('Serial Processing')
% End of script file.
```

The result of executing this program is illustrated in Figure 5.3. We see that both the block-processing and the serial-processing technique produce identical results. ■

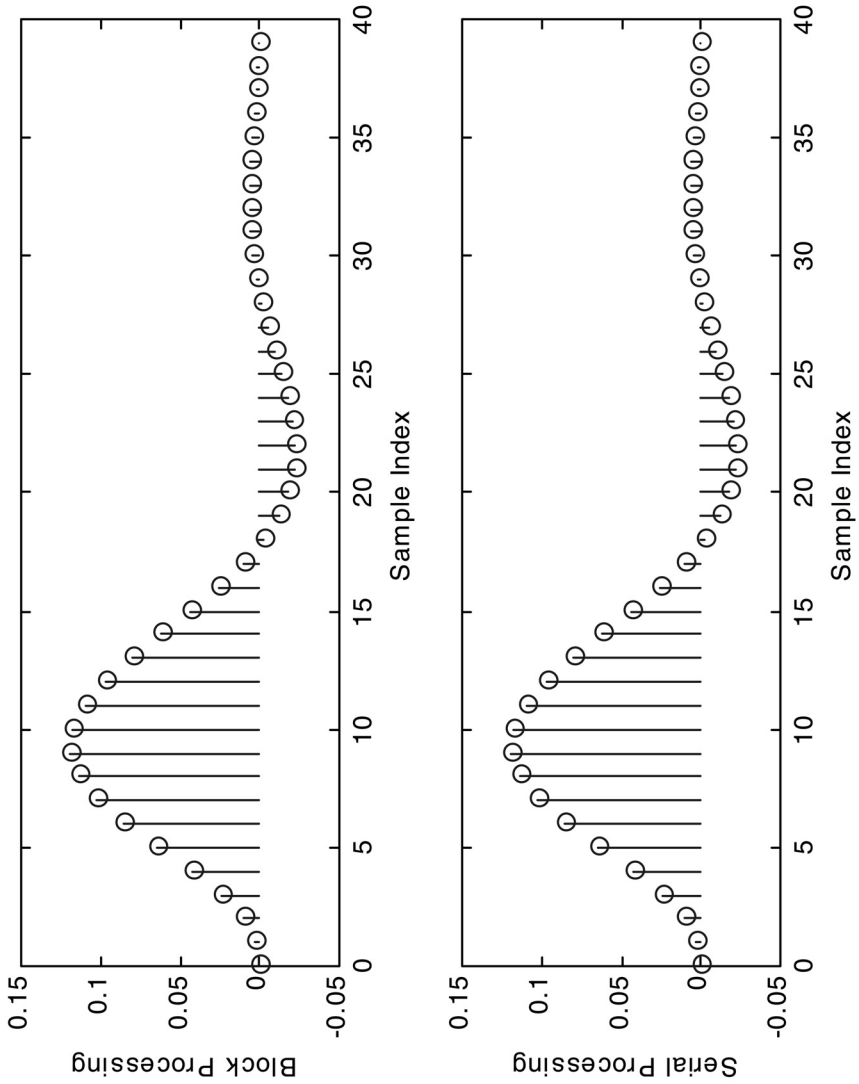


Figure 5.3 Comparison of block and serial processing.

5.3.2 FIR Filter Implementation

The implementation strategy for FIR digital filters follows immediately from Figure 5.2 by letting $a_k = 0$, $k \geq 1$. The resulting two structures are evident and are illustrated in Figure 5.4. These are referred to as tapped delay line or transversal delay line (TDL) structures, since they are implemented as delay lines (cascaded z^{-1} elements) with taps for the multiplication by the weights b_k.

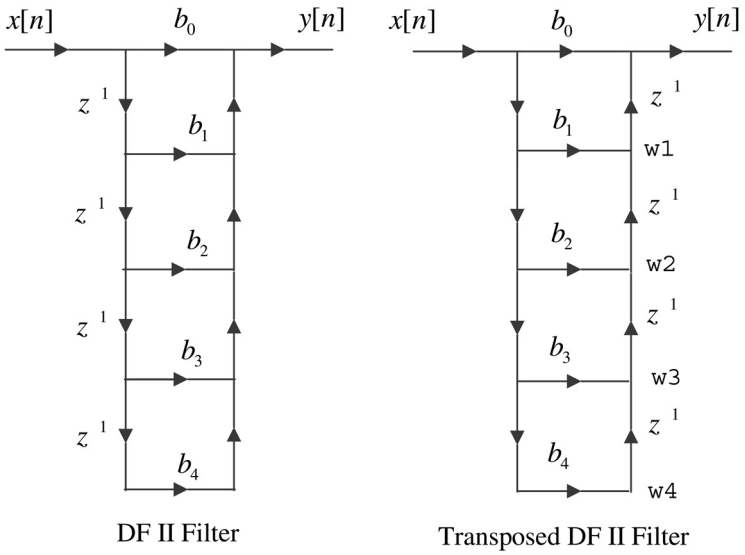


Figure 5.4 FIR filter structures.

5.4 IIR Filters: Synthesis Techniques and Filter Characteristics

IIR (infinite-duration impulse response) filters are typically designed from analog prototypes (Butterworth, Chebyshev, Elliptic, etc.) and are implemented using recursive structures. Modern communication systems do not rely on analog prototypes to the extent that was the case several decades ago, except possibly in the RF stages of the system. Many modern systems, such as the software radio, make extensive use of DSP techniques for implementing the system building blocks (including filters). In these DSP-based systems the problem of developing digital equivalents for analog prototypes does not arise, since the filters used in the physical hardware are already in digital form. As a result, the classical techniques for digital filter synthesis, those using an analog prototype as a starting point, are fading in importance. However, designs from analog prototypes are still used in many applications and are therefore worthy of consideration. Several computer-aided design techniques have been developed for IIR filters, one of which is considered here. Since the goal is to illustrate basic techniques, and the errors induced in a simulation through the use of these techniques, only very simple applications of those techniques will be considered. More complicated examples are provided in textbooks devoted to digital filter design, a few of which are cited at the end of this chapter.

5.4.1 Impulse-Invariant Filters

Assume that the unit pulse response of a digital filter is, except for an amplitude scaling, equal to the sampled impulse response of an analog filter. For this case the digital filter is known as the impulse-invariant realization of the analog filter. In other words, the transfer function of the digital filter, $H(z)$, is defined by [1]

$$H(z) = T\mathcal{Z}\left\{\mathcal{L}^{-1}\left[H_a(s)\right]|_{t=nT}\right\} \tag{5.17}$$

where $\mathcal{L}^{-1}(\cdot)$ denotes the inverse Laplace transform, $\mathcal{Z}(\cdot)$ denotes the z-transform, $H_a(s)$ denotes the transfer function of the analog filter, and T, which is amplitude scaling, denotes the sampling period.

In order to illustrate the impulse-invariance technique we consider the simplest possible analog prototype, which is

$$H_a(s) = \frac{a}{s+a} \tag{5.18}$$

Note that this is a first-order lowpass filter with a dc ($f = 0$) gain of one.

Example 5.2. From (5.17) the first step is to take the inverse Laplace transform of $H_a(s)$. This gives the impulse response

$$\mathcal{L}^{-1}\left[H_a(s)\right] = h_a(t) = a\exp(-at)u(t) \tag{5.19}$$

in which $u(t)$ is the unit step and is used to denote that the filter is causal. Prior to taking the z-transform the impulse response must be sampled at the sampling instants $t = nT$. This gives

$$\mathcal{L}^{-1}\left[H_a(s)\right]|_{t=nT} = h_a[n] = a\exp(-anT)u[n] \tag{5.20}$$

The z-transform of $h_a[n]$ is

$$\mathcal{Z}\left\{\mathcal{L}^{-1}\left[H_a(s)\right]|_{t=nT}\right\} = \sum_{n=0}^{\infty} a\left[\exp(-aT)\right]^n z^{-n} \tag{5.21}$$

Performing the sum and multiplying by the sampling period gives

$$H(z) = \frac{aT}{1-\exp(-aT)z^{-1}} \tag{5.22}$$

which is the impulse invariant equivalent of (5.18). The frequency-domain behavior of this filter will be explored in Example 5.5. ■

5.4.2 Step-Invariant Filters

The step-invariant filter is frequently used in simulations (we will see the reason for this in a following section). For the step-invariant filter, the unit pulse response of the digital filter is equivalent to the sampled step response of the analog filter. Since the step response of the analog filter, expressed in the frequency domain, is $H_a(s)/s$, the transfer function of a step-invariant filter is

$$\frac{1}{1-z^{-1}}H(z) = \mathcal{Z}\left\{\mathcal{L}^{-1}\left[\frac{1}{s}H_a(s)\right]|_{t=nT}\right\} \tag{5.23}$$

Note that the left-hand side of (5.23) is the response of a digital filter, having transfer function $H(z)$, to a sampled unit step. The right-hand side of (5.23) represents the z-transform of the sampled unit step response of the prototype analog filter.

Example 5.3. In this example we again use (5.18) as the analog prototype. The step response of the analog filter is

$$\frac{1}{s}H_a(s) = \frac{a}{s(s+a)} \tag{5.24}$$

Partial fraction expansion is then used to obtain

$$\frac{1}{s}H_a(s) = \frac{1}{s} - \frac{1}{s+a} \tag{5.25}$$

Taking the inverse Laplace transform and sampling at $t = nT$ gives

$$\mathcal{L}^{-1}\left[\frac{1}{s}H_a(s)\right]|_{t=nT} = u[n] - \exp(-anT)u[n] \tag{5.26}$$

and taking the z-transform gives

$$\frac{1}{1-z^{-1}}H(z) = \sum_{n=0}^{\infty} \{1 - [\exp(-aT)]^n\} z^{-n} \tag{5.27}$$

or

$$\frac{1}{1-z^{-1}}H(z) = \frac{1}{1-z^{-1}} - \frac{1}{1-\exp(-aT)z^{-1}} \tag{5.28}$$

The transfer function of the step-invariant digital filter is

$$H(z) = 1 - \frac{1-z^{-1}}{1-\exp(-aT)z^{-1}} \tag{5.29}$$

Writing the preceding in standard form as defined by (5.3) yields

$$H(z) = \frac{[1-\exp(-aT)]\, z^{-1}}{1-\exp(-aT)z^{-1}} \tag{5.30}$$

The frequency domain behavior of this filter will be explored in Example 5.5. The result will be compared with the impulse-invariant result. ■

5.4.3 Bilinear z-Transform Filters

The bilinear z-transform filter is perhaps the most commonly used filter in simulation programs. There are a number of reasons for this. First, as we will see in the following section, the synthesis of digital filters using the bilinear z-transform technique is very straightforward and is entirely algebraic. In addition, we will see that the bilinear z-transform filter does not exhibit aliasing.

Synthesis Technique

The bilinear z-transform synthesis technique maps an analog prototype, $H_a(s)$, to a digital filter using a simple algebraic transformation. Specifically, the transfer function of the digital filter, $H(z)$, is defined by [1, 2]

$$H(z) = H_a(s)|_{s=C(1-z^{-1})/(1+z^{-1})} \tag{5.31}$$

where C is a constant. The technique used to appropriately determine C will be discussed shortly. Since $H_a(s)$ is typically defined as a ratio of polynomials in s, the algebraic mapping defined by (5.31) gives $H(z)$ in the desired form of a ratio of polynomials in z^{-1}. Implementation of the filter follows as discussed in the previous section.

To show that the transfer function of the resulting digital filter has desirable properties, we substitute $\exp(s_dT)$, where s_d is the complex frequency variable for the digital filter, for z in the definition of the bilinear z-transform. This gives

$$s = \frac{1-\exp(-s_dT)}{1+\exp(-s_dT)} \tag{5.32}$$

Substitution of $j2\pi f_a$ for s and $j2\pi f_d$ for s_d in the preceding expression gives

$$j2\pi f_a = C\,\frac{\exp(j\pi f_d T) - \exp(-j\pi f_d T)}{\exp(j\pi f_d T) + \exp(-j\pi f_d T)} \tag{5.33}$$

which is

$$2\pi f_a = C\tan(\pi f_d T) \tag{5.34}$$

or

$$C = 2\pi f_a \cot(\pi f_d T) \tag{5.35}$$

This defines the relationship between the frequency responses of the prototype analog filter and the resulting digital filter. Adjusting C in this manner so that $f_c = f_d$ for an arbitrary sampling frequency is called prewarping.

The transformation relating f_a and f_d is clearly nonlinear. In developing simulation models for communications systems, as well as for many other types of systems, it is usually desirable to maintain the shape of the transfer function, in both amplitude and phase, as the analog filter is mapped to its digital equivalent. Maintaining the shape of the transfer function requires that the transformation relating f_d and f_a be, to the extent possible, linear. Since $\tan(x) \approx x$ for small x an approximately linear transformation requires that $\pi f_d T \ll 1$ or

$$f_d \ll \frac{1}{\pi} f_s \tag{5.36}$$

where f_d ranges over the frequencies of interest and f_s is the sampling frequency. The value of C that results in an approximately linear transformation with $f_d \approx f_a$ is easy to determine. Linearity requires that

$$2\pi f_a \approx C(\pi f_d T) \tag{5.37}$$

If we also require that $f_a \approx f_d$ we have

$$C = \frac{2}{T} = 2f_s \tag{5.38}$$

Unfortunately, maintaining $f_a \approx f_d$ for all frequencies of interest, such as the frequency range of a filter passband, typically requires an excessively high sampling frequency. This leads to simulation run times that are often impractical.

Example 5.4. In order to illustrate the bilinear z-transform synthesis method we once again turn to the simple first-order analog prototype defined by (5.18). By definition, the bilinear z-transform filter has the transfer function

$$H(z) = \frac{a}{s+a}\bigg| s = C\frac{1-z^{-1}}{1+z^{-1}} \tag{5.39}$$

This gives

$$H(z) = \frac{a(1+z^{-1})}{C(1-z^{-1}) + a(1+z^{-1})} \tag{5.40}$$

which, in standard form, is

$$H(z) = \frac{\frac{a}{C+a} + \frac{a}{C+a}z^{-1}}{1 - \frac{C-a}{C+a}z^{-1}} \tag{5.41}$$

The frequency-domain behavior of this filter will be explored in the following example. ■

Example 5.5. In this example the responses of the three filters designed in the last three examples will be compared. Assume that the sampling frequency is 100 Hz ($T = 0.01$) and that the parameter a is $2\pi(10)$ so that the 3 dB frequency of the analog filter is 10 Hz. With these assumed values we have

$$aT = 0.628319 \tag{5.42}$$

and

$$\exp(-aT) = 0.533488 \tag{5.43}$$

Thus, from (5.22), the transfer function of the impulse-invariant filter is

$$H_{ii}(z) = \frac{0.628319}{1 - 0.533488z^{-1}} \tag{5.44}$$

For the step-invariant filter we have, from (5.30):

$$H_{si}(z) = \frac{(1 - 0.533488)\, z^{-1}}{1 - 0.533488z^{-1}} \tag{5.45}$$

Two versions of the bilinear z-transform filter will be given. First, if no frequency prewarping is used, so that the amplitude responses of the analog prototype filter and the digital filter agree closely for low frequencies

$$C = \frac{2}{T} = 200 \tag{5.46}$$

Substituting the values of C and a into (5.41) gives

$$H_{bl}(z) = \frac{0.239057 + 0.239057z^{-1}}{1 - 0.5218861z^{-1}} \tag{5.47}$$

Now suppose that the value of C is chosen so that the frequency responses are matched at the 3 dB frequency of the analog filter. For this case

$$f_a = f_d = \frac{a}{2\pi} \tag{5.48}$$

so that, from (5.35), the value of C is

$$C = a\cot\left(\frac{aT}{2}\right) \tag{5.49}$$

which gives

$$C = 40\pi\cot(0.2\pi) = 172.961 \tag{5.50}$$

Using this value of C in (5.41) gives

$$H_{bl}(z) = \frac{0.420808 + 0.420808z^{-1}}{1 - 0.158384z^{-1}} \tag{5.51}$$

The student should compare the responses defined by (5.47) and (5.51). (See Problem 5.2.)

The amplitude responses of the impulse invariant, the step invariant, and the first bilinear z-transform filter, described by (5.47), are generated using the following MATLAB code:

```
% File:  c5_threefilters.m
T = 0.01;
f = 0:0.1:50;
z = exp(-i*2*pi*f*T);                          % see (5.4)
a0 = 0.239057; a1=0.239057; b1=0.521886;    % bilinear z-transform
num = a0+a1*z;
den = 1-b1*z;
ampx = abs(num./den);
a0 = 0.628319; b1 = 0.533488;                % impulse invariant
num = a0;
den = 1-b1*z;
ampy = abs(num./den);
a0 = 1.0; a1 = 0.533488; b1 = 0.533488;     % step invriant
num = (a0-a1)*z;
den = 1-b1*z;
ampz = abs(num./den);
plot(f,ampx,f,ampy,f,ampz)
xlabel('Frequency - Hz')
ylabel('Amplitude Response')
% End of script file.
```

The results are illustrated in Figure 5.5. Note that both the step-invariant filter and the bilinear z-transform filter have unity gain at dc. The dc gain of the impulse-invariant filter is considerably larger than unity as a result of aliasing. Note also that the bilinear z-transform filter has a zero at the Nyquist frequency ($f_s/2$). This results since the analog prototype has a zero at $f = \infty$ and this zero is mapped to the Nyquist frequency under the bilinear z-transformation. ■

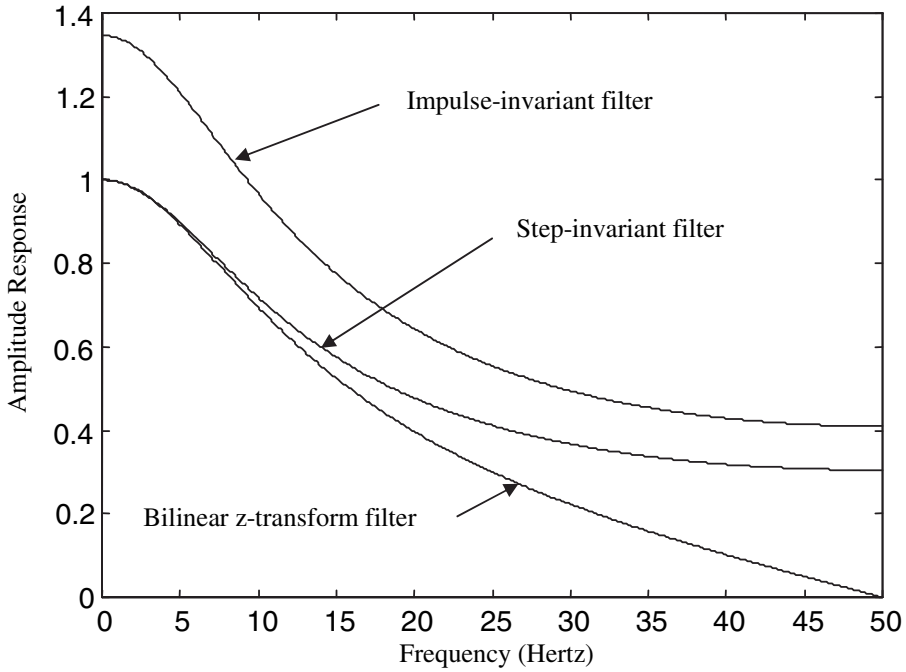


Figure 5.5 Comparison of amplitude responses.

Example 5.6. The filter developed in the previous example was based on a very simple first-order prototype and the filter coefficients: the values of a_k and b_k were determined without computer aid. In this example we consider a fifth-order elliptic filter with 1 dB passband ripple and a minimum 20 dB stop-band attenuation. While it is certainly possible to determine the filter coefficients using the analytical techniques illustrated in the preceding example, considerable effort would be required because of the complexity of the transfer function of the analog prototype.

MATLAB, however, has a number of digital filter synthesis tools in the Signals and Systems Toolbox. Among these are routines for the design of impulse-invariant and bilinear z-transform digital filters. The MATLAB code for the synthesis of the fifth-order elliptic filter is as follows:

```
% File:  c5_ellipexam.m
fs = 100;                          % set sampling frequency
fc = 20;                           % set cuttoff frequency
f = 0:0.1:50;                      % define frequency vector
[b,a] = ellip(5,1,20,2*pi*fc,'s'); % synthesize elliptic filter
h = freqs(b,a,2*pi*f);             % amp.  resp.  of analog filter
[bz1,az1] = impinvar(b,a,fs);      % impulse invariant digital filter
```

```
h1 = freqz(bz1,az1,f,fs);          % amplitude response of above
[bz2,az2] = bilinear(b,a,fs);      % bilinear z filter (not prewarped)
h2 = freqz(bz2,az2,f,fs);          % amplitude response of above
[bz3,az3] = bilinear(b,a,fs,fc);   % bilinear z filter (prewarped)
h3 = freqz(bz3,az3,f,fs);          % amplitude response of above
subplot(211)                       % subplot 1
plot(f,abs(h),f,abs(h1))           % plot
xlabel('Frequency - Hz')           % label x axis
ylabel('Amplitude Response')       % label y axis
subplot(212)                       % subplot 2
plot(f,abs(h2),f,abs(h3))          % plot
xlabel('Frequency - Hz')           % label x axis
ylabel('Amplitude Response')       % label y axis
% End of script file.
```

Note that four frequency responses are generated. First, the amplitude response of the analog prototype is generated to serve as a basis of comparison. The amplitude response of the impulse-invariant digital filter is generated and plotted with the amplitude response of the analog prototype. Two bilinear z-transform digital filters are synthesized: one with frequency warping and one without frequency warping. These are plotted in order to illustrate the effect of frequency warping.

The amplitude response of the analog prototype, the impulse-invariant digital filter, and the two bilinear z-transform filters are compared in Figure 5.6. Two separate plots are used to reduce clutter. The top plot compares the amplitude response of the impulse-invariant digital filter with the analog prototype. Note that, except for the bandwidth, they compare poorly. The reason for this is aliasing error, which is quite pronounced in the case of an elliptic filter, since the amplitude response of the analog prototype is not bandlimited because of the stopband behavior of the filter. This can be seen in the elevated amplitude response in both the filter passband and in the filter stopband.

The bottom plot illustrates that the bilinear z-transform synthesis method performs much better. If frequency warping is not used, the cutoff frequency is reduced from 20 Hz to approximately 18 Hz as expected. Prewarping the cutoff frequency of the digital filter to the cutoff frequency of the analog prototype results in an amplitude response that closely matches that of the analog prototype over most of the frequency range. There is, however, one notable difference. At the Nyquist frequency ($f_s/2 = 50$ Hz) the amplitude response of the analog prototype is 0.1, corresponding to 20 dB attenuation, while the amplitude response of the bilinear z-transform digital filter is 0 ($-\infty$ dB). The analog filter has a zero at $f = \infty$ and, under the bilinear z-transformation, this zero is mapped to the Nyquist frequency. ■

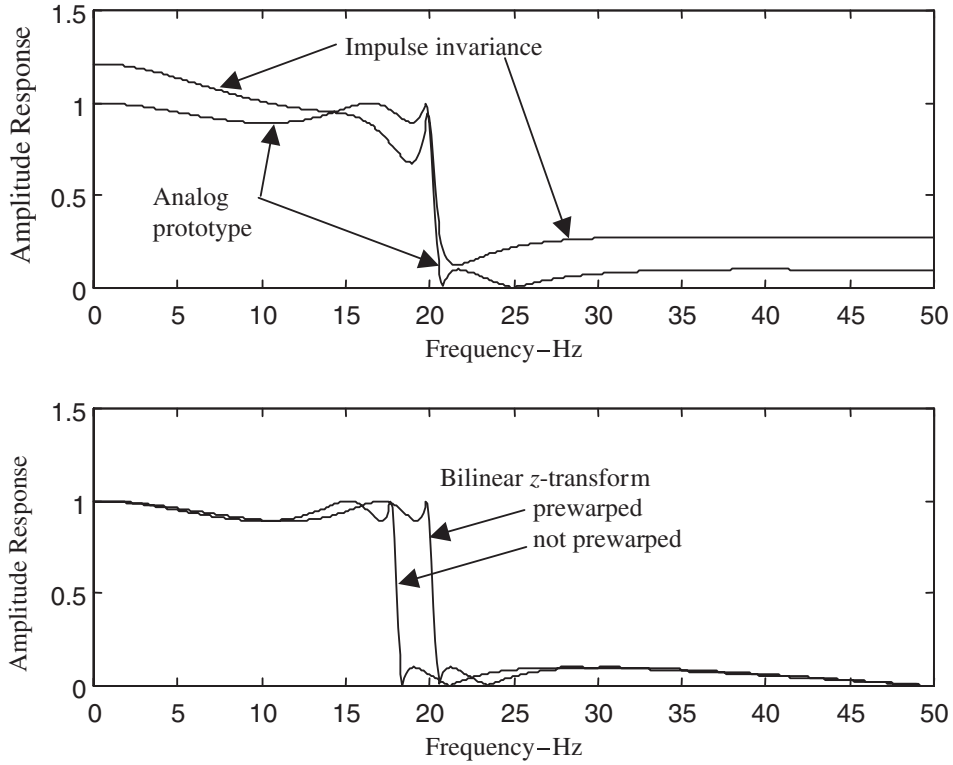


Figure 5.6 Comparison of amplitude responses.

Special Case: Trapezoidal Integration

An important integration algorithm, frequently used in simulation programs, is trapezoidal integration. In general, discrete-time integration is defined by the expression

$$y[n] = y[n-1] + \triangle(n-1, n) \tag{5.52}$$

where $y[n]$ and $y[n-1]$ denote the integrator output at index n and $n-1$, respectively, and $\triangle(n-1, n)$ denotes the incremental area added to the integral in the time increment $(n-1)T < t \leq nT$. The incremental area is illustrated in Figure 5.7 and is given by

$$\triangle(n-1, n) = \frac{T}{2}\left(x[n] + x[n-1]\right) \tag{5.53}$$

Thus

$$y[n] - y[n-1] = \frac{T}{2}\left(x[n] + x[n-1]\right) \tag{5.54}$$

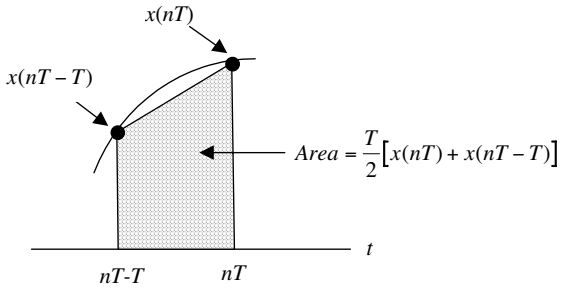


Figure 5.7 Trapezoidal integration.

Taking the z-transform of both sides of (5.54) yields the transfer function of the trapezoidal integrator. The result is

$$H(z) = \frac{Y(z)}{X(z)} = \frac{T}{2}\frac{1+z^{-1}}{1-z^{-1}} \tag{5.55}$$

If we design an integration algorithm based on the bilinear z-transform synthesis technique, the result is

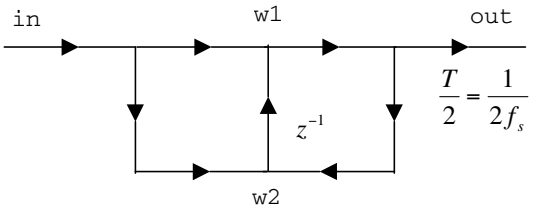
$$H(z) = H_a(s)\big|_{s=\frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}}} \tag{5.56}$$

or, since integration is equivalent to division by s

$$H(z) = \frac{1}{s}\bigg|_{s=\frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}}} = \frac{T}{2}\frac{1+z^{-1}}{1-z^{-1}} \tag{5.57}$$

which is identical to the trapezoidal integration rule expressed by (5.55).

The signal-flow graph corresponding to the transposed DF II implementation of the trapezoidal integrator is illustrated in Figure 5.8. Note that, with the exception of the amplitude scaling by $T/2$, all multiplies are unity. The MATLAB simulation




```
w1 = in + w2;
w2 = in + w1;
out = w1/twofs;
```

Figure 5.8 Signal-flow graph for trapezoidal integrator and MATLAB code.

code is also shown in Figure 5.8. We see that the delay is realized by the order in which the states, `w1` and `w2`, are computed.

Trapezoidal integration is only one of many different integration algorithms used in simulation programs. Having a variety of algorithms provides one with the capability of making a number of speed/performance tradeoffs. Some of these algorithms will be treated in Chapter 12.

5.4.4 Computer-Aided Design of IIR Digital Filters

Over the past few decades a number of computer-aided design techniques have been developed for both IIR and FIR digital filters [2]. The FIR techniques have received more widespread application than the IIR techniques, but IIR techniques are sometimes used, since smaller structures (fewer delay elements) often result. One advantage of the computer-aided design methods is that they allow digital filters to be developed that have no analog equivalent. In addition, the use of computer-aided design methods allow implementation tradeoffs, such as complexity (the required number of coefficients) and accuracy (the error between an actual and a target amplitude response), to be studied quickly and systematically.

A common technique for developing IIR filters is to adjust the filter coefficients so that the best fit to a desired response is obtained. In the most common technique, one specifies a desired filter characteristic, which is most often the amplitude response of the desired filter, and the order N of the digital filter. This leads to a constrained optimization problem in which the order N is the constraint and the best fit, in the minimum mean square error sense, is determined. Various algorithms for accomplishing this task are described in the literature. We consider here an example.

Example 5.7. A channelizing filter is to be developed having the characteristic illustrated in Figure 5.9. Note that the filter passes two channels, only one of which has been translated to dc. Note also that frequency in Figure 5.9 is normalized with respect to the Nyquist frequency (one-half the sampling frequency). In terms of this normalized frequency $f_N = f/(f_s/2)$, the filter amplitude response is defined as

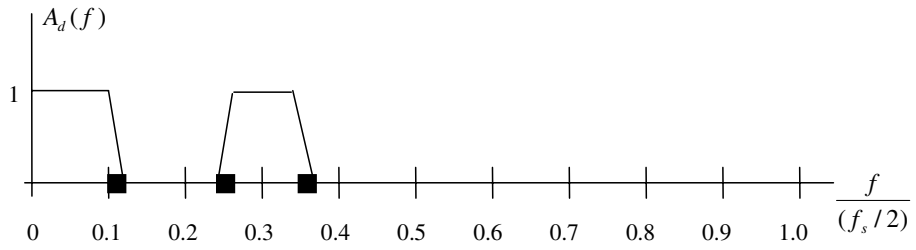


Figure 5.9 Desired amplitude response.

$$A_d(f) = \begin{cases} 1, & |f| < 0.1, \quad 0.25 < f < 0.35 \\ 0, & 0.12 < f < 0.23, \quad 0.37 < f < 1 \end{cases} \tag{5.58}$$

The three frequency bands not included in the expression for $A_d(f)$ are transition bands. These are indicated by the heavy black regions in Figure 5.9.

The MATLAB program for designing the filter is as follows:

```
% File:  c5_yw.m
order = 20,                                  % degree of polynomials
f = [0 0.1 0.12 0.23 0.25 0.35 0.37 1];  % frequency points
amp = [1 1 0 0 1 1 0 0];                     % amplitude response
[b,a] = yulewalk(order,f,amp);               % synthesize filter
freqz(b,a)                                   % display results
% End of script file.
```

Note that a 20^{th} order filter is generated so that both the denominator and the numerator polynomials of $H(z)$ are polynomials of degree 20 in z^{-1}. Executing the program yields the result illustrated in Figure 5.10. Note that the stopband attenuation is approximately 30 dB or better. The passbands are reasonably well shaped and the phase response is approximately linear across the passbands. Whether or

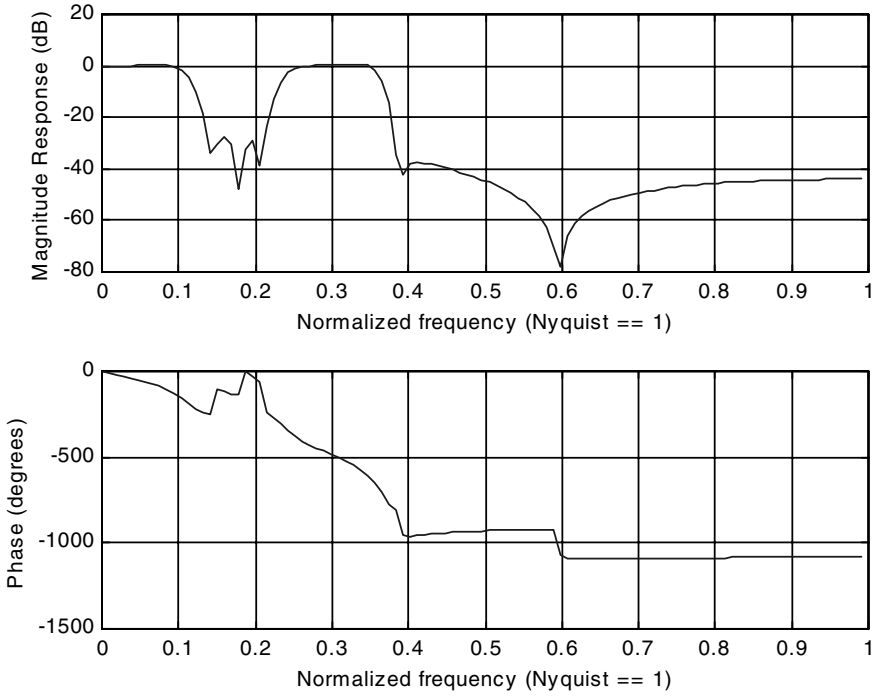


Figure 5.10 Computer-aided design results for IIR filter.

not this filter is a satisfactory approximation to the ideal response defined by (5.58) is a matter of judgment, and a system simulation using this filter may be required to answer that question. Increasing the filter order will improve the approximation error but at the cost of increased complexity resulting in increased simulation runtime. ■

5.4.5 Error Sources in IIR Filters

In this section we summarize the error sources resulting in the approximation of analog filters by IIR digital filters. We have seen that the source of error is dependent upon the synthesis method used. For the most part these errors may be minimized by using a very high sampling frequency in the simulation. However, unnecessarily high sampling frequencies result in simulations that take an unnecessarily long time to execute. As is usually the case, selection of a sampling frequency for a simulation involves a tradeoff between accuracy and the time required to execute the simulation. The error sources are summarized in Table 5.1.

Table 5.1 IIR Digital Filter Error Sources

Synthesis Technique	**Error Source**	**To Mimimize Error**
Impulse invariant	Aliasing	Choose a higher sampling frequency
Step invariant	Aliasing	Choose a higher sampling frequency
Bilinear z-transform	Frequency warping	Select a sampling frequency much larger than the highest critical frequency
CAD method	Approximation error	Increase filter order or use another synthesis method more suitable to the application

5.5 FIR Filters: Synthesis Techniques and Filter Characteristics

If the impulse response of a filter $h[n]$ is finite, or has been made finite by truncating an originally infinite duration impulse response, the filter output in the discrete time domain is given by

$$y(nT) = T\sum_{k=0}^{N} h(kT)x\{(n-k)T\} \tag{5.59}$$

which, using standard DSP notation, is

$$y[n] = \sum_{k=0}^{N} h[n]x[n-k] = \sum_{k=0}^{N} b_k x[n-k] \tag{5.60}$$

or, in terms of the z transform,

$$Y(z) = \sum_{k=0}^{N} b_k z^{-k} X(z) \tag{5.61}$$

Equation (5.60), which defines an FIR digital filter, is the discrete-time version of the convolution integral for analog (continuous time) filters. Note that (5.60) and (5.61) are the same as the filter model defined by (5.1) and (5.3) with $a_k = 0$ for $k \geq 1$. The filter is FIR (Finite duration Impulse Response) since there are at most $N+1$ nonzero terms in $h[n]$. Since $a_k = 0$ for $k \geq 1$ the FIR filter has no feedback paths. The convolution operation in time domain defined in (5.60) can be simulated using the MATLAB function `filter`.

The FIR filter is attractive for a number of reasons, the most significant of which are as follows:

1. Not all filters in a communication system can be expressed in terms of a transfer function in the Laplace transform domain and hence the IIR techniques described in the previous sections cannot be applied directly. Two important filters that fall into this category are the square root raised cosine (SQRC) pulse shaping filter, and the Jakes Doppler filter. These filters are easily simulated using the FIR approach.

2. In many simulation applications, filter data may be given empirically in the form of measured frequency response or impulse response data. It is much easier to simulate these filters using the FIR approach. While there are some techniques available for fitting an ARMA model to the frequency response data, such approximations do not yield satisfactory results when the frequency response data is not relatively smooth.

3. With the FIR approach we can specify arbitrary amplitude and phase responses, and they can be independent of each other. Thus it is possible, for example, to simulate an ideal brick-wall filter with linear phase response.

4. The FIR filters lack feedback and are therefore always stable

The FIR simulation model does have one main drawback, namely, it is computationally not as efficient as the IIR implementation. Direct implementation of the convolution operation given in (5.60) requires N (complex) multiplication and additions for each output sample to be generated. If the impulse response is very long, say $N > 1024$ points, the FIR algorithm will execute much more slowly than a typical IIR filter algorithm. For an IIR filter the number of additions and multiplications required to generate each output sample are determined by the order

of the filter and not by the length of the impulse response as is the case for the FIR filter.

The computational efficiency of the FIR model can be improved by using DFT/FFT operators to perform the convolution operation [2]. When using DFT/FFT operators it is important to pay attention to the following:

1. DFT/FFT operators are periodic in nature and they produce circular or periodically convolved outputs. To obtain linearly convolved outputs, the input sequence and the impulse response sequence must be zero-padded [2]. In order to get the maximum computational efficiency, the extended length of the padded vectors should be a power of two so that radix-2 FFT algorithms can be utilized.

2. If the input sequence is very long, the input sequence is typically broken up into smaller non overlapping blocks and convolution is performed for each input block to find the corresponding output blocks, which are then overlapped properly and added [2] to produce the overall output. If the length of the input block is relatively short, say less than say 216 samples, the output block is typically computed in "one shot". Otherwise, an overlap and add approach is recommended. The overlap and add method of DFT/FFT based convolution can be performed using the MATLAB function `fftfilt`.

3. DFT/FFT based convolution is a block processing operation and the first sample in the output block is produced only after sufficient input samples have been accumulated to carry out the DFT/FFT operation. Because of this built-in delay, DFT/FFT filters cannot typically be used to simulate a filter that is part of a feedback loop.

It is well known that the computational efficiency of the time domain convolution versus DFT based operation is proportional to $\log_2 N/N$. If the length of the impulse response is less than 128 samples, there is not a significant difference in the computational load between the time domain convolution and the DFT/FFT based implementations.

The computational efficiency of FIR methods (both time domain and DFT/FFT based convolution) improve if the length of the impulse response is shortened by truncation. Truncation is equivalent to multiplying the impulse response $h[k]$, $k = 0.1.2\cdots$, by a window function $w[k]$ with a duration of N samples. The value of N is typically chosen such that at least 98% of the total energy in $h[n]$ is contained within the window. In other words

$$\sum_{k=0}^{N} |h[k]|^2 = 0.98 \sum_{k=0}^{\infty} |h[k]|^2 \tag{5.62}$$

The simplest window function is the rectangular window function which zeroes out the impulse response samples for $k > N$. In other words

$$w[k] = \begin{cases} 1, & 0 \leq k \leq N \\ 0, & \text{otherwise} \end{cases} \tag{5.63}$$

Many other window functions can be used. They are described in the literature.

Truncation (windowing) in time is equivalent to convolution in the frequency domain, that is, $H_T(f) = H(f) \circledast W(f)$, where, as always $\circledast$ denotes convolution. Ideally the window function in the frequency domain should be defined by $W(f) = \delta(f)$ since convolution with an impulse will not modify $H(f)$. An impulse in the frequency domain, however, corresponds to a constant in the time domain, which is infinite in duration rather than finite in duration. Thus, the ideal characteristic in the frequency domain gives rise to nonideal characteristics in the time domain. The inverse is also true. The selection of a window function is therefore a compromise and the resulting window introduces distortion. This distortion can be minimized by the appropriate choice of a window function. The desirable characteristics of the window function include (keep in mind that we would ideally like an impulse in the frequency domain):

1. "narrow main lobe" in the frequency domain containing most of the energy

2. small side lobes

The most commonly used window functions are the Rectangular window, the Hamming window, and the Kaiser window. In selecting a window function, the trade-off is between minimizing signal distortion versus computational load of applying the window.

5.5.1 Design from the Amplitude Response

A fundamental technique for designing FIR digital filters is based on the fact that the frequency response (magnitude and phase) and the unit impulse response of the filter are a Fourier transform pair. We typically derive the unit impulse response by specifying a desired amplitude response $A(f)$ and calculating the inverse Fourier transform. The target amplitude response is usually specified to be real and even so that the resulting unit impulse response is real and even. Since the impulse response is even, it is noncausal and, in order to implement the system in the time domain, the impulse response must be truncated so that it is of finite extent and must be shifted in time so that it is causal. Truncating the impulse response must be done with care in order not to introduce significant errors. The use of appropriate windowing can often reduce the impact of these errors. Shifting the impulse in time simply gives the filter a linear phase response equivalent to a group delay equal to the time shift. This technique produces a filter with an arbitrary amplitude response and a linear phase shift. If one desires a filter with both a target amplitude and phase response, a complex transfer function is specified. The final filter, assuming that the errors induced by truncation of the impulse response are

not significant will, except for a constant group delay, satisfy both the amplitude and phase response requirements of the target filter.

The amplitude response of a digital filter, which is a continous function of frequency f, is periodic in the sampling frequency and can be expresed as a Fourier series. The Fourier coefficients, which are discrete, give the impulse response of the desired digital filter. As we will see in the following two examples, the impulse response may be obtained by formally determining the inverse Fourier transform of the desired frequency response or may be obtained by applying the inverse FFT to a set of samples of the desired amplitude response. These techniques are very basic and, if used with care, result in useful filter designs.

Since the amplitude response of a digital filter is periodic in the sampling frequency, the amplitude response can be expanded in a Fourier series of the form

$$H(e^{j2\pi fT}) = \sum_{n=0}^{M-1} h[n] \exp(-j2\pi nfT) \tag{5.64}$$

where $h[n]$ represents the Fourier coefficients, M represents the length of the impulse response and $T = 1/f_s$. Note that (5.64) is precisely (5.6) with (5.4) used to obtain the steady-state frequency response from the transfer function $H(z)$. We now assume that $M = 2L + 1$ for reasons that will be apparent later. With this substitution and the change of index $k = n - L$ we have

$$H(e^{j2\pi fT}) = \sum_{k=-L}^{L} h[k + L] \exp[-j2\pi(k + L)fT] \tag{5.65}$$

or

$$H(e^{j2\pi fT}) = \exp(-j2\pi LfT) \sum_{k=-L}^{L} h[k + L] \exp(-j2\pi kfT) \tag{5.66}$$

which can be written in the form

$$H(e^{j2\pi fT}) = \exp(-j2\pi LfT) H_1(e^{j2\pi fT}) \tag{5.67}$$

where

$$H_1(e^{j2\pi fT}) = \sum_{k=-L}^{L} h_1[k] \exp(-j2\pi kfT) \tag{5.68}$$

Clearly $h_1[k] = h[k + L]$, which is the impulse response of the causal filter defined by (5.64) shifted shifted L samples. Note that the filter defined by (5.68) is not a casual filter. It is, however, very easy to design using basic Fourier techniques and can be made causal by simply shifting the impulse response by an appropriate amount. The amplitude response of the filters defined by $h[n]$ and $h_1[n]$ will clearly be identical and the transfer functions will differ only by a linear phase shift as defined in (5.67). Thus we base our design procedure on (5.68).

Assume that a filter is to have a given amplitude response $H_1(e^{j2\pi fT}) = A(f)$. Multiplying both sides of (5.68) by $\exp(j2\pi mfT)$ gives

$$A(f)\exp(j2\pi mfT) = \sum_{k=-L}^{L} h_1[k]\exp[j2\pi(m-k)fT] \tag{5.69}$$

Integrating both sides of (5.69) over the simulation bandwidth, which is one period of $H_1(e^{j2\pi fT})$, gives

$$\sum_{k=-L}^{L} h_1[k]I(m,k) = \int_{-f_s/2}^{f_s/2} A(f)\exp(j2\pi mfT)\,df \tag{5.70}$$

where

$$I(m,k) = \int_{-f_s/2}^{f_s/2} \exp[j2\pi(m-k)fT]\,df \tag{5.71}$$

Integrating, and recognizing that $f_sT = 1$, yields

$$I(m,k) = \frac{1}{T}\frac{\sin\pi(m-k)}{\pi(m-k)} = \frac{1}{T}\delta(m-k) \tag{5.72}$$

Substituting this result into (5.70) gives

$$h_1[m] = T\int_{-f_s/2}^{f_s/2} A(f)\exp(j2\pi mfT)\,df, \quad -L \le m \le L \tag{5.73}$$

This will be our basic design equation.

Example 5.8. In this example we design a digital lowpass filter that approximates an ideal digital filter with a bandwidth of $\lambda f_N = \lambda f_s/2$, where f_N is the Nyquist rate, f_s is the sampling frequency, and λ is a parameter between 0 and 1. Thus the desired amplitude response is

$$A(f) = \begin{cases} 1, & |f| < \lambda f_s/2 \\ 0, & \text{otherwise} \end{cases} \tag{5.74}$$

From (5.73) we have

$$h_1[m] = T\int_{-\lambda f_s/2}^{\lambda f_s/2} (1)\exp(j2\pi mfT)\,df \tag{5.75}$$

Performing the integration gives

$$h_1[m] = T\frac{1}{\pi mT}\frac{1}{j2}\left[\exp(j\pi m\lambda f_sT) - \exp(-j\pi m\lambda f_sT)\right] \tag{5.76}$$

Since $f_sT = 1$, (5.76) can be written

$$h_1[m] = \frac{1}{\pi m}\sin(\pi m\lambda) \tag{5.77}$$

Note that $h_1[0] = h[L] = \lambda$.

The following MATLAB code is used to explore the preceding technique:

```
% File:  c5_FIRdesign.m
L = 30;                                  % 2L+1 total points
lam = 0.3;                               % normalized cutoff frequency
m = -L:1:L;                              % vector of points
bp = sin(pi*lam*(m+eps))./(pi*(m+eps)); % impulse response
stem(0:2*L,bp,'.')                       % plot impulse response
xlabel('Sample index')
ylabel('Impulse response')
figure; a = 1; freqz(bp,a)               % plot amp and phase response
figure; subplot(2,1,1)                   % new figure
[H w] = freqz(bp,a);
plot(w/pi,abs(H)); grid;                 % unwindowed amp response
xlabel('Frequency (normalized to the Nyquist frequency = fs/2)')
ylabel('|H(f)| (unwindowed)')
subplot(2,1,2)
w = 0.54+0.46*cos(pi*m/L);               % Hamming window
wbp = bp.*w;                             % apply window
[H w] = freqz(wbp,a);
plot(w/pi,abs(H)); grid;                 % windowed amp response
xlabel('Frequency (normalized to the Nyquist frequency = fs/2)')
ylabel('|H(f)| (windowed)')
% End of script file.
```

In the preceding code we have used $L = 30$ and $\lambda = 0.3$. Note that `eps` is added to the index m in order to prevent an indeterminate form at $m = 0$. The required shift of $L = 30$ samples to make the impulse causal is accomplised by the way in which MATLAB indexes vectors.

The first output generated by the MATLAB program is the impulse response, which is illustrated in Figure 5.11. Note that $h[30] = 0.3$ as determined earlier. Note also that the impulse is even about the center weight at $m = L = 30$.

The amplitude and phase response of the FIR filter, generated using the MATLAB command `freqz` in the default plot mode, is illustrated in Figure 5.12. As expected, the filter has linear phase since the impulse response is even about the center weight. The linear phase shift (constant group delay) results from shifting the impulse response L samples to make the filter causal. The sawtooth-like phase response in the stopband results from the sign changes in the amplitude response.

The magnitude response in the filter passband appears to be flat in Figure 5.12. In order to verify this, the magnitude response is plotted a second time using a linear scale in order to avoid the amplituder compression induced by the log scale. The result is illustrated in the top pane in Figure 5.13. We see that there is considerable

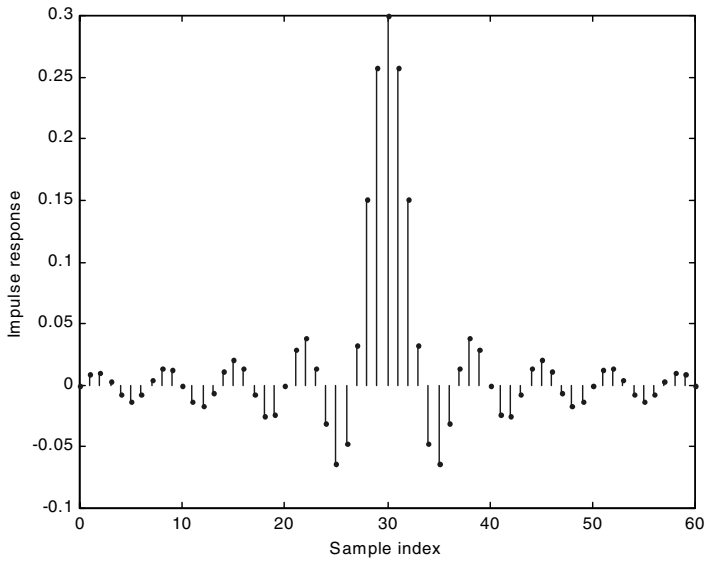


Figure 5.11 FIR filter impulse response.

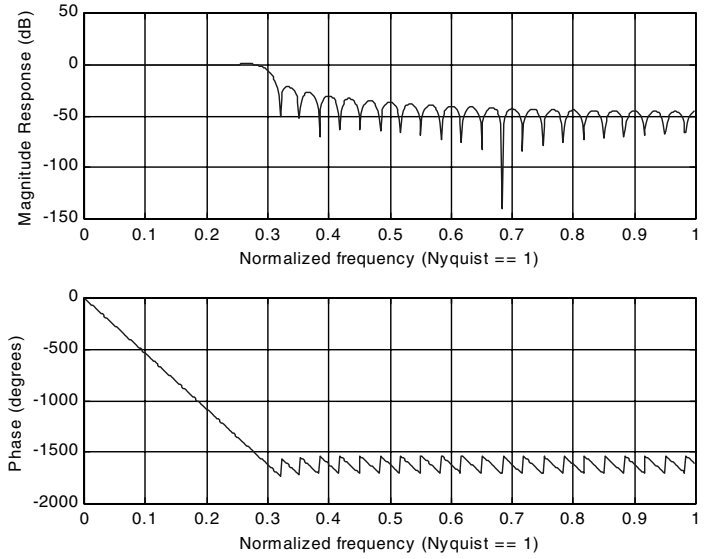


Figure 5.12 Amplitude and phase response of example filter.

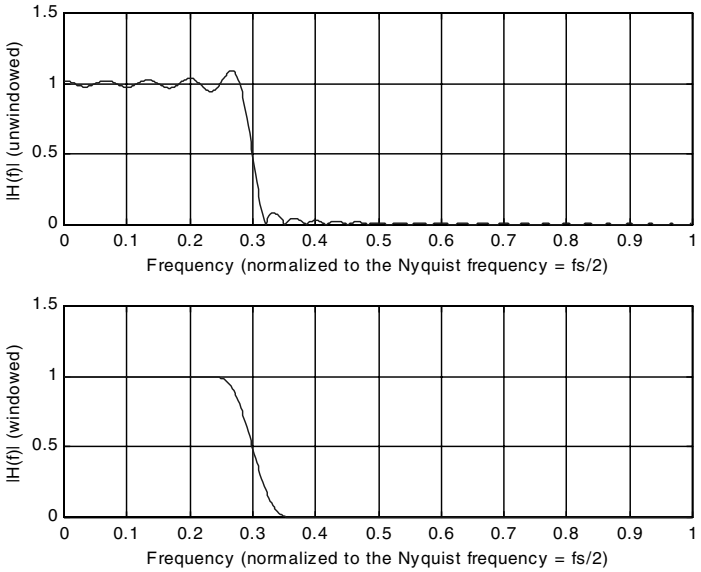


Figure 5.13 Effect of windowing.

ripple in the passband. This ripple is caused by truncating the impulse response by multipling the impulse response by a rectangular window which, in the frequency domain, is equivalent to convolving the ideal frequency response with $\sin(\beta f)/\beta f$. (The value of β is determined by the width of the window.) The passband ripple resulting from the rectanglar window can be reduced by using a window function that has a smoother transition from $w[n] = 1$ to $w[n] = 0$. (Recall that we desire a window function that is more like an impulse function in the frequency domain.) A simple and frequently used window function is the Hamming window defined by the weights

$$w[n] = 0.54 + 0.46 \cos\left(\frac{\pi n}{L}\right), \quad -L \leq n \leq L \tag{5.78}$$

Note that the window function $w[n]$ must be shifted in time by L samples so that $w[0]$ is centered on $h[n]$. The result of using the Hamming window is illustrated in the bottom pane of Figure 5.13. Note that both the passband and the stopband ripple is supressed when the Hamming filter is used. ■

Example 5.9. In this example we design a digital lowpass filter having the amplitude response of an analog Butterworth filter but also having linear phase. There

is no analog equivalent to this filter. The Butterworth filter is defined by the amplitude response

$$A(f) = \frac{1}{\sqrt{1 + (f/f_c)^n}} \tag{5.79}$$

where f_c is the bandwidth, or 3-dB frequency and n is the order of the filter. The next step is to sample the frequency response. The frequency response samples are given by

$$A(f_k) = \frac{1}{\sqrt{1 + (f_k/f_c)^n}} \tag{5.80}$$

An inverse FFT operation is then performed on these frequency samples to generate the impulse response. Keep in mind that, for an N-point inverse FFT, the negative time samples will appear in the returned vector with index ranging from $(N/2) + 1$ to N and must be reindexed to give the proper impulse response. The code for computing the impulse response samples for a linear phase filter having a Butterworth amplitude response follows.

```
% File:  c5_firbutter.m
order = 30; fc =5;                 % set filter parameters
fmax = 100;                        % set max frequency
npts = 256;                        % set number of samples
f = (0:(npts-1))*(fmax/(npts-1));  % frequency vector
nn = 2*npts;                       % size ifft
H = zeros(1,nn);                   % initialize vector
Ha = 1./(sqrt(1+(f/fc).^order));   % amplitude response
H = [Ha 0 fliplr(Ha(2:npts))];     % even amplitude response
[cimp_resp] = ifft(H,nn);          % complex impulse response
imp_resp = real(cimp_resp);        % take real part
aa = imp_resp(1:npts);             % time > = 0
bb = imp_resp((npts+1):nn);        % time < 0
reimpulse = [bb aa];               % real and even imp.  resp.
plot(reimpulse)                    % plot result
% End of script file.
```

The impulse response is illustrated in Figure 5.14 for `order = 1` and `order = 30`. The response is a two-sided decreasing exponential (two-sided since we assumed the amplitude response $A(f)$ real). For `order = 30`, the impulse response should closely approximate a sinc function since a Butterworth filter of very high order is a good approximation of an ideal (brick wall) filter. Thus the results illustrated in Figure 5.14 appear correct.

In practice one would wish to minimize the computational burden in a simulation by minimizing the number of weights (b_k terms) used to represent the impulse response. Minimizing the number of weights minimizes the number of multiply operations and the number of stages in the tapped delay line (shift register). It appears

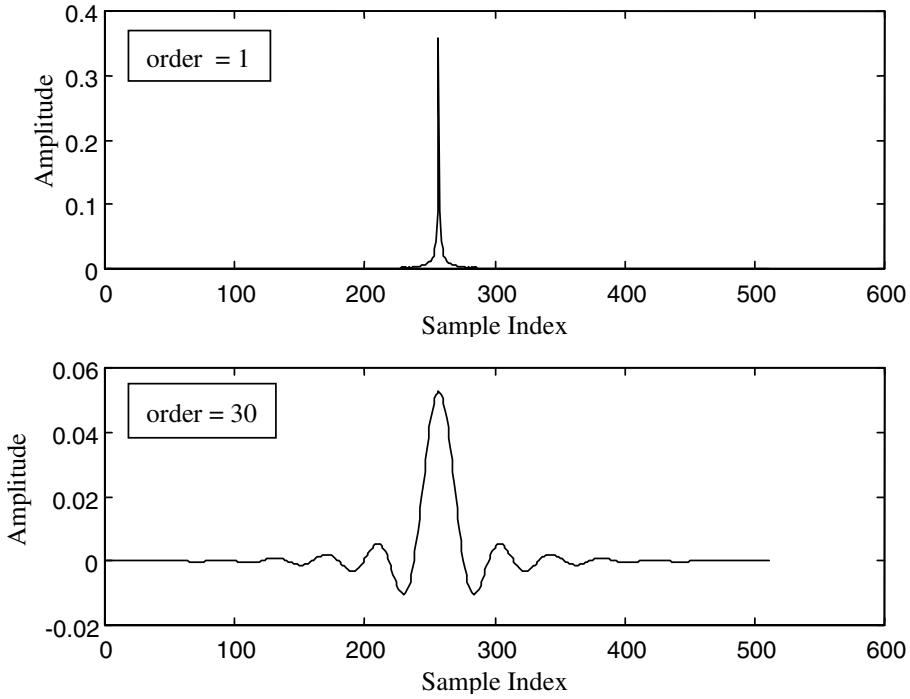


Figure 5.14 Filter synthesis results—impulse responses

that many of the weights in the `order = 1` impulse response are approximately zero and can therefore be eliminated with negligible impact on the amplitude response. The same may be true (but to a lesser extent) in the `order = 30` impulse response. One therefore should window the impulse response with a rectangular window, having sufficient width to contain all significant terms in the impulse response, and discard those terms outside the window. ■

5.5.2 Design from the Impulse Response

The previous example, in which the digital filter was synthesized from the specification of the amplitude and phase response characteristics, is an application of frequency sampling. In many applications, an analytical expression is known for the impulse response of the filter. When this is the case, sampling the impulse response gives results in an FIR filter design. This is particularly true of filters used to form the pulse shape to be used for signal transmission. In this case the data is viewed as

a sequence of impulses separated in time by the pulse duration, T. For the binary case we have

$$d(t) = \sum_k d_k \delta(t - kT) \tag{5.81}$$

where $d_k = 1$ for a binary 1 and $d_k = -1$ for a binary 0. Passing this through a pulse-shaping filter having impulse response $p(t)$ gives the waveform

$$x(t) = \sum_k d_k p(t - kT) \tag{5.82}$$

The following example gives two common examples.

Example 5.10. The pulse shape $p(t)$ used for data transmission is often chosen to be a pulse shape satisfying the Nyquist zero-ISI (intersymbol interference) property [3, 4, 5]. One example of a zero-ISI pulse shape is the raised cosine pulse given, in the frequency domain, by

$$P(f) = \begin{cases} T, & 0 \le |f| \le \frac{1-\beta}{2T} \\ \frac{T}{2}\left[1 + \cos\frac{\pi T}{\beta}\left(|f| - \frac{1-\beta}{2T}\right)\right], & \frac{1-\beta}{2T} < |f| \le \frac{1+\beta}{2T} \\ 0, & |f| > \frac{1-\beta}{2T} \end{cases} \tag{5.83}$$

where T is the pulse duration or symbol time. Taking the inverse Fourier transform gives the pulse shape

$$p(t) = \frac{\sin \pi t/T}{\pi t/T} \frac{\cos(\pi\beta t/T)}{1 - 4\beta^2 t^2/T^2} \tag{5.84}$$

This is clearly a noncausal pulse. Typically one delays the pulse by an integer number of symbol periods, say mT, and truncates the pulse to $2mT$. The value of m is a tradeoff between convenience and accuracy requirements. We then take k samples per symbol period so that $T = kT_s$ where T_s is the sampling period. Replacing t by $t - t_d = t - mT$, letting $t = nT_s$ and $T = kT_s$ yields

$$\frac{t}{T} \to \frac{nT_s - mkT_s}{kT_s} = \frac{n}{k} - m \tag{5.85}$$

Making this substitution in (5.84) gives the sample sequence

$$p[n] = \frac{\sin \pi\{(n/k) - m\}}{\pi\{(n/k) - m\}} \frac{\cos(\pi\beta\{(n/k) - m\}}{1 - 4\beta^2\{(n/k) - m\}^2}, \qquad 0 \le n \le 2m \tag{5.86}$$

representing the impulse response of the digital filter.

In order to illustrate the raised cosine response, we let $x[n] = \delta[n - 1]$ be the input to a filter having the impulse response $p[n]$. The output is

$$y[n] = p[n] \circledast x[n] = p[n - 1] \tag{5.87}$$

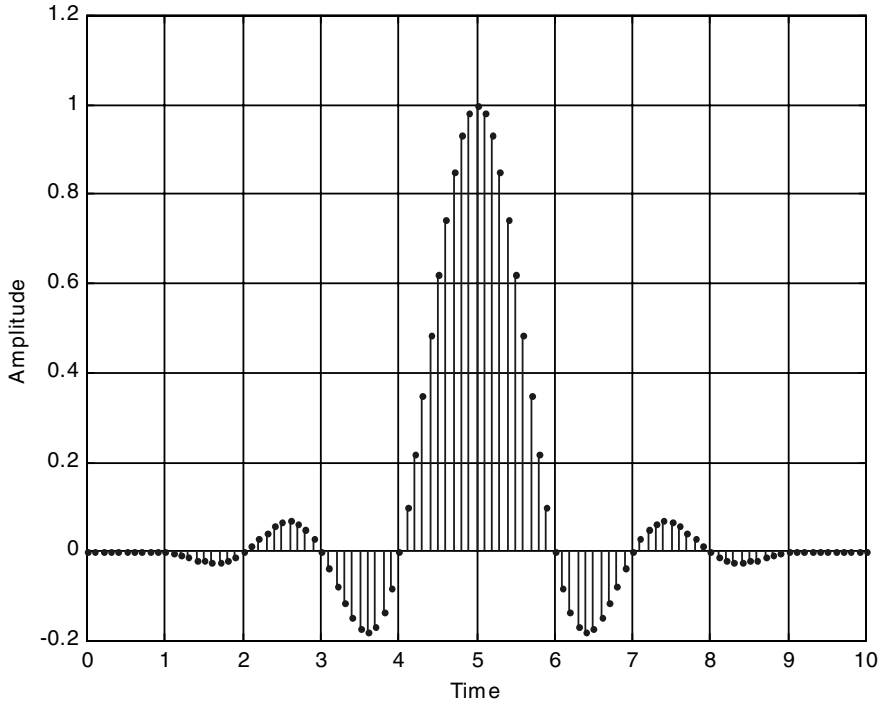


Figure 5.15 Raised cosine pulse example.

The result is illustrated in Figure 5.15 for $\beta = 0.32$, $k = 10$ samples per symbol and $m = 4$ symbols. It can clearly be seen that $p[n]$ is identically equal to zero at integer multiples of the assumed pulse duration $T = 1$. The MATLAB program used to generate Figure 5.15 is given in Appendix A.

In many system designs the transfer function $P(f)$ is realized as the cascade of two filters with each having the transfer function $\sqrt{P(f)}$. One of these filters is part of the transmitter and the other filter is part of the receiver. This gives

$$p_{SQRC}(t) = 4\beta \frac{\cos[(1+\beta)\pi t/T] + \sin[(1-\beta)\pi t/T](4\beta t/T)^{-1}}{\pi\sqrt{T}[1 - 16\beta^2 t^2/T^2]} \tag{5.88}$$

The substitution defined by (5.85) can be applied to implement the delay and sampling operations. The result is

$$p_{SQRC}[n] = 4\beta \frac{\cos\{(1+\beta)\pi[(n/k) - m]\} + \sin\{(1-\beta)\pi[(n/k) - m]\}\{4\beta[(n/k) - m]\}^{-1}}{\pi\sqrt{T}[1 - 16\beta^2[(n/k) - m]^2]} \tag{5.89}$$

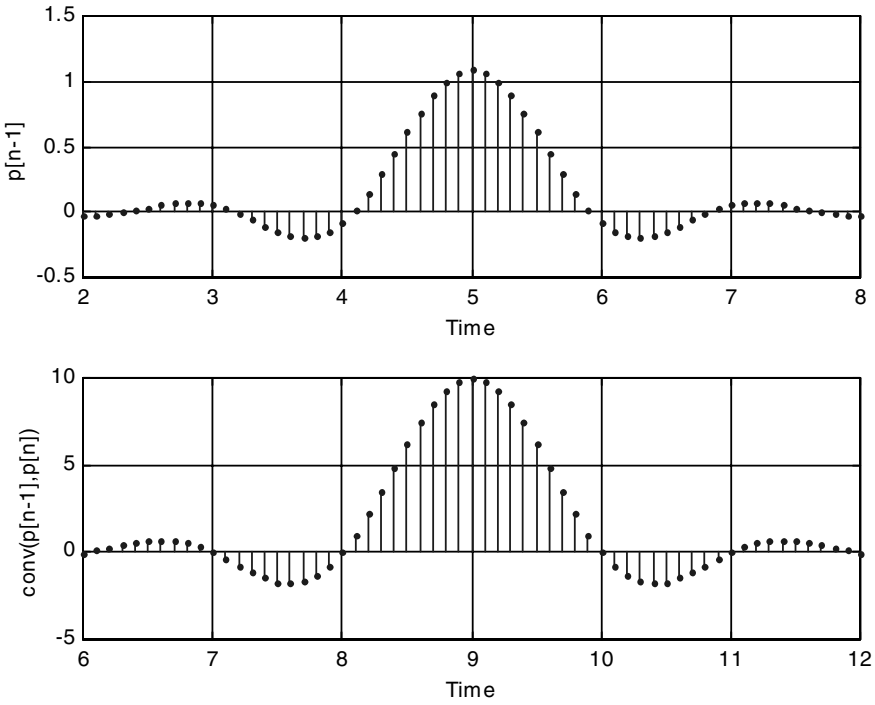


Figure 5.16 Square root raised cosine pulse.

Figure 5.16 (top pane) illustrates the response of the filter having the impulse response $p_{SQRC}[n]$ to input $\delta[n-1]$. This is

$$p_{SQRC}[n] \circledast \delta[n-1] = p_{SQRC}[n-1] \tag{5.90}$$

The filter parameters are $\beta = 0.32$, $k = 10$ samples per symbol, $T = 1$, and $m = 4$ symbols. Note that the zero crossings do not exactly occur at integer multiples of $T = 1$. The bottom pane of Figure 5.16 illustrates the convolution of the sample sequence shown in the top frame with $p_{SQRC}[n]$. This essentially represents the convolution of $p_{SQRC}[n]$ with itself and is the equivalent of two SQRC filters in cascade. Note that the zero crossings now fall at integer multiples of $T = 1$, since the cascade combination of two SQRC filters is a zero-ISI filter. The MATLAB program used to compute these results is given in Appendix B. ■

5.5.3 Implementation of FIR Filter Simulation Models

We have seen that FIR models play a central role in the simulation of communication systems. In the previous sections, design techniques were demonstrated for the case in which the transfer function, or the impulse response, were known analytically so

that they could be sampled. Another important use of the FIR simulation model is during the later stages of a design process. At this point, filters may have already been designed and built and so that the measured frequency response is available. FIR simulation models are especially suitable for simulating filters whose frequency response is given in empirical form (measured or otherwise).

Let us look at the important steps involved in simulating a filter whose frequency response is arbitrary and is specified empirically in table form (this data is usually obtained from network analyzer measurements). The first step is the selection of two key parameters, namely, the sampling rate and the time duration of the truncated impulse response. These parameters for implementing an FIR filter must be chosen carefully in order to minimize the computational complexity while at the same time providing satisfactory resolution in both the time and the frequency domains. The guidelines for choosing these parameters are:

- The sampling rate, f_s typically satisfies $f_s > 16B$ to $32B$, where B is the bandwidth of the filter. The time resolution, the time between samples, T_s is then $1/f_s$.
- The frequency resolution, Δ_f is $1/(NT_s)$ where $N = f_s/\Delta_f$. The preferred value is between $B/64$ and $B/32$.
- The number of samples/symbol is a integer, which is typically a power of 2. The minimum value is 8.
- The duration of the impulse response is typically 8 to 16 symbols.

These considerations will typically lead to $N = 1024$, where N is the number of filter taps or the number of samples of the impulse response. After choosing the key parameters f_s and N (and hence T_s and Δ_f) we then preprocess the given frequency response data and implement the simulation model.

The first step in the implementation of the model involves the preprocessing of the frequency-response data. This step consists of applying a bandpass-to-lowpass transformation (if the filter is a bandpass filter), resampling the frequency response data, and converting group delay to phase by numerically integrating the group delay data. The bandpass-to-lowpass transformation is simply a relabeling of the frequency axis ($f_1 = f - f_c$).

This operation may result in a lowpass equivalent that does not have conjugate symmetry about $f = 0$. In this case the impulse response of the lowpass equivalent filter may be complex valued. As a practical example, this effect occurs when the bandpass-to-lowpass transformation is applied to a vestigal sideband filter.

The second step (resampling) will be necessary, for example, if the network analyzer measurements are given at frequency points different from those to be used in the FIR implementation. For example suppose that 100 samples of the frequency response are collected using the network analyzer and the FIR implementation is to be based on $N = 1024$ samples. The frequency response in this case must be interpolated. A simple linear interpolation will be adequate in most cases [1]. The last step is the conversion of group delay response to phase response. Again, group

delay data can be converted to phase response by numerically integrating the group delay over frequency.

Once the frequency response of the lowpass equivalent version of the filter is available, the MATLAB implementation of the FIR model consists of the following steps:

1. Extend the frequency response data to $-f_s/2$ to $f_s/2$, and obtain sampled values of the transfer function $H(f)$, $-f_s/2 < f < f_s/2$ where $f = k\Delta_f$, $k = -N/2$ to $N/2-1$, where Δ_f is the resolution in frequency domain ($\Delta_f = 1/(NT_s)$.

2. Move the negative frequency portion of $H(f)$ to $N/2+1$ to N so that the frequency response is now contained in $H(k\Delta_f)$, $k = 1, 2, \cdots, N$.

3. Take the inverse FFT to obtain the impulse response [the sampling rate in time domain will be $T_s = 1/(N\Delta_f)$].

4. Apply a window to the impulse response if necessary (make sure that the impulse response is "rotated" properly, and centered within the window function). Normalize the windowed impulse response to have unit energy.

After the truncated impulse response is obtained, the filter can be simulated in MATLAB using the `filter` function with parameters `a = 1` and `b = h`, where `h` is the impulse response array, that is, the filter output is computed using `output = filter (b,a,input)`. If the input sequence is very long, the output can be computed using the function `output = fftfilt(b,input,N)`, which implements the FIR filter in blocks using the overlap and add method with a block size of N as mentioned previously.

The following MATLAB example shows how to simulate a 125 MHz bandpass channelizing filter used in satellite applications. The measured characteristics of the data are given in terms of amplitude response and delay characteristics (see Appendix C).

Example 5.11. The MATLAB program for designing an FIR filter using the technique just described is contained in Appendix C. The measured data upon which the design is based is also contained in Appendix C (Tables 5.4 and 5.5). The results are illustrated in Figures 5.17 and 5.18 with Figure 5.17 illustrating the windowed and unwindowed impulse responses and Figure 5.18 illustrating the windowed and unwindowed amplitude responses. The windowed and unwindowed responses are plotted separately since they match almost perfectly and they would be indistinguishable from each other if plotted together. The only significant difference between the two amplitude responses is the flat region in the neighborhood of sample index 150 to 200. The attenuation is slightly greater when the Hamming window is used. ■

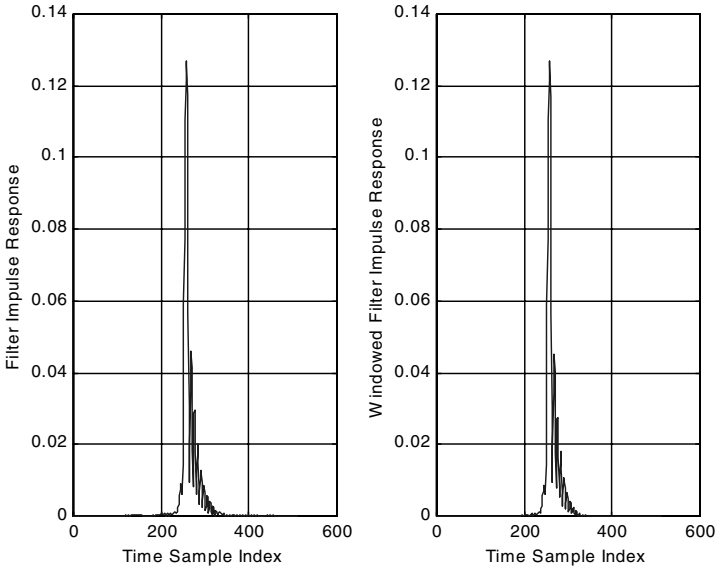


Figure 5.17 Impulse responses (unwindowed and windowed).

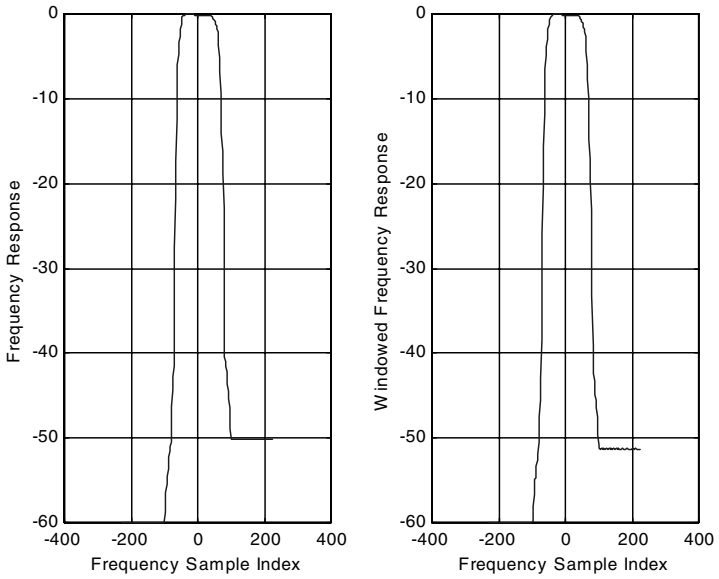


Figure 5.18 Amplitude responses (unwindowed and windowed).

5.5.4 Computer-Aided Design of FIR Digital Filters

The most popular of the CAD techniques for FIR filter design is the Parks-McClellan algorithm, which is applicable to filters whose ideal amplitude response characteristic is piecewise constant. As an example of the application of the Parks-McClellan algorithm consider again the synthesis of the channelizing filter having the amplitude response characteristic shown in Figure 5.9. The Parks-McClellan algorithm makes use of a Chebyshev polynomial fit to the desired amplitude response and is an equiripple approximation to the desired amplitude response. That is, on a linear scale, the amplitude of the passband and the stopband ripples are equal. The optimization is in the minimax sense in that the maximum error is minimized.

Example 5.12. As an example of computer-aided design of FIR filters and the Parks-McClellan algorithm, consider the desired amplitude response shown in Figure 5.9. The MATLAB program for determining the impulse of the desired filter follows:

```
% File:  c5_pmc.m
order = 50,                                % points in impulse response
f = [0 0.1 0.12 0.23 0.25 0.35 0.37 1];  % frequency points
amp = [1 1 0 0 1 1 0 0];                   % amplitude response
b = remez(order,f,amp);                    % synthesize filter
stem(b,'.k')                               % plot impulse response
xlabel('Sample Index')                     % label x axis
ylabel('Amplitude')                        % label y axis
pause                                      % pause
freqz(b,1)                                 % plot results
% End of script file.
```

Note that the vectors designating the frequency points and the desired amplitude response at those frequency points are common to both the IIR and FIR filter synthesis programs.

Executing the program yields the impulse response illustrated in Figure 5.19. Note that there are 51 terms in the impulse response, corresponding to `order = 50`, and that the impulse response is even about the center term. Implementation of the filter takes the form of a TDL with the values illustrated in Figure 5.19 as the tap weights. Design of the filter can be performed "off line" or it can be linked directly to the simulation program so that the filter design process becomes part of the simulation process.

The amplitude and phase response of the digital filter are illustrated in Figure 5.20. Note the passband and the stopband ripple, which is characteristic of the Parks-McClellan design process. We see that the stopband attenuation is almost 20 dB. The stopband attenuation could be increased by increasing the order of the filter. Increasing the order of the filter will also reduce the passband ripple. The required filter order is determined by the allowable passband ripple (small passband ripple requires a higher-order filter), the stopband attenuation (greater stopband attenuation requires a higher-order filter), and the width of the transition bands

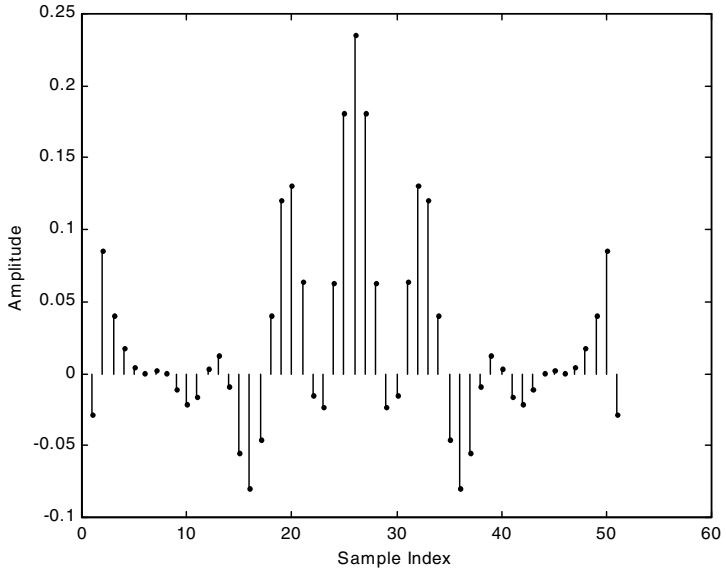


Figure 5.19 Impulse response of example FIR filter.

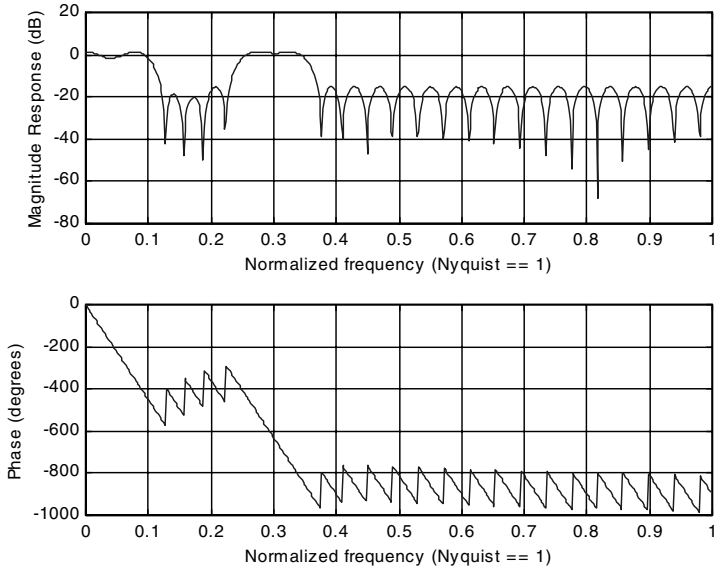


Figure 5.20 Amplitude and phase response of example FIR filter.

(narrow transition bands require a higher-order filter). Typically design using the Parks-McClellan process is an iterative process in which specifications are adjusted within allowable limits in the hope that an impulse response with a reasonable length will result.

Since the impulse response takes the form of a delayed even function, the phase response will be linear as can be seen in Figure 5.20. The 180-degree jumps in the phase response correspond to the sign changes in the amplitude response as the amplitude response goes through zero ($-\infty$ dB). The slope of the phase response, the group delay of the filter, is determined by the delay required to make the impulse response causal. ■

5.5.5 Comments on FIR Design

As previously mentioned, FIR filters can be designed that have no analog equivalent. This is of increasing importance since communications systems are becoming more software based with filtering operations performed digitally. In terms of implementation, the most important advantages and disadvantages are given in Table 5.2. Figure 5.21 summarizes FIR filter design and simulation techniques. Note that design may be performed using either time-domain or frequency-domain data. A time-domain simulation or a frequency-domain simulation may result from either choice of design data.

Table 5.2 Advantages and Disadvantages of FIR Filter Implementation Techniques

Implementation	Advantages	Disadvantages
Time Domain	Simple implementation	Time consuming for long impulse response sequences
Frequncy Domain	Fast processing Easy to design using frequency response data	Introduces artificial delay equal to FFT block length Cannot be used in systems with feedback
CAD Methods	Leads to linear phase filters	Long impulse responses typical

5.6 Summary

In this chapter we considered the design of digital filters for use in simulations. A number of techniques are available for this task and each technique has both advantages and disadvantages. Design can proceed from an analog prototype or one can design a digital filter for which no analog prototype exists. In addition, digital filter design can be accomplished using a time-domain criterion or a frequency-domain criterion. The basic filter classifications are IIR and FIR. The technique that is selected for a given simulation is often dictated by experience and insight rather

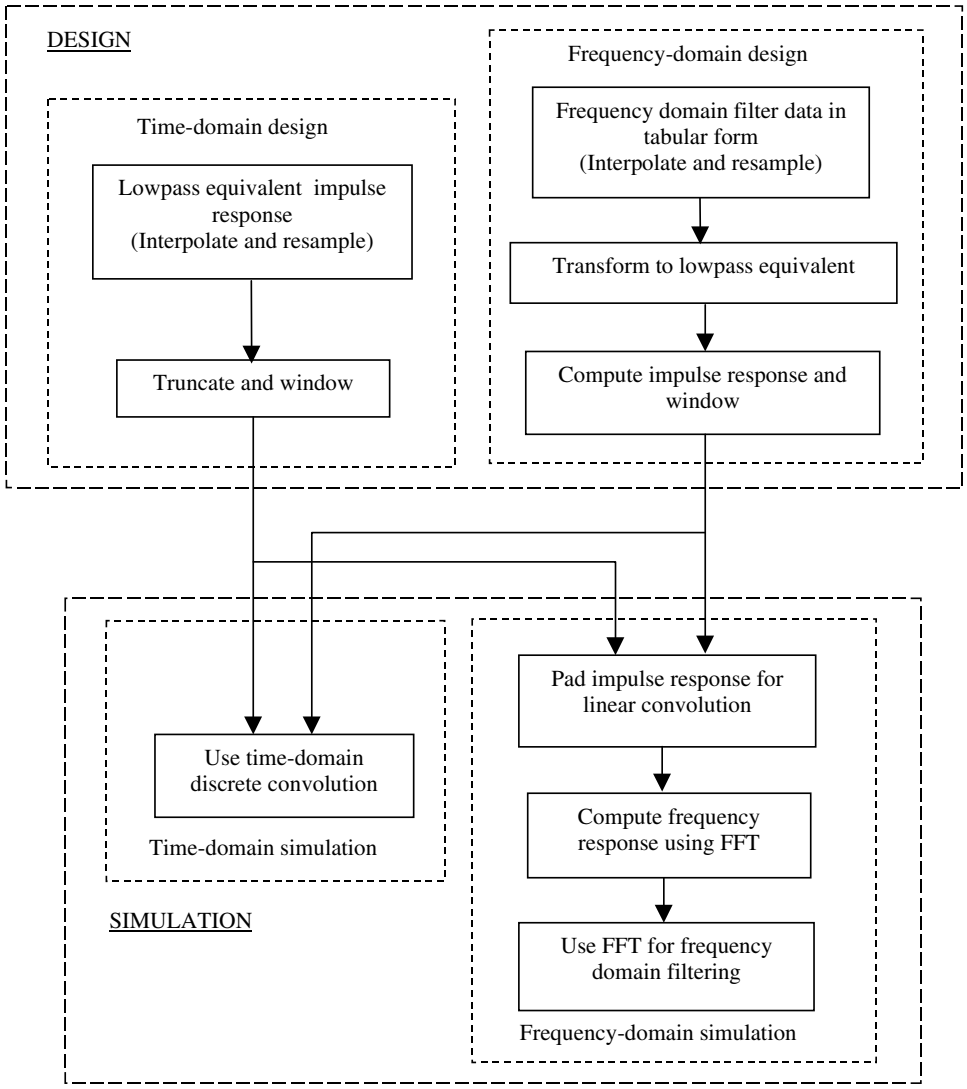


Figure 5.21 FIR filter design and simulation techniques.

than by any specific analytical technique. Thus, selection of a given methodology for representing filters in a simulation is, as discussed in Chapter 2, often more "art" than "science."

The transposed Direct Form II filter structure is almost universally used representing IIR filters in simulation applications. The reason for this lies in the fact that unit delays, the basic building block of filter structures, can be implemented

through the order in which filter states are called in a simulation program. Use of the transposed Direct Form II structure results in very fast execution of the filter program.

The most common techniques for designing IIR filters based on analog prototypes are the impulse-invariant method, the step-invariant method, and the bilinear z-transform method. The first two of these techniques, impulse invariance and step invariance, are time-domain synthesis techniques, since the goal is to match the impulse or step response of the digital filter to the impulse response or step response of the analog prototype. The resulting frequency response of the digital filter, when compared to the frequency response of the analog prototype, exhibits errors due to aliasing. This effect is worse in the impulse-invariant digital filter than in the step-invariant digital filter, which is why the step-invariant filter is frequently preferred in simulation applications. In order to minimize aliasing errors, the sampling frequency must be increased. The bilinear z-transform synthesis technique is a frequency-domain synthesis method, since one attempts to match the frequency response of the digital filter to the frequency response of the analog prototype. This frequency response matching can be accomplished only at a limited number of frequencies (usually two). Errors result because of frequency warping. These errors can be minimized by choosing a sampling frequency much higher than the highest critical frequency in the filter. We see that increasing the sampling frequency is a general requirement for reducing the effect of the error sources inherent in the various synthesis techniques. This, in turn, increases the execution time for the simulation. Thus, we have a tradeoff between simulation run time and error.

A special case of IIR synthesis, based on the bilinear z-transform, was the trapezoidal integrator. We will see in the following chapter that, by using digital integration, we have the capability to simulate any system described by a differential equation. Numerical integration, based on the trapezoidal integration technique, will be used frequently in the work to follow.

FIR digital filters are typically realized in the time domain using a shift register architecture, often referred to as a transversal delay line. In the frequency domain FIR filters are implemented by taking the Fourier transform of the desired amplitude response. This can obviously be implemented using the FFT. Several design techniques for FIR filters were considered. One technique is to sample the target frequency response and inverse transform the samples to obtain the impulse response. If the impulse response of an analog filter is known, it may be sampled in order to define the digital filter. A technique for designing a FIR filter using measured data was also demonstrated. The other technique was based on computer-aided design. The Parks-McClellan algorithm is the most common computer-aided design method and results on a filter that is perfectly linear phase.

The most important simulation advantages and disadvantages of IIR and FIR filters are summarized in Table 5.3.

Table 5.3 Advantages and Disadvantages of IIR and FIR Filters

Filter Type	Advantages	Disadvantages
IIR	Fast execution Simple design using analog prototypes	An analog prototype typically required Phase response is typically disregarded Errors due to aliasing or frequency warping are often significant
FIR	Arbitrary amplitude and phase responses are easily modeled	Impementation using direct convolution is insfficient (excessive simulation time) FFT implementation is often complicated and leads to artificial delay due to block processing Errors due to aliasing and truncation of the impulse response are often significant

5.7 Further Reading

A number of textbooks have been written that provide details on the design of digital filters. The following is just a small sampling.

A. V. Oppenheim and R. W. Shafer, *Discrete-Time Signal Processing*, Upper Saddle River, NJ: Prentice Hall, 1989.

S. K. Mitra, *Digital Signal Processing: A Computer-Based Approach*, New York: McGraw-Hill, 1998.

A. Antoniou, *Digital Filters: Analysis, Design and Applications*, New York: McGraw-Hill, 1993.

R. W. Hamming, *Digital Filters*, 3rd ed., Upper Saddle River, NJ: Prentice Hall, 1989.

5.8 References

1. R. E. Ziemer, W. H. Tranter, and D. R. Fannin, *Signals and Systems: Continuous and Discrete*, 4th ed., Upper Saddle River, NJ: Prentice Hall, 1998.

2. A. V. Oppenheim and R. W. Shafer, *Discrete-Time Signal Processing*, Upper Saddle River, NJ: Prentice Hall, 1989.

3. T. S. Rappaport, *Wireless Communications: Principles and Practice*, 2nd ed., Upper Saddle River, NJ: Prentice Hall PTR, 2002.

4. G. L. Stuber, *Principles of Mobile Communication*, Boston: Kluwer, 1996.

5. J. D. Gibson, ed., *The Communications Handbook*, Boca Raton, FL: CRC Press and IEEE Press, 1997.

5.9 Problems

5.1 Show that the two structures shown in Figure 5.2 have the same transfer function.

5.2 Develop a MATLAB program to compare (5.47) and (5.51). Plot the amplitude responses and explain any differences.

5.3 Derive and plot the amplitude and phase responses for the filters shown in Figure 5.5 assuming a sampling frequency of 200 Hz. Show that the aliasing error is reduced by increasing the sampling frequency from 100 Hz to 200 Hz.

5.4 Add to Figure 5.5 the amplitude response of the analog prototype on which the three digital filters are based.

5.5 Use the MATLAB commands `impinvar` to design the impulse-invariant filter discussed in Example 5.1 and the MATLAB program `bilinear` to design the bilinear z-transform filter discussed in Example 5.3. By examining the coefficients of the numerator and denominator, as generated by the MATLAB program, compare the results.

5.6 Figure 5.6 compares the amplitude response of three digital filters with the analog prototype from which they were derived. Supplement the MATLAB program with the code required to generate the phase responses of the four filters. Plot and compare the phase responses.

5.7 A linear filter used in a simulation is defined by the two coefficient vectors

$$a = [1.0000 \quad -1.8777 \quad 1.6181 \quad -0.5724]$$

and

$$b = [0.0210 \quad 0.0630 \quad 0.0630 \quad 0.0210]$$

The sampling frequency is 100 Hz. Define the filter by determining the filter type and all revelent parameters (order, bandwidth,etc.). Document how this is accomplished and show all steps in the analysis.

5.8 Repeat the preceding problem assuming that the two coefficient vectors are

$$a = [1.0000 \quad -2.6649 \quad 3.2814 \quad -2.0817 \quad 0.5798]$$

and

$$b = [0.0051 \quad 0.0203 \quad 0.0304 \quad 0.0203 \quad 0.0051]$$

As before, the sampling frequency is 100 Hz.

5.9 Consider the program for generating the impulse response of a FIR digital filter having the amplitude response of an analog Butterworth filter. It was pointed out in the discussion that the digital filter will have linear phase. Show this by determining and plotting the phase response of the filter for `order=1` and `order=30`.

5.10 Plot the amplitude and phase responses of the filters defined by (5.47) and (5.51). Discuss the results.

5.11 Plot the impulse response of a raised cosine filter given by (5.84) $\beta = 0.2$, 0.5, 0.7, and 0.9.

5.12 Plot the impulse response of a square root raised cosine filter given by (5.88) $\beta = 0.2$, 0.5, 0.7, and 0.9.

5.13 An ideal lowpass filter is defined by

$$A(f) = \begin{cases} 1, & |f| < 20 \text{ Hz} \\ 0, & |f| > 22 \text{ Hz} \end{cases}$$

The transition band is defined by $20 < |f| < 22$.

(a) Using the Parks-McClellen algorithm, design a digital filter that approximates the given analog filter.

(b) Assume a sampling frequency of 100 Hz and let the order of the filter be 50. Plot the frequency and phase response and measure the passband ripple and the stopband ripple.

(c) Now assume that the transition band is defined by $20 < |f| < 25$. What filter order is required to give the same passband and stopband ripple as was determined in (b).

5.14 Determine the Fourier transform of the amplitude response of the channelizing filter illustrated in Figure 5.9. Determine the amplitude response that results using a 51 point series expansion of the desired amplitude response. Compare this result with the result obtained using the Parks-McClellan synthesis technique (Example 5.12). Comment on the results.

5.15 Design an FIR digital filter to simulate a VSB (vestigal side band) bandpass filter using the approach demonstrated in Example 5.10. The amplitude response to be approximated is illustrated in Figure 5.22. Note that VSB filters have an amplitude response that is nonsymmetrical about the carrier.

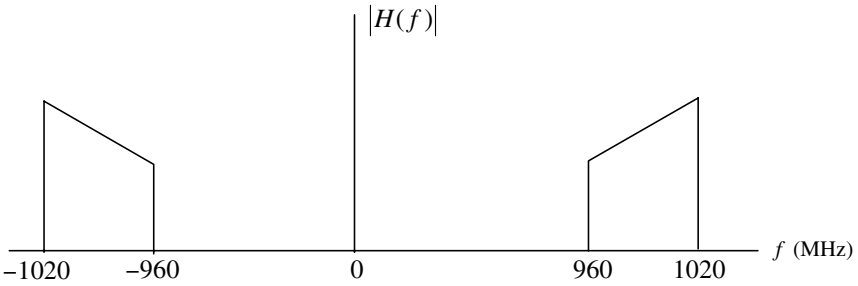


Figure 5.22 Amplitude response for Problem 5.15.

5.10 Appendix A: Raised Cosine Pulse Example

5.10.1 Main program c5_rcosdemo.m

```
% File: c5_rcosdemo.m
k = 10;                                % samples per symbol
m = 4;                                 % delay
beta = 0.32;                           % bandwidth factor
h=rcos(k,m,beta);                      % impulse response
in = zeros(1,101); in(11) = 1;         % input
out = conv(in,h);                      % output
t = 0:0.1:10;                          % time vector for plot
stem(t,out(1:101),'.')                 % plot output}
grid
xlabel('Time')
ylabel('Amplitude')
% End of script file.
```

5.10.2 Function file c5_rcos.m

```
% File: c5_rcos.m
function h=rcos(k,m,beta)
% k - samples per symbol
% m - delay is mT
% beta - bandwidth factor
%
beta = beta;
n = 0:2*m*k;
z = (n/k)-m+eps;
t1 = cos(beta*pi*z);
t2 = sin(pi*z)./(pi*z);
t3 = 1-4*beta*beta*z.*z;
h = t2.*t1./(t3);
% End of function file.
```

5.11 Appendix B: Square Root Raised Cosine Pulse Example

5.11.1 Main Program c5_sqrcdemo.m

```
% File: c5_sqrcdemo.m
T = 1;                                  % symbol time
k = 10;                                 % samples per symbol
m = 4;                                  % delay
beta = 0.32;                            % bandwidth factor
h=sqrc(T,k,m,beta);                     % impulse response
in = zeros(1,101); in(11) = 1;          % input
out = conv(in,h);                       % output h[10]
out1 = conv(out,h);                     % conv of h[n-10] and h[n]
t = 2:0.1:8;                            % time vector for plot
subplot(2,1,1)
stem(t,out(21:81),'.')                  % plot h[n-10]
grid
xlabel('Time')
ylabel('h[n-10]')
subplot(2,1,2)
t = 6:0.1:12;                           % time vector for plot
stem(t,out1(61:121),'.')                % plot conv of h[n-10] and h[n]
grid
xlabel('Time')
ylabel('conv(h[n-10],h[n])')
% End of script file.
```

5.11.2 Function file c5_sqrc.m

```
% File: c5_sqrc.m
function h=sqrc(T,k,m,beta)
% T - symbol time
% k - samples per symbol
% m - delay is mT
% beta - bandwidth factor
%
n = 0:2*m*k;
z = (n/k)-m+eps;
t1 = cos((1+beta)*pi*z);
t2 = sin((1-beta)*pi*z);
t3 = 1./(4*beta*z);
den = 1-16*beta*beta*z.*z;
num = t1+t2.*t3;
c = 4*beta/(pi*sqrt(T));
h = c*num./den;
% End of function file.
```

5.12 Appendix C: MATLAB Code and Data for Example 5.11

The code in Appendix C implements MATLAB Example 5.11.

Main Program: `c5_FIRFilterExample.m` (script file)

Called Functions: `FIR_Filter_AMP_Delay.m, Log_psd.m, shift_ifft.m`

Note 1: The filter data is given in Tables 5.4 and 5.5.
Note 2: `hamming.m` is a MATLAB library function.

Table 5.4 Negative Frequency Data

Freq.	$\|H(f)\|^2$	**Grp.Del.**	**Freq.**	$\|H(f)\|^2$	**Grp.Del.**
−500.0	−60.0	33.3333	−54.0	−2.2	10.6667
−400.0	−60.0	33.3333	−52.0	−1.8	7.4667
−380.0	−60.0	33.3333	−50.0	−1.5	5.6667
−360.0	−60.0	33.3333	−48.0	−0.24	4.0000
−340.0	−60.0	33.3333	−46.0	−0.1	3.0000
−320.0	−60.0	33.3333	−44.0	−0.04	2.0000
−300.0	−60.0	33.3333	−42.0	0.0	1.4000
−280.0	−60.0	33.3333	−40.0	0.08	0.8000
−260.0	−60.0	33.3333	−38.0	0.12	0.5333
−240.0	−60.0	33.3333	−36.0	0.14	0.2667
−220.0	−60.0	33.3333	−34.0	0.16	0.1333
−200.0	−60.0	33.3333	−32.0	0.19	0.0
−180.0	−60.0	33.3333	−30.0	0.2	0.0
−160.0	−60.0	33.3333	−28.0	0.2	0.0
−140.0	−60.0	33.3333	−26.0	0.2	0.0
−120.0	−60.0	33.3333	−24.0	0.18	0.0
−100.0	−60.0	33.3333	−22.0	0.17	0.0
−80.0	−50.0	33.3333	−20.0	0.16	0.0
−70.0	−40.0	33.3333	−18.0	0.15	0.0
−69.0	−30.0	31.6667	−16.0	0.14	0.0
−68.0	−25.0	30.0000	−14.0	0.13	0.0
−66.0	−20.0	26.6667	−12.0	0.12	0.0
−64.0	−15.0	24.3333	−10.0	0.11	0.0
−62.5	−10.0	22.6667	−8.0	0.1	0.0
−62.0	−8.0	20.0000	−6.0	0.06	0.0
−60.0	−6.0	18.3333	−4.0	0.04	0.0
−58.0	−4.5	16.6667	−2.0	0.02	0.0
−56.0	−3.0	14.1333			

Table 5.5 Positive Frequency Data

Freq.	$\|H(f)\|^2$	**Grp. Del.**	**Freq.**	$\|H(f)\|^2$	**Grp. Del.**
0.0	0.0	0.0	56.0	−1.2	12.6667
2.0	0.0	0.0	58.0	−1.6	14.3333
4.0	0.0	0.0	60.0	−2.04	18.6667
6.0	−0.02	0.0	62.0	−2.5	20.3333
8.0	−0.02	0.0	62.5	−3.0	22.0000
10.0	−0.02	0.0	64.0	−4.0	24.3333
12.0	−0.02	0.0	66.0	−6.0	26.6667
14.0	−0.01	0.0	68.0	−8.0	30.0000
16.0	0.0	0.0	69.0	−10.0	33.3333
18.0	0.01	0.0	70.0	−12.0	33.3333
20.0	0.02	0.0	80.0	−40.0	33.3333
22.0	0.025	0.0	100.0	−50.0	33.3333
24.0	0.03	0.0	120.0	−50.0	33.3333
26.0	0.04	0.0	140.0	−50.0	33.3333
28.0	0.06	0.0	160.0	−50.0	33.3333
30.0	0.06	1.0000	180.0	−50.0	33.3333
32.0	0.06	1.2667	200.0	−50.0	33.3333
34.0	0.06	1.5000	220.0	−50.0	33.3333
36.0	0.06	1.6667	240.0	−50.0	33.3333
38.0	0.01	2.0000	260.0	−50.0	33.3333
40.0	−0.02	3.0000	280.0	−50.0	33.3333
42.0	−0.1	3.5000	300.0	−50.0	33.3333
44.0	−0.16	4.0000	320.0	−50.0	33.3333
46.0	−0.3	5.2000	340.0	−50.0	33.3333
48.0	−0.46	6.5000	360.0	−50.0	33.3333
50.0	−0.7	8.0000	380.0	−50.0	33.3333
52.0	−0.804	9.3333	400.0	−50.0	33.3333
54.0	−1.2	10.6667	500.0	−50.0	33.3333

5.12.1 c5_FIRFilterExample.m

```
% File: c5_FIRFilterExample.m
fscale=1; fshift=0.0; dscale = 1000;          % scaling parameters
c5_Filter_Data;                               % load data
Freq_Resp = data; fs = 900; filtsize = 512; ts = 1/fs;
[himp time] = FIR_Filter_AMP_Delay(Freq_Resp,fs,filtsize,fscale,...
    fshift,dscale);

%
% Apply a window
%
```

```
nw = 256; window1 = hamming(nw); window = zeros(filtsize,1);
%
% Make sure the window is centered properly
%
wstart = (filtsize/2)-(nw/2); wend = (filtsize/2)+(nw/2)-1;
window(wstart:wend) = window1;
impw = himp.*window';
%
figure; subplot(1,2,1); plot(abs(himp)); grid;
xlabel('Time Sample Index'); ylabel('Filter Impulse Response');
subplot(1,2,2); plot(abs(impw)); grid;
xlabel('Time Sample Index'); ylabel('Windowed Filter Impulse...
    Response');
[logpsd,freq,ptotal,pmax] = log\_psd(himp,filtsize,ts);
[logpsdw,freq,ptotal,pmax] = log\_psd(impw,filtsize,ts);
figure; subplot(1,2,1)
plot(freq(128:384),logpsd(128:384)); grid;
xlabel('Frequency Sample Index'); ylabel('Frequency Response');
subplot(1,2,2)
plot(freq(128:384),logpsdw(128:384)); grid;
xlabel('Frequency Sample Index'); ylabel('Windowed Frequency...
    Response');
% End of script file.
```

5.12.2 FIR_Filter_AMP_Delay.m

```
% File: FIR_Filter_AMP_Delay.m
function [h,times] = FIR_Filter_AMP_Delay (H,fs,n,fscale,fshift,...
    dscale)
% This function returns the impulse response of an FIR filter
% h = row vector of impulse response values at t=times
% h is rotated to center the impulse response array at n/2*ts
% It is assumed that there is no 'constant delay' in the freq...
%     response given
% H is an array of frequency response
% Column 1 : Frequencies fk in ascending order
% After translation and scaling frequencies must be -fs/2 < f < fs/2
% Column 2 : 20*log(|H(fk)|;
% Column 3: group delay in units 1/frequency
% (i.e., if freq is given in Mhz, then delay should be in micro...
%     seconds)
% Otherwise use dscale to adjust delay = delay/dscale
% Ex: If delay is given in ns then delay in microsecs = delay ns/1000
% Phase response is obtained by integration delay from f= 0
% Phase((k+1)df) = phase(kdf) + 2*pi* (fs/nfft)delay (kdf)
% =phase(kdf) + (2*pi*/nfft) (delay (kdf)/ts)
```

```
% fscale and fshift: f = (f-fshift)/fscale
% fs : Sampling rate
% n: duration of the impulse response; The frequency response
% is resampled from (-fs/2)+df/2 to fs/2-df/2 using df = fs/n
%
ts =1/fs; df = fs/n;
%
% Pick up the frequency, magnitude and phase response arrays
% Convert dbs to real magnitudes; Rescale frequencies
%
Hfreq=H(:,1); Hmag=H(:,2); Hdelay=H(:,3);
nn=max(size(Hmag)); Hreal=10.^(Hmag/20);
Hfreq=(Hfreq-fshift)/fscale; Hdelay = Hdelay/dscale;
%
% Set up index array for frequencies and times
%
index1=[0:1:(n/2)]; index2=[-(n/2)+1:1:-1]; index=[index1 index2]';
frequencies=(index*df); times=index*ts;
%
% Use shift fft function to change rotate the time indices
%
times=shift_ifft(times,n);
%
% Freq array goes from [0, df, 2df, ...to fs/2 -fs/2+df,....-df]
% Time array goes from 0 to n/2*ts -ts to - (n/2-1)ts
% For interpolation purposes add two more entries at -fs/2 and fs/2
% to the frequency response data
%
fmin=min(min(frequencies));fmax=max(max(frequencies));
%
% Extend freq and other arrays to cover from -fs/2 + df to fs/2 + df
%
Hfreq1 =Hfreq; Hreal1 = Hreal; Hdelay1 = Hdelay;
if fmin < Hfreq(1,1) % If the lower end does not extend to -fs/2...
                     %     add a point
    Hfreq1=[fmin;Hfreq];
    Hreal1=[1e-10;Hreal] ;
    Hdelay1=[Hdelay(1,1);Hdelay];
end
if fmax > Hfreq(nn,1) % If the higher end does not extend to fs/2...
                      %     add a point
    Hfreq1=[Hfreq1; fmax];
    Hreal1=[Hreal1;1e-10] ;
    Hdelay1=[Hdelay1 ;Hdelay(nn,1)];
end
```

```
%
% Interpolate the frequency response data and compute the complex
% transfer function mag*exp(i*phase)
%
Hreal_interpolated=interp1(Hfreq1,Hreal1,frequencies);
Hdelay_interpolated=interp1(Hfreq1,Hdelay1,frequencies);
%
% Integrate delay to find phase response
%
sum=0.;
Hphase(1)=0.;                          % Phase at carrier freq = 0
for k = 2:(n/2)+1                      % Integrate fore f >0
    sum= sum -(Hdelay_interpolated(k,1)/ts)* (2*pi/n);
    Hphase(k,1) = sum;
end
sum=0.0;
for k = n:-1:(n/2)+2                   % Integrate for f <0
    sum = sum+(Hdelay_interpolated(k,1)/ts)* (2*pi/n);
    Hphase(k,1)=sum;
end
Hcomplex =Hreal_interpolated.*exp(i*Hphase);
%
% Find the inverse fft and rotate it
%
hh=ifft(Hcomplex); h=(shift_ifft(hh,n));
%
% End of filter design and end of function file.
```

5.12.3 shift_ifft.m

```
% File: shift_ifft.m
function y = shift_ifft(x,n)
% Circular shift ifft array
for k=1:(n/2)-1
 y(k)=x((n/2)+k+1);
end
for k=1:n/2+1
 y((n/2)-1+k)=x(k);
end
% End of function file.
```

5.12.4 log_psd.m

```
% File: log_psd.m
function [logpsd,freq,ptotal,pmax] = log_psd(x,n,ts)
% This function takes the n time domain samples (real or complex)
```

```
% and finds the psd by taking (fft/n)^2. The two sided spectrum is
% produced by shifting the psd; The array freq provides the
% appropriate frequency values for plotting purposes.
% By taking 10*log10(psd/max(psd)) the psd is normalized; values
% below 60db are set equal to -60db
%
% n must be an even number, preferably a power of 2
%
y = zeros(1,n); % initialize y vector
%
h = waitbar(0,'For Loop in PSD Calculation');
for k=1:n
    freq(k) =(k-1-(n/2))/(n*ts);
    y(k) = x(k)*((-1.0)^k);
    waitbar(k/n)
end;
%
v = fft(y)/n;
psd = abs(v).^2;
pmax = max(psd);
ptotal = sum(psd);
logpsd = 10*log10(psd/pmax);
%
% Truncate negative values at -60 dB
for k = 1:n
    if(logpsd(k)<-60.0)
        logpsd (k) = -60.0;
    end
end
close(h)
% End of function file.
```

Chapter 6

CASE STUDY: PHASE-LOCKED LOOPS AND DIFFERENTIAL EQUATION METHODS

Even though it is still early in our study of simulation, we now have the tools required to use simulation effectively to investigate the design and operating characteristics of a number of important systems. While there are many candidate systems that could be explored at this point in our studies, our focus in this chapter is on the phase-locked loop (PLL). There are a variety of justifications for this choice. First, the PLL is a basic building block for many subsystems used in the implementation of modern communication systems. PLLs are widely used in frequency synthesis, for frequency multipliers and dividers, for carrier and symbol synchronization, and in the implementation of coherent receivers [1, 2]. There are many other applications for the PLL. Since the PLL is a nonlinear system, and is therefore very difficult to analyze and design using traditional (nonsimulation based) methods, simulation is widely used for the design and analysis of systems based on the PLL. We will see that a basic knowledge of numerical integration methods, such as we gained in the

preceding chapter, provides the concepts necessary to develop effective simulations for PLLs and for many other systems of interest.

Another reason for using the PLL for a detailed example simulation is that the PLL involves feedback. We will see that systems involving feedback must be simulated with care, especially with respect to the sampling frequency, if significant simulation errors are to be avoided.

As discussed in Chapter 4, analysis based on lowpass system models is equivalent to an analysis based on the corresponding bandpass system models, assuming of course that the lowpass models are correctly developed. In addition, simulation models derived from lowpass analytical models execute much more quickly than simulations based on bandpass models. Thus, much of the work in this chapter will be based on lowpass models. Finally, all simulations presented in this chapter are deterministic simulations, as discussed in Chapter 1.

Although the focus of this chapter is to illustrate a number of elementary simulation techniques, and we use the PLL only as an interesting case study relevant to the study of communications, sufficient theory is presented to make the results understandable and to provide a "sanity check" on the simulation results. It is important, as discussed in Chapter 1, that analysis and simulation work hand in hand. This chapter, therefore, begins with a basic description of the PLL. Using the first-order loop as an example, we show how the loop achieves phase lock. Attention is then turned to the second-order loop, which is more useful in system implementations. We use simulation to illustrate the cycle-slipping phenomena and to determine the time required to achieve phase lock. Simulation is accomplished by developing a discrete-time simulation model for the PLL by replacing continuous-time integration by the trapezoidal integration rule explored in the previous chapter.

Later in this chapter we will appreciate that, in addition to having the tools for studying PLLs, we also have the required tools for using simulation to study any system that can be defined in terms of a differential equation. The differential equation, and therefore the underlying system, can be nonlinear, time-varying, or both nonlinear and time-varying. Thus, upon completing this chapter, we will have the tools required to simulate a wide variety of extremely complicated systems.

6.1 Basic Phase-Locked Loop Concepts

The basic model of a PLL is illustrated in Figure 6.1. The input signal is assumed to be

$$x_{in}(t) = A_c \cos\left[2\pi f_c t + \phi(t)\right] \tag{6.1}$$

and the signal at the output of the voltage-controlled oscillator (VCO) is assumed to have the form

$$x_{vco}(t) = -A_v \sin\left[2\pi f_c t + \theta(t)\right] \tag{6.2}$$

The basic role of a PLL is to synchronize the phase of the VCO with the phase of the input signal so that the phase error, $\phi(t) = \theta(t)$, is small. We will see how this is accomplished in the following sections.

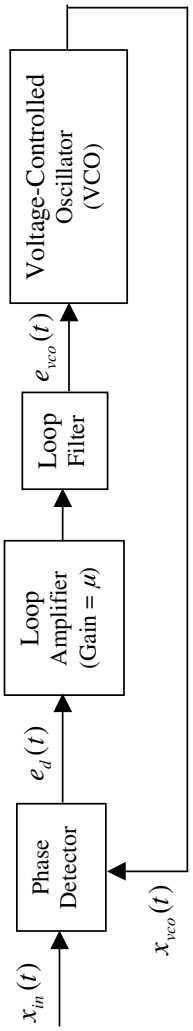


Figure 6.1 Block diagram of PLL.

PLLs typically operate in one of two modes, acquisition and tracking. In the acquisition mode, the PLL is attempting to synchronize, in both frequency and phase, the VCO output with an input signal. We will see that, in the acquisition mode, phase errors can be quite large. In this case, PLL operation is distinctly nonlinear, and a nonlinear model is required for analysis. Analysis of nonlinear models is extremely difficult and simulation is often required. In the tracking mode, however, the phase error is often small for long periods of time, and analysis using a simple linear model can often provide satisfactory results without the need for simulation. The standard loop parameters, as we will see, are defined in terms of the linear model. Thus, we have interest in both linear and nonlinear models. The focus of the simulations developed in this chapter will be on the nonlinear behavior of the system.

6.1.1 PLL Models

The first step in the development of a simulation model for a PLL is to model the phase detector. The characteristics of the phase detector determine, in large part, the operating characteristics of the PLL. There are many different types of phase detectors, and the choice of the phase detector model to be used in a given situation is dependent upon the application. The most common phase detector model, referred to as the sinusoidal phase detector, is one in which the output of the phase detector is proportional to the sine of the phase error. The sinusoidal phase detector can be viewed as consisting of a multiplier and a lowpass filter as shown in Figure 6.2. The only function of the lowpass filter is to remove the second harmonic of the carrier frequency resulting from the multiplication. We will see later that, in practical applications, the lowpass filter is not necessary.

Using the phase-detector model shown in Figure 6.2, the output of the phase detector is

$$e_d(t) = \frac{1}{2} A_c A_v \sin\left[\phi(t) - \theta(t)\right] \tag{6.3}$$

where the quantity $\phi(t) - \theta(t)$ is referred to as the phase error. Later we will denote the phase error by $\psi(t)$, but for now it is better to keep all of our expressions in terms of input phase, $\phi(t)$, and VCO phase, $\theta(t)$. We desire the VCO phase to be

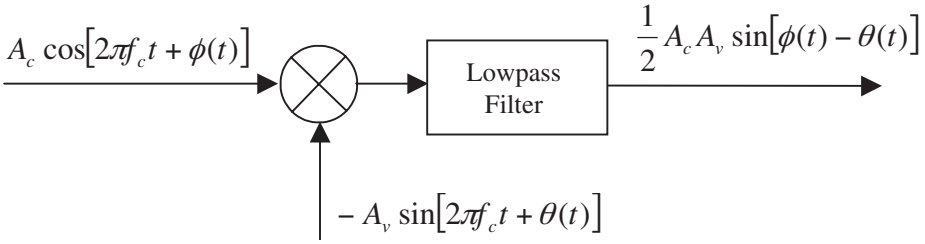


Figure 6.2 Model for a sinusoidal phase detector.

an estimator of the input phase and therefore proper operation of the PLL requires that the phase error be driven toward zero. The steady-state phase error may or may not be zero depending upon the characteristics of the input signal and the loop filter.

Note that the PLL input and the VCO output are in phase quadrature for $\phi(t) = \theta(t)$. This is required if the phase detector output is to be an *odd function* of the phase error. It is easily seen that if cosine functions are used in both (6.1) and (6.2), with the arguments unchanged, the phase detector output will be proportional to $\cos[\phi(t) - \theta(t)]$, which is an even function of the phase error. This, of course, yields the undesirable situation in which negative phase errors are not distinguishable from positive phase errors.

After multiplication by the loop amplifier gain, μ, the phase detector output $e_d(t)$ is filtered by a loop filter having the transfer function $F(s)$ and unit impulse response $f(t)$. The input to the VCO is therefore given by

$$e_{vco}(t) = \int_{-\infty}^{\infty} \mu e_d(\lambda) f(t-\lambda)\, d\lambda \tag{6.4}$$

which is simply the convolution of the loop filter impulse response with the loop filter input. For a sinusoidal phase detector this becomes

$$e_{vco}(t) = \int_{-\infty}^{\infty} \frac{1}{2}\mu A_c A_v \sin[\phi(\lambda) - \theta(\lambda)] f(t-\lambda)\, d\lambda \tag{6.5}$$

The next step is to relate the VCO phase deviation $\theta(t)$ to the VCO input. By definition, the frequency deviation of the VCO output is proportional to the VCO input signal so that

$$\frac{d\theta}{dt} = 2\pi K_d e_{vco}(t) \tag{6.6}$$

where K_d is the VCO constant and has units of Hz/v. Solving for $\theta(t)$ gives

$$\theta(t) = 2\pi K_d \int_{-\infty}^{t} e_{vco}(\lambda)\, d\lambda \tag{6.7}$$

Substituting (6.5) for $e_{vco}(t)$ yields

$$\theta(t) = G \int_{-\infty}^{t} \int_{-\infty}^{\infty} \sin[\phi(\lambda) - \theta(\lambda)] f(\tau-\lambda)\, d\lambda d\tau \tag{6.8}$$

where G is defined as the loop gain and is given by

$$G = \pi K_d \mu A_c A_v \tag{6.9}$$

Equation (6.8) is the nonlinear integral equation relating the phase deviation of the input $\phi(t)$ to the VCO phase deviation $\theta(t)$. Keep in mind that the impulse response of the loop filter, $f(t)$, is still arbitrary.

In developing system simulations, one should be careful about combining parameters as we did in (6.9). Combining terms is a valid step if the purpose of the simulation is to determine the input-output characteristics of the system or the characteristics of the system as a whole, such as the time required for a PLL to achieve phase lock. On the other hand, if the simulation is being performed to examine the waveforms present at the input or the output of various functional blocks within the system, the parameters that define G cannot be grouped together. It they are grouped together, the waveforms present at various points in the system will not be scaled properly or may not even be identifiable.

6.1.2 The Nonlinear Phase Model

It is apparent from (6.8) that the relationship between $\theta(t)$ and $\phi(t)$ does not depend in any way on the carrier frequency f_c, and therefore the carrier frequency need not be considered in the simulation model. We therefore seek a model that establishes the proper relationship between $\theta(t)$ and $\phi(t)$ without consideration of the carrier frequency. This model is shown in Figure 6.3 and is known as the nonlinear phase model of the PLL. It is a nonlinear model because of the sinusoidal nonlinearity and is a phase model because the model establishes the relationship between the input phase deviation and the VCO phase deviation rather than establishing the relationship between the actual loop input and VCO output signals as expressed by (6.1) and (6.2), respectively. It is important to remember that the input to the model illustrated in Figure 6.1 is the actual bandpass signal present in the system under study, while the input to the nonlinear phase model shown in Figure 6.3 is the phase deviation of the input bandpass signal. If the phase deviation of the input signal, $\phi(t)$, the carrier frequency, and the signal amplitude are known, (6.1) is completely determined. In like manner, if the VCO phase deviation, $\theta(t)$, the carrier frequency, and the signal amplitude are known, the VCO output expressed

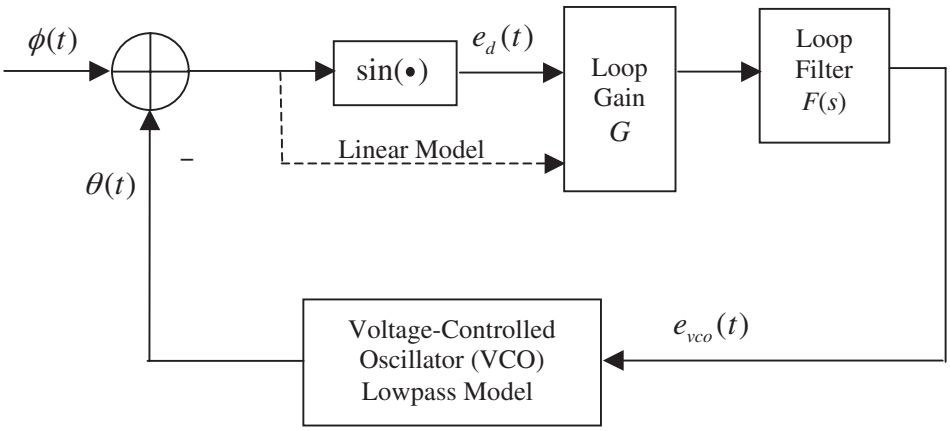


Figure 6.3 Nonlinear PLL phase model (linear model indicated by dotted line).

by (6.2) is completely determined. Thus, the nonlinear phase model expresses the relationship between the input phase and the VCO phase, which are the important quantities of interest. In simulation applications the nonlinear phase model pays additional dividends. Since the loop input and VCO output phase deviations are lowpass signals, they can be sampled at a much lower sampling rate than the signals expressed by (6.1) and (6.2), which are bandpass signals.

At this point another word is in order concerning the lowpass filter used to remove the second harmonic of the carrier produced by the multiplier present in the phase detector model. This filter is simply part of a conceptual model and, as mentioned previously, need not be present in the physical device. It is easily seen that this filter may be eliminated. Equation (6.7) shows that the VCO model is simply an integrator. Since an integrator is a lowpass filter, which has infinite gain at $f = 0$ and unity gain at $f = 1/(2\pi)$ Hz, the VCO will prevent the second harmonic of the carrier frequency from propagating around the loop and appearing at the VCO output.

Example 6.1. In this example a general technique for modeling phase detectors is examined. Although a sinusoidal phase detector is assumed throughout this chapter, it is easy to model a phase detector having an arbitrary characteristic. A general technique is to represent the function relating the output of the phase detector to the input of the phase detector by a Fourier series. This gives

$$e_d(t) = \sum_{k=1}^{N} c_k \sin\left[k\psi(t)\right] \tag{6.10}$$

where c_k represents the Fourier coefficients of the phase detector characteristic and $\psi(t) = \phi(t) - \theta(t)$ is the phase error. A given phase detector characteristic can be modeled to any required accuracy by adjusting the number of terms, N, used in the series expansion. Note that only odd terms are included in the series so that $e_d(t)$ will be an odd function of the phase error. As a simple example, the sinusoidal phase detector defined in Figure 6.2 is represented by

$$c_k = \begin{cases} \frac{1}{2}A_c A_v, & k = 1 \\ 0, & \text{otherwise} \end{cases} \tag{6.11}$$

where $N \geq 1$.

The MATLAB code for the phase detector described by (6.10) is easily written in vector form. The vector B is defined as

$$\mathrm{B} = [1 \quad 2 \quad 3 \quad \cdots \quad N]$$

and the vector of Fourier coefficients is

$$\mathrm{C} = [c_1 \quad c_2 \quad c_3 \quad \cdots \quad c_N]$$

Assuming that `pdin` and `pdout` are the input and output of the phase detector model, respectively, the phase detector output can be written

$$\texttt{pdout} = (\sin(\texttt{B} * \texttt{pdin})) * \texttt{C'}$$

where C' is the transpose of C. Since B and C are completely defined by the phase detector model, they are fixed and should be defined outside of the simulation loop.

We will later see that the most general method for simulating systems is to use the Monte Carlo technique, which requires that the simulation loop be executed a large number of times. When the Monte Carlo technique is used, very long simulation run times often result and it becomes important to use the most efficient algorithms possible. A vector formulation is therefore used to compute pdout in order to avoid the looping operation typically used to evaluate summations. ■

6.1.3 Nonlinear Model with Complex Input

Phase-locked loops are usually modeled so that the loop input is the phase deviation $\phi(t)$. When the PLL is used within a system, one often wishes to develop a model in which the loop input is a complex envelope lowpass signal $A\exp[j\phi(t)]$ representing (6.1). Such a model is illustrated in Figure 6.4 for a sinusoidal phase detector. Phase detectors having other characteristics are easily derived.

As illustrated in Figure 6.4 the input signal, in complex envelope form, is first passed through a bandpass limiter, which was discussed in Chapter 4, Example 4.10. (Here we assume that the parameter B, defined in Example 4.1, is set equal to one.)[1] The operation of the complex phase detector model should be clear from the expressions given in Figure 6.4 defining the signal at each point in the model. Note that the inputs to the multiplier are expressed in complex exponential form and that the imaginary part of the multiplier output gives $\sin[\phi(t)-\theta(t)]$.

6.1.4 The Linear Model and the Loop Transfer Function

If the phase error is small so that the linear approximation

$$\sin[\phi(t)-\theta(t)] \approx \phi(t)-\theta(t) \tag{6.12}$$

can be made, the loop equation, (6.8), becomes

$$\theta(t) = G\int_{-\infty}^{t}\int_{-\infty}^{\infty}[\phi(\lambda)-\theta(\lambda)]\,f(\tau-\lambda)\,d\lambda d\tau \tag{6.13}$$

This results in the linear phase model of the PLL, which is identical to the nonlinear phase model except that the sinusoidal nonlinearity is removed. This is shown by the dotted line in Figure 6.3.

Taking the Laplace transform of (6.13), recognizing that integration is equivalent to division by s and that convolution in the time domain is equivalent to multiplication in the frequency domain, yields

$$\Theta(s) = G[\Phi(s)-\Theta(s)]\frac{F(s)}{s} \tag{6.14}$$

[1] Recall from basic communication theory that a bandpass limiter is often used at the input to a PLL in order to supress variations in the input envelope. This is, for example, the case when PLLs are used for FM demodulation.

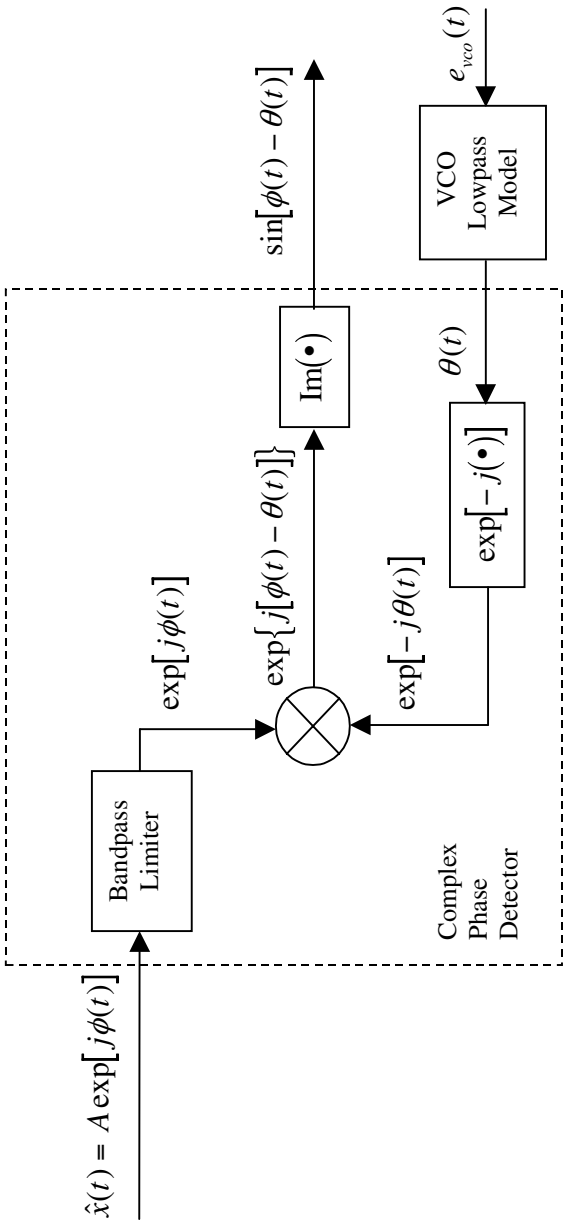


Figure 6.4 Complex lowpass phase detector model.

The transfer function $H(s)$ relating the VCO phase to the input phase is therefore given by

$$H(s) = \frac{\Theta(s)}{\Phi(s)} = \frac{GF(s)}{s + GF(s)} \tag{6.15}$$

It must be kept in mind that the transfer function given in (6.15) is based on a linear assumption and, in general, is not valid. Strictly speaking, transfer functions for nonlinear systems do not exist.

Even though simulation is typically used to study the nonlinear behavior of the PLL, linear models are useful for several purposes. First, as we will see in a following section, loop parameters are almost always defined in terms of the linear model. In addition, analytical analysis based on the linear model is usually easy and can often help verify (sanity check) the simulation results. Finally, linear models are often used to study tracking behavior for those applications in which the signal being tracked is varying at a rate slower than the loop bandwidth.

6.2 First-Order and Second-Order Loops

The acquisition and tracking behavior of a PLL depends in large measure upon the *order* of the loop, and for this reason we consider two choices for the loop filter transfer function. The order of a PLL implementation is equal to the number of finite poles in the transfer function $H(s)$ as given by (6.15). It follows that the order of a given PLL implementation exceeds the number of poles of $F(s)$ by one, with the extra pole resulting from the integration in the VCO model. We now briefly look at first-order (no loop filter) and second-order PLLs (first-order loop filter). Although the first-order PLL does not have desirable operating characteristics, as will be shown, we will use the first-order PLL to illustrate the mechanism by which the loop achieves phase lock.

6.2.1 The First-Order PLL

For the first-order PLL $F(s) = 1$ so that

$$f(t) = \delta(t) \tag{6.16}$$

Substitution into (6.8) yields

$$\theta(t) = G \int_{-\infty}^{t} \int_{-\infty}^{\infty} \sin[\phi(\lambda) - \theta(\lambda)] \, \delta(\tau - \lambda) \, d\lambda d\tau \tag{6.17}$$

Performing the integration on λ using the sifting property of the impulse (delta) function gives

$$\theta(t) = G \int_{-\infty}^{t} \sin[\phi(\tau) - \theta(\tau)] \, d\tau \tag{6.18}$$

Differentiating with respect to t yields the differential equation

$$\frac{d\theta}{dt} = G \sin\left[\phi\left(t\right) - \theta\left(t\right)\right] \tag{6.19}$$

Writing (6.19) in terms of the phase error $\psi\left(t\right) = \phi\left(t\right) - \theta\left(t\right)$ provides the differential equation relating the phase error and the input phase. This is

$$\frac{d\phi}{dt} - \frac{d\psi}{dt} = G \sin \psi\left(t\right) \tag{6.20}$$

We now determine the phase error for an input frequency step.

In order to study the response of a first-order PLL to a step in frequency of f_Δ Hz at time t_0 we let

$$\frac{d\phi}{dt} = 2\pi f_\Delta u\left(t - t_0\right) \tag{6.21}$$

so that (6.20), for $t > t_0$, becomes

$$\frac{d\psi}{dt} = 2\pi f_\Delta - G \sin \psi\left(t\right) \tag{6.22}$$

This gives the relationship between the frequency error and the phase error for $t > t_0$.

Equation (6.22) is illustrated in Figure 6.5 and is called the *phase plane equation* or, simply, the *phase plane,* and describes the dynamic behavior of the system. The phase plane has a number of important properties, and understanding a few of them provides insight into how, and under what conditions, the loop achieves lock. Note that the relationship between phase error and frequency error must satisfy (6.22) at each point in time. These time-dependent points are known as operating points. In the upper-half phase plane, the operating point moves from left to right and in the lower-half phase plane the operating point moves from right to left. This is easily seen. First we let

$$\frac{d\psi}{dt} \approx \frac{\Delta\psi}{\Delta t} \tag{6.23}$$

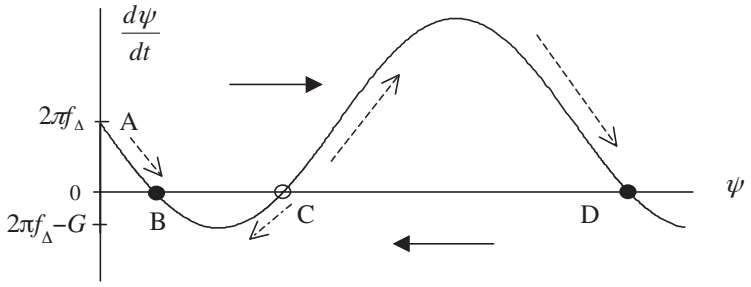


Figure 6.5 First-order phase plane.

where $\Delta\psi$ and Δt denote small increments in phase error and time, respectively. Clearly $\Delta t > 0$ for all t, since time always increases. Thus, $d\psi > 0$ in the upper-half phase plane and $d\psi < 0$ in the lower-half phase plane. The phase error therefore increases (moves from left to right) in the upper-half phase plane and the phase error decreases (moves from right to left) in the lower-half phase plane. This is illustrated by the solid arrows in Figure 6.5. A stationary operating point can lie only on the boundary between the upper-half phase plane and the lower-half phase plane. This, of course, denotes that the phase error is constant or, equivalently, that the frequency error is zero. An operating point is stable if, after a small perturbation, the operating point returns to its original location. If a small perturbation results in the operating point moving to a new position, the original operating point is called unstable. Thus points B and D in Figure 6.5 are stable operating points and point C is an unstable operating point. Movement of the operating points is indicated by the dotted lines in Figure 6.5.

It can be seen from (6.22) that if $2\pi f_\Delta < G$, the steady-state operating point is the stable point B given that the initial operating point due to the frequency step is A. At this point the frequency error is zero and the steady-state phase error is the solution of (6.22) with $d\psi/dt = 0$. This gives the steady-state phase error

$$\psi_{ss} = \sin^{-1}\left(\frac{2\pi f_\Delta}{G}\right) \tag{6.24}$$

As a final observation of the phase plane, note that if $2\pi f_\Delta > G$ there is no solution to (6.22) for zero frequency error $d\psi/dt$ and the operating point will move to the right for all time for $f_\Delta > 0$ and will move to the left for all time for $f_\Delta < 0$. The loop gain G therefore becomes the *lock range* for the first-order loop. Note from (6.15) that G is also the loop bandwidth (in rad/s) for the first-order PLL.

Example 6.2. As a simple example suppose that $f_\Delta = 5$ so that $2\pi f_\Delta = 31.42$. Also assume that the loop gain takes on two different values, namely, $G = 30$ and $G = 40$. The resulting phase planes are shown in Figure 6.6. The phase plane shows that for $f_\Delta = 5$, $G = 40$, and $2\pi f_\Delta < G$, the frequency error decreases to zero monotonically. There is no overshoot, since the system is first order. For $f_\Delta = 5$ and $G = 30$, $2\pi f_\Delta > G$, and (6.22) has no solution for zero frequency error. For this case phase lock will not be achieved and the system will forever oscillate. ■

The phase plane is made clearer by Figure 6.7, which shows the input frequency deviation, $d\phi/dt$, and the VCO frequency deviation, $d\theta/dt$, for $G = 30$ and $G = 40$. The resulting input frequency deviation and VCO frequency deviation are shown in Figure 6.7 for $2\pi f_\Delta = 31.42$ with $G = 30$ and $G = 40$. Note that for $G = 40$ the loop achieves phase lock while for $G = 30$ phase lock is not achieved and the VCO phase deviation oscillates forever.

We have seen that both the lock range of the first-order PLL and the loop bandwidth are determined by the parameter G. For most applications a large lock range and a small loop bandwidth are desired. This cannot be acccomplished with a first-order loop and, therefore, the first-order PLL is not usually practical. The

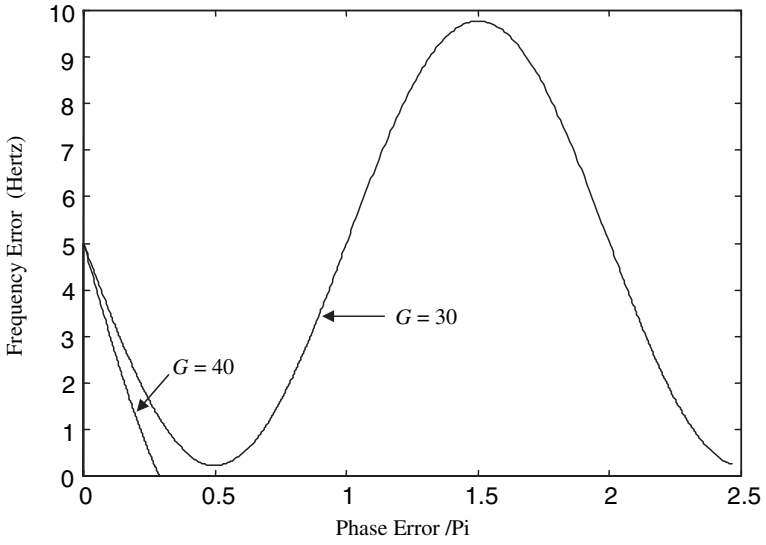


Figure 6.6 Phase plane plot for first-order PLL with $G = 30$ and $G = 40$.

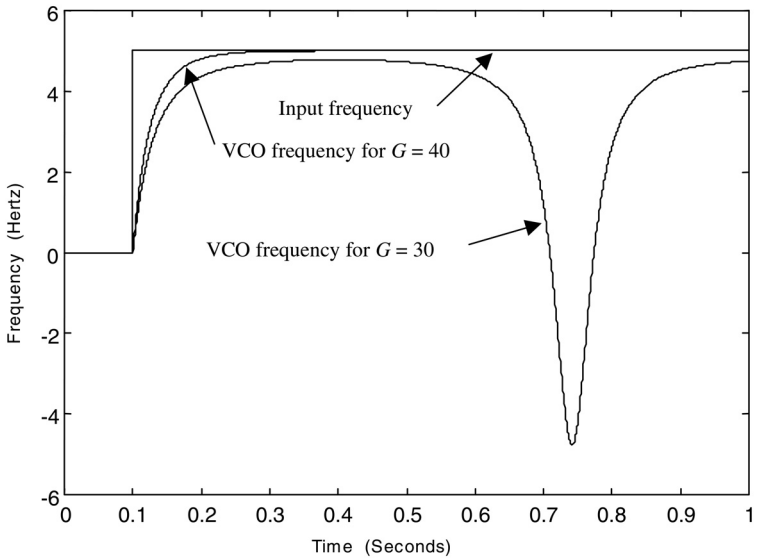


Figure 6.7 Input frequency and VCO frequency for $G = 30$ and $G = 40$.

second-order loop is a practical system for many applications and will provide aninteresting simulation case study.

6.2.2 The Second-Order PLL

We saw in the preceding section that the first-order PLL has a limited lock range. In addition, since the loop has only a single parameter, the ability to adjust loop parameters to meet a given set of operating specifications is severely limited. Improved operating characteristics and design capabilities are achieved by changing the loop filter so that a second-order PLL results.

The general form of the loop filter for a second-order PLL is

$$F(s) = \frac{s+a}{s+\lambda a} \tag{6.25}$$

The perfect second-order PLL is defined by $\lambda = 0$, which denotes a loop filter containing a perfect integration (pole at $s = 0$). In typical applications, $\lambda \ll 1$. Substituting the loop filter transfer function into (6.15) gives

$$H(s) = \frac{G(s+a)}{s^2 + (G+\lambda a)s + Ga} \tag{6.26}$$

for the loop linear-model transfer function.

Linear second-order systems are usually parameterized in terms of the system damping factor, denoted ζ, and the system natural frequency, denoted ω_n. The denominator of the transfer function is often referred to as the characteristic polynomial, which, for a second-order system, is expressed in the standard form

$$s^2 + 2\zeta\omega_n s + \omega_n^2$$

Equating the characteristic polynimial in (6.26) to the standard form yields

$$s^2 + (G+\lambda a)s + Ga = s^2 + 2\zeta\omega_n s + \omega_n^2 \tag{6.27}$$

In typical applications the PLL is designed for a given damping factor and natural frequency by specifying ζ and ω_n. The required physical parameters (in this case G and a) are then determined so that the design values of ζ and ω_n are achieved. Equating terms having like powers of s in (6.27) gives

$$2\zeta\omega_n = G + \lambda a \tag{6.28}$$

and

$$\omega_n^2 = Ga \tag{6.29}$$

Assuming that λ is a known constant, we may solve (6.29) for a and substitute the result in (6.28). This gives the quadratic equation

$$G^2 - 2\zeta\omega_n G + \lambda\omega_n^2 = 0 \tag{6.30}$$

which, upon solving for G yields

$$G = \zeta\omega_n + \omega_n\sqrt{\zeta^2 - \lambda} \tag{6.31}$$

This result, together with (6.29) gives

$$a = \frac{\omega_n}{\zeta + \sqrt{\zeta^2 - \lambda}} \tag{6.32}$$

Note that since a is a real parameter, λ must be less than ζ^2. Typical values of ζ^2 lie in the neighborhood of $1/2$ ($\zeta = 1/\sqrt{2}$ is a common choice) and, as previously mentioned, $\lambda \ll 1$. Note that for a perfect second-order PLL, $G = 2\zeta\omega_n$ and $a = \omega_n/2\zeta$.

6.3 Case Study: Simulating the PLL

We are now in a position to simulate a second-order PLL and use our very basic knowledge of PLL theory to sanity check the results. First, however, we pause to consider a simple architecture for the overall simulation program. The simulation model is then developed and the simulation is executed. Finally, the error sources in the simulation are briefly discussed.

6.3.1 The Simulation Architecture

It is often useful to divide the software for a simulation into several distinct programs as shown in Figure 6.8. Here we see three separate elements of the overall simulation program. These elements are designated the preprocessor, the simulation engine, and the postprocessor. These three programs perform three distinctly different functions and, if desired, can be developed in three different languages. Partitioning a simulation in this manner results in simulation code that is more easily developed, understood, and maintained. It is also efficient, since code segments can often be reused in different applications. This is especially true of the postprocessor.

The purpose of the preprocessor is to specify all parameters necessary to define the system under study (system parameters) and to set the parameters that manage execution of the simulation (intrinsic parameters). Example system parameters include items such as filter orders, filter types and bandwidths, amplifier gains, code rate, spreading ratios, carrier frequencies, bit rates, and signal-to-noise ratios. Intrinsic parameters include the sampling frequency, settle times (required to ensure that startup transients have decayed to negligible values), and the number

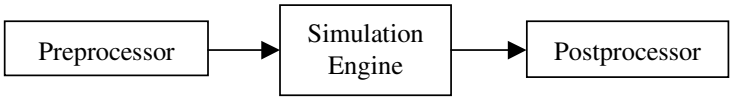


Figure 6.8 Simulation architecture.

of samples to be processed. In addition, the data to be passed to the postprocessor for analysis or plotting must be saved by the simulation and, therefore, the data files required by the postprocessor must be specified by the preprocessor so that the appropriate data is saved as the simulation is executed. An example might be a vector of waveform samples and a vector of sample times used in the postprocessor to calibrate the time axis of signal plots. Once all of the necessary information has been specified in the preprocessor, the data created within the preprocessor is typically written to a file so that it is available to the simulation engine and to the postprocessor. In the MATLAB examples to follow, the preprocessor data is written to the MATLAB workspace.

The simulation engine reads the data stored in the file (or in the workspace) created by the preprocessor and executes the simulation. The purpose of a simulation is, of course, to generate data for later investigation using the postprocessor. This may take the form of numbers (signal-to-noise ratios, bit error rates, code gain, etc.) or may be vectors of sample values for additional processing by the postprocessor. The information generated by the simulation engine is stored in files and passed to the postprocessor. In the case of MATLAB it is usually most convenient to simply leave the data generated by the simulation engine in the MATLAB workspace.

The postprocessor takes the data generated by the simulation engine and generates the final simulation products required by the user. These may include waveform plots, signal constellations, plots of the bit error rate as a function of E_b/N_0, the power spectral density at a point in the system under study, eye diagrams, and histograms. The list of possibilities is almost endless. Appropriately derived graphical displays can greatly facilitate an understanding of the system under study. We consider postprocessing and the generation of graphical displays in more detail in Chapter 8.

The postprocessor clearly requires a significant level of graphical support. This was one reason to relay on MATLAB for the applications presented here. Postprocessors are most useful when they are menu driven and provide a variety of signal processing and display options. While the simulations to follow in this chapter are quite simple, as are the preprocessor and the postprocessor, they serve to demonstrate the roles of each of these elements of a simulation.

As previously mentioned, the preprocessor, the simulation engine, and the postprocessor can be written in different languages. A recent project by one of the authors used Visual Basic for the preprocessor, C++ for the simulation engine (chosen for execution speed), and MATLAB for the postprocessor. MATLAB is frequently chosen for postprocessor development because of MATLAB's rich graphics library.

6.3.2 The Simulation

The simulation model is straightforward except, perhaps, for the loop filter. Since the loop filter transfer function, as defined by (6.25) is not a proper function,[2] long division is applied to yield

[2]Recall that a proper function is one in which the degree of the denominator polynomial exceeds the degree of the numerator polynomial by at least one.

$$F(s) = 1 + \frac{(1-\lambda)a}{s+\lambda a} = 1 + F_1(s) \tag{6.33}$$

Thus, $F(s)$ can be realized as a parallel combination of two transfer functions, the first of which is a constant and the second of which is $F_1(s)$. Clearly

$$F_1(s) = \frac{(1-\lambda)a}{s+\lambda a} = \frac{Y_1(s)}{X_1(s)} \tag{6.34}$$

where $Y_1(s)$ and $X_1(s)$ represent the Laplace transform of the output and input, respectively, of the subfilter $F_1(s)$. Cross-multiplying gives

$$sY_1(s) = (1-\lambda)\, aX_1(s) - \lambda a Y_1(s) \tag{6.35}$$

which in the time domain is

$$\frac{dy_1}{dt} = (1-\lambda)ax_1(t) - \lambda a y_1(t) \tag{6.36}$$

It follows that the loop filter is realized by the system illustrated in Figure 6.9.

The next step in developing the simulation code for the second-order PLL is to develop a signal-flow graph[3] for the system and to designate the points on the signal-flow graph at which the signals are to be defined. The resulting signal-flow graph is illustrated in Figure 6.10. Note that the loop filter model follows directly from Figure 6.9 with

$$a1 = (1-\lambda)a \tag{6.37}$$

and

$$a2 = \lambda a \tag{6.38}$$

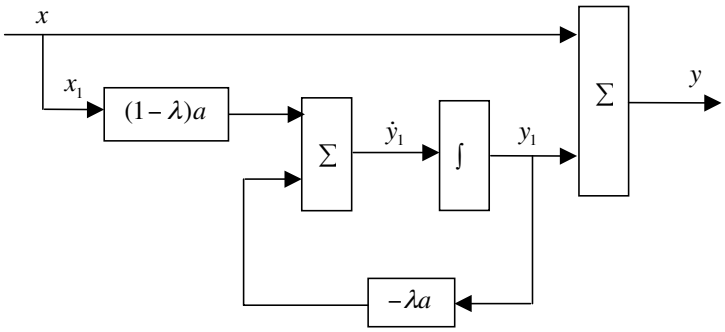


Figure 6.9 Loop filter model.

[3] A signal-flow graph is used rather than a block diagram simply for compactness.

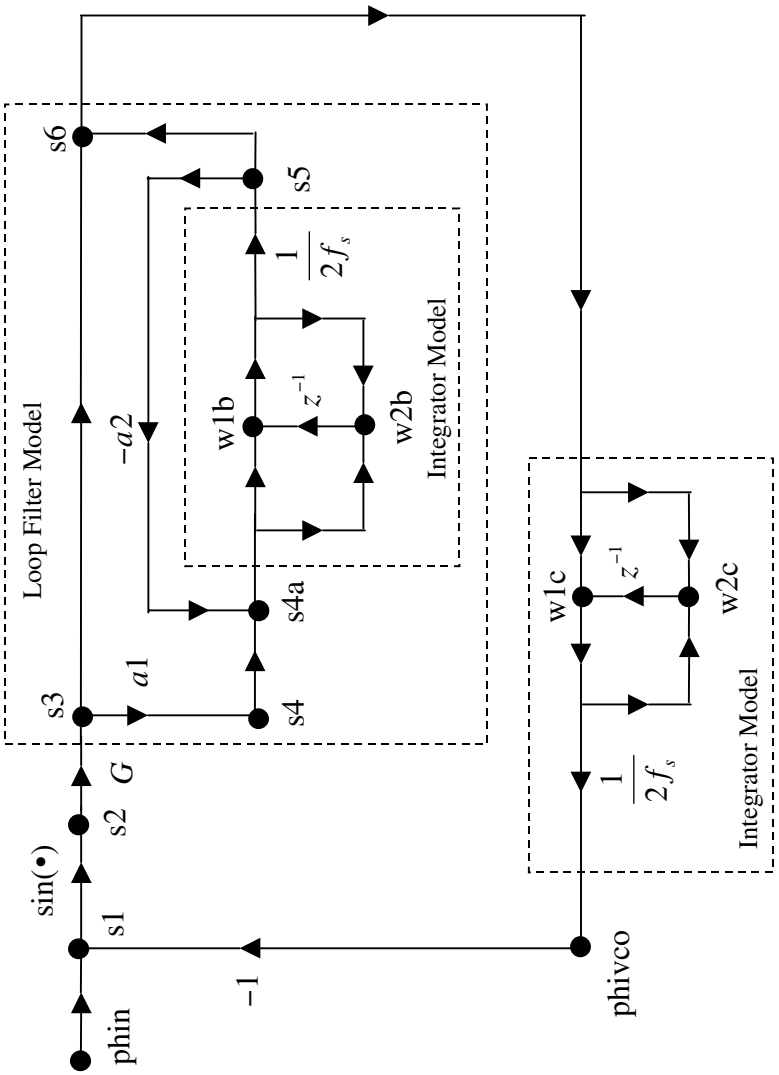


Figure 6.10 Signal-flow graph for second-order PLL.

The points at which the signals are to be defined in the simulation code are represented by the heavy black dots on the signal-flow graph. Note that we are using trapezoidal integration as developed in the previous chapter.

The simulation code follows directly from the signal-flow graph. Each sample value computed in the simulation loop carries an identification corresponding to the identification defined in the signal-flow graph. For example, the line of code

```
s2 = sin(s1)
```

defines the sinusoidal nonlinearity representing the phase detector, where s1 and s2 are defined in the signal-flow graph. All other lines of code in the simulation loop follow in a similar manner. Thus, the MATLAB code that realizes the main simulation loop is

```
% beginning of simulation loop
for i=1:npts
    s1 = phin-phivco;                % phase error
    s2 = sin(s1);                    % sinusoidal phase detector
    s3 = G*s2;
    s4 = a1*s3;
    s4a = s4-a2*s5;                  % loop filter integrator input
    w1b = s4a+w2b;                   % filter integrator (step 1)
    w2b = s4a+w1b;                   % filter integrator (step 2)
    s5 = w1b/twofs;                  % generate filter output
    s6 = s3+s5;                      % VCO integrator input
    w1c = s6+w2c;                    % VCO integrator(step 1)
    w2c = s6+w1c;                    % VCO integrator(step 2)
    phivco = w1c/twofs;              % generate VCO output
end
% end of simulation loop
```

The constant `twofs` is twice the sampling frequency, as required for the trapezoidal integrator, and is computed outside of the simulation loop.

The complete simulation code for the second-order PLL is given in Appendix A. Note that the code given in Appendix A differs slightly from the code given here, since vectors are developed for the input phase, the phase error, and the VCO frequency. These vectors are required by the postprocessor for plotting waveforms. The MATLAB code for the preprocessor and the postprocessor are given in Appendices B and C, respectively. Note that the postprocessor is menu driven. Menu-driven postprocessors are typical in simulation packages.

6.3.3 Simulation Results

An example simulation was performed assuming that the PLL input frequency deviation is a unit step at time t_0. Thus:

$$\frac{d\phi}{dt} = 2\pi f_\Delta u\,(t - t_0) \tag{6.39}$$

which is the same input that we assumed for the first-order PLL previously. The following PLL parameters are used:

$$\begin{aligned} \text{frequency step, } f_\Delta &= 40 \text{ Hz} \\ \text{loop natural frequency, } \omega_n/2\pi &= 10 \text{ Hz} \\ \text{relative pole offset, } \lambda &= 0.10 \\ \text{loop damping factor, } \zeta &= 1/\sqrt{2} \\ \text{sampling frequency, } f_s &= 5000 \text{ Hz} \\ \text{tstop} &= 0.8 \text{ s} \end{aligned} \tag{6.40}$$

These items are entered using the preprocessor.

The menu selections used in the example postprocessor (Appendix C) allow the simulation user to examine a number of items of interest with ease. These include the input frequency and VCO frequency, the frequency error, and the phase plane. For space considerations we illustrate only two postprocessor-generated plots here. The phase plane is illustrated in the top pane of Figure 6.11. The input frequency (the unit step of 40 Hz at $t_0 = 0.08$) and the VCO frequency (the waveform that oscillates) are illustrated in the bottom pane of Figure 6.11. It should be remembered that we are working with lowpass models. Thus, input frequency and VCO frequency actually refer to the input frequency deviation and the VCO frequency deviation from the nominal carrier frequency, f_c.

Note from the phase plane that the five cycles are slipped and that the steady-state error is slightly greater than 10π. From Figure 6.11 we see that the input frequency steps by 40 Hz as specified. The VCO frequency, however, oscillates through five cycles and then phase lock is achieved on the sixth cycle. This behavior is referred to as "cycle slipping" and is characteristic of nonlinear synchronizers when the change in input frequency significantly exceeds the natural frequency.

6.3.4 Error Sources in the Simulation

There are a number of error sources present in this simulation. These error sources, as discussed in Chapter 1, result both from mapping the physical device to an analytical model and from mapping the analytical model to the simulation model. These error sources are briefly discussed in the following paragraphs.

The Analytical Model

In developing an analytical model for a device, a number of assumptions are often made. These approximations typically involve idealizations of the various loop components and may not be valid if highly accurate simulations of a physical device are to be obtained. Where highly accurate simulation results are required, laboratory measurements are often necessary to determine sufficiently accurate component models. Various restrictions may also apply to signals within the loop that are not accounted for in a basic analytical model. For example, in the simulation just developed, the signal levels at any point in the loop were allowed to rise to any value dictated by the loop equations. In practice, however, the maximum value

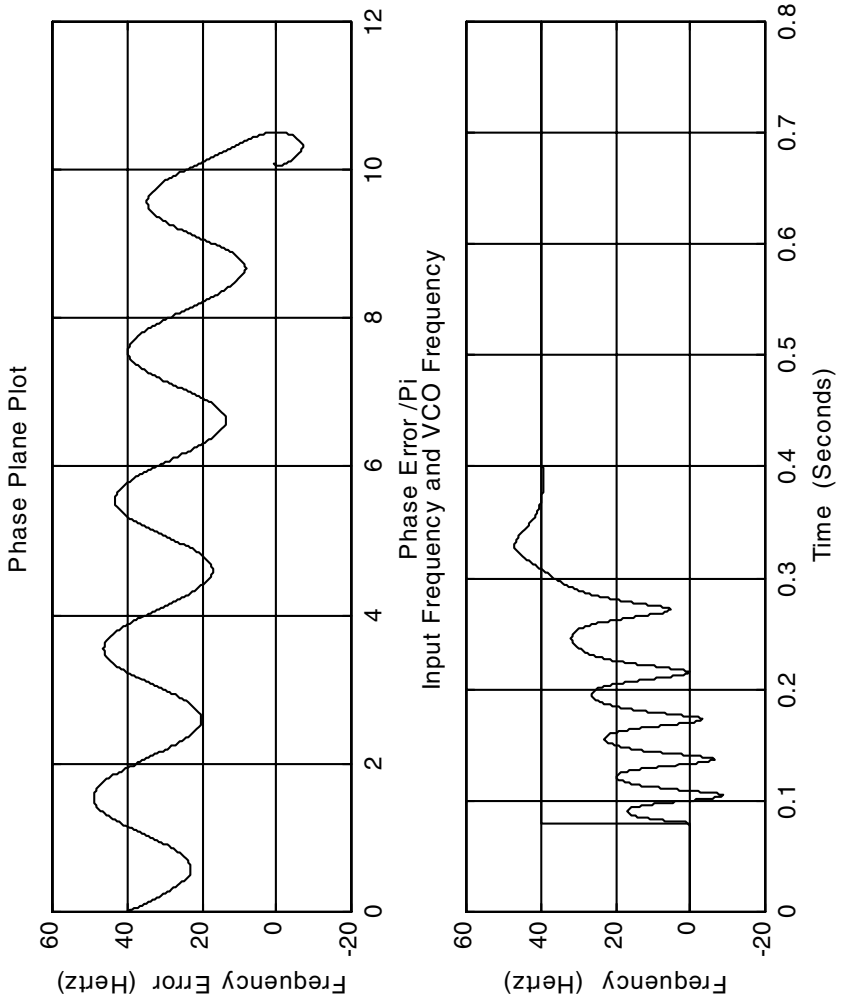


Figure 6.11 Simulation results.

of any signal in the loop will be constrained by the power supply voltages. Other approximations may also require consideration. Depending on the requirements of the simulation user, a more detailed analytical model than the one used here for a second-order PLL may be necessary.

The Simulation Model

There are a number of potential error sources associated with the process of mapping the analytical model to the simulation model. For example, while the physical system processes continuous-time signals, the simulation, of necessity, processes samples of continuous-time signals. Sampling, of course, leads to aliasing errors. In some cases quantizing errors must be considered. In addition, analog filters in the hardware or analytical model must be replaced by digital filters in the simulation model. As we previously saw, the digital filter never has the same amplitude and phase response as the analog filter it replaces. Errors resulting from frequency warping or aliasing must often be considered.

An important error source is present in all systems involving feedback. For example, the output of the phase detector in the analog (hardware) PLL is defined by

$$e_d(t) = \sin\left[\phi(t) - \theta(t)\right] \tag{6.41}$$

Ideally, a discrete-time model for this would be

$$e_d(nT) = \sin\left[\phi(nT) - \theta(nT)\right] \tag{6.42}$$

where $\phi\,(nT)$ is the "current" input to the PLL. However, $\theta\,(nT)$, the phase deviation at the output of the VCO, is not available, since $e_d(nT)$ is needed to compute it. The computational "deadlock" resulting from the interdependency between $e_d(nT)$ and $\theta\,(nT)$ is handled by using the previously computed value of the VCO phase deviation, $\theta\,((n-1)T)$, to compute $e_d(nT)$. In other words, the model defined by (6.42) is replaced by

$$e_d(nT) = \sin\left[\phi(nT) - \theta((n-1)T)\right] \tag{6.43}$$

Thus, a one-sample delay has been introduced in the feedback loop that is not present in the physical model or in the analytical model. While this one-sample delay may have a negligible effect on the accuracy of the simulation for small simulation step size (high sampling frequency), it may well have an effect for practical simulation step sizes. The effect of this one-sample delay is to induce a linear phase shift (a constant time delay) on the open-loop transfer function. This reduces the phase margin of the system and, if the simulation step size is sufficiently large, can drive the system into instability.

If a simulation is to have a given level of accuracy the simulation models must be based on analytical models having at least the same level of accuracy. If the physical devices being modeled exhibit significant variation across a group of similar devices, it is usually necessary to base the simulation models on measured data carefully obtained using accurately calibrated instrumentation. Nonlinear amplifiers often fall into this category, as do devices containing components that exhibit

significant aging effects. Other examples are models for channels that exhibit significant multipath effects.

Only a few error sources have been considered here, but it is important to realize that, as discussed in Chapter 1, the analytical model captures only a portion of the characteristics (hopefully the most important characteristics) of the physical device, and additional errors are incurred as the analytical model is mapped to a simulation model. These error sources are dependent upon both the system (analytical) model and the simulation methodology. The simulation user must identify these error sources and ensure that the accumulated effect of these sources is sufficiently small to ensure that the simulation results are valid. If this important step is neglected by the simulation user, the results will be suspect and of little value.

6.4 Solving Differential Equations Using Simulation

Back in the days prior to the widespread use of digital computers, the analog computer provided a convenient tool for solving differential equations. Analog computer techniques proved most useful for nonlinear and/or time-varying equations for which general analytical solution techniques do not exist. Since the "analog computer technique" could be applied to systems that were nonlinear or time-varying, or both, analog computers found widespread use. While the "analog computer solution" did not take the form of an equation, which is the standard form for expressing the solution to a differential equation, it did allow waveforms at various points present in a system to be plotted with ease. Parametric studies are easily conducted and it is therefore possible to gain significant insight into the operating characteristics of complex systems using the analog computer. Not surprisingly, the analog computer developed into a powerful simulation tool.

While the analog computer is a powerful tool, using it is not always a simple task. The basic component of the analog computer is the operational amplifier (OpAmp). OpAmps, especially those in use during the days in which the analog computer was enjoying popularity, were subject to drift. Frequent calibration was therefore necessary. In addition, the implementation of multiplication and division was difficult. Analog computers could only process signals having limited bandwidth and, as a result, frequency scaling and time scaling frequently had to be applied. When the digital computer became widely available and easy to use, the analog computer faded from use. However, much of the methodology developed for analog computers is applicable to simulation using digital computers. Thus, the body of knowledge developed for analog computers provides an important collection of simulation techniques applicable to digital computers. As an example, one of the early simulation programs, CSMP (Continuous System Modeling Program), developed by IBM for the System 360 family of computers, was basically a digital computer simulation of an analog computer. When the personal computer (PC) developed into a useful tool for scientists and engineers, various PC-based versions of CSMP were developed and several are still in use today. One of the early simulation programs targeted to communications systems, TOPSIM, was based on CSMP [3].

6.4.1 Simulation Diagrams

The first step in solving a differential equation using the analog computer technique is to develop the so-called analog computer simulation diagram. The process is straightforward. As a simple example, assume that a system is defined by the differential equation

$$\frac{d^2y}{dt^2} + a\frac{dy}{dt} + by = x(t) \tag{6.44}$$

where a and b are, for now, assumed constant. Solving the differential equation for d^2y/dt^2 gives

$$\frac{d^2y}{dt^2} = x(t) - a\frac{dy}{dt} - by \tag{6.45}$$

We then integrate d^2y/dt^2 to obtain dy/dt, and integrate dy/dt to obtain $y(t)$. These quantities can then be multiplied, or otherwise manipulated, by appropriate constants or functions to form the terms necessary for representation of the various terms in (6.45). Combining them appropriately yields the differential equation and the analog computer simulation diagram. A little thought shows that the analog computer simulation diagram for our example differential equation is as shown in Figure 6.12. While Figure 6.12 was developed for (6.45), which is a simple linear equation with constant coefficients and is solvable using a variety of methods, the beauty of the analog computer technique comes from the fact that the technique is applicable to nonlinear and time-varying systems of arbitrary order.

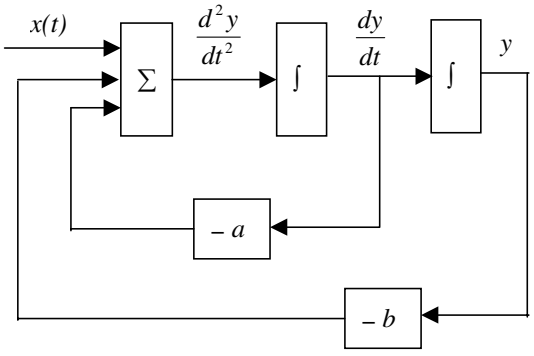


Figure 6.12 Simulation diagram for linear, time-invariant, second-order system.

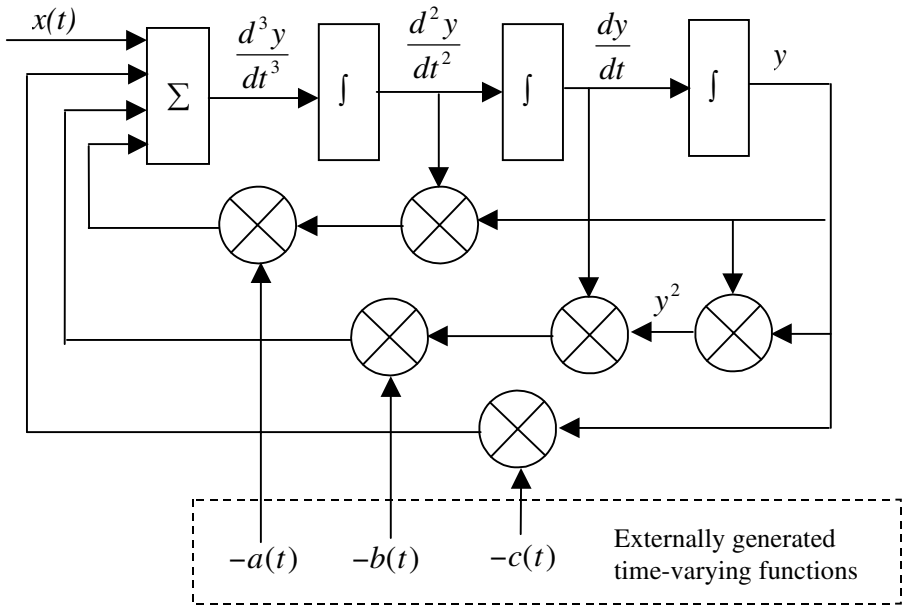


Figure 6.13 Simulation diagram for a nonlinear, time-varying, third-order system.

As an example of a system that is both nonlinear and time-varying, assume that a system of interest is defined by the differential equation

$$\frac{d^3y}{dt^3} + a(t)y(t)\frac{d^2y}{dt^2} + b(t)y^2(t)\frac{dy}{dt} + c(t)y(t) = x(t) \tag{6.46}$$

Solving for d^3y/dt^3 gives

$$\frac{d^3y}{dt^3} = x(t) - a(t)y(t)\frac{d^2y}{dt^2} - b(t)y^2(t)\frac{dy}{dt} - c(t)y(t) \tag{6.47}$$

from which the simulation diagram shown in Figure 6.13 immediately follows. The time-varying coefficients $a(t)$, $b(t)$, and $c(t)$ can be specified by appropriate equations or may be files of measured data collected experimentally.

6.4.2 The PLL Revisited

We now consider the simulation of the PLL using the differential equation approach. The first step is to derive the differential equation. From Figure 6.3 it follows that

$$E_{vco}(s) = GF(s)E_d(s) \tag{6.48}$$

Since the VCO in the PLL model illustrated in Figure 6.3 can be represented by an integrator, we have

$$\Theta(s) = \frac{1}{s} E_{vco}(s) \tag{6.49}$$

Substitution of (6.48) in (6.49) and using (6.25) for the loop filter gives

$$\Theta(s) = \frac{G}{s}\left(\frac{s+a}{s+\lambda a}\right) E_d(s) \tag{6.50}$$

We will simplify the problem slightly by assuming a "perfect" second-order loop for which $\lambda = 0$. This yields

$$s^2\Theta(s) = sGE_d(s) + GaE_d(s) \tag{6.51}$$

Since multiplication by s is equivalent to differentiation in the time domain, we have the differential equation

$$\frac{d^2\theta}{dt^2} = G\frac{d}{dt}e_d(t) + Gae_d(t) \tag{6.52}$$

By definition

$$e_d(t) = \sin\psi(t) \tag{6.53}$$

and

$$\frac{d^2\psi}{dt^2} = \frac{d^2\phi}{dt^2} - \frac{d^2\theta}{dt^2} \tag{6.54}$$

Equation (6.52) can then be written

$$\frac{d^2\psi}{dt^2} + G\frac{d}{dt}\{\sin\psi(t)\} + Ga\sin\psi(t) = \frac{d^2\phi}{dt^2} \tag{6.55}$$

where $\psi(t)$ is the PLL phase error and $\phi(t)$ is the phase deviation of the input signal. Prior to developing the analog computer simulation diagram, we first write the preceding equation in the form

$$\frac{d^2\psi}{dt^2} + G\cos\psi(t)\frac{d\psi}{dt} + Ga\sin\psi(t) = \frac{d^2\phi}{dt^2} \tag{6.56}$$

The simulation diagram for this system is illustrated in Figure 6.14. We desire to examine this system with a step change in the input frequency deviation at $t = t_0$. Thus, as before, we let

$$\frac{d\phi}{dt} = 2\pi f_\Delta u\,(t - t_0) \tag{6.57}$$

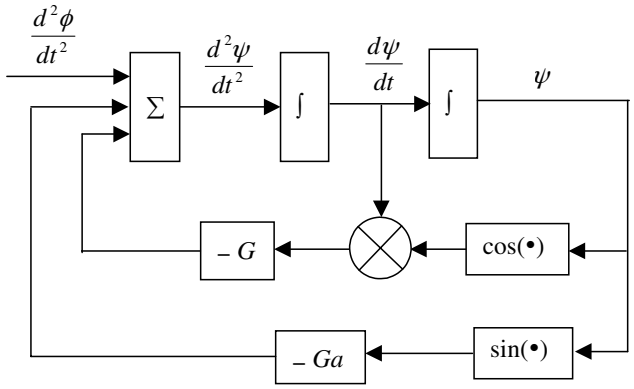


Figure 6.14 Simulation diagram for a perfect second-order PLL based on the differential equation.

so that

$$\frac{d^2\phi}{dt^2} = 2\pi f_\Delta \delta\left(t - t_0\right) \tag{6.58}$$

Thus the input shown in Figure 6.14 is an impulse.

It is often desirable to place the input at a point in the simulation diagram that allows for a more straightforward simulation. For example, as previously shown, the input in Figure 6.14 is an impulse. Integrating the impulse and moving it to the right of the first integrator as shown in Figure 6.15 allows us to use the step

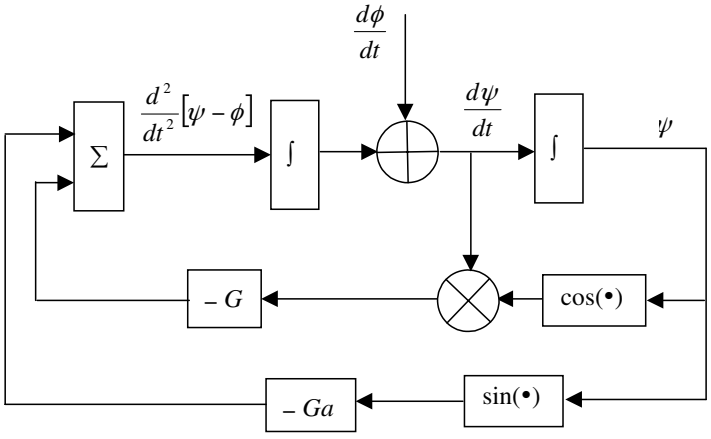


Figure 6.15 Simulation diagram with a frequency step as the input.

function defined by (6.57) as an "equivalent input." Simple modifications such as this can often significantly simplify a simulation.

Example 6.3. We end this chapter with a simulation of a system that is both nonlinear and time-varying. Assume that a system of interest is defined by the differential equations

$$\frac{d^2y}{dt^2} + 3\,|y(t)|\,\frac{dy}{dt} + 9y(t) = 4\exp(-t/2), \qquad t < 20 \tag{6.59}$$

and

$$\frac{d^2y}{dt^2} + 3\frac{dy}{dt} + 9y(t) = 4\exp(-t/2), \qquad t \geq 20 \tag{6.60}$$

The system is nonlinear because of the $|y(t)|\,\frac{dy}{dt}$ term in the equation defining the system for $t < 20$ and is time-varying, since the form of the differential equation is time-dependent. The fact that the system is characterized by two separate differential equations can be implemented by a switch that changes position at $t = 20$ seconds as shown in Figure 6.16. In position A the system is nonlinear, while in position B the system is linear. The phase plane will illustrate that, as expected, the response of a nonlinear system is quite different from the response of a linear system. As we know, solving nonlinear differential equations is a formidable task and we therefore resort to simulation. The MATLAB program to realize the example system is given in Appendix D. Note the similarity between this system and the second-order PLL. Since both systems are second order, two integrators are required in both cases. The only significant difference in the two systems is the equation at the output of the summing junction, which closes the simulation loop.

Executing the simulation program given in Appendix D results in the phase plane illustrated in Figure 6.17. Note that the phase trajectory both begins and

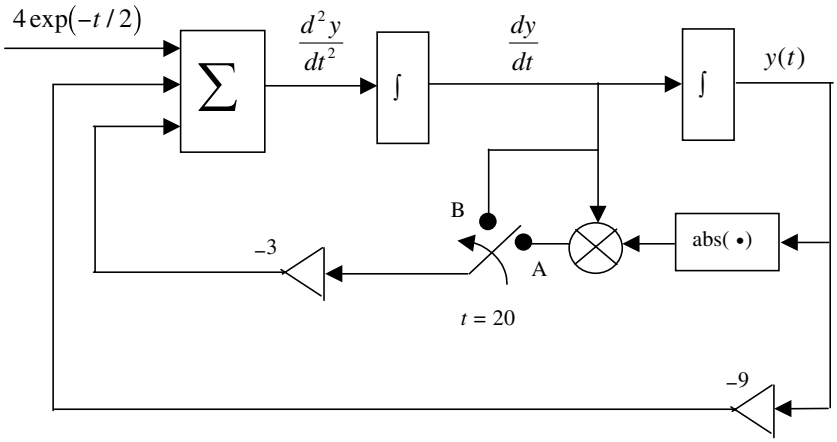


Figure 6.16 Example nonlinear time-varying system.

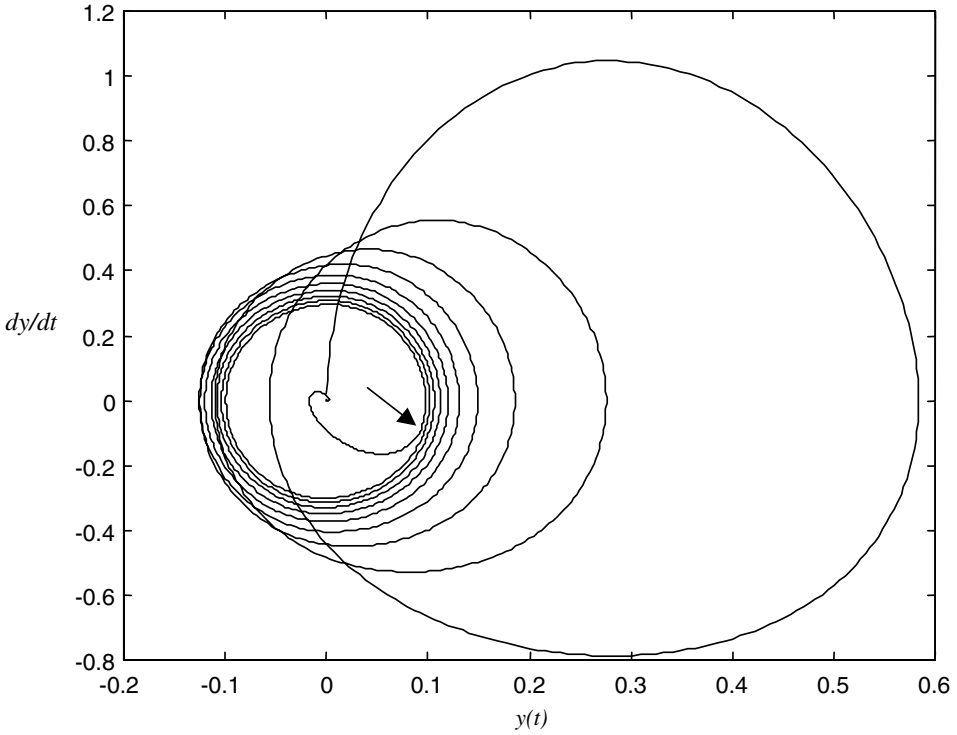


Figure 6.17 Phase plane for the example simulation.

ends at the origin (0,0). The point at which the switch changes, $t = 20$ seconds, is approximately indicated by the arrow. One sees that the phase plane approaches a limit cycle as indicated by the convergence of the phase trajectory to a closed loop. A system operating in the limit-cycle mode is an oscillator. The frequency of oscillation and the waveshape of $y(t)$, or its derivative, can be determined by plotting the time-domain waveforms. Oscillation will continue as long as the switch is in position A.

Moving the switch to position B results in a linear system. At the time the switch is moved to position B ($t = 20$), the input is negligible [actually $\exp(-10)$] and the system response closely approximates that of an unexcited second-order system, with the initial conditions given by the values of $y(t)$ and $\frac{dy}{dt}$ at the time the switch is moved from position A to position B. ■

Throughout this chapter we have used the trapezoidal numerical integration rule to approximate true integration. This was done for simplicity and, as mentioned earlier, the trapezoidal rule works quite well if the sampling frequency is sufficiently high. Other integration rules will be the subject of later study when a more general study of nonlinear systems is presented. (See Chapter 12.)

6.5 Summary

The trapezoidal integration rule, developed in the preceding chapter, gives us the capability to develop numerical solutions to differential equations. Since all lumped parameter systems can be described by a differential equation, it follows that a wide variety of systems can be simulated using trapezoidal integration to approximate continuous-time integration. Since the phase-locked loop is a fundamental building block of many communications systems, it provides a useful case study for illustrating a number of simulation concepts.

In order to develop a simulation model of the PLL, an analytical model must first be derived. The analysis required for model development also leads to an understanding of the basic operational characteristics of the PLL and, in turn, makes the simulation results understandable. Thus, the first two sections of this chapter focused on the development of an analytical model of the PLL. We saw that the major components of the PLL are the phase detector, the loop filter, and the voltage controlled oscillator. Two different phase detector models were developed, the sinusoidal phase detector and a generalized phase detector model based on the Fourier series expansion of the phase detector characteristic. It was shown that the VCO is an FM modulator. The loop filter and the loop gain control the PLL dynamics.

In general, the PLL is a nonlinear system with the nonlinearity resulting from the phase detector. Since the PLL is a nonlinear system, analysis using traditional analytical methods is difficult and we therefore turn to simulation. If, however, the phase error is small, system behavior becomes linear and the techniques of linear system theory are sufficient for analysis. Operation as a linear system is often referred to as tracking and nonlinear operation is referred to as acquisition. We therefore have two important PLL models, the nonlinear model, which is general, and the linear model, which can be used for those cases in which the phase error is small. The linear model is used to define the damping factor and the natural frequency of the second-order PLL.

Using a first-order PLL, for which the loop filter is zero order, the concept of a phase plane was introduced. The characteristics of the phase plane illustrated the mechanism by which the PLL achieves phase lock. It was shown that the first-order PLL has a finite lock range limited by the loop gain. As a result, the first-order PLL is not suitable for many applications. The second-order PLL has an infinite lock range and is widely used.

Prior to performing a simulation study of the second-order PLL, an architecture for the simulation code was developed. The simulation code is conveniently divided into three separate programs, a preprocessor, a simulation engine, and a postprocessor. The preprocessor is a short program through which the parameters necessary for defining the system under study and the parameters required to manage execution of the simulation are defined. The simulation engine contains the code for executing the simulation. The postprocessor takes the data produced by the simulation engine and generates the output required by the simulation user. Partitioning the code in this way makes the simulation software easier to maintain

and understand. In addition, the three code segments can be developed in different languages, which makes it possible to match the choice of language to the task of the program.

An example simulation was performed for a second-order PLL. The cycle slipping phenomena was clearly observable from both the phase plane and the time-domain waveforms.

After considering the PLL, attention was turned to the more general problem of simulating systems from the differential equation. By solving for the highest derivative and using repeated integration, we saw that all terms in the differential equation can be generated and, therefore, the differential equation could be solved. This approach was widely used as a simulation technique for analog computers. The analog computer technique for simulating a system is very general and can be used for systems that are nonlinear and/or time-varying.

6.6 Further Reading

A large number books have been written that treat the phase-locked loop in detail. A small sampling of this list includes the following:

R. E. Best, *Phase-Locked Loops: Theory, Design and Applications*, New York: McGraw-Hill, 1984.

A. Blanchard, *Phaselock Loops*, New York: Wiley, 1976.

F. M. Gardner, *Phaselock Techniques*, 2nd ed., New York: Wiley, 1981.

D. R. Stephens, *Phase-Locked Loops for Wireless Communications*, Boston: Kluwer Academic Publishers, 1998.

A. J. Viterbi, *Principles of Coherent Communications*, New York: McGraw-Hill, 1966.

6.7 References

1. R. E. Ziemer and W. H. Tranter, *Principles of Communications; Systems, Modulation and Noise*, 5th ed., New York: Wiley, 2002.

2. H. Meyr and G. Ascheid, *Synchronization in Digital Communications, Volume 1: Phase-, Frequency-Locked Loops, and Amplitude Control*, New York: Wiley Interscience, 1990.

3. K. Sam Shanmugan, "An Update on Software Packages for Simulation of Communication Systems (Links)," *IEEE Journal on Selected Areas in Communications*, Vol. 6, No. 1, January 1988, pp. 5–12. (This paper contains a discussion of many of the early simulation languages and a good set of references on these early tools.)

6.8 Problems

6.1 Develop and run a simulation to verify the results presented in Figures 6.6 and 6.7.

6.2 Using simulation, determine the steady-state phase error for a first-order PLL with $G = 40$ (see Figure 6.6). Estimate this value using the linear model. How do they compare? Repeat for $G = 50$ and $G = 100$. What do you conclude?

6.3 The results given in Figure 6.11 were obtained by executing the simulation program given in Appendix A together with the preprocessor and postprocessor given in Appendices B and C, respectively. A pole offset λ of 0.10 was assumed. An imperfect loop typically slips more cycles than a perfect loop and therefore takes longer to achieve phase lock. Examine the impact of the pole offset by executing the program assuming a perfect loop ($\lambda = 0$). Except for the pole offset use the same parameters as given in (6.40). How many cycles are slipped by the perfect loop? By how much is the lock time reduced?

6.4 Using simulation, determine the steady-state phase error for a second-order PLL with $\lambda = 0.10$ (see Figure 6.11 and consider using the `zoom on` MATLAB command). Estimate this value, mod(2π), using the linear model. How do they compare? Repeat for $\lambda = 0$, $\lambda = 0.05$, and $\lambda = 0.2$. What do you conclude?

6.5 In this problem we wish to examine the effect of the sampling frequency on the simulation program given in Appendix A. As a first step execute the simulation using the parameters given in (6.40). Repeat the simulation using sampling frequencies of 50 Hz, 100 Hz, and 500 Hz. Comment on the results. Repeat the simulation using a sampling frequency of 10,000 Hz. What does this last simulation tell you? How can you determine that a suitable sampling frequency is being used?

6.6 In this problem we consider the use of the technique described in Example 6.1 to model the triangular phase detector characteristic illustrated in Figure 6.18. Determine the values of the `B` and `C` vectors for $N = 11$. Modify the simulation code given in Appendix A to include this phase detector model.

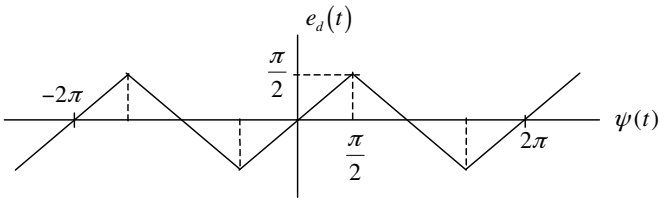


Figure 6.18 Triangular-wave phase detector characteristic.

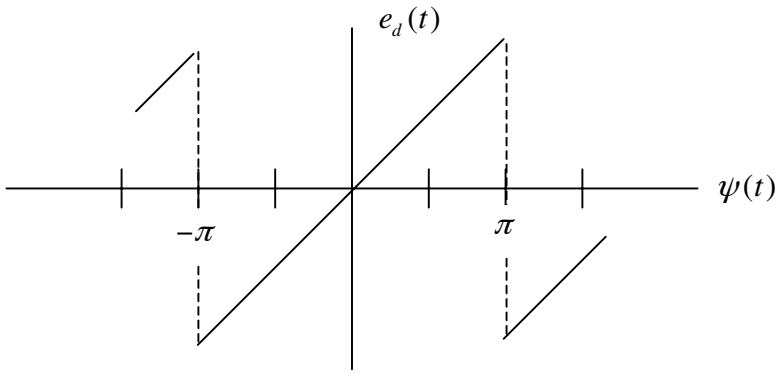


Figure 6.19 Sawtooth-wave phase detector characteristic.

6.7 Repeat the preceding problem for the sawtooth-wave phase detector input-output characteristic illustrated in Figure 6.19. As before, the slope of the input-output characteristic is +1 at all points except at odd multiples of π and it is periodic with period 2π.

6.8 Using the results of Problem 6.6, simulate a second-order PLL with a triangular-wave phase detector characteristic. Use the PLL parameters given in (6.40) that provided the results illustrated in Figure 6.11 for a sinusoidal phase detector. Compare the results obtained with the triangular-wave phase detector with those given in Figure 6.11.

6.9 Using the results of Problem 6.7, simulate a second-order PLL with a sawtooth-wave phase detector characteristic. As in the previous problem use the PLL parameters given in (6.40) that provided the results illustrated in Figure 6.11 for a sinusoidal phase detector. Compare the results obtained with a sawtooth-wave phase detector with those given in Figure 6.11.

6.10 Develop a signal-flow graph corresponding to Figure 6.15 and label the nodes that would be used in a simulation as was done in Figure 6.10. Using the signal-flow graph, write a MATLAB program for simulating the system and displaying the results. The preprocessor and postprocessor given in Appendices B and C, respectively, can be used. Execute the simulation for $\Delta f = 40$, $f_n = 10$, $\zeta = 1/\sqrt{2}$, and $\lambda = 0$. Compare the results to that obtained by executing the simulation code given in Appendix A.

6.11 We previously saw that in order to break the computational deadlock resulting from the presence of feedback, a one sample delay had to be inserted in the feedback loop. This gave rise to the following expression defining the relationship between input phase, the VCO phase, and the phase detector output:

$$e_d(nT) = \sin\left[\phi(nT) - \theta((n-1)T)\right]$$

In many models of PLLs used at high frequencies it is appropriate to use the model

$$e_d(nT) = \sin[\phi(nT) - \theta((n-1)T - kT)]$$

where kT represents "transport delay." Transport delay results from the finite time required for a signal to propagate from one functional block to another functional block in a hardware implementation. In principle, transport delay accounts for the propagation delay around the loop that is not accounted for by the sum of the group delays of the functional blocks.

Modify the simulation program given in Appendix A to include the effects of transport delay. Execute the simulation using the parameters defined by (6.40) and various values of transport delay. Since transport delay increases the time required to achieve phase lock, the value of `tstop` may have to be increased. At what value of k does instability result?

6.12 For the nonlinear time-varying system investigated in Example 6.3, determine the frequency of oscillation assuming that the switch has been in position "A" for a long time. Plot the waveforms for $y(t)$ and dy/dt describing limit cycle behavior.

6.13 In Example 6.3 the limit cycle, because of the default axis scaling used by MATLAB very nearly approximates a circle in Figure 6.17. However, since the abscissa and the ordinate in Figure 6.17 are not scaled equally, the limit cycle is not a circle but an oval. Modify the program given in Appendix D so that the true shape of the limit cycle is revealed. What is the equation of the limit cycle? Can you justify this equation?

6.14 Two systems are to be compared assuming a common input. One system is defined by the differential equation

$$\frac{d^2\theta}{dt^2} + |\theta| \frac{d\theta}{dt} + \theta = 0$$

and the second system is defined by

$$\frac{d^2\theta}{dt^2} + \theta \left| \frac{d\theta}{dt} \right| + \theta = 0$$

By solving both differential equations subject to initial conditions, $\theta(0) = 1$, $dy(t)/dt|_{t=0} = 0$, and $d^2y(t)/dt^2|_{t=0} = 0$, compare the responses of the two systems.

6.15 A Costas PLL, which is used for demodulation of DSB and PSK signals, is illustrated in Figure 6.20. The input signal is assumed to be

$$x(t) = A_c \cos[2\pi f_c t + \phi(t)]$$

(a) Develop a lowpass model using the input phase deviation $\phi(t)$ as the input to the model.

(b) Repeat using the lowpass complex envelope $\widetilde{x}(t)$ as the input to the model.

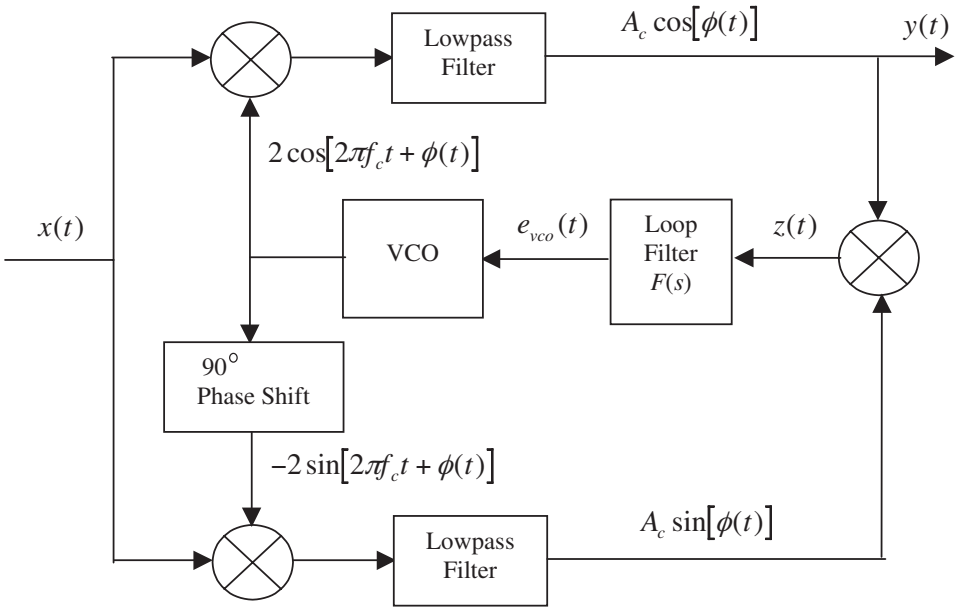


Figure 6.20 Costas PLL.

6.9 Appendix A: PLL Simulation Program

```
% File: c6_PLLsim.m
w2b = 0; w2c = 0; s5 = 0; phivco = 0;       % initialize
twopi = 2*pi;                               % define 2*pi
twofs = 2*fs;                               % define 2*fs
G = 2*pi*fn*(zeta+sqrt(zeta*zeta-lambda));  % set loop gain
a = 2*pi*fn/(zeta+sqrt(zeta*zeta-lambda));  % set filter parameter
a1 = a*(1-lambda); a2 = a*lambda;           % define constants
phierror = zeros(1,npts);                   % initialize vector
fvco = zeros(1,npts);                       % initialize vector

% beginning of simulation loop

for i = 1:npts
    s1 = phin(i) - phivco;              % phase error
    s2 = sin(s1);                       % sinusoidal phase detector
    s3 = G*s2;
    s4 = a1*s3;
    s4a = s4-a2*s5;                     % loop filter integrator input
    w1b = s4a+w2b;                      % filter integrator (step 1)
    w2b = s4a+w1b;                      % filter integrator (step 2)
    s5 = w1b/twofs;                     % generate filter output
    s6 = s3+s5;                         % VCO integrator input
    w1c = s6+w2c;                       % VCO integrator (step 1)
    w2c = s6+w1c;                       % VCO integrator (step 2)
    phivco = w1c/twofs;                 % generate VCO output
    phierror(i) = s1;                   % build phase error vector
    fvco(i) = s6/twopi;                 % build VCO input vector
end

% end of simulation loop
freqerror = fin-fvco;                   % build frequency error vector
% End of script file.
```

6.10 Appendix B: Preprocessor for PLL Example Simulation

```
% File: c6_PLLpre.m
clear all                                % be safe
disp(' ')                                % insert blank line
fdel = input('Enter the size of the frequency step in Hertz > ');
fn = input('Enter the loop natural frequency in Hertz > ');
lambda = input('Enter lambda, the relative pole offset > ');
disp(' ')
disp('Accept default values:')
disp(' zeta = 1/sqrt(2),')
disp(' fs = 200*fn, and')
disp(' tstop = 1')
dtype = input('Enter y for yes or n for no > ','s');
if dtype == 'y'
   zeta = 1/sqrt(2);
   fs = 200*fn;
   tstop = 1;
else
   zeta = input('Enter zeta, the loop damping factor > ');
   fs = input('Enter the sampling frequency in Hertz > ');
   tstop = input('Enter tstop, the simulation runtime > ');
end
npts = fs*tstop+1;                       % number of simulation points
t = (0:(npts-1))/fs;                     % default time vector
nsettle = fix(npts/10);                  % set nsettle time as 0.1*npts
tsettle = nsettle/fs;                    % set tsettle
%
% The next two lines establish the loop input frequency and phase
% deviations.
%
fin = [zeros(1,nsettle),fdel*ones(1,npts-nsettle)];
phin = [zeros(1,nsettle),2*pi*fdel*t(1:(npts-nsettle))];
disp(' ')                                % insert blank line
%
% End of script file.
```

6.11 Appendix C: PLL Postprocessor

6.11.1 Main Program

```
% File: c6_PLLpost.m
kk = 0;
while kk == 0
   k = menu('Phase Lock Loop Postprocessor',...
   'Input Frequency and VCO Frequency','Input Phase and VCO Phase',...
   'Frequency Error','Phase Error','Phase Plane Plot',...
   'Phase Plane and Time Domain Plots','Exit Program');
   if k == 1
      plot(t,fin,t,fvco)
      title('Input Frequency and VCO Frequency')
      xlabel('Time - Seconds'); ylabel('Frequency - Hertz');
      pause
   elseif k ==2
      pvco = phin-phierror;
      plot(t,phin,t,pvco)
      title('Input Phase and VCO Phase')
      xlabel('Time - Seconds'); ylabel('Phase - Radians');
      pause
   elseif k == 3
      plot(t,freqerror);
      title('Frequency Error')
      xlabel('Time - Seconds'); ylabel('Frequency Error - Hertz');
      pause
   elseif k == 4
      plot(t,phierror);
      title('Phase Error')
      xlabel('Time - Seconds'); ylabel('Phase Error - Radians');
      pause
   elseif k == 5
      ppplot
   elseif k == 6
      subplot(211);
      phierrn = phierror/pi;
      plot(phierrn,freqerror);
      grid;
      title('Phase Plane Plot');
      xlabel('Phase Error /Pi'); ylabel('Frequency Error - Hertz');
      subplot(212)
      plot(t,fin,t,fvco); grid
      title('Input Frequency and VCO Frequency')
      xlabel('Time - Seconds'); ylabel('Frequency - Hertz');
      subplot(111)
```

```
   elseif k == 7
      kk = 1;
   end
end
% End of script file.
```

6.11.2 Called Routines

Script File ppplot.m

```
   % ppplot.m is the script file for plotting phase plane plots. If
   % the phase plane is constrained to (-pi,pi) ppplot.m calls
   % pplane.m.
   %
   kz = 0;
   while kz == 0
   k = menu('Phase Plane Options',...
   'Extended Phase Plane',...
   'Phase Plane mod(2pi)',...
   'Exit Phase Plane Menu');
   if k == 1
      phierrn = phierror/pi;
      plot(phierrn,freqerror,'k')
      title('Phase Plane Plot')
      xlabel('Phase Error /Pi'); ylabel('Frequency Error - Hertz')
      grid
      pause
   elseif k == 2
      pplane(phierror,freqerror,nsettle+1)
      pause
   elseif k == 3
      kz = 1;
   end
end
% End of script file.
```

Function pplane.m

```
function [] = pplane(x,y,nsettle)
% Plots the phase plane with phase in the range (-pi,pi)
ln = length(x);
maxfreq = max(y);
minfreq = min(y);
close                                    % Old figure discarded
axis([-1 1 1.1*minfreq 1.1*maxfreq]);    % Establish scale
hold on                                  % Collect info for new fig
j = nsettle;
```

```
while j < ln
   i = 1;
   while x(j) < pi & j < ln
      a(i) = x(j)/pi;
      b(i) = y(j);
      j = j+1;
      i = i+1;
   end
   plot(a,b,'k')
   a = [];
   b = [];
   x = x - 2*pi;
end
hold off
title('Phase-Plane Plot')
xlabel('Phase Error / Pi'); ylabel('Frequency Error in Hertz')
grid
% End of script file.
```

6.12 Appendix D: MATLAB Code for Example 6.3

```
% File: c6_nltvde.m
w2b = 0; w2c = 0;                  % initialize integrators
yd = 0; y = 0;                     % initialize differential equation
tfinal = 50;                       % simulation time
fs = 100;                          % sampling frequency
delt = 1/fs;                       % sampling period
npts = 1+fs*tfinal;                % number of samples simulated
ydv = zeros(1,npts);               % vector of dy/dt samples
yv = zeros(1,npts);                % vector of y(t) samples

% beginning of simulation loop
for i=1:npts
   t = (i-1)*delt;                         % time
   if t < 20
      ydd = 4*exp(-t/2)-3*yd*abs(y)-9*y; % de for t<20
   else
      ydd = 4*exp(-t/2)-3*yd-9*y;         % de for t>=20
   end
   w1b = ydd+w2b;                          % first integrator - step 1
   w2b = ydd+w1b;                          % first integrator - step 2
   yd = w1b/(2*fs);                        % first integrator output
   w1c = yd+w2c;                           % second integrator - step 1
   w2c = yd+w1c;                           % second integrator - step 2
   y = w1c/(2*fs);                         % second integrator output
   ydv(1,i) = yd;                          % build dy/dt vector
   yv(1,i) = y;                            % build y(t) vector
 end}
% end of simulation loop

plot(yv,ydv)                       % plot phase plane
xlabel('y(t)')                     % label x axis
ylabel('dy/dt')                    % label y zaxis
% End of script file.
```

Chapter 7

GENERATING AND PROCESSING RANDOM SIGNALS

To this point we have been concerned with deterministic signals in simulations. In all communication systems of practical interest, random effects such as channel noise, interference, and fading, degrade the information-bearing signal as it passes through the system from information source to the final user. Accurate simulation of these systems at the waveform level requires that these random effects be modeled accurately. Therefore, algorithms are required to produce these random effects. The fundamental building block is the random number generator. While much can be said about random number generators (several books and many research papers have been written on the subject), the emphasis in this chapter is on the use of random number generators in the simulation of communication systems. Thus, we restrict our study to the essential task of generating sampled versions of random waveforms (signals, interference, noise, etc.) for use in simulation programs. In the simulation context, all random processes must be expressed by sequences of random variables. Generating and testing these random sequences are the subject of this chapter. Many programming languages useful for developing simulation programs, such as MATLAB, contain random number generators as part of the library of "built-in"

functions. Understanding the concepts upon which these number generators are based provides important insight into the overall simulation program. It is wise to ensure that these number generators are properly designed and appropriate for use in a given application.

We will see that these random number generators do not, strictly speaking, generate random numbers, but produce sequences that *appear random* over the observation (simulation) interval so that they can be used to approximate a sample function of a random process in a given simulation program. By "appear random" we mean that the generated sequences, over a given simulation interval, have the properties required to accurately model a random process, with the required level of accuracy, for a given application. We refer to such sequences as *pseudo-random sequences*, since, even though they are deterministic, they appear random when used for a given application. The accuracy required is dependent on the application. For example, if we must generate a waveform to represent the noise at the input of a PLL discriminator, higher accuracy is required to model the noise waveform for an input SNR of 50 dB than for 8 dB. More accuracy is required to model the noise component in a digital communications system if the bit-error probability is 10^{-7} than if the bit-error probability is 10^{-3}.

In this chapter we first consider the generation of sample functions of a random process. The concept of stationarity is examined in the simulation context. Simulation models for digital modulators are then briefly considered. After these preliminary discussions, we turn our attention to the main focus of this chapter and consider the following:

- Generating uncorrelated random numbers uniformly distributed in (0,1)
- Mapping random numbers that are uncorrelated and uniformly distributed to random numbers that are uncorrelated and have an arbitrary (desired) probability density function (pdf)
- Generating random numbers that are uncorrelated and have a Gaussian pdf
- Generating random numbers that are correlated and have a Gaussian pdf
- Generating random numbers that are correlated and have an arbitrary (desired) pdf

We then take a brief look at the generation of pseudonoise (PN) sequences and at several computational techniques applied to sequences of random numbers.

7.1 Stationary and Ergodic Processes

When simulating a communications system, the sample functions generated to represent signals, noise, and interference will be assumed ergodic. This is required, since we typically process time-domain samples of waveforms through the system sequentially and, at each point in the system, there is a single waveform (sample function). We make the assumption that the waveform processed by the simulation

is a typical member of the ensemble defined by the underlying statistical model. Various statistical quantities such as moments, signal-to-noise ratios, and bit-error rates will be computed as time average quantities. When comparing simulation results with corresponding theoretical results, there will usually be an underlying assumption that time averages, computed by the simulation, are equivalent to ensemble averages. As a result, there is an implied assumption that the underlying random processes are ergodic.

Ergodic processes are always stationary. Therefore, the sample functions generated within a simulation are always assumed to be members of a stationary random process. Recall from basic random process theory that the definition of stationarity is that all statistical quantities are independent of the time origin. In order to demonstrate several of these ideas, we pause to consider a simple example.

Example 7.1. Assume that the sample functions of a random process are defined by the expression

$$x(t, \xi_i) = A\cos(2\pi f t + \phi_i) \tag{7.1}$$

in which ξ_i is an outcome in the sample space of an underlying random experiment, and each outcome ξ_i is mapped to a phase ϕ_i. We also assume that the underlying random experiment consists of drawing a number from a uniform number generator. The result of this draw is the outcome $\xi_i = u_i$, where u_i is uniformly distributed in $(0, 1)$. The value of u_i is then mapped into a phase $\phi_i = ku_i$. With A and f fixed, the value of ϕ_i determines the waveform. In this example we have interest in two values of k, namely, $k = 2\pi$, in which the phase is uniformly distributed in $(0, 2\pi)$, and $k = \pi/2$, in which the phase is uniformly distributed in $(0, \pi/2)$. As a second example, assume that the random process is described by the expression

$$x(t, \xi_i) = A(1 + u_i)\cos 2\pi f t \tag{7.2}$$

In this case the amplitude is uniformly distributed in the range $(A, 2A)$.

The following MATLAB program produces three sets of sample functions of a random process. The first set of waveforms, denoted $x(t)$, corresponds to (7.1) with $k = 2\pi$. The second set of waveforms, denoted $y(t)$, corresponds to (7.1) with $k = \pi/2$. The third set of waveforms, denoted $z(t)$, are defined by (7.2). For all waveforms, $A = 1$ and $f = 1$. Two seconds of data and twenty sample functions are generated for each simulation.

```
% File:  c7_sinewave.m
f = 1;                  % frequency of sinusoid
fs = 100;               % sampling frequency
t = (0:200)/fs;         % time vector
for i=1:20
    x(:,i) = cos(2*pi*f*t+rand(1)*2*pi)';
    y(:,i) = cos(2*pi*f*t+rand(1)*pi/2)';
    z(:,i) = (1+rand(1))*cos(2*pi*f*t)';
end
subplot(3,1,1); plot(t,x,'k'); ylabel('x(t)')
```

```
subplot(3,1,2); plot(t,y,'k'); ylabel('y(t)')
subplot(3,1,3); plot(t,z,'k'); ylabel('z(t)')
% End of script file.
```

Executing this program yields the results illustrated in Figure 7.1.

The time averages of all sample functions comprising $x(t)$, $y(t)$, and $z(t)$ are all equal to zero. The reader can easily verify (see Problem 7.1 that the ensemble averages of $x(t)$ are approximately zero when computed at a large number of points, t_i for $0 \leq t_i \leq 2$. The time averages will converge to zero as the number of sample functions tends to ∞. For $y(t)$, however, the ensemble average is approximately 1 for t in the neighborhood of 0.875 and 1.875, is approximately -1 for t in the neighborhood of 0.375 and 1.375, and is approximately zero for t in the neighborhood of 0.125, 0.625, 1.125, and 1,625. This is an example of a cyclostationary process, in which the moments are periodic.

The sample functions denoted by $z(t)$ are also sample functions from a cyclostationary process. Note that sampling the process at $t = 0.5k$ generates a random variable, the mean of which is approximately $+1.5$ for k even and is approximately -1.5 for k odd. ■

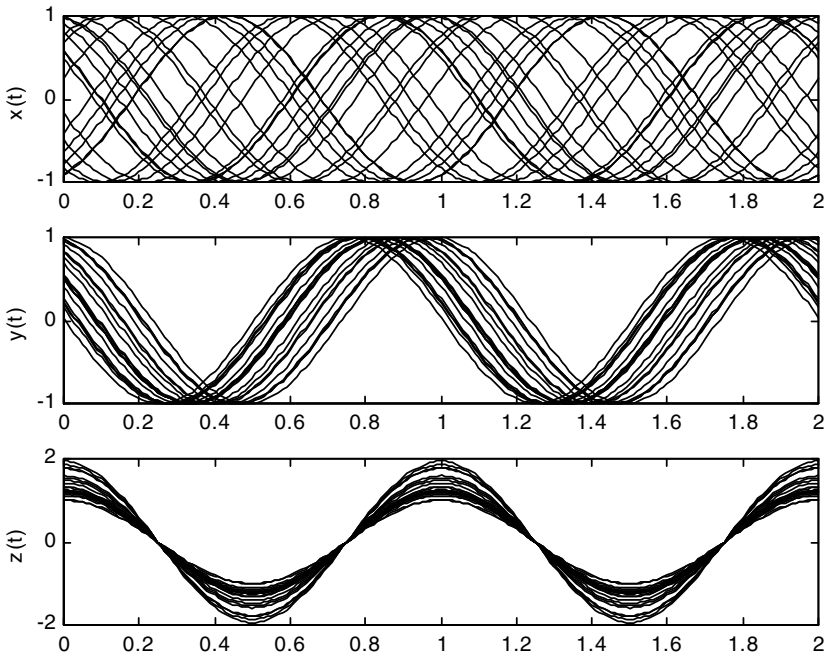


Figure 7.1 Sample functions for three different random processes.

The preceding example made use of the MATLAB uniform random number generator `rand`. In the following example we illustrate the use of a random number generator for modeling a digital modulator. In the following section the algorithms used for implementing uniform number generators is explored in detail.

Example 7.2. In the work to follow we have frequent need for models of digital modulators. The fundamental building block for these modulators will be the function `random_binary`, which produces a binary waveform having values +1 or −1. The number of bits produced, as well as the number of samples per bit, are arguments. The listing follows:

```
function [x, bits] = random_binary(nbits,nsamples)
% This function generates a random binary waveform of length nbits
% sampled at a rate of nsamples/bit.
x = zeros(1,nbits*nsamples);
bits = round(rand(1,nbits));
for m=1:nbits
    for n=1:nsamples
    index = (m-1)*nsamples + n;
    x(1,index) = (-1)^bits(m);
    end
end
```

The function `random_binary` can be used to simulate a number of digital modulators. For example a QPSK modulator can be simulated using the MATLAB statement

```
x = random_binary(nbits,nsamples)+i*random_binary(nbits,nsamples);
```

The following MATLAB program generates a QPSK signal for 10 bits with a sampling frequency of 8 samples per bit:

```
% File:  c7_example2.m
nbits = 10; nsamples = 8;
x = random_binary(nbits,nsamples)+i*random_binary(nbits,nsamples);
xd = real(x); xq = imag(x);
subplot(2,1,1)
stem(xd,'.'); grid; axis([0 80 -1.5 1.5]);
xlabel('Sample Index'); ylabel('xd')
subplot(2,1,2)
stem(xq,'.'); grid; axis([0 80 -1.5 1.5]);
xlabel('Sample Index'); ylabel('xq')
% End of script file.
```

Executing the program yields the QPSK signal, having the direct and quadrature components, illustrated in Figure 7.2. ■

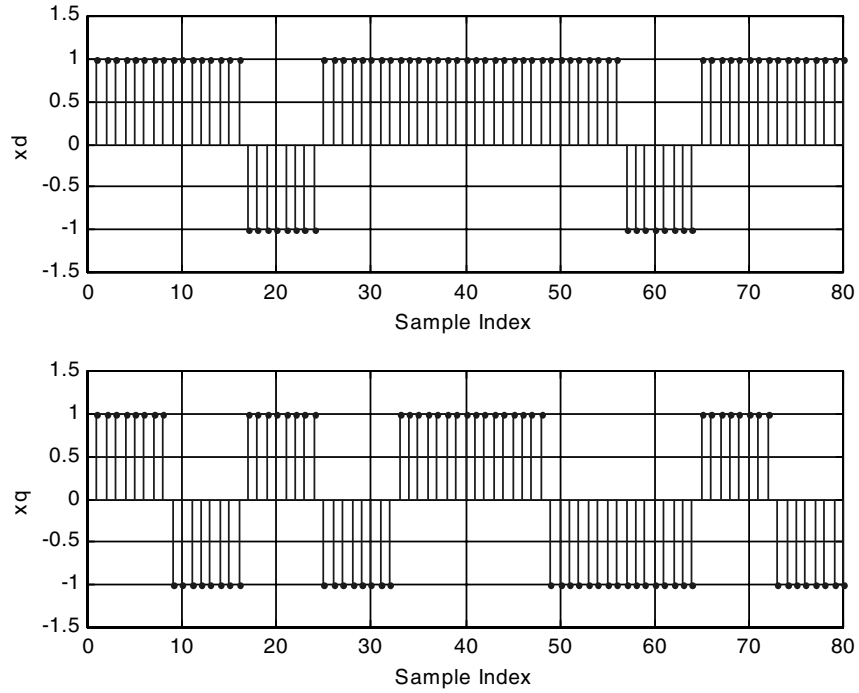


Figure 7.2 Direct and quadrature components of a QPSK signal.

7.2 Uniform Random Number Generators

A random variable having a uniform probability density function is easily transformed to a random variable having a desired pdf other than uniform. Therefore, the first step in the generation of a random variable having a specified pdf is to generate a random variable that is uniformly distributed on the interval (0,1). This is typically accomplished by first generating a sequence of numbers (integers) between 0 and M and then dividing each element of the sequence by M. The most common technique for implementing random number generators is known as linear congruence.

7.2.1 Linear Congruence

A linear congruence generator (LCG) is defined by the operation

$$x_{i+1} = [ax_i + c] \bmod(m) \tag{7.3}$$

where a and c are referred to as the multiplier and increment, respectively, and the parameter m is referred to as the modulus. This is, of course, a deterministic sequential algorithm in which successive values of x are generated in turn. The

initial value of x, denoted x_0, is referred to as the seed number of the generator. (We will have more to say about seed numbers later in this section.) Given that x_0, a, c, and m are integers, all numbers produced by the LCG will be integers. Since the operation $[ax_i + c]$ is evaluated mod(m) it follows that, at most, m distinct integers can be generated by (7.3). A desirable property of the generator output is that it has a long period, so that the maximum number of integers are produced in the output sequence before the sequence repeats. When the period is maximized, for a given value of m, we say that the generator is *full period*. In addition, application to a given simulation program places other demands on the LCG. For example, we usually require that the samples x_i and x_{i+1} be uncorrelated. In addition, the LCG output may be required to pass other statistical tests, depending on the application. The LCG can take many different forms. In this section, only the most common algorithms are considered.

Technique A: The Mixed Congruence Algorithm

The most general congruence algorithm is the "mixed" congruence algorithm for which $c \neq 0$. We refer to this algorithm as a mixed algorithm because both multiplication and addition are involved in the calculation of x_{i+1}. The mixed linear algorithm takes the form given in (7.3)

$$x_{i+1} = [ax_i + c] \bmod(m) \tag{7.4}$$

For $c \neq 0$, the generator has a maximum period of m. This period is achieved if and only if

- The increment c is relatively prime to m. In other words, c and m have no common prime factors.
- $a - 1$ is a multiple of p, where p represents the prime factors of the modulus m.
- $a - 1$ is a multiple of 4 if m is a multiple of m.

A proof of the foregoing statement is given by Knuth [1].

Example 7.3. We wish to design a mixed congruence generator having a period $m = 5,000$. Since

$$5000 = \left(2^3\right)\left(5^4\right) \tag{7.5}$$

we can ensure that m and c are relatively prime by setting c equal to a product of primes other than 2 and 5. This satisfies the first property. One of many possibilities is to set

$$c = \left(3^2\right)\left(7^2\right) = 1323 \tag{7.6}$$

The value of a must now be selected. The second property is satisfied by setting

$$a - 1 = k_1 p_1 \tag{7.7}$$

and

$$a - 1 = k_2 p_2 \tag{7.8}$$

where $p_1 = 2$ and $p_2 = 5$ (the factors of m), and k_1 and k_2 are arbitrary integers. Since 4 is a factor of $m = 5,000$, we satisfy the third bullet by setting

$$a - 1 = 4k_3 \tag{7.9}$$

where k_3 is an arbitrary integer. An obvious choice for a is to let

$$a - 1 = 4kp_1 p_2 \tag{7.10}$$

or

$$a - 1 = 2 \cdot 4 \cdot 5 \cdot k = 40k \tag{7.11}$$

where k is an integer. With $k = 6$, we have $a = 241$. Thus:

$$x_{i+1} = [241x_i + 1323] \bmod (5000) \tag{7.12}$$

is a full-period generator. Note that there are many other choices of parameters that will produce a full-period generator with $m = 5,000$. ■

Example 7.4. In this example we show that the LCG designed in the previous example does indeed have a period of $m = 5,000$. In the following MATLAB program, a seed number is entered and the program runs until the seed reoccurs. If n integers are generated and $n > m$ without the seed recurring, one assumes that the generator is caught in a loop in which a short sequence is repeatedly generated. The MATLAB program is

```
% File:  c7_LCGperiod.m
a = input('Enter multiplier a > ');
c = input('Enter offset c > ');
m = input('Enter modulus m > ');
seed = input('Enter seed > ');
n=1; ix = rem((seed*a+c),m);
while (ix~=seed)&(n<m+2)
    n = n+1; ix = rem((ix*a+c),m);
end
if n>m
    disp('Caught in a loop.')
else
    text = ['The period is ',num2str(n,15),'.'];
    disp(text)
end
% End of script file.
```

Executing the program yields the following dialog:

```
≫ c7_LCGperiod
Enter multiplier a > 241
Enter offset c > 1323
Enter modulus m > 5000
Enter seed > 1
The period is 5000.
```

We see that the period is indeed 5,000 as expected. ■

Technique B: The Multiplicative Algorithm With Prime Modulus

The multiplicative generator is defined as

$$x_{i+1} = [ax_i] \bmod(m) \tag{7.13}$$

which is the mixed algorithm with the increment c set equal to zero. Note that x_i cannot equal zero for $c = 0$. Therefore, the full period is $m - 1$ rather than m, as was the case previously. The multiplicative algorithm produces a full period if [1]

- m is prime (m is usually required to be large)
- a is a primitive element mod(m)

As we know, a prime number is a number evenly divisible only by 1 or by the number itself. The second property perhaps requires an explanation. We mean that a is a primitive element mod(m) if $a^i - 1$ is a multiple of m for $i = m - 1$, but for no smaller value of i. In other words, a is a primitive element mod(m) if

$$\frac{a^{m-1} - 1}{m} = k \tag{7.14}$$

and

$$\frac{a^i - 1}{m} \neq k, \quad i = 1, 2, 3, \cdots, m - 2 \tag{7.15}$$

for k an arbitrary integer. For a proof that (7.13) produces a full-period generator under the given conditions see [1].

Technique C: The Multiplicative Algorithm with Nonprime Modulus

The most important case in which the modulus m is not a prime number is m equal to a power of two. In other words:

$$x_{i+1} = [ax_i] \bmod(2^n) \tag{7.16}$$

for integer n. For the case defined by (7.16), the maximum period is $2^n/4 = 2^{n-2}$. This period is achieved if

- The multiplier a is 3 or 5 mod(8)
- The seed x_0 is odd

A proof is given by Knuth [1].

Since the product of two odd numbers is odd, it follows that all values generated by (7.16) are odd if x_0 is odd. Thus, no even values of x_i are generated, which reduces the period by a factor of two. The odd integers generated by (7.16) are divided into two sets, only one of which is generated for a given seed. This reduces the period by another factor of two. The set of odd integers actually generated depends upon the choice of the seed. (See Problem 7.3.)

The advantage of using $m = 2^k$ is that integer overflow can be used to perform the mod(m) operation. This reduces the computation time. While this is indeed desirable, the result is a program that is not easily transportable.

7.2.2 Testing Random Number Generators

The previous section provides us with the tools for generating pseudo-random numbers that are uniformly distributed between 0 and 1. To this point we have considered only the period of the sequence produced by an LCG. While we obviously wish this period to be long, there are other desirable attributes to be satisfied for a given application. At the very least, we desire the sequence to be delta correlated (white). Other requirements may be necessitated by the application.

A number of procedures have been developed for testing the *randomness* of a given sequence. Among the most popular of these are the Chi-square test, the Kolomogorov-Smirnov test, and the spectral test. A study of these is beyond the scope of the material presented here. The interested student is referred to the literature [1, 2]. The spectral test appears to be the most powerful of these tests. A brief description of the spectral test, applied to the Wichmann-Hill algorithm to be discussed later, is given in the paper by Coates [3].

For many of the applications to follow, the most important attribute to be satisfied is that the elements of a given sequence are independent, or at least uncorrelated. Toward this end, we consider two very simple tests: scatterplots and the Durbin-Watson test. It should be pointed out that the properties of a given sequence apply to the complete sequence (the full period). If one uses only a portion of the sequence, the properties of the complete sequence no longer apply.

Scatterplots

The scatterplot is best illustrated by an example.

Example 7.5. A scatterplot is a plot of x_{i+1} as a function of x_i, and represents an empirical measure of the quality of the number generator. For this example, we consider two number generators defined by

$$x_{i+1} = (65x_i + 1) \bmod(2048) \tag{7.17}$$

and

$$x_{i+1} = (1229x_i + 1) \bmod(2048) \tag{7.18}$$

Applying the program `c7_LCGperiod.m` presented in Example 7.2 shows that both of these generators are full period. The MATLAB code for generating the scatterplots for each of these generators is

```
% File:  c7_LCDemo1.m
m = 2048; c = 1; seed = 1;        % default values of m and c
a1 = 65; a2 = 1229;               % multiplier values
ix1 = seed; ix2 = seed;           % initialize algorithm
x1 = zeros(1,m); x2 = zeros(1,m); % initialize arrays
for i=1:m
    ix1 = rem((ix1*a1+c),m);
    x1(i) = ix1/m;
    ix2 = rem((ix2*a2+c),m);
    x2(i) = ix2/m;
end
subplot(1,2,1)
y1 = [x1(1,2:m),x1(1,1)];
plot(x1,y1,'.')                    % plot results for a1
subplot(1,2,2)
y2 = [x2(1,2:m),x2(1,1)];
plot(x2,y2,'.')                    % plot results for a2
% End of script file.
```

Executing the program yields the scatterplots illustrated in Figure 7.3. One seeks a scatterplot in which all combinations of the ordinate x_{i+1} and the abscissa x_i occur. For this case, the scatterplot is devoid of structure. It appears from Figure 7.3 that $a = 65$ yields a generator having smaller serial correlation than the generator with $a = 1{,}229$. We will see in Example 7.6 that this is indeed the case. ■

The Durbin-Watson Test

The Durbin-Watson test for independence is implemented by calculating the Durbin parameter

$$D = \frac{(1/N)\sum_{n=2}^{N}(X[n] - X[n-1])^2}{(1/N)\sum_{n=1}^{N} X^2[n]} \tag{7.19}$$

where $X[n]$ is a *zero-mean* random variable [4]. We will show that values of D in the neighborhood of 2 imply small correlation between $X[n]$ and $X[n-1]$.

In order to illustrate the properties of the Durbin-Watson test, assume that $X[n-1]$ and $X[n]$ are correlated and that $X[n]$ is an ergodic process. In order to simplify the notation we assume that N is large so that $N - 1 \approx N$ and write

$$D = \frac{E\left\{(X-Y)^2\right\}}{E\left\{X^2\right\}} = \frac{1}{\sigma_x^2} E\left\{(X-Y)^2\right\} \tag{7.20}$$

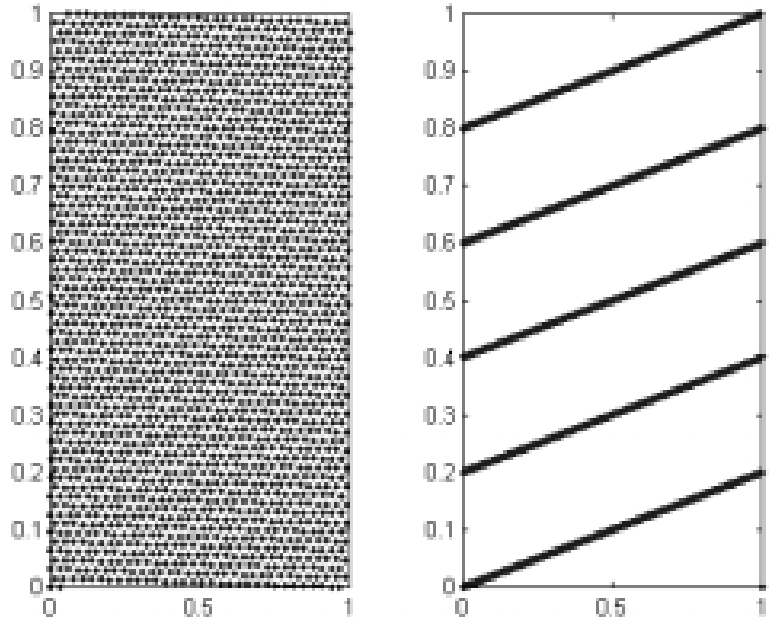


Figure 7.3 Scatterplots for $a_1 = 65$ (left) and $a_2 = 1{,}229$ (right).

where X denotes $X[n]$, Y denotes $X[n-1]$, and $E\{\cdot\}$ denotes expectation. Since we assumed that $X[n]$ and $X[n-1]$ are correlated we let[1]

$$Y = \rho X + \sqrt{1-\rho^2}Z \tag{7.21}$$

where X and Z are uncorrelated and ρ is the correlation coefficient relating X and Y. Note that X, Y, and Z all have equal variance, which we denote σ^2. Substituting (7.21) in (7.20) gives

$$D = \frac{1}{\sigma^2} E\left\{(1-\rho)^2 X^2 - 2(1-\rho)\sqrt{1-\rho^2}XZ + (1-\rho^2)Z^2\right\} \tag{7.22}$$

The middle term is zero, since X and Z are uncorrelated and zero-mean. Since X and Z have equal variance

$$D = \frac{(1-\rho)^2\sigma^2 + (1-\rho^2)\sigma^2}{\sigma^2} = 2(1-\rho) \tag{7.23}$$

Since $-1 \leq \rho \leq 1$, the Durbin parameter D varies between 0 and 4, with $D = 2$ if $\rho = 0$. Values of $D < 2$ imply positive correlation, while $D > 2$ implies negative

[1]This transformation will be discussed in Section 7.5.1.

values of ρ. The following MATLAB function computes the value of the Durbin parameter:

```
% File: c7_durbin.m
function D = durbin(x)
N = length(x);                  % length of input vector
y = x-mean(x);                  % remove dc
ydiff = y(2:N)-y(1:(N-1));      % numerator summand
Num = sum(ydiff.*ydiff);        % numerator factor of D
Den = sum(y.*y);                % denominator factor of D
D = Num/Den;                    % Durbin factor
% End of script file.
```

Example 7.6. In this example, we calculate the value of D for the two noise generators considered in Example 7.5. The MATLAB code is as follows:

```
% File:  c7_LCDemo2.m
m = 2048; c = 1; seed = 1;
a1 = 65; a2 = 1229;
ix1 = 1; ix2 = 1;
x1 = zeros(1,m); x2 = zeros(1,m);
for i=1:m
    ix1 = rem((ix1*a1+c),m);
    x1(i) = ix1;
    ix2 = rem((ix2*a2+c),m);
    x2(i) = ix2;
end
D1 = c7_Durbin(x1); D2 = c7_Durbin(x2); % calculate Durbin parameters
rho1 = 1-D1/2; rho2 = 1- D2/2; % calculate correlation
text1 = [‘The value of D1 is ’,num2str(D1),‘ and rho1 is ’,...
   num2str(rho1),‘.’];
text2 = [‘The value of D2 is ’,num2str(D2),‘ and rho2 is ’,...
   num2str(rho2),‘.’];
disp(text1)
disp(text2)
% End of script file.
```

Executing the program yields:

```
≫ c7_LCDemo2
The value of D1 is 1.9925 and rho1 is 0.0037273.
The value of D2 is 1.6037 and rho2 is 0.19814.
```

For $a_1 = 65$ the correlation is approximately 0, while for $a_2 = 1{,}229$ the correlation is approximately 0.2. It therefore follows from the Durbin-Watson test that $a_1 = 65$

gives superior results to $a_2 = 1,229$. This result is consistent with the scatterplots shown in Figure 7.3. ■

7.2.3 Minimum Standards

It is a major task to thoroughly test a given LCG for quality by showing that a variety of statistical tests for randomness are passed. This is especially true when the generated sequence is long. In order to partially solve this problem, a number of algorithms have been identified as *minimum standard* algorithms. A minimum standard algorithm is one that is [5]

- Full period
- Passes all applicable statistical tests for randomness
- Easily transportable from one computer to another

Once such an algorithm has been identified and properly documented, it becomes a minimum standard. The algorithm can then be used with confidence by others without additional testing. As pointed out in [5], if a minimum standard algorithm is used, one need not worry about the *correctness* of the algorithm, but must ensure that the algorithm is *implemented* correctly for the given computational environment.[2] An important programming concern is that all numbers generated by the algorithm be *uniquely* representable.[3]

Lewis, Goodman, and Miller Minimum Standard

The Lewis, Goodman, and Miller minimum standard is defined by [5]

$$x_{i+1} = (16807x_i) \bmod(2147483647) \tag{7.24}$$

in which m is the Mersenne[4] prime $2^{31} - 1$. This value of m was first suggested by Lehmer, who was responsible for much of the basic work on LCGs more than half a century ago [5]. It is widely used and is easily implemented in integer arithmetic on 32-bit computers, and in floating-point arithmetic if the mantissa exceeds 31 bits.[5]

The Wichmann-Hill Algorithm

The previous work has shown that we desire number generators having long periods. An effective technique for constructing a waveform having a long period is to sum several periodic waveforms having slightly different periods. For example, $\cos 2\pi(1)t$

[2]We generally assume that a computational environment is a general-purpose computer. However, for some applications the computational environment could be a special-purpose machine, an ASIC chip, an FPGA chip, or a programmable DSP chip.

[3]For speed of computation, LCGs are typically implemented using integer arithmetic. In MATLAB we use floating-point arithmetic. In order to uniquely represent each number defined by the algorithm, m must be less than the MATLAB constant eps, the default value of which exceeds 4×10^{15} on IEEE compliant computers (see Chapter 3).

[4]If $m = 2^k - 1$ is a prime number, m is known as a Mersenne prime.

[5]Recall from Chapter 3 that the IEEE floating-point standard assigns 51 bits to the mantissa.

has a period of 1 second and $\cos 2\pi(1.0001)t$ has a period of $10,000/10,001$, which is slightly less than 1 second. The composite waveform can be written in the form

$$x(t) = \cos 2\pi(10000/10000)t + \cos 2\pi(10001/10000)t \tag{7.25}$$

which has a period of $10,000$ seconds, or approximately 2.78 hours. During this period, the first component goes through $10,000$ periods and the second component goes through $10,001$ periods. Additional components can be used if desired.

The same technique can be applied to LCGs by combining several number generators having different, but approximately the same, periods [6]. The Wichmann-Hill algorithm is probably the best known example of a combined number generator. Many different variations of the Wichmann-Hill algorithm are possible. The original algorithm, which is nicely described in a paper by Coates [3], uses three component generators defined as

$$x_{i+1} = (171x_i) \bmod(30269) \tag{7.26}$$

$$y_{i+1} = (170y_i) \bmod(30307) \tag{7.27}$$

$$z_{i+1} = (172z_i) \bmod(30323) \tag{7.28}$$

The three component generators are indeed full-period generators (see Problem 7.11). The three component generators are combined to give the output

$$u_i = \left(\frac{x_i}{30269} + \frac{y_i}{30307} + \frac{z_i}{30323}\right) \bmod(1) \tag{7.29}$$

This Wichmann-Hill algorithm is equivalent to a multiplicative LCG with multiplier

$$a = 16,555,425,264,690 \tag{7.30}$$

and modulus

$$m = 30269 \cdot 30307 \cdot 30323 \approx 2.7817 \times 10^{13} \tag{7.31}$$

Since M is clearly not prime, the period is shorter than $m - 1$. It is shown in [3] that the period is approximately 7.0×10^{12} that, although less than m, is still extremely long.

The Wichmann-Hill algorithm, although somewhat different in architecture from the previously presented minimum standard, is considered a minimum standard uniform number generator, since it has been tested extensively, has been shown to pass all of the standard statistical tests, and is easily transported from one machine to another [3].

7.2.4 MATLAB Implementation

Prior to the release of MATLAB 5, the uniform random number generator `rand`, included in the MATLAB library, was the minimum standard number generator defined by (7.24). The uniform random number generator used in MATLAB versions

5 and 6 is based on a technique developed by Marsaglia.[6] This number generator, which is targeted at the generation of floating-point numbers rather than scaled integers, is briefly described in a short paper by Moler [7]. MathWorks claims that this number generator has a period exceeding 2^{1492} and is "fairly sure" that all floating-point numbers between `eps` and `1-eps/2` are generated, where the MATLAB constant `eps`, as described in Chapter 3, is 2^{-52}. The new number generator uses only addition and subtraction. Since no multiplications or divisions are used, the algorithm executes much faster than an LCG.

7.2.5 Seed Numbers and Vectors

Since the simulation examples presented in this book are based on MATLAB, it is important to briefly consider the way in which MATLAB handles seeds. The "old" MATLAB random number generator (prior to MATLAB 5 and defined by (7.24) used a single seed number. The "new" random number uses a vector seed, referred to as the state of the number generator. This vector consists of 35 elements (32 floating-point numbers, two integers, and a flag) that define the state of the number generator [7]. With MATLAB 5 or later, either number generator can be used. The new random number generator is the default. The old random number defined by (7.24) can be invoked by using the command `RAND('seed',0)` or `RAND('seed',J)`. As with all MATLAB commands, the user should carefully study the information provided by the `help` command. In addition, the user should be aware of the following (the term *seed* is used to cover both integer seeds and state vectors):

- The user can either use the default seed number or can specify a seed number.
- Closing and reopening MATLAB resets the seed to the default value. Thus, if one makes N calls to a random number generator, then closes and reopens MATLAB and makes N more calls to a random number generator, the same N numbers will be produced in both cases. This property can be used to advantage in MATLAB, since it allows one to reproduce identical sequences of results. This is useful for testing purposes.
- The system clock can be used to randomize the initial seed. (See MATLAB `help` for details.)
- Seed numbers are stored in a buffer and not on the MATLAB workspace. As a result, executing the command `clear all` has no effect.

7.3 Mapping Uniform RVs to an Arbitrary pdf

Many different methods have been developed for mapping a uniformly distributed random variable to a target random variable having a pdf that is not uniform. There are basically three different situations that occur:

1. The cumulative distribution for the target random variable is known in closed form. We will see that if the CDF of the target random variable is known in

[6] See The MathWorks Support Solution Number 8542.

closed form, a very simple technique, known as the inverse transform method, can be used.

2. The pdf of the target random variable is known in closed form, but the CDF is not known in closed form. The important Gaussian random variable falls into this category. A number of ad hoc methods exist for this case and, in addition, rejection methods can be used.

3. Neither the pdf nor the CDF are known in closed form. This situation is often encountered when one must develop a random number generator to fit the pdf of experimentally collected data.

We now examine techniques that can be used in each of these three cases.

7.3.1 The Inverse Transform Method

The inverse transform method allows us to transform a uniformly distributed uncorrelated random sequence U to an uncorrelated (independent samples) sequence X having a distribution function, $F_X(x)$. The transformation makes use of a memoryless nonlinearity as shown in Figure 7.4. The fact that the nonlinearity is memoryless ensures that the output sequence is uncorrelated if the input sequence is uncorrelated. Of course, by the Weiner-Khitchine theorem, a sequence of uncorrelated random numbers has a PSD which is constant (white). The technique is to simply set

$$U = F_X(X) \tag{7.32}$$

and solve for x, which gives

$$X = F_X^{-1}(U) \tag{7.33}$$

Application of the inverse transform technique requires that the distribution function, $F_X(x)$, be known in closed form.

It is easy to see that the inverse transform technique yields a random variable with the required distribution function [2, 8]. Recall that a distribution, $F_X(x)$, is a nondecreasing function of the argument x, as illustrated in Figure 7.5. By definition

$$F_X(x) = \Pr\{X \leq x\} \tag{7.34}$$

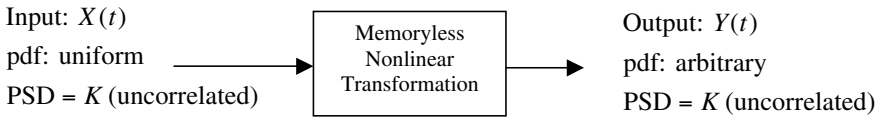


Figure 7.4 Inverse transform method.

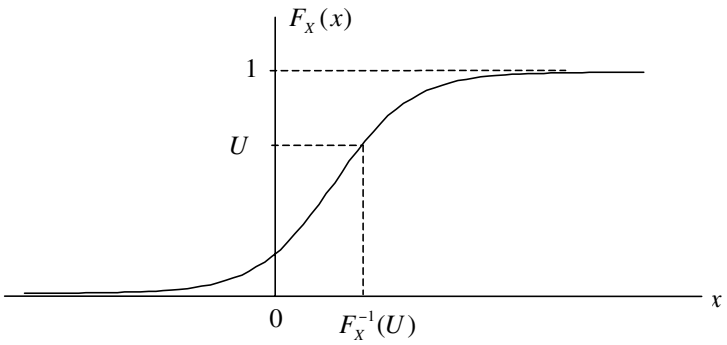


Figure 7.5 Cumulative distribution function.

Setting $X = F^{-1}(U)$ and recognizing that $F_X(x)$ is monotonic gives

$$F_X(x) = \Pr\left\{F^{-1}(U) \leq x\right\} = \Pr\{U \leq F_X(x)\} = F_X(x) \tag{7.35}$$

which is the desired result. We now pause to illustrate the technique through a simple example.

Example 7.7. In this example, a uniform random variable will be transformed into a random variable having the one-sided exponential distribution

$$f_X(x) = \beta \exp(-\beta x)\, u(x), \quad \beta > 0 \tag{7.36}$$

where $u(x)$ is the unit step defined by

$$u(x) = \begin{cases} 1, & x > 0 \\ 0, & x < 0 \end{cases} \tag{7.37}$$

The first step is to find the CDF. This is

$$F_X(x) = \int_0^x \beta \exp(-\beta y)\, dy = 1 - \exp(-\beta x) \tag{7.38}$$

Equating the distribution function to the uniform random variable U gives

$$1 - \exp(-\beta X) = U \tag{7.39}$$

which, when solved for X, provides the result

$$\exp(-\beta X) = 1 - U \tag{7.40}$$

Since the random variable $1 - U$ is equivalent to the random variable U ($Z = U$ and $Z = 1 - U$ have the same pdf), we may write the solution for X as

$$X = -\frac{1}{\beta} \ln(U) \tag{7.41}$$

The MATLAB code for implementing the uniform-to-exponential transformation follows:

```
% File:  c7_uni2exp.m
clear all                                % be safe
n = input('Enter number of points > ');
b = 3;                                   % set pdf parameter
u = rand(1,n);                           % generate U
y_exp = -log(u)/b;                       % transformation
[N_samp,x] = hist(y_exp,20);             % get histogram parameters
subplot(2,1,1)
bar(x,N_samp,1)                          % plot histogram
ylabel('Number of Samples')
xlabel('Independent Variable - x')
subplot(2,1,2)
y = b*exp(-3*x);                         % calculate pdf
del_x = x(3)-x(2);                       % determine bin width
p_hist = N_samp/n/del_x;                 % probability from histogram
plot(x,y,'k',x,p_hist,'ok')              % compare
ylabel('Probability Density')
xlabel('Independent Variable - x')
legend('true pdf','samples from histogram',1)
% End of script file.
```

The result for $\beta = 3$ and $N = 100$ is illustrated in Figure 7.6. The top portion of Figure 7.6 shows the histogram. The second part of Figure 7.6 shows both the theoretical pdf and the resulting "experimental" values for 100 samples. The relatively poor results obtained with $N = 100$ provide motivation to try again with a significantly larger number of samples. The result for $N = 2,000$ is illustrated in Figure 7.7. A significant improvement is noted. ■

Example 7.8. As a second example, consider the Rayleigh random variable described by the pdf

$$f_R(r) = \frac{r}{\sigma^2} \exp\left(-\frac{r^2}{2\sigma^2}\right) u(r) \tag{7.42}$$

where, as before, the unit step defines the pdf to be single-sided. The CDF is given by

$$F_R(r) = \int_0^r \frac{y}{\sigma^2} \exp\left(-\frac{y^2}{2\sigma^2}\right) dy = 1 - \exp\left(-\frac{r^2}{2\sigma^2}\right) \tag{7.43}$$

Setting $F_R(R) = U$ gives

$$1 - \exp\left(-\frac{R^2}{2\sigma^2}\right) = U \tag{7.44}$$

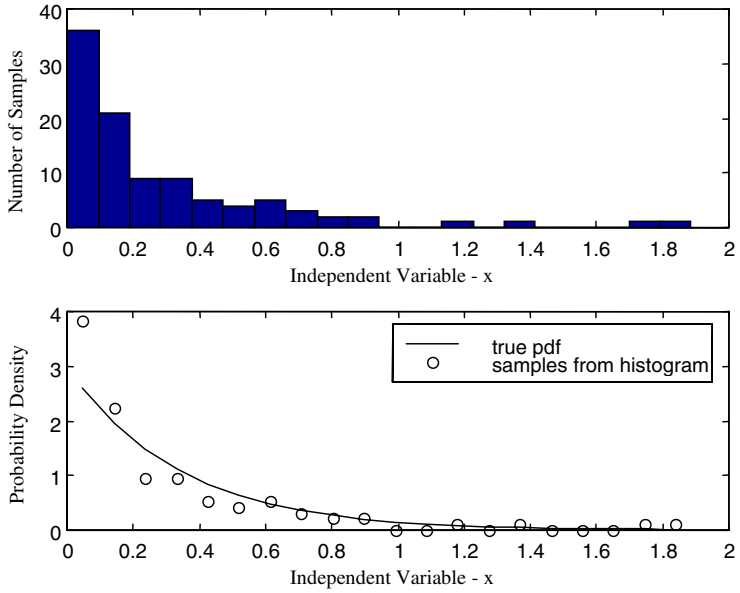


Figure 7.6 Uniform to exponential transformation for $N = 100$.

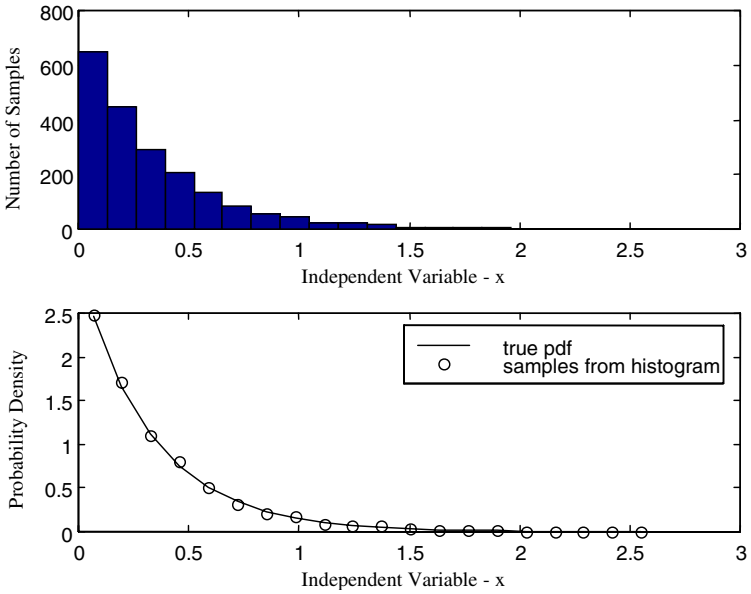


Figure 7.7 Uniform to exponential transformation for $N = 2,000$.

This is equivalent to

$$\exp\left(-\frac{R^2}{2\sigma^2}\right) = U \tag{7.45}$$

where we have once again recognized the equivalence of $1-U$ and U. Solving for R gives

$$R = \sqrt{-2\sigma^2 \ln(U)} \tag{7.46}$$

This transformation is the initial step in the Box-Muller algorithm, which is one of the basic algorithms for Gaussian number generation.

As with the previous example, it is interesting to implement the transformation and evaluate the performance as a function of the number of points transformed. The MATLAB program follows, and the results for $N = 10,000$ are shown in Figure 7.8. Other values of N should be used and the results compared to Figure 7.8.

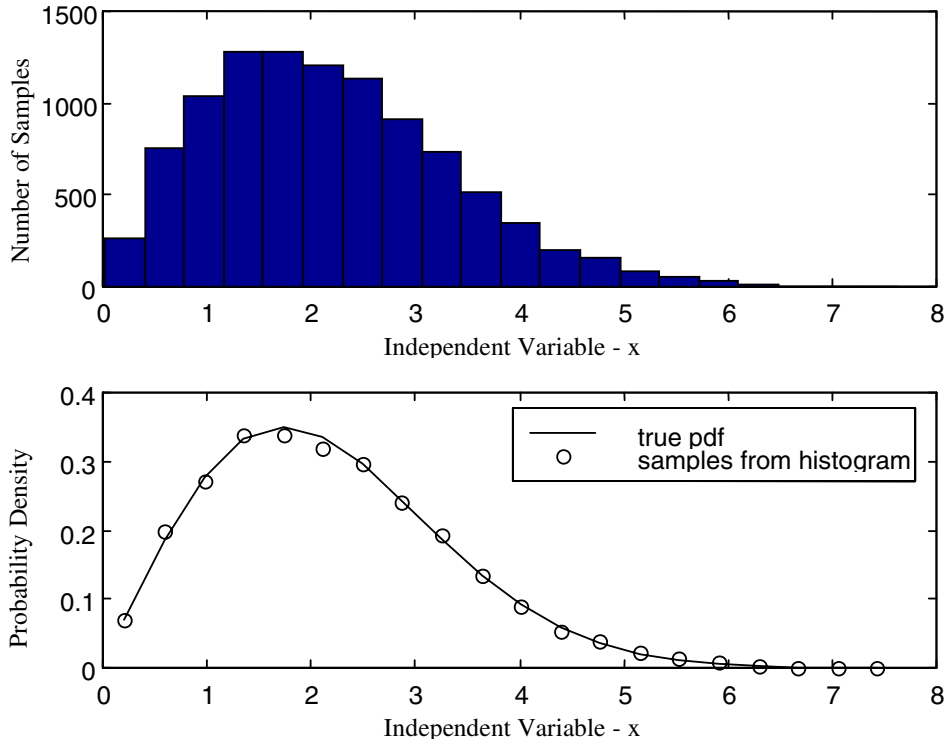


Figure 7.8 Uniform to Rayleigh transformation with $N = 10,000$.

```
% File:  c7_uni2ray.m
clear all                                % be safe
n = input('Enter number of points > ');
varR = 3;                                % set pdf parameter
u = rand(1,n);                           % generate U
y_exp = sqrt(-2*varR*log(u));            % transformation
[N_samp,r] = hist(y_exp,20);             % get histogram parameters
subplot(2,1,1)
bar(r,N_samp,1)                          % plot histogram
ylabel('Number of Samples')
xlabel('Independent Variable - x')
subplot(2,1,2)
term1 = r.*r/2/varR;                     % exponent
ray = (r/varR).*exp(-term1);             % Rayleigh pdf
del_r = r(3)-r(2);                       % determine bin width
p_hist = N_samp/n/del_r;                 % probability from histogram
plot(r,ray,'k',r,p_hist,'ok')            % compare results
ylabel('Probability Density')
xlabel('Independent Variable - x')
legend('true pdf','samples from histogram',1)
% End of script file.
```

■

The previous two examples illustrated the application of the inverse transform method to continuous random variables. The technique, however, can be applied to discrete random variables. The histogram method, which we now describe, is a numerical (discrete data) version of the inverse transform method.

7.3.2 The Histogram Method

Assume that we have a set of data collected experimentally. In such a situation, both the pdf and the CDF are unknown, even though the pdf can be approximated by a histogram of the data. Our problem is to develop an algorithm for generating a set of samples having a pdf approximating the pdf of the experimental data. The first step is to generate a histogram of the experimental data. Assume that the histogram illustrated in Figure 7.9 results. Once the histogram is generated, an approximation to the pdf and CDF are known and, therefore, the inverse transform method can be applied. The technique presented here is a simple extension of the inverse transform method studied in the previous section.

The probability that a sample value x lies in the i^{th} histogram bin is

$$P_i = \Pr\{x_{i-1} < x < x_i\} = c_i\,(x_i - x_{i-1}) \tag{7.47}$$

The CDF, evaluated at the point x, is denoted

$$F_X(x) = F_{i-1} + c_i(x - x_{i-1}) \tag{7.48}$$

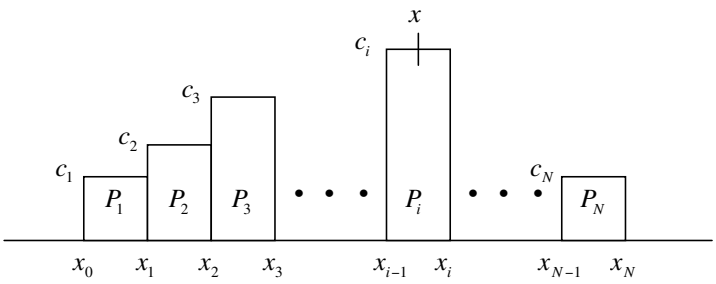


Figure 7.9 Histogram of experimental data.

where

$$F_{i-1} = \Pr\{X \leq x_{i-1}\} = \sum_{j=1}^{i-1} P_{i-1} \tag{7.49}$$

The next step is to set $F_X(X) = U$, where U is a uniform random variable. This gives

$$F_X(X) = U = F_{i-1} + c_i\,(X - x_{i-1}) \tag{7.50}$$

Solving for X gives

$$X = x_{i-1} + \frac{1}{c_i}\,(U - F_{i-1}) \tag{7.51}$$

The algorithm for the required number generator is defined by the following three steps:

1. Generate U by drawing a sample from a random number generator producing numbers uniformly distributed in $(0, 1)$.

2. Determine the value of i so that

$$F_{i-1} < U \leq F_i \tag{7.52}$$

   where F_i is defined in (7.49).

3. Generate X according to (7.51) and return X to the calling program.

The fidelity of the number generator clearly depends on the accuracy of the underlying histogram. The accuracy of the histogram, as we shall see in the following chapter, depends on the number of samples available.

7.3.3 Rejection Methods

The rejection (or acceptance) technique for generating random variables having a desired or "target" pdf, $f_X(x)$, basically involves bounding the target pdf by a function $Mg_X(x)$, in which $g_X(x)$ represents the pdf of an easily generated random variable and M is a constant suitably large to ensure that

$$Mg_X(x) \geq f_X(x), \quad \text{all } x \tag{7.53}$$

In the simplest form, $g_X(x)$ is uniform on $(0, a)$. If the target pdf $f_X(x)$ is zero outside the range $(0, a)$ we have

$$Mg_X(x) = \begin{cases} b = M/a, & 0 \leq x \leq a \\ 0, & \text{otherwise} \end{cases} \tag{7.54}$$

where, since $Mg_X(x)$ bounds $f_X(x)$,

$$b = \frac{M}{a} \geq \max\{f_X(x)\} \tag{7.55}$$

This is illustrated in Figure 7.10.

The algorithm for generating the random variable X having pdf $f_X(x)$ is defined by the following four steps:

1. Generate U_1 and U_2 uniform in $(0, 1)$.

2. Generate V_1 uniform in $(0, a)$, where a is the maximum value of X.

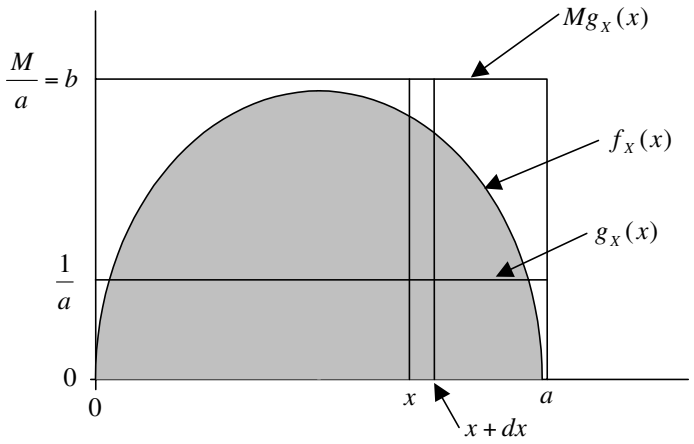


Figure 7.10 Rejection method for Example 7.9.

3. Generate V_2 uniform in $(0, b)$, where b is at least the maximum of $f_X(x)$.
4. If $V_2 \leq f_X(V_1)$, set $X = V_1$. If the inequality is not satisfied, V_1 and V_2 are discarded and the process is repeated from step 1.

It is easy to show that this procedure produces a random variable X having the target pdf, $f_X(x)$.

Since V_1 and V_2 are uniformly distributed, the point generated by the sample pair (V_1, V_2) has an equal probability of falling anywhere in the area ab. The probability that V_1 is accepted is the fraction of the ab area falling under the pdf $f_X(x)$. This is the ratio of the shaded area in Figure 7.10 to the total area ab. Thus:

$$\Pr\{V_1 \text{ is accepted}\} = \frac{\int_0^a f_X(x)\,dx}{ab} \tag{7.56}$$

Since the numerator in the preceding expression is one by definition:

$$\Pr\{V_1 \text{ is accepted}\} = \frac{1}{ab} = \frac{1}{M} \tag{7.57}$$

Thus:

$$\begin{aligned}\Pr\{x < V_1 \leq x + dx \,|\, V_1 \text{ is accepted}\} &= \frac{\Pr\{x < V_1 \leq x + dx, V_1 \text{ is accepted}\}}{\Pr\{V_1 \text{ is accepted}\}} \\ &= \frac{f_X(x)dx/ab}{1/ab} \qquad (7.58) \\ &= f_X(x)\,dx = \Pr\{x < X \leq x + dx\} \qquad (7.59)\end{aligned}$$

which defines the target pdf.

Example 7.9. In this example, we apply the technique just discussed to the pdf

$$f_X(x) = \begin{cases} \frac{4}{\pi R^2}\sqrt{R^2 - x^2}, & 0 \leq x \leq R \\ 0, & \text{otherwise} \end{cases} \tag{7.60}$$

The algorithm for generating the random variable X having pdf $f_X(x)$ is obtained by letting $a = R$ and $b = M/R$. Figure 7.10, modified for this specific case, is illustrated in Figure 7.11.

The MATLAB code follows:

```
% File:  c7_rejex1.m
R = 7;                                      % default value of R
M =4/pi;                                    % value of M
N = input('Input number of points N > ');   % set N
fx = zeros(1,N);                            % array of output samples
u1 = rand(1,N); u2 = rand(1,N);             % generate u1 and u2
v1 = R*u1;                                  % generate v1
v2 = (M/R)*rand(1,N);                       % generate v2 (g(x))
kpts = 0;                                   % initialize counter
```

```
for k=1:N
if v2(k)<(M/(R*R))*sqrt(R*R-v1(k)*v1(k));
kpts=kpts+1;                                % increment counter
fx(kpts)=v1(k);                             % save output sample
end
end
fx = fx(1:kpts);
[N_samp,x] = hist(fx,20);                   % histogram parameters
subplot(2,1,1)
bar(x,N_samp,1)                             % plot histogram
ylabel('Number of Samples')
xlabel('Independent Variable - x')
subplot(2,1,2)
yt = (M/R/R)*sqrt(R*R-x.*x);                % calculate pdf
del_x = x(3)-x(2);                          % determine bin width
p_hist = N_samp/kpts/del_x;                 % probability from histogram
plot(x,yt,'k',x,p_hist,'ok')                % compare
ylabel('Probability Density')
xlabel('Independent Variable - x')
legend('true pdf','samples from histogram',3)
text = ['The number of points accepted is ',...
num2str(kpts,15),' and N is ',num2str(N,15),'.'];
disp(text)
% End of script file.
```

Executing this program for $N = 3,000$ points yields the results illustrated in Figure 7.12. Of these 3,000 points, 2,301 were accepted and 699 were rejected. This

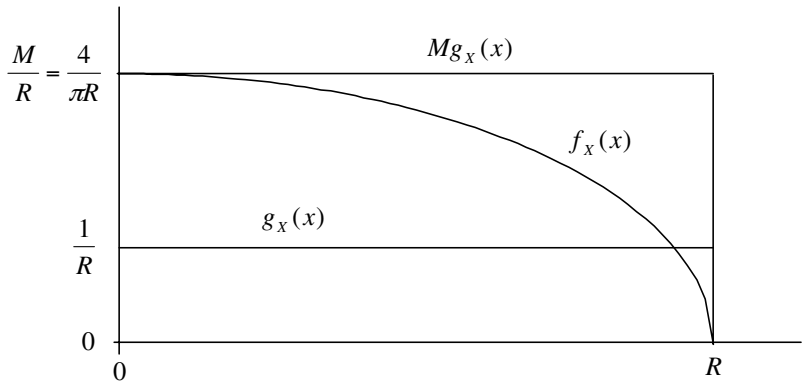


Figure 7.11 Rejection method for Example 7.9.

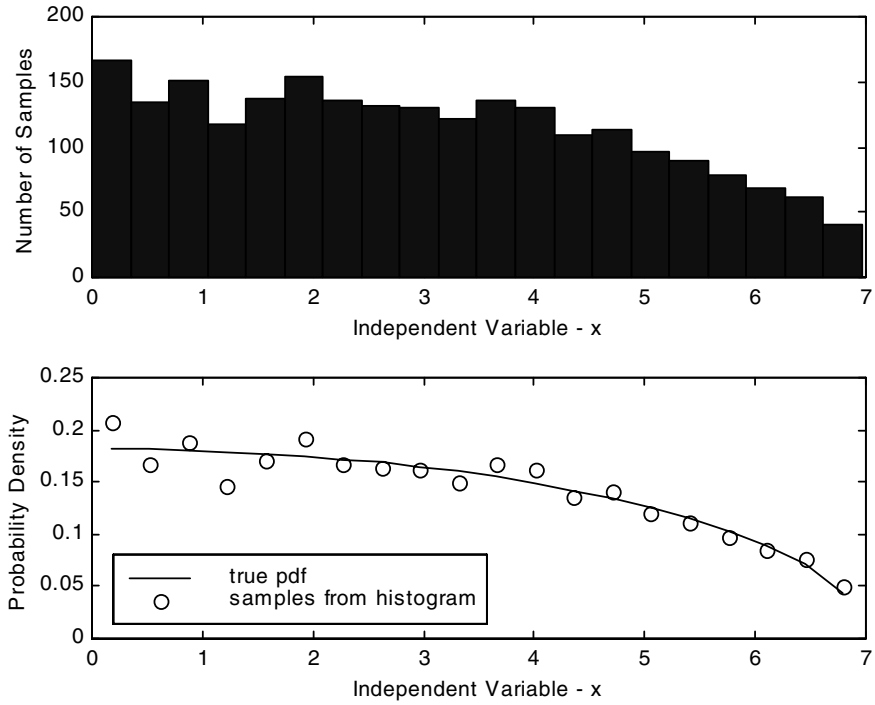


Figure 7.12 Results for rejection exercise.

gives an efficiency of 76.70%, which is close to the theoretical (as $N \rightarrow \infty$) value of 78.54%. ■

Even though we have illustrated the rejection method for the case in which the bounding pdf is uniform, the technique illustrated here is easily modified for the case in which the bounding pdf is not uniform. Ideally, the target pdf should be tightly bounded so that the probability of rejection is minimized. The rejection method works well in cases in which the target pdf has finite support (is nonzero only over a finite range). Finite support is, however, not necessary and the rejection technique, as illustrated here, can easily be extended to the infinite support case. A nice treatment of the rejection method is given in Rubinstein [2].

7.4 Generating Uncorrelated Gaussian Random Numbers

We know from our study of communication systems that the Gaussian random variable is frequently encountered and represents an appropriate model for thermal noise and a number of other phenomena. Gaussian noise generators are a fundamental

building block in many simulations and, as a result, a number of techniques have been developed for producing Gaussian random variables. The CDF is

$$F_X(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{y^2}{2\sigma^2}\right) dy = 1 - Q\left(\frac{x}{\sigma}\right) \tag{7.61}$$

where $Q(x)$ is the Gaussian Q-function defined by

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_{x}^{\infty} \exp(-y^2/2)\, dy \tag{7.62}$$

Since the Gaussian Q-function cannot be written in closed form, the inverse transform technique cannot be used. Rejection techniques can be applied but are not efficient. Thus, we seek other techniques for generating Gaussian random variables.

7.4.1 The Sum of Uniforms Method

The central limit theorem (CLT) provides an attractive avenue for developing a random variable having a Gaussian pdf. The CLT states that, under rather general conditions, the pdf of the sum of N independent random variables will converge to a Gaussian random variable as $N \to \infty$ [8].

Assume that we have N independent uniform random variables, U_i , $i = 1, 2, ..., N$. From these N uniform random variables we form

$$Y = B \sum_{i=0}^{N} \left(U_i - \frac{1}{2}\right) \tag{7.63}$$

where B is a constant that establishes the variance of Y. From the CLT, we know that Y converges to a Gaussian random variable as $N \to \infty$. Since $E\{U_i\} = \frac{1}{2}$, the mean of Y is

$$E\{Y\} = B \sum_{i=0}^{N} \left(E\{U_i\} - \frac{1}{2}\right) = 0 \tag{7.64}$$

The variance of Y is found by first noting that the variance of $U_i - \frac{1}{2}$ is

$$\text{var}\left\{U_i - \frac{1}{2}\right\} = \int_{-1/2}^{1/2} x^2 dx = \frac{1}{12} \tag{7.65}$$

Since the component random variables U_i are assumed independent

$$\sigma_y^2 = B^2 \sum_{i=1}^{N} \text{var}\left\{U_i - \frac{1}{2}\right\} \tag{7.66}$$

we have

$$\sigma_y^2 = \frac{NB^2}{12} \tag{7.67}$$

Thus, given the value of N, the variance of Y, σ_y^2, can be set to any desired value by the proper selection of B. The selection is N is a tradeoff between speed and the accuracy of the tails of the resulting pdf. The value of N is often set to 12, since $N = 12$ gives the simple result $B = \sigma_y$.

While the procedure of generating a Gaussian random variable based on the central limit theorem is straightforward, several significant difficulties occur when one attempts to apply it to practical problems in digital communications. First, since $U_i - \frac{1}{2}$ varies from $-\frac{1}{2}$ to $\frac{1}{2}$, it follows from (7.63) that Y varies from $-BN/2$ to $BN/2$. Thus, even though (7.63) may closely approximate the pdf of a Gaussian random variable in the neighborhood of the mean, the tails of the pdf are truncated to $\pm BN/2$. If one is simulating a digital communication system for the purpose of determining the probability of symbol error, the tails of the pdf are important, since the tails of the pdf represent the large noise values that result in transmission errors. The effect of the truncation of the tails of the pdf of Y can be minimized by choosing N sufficiently large.

Specifically, from (7.67), the value of B is

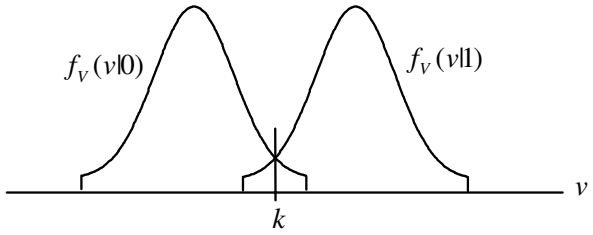
$$B = \sigma_y \sqrt{\frac{12}{N}} \tag{7.68}$$

Therefore the "approximately Gaussian" pdf is truncated so that it is nonzero only in the range

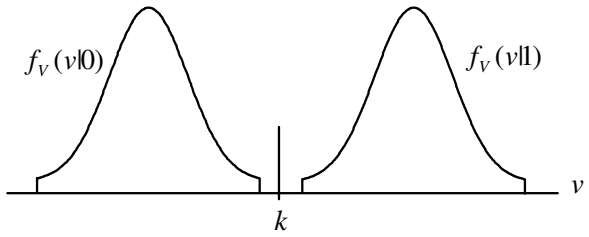
$$\pm \sigma_y \frac{N}{2} \sqrt{\frac{12}{N}} = \pm \sigma_y \sqrt{3N} \tag{7.69}$$

With $N = 100$, the Gaussian random variable is truncated at $\pm 17.32\sigma_y$. For some applications, truncation at 17 standard deviations may still yield significant errors. The appropriate value of N is application dependent.

The difficulty with truncating the tails of the probability density function of the noise in a digital communications system is illustrated in Figure 7.13, which shows the conditional pdfs of the output of a matched filter receiver for a low receiver input SNR and for a high receiver input SNR. (The tails of the pdfs are not actually discontinuous. Figure 7.13 is drawn to emphasize the effect of truncation.) The pdfs are conditioned on a binary 0 transmitted, denoted $f_V(v|0)$, and on a binary 1 transmitted, denoted $f_V(v|1)$. Figure 7.13 is constructed assuming that the noise variance is constant and that the SNR is adjusted by varying the signal power. For the case in which the receiver input SNR is sufficiently low, as in Figure 7.13(a), there is considerable overlap of the conditional pdfs and the probability of error may be determined with reasonable accuracy. As the signal power increases, the conditional pdfs are pushed farther apart and the simulation accuracy degrades. Due to the truncation of the tails of the conditional pdfs, increasing the signal power eventually results in a situation in which the conditional pdfs no longer overlap. This is illustrated in Figure 7.13(b). If the conditional pdfs do not overlap, the probability of error is zero independent of the SNR. This is clearly not a realistic situation.



(a) Low receiver input SNR



(b) High receiver input SNR

Figure 7.13 Pdf of matched filter output for low receiver input SNR and for high reciever input SNR.

Choosing N large, however, leads to the second difficulty with using (7.63) to approximate a Gaussian random variable. Since it takes N calls to the uniform random number to generate a single value of X, use of the algorithm defined by (7.63) can require excessive CPU time.

The two redeeming features of (7.63) are that it does a good job of approximating a Gaussian random variable in the neighborhood of the mean of Y, and that Y will be approximately Gaussian even if the pdfs of the constitute random variables, U_i, are not uniform.

While we are mainly concerned with simulation using serial processing with a single CPU, this is a good place to point out that algorithms that are not suitable for traditional serial processing applications may be quite suitable for use with parallel processing machines. For example, if a certain parallel processing machine uses 100 CPUs, it is possible to generate 100 values of a uniform random variable in the same time required to generate a single value of a uniform variate. Summing 100 uniform random variables may result in an excellent approximation to a Gaussian random variable for most applications and, using parallel processing, can be accomplished very quickly. This is just one example in which the choice of algorithm depends on the computational environment.

7.4.2 Mapping a Rayleigh RV to a Gaussian RV

From Example 7.8 we know that a Rayleigh random variable R can be generated from a uniform random variable U through the use of the transformation $R = \sqrt{-2\sigma^2 \ln U}$. We now consider the problem of mapping a Rayleigh random variable to a Gaussian random variable.

Assume that X and Y are two independent Gaussian random variables having equal variance σ^2. Since X and Y are independent, the joint pdf is the product of the marginal pdfs. Thus

$$\begin{aligned} f_{XY}(x,y) &= \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{x^2}{2\sigma^2}\right) \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{y^2}{2\sigma^2}\right) \\ &= \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \end{aligned} \tag{7.70}$$

With $x = r\cos\theta$ and $y = r\sin\theta$ we have

$$x^2 + y^2 = r^2 \tag{7.71}$$

and

$$\theta = \tan^{-1}\left(\frac{y}{x}\right) \tag{7.72}$$

The joint pdf $f_{R\Theta}(r,\theta)$ is found from $f_{XY}(x,y)$ by the transformation

$$f_{R\Theta}(r,\theta)\, dA_{R\Theta} = f_{XY}(x,y)\, dA_{XY} \tag{7.73}$$

where $dA_{R\Theta}$ and dA_{XY} represent differential areas in the R,Θ and X,Y planes, respectively. It follows from (7.73) that

$$f_{R\Theta}(r,\theta) = f_{XY}(x,y) \frac{dA_{XY}}{dA_{R\Theta}}\bigg|_{\substack{x=r\cos\theta \\ y=r\sin\theta}} \tag{7.74}$$

The ratio of the differential areas is the Jacobian of the transformation, which is

$$\frac{dA_{XY}}{dA_{R\Theta}} = \frac{\partial(x,y)}{\partial(r,\theta)} = \begin{vmatrix} dx/dr & dx/d\theta \\ dy/dr & dy/d\theta \end{vmatrix} \tag{7.75}$$

This gives

$$\frac{dA_{XY}}{dA_{R\Theta}} = \begin{vmatrix} \cos\theta & -r\sin\theta \\ \sin\theta & r\cos\theta \end{vmatrix} = r \tag{7.76}$$

Thus:

$$f_{R\Theta}(r,\theta) = \frac{r}{2\pi\sigma^2} \exp\left(-\frac{r^2}{2\sigma^2}\right), \quad 0 \le r < \infty, \quad 0 \le \theta < 2\pi \tag{7.77}$$

We now examine the marginal pdfs of R and Θ.

The pdf of R is

$$f_R(r) = \int_0^{2\pi} \frac{r}{2\pi\sigma^2} \exp\left(-\frac{r^2}{2\sigma^2}\right) d\theta = \frac{r}{\sigma^2} \exp\left(-\frac{r^2}{2\sigma^2}\right), \quad 0 \le r < \infty \tag{7.78}$$

and the pdf of Θ is

$$f_\Theta(\theta) = \int_0^{\infty} \frac{r}{2\pi\sigma^2} \exp\left(-\frac{r^2}{2\sigma^2}\right) dr = \frac{1}{2\pi}, \quad 0 \le \theta < 2\pi \tag{7.79}$$

Thus, R is a Rayleigh random variable and Θ is uniform. Since a Rayleigh random variable is generated from two orthogonal Gaussian random variables, it follows that orthogonal projections of a Rayleigh random variable produce a pair of Gaussian random variables. Therefore, assuming that R is Rayleigh and Θ is uniform over $(0, 2\pi)$, Gaussian random variables X and Y can be produced using

$$X = R\cos\Theta \tag{7.80}$$

and

$$Y = R\sin\Theta \tag{7.81}$$

Both X and Y are zero-mean random variables and both have variance σ^2. Since they are uncorrelated and Gaussian, it follows that X and Y are statistically independent. Thus, a pair of independent Gaussian random variables X and Y are generated from a pair of uniformly distributed random variables U_1 and U_2 by the algorithm

$$X = \sqrt{-2\sigma^2 \ln(U_1)} \cos 2\pi U_2 \tag{7.82}$$

and

$$Y = \sqrt{-2\sigma^2 \ln(U_1)} \sin 2\pi U_2 \tag{7.83}$$

where we have used (7.46) for R.

A MATLAB program for implementing the Box-Muller algorithm follows:

```
% File: c7_boxmul.m
function [out1,out2]=c7_boxmul(N)
u1 = rand(1,N);                          % generate first uniform RV
u2 = rand(1,N);                          % generate second uniform RV
ray = sqrt(-2*log(u1));                  % generate Rayleigh RV
out1 = ray.*cos(2*pi*u2);                % first Gaussian output
out2 = ray.*sin(2*pi*u2);                % second Gaussian output
% End of function file.
```

7.4.3 The Polar Method

Another algorithm for generating a pair of uncorrelated zero-mean Gaussian random variables is the polar method [9]. The polar algorithm consists of the following steps:

1. Generate two independent random variables, U_1 and U_2, both of which are uniform on the interval (0,1).

2. Let $V_1 = 2U_1 - 1$ and $V_2 = 2U_2 - 1$ so that V_1 and V_2 are independent and uniform on (-1,1).

3. Form $S = \sqrt{V_1^2 + V_2^2}$. If $S < 1$ proceed to step 4. If $S \geq 1$ discard S and go back to step 1.

4. Form $A(S) = \sqrt{(-2\sigma^2 \ln S)/S}$.

5. Set $X = A(S)V_1$ and $Y = A(S)V_2$.

The MATLAB code for generating a pair of Gaussian random vectors using the polar method follows:

```
% File: c7_polar.m
function [out1,out2]=c7_polar(N)
u1 = rand(1,N); u2 = rand(1,N);              % generate uniform RVs
v1 = 2*u1-1; v2 = 2*u2-1;                    % make uniform in -1 to +1
outa = zeros(1,N); outb = zeros(1,N);        % allocate memory
j = 1;                                       % initialize counter
for i=1:N
   s(i) = v1(i)^2 + v2(i)^2;                 % generate s
   if s(i) $<$ 1                             % test
   j = j+1;                                  % increment counter
   a(i) = sqrt((-2*log(s(i)))/s(i));
   outa(j) = a(i)*v1(i);                     % first Gaussian RV
   outb(j) = a(i)*v2(i);                     % second Gaussian RV
   end
end
out1 = outa(1,1:j); out2 = outb(1,1:j);      % truncate arrays
% End of function file.
```

Note that the MATLAB library function `rand` was used to generate the required pair of uniform random variables in the preceding example code. Obviously a user-supplied uniform random number generator, based on LCG techniques, could have been used.

The polar method is an example of a rejection method, since, as we see from step 3, some values of S are rejected. Thus, fewer than N random variables will be generated with each call to this function. The probability of rejection is easily determined from Figure 7.14. The random variables V_1 and V_2 are uniformly

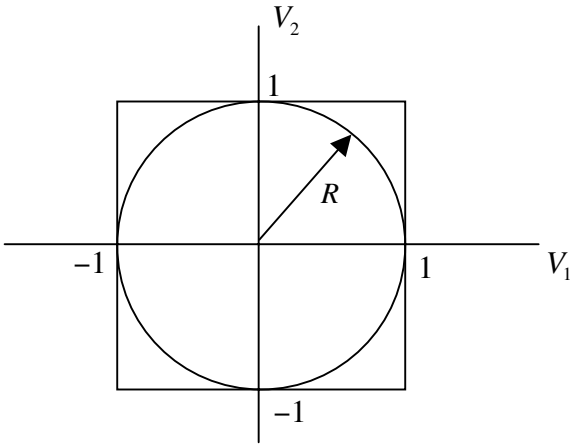


Figure 7.14 Polar method.

distributed over the box, having area $A_{box} = 4$, that bounds the circle of radius $R = 1$ and area $A_{circle} = \pi$. The probability that S is not rejected is therefore

$$\Pr\{\text{Rejection}\} = 1 - \frac{A_{circle}}{A_{box}} = 0.2146 \tag{7.84}$$

The polar algorithm often provides results that have better correlation properties than the Box-Muller algorithm. There is, however, a disadvantage with the polar method. Note that N calls to the Box-Muller algorithm will generate N pairs of Gaussian random variables. If the polar algorithm is called N times, the number of pairs of Gaussian random variables generated will be a random variable having mean $(\pi/4)N$. The fact that an unknown number of calls must be made to the polar algorithm in order to generate a given number of Gaussian random variables often complicates the simulation program.

7.4.4 MATLAB Implementation

Prior to the release of MATLAB 5, the Gaussian random number contained in the MATLAB library, `randn`, made use of the minimum standard number generator given in (7.24) and the polar method to map the uniformly distributed random numbers to random numbers having a Gaussian distribution. Starting with MATLAB 5, a completely different algorithm, involving no multiplications or divisions, replaced the previously used algorithm. The new random number generator produces numbers having a Gaussian pdf directly without requiring a transformation of uniformly distributed random numbers. Since multiplication and division are not used in the algorithm, and the uniform to Gaussian step is not needed, it is very

fast. The algorithm used in later versions of MATLAB is a refined version of the basic Ziggurt algorithm [1] and is described in detail in a paper by Marsaglia and Tsang [10].

7.5 Generating Correlated Gaussian Random Numbers

So far, the goal has been to generate uncorrelated random numbers. We now turn our attention to the situation in which the goal is to generate random numbers that have a Gaussian pdf and are correlated. We first examine a simple technique for generating two sequences that are related through a given correlation coefficient (first-order correlation). We then turn our attention to the more general case in which the generated sequence is to have a given power spectral density (PSD). Establishing the PSD is, of course, equivalent to establishing a given autocorrelation function.

7.5.1 Establishing a Given Correlation Coefficient

We have seen two methods for generating a pair of (approximately) uncorrelated Gaussian random variables. It is an easy task to map a pair of uncorrelated Gaussian random variables, denoted X and Y, to a pair of Gaussian random variables having a specified level of correlation. Assuming that X and Y are zero mean and uncorrelated, the next step is to generate a third random variable Z, defined by

$$Z = \rho X + \sqrt{1-\rho^2}Y \tag{7.85}$$

where ρ is a parameter with $|\rho| \leq 1$. With this algorithm, X and Z are zero mean random variables with equal variance. We will show that ρ is the correlation coefficient relating X and Z.

The proof is simple. It is clear that Z is a Gaussian random variable, since it is a linear combination of Gaussian random variables. It also follows that Z is zero mean if X and Y are zero mean. The variance of Z is

$$\begin{aligned}\sigma_Z^2 &= E\left\{[\rho X + \sqrt{1-\rho^2}Y]^2\right\} \\ &= \rho^2 E\left\{X^2\right\} + 2\rho\sqrt{1-\rho^2}E\left\{XY\right\} + (1-\rho^2)E\left\{Y^2\right\}\end{aligned} \tag{7.86}$$

Since $E\{XY\} = E\{X\}E\{Y\} = 0$ and $\sigma_X^2 = \sigma_Y^2 = \sigma^2$, the preceding becomes

$$\sigma_Z^2 = \rho^2\sigma^2 + (1-\rho^2)\sigma^2 = \sigma^2 \tag{7.87}$$

The covariance $E\{XZ\}$ is

$$\begin{aligned}E\{XZ\} &= E\{X[\rho X + (1-\rho)Y]\} \\ &= \rho E\left\{X^2\right\} + (1-\rho)E\{XY\} \\ &= \rho E\left\{X^2\right\} = \rho\sigma^2\end{aligned} \tag{7.88}$$

where the last step follows because X and Y are independent and zero mean. The correlation coefficient ρ_{XZ} is

$$\rho_{XZ} = \frac{E\{XZ\}}{\sigma_X \sigma_Z} = \frac{\rho\sigma^2}{\sigma^2} = \rho \tag{7.89}$$

as desired.

7.5.2 Establishing an Arbitrary PSD or Autocorrelation Function

The general technique used for establishing a sequence of random numbers with a given autocorrelation function, or equivalently a given PSD, is to filter a set of uncorrelated samples so that the target PSD is established. Uncorrelated samples have, by definition, a PSD that is constant over the simulation bandwidth $|f| < f_s/2$. The variance is, by definition, the area under $S_n(f)$. This is, as illustrated in Figure 7.15,

$$\sigma_n^2 = \frac{1}{2} N_0 f_s \tag{7.90}$$

Thus, in order to establish a given noise PSD, N_0, the random number (noise) generator variance, and the sampling frequency must satisfy

$$f_s = \frac{2\sigma_n^2}{N_0} \tag{7.91}$$

Thus, the required noise generator variance for a given PSD is a function of the sampling frequency.

The establishment of a desired PSD for a random sequence is relatively straightforward using a fundamental result from basic stochastic process theory. We know that if the input to a linear system is a random process having a PSD $S_X(f)$, the PSD of the system output is

$$S_Y(f) = |H(f)|^2 S_X(f) \tag{7.92}$$

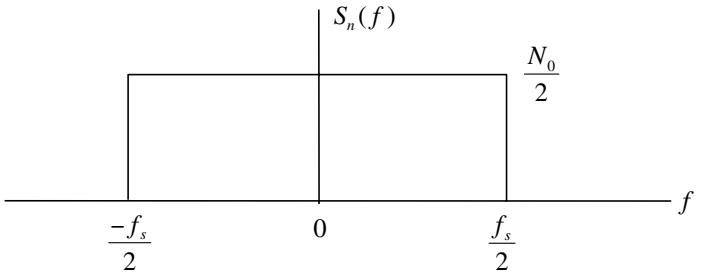


Figure 7.15 PSD for independent samples.

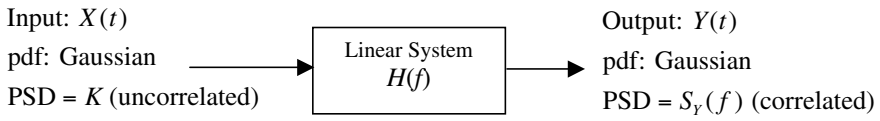


Figure 7.16 Generation of a correlated random sequence.

where $H(f)$ is the transfer function of the linear system. This is illustrated in Figure 7.16. If the noise samples are independent, the PSD of the input is constant at $K = N_0/2$ Watts/Hz,

$$S_Y(f) = |H(f)|^2 K \tag{7.93}$$

and the required $H(f)$ to establish the target PSD is

$$H(f) = \sqrt{S_Y(f)/K} \tag{7.94}$$

Therefore, the problem of shaping the power spectral density to meet a given requirement reduces to the problem of finding a filter with a transfer function $H(f)$ so that $H^2(f)$ gives the required spectral shape.

Example 7.10. A convenient method for synthesizing a filter having a given transfer function is to determine the best fit, in the minimum mean-square error sense, to the transfer function. Solving the Yule-Walker equations using the MATLAB function `yulewalk` accomplishes this task. Specifically, the function `yulewalk` determines the filter coefficients b_k and a_k so that the transfer function

$$H(z) = \frac{b_0 + b_1 z^{-1} + \cdots + b_n z^{-n}}{1 + a_1 z^{-1} + \cdots + a_n z^{-n}} \tag{7.95}$$

is the minimum mean-square error fit to a given transfer function.

In order to illustrate the technique, assume that a given PSD is to have the form $S(f) = K/f$. This is referred to as flicker or one-over-f noise [11] and is often used to model phase noise in oscillators. In order to generate this noise, we must generate a filter having a transfer function of the form $H(f) = K/\sqrt{f}$. Passing white noise through this filter generates the required flicker noise process. Since $H(f) \to \infty$ as $f \to 0$, the transfer function is defined as

$$H(f) = \begin{cases} 0, & |f| < f_0 \\ K/\sqrt{f}, & |f| > f_0 \end{cases} \tag{7.96}$$

The following MATLAB program generates an approximation to the required $H(f)$ in which frequency is normalized to the Nyquist frequency f_N and $f_0 = f_N/20$:

```
% File:  c7_flicker.m
f = 0:100;                          % frequency points
fn = 100;                           % Nyquist rate
```

```
F = f/fn;                          % frequency vector
M = abs(100./sqrt(f));             % normalized fequency response
M = [zeros(1,6),M(6:100)];         % bound from zero frequency
[b1,a1] = yulewalk(3,F,M);         % generate order=3 filter
[b2,a2] = yulewalk(20,F,M);        % generate order=20 filter
[h1,w1] = freqz(b1,a1);            % generate 3-rd order H(f)
[h2,w2] = freqz(b2,a2);            % generate 20-th order H(f)
subplot(2,1,1)
plot(F,M,':',w1/pi,abs(h1))
xlabel('Normalized Frequency')
ylabel('Squared Magnitude Response')
subplot(2,1,2)
plot(F,M,':',w2/pi,abs(h2))
xlabel('Normalized Frequency')
ylabel('Squared Magnitude Response')
%End of script file.
```

Executing the program yields the results illustrated in Figure 7.17. The desired transfer function is illustrated by the dashed line. The third-order approxima-

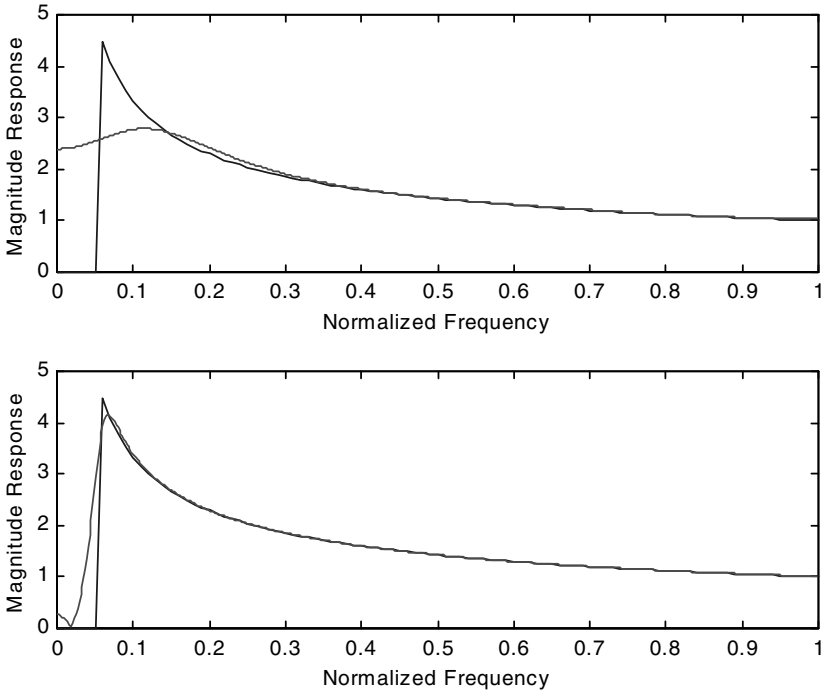


Figure 7.17 Generation of flicker noise.

tion is illustrated in the top pane and the twentieth-order approximation, which is clearly superior to the third-order approximation, is illustrated in the bottom pane. Note that a low-order approximation is adequate in the region where $H(f)$ is relatively smooth. In a frequency region where $H(f)$ is changing rapidly, a higher-order approximation is required. This approach is a powerful tool for generating arbitrary PSDs. ■

Example 7.11. The simulation of wireless systems, in which motion is present, requires the generation of a process having the power spectral density

$$S(f) = \begin{cases} \frac{1}{\sqrt{1-(f/f_d)^2}}, & |f| < f_d \\ 0, & \text{otherwise} \end{cases} \tag{7.97}$$

to represent the effect of doppler. The quantity f_d in (7.97) represents the maximum doppler frequency. As illustrated in (7.94), the required filter transfer function is

$$H(f) = \begin{cases} \left[1-(f/f_d)^2\right]^{-1/4}, & |f| < f_d \\ 0, & \text{otherwise} \end{cases} \tag{7.98}$$

This filter is implemented as an FIR filter whose impulse response is obtained by taking the inverse DFT of sampled values of (7.98).

The MATLAB program for generating the impulse response of the Jakes filter, and filtering white noise with the filter, is contained in Appendix A. Executing the code yields the results illustrated in Figures 7.18 and 7.19. Figure 7.18 illustrates

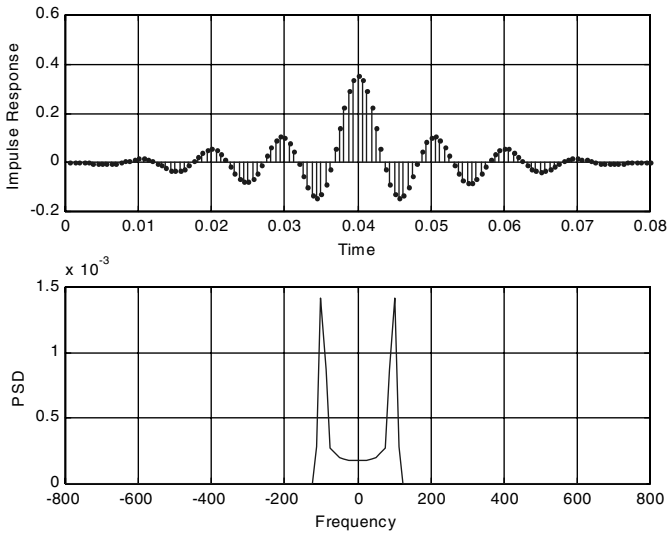


Figure 7.18 Impulse response and target PSD.

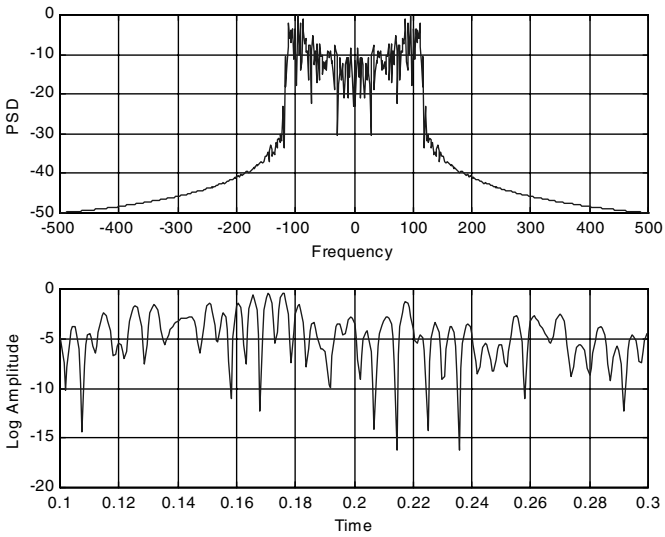


Figure 7.19 Estimated PSD and envelope function.

the impulse response and the transfer function $H(f)$ (bottom pane). Figure 7.19 illustrates the effect of passing complex white noise through the filter. The top pane shows the estimated PSD at the filter output. The bottom pane illustrates the magnitude of the envelope on a log scale. In a wireless communication system, this corresponds to the fading envelope. ■

7.6 Establishing a pdf and a PSD

Generating a noise waveform that simultaneously satisfies a given PSD and pdf requirement is, in general, a difficult task. There is, however, one situation in which this problem is not difficult. If the pdf of the system output is to be Gaussian, one can simply generate a sample sequence for the filter input in which the samples are Gaussian and independent. Since the input samples are independent, the PSD will be constant at K Watts/Hz. The PSD can be shaped using a linear filter, as described in the previous section. Since the PSD shaping filter is linear, the pdf of the output will be Gaussian. We have this simple result because any linear transformation of a Gaussian process yields another Gaussian process [8]. Fortunately, many practical problems can be handled in this manner.

If both the pdf and the PSD of a target waveform are specified and the pdf is required to be something other than Gaussian, the problem is much more difficult. A solution to this problem was proposed by Sondhi [12]. The basic scheme for implementing the Sondhi algorithm is illustrated in Figure 7.20. As always, we start with a sequence of samples $u[n]$ that are uniformly distributed on the interval

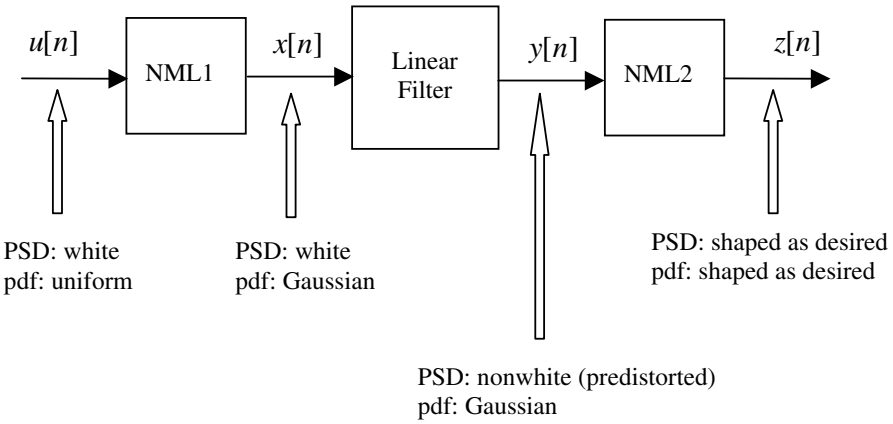


Figure 7.20 The Sondhi algorithm.

(0,1). In addition, the sequence $\{u[n]\}$ is assumed to be delta correlated so that the PSD is white. The job of the first memoryless nonlinearity, denoted MNL1, is to map the sequence $\{u[n]\}$ into a sequence $\{x[n]\}$, which is white but has a Gaussian pdf. We have studied several techniques for accomplishing this mapping. The filtering operation predistorts the spectrum so that the PSD is $S_Y(f)$. Since the filter is linear, the pdf of the sequence $\{y[n]\}$ remains Gaussian. The basic operation of the second memoryless nonlinearity, denoted MNL2, is to map the Gaussian pdf of the sequence $\{y[n]\}$ to the final desired pdf. However, the second nonlinearity also affects $S_Y(f)$. Thus, the filter must modify the predistorted PSD $S_Y(f)$, so that passing the sample sequence through the second nonlinearity will result in the sequence $\{z[n]\}$ having both the desired PSD and pdf.

A detailed example of the Sondhi algorithm is beyond the scope of our current effort and the interested student is referred to [12]. An interesting simple example in which a number generator is developed having a first-order (single pole) PSD is given in the paper by Coates et al. [3].

7.7 PN Sequence Generators

Pseudonoise (PN) sequence generators are used in a number of applications, especially in the area of synchronization. As one application, PN sequences are used for approximating a random variable having a uniform probability density function. A PN sequence generator can take a number of forms [13] but the most common form, and the one upon which we shall concentrate, is illustrated in Figure 7.21.

In the simulation context, the most important reason for using PN sequences is for modeling data sources. By using a PN sequence generator almost all possible bit combinations having a given length can be can be produced over the shortest

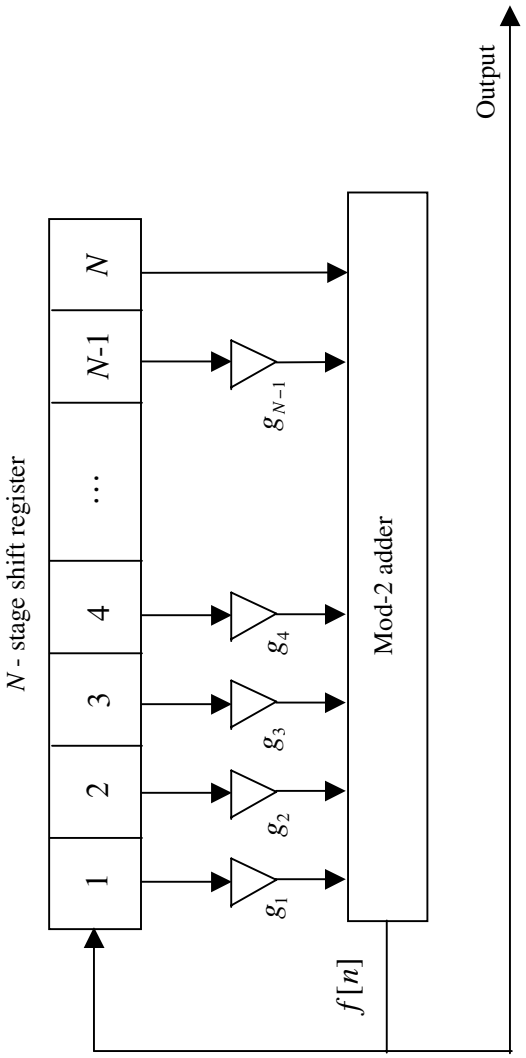


Figure 7.21 PN sequence generator.

possible simulation length. We will use this fact to study the impact of intersymbol interference (ISI) when we consider semianalytic simulation techniques in Chapter 10.

A PN sequence generator consists of three basic components: an N-stage shift register, a mod-2 adder, and a connection vector that defines the connections between specific shift register stages and the mod-2 adder. The connection vector establishes the performance characteristics of the generator and is defined by the polynomial

$$g(D) = 1 + g_1 D + g_2 D^2 + \cdots g_{N-1} D^{N-1} + D^N \tag{7.99}$$

If $g_i = 1$, a connection exists between the i^{th} stage of the shift register and the mod-2 adder. If $g_i = 0$, there is no connection. Note that in the polynomial $g(D)$, both g_0 and g_N are equal to 1.

It can be shown that the maximum period of the PN sequence generator output is

$$L = 2^N - 1 \tag{7.100}$$

and is achieved if, and only if, the polynomial $g(D)$ is primitive [13].[7] The autocorrelation function $R[m]$ of the output of the PN sequence generator is illustrated in Figure 7.22, in which we assume that the data values (symbols) are ± 1. For $m = 0$ or a multiple of L, the autocorrelation is one. For $0 < m < L$, the autocorrelation $R[m]$ is $-1/L$, which for large L is approximately zero. Thus, for PN sequences having a large period, the autocorrelation function approaches an impulse. The PSD is therefore approximately white as desired. The PN sequence has many interesting properties, many of which are summarized in [13]. Several properties of interest in the simulation context include the following:

- The sequence in nearly balanced. In other words, in one period of the sequence, the number of ones will exceed the number of zeros by 1. By adding an extra zero at the right point, the sequence can be balanced.

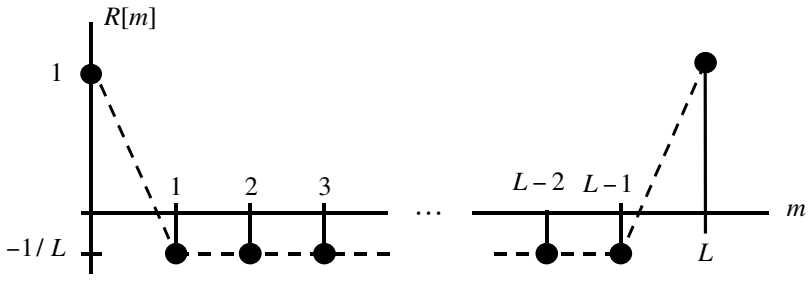


Figure 7.22 Autocorrelation of register contents for a maximal length PN sequence.

[7] A polynomial $g(D)$ of degree N is primitive if the smallest integer k for which $g(D)$ divides $D^k + 1$ without remainder is $k = 2^N - 1$.

- All possible bit combinations appear within one period, except for the fact that there is no sequence of N zeros but there is a sequence of N ones. (Note that if all shift registers contain a binary zero, the generator will become stuck in the all-zero state. By adding an extra zero at the point where there are $N-1$ zeros, the sequence can be balanced. The result will be a deBruijn sequence [14]).

- The autocorrelation function, although periodic, is very nearly the same as that of a random binary waveform.

The design of a PN sequence generator based on an N-stage shift register reduces to finding a primitive polynomial of degree N. For large N this can be a very difficult task. Fortunately, extensive tables of primitive polynomials are available in the literature [13]. A small sample, represented in a form consistent with (7.99), is given in Table 7.1.

Developing a simulation program for the PN sequence generator is quite simple. First, let the shift register contents be represented by the vector

$$B = [\ b_1 \quad b_2 \quad \cdots \quad b_{N-1} \quad b_N\ ] \tag{7.101}$$

and let the connection vector be represented by

$$G = [\ g_1 \quad g_2 \quad \cdots \quad g_{N-1} \quad g_N\ ] \tag{7.102}$$

The output of the mod-2 adder, as illustrated in Figure 7.21, is the feedback symbol $f[n]$. It is defined by

$$f[n] = \sum_{i=1}^{N} b_i g_i = B(G^T) \tag{7.103}$$

Table 7.1 Short Table of Primitive Polynomials

N	g_1	g_2	g_3	g_4	g_5	g_6	g_7	g_8	g_9	g_{10}	g_{11}	g_{12}	g_{13}	g_{14}
3	1	0	1											
4	1	0	0	1										
5	0	1	0	0	1									
6	1	0	0	0	0	1								
7	0	0	1	0	0	0	1							
8	0	1	1	1	0	0	0	1						
9	0	0	0	1	0	0	0	0	1					
10	0	0	1	0	0	0	0	0	0	1				
11	0	1	0	0	0	0	0	0	0	0	1			
12	1	0	0	1	0	1	0	0	0	0	0	1		
13	1	0	1	1	0	0	0	0	0	0	0	0	1	
14	1	0	0	0	0	1	0	0	0	1	0	0	0	1

The feedback signal is also the next value of b_1. The register must obviously be initialized with a vector B in which at least one value of b_j is nonzero.

Example 7.12. Our goal in this example is to design a PN sequence generator with $N = 10$. With $g(D)$ primitive, the period will be, from (7.100), $L = 1{,}023$. The connection vector is, from Table 7.1,

$$G = [\ 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1\ ] \tag{7.104}$$

indicating a connection from the third and last stages of the shift register. This yields the configuration shown in Figure 7.23. The MATLAB code for simulating the PN generator is as follows:

```
% File:  c7_PNdemo.m
pntaps = [0 0 1 0 0 0 0 0 0 1];     % shift register taps
pninitial = [0 0 0 0 0 0 0 0 0 1];  % initial shift register state
pndata = zeros(1,1023);             % initialize output vector
samp_per_sym = 1;                   % samples per symbol
pnregister = pninitial;             % initialize shift register
n = 0;                              % initialize counter
kk = 0;                             % set terminator indicator
while kk == 0
n = n+1;                                    % increment counter
pndata(1,n) = pnregister(1,1);              % save output
feedback = rem((pnregister*pntaps'),2);     % calculate feedback
pnregister = [feedback,pnregister(1,1:9)];  % increment register
if pnregister == pninitial; kk = 1; end     % reset termination
end
text = ['The period is ',num2str(n,15),'.'];
disp(text) % display period
pndata=replicate(pndata,samp_per_sym);      % replicate data
kn = n*samp_per_sym;                        % output vector length
pndata = 2*pndata - 1;                      % make output +/- one
a = fft(pndata);
b = a.*conj(a);                             % PSD of data
Rm = real(ifft(b))/kn;                      % autocorrelation
x1 = (0:length(Rm)-1)/samp_per_sym;
x2 = 0:100;
subplot(3,1,1)
plot(x1,Rm,'.k'); ylabel('R[m]')
subplot(3,1,2)
stem(x2,Rm(1:101),'.k'); ylabel('Partial R[m]')
subplot(3,1,3)
stem(x2,pndata(1:101),'.k'); ylabel('First 100 outputs')
axis([0 100 -1.5 1.5]);
% End of script file.
```

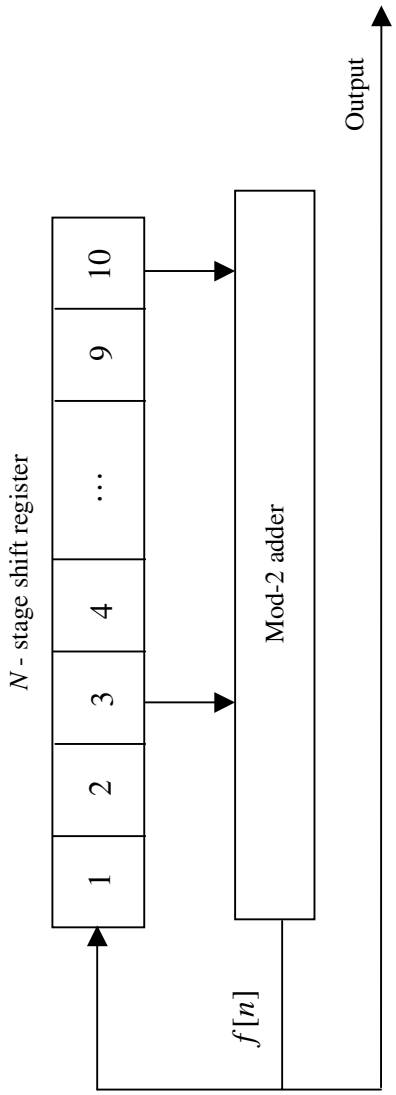


Figure 7.23 PN sequence generator for Example 7.5.

Note that the program performs a test to ensure that the PN sequence generator is indeed full period. (For this case, executing the program returns the period as 1023, which is full period for $N = 10$.) The program also generates a plot of the autocorrelation function, the first 101 samples of the autocorrelation, and the first 101 samples at the generator output. These results for one sample/symbol are illustrated in Figure 7.24. Changing the sampling rate to five samples results in Figure 7.25. ■

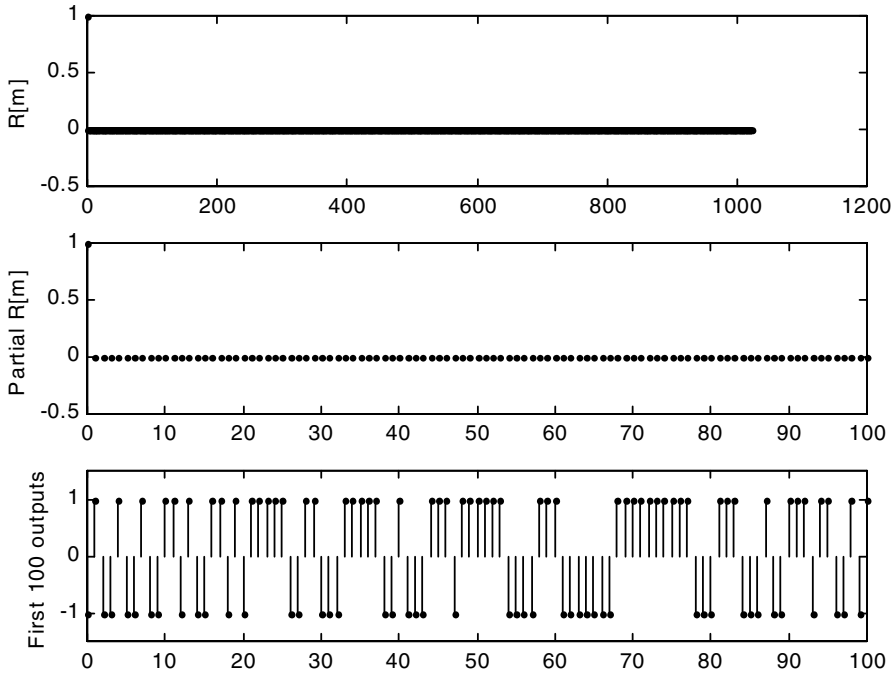


Figure 7.24 Generated plots for one sample per symbol.

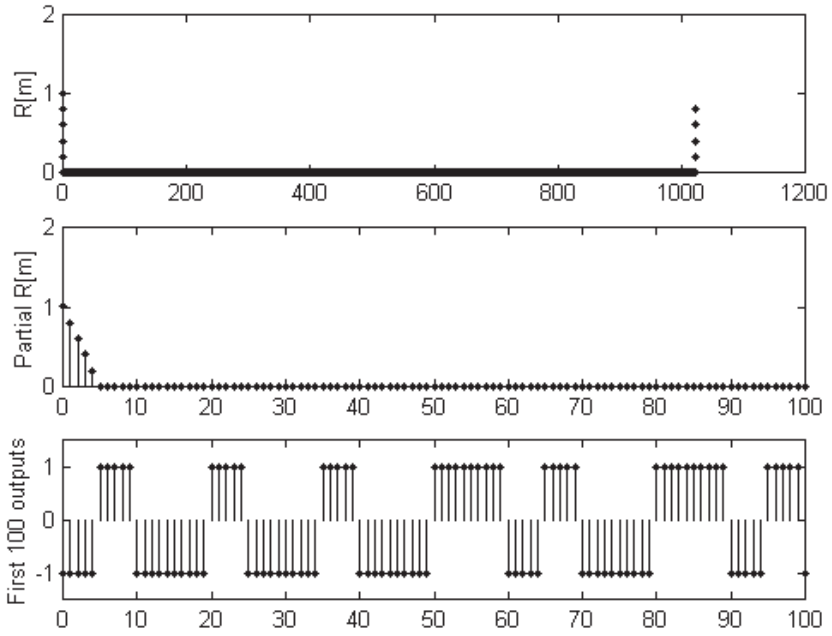


Figure 7.25 Generated plots for five samples per symbol.

7.8 Signal Processing

In this section we examine several input-output relationships for linear systems for the case in which the system input, and consequently the system output, is random. We have interest in the following basic results:

- Relationship between the mean of the system input and the system output
- Relationship between the variance of the system input and the system output
- Input-output cross-correlation
- Relationship between the autocorrelation and PSD of the system input and the system output

These relationships, which are useful in the study of simulation, give us the basic tools for signal processing for the case in which the system input is a sample function of a random process. Throughout the analyses to follow, we will assume that the system of interest is linear and fixed, and that the input to the system is wide-sense

stationary.[8] More detail on processing random signals can be found in a variety of books. An excellent reference is Oppenheim and Schafer [15].

7.8.1 Input/Output Means

Since the system is assumed linear, output samples $y[n]$ are related to input samples $x[n]$ by the well-known discrete convolution relationship. Thus, as always

$$y[n] = \sum_{k=-\infty}^{\infty} h[k]x[n-k] \tag{7.105}$$

The mean, or dc value, of the output is, by definition

$$E\{y[n]\} = E\left\{\sum_{k=-\infty}^{\infty} h[k]x[n-k]\right\} = \sum_{k=-\infty}^{\infty} h[k]E\{x[n-k]\} \tag{7.106}$$

since the expected value of a sum is the sum of the expected values. We also note that, for a fixed system, the terms representing the unit impulse response $h[k]$ are constants. From the stationarity assumption, $E\{x[n-k]\} = E\{x[n]\}$. This gives the simple result

$$m_y = m_x \sum_{k=-\infty}^{\infty} h[k] \tag{7.107}$$

where m_x and m_y are the mean of $x[n]$ and $y[n]$, respectively. Note that since $\sum_{k=-\infty}^{\infty} h[k] = H(0)$, the preceding expression can be written

$$m_y = H(0)m_x \tag{7.108}$$

where $H(0)$ is the dc gain of the filter.

7.8.2 Input/Output Cross-Correlation

The input-output cross-correlation is defined by

$$E\{x[n]y[n+m]\} = R_{xy}[m] = E\left\{x[n]\sum_{j=-\infty}^{\infty} h[j]x[n-j+m]\right\} \tag{7.109}$$

which is

$$R_{xy}[m] = \sum_{j=-\infty}^{\infty} h[j]E\{x[n]x[n-j+m]\} \tag{7.110}$$

[8] Recall that a wide-sense stationary process is one for which means and variances are independent of the time origin, and the autocorrelation function is only dependent on the time lag between samples.

which is

$$R_{xy}[m] = \sum_{j=-\infty}^{\infty} h[j] R_{xx}[m-j] \tag{7.111}$$

The input-output cross-correlation is used in the development of a number of performance estimators. One of these, an estimator for the signal-to-noise ratio at a point in a system, will be developed in Chapter 8.

7.8.3 Output Autocorrelation Function

By definition, the autocorrelation function of the system output is, at lag m, $E\{y[n]y[n+m]\}$. This gives

$$\begin{aligned} E\{y[n]y[n+m]\} &= R_{yy}[m] \\ &= E\left\{ \sum_{j=-\infty}^{\infty} h[j]x[n-j] \sum_{k=-\infty}^{\infty} h[k]x[n-k+m] \right\} \end{aligned} \tag{7.112}$$

which is

$$R_{yy}[m] = \sum_{j=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} h[j]h[k] E\left\{x[n-j]x[n+m-k]\right\} \tag{7.113}$$

Once again we rely on the assumption that $x[n]$ is stationary to go to the next step. If $x[n]$ is stationary, the autocorrelation $E\left\{x[n-j]x[n-k+m]\right\}$ depends only on the lag $m-k+j$ and

$$R_{yy}[m] = \sum_{j=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} h[j]h[k] R_{xx}(m-k+j) \tag{7.114}$$

where $R_{xx}[m]$ is the autocorrelation of $x[n]$ at lag m. Unfortunately, the expression for $R_{yy}[m]$ cannot, in general, be simplified further without knowledge of the statistics of $x[n]$. The one exception is when $x[n]$ is a delta-correlated (white) sequence. Taking the discrete Fourier transform of both sides of (7.114) yields the input-output PSD relationship

$$S_y(f) = S_x(f) \left|H(f)\right|^2 \tag{7.115}$$

which was used to generate correlated sequences.

If the input sequence is delta-correlated (i.e., white noise), by definition

$$R_{xx}[m] = E\{x[n]x[n+m]\} = \left\{ \begin{array}{cc} \sigma_x^2, & m=0 \\ 0, & m \neq 0 \end{array} \right\} = \sigma_x^2 \delta[m] \tag{7.116}$$

Substitution into (7.114) gives

$$R_{yy}[m] = \sigma_x^2 \sum_{j=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} h[j]h[k]\delta(m-k+j) = \sigma_x^2 \sum_{j=-\infty}^{\infty} h[j]h[j+m] \tag{7.117}$$

where we have used the sifting property of the delta function.

7.8.4 Input/Output Variances

The variance of the system output is, by definition, $R_{yy}[0] = E\left\{y^2[n]\right\}$. For the general case, the output variance follows from (7.114) with $m = 0$. The result is

$$\sigma_y^2 = \sum_{j=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} h[j]h[k]R_{xx}[j-k] \tag{7.118}$$

If $x[n]$ is a delta-correlated (white noise) sequence, σ_y^2 can be found by simply letting $m = 0$ in (7.117). This gives

$$\sigma_y^2 = \sigma_x^2 \sum_{j=-\infty}^{\infty} h^2[j] \tag{7.119}$$

This result will be encountered again when we study equivalent noise bandwidth in Chapter 10.

7.9 Summary

Our goal in this chapter was to explore techniques for generating and processing random signals. The results of our study give us the tools to represent noise, interference, random information-bearing signals, and other random phenomena in communication systems.

First we explored methods for generating a pseudo-random set of uniformly distributed integers. A fundamental technique for generating uniformly distributed integers is the linear congruence generator (LCG). While there are many variations of the LCG, the goal is to generate a sequence having the longest possible period. We also wish to generate a set of integers having the lowest possible correlation. Two tests for correlation were considered: the scatterplot and the Durbin-Watson test. Many other tests are available, but they tend to be more complicated to implement.

After a pseudo-random sequence is generated, the next step is to shape the pdf and the PSD so that a given waveform in the system being simulated can be modeled with a required level of accuracy. Several methods were studied for shaping the pdf. The simplest technique was the inverse transform method, which required that the CDF be known in closed form. Two ad hoc techniques were explored for generating independent pairs of Gaussian random variables. These were the Box-Muller algorithm and the polar algorithm. A technique for emulating experimental data, based on the histogram, was also explored. Shaping the PSD is a filtering operation. The filtering operation preserves the pdf if the pdf is Gaussian. If one must shape both the pdf and the PSD, and if the target pdf is not Gaussian, a difficult problem results. This problem can be solved using the Sondhi algorithm.

Sequences having low correlation can also be generated using the PN sequence generator, which is implemented as a shift register having an appropriately chosen feedback function. In typical applications, the LCG is used to model noise and interference waveforms, and the PN sequence generator is used to model data sources.

7.10 Further Reading

This chapter covered a number of diverse topics. Random number generation is the subject of many papers and books. For an overview, the paper by Park and Miller [5] is strongly recommended. For a detailed treatment, complete with proofs, Knuth [1] is highly recommended. The general subject of mapping uniform pdfs to a specific target pdf is also contained in many books. Rubinstein [2] and Ross [9] are reasonably complete.

7.11 References

1. D. E. Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, 2nd ed., Reading, MA: Addison-Wesley, 1981.

2. R. Y. Rubinstein, *Simulation and the Monte Carlo Method*, New York: Wiley, 1981.

3. R. F. W. Coates, G. J. Janacek, and K. W. Lever, "Monte Carlo Simulation and Random Number Generation," *IEEE Journal on Selected Areas in Communications*, Vol. 6, No. 1, January 1988, pp. 58–66.

4. K. S. Shanmugan and A. M. Breipohl, *Random Signals: Detection, Estimation and Data Analysis*, New York: Wiley, 1988.

5. S. K. Park and K. W. Miller, "Random Number Generators: Good Ones Are Hard to Find," *Communications of the ACM*, Vol. 31, No. 10, October 1988, pp. 1192–1201.

6. P. L'Ecuyer, "Efficient and Portable Combined Random Number Generators," *Communications of the ACM*, Vol. 31, No. 6, June 1988, pp. 742–774.

7. C. Moler, "Random Thoughts: 10^{435} Is a Very Long Time," *Matlab News & Notes*, Natick, MA: MathWorks, Fall, 1995.

8. A. Papoulis, *Probability, Random Variables and Stochastic Processes*, 3rd ed., New York: McGraw-Hill, 1991.

9. S. M. Ross, *A Course in Simulation*, New York: Macmillan, 1990.

10. G. Marsaglia and W. W. Tsang, "A Fast, Easily Implemented Method for Sampling From Decreasing or Symmetric Unimodal Density Functions," *SIAM Journal of Scientific and Statistical Computing*, Vol. 1, No. 2, June 1989, pp. 349–359.

11. H. W. Ott, *Noise Reduction Techniques in Electronic Systems*, New York: Wiley, 1976.

12. M. M. Sondhi, "Random Processes With Specified Spectral Density and First-Order Probability Density," *Bell System Technical Journal*, Vol. 62, 1983, pp. 679–700.

13. R. E. Ziemer and R. L. Peterson, *Introduction to Digital Communication*, 2nd ed., Upper Saddle River, NJ: Prentice Hall, 2001.

14. S. Golomb, *Shift Register Sequences*, Laguna Hills, CA: Aegean Press, 1982.

15. A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, Upper Saddle River, NJ: Prentice Hall, 1989.

7.12 Problems

7.1 Write a MATLAB program to compute the ensemble averages of $x(t)$, $y(t)$, and $z(t)$ as defined in Example 7.1 for $N = 20$, 50, and 100, where N is the number of sample functions. Do this for 51 values of t for $0 \leq t \leq 2$. Discuss the results. For $N = 20$, plot the ensemble average as a function of t. Why is it a sinusoid?

7.2 Consider $y(t)$ as defined in Example 7.1. Compute the ensemble average $E\{y(t)\}$. Estimate $E\{y(t)\}$ using $N = 5$ sample functions at 51 points t_i where $0 \leq t_i \leq 2$. Plot the result and compare with $E\{y(t)\}$. Repeat for $N = 20$, $N = 50$, and $N = 100$. What do you conclude?

7.3 Repeat the preceding problem for $z(t)$ as defined in Example 7.1.

7.4 A random telegraph waveform is defined as a waveform that takes on values $+1$ or -1 with the switching instants occuring at random. Develop a MATLAB program for computing and plotting three sample functions of a random telegraph waveform.

7.5 Using Example 7.2 as a guide, develop a MATLAB program for simulating a 8-PSK modulator.

7.6 Using Example 7.2 as a guide, develop a MATLAB program for simulating a 16-QAM modulator.

7.7 Develop a MATLAB program for testing whether or not a number m is prime. Also develop a program to identify primitive elements mod(m). Using this program, determine which elements are primitive mod(89). Using the results of this investigation develop a multiplicative LCG and show that it has period 88.

7.8 Suppose we wish to develop a uniform number generator with $m = 5{,}000$, $a = 241$, and $c = 1,323$. Unfortunately, a small error results in the multiplier a being entered as 240. Using the seed $x_0 = 1$, describe the impact of this error.

7.9 Design a mixed congruential generator having a period of 6,000. Using the MATLAB routine `c7_LCGPeriod.m` show that a period of 6,000 is actually achieved.

7.10 A mixed congruential generator has $a = c = 1$ and $m = 256$. Assuming that the seed is $x_0 = 0$, describe the generator output. Is this a full-period generator? Determine the scatterplot and test for independence using the Durbin-Watson algorithm.

7.11 Show that the three component generators of the Wichmann-Hill algorithm have periods of 30268, 30306 and 330322. You may use the MATLAB routine `c7_LCGPeriod.m` to solve this problem.

7.12 Show, using both analysis and simulation, that the random number generator defined by

$$x_{i+1} = 5x_i \bmod(7)$$

is a full-period generator. Determine the generated sequence for seeds $x_0 = 1$ and $x_0 = 4$. Compare the sequences and comment on the results.

7.13 Show, using both analysis and simulation, that the LCG described by

$$x_{i+1} = 133x_i \bmod(256)$$

has a period of 64. Using an appropriate MATLAB program with the seed $x_0 = 1$, determine the generated sequence.

7.14 Run the MATLAB program given in Example 7.8 for $N = 100$, 1,000, and $20,000$. Compare the results to those illustrated in Figure 7.8. Comment on the results.

7.15 A Weibull random variable is defined by the pdf

$$f_X(x) = ax^{a-1}\exp(-x^a)u(x)$$

where a is a parameter and $u(x)$ denotes the unit step. Using the inverse transform method, develop an algorithm for generating a sequence of random numbers having a Weibull distribution. Using the algorithm just generated with $a = 5$, generate 500 samples of a Weibull random variable. Plot the histogram and compare with the pdf.

7.16 A Laplacian random variable is defined by the pdf

$$f_X(x) = \frac{a}{2}\exp(-a\,|x|), \qquad a > 0$$

Note that this pdf is double-sided. Using the inverse transform method, determine an algorithm for generating X from a uniforrmly distributed random variable. Let $a = 3$ and generate 1,000 samples of X. Determine the resulting histogram and compare the result with the theoretical pdf.

7.17 (a) Generate $N = 1000$ pairs of zero-mean Gaussian random variables using the Box-Muller algorithm. The desired variance is $\sigma^2 = 5$. From the set of generated samples determine m_X, m_Z, σ_X^2, σ_Z^2, and ρ_{XZ}. Comment on the results.

(b) Repeat part (a) but generate *approximately* $N = 1,000$ random variables using the polar method.

7.18 Both the Box-Muller and the polar techniques for generating a pair of random numbers require that a pair of independent uniformly distributed random numbers be generated. This can be accomplished using the following two lines of code:

```
u1 = rand(1,N);
u2 = rand(1,N);
```

Will the code segment

```
u1 = rand(1,N);
u2 = [u1(1,N),u1(1,1:N-1)];
```

work just as well? Why or why not? Test both of the code segments to demonstrate the validity of your answer.

7.19 Rework Example 7.8 for the pdf

$$f_X(x) = \begin{cases} \frac{4}{2\pi R^2}\sqrt{R^2 - x^2}, & -R \leq x \leq R \\ 0, & \text{otherwise} \end{cases}$$

Note that this will require a simple modification to the algorithm used in Example 7.9.

7.20 The unit impulse response of the Jakes filter defined by (7.98) is given by

$$h(t) = K\left(\pi f_d t\right)^{-1/4}\Gamma\left(3/4\right) f_d J_{1/4}(2\pi f_d t)$$

where $J_{1/4}(\cdot)$ is the fractional Bessel function and $\Gamma(\cdot)$ denotes the gamma function. Let $f_d = 10$ Hz and plot $h(t)$. Compare with the sampled unit impulse response derived by taking the inverse FFT of $H(f)$ as determined in Example 7.11.

7.21 In this problem you are to extend the rejection method so that a random variable having infinite support can be generated. The target pdf describes the generalized Gaussian random variable, which is defined by

$$f_X(x) = \frac{v}{\sqrt{8}\sigma\Gamma(1/v)}\exp\left(-\left|\frac{x-m}{\sqrt{2}\sigma}\right|^v\right)$$

where m is the mean, v is the mode σ is a parameter, and $\Gamma(\cdot)$ denotes the gamma function. Note that for $v = 2$, $f_X(x)$ is Gaussian.

(a) Plot the generalized Gaussian pdf for $m = 0$ and $v = 1$, 1.5, 2, and 2.5.

(b) Develop, using a rejection method, an algorithm for generating X with $v = 1.5$ and $m = 0$. Select a bounding function and plot both the bounding function and $f_X(x)$.

(c) Using the algorithm with $v = 1.5$ and $m = 0$, generate 10,000 values of X. Plot the resulting histogram and compare with the theoretical pdf.

7.22 Using the algorithm defined by (7.87), generate $N = 50$ pairs of zero-mean Gaussian random variables with $\sigma^2 = 3$ and $\rho = 0.6$. From the set of generated samples, determine m_X, m_Z, σ_X^2, σ_Z^2, and ρ_{XZ}. Repeat for $N = 5{,}000$. Comment on the results.

7.23 Develop an algorithm for generating a set of uncorrelated samples having the PSD

$$S(f) = [1 - \cos(\pi f / f_s)]^2$$

where f_s is the sampling frequency. Generate 10,000 samples having the target PSD and verify the algorithm. Using the generated data, determine the autocorrelation function and compare with the target autocorrelation function.

7.13 Appendix A: MATLAB Code for Example 7.11

7.14 Main Program: c7_Jakes.m

```
% File c7_Jakes.m
% Generate and test the impulse response of the filter.
%
fd = 100;                                % maximum doppler
impw = jakes_filter(fd);                 % call to Jakes filter
fs = 16*fd; ts = 1/fs;                   % sampling frequency and time
time = [1*ts:ts:128*ts];                 % time vector for plot
subplot(2,1,1)
stem(time,impw,'.'); grid;
xlabel('Time'); ylabel('Impulse Response')
%
% Square the fft and check the power transfer function.
%
[h f] = linear_fft(impw,128,ts);         % generate H(f) for filter
subplot(2,1,2)
plot(f,abs(h.*h)); grid;
xlabel('Frequency'); ylabel('PSD')
%
% Put Gaussian noise through and check the output psd.
%
x = randn(1,1024);                       % generate Gaussian input
y = filter(impw,1,x);                    % filter Gaussian input
[output_psd ff] = log_psd(y,1024,ts);    % log of PSD figure;
subplot(2,1,1)
plot(ff,output_psd); grid;
axis([-500 500 -50 0])
xlabel('Frequency'); ylabel('PSD')
%
% Filter complex noise and look at the envelope fading.
%
z = randn(1,1024)+i*randn(1,1024);       % generate complex noise
zz = filter(impw,1,z);                   % filter complex noise
time = (0.0:ts:1024*ts);                 % new time axis
%
% Normalize output and plot envelope.
%
zz = zz/max(max(abs(zz)));               % normalize to one
subplot(2,1,2)
plot(time(161:480),10*log10(abs(zz(161:480)))); grid;
axis([0.1 0.3 -20 0])
xlabel('Time'); ylabel('Log Amplitude')
% End of function file.
```

7.14.1 Supporting Routines

Jakes_filter.m

```
% File: Jakes_filter.m
function [impw] = jakes_filter(fd)
% FIR implementation of the Jakes filter (128 points)
n = 512; nn = 2*n;                    % nn is FFT block size
fs = 0:fd/64:fd;                      % sampling frequency = 16*fd
H = zeros(1,n);                       % initialize H(f)
for k=1:(n/8+1)                       % psd for k=1:65
   jpsd(k)=1/((1-((fs(k))/fd)^2)^0.5);
   if(jpsd(k))>1000
      jpsd(k)=1000;
   end
   H(k)=jpsd(k)^0.5;                  % first 65 points of H
end
for k=1:n                             % generate negative frequencies
   H(n+k) = H(n+1-k);
end
[inv,time] = linear_fft(H,nn,fd/64); % inverse FFT
imp = real(inv(450:577));             % middle 128 points
impw = imp.*hanning(128)';            % apply hanning window
energy = sum(impw.^2);                % compute energy
impw = impw/(energy^0.5);             % normalize
% End of function file.
```

linear_fft.m

```
% File: linear_fft.m
function [fftx,freq] = linear_fft(x,n,ts)
% This function takes n (must be even) time domain samples (real or
% complex) and finds the PSD by taking (fft/n)^2. The two sided
% spectrum is produced by shifting the PSD. The array freq provides
% the appropriate frequency values for plotting purposes.
y = zeros(1,n);
for k=1:n
   freq(k) =(k-1-(n/2))/(n*ts);
   y(k) = x(k)*((-1.0)^(k+1));
end;
fftx = fft(y)/n;
% End of function file.
```

log_psd.m

```
% File: log_psd.m
function [logpsd,freq,ptotal,pmax] = log_psd(x,n,ts)
```

```
% This function takes the n time domain samples (real or complex)
% and finds the psd by taking (fft/n)^2. The two sided spectrum is
% produced by shifting the psd; The array freq provides the
% appropriate frequency values for plotting purposes.
% By taking 10*log10(psd/max(psd)) the psd is normalized; values
% below -60 dB are set equal to -60dB.
%
% n must be an even number, preferably a power of 2
%
y = zeros(1,n);                                % initialize y vector
%
h = waitbar(0,'For Loop in PSD Calculation');
for k=1:n
 freq(k) =(k-1-(n/2))/(n*ts);
 y(k) = x(k)*((-1.0)^k);
 waitbar(k/n)
end;
%
v = fft(y)/n;
psd = abs(v).^2;
pmax=max(psd);
ptotal=sum(psd);
logpsd = 10*log10(psd/pmax);
%
% Truncate negative values at -60 dB
%
for k =1:n
 if(logpsd(k)<-60.0)
 logpsd (k) =-60.0;
 end
end
close(h)
% End of function file.
```

Chapter 8

POSTPROCESSING

In this chapter we briefly introduce the important topic of postprocessing. As discussed previously, the role of the postprocessor is to manipulate the data created by the simulation into a useful form. Postprocessors are usually graphics intensive, since visual displays are more easily interpreted than numerical listings, which are the most common data output from a simulation program. For example, a plot of the bit error rate for several different systems conveys information more quickly than a numerical table containing the same information.

Postprocessing routines may, or may not, involve significant computational complexity. Some postprocessors simply take data created by a simulation and, after properly formatting the data, generate the appropriate graphical output. An example is a routine for generating a plot of bit or symbol error probability, P_E, as a function of E_b/N_0. The values of P_E, along with the accompanying values of E_b/N_0, are created by the simulation and passed to the postprocessor as data files. The postprocessor simply formats the data and creates the required plot. Other examples of postprocessors that generate graphical output with minimal processing are those for displaying signal waveforms, eye diagrams, and scatter plots.

On the other hand, some postprocessing routines involve significant data processing. Most of these involve some type of estimation. A simple example is the generation of a histogram, which is an estimator of a probability density function. More complex examples are estimators for time delay, signal-to-noise ratio (SNR), and power spectral density. Other examples considered in this chapter involve the

mapping of the channel symbol error rate to a decoded bit error rate for a system utilizing error-control coding. The list of possible postprocessing operations is virtually endless and, in this chapter, we only explore a few examples. All of these operations are considered postprocessing, since they make use of the data created by a simulation and are implemented after the simulation engine has completed its work. (Recall the discussion of the simulation engine in Chapter 6.)

8.1 Basic Graphical Techniques

In order to illustrate the graphical techniques used in a typical simulation postprocessor, the concepts are considered within the context of an example system. The choice for an example system is, of course, arbitrary. We consider here $\pi/4$ DQPSK, since it has a number of interesting characteristics and is used in a number of wireless systems [1].[1]

8.1.1 A System Example—$\pi/4$ DQPSK Transmission

A block diagram of a $\pi/4$ DQPSK transmitter is illustrated in Figure 8.1. The output of the data source is assumed to be a sequence, a, of the form

$$a(1)a(2)a(3)a(4)\cdots a(k)\cdots$$

The parallel-to-serial converter assigns alternate (odd-indexed) symbols to the direct channel and the remaining (even-indexed) symbols to the quadrature channel. Thus:

$$a(1)a(2)a(3)a(4)\cdots a(k)\cdots = d(1)q(1)d(2)q(2)\cdots d\left(\frac{k+1}{2}\right)q\left(\frac{k}{2}+1\right)\cdots$$

The transmitted signal is given by

$$x_c(t) = A\cos\left[2\pi f_c t + \theta(k)\right], \qquad (k-1)T_s < t < kT_s \tag{8.1}$$

where T_s is the symbol period. The phase deviation of the transmitted signal is determined by the values of $d(k)$ and $q(k)$ as well as the phase deviation $\theta(k-1)$, which is the phase deviation during the previous symbol period. This dependence of the previous symbol period is, of course, what makes $\pi/4$ DQPSK a differential modulation technique. The relationship between $\theta(k)$ and $\theta(k-1)$ is

$$\theta(k) = \theta(k-1) + \phi(k) \tag{8.2}$$

where $\phi(k)$ is an explicit function of $d(k)$ and $q(k)$, and is defined in Table 8.1. The required transmitted phases are generated in the phase mapper shown in Figure 8.1. The phase mapper uses $d(k)$, $q(k)$, and $\theta(k-1)$ to generate the new values $d'(k)$ and

[1] $\pi/4$ DQPSK has been adopted as the modulation format in a number of system standards. These include the USDC (United States Digital Cellular) system, the PACS (Personal Access Communication System) PCS system, the PDC (Pacific Digital Cellular) system, and the PHS (Personal Handy Phone) cordless system. [1]

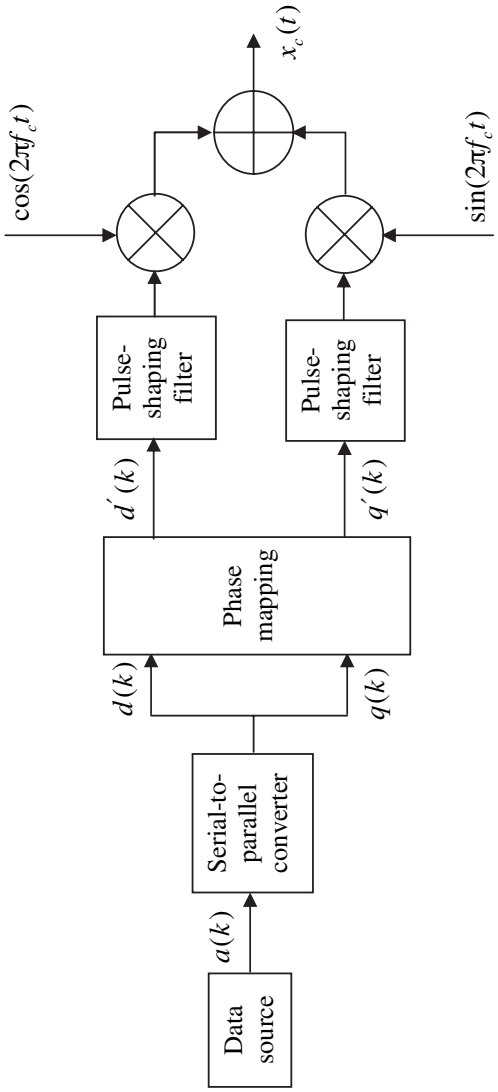


Figure 8.1 $\pi/4$ DQPSK transmitter.

Table 8.1 Differential Phase Shifts for $\pi/4$ DQPSK

Information Symbols, $d(k)$ and $q(k)$	Differential Phase Shift, $\phi(k)$
1 1	$\pi/4$
0 1	$3\pi/4$
0 0	$-3\pi/4$
1 0	$-\pi/4$

$q'(k)$, so that the transmitted signal has the proper phase. After appropriate pulse shaping, the direct and quadrature channel signals are translated to the transmission frequency, f_c, as shown.

As an example, assume that the output of the data source is the binary sequence

$$0010110111\cdots \tag{8.3}$$

and that the initial phase is defined by $\theta(0) = 0$. Since the first two data symbols are 00, it follows from Table 8.1 that $\phi(1) = -3\pi/4$. From (8.2) we then have

$$\theta(1) = 0 - \frac{3\pi}{4} = -\frac{3\pi}{4}$$

The next two data symbols are 10, so that $\phi(2) = -\pi/4$. Thus:

$$\theta(2) = -\frac{3\pi}{4} - \frac{\pi}{4} = -\pi$$

The next two data symbols are 11. Thus, $\phi(3) = \pi/4$, which gives

$$\theta(3) = -\pi + \frac{\pi}{4} = -\frac{3\pi}{4}$$

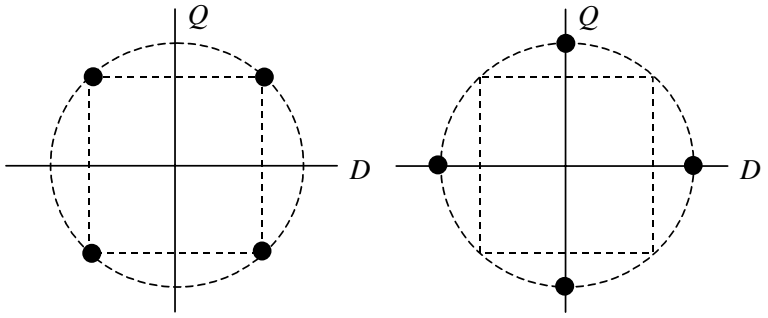
In like manner, $\phi(4) = 3\pi/4$, so that

$$\theta(4) = -\frac{3\pi}{4} + \frac{3\pi}{4} = 0$$

and $\phi(5) = \pi/4$, which gives

$$\theta(5) = 0 + \frac{\pi}{4} = \frac{\pi}{4}$$

A quick observation illustrates that $\theta(1)$, $\theta(3)$, and $\theta(5)$ are phases from the first QPSK signal constellation illustrated in Figure 8.2(a), and that $\theta(2)$ and $\theta(4)$ are phases from the second QPSK signal constellation illustrated in Figure 8.2(b). Thus, $\pi/4$ DQPSK operates by transmitting signal points from alternating QPSK signal constellations where the two QPSK signal constellations are displaced by a $\pi/4$ phase rotation. Although this has been demonstrated using a specific data sequence, we see that the result is general, since the differential phases only take on the values $\pm\pi/4$ or $\pm 3\pi/4$.



(a) QPSK: Phase set 1 (b) QPSK: Phase set 2

Figure 8.2 Signal constellations for $\pi/4$ DQPSK.

8.1.2 Waveforms, Eye Diagrams, and Scatter Plots

Prior to demonstrating the basic programs for plotting waveforms, eye diagrams, and scatter plots, we first pause to illustrate the relationship between these plots. Suppose that we develop a three-dimensional coordinate system as shown in Figure 8.3 with the axes labeled as shown. Note that three intersecting planes can be formed, each of which contain two of the axes. These planes are formed by the D and t axes, the Q and t axes, and the Q and D axes. If the direct-channel signal $x_d(t)$ is plotted on the (D, t) plane and the quadrature channel signal $x_q(t)$ is plotted on the (Q, t) plane, a three-dimensional signal, parameterized on t, is

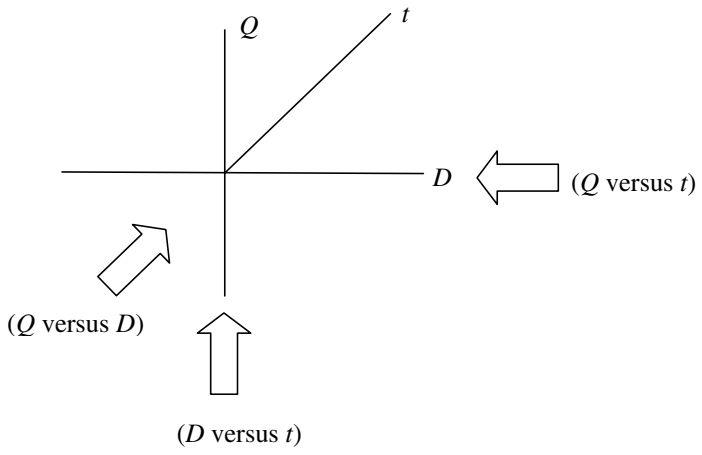


Figure 8.3 Three-dimensional coordinate system.

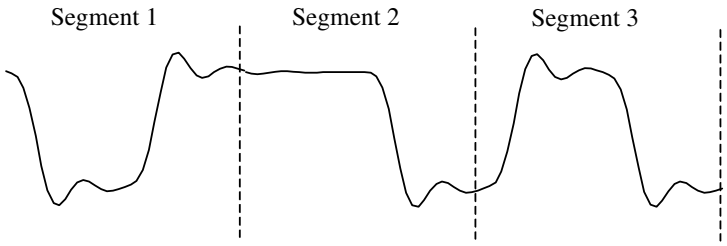
generated. Projecting this signal onto a given subspace (D,t), (Q,t), or (D,Q), generates $x_d(t)$, $x_q(t)$, or the scatter plot, which is a plot of $x_q(t)$ as a function of $x_d(t)$. This is illustrated in Figure 8.3. Looking in from the right so that the (D,t) plane is seen edge-on shows the quadrature signal $x_q(t)$. In the same manner, the direct channel signal, $x_d(t)$, is obtained by viewing the three-dimensional image from below so that the (Q,t) plane is seen edge-on. Looking down the time axis so that the time axis becomes a point reveals the scatter plot.

While the three-dimensional (Q,D,t) image is seldom generated in practice, visualizing the (Q,D,t) image is a good learning tool and shows clearly the relationship between $x_d(t)$, $x_q(t)$, and the scatter plot. An interesting tutorial was based on this concept [2].

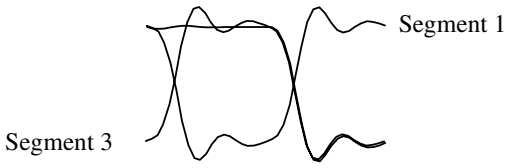
Eye Diagrams

The eye diagram gives a qualitative measure of system performance [3]. A well-defined and open eye usually indicates good performance, while a poorly defined eye usually indicates poor performance. In addition, the size of the eye relates to the accuracy required of the symbol synchronizer. While the eye diagram does not provide a quantitative measure of system performance, it is difficult to conceive of a high-performance system having a poorly defined eye diagram.

The generation of an eye diagram is illustrated in Figure 8.4. Three segments of a waveform, with each segment corresponding to a symbol period, are shown



(a) Three segments (60 samples) of a waveform



(b) Three-segment eye diagram

Figure 8.4 Generation of an eye diagram.

in Figure 8.4. The waveform corresponding to three data symbols is illustrated in Figure 8.4(a). Assume that this waveform is displayed on an oscilloscope and that the oscilloscope is triggered at the points denoted by the dotted vertical lines. The result will be the three-segment eye diagram illustrated in Figure 8.4(b).

Example 8.1. In this example, several important signals present in a $\pi/2$ DPSK system are generated and displayed. The MATLAB program for simulating the system and generating the graphical output is given in Appendix A. Upon entering the program name, `c8_pi4demo`, at the MATLAB prompt, a menu is presented. From this menu the user may select one of the following seven options (after a plot is generated, hitting the space bar will display the menu so that another selection can be made):

1. Unfiltered $\pi/4$ DQPSK signal constellation
2. Unfiltered $\pi 4$ DQPSK eye diagram
3. Filtered $\pi/4$ DQPSK signal constellation
4. Filtered $\pi 4$ DQPSK eye diagram
5. Unfiltered direct and quadrature signals
6. Filtered direct and quadrature signals
7. Exit program (return MATLAB prompt)

The student should study the material in Appendix A closely, as it illustrates many of the common postprocessing procedures. In addition, the code used for generating the various plots can be used in the postprocessor of other simulation programs. Here we illustrate three of the more interesting results. Figures 8.5, 8.6, and 8.7 illustrate the scatter plot (signal constellation), the direct and quadrature channel signals, and the direct and quadrature channel eye diagrams, respectively. [Note that by visualizing the three-dimensional signal in the (D, Q, t) space as previously discussed, the relationship between Figures 8.5 and 8.7 is easily seen.] ■

8.2 Estimation

Many useful estimation routines are based on data generated by a simulation program. Here we consider only a small sampling of the many possibilities.

8.2.1 Histograms

When a set of samples of a random process is available, as will be the case in a simulation environment, a histogram formed from that set of samples is frequently used as an estimator of the underlying probability density function (pdf). The histogram is formed by grouping data, consisting of N total samples, into B bins

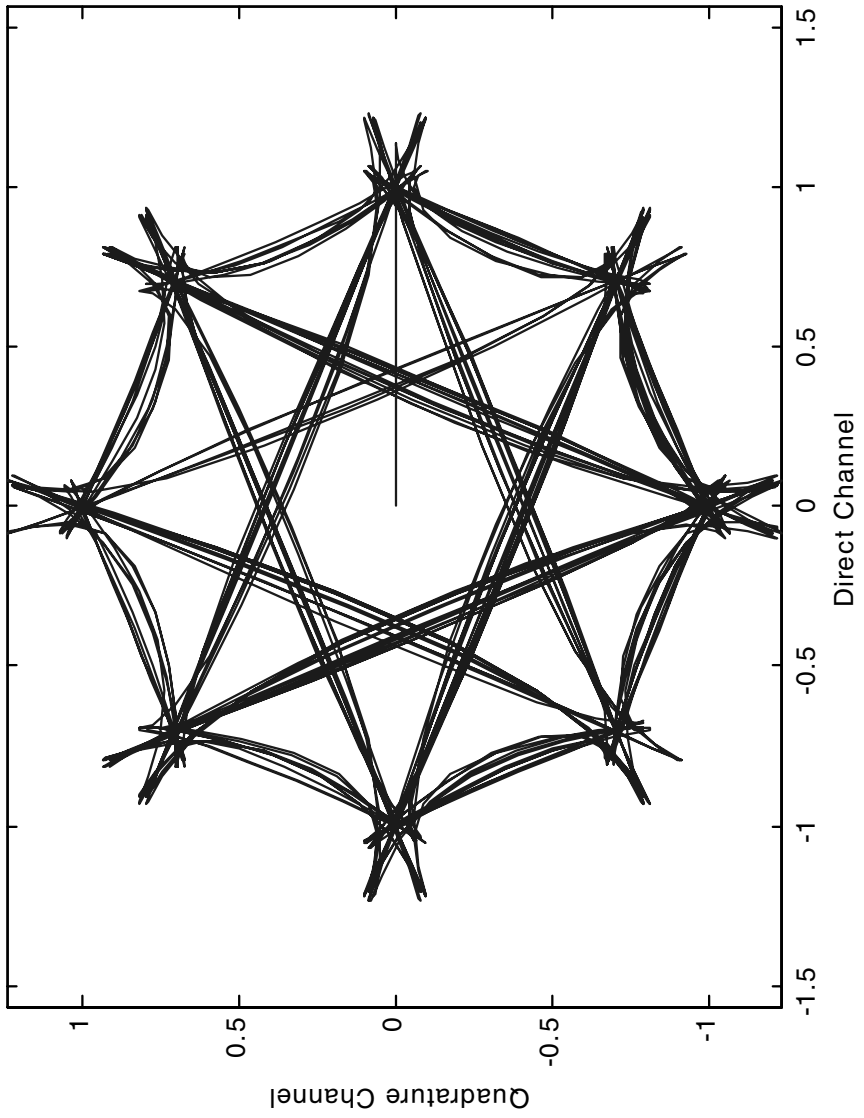


Figure 8.5 Filtered signal constellation.

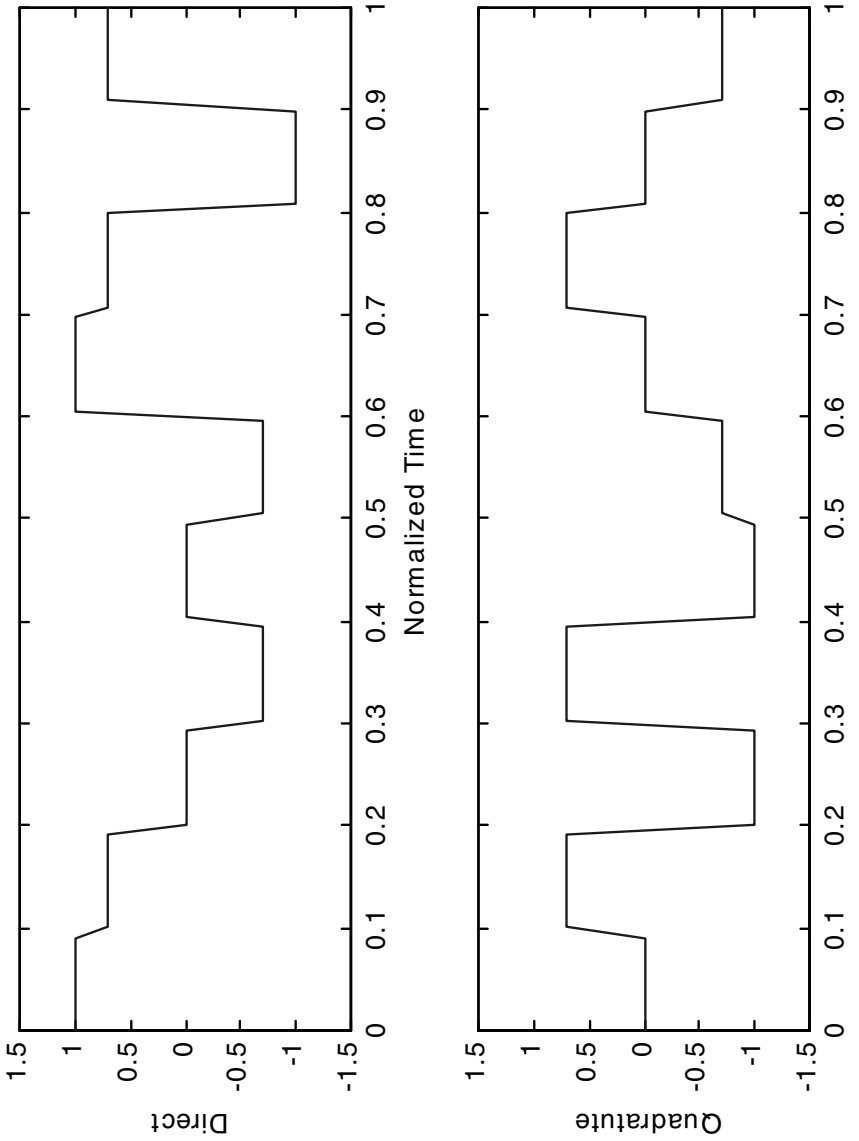


Figure 8.6 Unfiltered direct and quadrature signals.

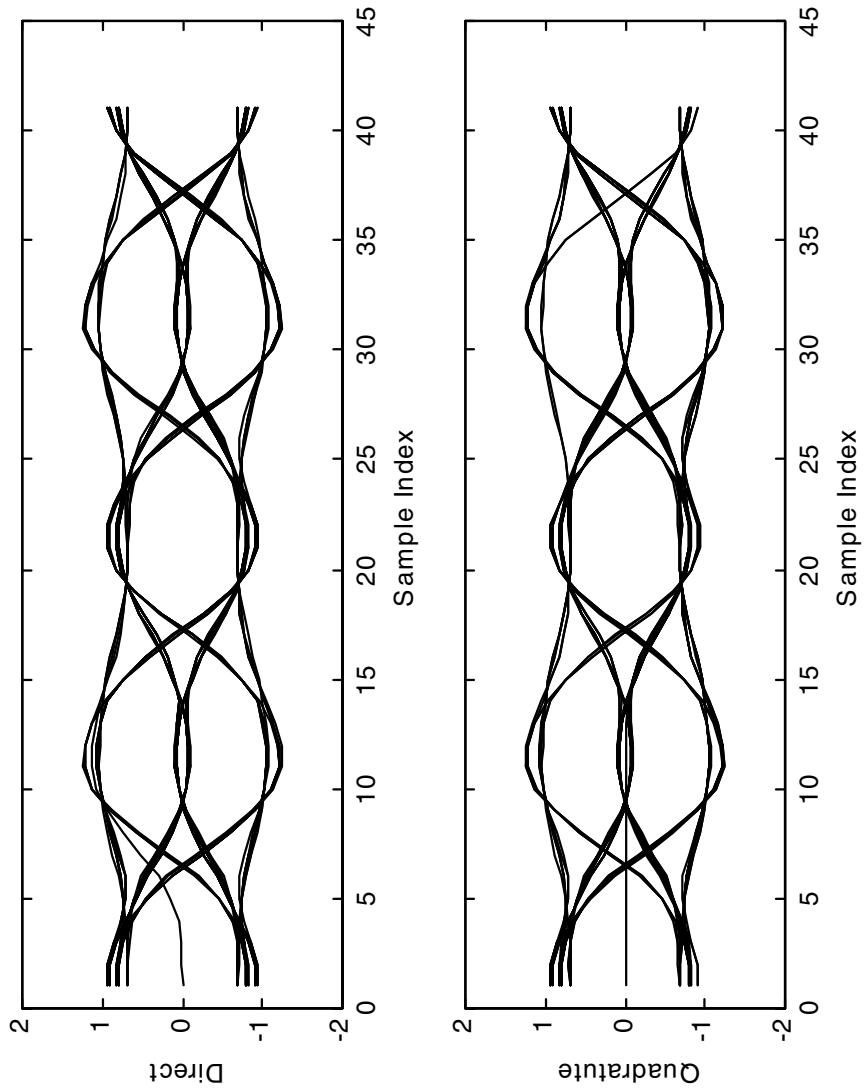


Figure 8.7 Filtered eye diagram.

or cells. Each bin is assumed to have equal width W, and the center of each bin is denoted b_i. A given sample $x[n]$ falls into the i^{th} bin if

$$b_i - \frac{W}{2} < x[n] \leq b_i + \frac{W}{2} \tag{8.4}$$

The quantity of interest is N_i, which denotes the number of samples falling into the i^{th} bin. Clearly

$$N = \sum_{i=1}^{B} N_i \tag{8.5}$$

We adopt the notation Count$\{N : R\}$ to represent the number of samples, in the set of N total samples, falling into the histogram bin defined by R. Thus

$$N_i = \text{Count}\left\{N : b_i - \frac{W}{2} < x[n] \leq b_i + \frac{W}{2}\right\} \tag{8.6}$$

A bar graph is then plotted in which the height of each bar is proportional to N_i, and each bar is centered at b_i. In order to be an estimator of the pdf, the histogram is scaled so that the total area is one. This is accomplished by dividing N_i by NW. The height of each bar is then N_i/NW. The area of the bar, A_i, representing the i^{th} histogram bin, is found by multiplying the height by the width W. Thus

$$A_i = \left(\frac{N_i}{NW}\right) W = \frac{N_i}{N} \tag{8.7}$$

Note that A_i represents the relative frequency of the i^{th} histogram bin. The total area is

$$\sum_{i=1}^{B} \frac{N_i}{N} = 1 \tag{8.8}$$

as required if the histogram is to represent a probability density function.

Note that each histogram bin represents the pdf over a finite span of width W by a constant. For some point within a given bin, the estimator of the pdf will be unbiased. However, over most of the range defined by a given histogram bin, the estimator will be biased. It is easily shown, by expanding the pdf $f_X(x)$ in a Taylor series about the center of the i^{th} bin, that the bias is [4]

$$E\left\{f_X(b_i) - \widehat{f}_X(b_i)\right\} \approx \left.\frac{d^2 f_X(x)}{dx^2}\right|_{x=b_i} \frac{W^2}{24} \tag{8.9}$$

Thus, the bias can be reduced only by decreasing the bin width, W.

It can also be shown that the variance of the estimator is [4]

$$\text{var}\left\{\widehat{f}_X(b_i)\right\} \approx \frac{1}{NW} f_X(b_i)\,[1 - W f_X(b_i)] \tag{8.10}$$

Note that $Wf_X(b_i)$ is the probability of the event that a given sample falls into bin b_i. Since it is a probability, $Wf_X(b_i) \leq 1$ is less than one and is usually much less than one. Thus

$$\text{var}\left\{\widehat{f}_X(b_i)\right\} \approx \frac{1}{NW} f_X(b_i) \tag{8.11}$$

We see that, for fixed N, increasing W decreases the variance of the estimator. Unfortunately, from (8.9), we see that the effect of increasing W increases the bias. We therefore desire small W and large NW. Thus, N and W must be related. For example, if $W = 1/\sqrt{N}$ and $NW = \sqrt{N}$, $W \to 0$ as $N \to \infty$ and also $NW \to \infty$ as $N \to \infty$. In this case, for sufficiently large N, the result will be a pdf with negligible bias and negligible variance. The following example illustrates the histogram for various choices of N and W.

Example 8.2. Assume that we generate N samples of a zero-mean unit-variance Gaussian random variable. The pdf is estimated by constructing a histogram with B bins. In this example, we wish to examine the impact of varying both N and B. In order to accomplish this, the following MATLAB problem is used:

```
% File c9_hist.m
subplot(2,2,1)
x = randn(1,100); hist(x,20)
ylabel('N_i'); xlabel('(a)')
subplot(2,2,2)
x = randn(1,100); hist(x,5)
ylabel('N_i'); xlabel('(b)')
subplot(2,2,3)
x = randn(1,1000); hist(x,50)
ylabel('N_i'); xlabel('(c)')
subplot(2,2,4)
x = randn(1,100000); hist(x,50)
ylabel('N_i'); xlabel('(d)')
% End of script file.
```

Executing this program gives the results illustrated in Figure 8.2. Figure 8.2(a) illustrates the results for $N = 100$ and $B = 20$. As we can see, the histogram is not well defined, since $N/B = 5$, which is much too small to yield a reliable estimator for the number of samples falling into a given histogram bin. Figure 8.2(b) illustrates the result for $N = 100$ and $B = 5$. While the ratio for N/B is now much larger, the number of bins is too small. Note that with N fixed, the histogram with $B = 5$ will exhibit greater bias than $B = 20$. Figure 8.2(c) illustrates the result for $N = 1{,}000$ and $B = 50$. While this provides a better estimator, the ratio N/B is once again too small and we therefore increase N. Figure 8.2(d) illustrates the result for $N = 100{,}000$ and $B = 50$. This yields $N/B = 2{,}000$ and the resulting histogram has the predicted Gaussian shape. If the underlying pdf has a complicated shape, a very large number of samples is often required if the histogram is to be an accurate estimator of the pdf. ■

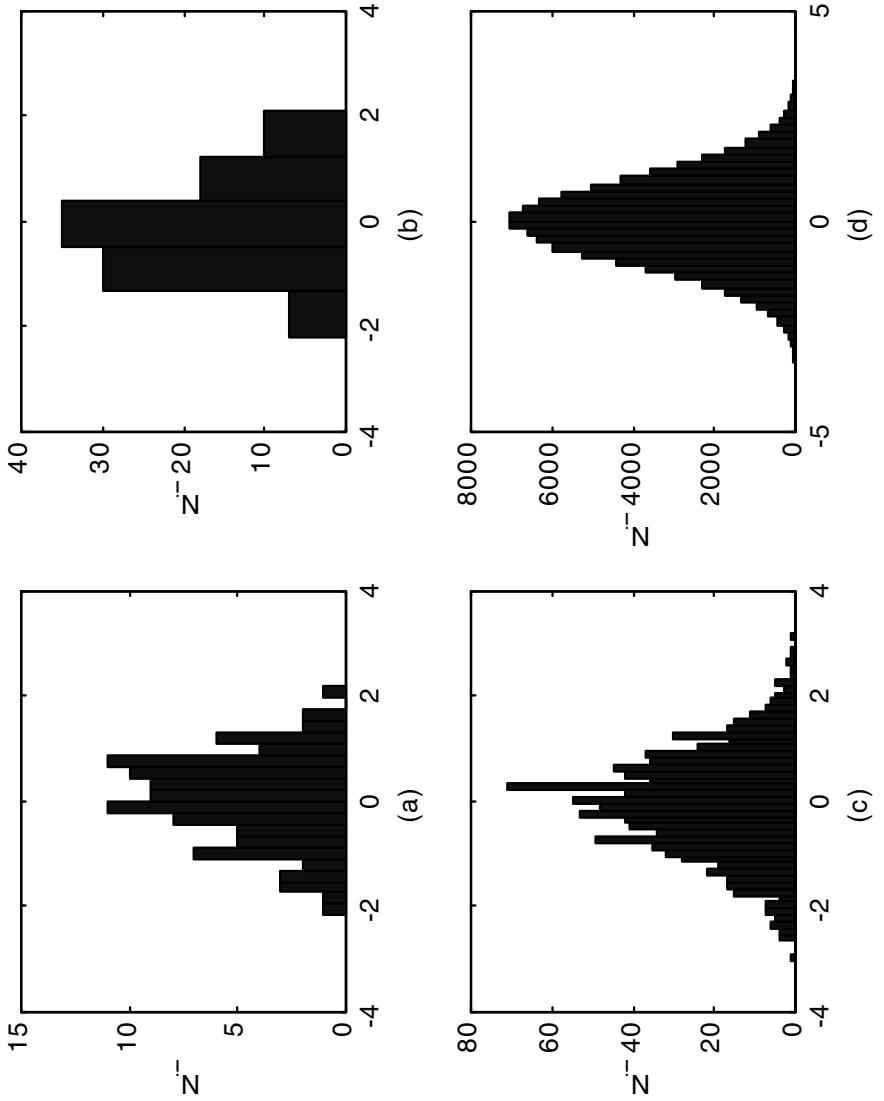


Figure 8.8 Histograms for a Gaussian random variable (a) $N = 100$, $B = 20$; (b) $N = 100$, $B = 5$; (c) $N = 1{,}000$, $B = 50$; (d) $N = 100{,}000$, $B = 50$.

8.2.2 Power Spectral Density Estimation

Another postprocessing operation that is frequently used in a simulation study is the estimation of the power spectral density (PSD) of a signal at a point in a system. This is a more difficult task than we might generally assume. Typically the waveform of interest is a sample function of a stochastic process. This leads to a situation in which the PSD at a given value of frequency f_1 is a random variable. It then becomes, as with most problems in estimation theory, necessary to minimize the variance of the spectral estimate $\widehat{S}(f_1)$. A number of books [5–9] and many papers have been written on this topic and many techniques have been developed for PSD estimation. In this section we consider only the most fundamental techniques. The techniques considered here are based on the fast Fourier transform (FFT) and are frequently used in the simulation context.

The Periodogram

The simplest, fastest, and most frequently used PSD estimation algorithm is the periodogram. It is defined by

$$\widehat{S}(kf_\Delta) = \frac{1}{N}\left|\sum_{n=0}^{N-1} x[n]\exp(-j2\pi kf_\Delta n)\right|^2 = \frac{1}{N}\left|I_N(kf_\Delta)\right|^2, \quad k = 0, 1, \cdots, N-1 \quad (8.12)$$

in which N is the total number of samples in the data record, and $I_N(kf_\Delta)$ is the N-point FFT of the data for which the PSD estimate at frequency $f = kf_\Delta$ is to be computed. The computational efficiency of the periodogram comes from the use of the FFT to form the PSD estimate. The result provides us with N frequency domain estimates having a resolution of $f_\Delta = f_s/N$, where f_s is the sampling frequency associated with the time-domain points for which the spectral estimate is being performed. Note that f_s in this context is not always the sampling frequency associated with the simulation. For example, if a given signal in a simulation is significantly oversampled, the samples may be decimated[2] prior to forming the PSD estimate.

The difficulty with the periodogram is that it is biased and is not a consistent estimator of the PSD at a frequency f. For many applications, the variance of the periodogram is unacceptably high. The bias results from the unavoidable fact that the data record is finite. However, for sufficiently large N, the bias can be neglected. Therefore, the main difficulty results from the high variance. Assuming that the data samples $x[n]$ are independent, the variance of the spectral estimate at frequency f is [6]

$$\text{var}\left(\widehat{S}(f)\right) = \sigma_x^4\left\{1 + \left(\frac{\sin[2\pi fN]}{N\sin[2\pi f]}\right)^2\right\} \quad (8.13)$$

[2]Recall the discussion of decimation in Chapter 3.

where σ_x^2 is the variance of the data samples $x[n]$. We observe that the variance of $\widehat{S}(f)$ does not tend to zero as $N \to \infty$ and, for large N, the variance of the spectral estimate is independent of frequency. The periodogram, however, despite this serious flaw, is useful for a "quick look" at the PSD.

The Periodogram With a Data Window

If a data window is not explicitly specified, the default rectangular window is used.[3] For a rectangular window each sample value $x[n]$ is multiplied by $w[n] = 1$ for $0 \leq n \leq N-1$. The impact of the rectangular window is to convolve the data samples $x[n]$, with the Fourier transform of $w[n]$, which has the amplitude spectrum $\sin(\pi N f)/N \sin(\pi f)$. The sidelobe structure of this data window, when viewed in the frequency domain, results in considerable spectral leakage [6]. This spectral leakage distorts and reduces the dynamic range of the estimated spectrum. (See Problem 8.7.)

When an arbitrary data window is used, $\widehat{S}(kf_\Delta)$ takes the form

$$\widehat{S}(kf_\Delta) = \frac{1}{U}\left|\sum_{n=0}^{N-1} x[n]w[n]\exp(-j2\pi k f_\Delta n)\right|^2 \tag{8.14}$$

where U is the energy in the data window, which is given by

$$U = \sum_{n=0}^{N-1} w^2[n] \tag{8.15}$$

Note that for the rectangular window, for which $w[n] = 1$ for all n, $U = N$ and (8.12) results. The choice of data window represents a number of tradeoffs. The ideal data window must have finite duration in the time domain so that $I_N(kf_\Delta)$, the Fourier transform of the data, can be accurately estimated using a finite data record. In addition, the estimated Fourier transform of the data record must not be adversely affected by the window function. Since multiplication in the time domain is convolution in the frequency domain, and only convolution with an impulse function leaves the transform unchanged, the ideal window function is an impulse in the frequency domain. Since the Fourier transform of an impulse is not of finite extent, these are conflicting requirements. We therefore seek a data window that, in the frequency domain, exhibits a narrow main lobe about $f = 0$, and sidelobes that are greatly attenuated. A variety of window functions are discussed in the classic paper by Harris [10].

Segmented Periodograms

A common technique for reducing the variance associated with the periodogram is to divide the N-sample data record into K segments, with each segment consisting of M samples. The FFT is computed for each segment and the results are averaged.

[3] In MATLAB the rectangular window is called the BOXCAR window.

The averaging process reduces the variance of the spectral estimate. The segments may or may not be overlapping. If the segments do not overlap, $K = M/N$; otherwise $K > M/N$.

The periodogram of the i^{th} data segment is given by

$$I_M^{(i)}(kf_\Delta) = \frac{1}{U}\left|\sum_{n=0}^{M-1} x^{(i)}[n]w[n]\exp(-j2\pi kf_\Delta n)\right|^2 \qquad i = 1, 2, \cdots, K \tag{8.16}$$

where $x^{(i)}[n]$ represents the samples in the i^{th} data record and $f_\Delta = f_s/M$. The K periodograms are then averaged to produce the PSD estimator

$$\widehat{S}(kf_\Delta) = \frac{1}{K}\sum_{i=1}^{K} I_M^{(i)}(kf_\Delta), \qquad k = 0, 1, \cdots M-1 \tag{8.17}$$

This estimator is biased, of course, since the data record is finite. Assuming that the K periodograms are independent

$$\text{var}\left(\widehat{S}(kf_\Delta)\right) = \frac{1}{K}S^2(kf_\Delta) \tag{8.18}$$

which tends to zero as $K \to \infty$.

Comparing (8.12) and (8.16) reveals an obvious problem. The periodogram defined by (8.12) has a frequency resolution of $f_\Delta = f_s/N$, while the periodogram defined by (8.16) has a frequency resolution of $f_\Delta = f_s/M$. Since $M < N$ for $K > 1$, the frequency resolution is degraded by segmenting the original N-sample data record. Thus, using the segmentation technique gives rise to a tradeoff between resolution and variance. Also, the validity of (8.18) requires that the K periodograms used in the averaging process be independent. Since we desire the largest possible value of K for a fixed N, the segments are often overlapped. A 50 percent overlap is often used. When using a 50 percent overlap all samples $x[n]$ are used twice except for the $M/2$ samples at each end of the N sample data record, and the value of K is increased from N/M to $2(N/M) - 1$. If data segments are overlapped, however, the K periodograms are no longer independent and the reduction in the variance of the PSD estimator is less than that predicted by (8.18). The use of a data window, at least partially, helps to restore the independence of the K segments.[4]

While there are many data windows that can be used in (8.16), the Hanning window is frequently used for PSD estimation. The Hanning window is defined by

$$w[n] = \begin{cases} \frac{1}{2}\left[1 - \cos\left(\frac{2\pi n}{M-1}\right)\right], & 0 \le n \le M-1 \\ 0, & \text{otherwise} \end{cases} \tag{8.19}$$

[4]One obviously wishes to select the overlap so that a minimum-variance spectral estimator results. Marple [8] states that, for a Gaussian process, the minimum variance is achieved with an overlap of 65 percent when using a Hanning window.

Example 8.3. In this example we pass independent (white noise) samples through a Chebyshev filter having 5 dB passband ripple. The problem is to estimate the PSD at the filter output. The MATLAB program for accomplishing this follows:

```
% File:  c8_PSDexample.m
settle = 100;                    % ignore transient
fs = 1000;                       % sampling frequency
N = 50000;                       % size of data record
f = (0:(N-1))*fs/N;              % frequency scale
[b,a] = cheby1(5,5,0.1);         % filter
NN = N+settle;                   % allow transient to die
in = randn(1,NN);                % random input
out = filter(b,a,in);            % filter output
out = out((settle+1):NN);        % strip off initial samples
window = hanning(N)';            % set window function
winout = out.*window;            % windowed filter output
fout = abs(fft(winout,N)).^2;    % transform and square mag
U = sum(window.*window);         % window energy
f1out = fout/U;                  % scale spectrum
psd1 = 10*log10(abs(f1out));     % log scale
subplot(2,1,1)
plot(f(1:5000),psd1(1:5000))
grid; axis([0 100 -70 10]);
xlabel('Frequency, Hz')
ylabel('PSD')
%
K = 25;                          % number of segments
M = N/K;                         % block size
fK = (0:(M-1))*fs/M;             % frequency scale
d = zeros(1,M);                  % initialize vector
psdk = zeros(1,M);               % initialize vector
window = hanning(M)';            % set window function
U = sum(window.*window);         % window energy
for k=1:K
    for j=1:M
        index = (k-1)*M+j;
        d(j) = out(index);
    end
    dwin = d.*window;
    psdk = (abs(fft(dwin,M)).^2)/U + psdk;
end
psd2 = 10*log10(psdk/K);
subplot(2,1,2)
plot(fK(1:250),psd2(1:250))
grid; axis([0 100 -70 10]);
xlabel('Frequency, Hz')
```

```
ylabel('PSD')
% End of script file.
```

Executing the program yields the result shown in Figure 8.9. Note that for the nonoverlapped case (top pane), the variance is large. Also note that the variance is independent of frequency. Using 25 segments (bottom pane) results in a much smaller variance at the cost of reduced frequency resolution. Finally, note that the 5 dB passband ripple of the Chebyshev filter is much more obvious with $K = 25$ than for $K = 1$. ■

The PSD estimator illustrated in Example 8.3 was developed using basic MATLAB commands. The MATLAB Signal Processing Toolbox contains a number of routines for PSD estimation. Two of these are `psd` and `pwelch`. The interested student should study these in some detail. Here we illustrate the Welch periodogram from the Signal Processing Toolbox.

Example 8.4. In this example, we estimate the PSD of a QPSK signal. Rectangular pulse shaping is assumed, and the direct and quadrature components of the QPSK signal are sampled at 16 samples per symbol. The MATLAB code follows:

```
% File:  c8_welchp.m
fs = 16;
x = random_binary(1024,fs)+i*random_binary(1024,fs);
for nwin=1:4
    nwindow = nwin*1024;
    [pxx,f] = pwelch(x,nwindow,fs);
    pxx = pxx/sum(sum(pxx));
    n2 = length(f)/2;
    pxxdB = 10*log10(pxx/pxx(1));
    ptheory = sin(pi*f+eps)./(pi*f+eps);
    ptheory = ptheory.*ptheory;
    ptheorydB = 10*log10(ptheory/ptheory(1));
    subplot(2,2,nwin)
    plot(f(1:n2),pxxdB(1:n2),f(1:n2),ptheorydB(1:n2))
    ylabel('PSD in dB')
    xx = ['window length = ',num2str(nwindow)];
    xlabel(xx)
    axis([0 8 -50, 10]); grid;
end
% End of script file.
```

Executing the preceding code yields the results illustrated in Figure 8.10. Note that $16 \times 1{,}024$ points are generated and that window sizes (`nwindow`) of 1,024, 2,048, 3,072, and 4,096 are used. As with the preceding example, larger values for `nwindow` yield less averaging and, as a result, the estimated PSD exhibits greater variance. Smaller values for `nwindow` yield reduced variance at the cost of reduced resolution. These trends can be seen in Figure 8.10. ■

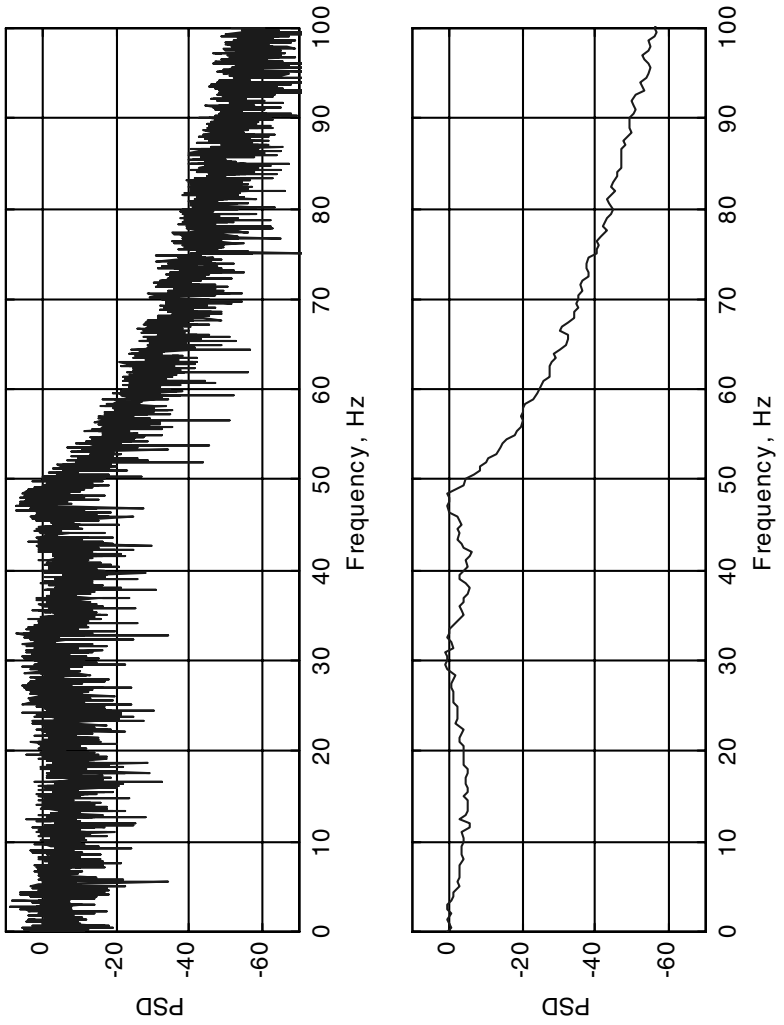


Figure 8.9 Power spectral density estimates, not averaged (top frame) and averaged (bottom frame).

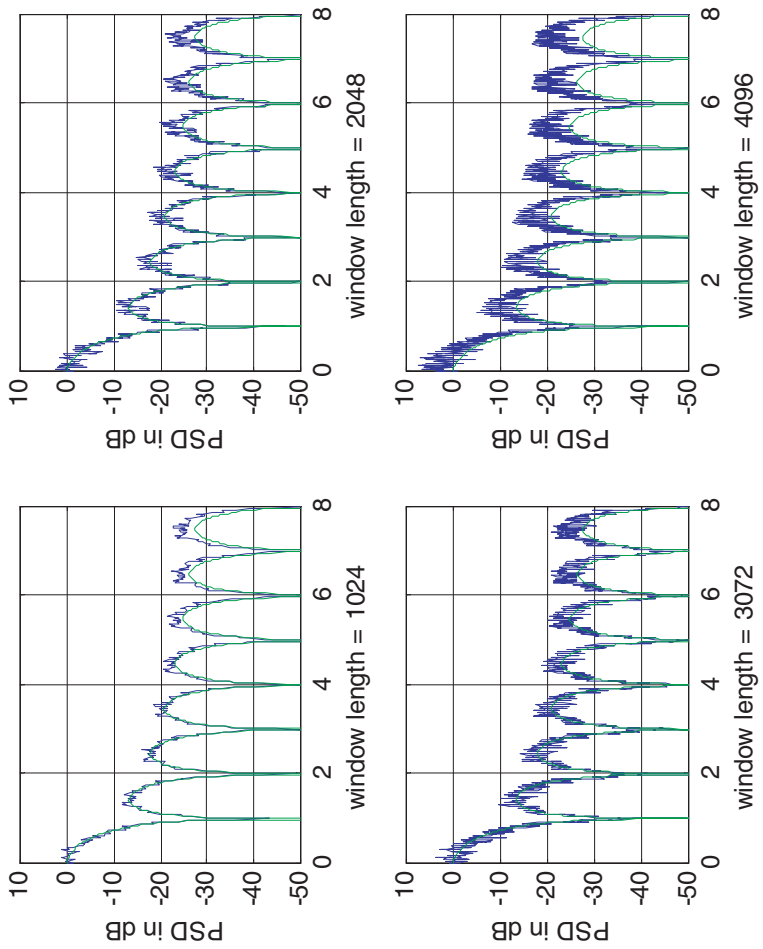


Figure 8.10 PSD Estimates generated in Example 8.4.

8.2.3 Gain, Delay, and Signal-to-Noise Ratios

The signal-to-noise ratio is a commonly used figure of merit for evaluating the performance of a communications system. The SNR estimation technique presented here originated from a method for measuring channel distortion errors in wideband telementy systems [11], in which the noise in a signal at a point in a system is defined as the mean-square error (MSE) between the actual signal and a desired signal at that point. The SNR can be estimated by defining the desired signal as an amplitude-scaled and time-delayed version of the information-bearing signal at the system input. In the past, applications of this technique have included monitoring reliable transmission of digital pulse code modulated data [12] and estimation of the carrier-to-intermodulation ratio in a nonlinear channel [13].

Theoretical Development for Real Lowpass Signals

For a linear time-invariant distortionless system, the signal $y(t)$ at any point in the system is an amplitude-scaled and time-delayed version of the input reference signal $x(t)$. Therefore, we can write the distortionless signal as

$$z(t) = Ax(t-\tau) \tag{8.20}$$

where A is the gain and τ is the group delay to the point in the system at which the SNR is to be defined. Let $x(t)$ be the reference signal and $y(t)$ be the measurement signal, such that

$$y(t) = Ax(t-\tau) + n(t) + d(t) \tag{8.21}$$

where $n(t)$ represents the external additive noise and $d(t)$ is the signal-dependent internal distortion induced by the system, which could result from intersymbol interference or a nonlinearity. A block diagram that depicts the relationships among different signals in the system is given in Figure 8.11.

The noise power is defined as the MSE between $y(t)$ and the output of the distortionless system $z(t) = Ax(t-\tau)$. That is:

$$\varepsilon(A,\tau) = E\{[y(t) - Ax(t-\tau)]^2\} \tag{8.22}$$

The desired estimates for A and τ are the values for which $\varepsilon(A,\tau)$ is minimized. The preceding expression can be written

$$\varepsilon(A,\tau) = E\{y^2(t) + A^2x^2(t-\tau) - 2Ax(t-\tau)y(t)\} \tag{8.23}$$

For stationary signals, the moments are independent of the time origin. In addition, the expectation of a sum is the sum of the expected values. Equation (8.23) can therefore be written

$$\varepsilon(A,\tau) = E\{y^2(t)\} + A^2E\left\{x^2(t)\right\} - 2AE\{x(t)y(t+\tau)\} \tag{8.24}$$

or

$$\varepsilon(A,\tau) = P_y + A^2P_x - 2AR_{xy}(\tau) \tag{8.25}$$

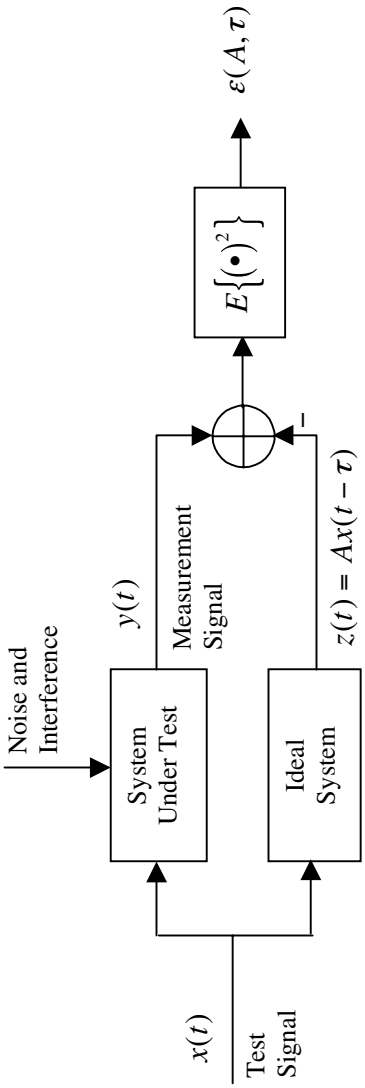


Figure 8.11 Test procedure for estimating gain, delay, and the signal-to-noise ratio.

where P_x and P_y represent the average powers in $x(t)$ and $y(t)$, respectively. Clearly, the value of τ which minimizes $\varepsilon(A,\tau)$ is the value of τ, denoted τ_m, for which $R_{xy}(\tau)$ is maximized. We refer to this as the system time delay. The system gain, A_m, is the value of A for which

$$\frac{d}{dA}\varepsilon(A,\tau_m) = \frac{d}{dA}\left[P_y + A^2P_x - 2AR_{xy}(\tau_m)\right] = 0 \tag{8.26}$$

This gives

$$A_m = \frac{R_{xy}(\tau_m)}{P_x} \tag{8.27}$$

The power in the error signal is found from (8.25) with $A = A_m$ and $\tau = \tau_m$. Since this is the component of $y(t)$ orthogonal to the signal $x(t)$, we define the power in the error signal as the noise power, N. Thus:

$$N = P_y - \frac{R_{xy}^2(\tau_m)}{P_x} \tag{8.28}$$

The power in the signal component of $y(t)$ is

$$S = P_z = A_m^2P_x = \frac{R_{xy}(\tau_m)}{P_x} \tag{8.29}$$

Therefore, the signal-to-noise ratio is

$$\frac{S}{N} = \frac{R_{xy}^2(\tau_m)}{P_x}\left[\frac{P_x}{P_xP_y - R_{xy}^2(\tau_m)}\right] = \frac{R_{xy}^2(\tau_m)}{P_xP_y - R_{xy}^2(\tau_m)} \tag{8.30}$$

The correlation coefficient, ρ, relating the signals $x(t)$ and $y(t)$, is defined as

$$\rho = \frac{R_{xy}^2(\tau_m)}{\sqrt{P_xP_y}} \tag{8.31}$$

With this definition, the SNR at the measurement point $y(t)$ takes the very simple form

$$\frac{S}{N} = \frac{\rho^2}{1-\rho^2} \tag{8.32}$$

For systems with real signals, this problem has been studied by Turner, Tranter, and Eggleston [4], and Jeruchim and Wolfe [5]. To minimize $\varepsilon\,(A,\tau)$ is equivalent to the maximization of $R_{xy}\,(\tau)$, the cross-correlation function between the reference signal $x\,(t)$, and the measured signal $y\,(t)$.

A MATLAB function for implementing a postprocessor for estimating the system gain, delay, and the SNR follows:

```
function [gain,delay,px,py,rxy,rho,snrdb] = snrmse(x,y)
ln = length(x);           % length of the reference (x) vector
fx = fft(x,ln);           % FFT the reference (x) vector
fy = fft(y,ln);           % FFT the measurement (y) vector
fxconj = conj(fx);        % conjugate the FFT of the reference vector
sxy = fy .* fxconj;       % determine the cross PSD
rxy = ifft(sxy,ln);       % determine the cross correlation function
rxy = real(rxy)/ln;       % take the real part and scale
px = x*x'/ln;             % determine power in reference vector
py = y*y'/ln;             % determine power in measurement vector
[rxymax,j] = max(rxy);    % find the max of the cross correlation
gain = rxymax/px;         % system gain
delay = j-1;              % system delay
rxy2 = rxymax*rxymax;     % square rxymax for later use
rho = rxymax/sqrt(px*py); % correlation coefficient
snr = rxy2/(px*py-rxy2);  % snr
snrdb = 10*log10(snr);    % snr in db
% End of function file.
```

We now pause to work a simple example. (Note: The technique used here for estimating delay will be used in Chapter 10 when we consider semianalytic simulation.)

Example 8.5. In order to illustrate the preceding techniques, assume that $x(t)$ is the sinusoidal signal

$$x(t) = A\sin(2\pi f_d t) \tag{8.33}$$

and that the measurement signal (the signal for which the SNR is to be determined) is

$$y(t) = GA\sin(2\pi f_d t + \phi) + B\sin(2\pi f_i t) + \sigma_n n(t) \tag{8.34}$$

where G is the system gain, $n(t)$ is a zero-mean unit variance white Gaussian noise process, and σ_n is the standard deviation of the additive noise process. The PSD of $y(t)$ is illustrated in Figure 8.12, where P_d is the signal power (the power in the desired component), P_i is the power in the interfering tone, and N_0 is the single-sided power spectral density of the noise component. It was shown in Chapter 7 that N_0 and σ_n are related by

$$\sigma_n^2 = N_0\frac{f_s}{2} \tag{8.35}$$

where f_s is the sampling frequency.

For this example, the reference signal is defined by

$$x(t) = 80\sin[2\pi(2)t] \tag{8.36}$$

The signal at the receiver input is assumed to be

$$y(t) = 20\sin\left[2\pi(2)t - \frac{\pi}{4}\right] + 4\sin[2\pi(8)t] + 0.8n(t) \tag{8.37}$$

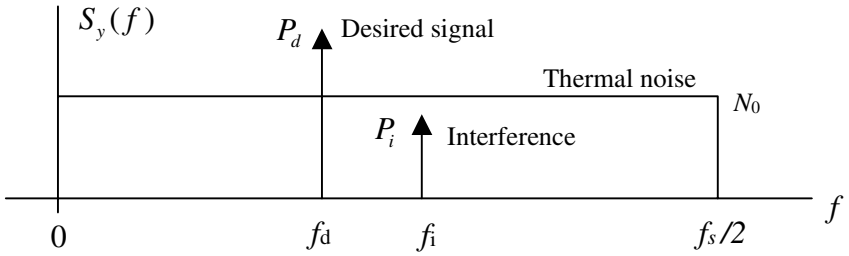


Figure 8.12 Single-sided PSD of the measurement signal $y(t)$.

where $n(t)$ is a sample function of a zero-mean unit-variance process. The MATLAB program for this senario follows:

```
% File:  c8_snrexample.m
kpts = 1024;                        % FFT Block size
k = 1:kpts;                         % sample index vector
fd = 2;                             % desired signal frequency
fi = 8;                             % interference frequency
Ax = 80; Ayd = 20; Ayi =4;          % amplitudes
phase = pi/4;                       % phase shift
nstd = 0.8;                         % noise standard deviation
%
theta = 2*pi*k/kpts;                % phase vector
x = Ax*sin(fd*theta);               % desired signal
yd = Ayd*sin(fd*theta+pi/4);        % desired signal at receiver input
yi = Ayi*sin(fi*theta);             % interference
noise = nstd*randn(1,kpts);         % noise at receiver input
yy = yd+yi+noise;                   % receiver input
[gain,delay,px,py,rxy,rho,snrdb] = snrmse(x,yy);
%
% display results
%
cpx = [‘The value of Px is ’,num2str(px),‘.’];
cpy = [‘The value of Py is ’,num2str(py),‘.’];
cgain = [‘The value gain is ’,num2str(gain),‘.’];
cdel = [‘The value of delay is ’,num2str(delay),‘.’];
csnrdb = [‘The value of SNR is ’,num2str(snrdb),‘ dB.’];
disp(‘ ’)                           % insert blank line
disp(cpx)
disp(cpy)
disp(cgain)
disp(cdel)
```

```
disp(csnrdb)
% End of script file.
```

Executing the program yields the following results:

```
The value of Px is 3200.
The value of Py is 208.7872.
The value gain is 0.25012.
The value of delay is 64.
The value of SNR is 13.6728 dB.
```

The theoretical values are easily computed. Since the reference signal is a sinusoid having a peak value of 80:

$$P_x = \frac{1}{2}(80)^2 = 3200 \tag{8.38}$$

Three components are present at the receiver input: the sinusoidal signal component at 2 Hz, the sinusoidal interference component at 8 Hz, and the white noise component. The power P_y is the sum of these components. This yields

$$P_y = \frac{1}{2}(20)^2 + \frac{1}{2}(4)^2 + (0.8)^2 = 208.64 \tag{8.39}$$

The gain is the ratio of the amplitude of the measurement signal to the amplitude of the corresponding component at the receiver input (the component at 2 Hz). This gives

$$G = \frac{20}{80} = 0.25 \tag{8.40}$$

Noting that the signal component has a period of 512 samples [$x(t)$ goes through two periods in the span of 1,024 samples], and that the phase delay is $\pi/4$, the delay is

$$\tau = \frac{\pi/4}{2\pi}(512) = 64 \text{ samples} \tag{8.41}$$

The SNR is the ratio of the interference plus noise power to the signal at the input to the receiver. This is the ratio of the first term in (8.39) to the sum of the last two terms in (8.39). (Note that the interference is considered noise, since the interference is orthogonal to the signal component.) This gives

$$\frac{S}{N} = \frac{200}{8.64} = 23.1481 = 13.6452 \text{ dB} \tag{8.42}$$

These results are summarized in Table 8.2. The small errors are due to the fact that the noise variance σ_n^2 is a random variable since the record length is finite. Modifying

Table 8.2 Summary of Results for Example 8.2

Parameter	Theoretical Value	Estimated Value
P_x	3,200	3,200
P_y	208.64	208.7872
G	0.25	0.25012
τ	64 samples	64 samples
S/N	13.6452 dB	13.6728 dB

the program slightly so that five estimates of the SNR (in dB) are generated results in the vector output

$$[13.6572 \quad 13.6524 \quad 13.5016 \quad 13.5245 \quad 13.5201]$$

We clearly see that the estimated SNR is a random variable.

The single biggest difficulty with this method is the accurate determination of delay. Note that if a small error in the estimation of delay occurs, a small error will result in the estimated value of $R_{xy}^2(\tau_m)$. If the SNR is large, $P_x P_y - R_{xy}^2(\tau_m) \lessapprox 0$ will result and, as we see from (8.30) a small error in the estimation of $R_{xy}^2(\tau_m)$ will result in a large error in the estimated signal-to-noise ratio. We must therefore be able to accurately determine the peak value of $R_{xy}^2(\tau)$. This may require that $R_{xy}^2(\tau)$ be closely sampled, which requires a high sampling frequency for the simulation. Thus, we have the ubiquitous tradeoff between accuracy and the time required to execute the simulation. ■

In this development, we have assumed that the signals are real. The technique illustrated here can be applied with equal ease to signals defined by complex envelopes. The estimator for this case is derived by replacing $x(t)$ by $x_d(t) + jx_q(t)$ and $y(t)$ by $y_d(t) + jy_q(t)$. The resulting expression for the SNR is, once again, (8.32). The details of the development are left to the interested student.

8.3 Coding

When simulation is used to determine the bit error rate (BER) of a digital communication system that makes use of error control coding, one usually does not use the simulation to count errors at the output of the decoder. There are a variety of reasons for this. First, the BER at the decoder output is usually very small. Consequently, very long simulation run times are required to collect a sufficient number of errors to generate accurate estimates of the BER. Also, many decoding algorithms are computationally complex, which also significantly increases the simulation run time. In addition, both coders and decoders are deterministic devices. Once the code is defined, the source data uniquely determines the codewords. Similarly, the pattern of errors at the receiver output uniquely determines the BER at the decoder output. This suggests a semianalytic approach in which the symbol error rate (SER) at the receiver input, determined using simulation, is mapped to the decoded BER using analysis. Performing this mapping is, in general, a complex

task if exact results are desired. Fortunately, however, exact results are seldom necessary, and a number of useful approximations and bounds have been developed to simplify this task.

A waveform-level simulation is typically used to determine the symbol error rate, SER, at the receiver input. An alternative method is to use discrete channel models implemented as Markov models (HMMs). The HMM is a computationally efficient technique for simulating systems for a given set of channel conditions and is therefore very useful for studying the impact of various coding/decoding algorithms. The discrete channel model and the HMM will be studied in detail in Chapter 15.

8.3.1 Analytic Approach to Block Coding

As we know, block codes are formed by grouping information symbols into blocks of length k. To each k-symbol block is appended $(n-k)$ parity symbols to form codewords of length n. These codewords are then transmitted through the channel and, due to disturbances in the channel, random errors may result. In most practical applications, the n-symbol codewords are transmitted in a time slot of duration kT_b, where T_b denotes the time for transmitting a single information bit without coding. If the transmitted power is the same with and without coding, a typical assumption, the energy associated with transmission of the code symbols is $(k/n)E_b$, where E_b is the energy per bit and k/n is the code rate. Since the energy per transmitted symbol is reduced through the use of error control coding, the channel symbol error probability with coding is increased over the symbol (bit) error probability without coding. One hopes that the added redundancy, through the addition of parity symbols, will provide sufficient error correction capability to provide a net increase in system performance. This may or may not be true.

Assume that a given code can correct up to t errors in each n-symbol block. Also assume that error events are independent, which can at least be approximately ensured by using interleaving. The probability of error associated with the code symbols transmitted through the channel is denoted P_{sc}, where the subscript denotes channel symbols as opposed to information bits. Since t errors per n-symbol codeword can be corrected by the decoder, the probability that the decoded *word* will be in error, P_{cw}, is

$$P_{cw} \leq \sum_{i=t+1}^{n} \binom{n}{i} P_{sc}^{i}(1-P_{sc})^{n-i} \tag{8.43}$$

Equality holds in (8.43) if all received blocks of n symbols containing t or fewer errors are decoded correctly and no blocks of n symbols containing $t+1$ or more errors are decoded correctly. These are known as perfect codes. The only perfect binary codes are the repetition codes for which n is odd, the single error-correcting Hamming codes and the triple error-correcting (23,12) Golay code. For all other codes (8.43) provides a useful bound.

The decoded word error probability, P_{cw}, does not allow direct comparison of different codes. In order to compare different codes it is necessary to map the decoded word error probability to a decoded information bit error probability, which we de-

note P_b. An exact mapping is a function of the generator matrix of the code, which determines the code weight distribution. Fortunately, a highly accurate approximation has been developed [16, 17]. This approximation is

$$P_b \approx \frac{q}{2(q-1)} \cdot \left[\frac{d}{n} \sum_{i=t+1}^{d} \binom{n}{i} P_{sc}^i (1-P_{sc})^{n-i} + \frac{1}{n} \sum_{i=d+1i}^{n} \binom{n}{i} P_{sc}^i (1-P_{sc})^{n-i} \right] \tag{8.44}$$

where q denotes a q-ary channel. In other words, for binary channels $q = 2$, and for Reed-Solomon codes, the most popular nonbinary block code, $q = 2^k - 1$.

Example 8.6. We now illustrate the use of (8.44). Assume a binary ($q = 2$) phase shift keying (PSK) communications system operating in an additive, white, Gaussian noise (AWGN) environment. For this case the bit error probability, without coding, is

$$P_b = Q\left(\sqrt{2z}\right) \tag{8.45}$$

where z represents E_b/N_0. With an (n, k) block code, the channel symbol error probability is

$$P_{sc} = Q\left(\sqrt{\frac{2kz}{n}}\right) \tag{8.46}$$

Two different binary codes are considered: a $(23, 12)$ Golay code for which $n = 23$, $t = 3$, and $d = 7$, and a $(15, 11)$ Hamming code for which $n = 15$, $t = 1$, and $d = 3$.

At this point, all parameters and variables in (8.44) are known. Prior to evaluating (8.44), however, we must evaluate

$$\binom{n}{i} = \frac{n!}{i!(n-i)!} \tag{8.47}$$

The MATLAB function for evaluating (8.47) follows:[5]

```
function out = nkchoose(n,k)
a = sum(log(1:n));              % ln of n!
b = sum(log(1:k));              % ln of k!
c = sum(log(1:(n-k)));          % ln of (n-k)!
out = round(exp(a-b-c));        % result
% End of function file.
```

[5]MATLAB contains the function **nchoosek** $= n!/\big(k!(n-k)!\big)$ as a standard m-file. The routine given here is named **nkchoose** in order to avoid an obvious conflict with **mchoosek**. The technique given here uses logarithms to increase the dynamic range of the computation and is handy in those applications where n is so large that $n!$ results in an overflow.

The MATLAB routine for computing the performance curves for a (15,11) Hamming code and a triple error correcting (23,12) Golay code follows: (Note that PSK modulation and an AWGN channel is assumed.)

```
% File c8_cerdemo
zdB = 0:0.1:10;                          % set Eb/No axis in dB
z = 10.^(zdB/10);                        % convert to linear scale
ber1 = Q(sqrt(2*z));                     % PSK result
ber2 = Q(sqrt(12*2*z/23));               % CSER for (23,12) Golay code
ber3 = Q(sqrt(11*z*2/15));               % CSER for (15,11) Hamming code
berg = cer2ber(2,23,7,3,ber2);           % BER for Golay code
berh = cer2ber(2,15,3,1,ber3);           % BER for Hamming code
semilogy(zdB,ber1,zdB,berg,zdB,berh)  % plot results
xlabel('E_b/N_o in dB')                  % label x axis
ylabel('Bit Error Probability')          % label y axis
% End of scrit file.
```

The preceding MATLAB code makes use of the function `cer2ber`, which converts the channel symbol error probability to the approximation of the decoded bit error probability given by (8.44). The MATLAB code for implementing this function is as follows:

```
function [ber] = cer2ber(q,n,d,t,ps)
% Converts channel symbol error rate to decoded BER.
lnps = length(ps);                         % length of error vector
ber = zeros(1,lnps);                       % initialize output vector
for k=1:lnps                               % iterate error vector
    cer = ps(k);                           % channel symbol error rate
    sum1 = 0; sum2 = 0;                    % initialize sums
    %
    % first loop eveluates first sum
    %
    for i=(t+1):d
        term = nkchoose(n,i)*(cer^i)*((1-cer))^(n-i);
        sum1 = sum1+term;
    end
    %
    % second loop evaluates second sum
    %
    for i=(d+1):n
        term = i*nkchoose(n,i)*(cer^i)*((1-cer)^(n-i));
        sum2 = sum2+term;
    end
    %
    % compute BER (output)
    %
```

```
   ber(k) = (q/(2*(q-1)))*((d/n)*sum1+(1/n)*sum2);
end
% End of function file.
```

The result of these computations are illustrated in Figure 8.13. ■

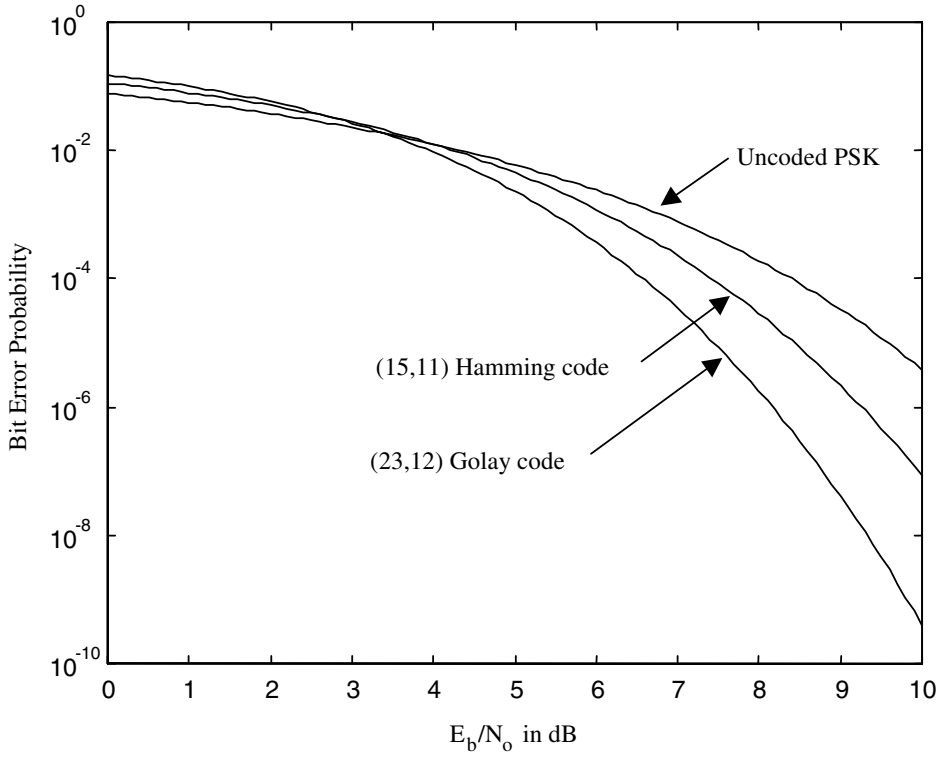


Figure 8.13 Performance comparisons for Hamming and Golay block codes.

8.3.2 Analytic Approach to Convolutional Coding

A number of analytic approximations can be used to map the channel symbol error probability to a decoded bit error probability for the convolutional code case. These mappings take the form of upper bounds on the error probability and are therefore the convolutional code equivalent of (8.44). These bounds are usually based on the Viterbi decoding algorithm, which asymptotically approaches the maximum likelihood decoder performance, and is the standard for decoding convolutional codes.

A frequently used bound is based on the transfer function of the convolutional code. The transfer function describes the distance properties of the convolutional code and can be derived from the state transition diagram of the code.

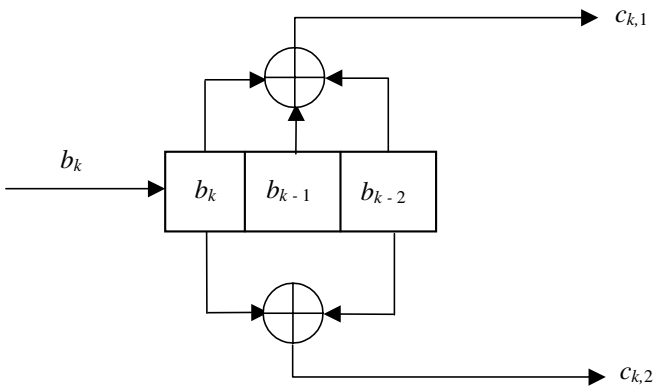


Figure 8.14 Rate 1/2 convolutional coder for Example 8.6.

The transfer function for the rate 1/2 convolutional coder shown in Figure 8.14 is given by [18, 19]

$$T(D, L, I) = \frac{D^5 L^3 I}{1 - DL(1 + L)I} \tag{8.48}$$

Expressing (8.48) in polynomial form gives

$$T(D, L, I) = \sum_{k=0}^{\infty} D^{5+k} L^{3+k} (1 + L)^k I^{1+k} = D^5 L^3 I + D^6 L^4 (1 + L) I^2 + \cdots \tag{8.49}$$

which describes the distance properties of various paths in the trellis for the code that starts at state 0 and merge to state 0 later on. The power of D denotes the Hamming distance (the number of binary ones) separating the given path from the all-zeros path in the decoding trellis. The power of L indicates the length of a given path. In other words, the exponent of L is incremented each time a branch in the trellis is traversed. The power of I is incremented if the branch transition results from a binary one input, and is not incremented if the branch transition results from a binary zero input. For example, the term $D^5 L^3 I$ represents a path having Hamming distance 5 from the all-zeros path. This path has length 3 and results from input data having 1 binary one and 2 binary zeros (100 to be exact). The next term, $D^6 L^4 (1 + L) I^2 = D^6 L^4 I^2 + D^6 L^5 I^2$, represents two paths, each of which lie Hamming distance 6 from the all-zeros path. One path has a length of 4 branches and the other path has a length of 5 branches. The path of length 4 results from an input of 2 ones and 2 zeros, and the second a results from an input having 2 ones and 3 zeros. The smallest output weight of all the paths that begin and merge with the state of all zeros represents the minimum free distance, d_f, of the code, which is 5 in this case.

In order to approximate the decoded bit error probability, we first let $L = 1$ in (8.48), since we do not have interest in the path lengths. This gives

$$T(D, I) = T(D, L, I)|_{L=1} = \frac{D^5 I}{1 - 2DI} \tag{8.50}$$

For antipodal signaling (PSK) in an AWGN environment, the decoded symbol error probability is given by [18]

$$P_E < R \left. \frac{\partial T(D, I)}{\partial I} \right|_{I=1, D=\exp(RE_b/N_0)} \tag{8.51}$$

where R is the code rate ($R = 1/2$ in this case.). This result is used in Example 8.6. For a general binary symmetric channel, the Bhattachayya bound is used. This gives [18, 19]

$$P_E < \left. \frac{\partial T(D, I)}{\partial I} \right|_{I=1, D=d} \tag{8.52}$$

where

$$d = \sqrt{4q(1 - q)} \tag{8.53}$$

and q is the channel symbol error probability determined by simulation. The bounds defined by (8.51) and (8.52) can be rather loose. This is especially true of short constraint length codes.

Equations (8.51) and (8.52) assume hard decision decoding. With soft decision decoding, d in (8.51) is replaced d_0, where

$$d_0 = \sum_{i=1}^{N} \sqrt{\Pr(y_i|0) \Pr(y_i|1)} \tag{8.54}$$

in which N is the number of quantizer output values, y_i is the i^{th} quantized output value, and the conditional probabilities represent the probability of a 0 or 1 at the channel input appearing at the quantizer output as level y_i. These conditional probabilities are estimated using either a Monte Carlo or semianalytic technique over the waveform channel.

Example 8.7. We now apply (8.51) to the rate 1/2 code defined by (8.50), and illustrated in Figure 8.14. Substitution of (8.50) into (8.52) yields

$$P_E < \frac{1}{2} \frac{\partial}{\partial I} \left(\frac{D^5 I}{1 - 2DI} \right) \Bigg|_{I=1, D=\exp(RE_b/N_0)} = \frac{1}{2} \left. \frac{D^5}{(1 - 2D)^2} \right|_{D=\exp(RE_b/N_0)} \tag{8.55}$$

The following MATLAB code results:

```
% File c8_convcode.m
zdB = 2:0.1:10;                  % set Eb/No axis in dB
z = 10.^(zdB/10);                % convert to linear scale
puc = Q(sqrt(2*z));              % uncoded BER
W = exp(-z/2);
Num = W.^5;
Den = 1-4*W+4*W.*W;
ps = 0.5*Num./Den;
semilogy(zdB,puc,'-.',zdB,ps)
grid
legend('uncoded','coded')
xlabel('E_b/N_o in dB')          % label x axis
ylabel('Bit Error Probability')  % label y axis
% End of script file.
```

Executing the code results in Figure 8.15 in which the bound on decoded error probability and the uncoded error probability are compared. ■

8.4 Summary

In this chapter we have considered the topic of postprocessing by giving a number of examples. The generation of waveform plots, signal constellations (scatter plots) and eye diagrams were demonstrated within the context of a $\pi/4$ DQPSK modulator. We also considered the development of estimators based on data generated by a simulation. First we considered the histogram, which is an estimator for probability density functions. While the histogram is, in general, a biased and nonconsistent estimator for a pdf, we saw that if sufficient data is available, both the bias and the variance can be made negligible. Next we considered variations of the basic periodogram as an estimator of the PSD of a signal. We saw that while the variance of the basic periodogram is often unacceptably large, the variance can be reduced by averaging periodograms of windowed data segments. This process of segmenting and averaging periodograms involves a tradeoff between estimator variance and resolution. The next estimator considered allowed estimation of system gain, time delay, and signal-to-noise ratio. The techniques developed here will be used in a following chapter when semianalytic techniques are considered. The final topic considered in this chapter was the estimation of decoded error probability in systems using error control coding. The technique here is to determine the channel symbol rate using simulation and to map the channel symbol error rates to the decoded bit error rate using bounding techniques.

8.5 Further Reading

For detail on MATLAB graphics capabilities, the reader is encouraged to study the latest version of the MATLAB manual. With the exception of the simple menu

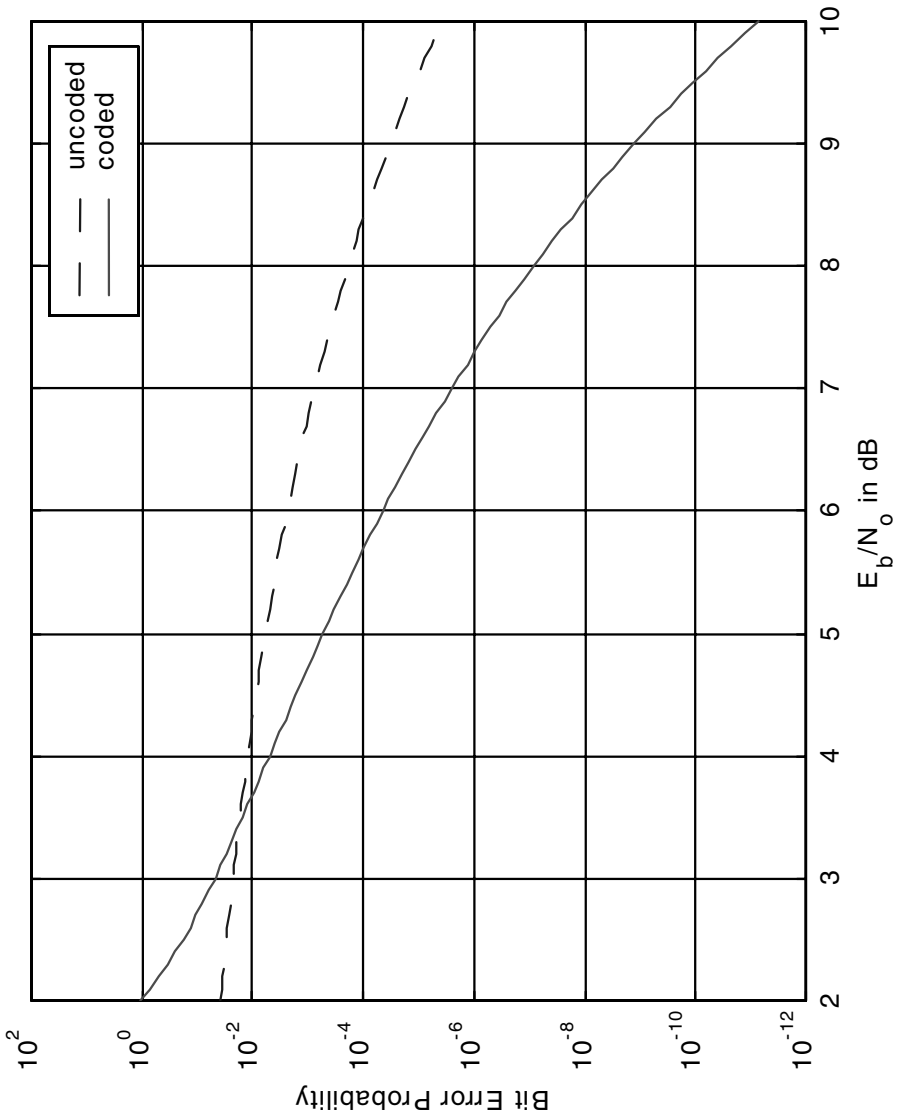


Figure 8.15 Transfer function bound for example rate 1/2 convolutional code.

given in Example 8.1, postprocessor user interfaces were not covered in this chapter. MATLAB provides routines for developing user interfaces, and the reader is encouraged to study this material. User interfaces can be used to advantage in the development of general-purpose postprocessors.

The other topic covered in this chapter dealt with estimators for probability density functions, power spectral density, gain, delay, and signal-to-noise ratio, as well as estimators for approximating the performance of coded systems based on the uncoded symbol error rate on the channel. The references given below provide detailed information on these topics.

8.6 References

1. T. S. Rappaport, *Wireless Communications: Principles and Practice*, 2nd ed., Upper Saddle River, NJ: Prentice Hall, 2002.

2. *I-Q Tutor: HP Digital Microwave Communications Guide*, Hewlett Packard Part Number 11736-90002, 1985.

3. E. A. Lee and D. G. Messerschmitt, *Digital Communication*, 2nd ed., Boston: Kluwer Academic Publishers, 1994.

4. K. S. Shanmugan and A. M. Breipohl, *Random Signals: Detection, Estimation and Data Analysis*, New York: Wiley, 1988.

5. P. Stoick and R. Moses, *Introduction to Spectral Analysis*, Upper Saddle River, NJ: Prentice Hall, 1977.

6. A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing*, Englewood Cliffs, NJ: Prentice Hall, 1975.

7. A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, Englewood Cliffs, NJ: Prentice Hall, 1989.

8. S. L. Marple, Jr., *Digital Spectral Analysis With Applications*, Upper Saddle River, NJ: Prentice Hall, 1987.

9. S. M. Kay, *Modern Spectral Estimation: Theory and Applications*, Upper Saddle River, NJ: Prentice Hall, 1988.

10. F. J. Harris, "On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform," *Proceedings of the IEEE*, Vol. 66, January 1978, pp. 51–83.

11. T. H. Shepertycki, "Telemetry Error Measurements Using Pseudo-Random Signals," *IEEE Transactions on Space Electronics and Telemetry*, Vol. 10, September 1964, pp. 111–115.

12. R. M. Gagliardi and C. M. Thomas, "PCM Data Reliability Monitoring Through Estimation of Signal-to-Noise Ratio," *IEEE Transactions on Communications Technology*, Vol. 16, June 1968, pp. 479–486.

13. M. S. Rafie, J. L. Fernandez, and K. S. Shanmugan, "Simulation-Based Estimation of Intermodulation Distortion and C/IM," IEEE GLOBECOM Conference, 1992, pp. 700–706.

14. M. D. Turner, W. H. Tranter, and T. W. Eggleston, "The Estimation of Signal-to-Noise Ratios in Digital Computer Simulations of Lowpass and Bandpass Systems," 1997 IEEE MIDCON Conference. November 1977, pp. 1–12.

15. M. C. Jeruchim and R. J. Wolfe, "Estimation of the Signal-to-Noise Ratio (SNR) in Communication Simulation," IEEE GLOBECOM Conference, 1989, pp. 35.1.1–35.1.5.

16. D. J. Torrieri, "The Information-Bit Error Rate for Block Codes," *IEEE Transactions on Communications*, Vol. 32, No. 4, April 1984, pp. 474–476.

17. D. J. Torrieri, *Principles of Secure Communications*, 2nd ed., Boston: Artech House, 1992.

18. S. G. Wilson, *Digital Modulating and Coding,* Upper Saddle River, NJ: Prentice Hall, 1996.

19. A. J. Viterbi and J. K. Omura, *Principles of Digital Communication and Coding*, New York: McGraw-Hill, 1979.

8.7 Problems

8.1 Based on our knowledge of a $\pi/4$ DQPSK transmitter as illustrated in Figure 8.1, draw a block diagram of a $\pi/4$ DQPSK receiver. Develop a MATLAB simulation of a $\pi/4$ DQPSK receiver. By combining the receiver simulation with the transmitter simulation previously developed, show that the reciever simulation works properly.

8.2 Using the MATLAB code developed for Example 8.1, determine and plot the magnitude of the complex envelope of both the unfiltered and the filtered $\pi/4$ DQPSK signal. What do you observe? Explain the results.

8.3 Using the MATLAB plotting routines, generate the (D, Q, t) coordinate system as illustrated in Figure 8.3 and plot $x_d(t)$ and $x_q(t)$ for a $\pi/4$ DQPSP signal. Use four symbols of both $x_d(t)$ and $x_q(t)$. Using the resulting MATLAB program, illustrate the generation of $x_d(t)$, $x_q(t)$, and the scatter plot from the three-dimensional image. In addition, generate, the real envelope signal. (Note: Consider the MATLAB commands `plot3`, `rotate3d`, and `view`.)

8.4 By appropriately modifying the MATLAB program in Appendix A, rework Example 8.1 so that the filter is a raised cosine filter as described in Chapter 5. Plot the direct channel and the quadrature channel signals at the filter output. Also plot the eye diagram. Compare the resulting eye diagrams with those illustrated in Figure 8.7. Use rolloff factors of 0.5 and 0.7.

8.5 A Gaussian mixture is a random process defined by the pdf

$$f_X(x) = \frac{a}{\sqrt{2\pi}\sigma_1} \exp\left[\frac{(x-m_1)^2}{2\sigma_1^2}\right] + \frac{1-a}{\sqrt{2\pi}\sigma_2} \exp\left[\frac{(x-m_2)^2}{2\sigma_2^2}\right]$$

Using the parameters $a = 0.8$, $m_1 = 0$, $m_2 = 1$, and $\sigma_1 = \sigma_2 = 1$, plot $f_X(x)$. Rework Example 8.2 using the pdf for the Gaussian mixture. Discuss the results.

8.6 Develop a postprocessor that has as input a file of N samples generated by a MATLAB program. The postprocessor is to be menu driven and is to generate a histogram, a PSD estimate of the input data, and the autocorrelation of the input data. Any necessary parameters required for operation of the postprocessor are to be entered through a parameter file read by the postprocessor. Test the postprocessor using a vector of $N = 5{,}000$ samples generated by passing N independent samples of a zero-mean, unit-variance, Gaussian process through a third-order Butterworth filter having a 3 dB break frequency of $0.2f_N$, where f_N is the Nyquist rate.

8.7 Rework Example 8.3 using a rectangular data window rather than a Hanning window. Compare the results using a rectangular window with the results using a Hanning window. How are they different? Explain the reasons for the noted differences.

8.8 Determine the theoretical PSD for Example 8.3. Compare the results given in Example 8.3 with the theoretical results.

8.9 Modify the MATLAB program given in Example 8.3 so that overlapped segments can be used. Let the number of samples that are overlapped be user specified. Rework Example 8.3 using a 50 percent overlap ($M/2$ samples) and compare the results with the result given in Example 8.3.

8.10 Using the periodogram approach, develop a "power meter" for estimating the power in a given frequency band $f_1 \leq f \leq f_2$. Demonstrate the operation of your power meter by computing the power, in a given frequency band, at the output of a suitably chosen linear system with a white noise at the system input. Your choice of a system is arbitrary, but you must verify the validity of the results given by your power meter.

8.11 The input of a linear system is

$$x[n] = 1\sin\left(\frac{4\pi n}{1024}\right), \qquad n = 1, 2, \cdots, 1024$$

Consider two outputs

$$y_1[n] = 5\sin\left(\frac{4\pi n}{1024}\right), \qquad n = 1, 2, \cdots, 1024$$

and

$$y_2[n] = 5\sin\left(\frac{4\pi n}{1024} - \frac{\pi}{2} - \frac{\pi}{512}\right), \qquad n = 1, 2, \cdots, 1024$$

For each output compute the SNR (in dB) of the measurement signal relative to the reference signal $x[n]$. Fully explain any differences. What is the theoretical SNR for each case?

8.12 The input to a linear system is defined by

$$x[n] = \sin(10\pi nT) + \sin(20\pi nT) = 2\sin(30\pi T)$$

and the output is defined by

$$y[n] = 5\sin\left(10\pi nT - \frac{\pi}{2}\right) + 5\sin\left(20\pi nT - \frac{3\pi}{4}\right) + 5\sin(30\pi nT - \pi)$$

(a) Using the input $x[n]$ as a reference, determine the system gain and delay from the input to the point where $y[n]$ is measured. Express the delay in both sample periods and in seconds. Also detemine the SNR of $y[n]$ relative to $x[n]$. You are free to choose the sampling frequency and the number of samples processed. However, you are to justify these choices.

(b) Discuss the sources of error in your results given in (a). Conduct an appropriate experiment to show that these errors are not significant.

(c) Does the system exhibit amplitude distortion? Does the system exhibit phase (delay) distortion?

(d) Suppose $y[n]$ is given by

$$y[n] = 5\sin\left(10\pi nT - \frac{\pi}{2}\right) + A\sin\left(20\pi nT - a\right) + B\sin(30\pi nT - b)$$

where A, B, a, and b are parameters. What are the values of these parameters if the system is to be distortionless? Using the techniques illustrated in Example 8.5, show that your answers are correct.

8.13 BCH codes are binary block codes that allow multiple errors per codeword to be corrected. An (n, k, t) BCH code has rate k/n, and can correct t errors per block of n symbols. By using the technique illustrated in Example 8.6, examine the relative performance of (63,30,6) and (255,123,19) BCH codes. Assume PSK modulation and compare both BCH codes to the performance of the uncoded system.

8.14 Extend Example 8.6 so that one may plot the decoded bit error probability as a function of the channel symbol error probability. Illustrate the resulting algorithm using a Golay code.

8.15 Rework Example 8.7 using the Bhattachayya bound defined by (8.53). Compare the result to that given in Example 8.7.

8.8 Appendix A: MATLAB Code for Example 8.1

8.8.1 Main Program: c8_pi4demo.m

```
% File: c8_pi4demo.m
m = 200; bits = 2*m; % number of symbols and bits
sps = 10; % samples per symbol
iphase = 0; % initial phase
order = 5; % filter order
bw = 0.2; % normalized filter bandwidth
%
% initialize vectors
%
data = zeros(1,bits); d = zeros(1,m); q = zeros(1,m);
dd = zeros(1,m); qq = zeros(1,m); theta = zeros(1,m);
thetaout = zeros(1,sps*m);
%
% set direct and quadrature bit streams
%
data = round(rand(1,bits));
dd = data(1:2:bits-1);
qq = data(2:2:bits);
%
% main programs
%
theta(1) = iphase; % set initial phase
thetaout(1:sps) = theta(1)*ones(1,sps);
for k=2:m
    if dd(k) == 1
        phi_k = (2*qq(k)-1)*pi/4;
    else
        phi_k = (2*qq(k)-1)*3*pi/4;
    end
    theta(k) = phi_k + theta(k-1);
    for i=1:sps
        j = (k-1)*sps+i;
        thetaout(j) = theta(k);
    end
end
d = cos(thetaout);
q = sin(thetaout);
[b,a] = butter(order,bw);
df = filter(b,a,d);
qf = filter(b,a,q);
%
% postprocessor for plotting
```

```
%
kk = 0; % set exit counter
while kk == 0 % test exit counter
k = menu('pi/4 QPSK Plot Options',...
    'Unfiltered pi/4 QPSK Signal Constellation',...
    'Unfiltered pi/4 QPSK Eye Diagram',...
    'Filtered pi/4 QPSK Signal Constellation',...
    'Filtered pi/4 OQPSK Eye Diagram',...
    'Unfiltered Direct and Quadrature Signals',...
    'Filtered Direct and Quadrature Signals',...
    'Exit Program');
    if k == 1
        sigcon(d,q) % plot unfiltered signal con.
        pause
    elseif k ==2
        dqeye(d,q,4*sps) % plot unfiltered eye diagram
        pause
    elseif k == 3
        sigcon(df,qf) % plot filtered signal con.
        pause
    elseif k == 4
        dqeye(df,qf,4*sps) % plot filtered eye diagram
        pause
    elseif k == 5
        numbsym = 10; % number of symbols plotted
        dt = d(1:numbsym*sps); % truncate d vector
        qt = q(1:numbsym*sps); % truncate q vector
        dqplot(dt,qt) % plot truncated d and q signals
        pause
    elseif k == 6
        numbsym = 10; % number of symbols to be plotted
        dft=df(1:numbsym*sps); % truncate df to desired value
        qft=qf(1:numbsym*sps); % truncate qf to desired value
        dqplot(dft,qft) % plot truncated signals
        pause
    elseif k == 7
        kk = 1; % set exit counter to exit value
    end
end
% End of script file.
```

8.8.2 Supporting Routines

sigcon.m

```
function []=sigcon(x,y)
plot(x,y)
axis('square')
axis('equal')
title('SIGNAL CONSTELLATION')
xlabel('Direct Channel')
ylabel('Quadrature Channel')
% End of function file.
```

dqeye.m

```
function [] = dqeye(xd,xq,m)
lx = length(xd); % samples in data segment
kcol = floor(lx/m); % number of columns
xda = [0,xd]; xqa = [0,xq]; % append zeros
for j = 1:kcol % column index
    for i = 1:(m+1) % row index
        kk = (j-1)*m+i; % sample index
        y1(i,j) = xda(kk);
        y2(i,j) = xqa(kk);
    end
end
subplot(211) % direct channel
plot(y1,'k');
title('D/Q EYE DIAGRAM');
xlabel('Sample Index');
ylabel('Direct');
subplot(212) % quadrature channel
plot(y2,'k');
xlabel('Sample Index');
ylabel('Quadratute');
subplot(111)
% End of function file.
```

dqplot.m

```
function [] = dqplot(xd,xq)
lx = length(xd);
t = 0:lx-1;
nt = t/(lx-1);
nxd = xd(1,1:lx);
nxq = xq(1,1:lx);
subplot(211)
```

```
plot(nt,nxd);
a = axis;
axis([a(1) a(2) 1.5*a(3) 1.5*a(4)]);
title('Direct and Quadrature Channel Signals');
xlabel('Normalized Time');
ylabel('Direct');
subplot(212)
plot(nt,nxq);
a = axis;
axis([a(1) a(2) 1.5*a(3) 1.5*a(4)]);
xlabel('Normalized Time');
ylabel('Quadratute');
subplot(111)
% End of function file.
```

Chapter 9

INTRODUCTION TO MONTE CARLO METHODS

The purpose of this chapter is to briefly introduce the basics of the Monte Carlo technique for estimating the value of a parameter. There is no attempt to be rigorous, and this chapter covers only a few of the important aspects of Monte Carlo estimation techniques. The goal of this chapter is to define the Monte Carlo method and examine some of the basic techniques in a simple and easily understood context. The important issues of confidence intervals and convergence are briefly examined. Throughout this chapter we assume that observations used by the estimator are independent. This assumption will be relaxed in the following chapter, in which we consider simulation techniques in more detail.

9.1 Fundamental Concepts

Monte Carlo simulations are based on games of chance. This is of course the reason for the name "Monte Carlo," the Mediterranean city famous for casino gambling. In the material to follow, we use the closely related terms "Monte Carlo estimation"

and "Monte Carlo simulation" almost interchangeably. Monte Carlo simulation describes a simulation in which a parameter of a system, such as the bit error rate (BER), is estimated using Monte Carlo techniques. Monte Carlo estimation is the process of estimating the value of a parameter by performing an underlying stochastic, or random, experiment.

9.1.1 Relative Frequency

Monte Carlo estimation is based on the relative frequency interpretation of probability [1]. In defining relative frequency, the first step is to specify a random experiment and an event of interest, A. We recall from basic probability theory that a random experiment is an experiment in which the result, or outcome of performing the experiment, cannot be predicted exactly but can be defined statistically. The most basic random experiment is flipping a coin in which there are two outcomes of interest defined by the set $\{Heads, Tails\}$. Prior to flipping the coin it is unknown which outcome will occur. However, if it is known that the coin is an "honest" or unbiased coin, we know that the probability of each outcome in the set $\{Heads, Tails\}$ will occur with equal probability and that outcomes are independent. Performance of the random experiment determines the outcome.

An event is an outcome, or set of outcomes, associated with a random experiment. Using a digital communication system as an example, the random experiment may simply be defined as transmitting a binary 1. The result at the output of the receiver will be an estimate of the transmitted binary symbol, which will be either a binary 0 or a binary 1. The event of interest may be that an error occurred in the transmission of the binary 1. Determination of the system BER involves estimation of the conditional probability that a binary 0 was received given that a binary 1 was transmitted.

Having defined a random experiment and an event of interest, we now consider the next step in the Monte Carlo method, which is to execute the random experiment a large number of times, N. We count the number of occurrences, N_A, corresponding to an event, A, of interest. The probability of the event A is approximated by the relative frequency of the event, which is defined by N_A/N [1]. The probability of the event A, defined in the relative frequency sense, is obtained by replicating the random experiment an infinite number of times. This gives

$$\Pr(A) = \lim_{N\to\infty} \frac{N_A}{N} \tag{9.1}$$

In the context of estimating the error probability in a digital transmission system N is the total number of bits or symbols (either actually transmitted over the system or simulated) and N_A is the number of errors (either measured or simulated).

For $N < \infty$, an obvious practical necessity in Monte Carlo simulations, the quantity N_A/N, is an estimator of $\Pr(A)$. This estimator is denoted $\widehat{\Pr}(A)$. It is important to note that, because of the underlying random experiment, N_A will, for finite N, be a random variable and, consequently, $\widehat{\Pr}(A)$ is a random variable. The statistics of this random variable determine the accuracy of the estimator and, therefore, the quality of the simulation.

9.1.2 Unbiased and Consistent Estimators

In order to be useful, Monte Carlo estimators must satisfy several important properties. First, we desire that Monte Carlo estimators be *unbiased*. That is, if $\widehat{A}$ is the estimate of a parameter A, we desire that

$$E\left\{\widehat{A}\right\} = A \tag{9.2}$$

In other words, on the average the correct result is obtained.

Assume that a Monte Carlo simulation is performed a number of times resulting in a collection of estimates of the random variable of interest. Clearly we desire that these estimates exhibit a small variance. If the estimates are unbiased and have small variance, the estimator will produce estimates that cluster about the correct value of the parameter being estimated, and the spread of the estimates will be small. Analytical determination of the variance of a Monte Carlo estimator is typically a difficult task unless the underlying events are statistically independent. Almost always, however, the variance of the estimated values decrease as the simulation run length (the number of times that the underlying random experiment is replicated) increases. We refer to estimators satisfying this property as consistent. For consistent estimators, $\sigma^2_{\widehat{A}} \to 0$ as $N \to \infty$, where N represents the number of times that the random experiment is replicated. For unbiased and consistent estimators, the error

$$e = A - \widehat{A} \tag{9.3}$$

is zero-mean and the error variance, σ_e^2, converges to 0 as $N \to \infty$. Unfortunately this convergence is often very slow.

9.1.3 Monte Carlo Estimation

As a simple example of a Monte Carlo estimator, consider the determination of the area of a region having a nontrivial shape. Assume that the region whose area is to be estimated is completely bounded by a box of known area. Define the random experiment as taking random samples over the bounding box and define the event of interest, A, as the event that a sample falls within the region whose area is to be determined. For an unbiased estimator of an unknown area, it is essential that the random sample points be uniformly distributed within the bounding region of known area. This can easily be accomplished using a computer program with two uniform random number generators. The result of generating $N = 500$ uniformly distributed sampling points is illustrated in Figure 9.1. The 500 points shown in Figure 9.1 were generated using the following MATLAB code:

```
x = rand(1,500);
y = rand(1,500);
plot(x,y,'k+')
axis square
```

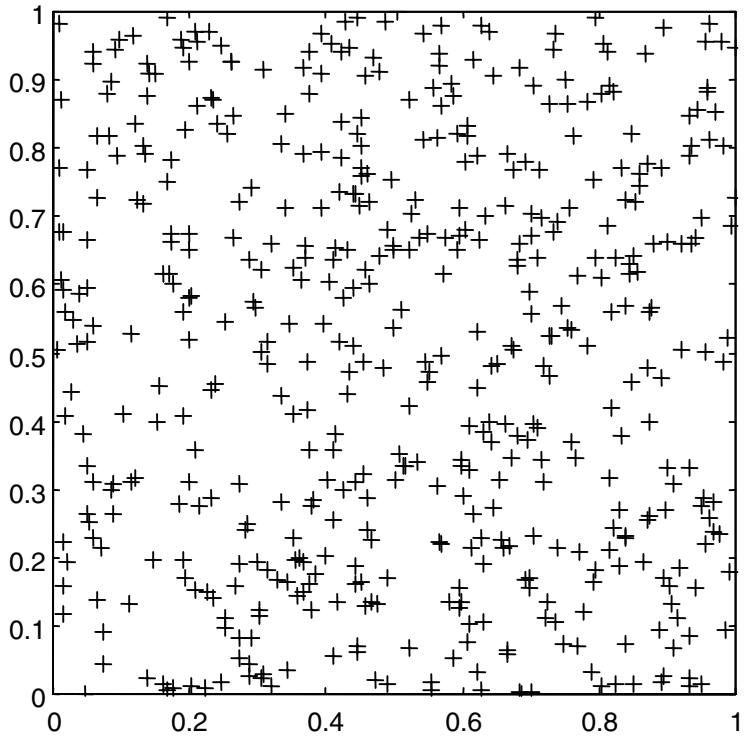


Figure 9.1 Uniformly distributed random points.

The next step is to define the event A of interest. We wish to estimate the area of the sunburst illustrated in Figure 9.2. The quantities N_{box} and $N_{sunburst}$ are defined as the number of samples falling into the bounding box and in the sunburst, respectively. Since the sample points are uniformly distributed within the bounding box, the ratio of the area of the sunburst to the area of the bounding box, $A_{suburst}/A_{box}$, is approximately equal to the ratio of the number of sample points falling in the sunburst to the number of points falling in the bounding box, $N_{sunburst}/N_{box}$. In other words

$$\frac{A_{sunburst}}{A_{box}} \approx \frac{N_{sunburst}}{N_{box}} \tag{9.4}$$

which gives

$$A_{sunburst} \approx A_{box} \frac{N_{sunburst}}{N_{box}} \tag{9.5}$$

Subject to the condition that the sample points are uniformly distributed, the approximation improves as the number of sample points are increased.

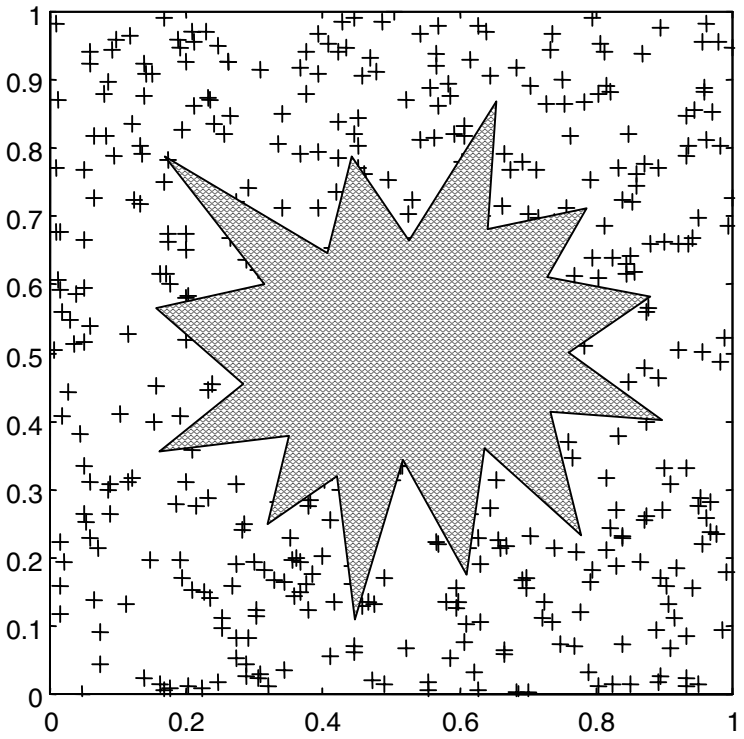


Figure 9.2 Monte Carlo estimation of an area.

In order to illustrate the Monte Carlo technique in a simple and straightforward manner, we consider a Monte Carlo estimator for the value of π. Note that the estimator is a stochastic simulation in that it is a simulation of a random experiment. This example therefore serves as an introduction to the material to be presented in the later chapters of this book.

9.1.4 The Estimation of π

One method of estimating the value of π[1] is to bound a pie-shaped area, corresponding to first quadrant of a circle with radius one, by a box of unit area. This is illustrated in Figure 9.3 together with N_{box} total sample points. If the box spans

[1]The problem of determining the numerical value of π has a rich and very interesting history that is, surprisingly, closely tied to the history of Monte Carlo simulation. Even though we often associate the development of Monte Carlo techniques with the development of the digital computer, the Monte Carlo method was apparently first suggested by Pierre Laplace approximately 200 years ago. One problem considered by Laplace was a technique for estimating π based on a problem known as Buffon's needle. This problem was posed and solved by the French scientist Count Buffon (George Leclerc) in 1777 [2].

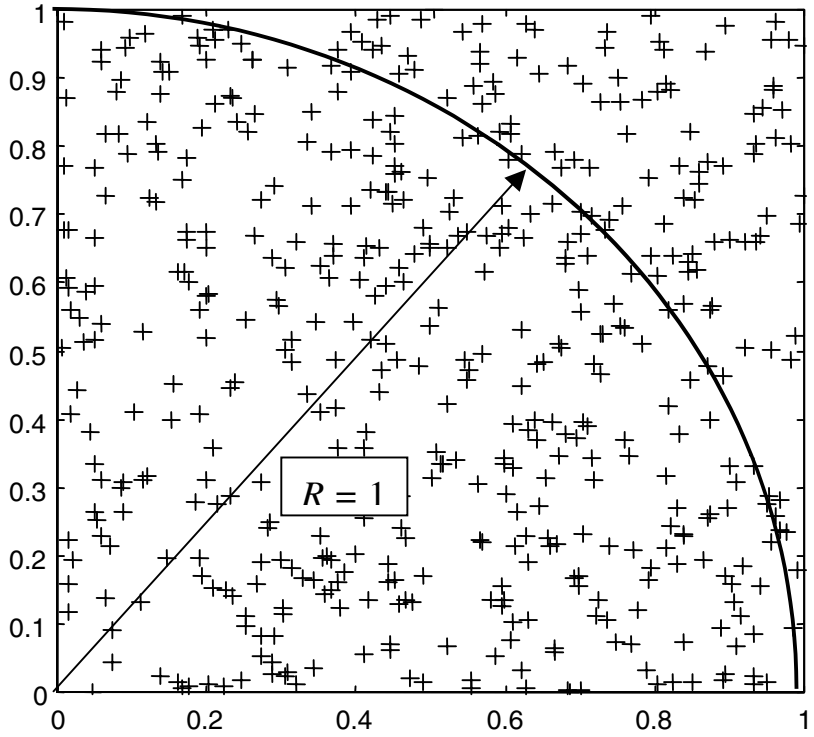


Figure 9.3 Estimation of π.

the range (0,1) on the x axis and the range (0,1) on the y axis, it is clear that $A_{box} = 1$ and that the area of the pie-shaped region (quarter circle) is

$$A_{pie\_slice} = \frac{1}{4}\left[\pi R^2\right]_{R=1} = \frac{\pi}{4} \tag{9.6}$$

It follows that

$$\frac{A_{pie\_slice}}{A_{box}} = \frac{\pi}{4} \tag{9.7}$$

Assuming that the samples are uniformly distributed, the ratio of N_{pie_slice} to N_{box} will constitute an unbiased and consistent estimator of A_{pie_slice}/A_{box}. Thus

$$\frac{N_{pie\_slice}}{N_{box}} \approx \frac{A_{pie\_slice}}{A_{box}} = \frac{\pi}{4} \tag{9.8}$$

The estimator of π, denoted $\widehat{\pi}$, is

$$\widehat{\pi} = \frac{4N_{pie\_slice}}{N_{box}} \tag{9.9}$$

It therefore follows that the value of π may be estimated by covering the bounding box with uniformly distributed points, counting the points falling within the inscribed circle, and applying (9.9).

Example 9.1. A MATLAB program can easily be written to implement the procedure just described. The results are shown in Figure 9.4 for the case in which five different estimates of π are generated with each estimate based on 500 replications of the underlying random experiment. The resulting five estimated values of π are defined by the vector

$$\widehat{\pi} = \begin{bmatrix} 3.0960 & 3.0720 & 2.9920 & 3.1600 & 3.0480 \end{bmatrix} \tag{9.10}$$

If all five estimates are averaged, the result is $\widehat{\pi} = 3.0736$. This result is equivalent to a single estimate based on 2,500 trials. The MATLAB program used to generate these results follows:

```
% File:  c9_estimatepi.m
m = input('Enter M, number of experiments > ');
n = input('Enter N, number of trials / experiment > ');
z = zeros(1,m);                    % initialize array
```

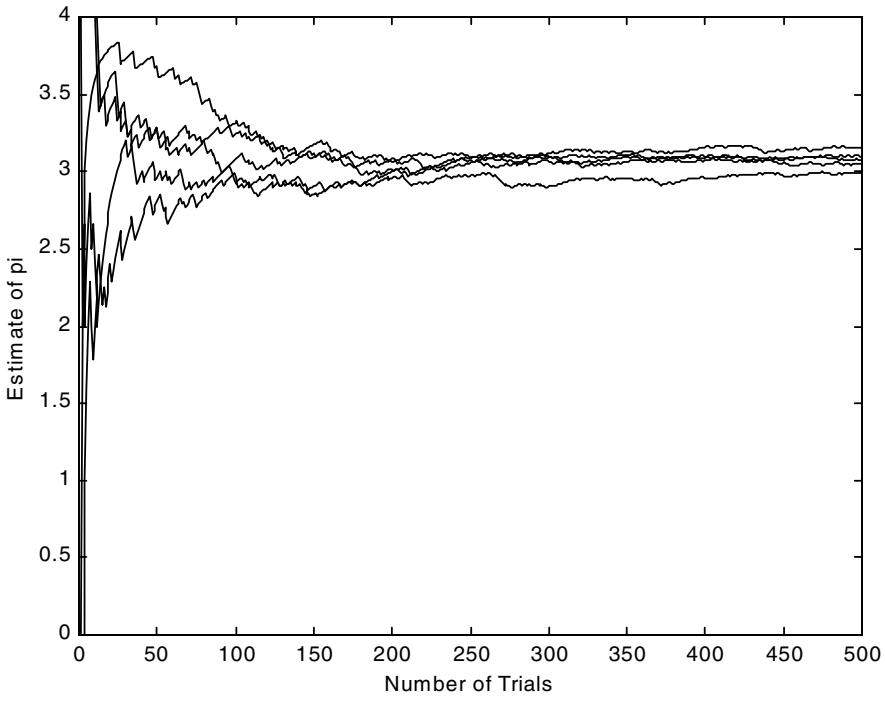


Figure 9.4 Monte Carlo estimate of π.

```
data = zeros(n,m);                  % initialize array
for j=1:m
    x = rand(1,n);
    y = rand(1,n);
    k = 0;
    for i=1:n
       if x(i)^2+y(i)^2<= 1         % Fall in pie slice?
          k=k+1;
       end
    data(i,j) = 4*(k/i);            % jth estimate of pi
    end
    z(j) = data(n,j);               % Store data
end
plot(data,'k')                      % Plot curves
xlabel('Number of Trials')
ylabel('Estimate of pi')
% End of script file.
```

■

While the preceding example was very simple, it shares a number of important attributes with all Monte Carlo simulations. There is a test condition and a couple of counters. The first counter is incremented each time the random experiment is performed and the second counter is incremented each time the test condition is satisfied. In simulations of digital communication systems, in which the goal is to estimate the bit error rate, the test condition determines whether or not an error is made on the transmission of a given bit or data symbol. The first counter is incremented each time a bit, or data symbol, is processed by the simulation, The second counter is incremented each time an error is observed. This will be demonstrated in a couple of examples in the following section. First, however, we pause to examine the characteristics of an AWGN channel.

9.2 Application to Communications Systems—The AWGN Channel

To estimate the performance of a digital communications system using Monte Carlo simulation, N symbols are passed through the system (actually a computer simulation model of the system) and the number of transmission errors N_e are counted. If N_e errors occur in N symbol transmissions, the estimator of the probability of symbol error is

$$\widehat{P}_E = \frac{N_e}{N} \tag{9.11}$$

Is this estimate biased or unbiased? Is the estimate consistent?

In order to investigate these important questions in the simplest possible context, we will assume an AWGN (additive, white, Gaussian noise) channel. In the AWGN environment the error events arising from channel noise are independent and the number of errors N_e in the transmission of N symbols is described by a binomial

distribution. We therefore pause to consider the binomial distribution in some detail. Following the discussion of the binomial distribution, we consider (9.11) as the estimator for the symbol error probability in two highly idealized communication systems.

9.2.1 The Binomial Distribution

Our task now is to determine the statistical behavior of $\widehat{P}_E$. The first step is to determine the mean and variance of N_e. For independent error events, the probability of N_e errors in N symbol transmissions is given by the binomial distribution

$$p_N(N_e) = \binom{N}{N_e} P_E^{N_e} (1 - P_E)^{N - N_e} \tag{9.12}$$

where

$$\binom{N}{k} = \frac{N!}{k!\,(N-k)!} \tag{9.13}$$

is the binomial coefficient and P_E is the probability of error on a single transmission.

The mean and variance of a random variable obeying a binomial distribution, are easily derived (see Problem 9.7). The mean of N_e is given by

$$E\{N_e\} = NP_E \tag{9.14}$$

and the variance of N_e is given by

$$\sigma_{N_e}^2 = NP_E(1 - P_E) \tag{9.15}$$

Using these results in (9.11), the mean of the Monte Carlo estimator for the probability of error is

$$E\{\widehat{P}_E\} = \frac{E\{N_e\}}{N} \tag{9.16}$$

Substitution of (9.14) into (9.16) gives

$$E\{\widehat{P}_E\} = \frac{NP_E}{N} = P_E \tag{9.17}$$

which shows that the Monte Carlo estimator of the error probability is unbiased. The variance of the Monte Carlo estimator of the probability of error is

$$\sigma_{\widehat{P}_E}^2 = \frac{\sigma_{N_e}^2}{N^2} \tag{9.18}$$

Substitution of (9.15) into (9.18) gives

$$\sigma_{\widehat{P}_E}^2 = \frac{P_E(1 - P_E)}{N} \tag{9.19}$$

which shows that the estimator is consistent, since the variance decreases as $N \rightarrow \infty$. Keep in mind that both (9.17) and (9.19) assume an underlying binomial distribution, which is valid only if the error events are independent.

In using Monte Carlo simulation to estimate a performance parameter of a communications system, such as the symbol error probability, unbiased and consistent estimates are clearly desirable. If an estimator is unbiased we know that, *on the average*, Monte Carlo simulation provides the correct result. In addition, if an estimator is to be useful it must have small variance so that, with high probability, the estimate lies in the neighborhood of the true value being estimated. If an estimator is unbiased and consistent we know that simulating more symbol transmissions, so that more errors are counted in a simulation, reduces the variance of the estimator. Equation (9.19) gives us a feel for the number of errors that must be counted in order for an estimate to have a given variance and this, in turn, provides a feel for the time required to execute a simulation. A practical problem with (9.19), however, is that it cannot be used to determine the required value of N for a given variance since P_E is unknown prior to conducting the simulation. In many practical problems, however, we may be able to determine P_E to within an order of magnitude or so by applying bounds or other analysis tools so that (9.19) may still be useful. Estimators that are biased but consistent converge to the incorrect value, which is clearly a highly undesirable situation unless we know how to remove the bias.[2]

Although it is important to know the characteristics of an estimator, in many situations proving that a given estimate is unbiased and consistent is a difficult task. It should be emphasized that all of the results obtained in this section, nice as they are, are valid only if the errors induced by the channel noise are independent so that the underlying error distribution is binomial. If the error events are correlated, such as in a bandlimited channel, the results given here are no longer valid and we are confronted with a more difficult problem. If error events are not independent, (9.11) is still a valid estimator of the error probability, however.

Example 9.2. When error events are independent, binary transmission can be modeled as a coin-tossing experiment. The transmission of N symbols is modeled by N tosses of a biased coin. We assume that outcome "tails" on the i^{th} toss corresponds to a correct decision on the i^{th} transmission and outcome "heads" on the i^{th} toss corresponds to an error on the i^{th} transmission. In this example the statistics associated with the coin-tossing experiment are determined by simulation. Since the coin tosses are independent, this experiment models binary data transmission in an AWGN channel.

Assume that outcome "tails" (no error) occurs with probability $1 - p$ and that outcome "heads" (error) occurs with probability p and that we wish to estimate the value of p by tossing the coin N times. The Monte Carlo estimator of p is

$$\widehat{p} = \frac{N_{Heads}}{N} \tag{9.20}$$

[2]Importance sampling, which will be briefly studied in Chapter 16, is a simulation technique in which an intentional bias is induced for the purpose of reducing the variance of the estimator for a given value of N. The effect of the bias is then removed so that an ubiased estimator results. In physical experiments bias may be a serious problem and is often due to calibration errors.

where N_{Heads} represents the number of "heads" that occur in a sequence of N tosses. Of course, for a given sequence of N tosses, the value of N_{Heads} can be any number between 0 and N but the probability of k "heads" in N tosses is

$$p_N(k) = \binom{N}{k} p^k (1-p)^{N-k} \tag{9.21}$$

We therefore must conduct this experiment a number of times, M, in order to estimate the statistical distribution of N_{Heads} and determine $\widehat{p}$, the estimator of p. The MATLAB program for simulating the coin tossing experiment follows:

```
% File:  c9_cointoss.m
M = 2000;                       % number of experiments
N = 500;                        % Number of tosses / experiment
H = zeros(1,M);                 % initialize array
H_theor = zeros(1,M);           % initialize array
for j=1:M
    A = rand(1,N);
    heads = 0;                  % initialize counter for heads
    for k=1:N
       if A(k)<=0.2
          heads = heads+1;      % increment counter for heads
       end
    end
    H(j) = heads;
end
H_max = max(H); H_min = min(H);
r = H_min:H_max;
[Nb] = hist(H,r);               % generate data for histogram
%
for k=H_min:H_max
    H_theor(k) = M*nkchoose(N,k)*((0.2)^k)*((0.8)^(N-k));
end
subplot(2,1,1)
hist(H,r)                       % plot histogram
xlabel('Number of heads')
ylabel('Number of occurences')
subplot(2,1,2)
plot(r,Nb,'ok',r,H_theor(1,H_min:H_max),'k')
xlabel('Number of heads')
ylabel('Number of occurences')
% End of script file.
```

Executing this program yields the result illustrated in Figure 9.5. The histogram is shown in the top pane and the outcomes of the individual experiments, along with the theoretical result, are illustrated in the bottom pane. Note that the theoretical

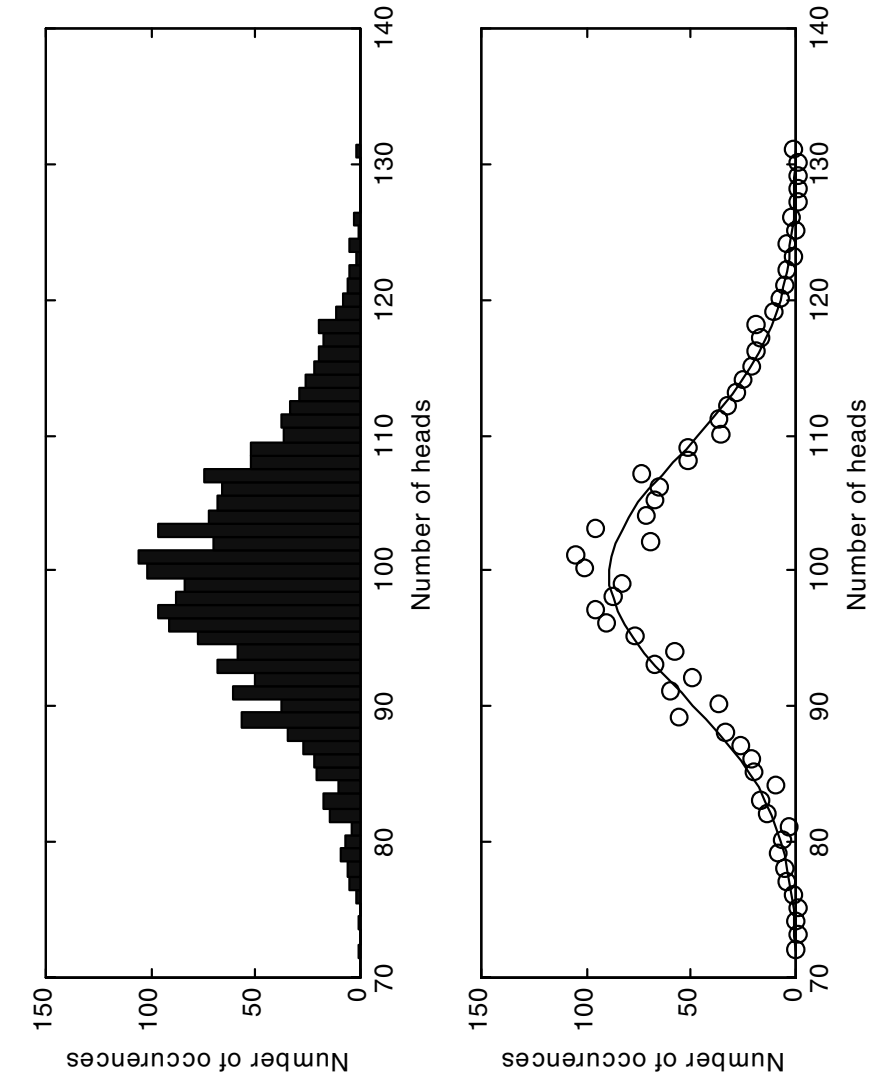


Figure 9.5 Result of coin-tossing experiment.

result is approximately Gaussian as predicted the Laplace approximation [1]. The binomial coefficient is computed using the function `nkchoose` as follows:

```
function out=nkchoose(n,k)
% Computes n!/k!/(n-k)!
a = sum(log(1:n));              % ln of n!
b = sum(log(1:k));              % ln of k!
c = sum(log(1:(n-k)));          % ln of (n-k)!
out = round(exp(a-b-c));        % result
% End of function file.
```

The binomial coefficient is computed in this way in order to illustrate an algorithm that is useful for large values of n. Although MATLAB has large dynamic range, and the technique illustrated in the preceding code is not required for this example, the technique is often useful when other languages are used. ■

9.2.2 Two Simple Monte Carlo Simulations

In this section we consider, for the first time, the Monte Carlo simulation of a communication system. The following assumptions are made:

- There is no pulse shaping performed at the transmitter.
- The channel is assumed AWGN.
- Data symbols at the source output are independent and equally probable.
- There is no filtering within the system and, as a result, there is no intersymbol interference.

As a result of these assumptions both the systems and the accompanying simulations considered in this section are extremely simple. The systems are analytically tractable and the probability of error could be written by inspection by any beginning student of digital communication theory.

Despite the simplicity of the following examples, they are important in that several important observations are made that will be useful in our future work. In addition, the basic structure of a simulation program will be established. We will also see the behavior of Monte Carlo simulations when applied to problems more focused on the subject of our study; namely, digital communication systems. The basic simulation model is illustrated in Figure 9.6. Due to the absence of filtering the delay through the system is zero. Consequently, the delay block (discussed in Chapter 1), used to line up or synchronize corresponding symbols, prior to comparing the transmitted symbol $d[n]$ and the received symbol $\widehat{d}[n]$, is not needed for the simulations considered here. The delay block is shown in Figure 9.6, outlined by dotted lines, to remind us that this important element is required in almost all simulations.

Due to the preceding assumptions, the only source of error is channel noise. We therefore take the approach of defining the direct and quadrature signal components,

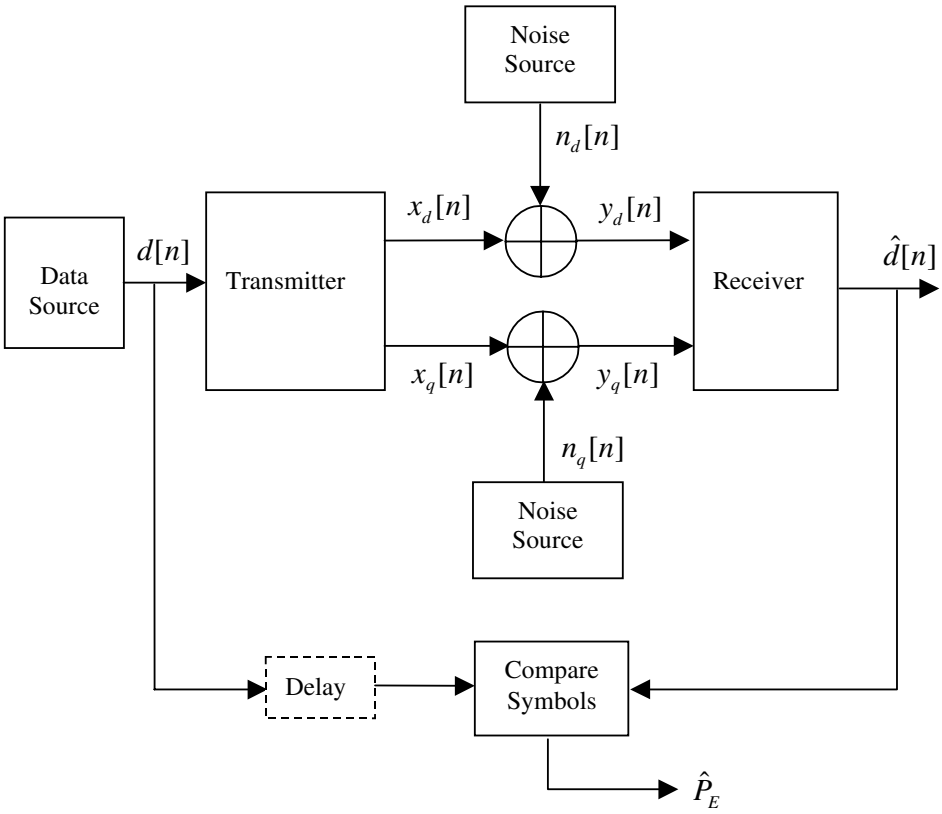


Figure 9.6 Simulation model for simple communication system.

$x_d(t)$ and $x_q(t)$, so that they specify the signal-space components of the signal rather than samples of time-domain waveforms. The advantage of this approach is that signal-space components can be specified using a single sample per transmitted symbol. Processing simulations based upon single samples per symbol execute very rapidly.[3]

Using this approach, the assumed bandpass signal at the output of the modulator is can be expressed

$$x(t,n) = A_c \cos[2\pi f_c t + k_m d[n] + \theta] \tag{9.22}$$

[3]This method can often be applied to spread direct-sequence (DS) spectrum systems. If the processing gain is large, the chip rate is often sufficiently high to justify the assumption that the change in the waveform over a chip interval is negligible. If this is the case a single sample per chip can be made. The example code division multiple access (CDMA) simulation in Chapter 18 is based on this assumption.

where A_c represents the carrier amplitude, k_m is a modulation-dependent constant, $d[n]$ is the n^{th} data symbol ($d[n] = 0$ or 1), and θ is a reference phase. It follows by inspection that the complex envelope of $x(t, n)$ is a function of only the symbol index n and is given by

$$\widetilde{x}[n] = A_c \exp\{k_m d[n] + \theta\} \tag{9.23}$$

For the examples considered here we will assume that $\theta = 0$. Thus, in Figure 9.6

$$x_d[n] = A_c \cos(k_m d[n]) \tag{9.24}$$

and

$$x_q[n] = A_c \sin(k_m d[n]) \tag{9.25}$$

In order to determine and plot the BER as a function of E_b/N_0 for the system illustrated in Figure 9.6, the value of E_b is held constant and the noise power is incremented over the range of interest. This requires calibration of the noise power at the output of the noise generator in Figure 9.6. From Chapter 7, we know that the noise variance is related to the noise power spectral density (PSD) by

$$\sigma_n^2 = \frac{N_0 f_s}{2} \tag{9.26}$$

or

$$N_0 = \sigma_n^2 \frac{2}{f_s} \tag{9.27}$$

The signal-to-noise ratio SNR is defined as E_b/N_0 where f_s is the sampling frequency. Thus

$$SNR = \frac{f_s}{2} \frac{E_b}{\sigma_n^2} \tag{9.28}$$

If the energy E_b and the sampling frequency f_s are both normalized to one, we have

$$\sigma_n = \sqrt{\frac{1}{2} \frac{1}{SNR}} \tag{9.29}$$

This expression is used to establish the noise standard deviation in the simulations that follow.

Example 9.3. (Binary Phase Shift Keying, PSK). In order to generate the direct and quadrature signal space components of a binary PSK signal we let $A_c = 1$ and $k_m = \pi$ in (9.24) and (9.25). This gives

$$x_d[n] = \cos(\pi d[n]) = \begin{cases} 1, & d[n] = 0 \\ -1, & d[n] = 1 \end{cases} \tag{9.30}$$

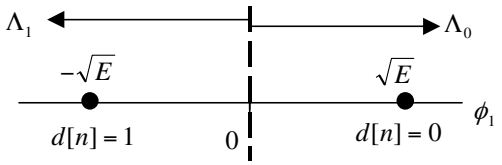


Figure 9.7 Signal space representation of binary PSK.

Also

$$x_q[n] = \sin(\pi d[n]) \tag{9.31}$$

so that the $x_q(t) = 0$ for both $d[n] = 0$ and $d[n] = 1$. This gives the signal space representation of binary PSK that is illustrated in Figure 9.7, in which ϕ_1 is the basis function of the signal space. Since the signal space is one dimensional (generated by a single basis function), only the direct components of the signal and noise need be generated in the simulation. The quadrature path illustrated in Figure 9.6 can be discarded.[4] Figure 9.7 also illustrates the decision regions Λ_0 and Λ_1. If the received signal point falls in the Λ_0 region (region to the right of $\phi_1 = 0$) the receiver makes the decision $\widehat{d}[n] = 0$. If the received signal point falls in the Λ_1 region (region to the left of $\phi_1 = 1$) the receiver makes the decision $\widehat{d}[n] = 1$. The receiver threshold is zero, which is always the case for equally probable, equal energy, signals in an AWGN environment [1]. Thus, the decision rule is

$$\widehat{d}[n] = \begin{cases} 0, & y_d[n] > 0 \\ 1, & y_d[n] < 0 \end{cases} \tag{9.32}$$

These considerations give the following MATLAB simulation program:[5]

```
% File:  c9_MCBPSK.m
snrdB_min = -3; snrdB_max = 8;        % SNR (in dB) limits
snrdB = snrdB_min:1:snrdB_max;
Nsymbols = input('Enter number of symbols > ');
snr = 10.^(snrdB/10);                  % convert from dB
h = waitbar(0,'SNR Iteration');
```

[4]Ignoring the quadrature channel is an example of the theorem of irrelevance, which basically states that, under certain circumstances, a portion of the data present at the receiver input may be discarded without adversely affecting the system performance [3]. For the problem at hand the quadrature channel can be discarded, since it contains only noise (no signal component is present) and the quadrature channel noise is not correlated with the direct channel noise. As an example of the importance of this theorem, recall that white noise has infinite dimensionality. However, in simulating a system operating in a white noise environment, it is necessary to generate (and process) only those noise components that fall within the space defined by the signal.

[5]Note the use of the `waitbar` in this and in other simulations to follow. Since many Monte Carlo simulations take many hours, or even days, to execute, it is good practice to pass information to the simulation user that provides confidence that the simulation is progressing normally. It is also useful, where possible, to provide information that gives insight into the required execution time.

```
len_snr = length(snrdB);
for j=1:len_snr                          % increment SNR
    waitbar(j/len_snr)
    sigma = sqrt(1/(2*snr(j)));          % noise standard deviation
    error_count = 0;
    for k=1:Nsymbols                     % simulation loop begins
        d = round(rand(1));              % data
        x_d = 2*d - 1;                   % transmitter output
        n_d = sigma*randn(1);            % noise
        y_d = x_d + n_d;                 % receiver input
        if y_d > 0                       % test condition
           d_est = 1;                    % conditional data estimate
        else
           d_est = 0;                    % conditional data estimate
        end
        if (d_est ~= d)
           error_count = error_count + 1; % error counter
        end
    end % simulation loop ends
    errors(j) = error_count;             % store error count for plot
end
close(h)
ber_sim = errors/Nsymbols;               % BER estimate
ber_theor = q(sqrt(2*snr));              % theoretical BER
semilogy(snrdB,ber_theor,snrdB,ber_sim,'o')
axis([snrdB_min snrdB_max 0.0001 1])
xlabel('SNR in dB')
ylabel('BER')
legend('Theoretical','Simulation')
% End of script file.
```

Executing this program, with `Nsymbols` = 10000 symbols for each value of SNR, yields the result illustrated in Figure 9.8. Note that the reliability of the BER estimator degrades as the SNR increases due to the fact that fewer errors are counted. This observation suggests that one may wish to relate the number of simulated symbols to the SNR or continue executation of the simulation until the same number of errors are counted at each value of the SNR. ■

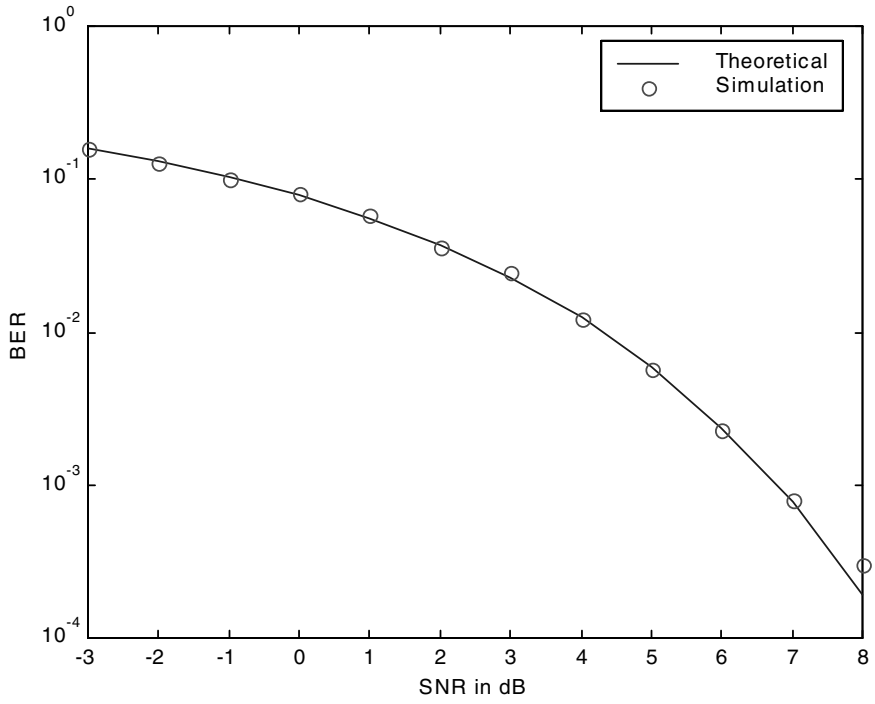


Figure 9.8 Binary phase-shift keying.

Example 9.4. (Binary Frequency-Shift Keying, FSK). In order to generate the direct and quadrature components of a binary FSK signal space we let $k_m = \pi/2$ in (9.24) and (9.25). This gives

$$x_d[n] = \cos\left(\frac{\pi}{2}d[n]\right) = \begin{cases} 1, & d[n] = 0 \\ 0, & d[n] = 1 \end{cases} \tag{9.33}$$

In a similar manner

$$x_q[n] = \sin\left(\frac{\pi}{2}d[n]\right) = \begin{cases} 0, & d[n] = 0 \\ 1, & d[n] = 1 \end{cases} \tag{9.34}$$

This gives the signal space representation of binary PSK is illustrated in Figure 9.9, in which ϕ_1 and ϕ_2 are the basis functions of the signal space. (Recall from Chapter 4 that, for a two-dimensional space, the basis functions can be viewed as defining the direct and quadrature components of the lowpass complex envelope signal.) Since the signal space for binary FSK is two dimensional, both the direct and quadrature components of the signal and noise must be generated in the simulation. Figure 9.9

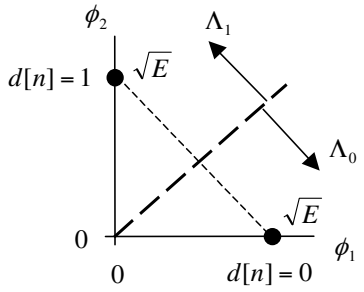


Figure 9.9 Signal-space representation for binary FSK.

also illustrates the decision regions. If the received signal point falls in the Λ_0 region (region below and to the right of the decision boundary), the receiver makes the decision $\widehat{d}[n] = 0$. If the received signal point falls in the Λ_1 region (region above and to the left of the decision boundary), the receiver makes the decision $\widehat{d}[n] = 1$. Note that, for a given point in signal space representing a received signal ($y_d[n]$ and $y_q[n]$ in Figure 9.6) the decision rule is

$$\widehat{d}[n] = \begin{cases} 0, & y_d[n] > y_q[n] \\ 1, & y_d[n] < y_q[n] \end{cases} \tag{9.35}$$

The following MATLAB program implements the simulation:

```
% File:  c9_MCBFSK.m
clear all
snrdB_min = 0; snrdB_max = 10;        % SNR (in dB)limits
snrdB = snrdB_min:1:snrdB_max;
Nsymbols = input('Enter number of symbols > ');
snr = 10.^(snrdB/10);                 % convert from dB
h = waitbar(0,'SNR Iteration');
len_snr = length(snrdB);
for j=1:len_snr                       % increment SNR
    waitbar(j/len_snr)
    sigma = sqrt(1/(2*snr(j)));       % noise standard deviation
    error_count = 0;
    for k=1:Nsymbols                  % simulation loop begins
        d = round(rand(1));           % data
        if d ==0
            x_d = 1;                  % direct transmitter output
            x_q = 0;                  % quadrature transmitter output
        else
            x_d = 0;                  % direct transmitter output
            x_q = 1;                  % quadrature transmitter output
```

```
        end
        n_d = sigma*randn(1);            % direct noise component
        n_q = sigma*randn(1);            % quadrature noise component
        y_d = x_d + n_d;                 % direct receiver input
        y_q = x_q + n_q;                 % quadrature receiver input
        if y_d > y_q                     % test condition
            d_est = 0;                   % conditional data estimate
        else
            d_est = 1;                   % conditional data estimate
        end
        if (d_est ~= d)
            error_count = error_count + 1; % error counter
        end
    end                                  % simulation loop ends
    errors(j) = error_count;             % store error count for plot
end
close(h)
ber_sim = errors/Nsymbols;               % BER estimate
ber_theor = q(sqrt(snr));                % theoretical BER
semilogy(snrdB,ber_theor,snrdB,ber_sim,'o')
axis([snrdB_min snrdB_max 0.0001 1])
xlabel('SNR in dB')
ylabel('BER')
legend('Theoretical','Simulation')
% End of script file.
```

Executing this program, with `Nsymbols` = 10000 symbols for each value of SNR, yields the result illustrated in Figure 9.10. Once again note that the reliability of the estimator degrades as SNR increases due to the fact that fewer errors are counted. Appropriate corrective actions were suggested in the previous example. ■

9.3 Monte Carlo Integration

The subject of Monte Carlo integration arises naturally in our study of communications. Recall that, for an AWGN channel, the sufficient statistic V, formed by sampling the output of an integrate-and-dump detector, is a Gaussian random variable with the mean determined by the data symbol and the variance determined by the channel noise. The conditional probability density functions (pdfs), conditioned on $d[n] = 0$ and $d[n] = 1$, are illustrated in Figure 9.11 where k_T is the receiver threshold. The conditional error probability, conditioned on $d[n] = 1$ is

$$\Pr\left(E|d[n]=1\right) = \int_{k_T}^{\infty} \frac{1}{\sqrt{2\pi}\sigma_n} \exp\left[-\frac{1}{2\sigma_n^2}\left(x-\nu_1(T)\right)^2\right] dx \tag{9.36}$$

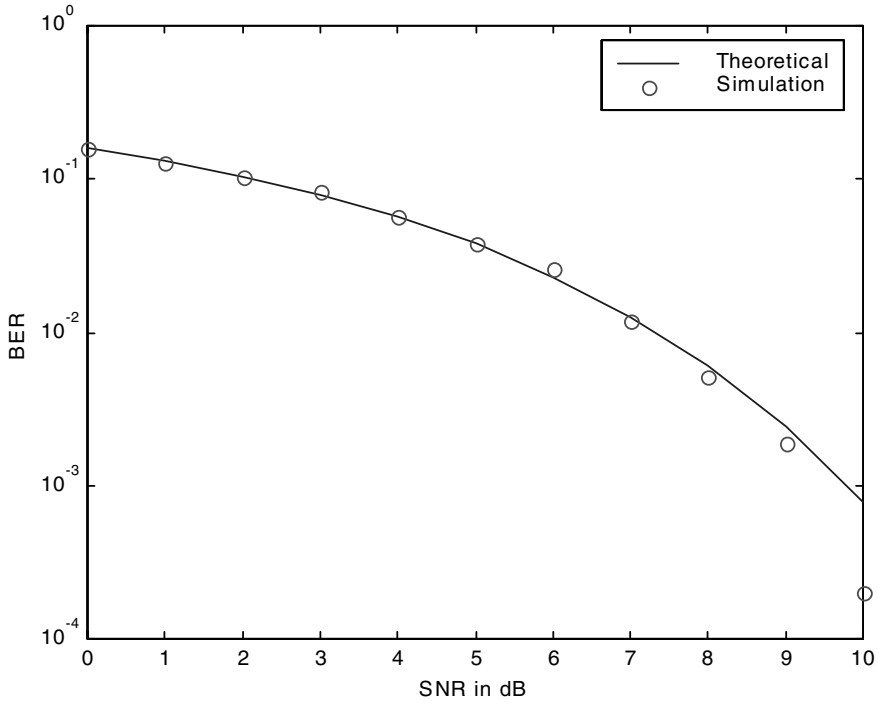


Figure 9.10 Binary frequency-shift keying.

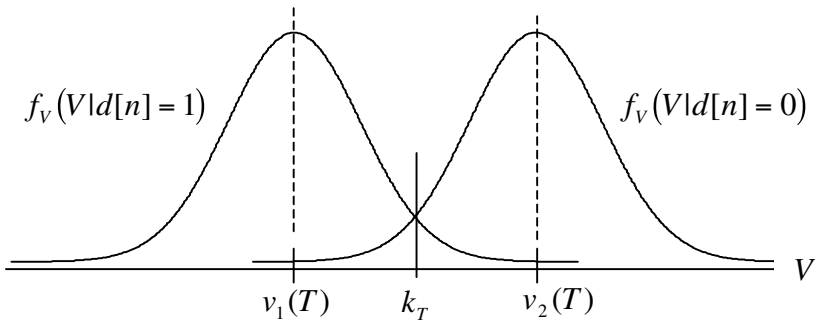


Figure 9.11 Conditional pdfs for binary signaling in Gaussian noise.

A similar expression follows for $\Pr(E|d[n]=0)$. It follows that estimation of the system error probability

$$P_E = \frac{1}{2}\Pr(E|d[n]=1) + \frac{1}{2}\Pr(E|d[n]=0) \tag{9.37}$$

involves estimation of the value of an integral.

The material presented in this section is based on developments by Ross [4], Borse [5], Papoulis [6], and Rubenstein [7]. A brief study of Monte Carlo integration provides additional insight into the Monte Carlo simulation technique. For example, a study of Monte Carlo integration provides a simple context within which to illustrate the convergence properties of a Monte Carlo estimator.

9.3.1 Basic Concepts

Assume that we wish to evaluate the integral

$$I = \int_0^1 g(x)\,dx \tag{9.38}$$

where $g(x)$ is a function bounded on the range of integration. From basic probability theory we know that the expected value (ensemble average) of the function $g(x)$ is given by

$$E\{g(X)\} = \int_{-\infty}^{\infty} g(x) f_X(x)\,dx \tag{9.39}$$

where $f_X(x)$ is the probability density function of the random variable X. If the density function for X satisfies $f_X(x) = 1$ on the interval (0,1) and is zero elsewhere, it follows that $E\{g(X)\} = I$. Thus, if U is a random variable uniformly distributed in the interval (0,1), it follows that

$$I = E\{g(U)\} \tag{9.40}$$

Using relative frequency arguments we can write

$$\lim_{N\to\infty}\left[\frac{1}{N}\sum_{i=0}^{N} g(U_i)\right] = E\{g(U)\} = I \tag{9.41}$$

Thus, we simulate the integrand in order to sample it at N points in the (0,1) interval. The average value of the samples then provides an estimator for the value of the integral. A Monte Carlo simulation of a system does much the same thing. Since we do not usually have a closed-form expression for the sufficient statistic over the error region, samples of the statistic are generated using a simulation of the system.

If we fail to take the limit in (9.41), which will always be the case in practical applications, an approximation results. Denoting this approximation by $\widehat{I}$ yields

$$\frac{1}{N}\sum_{i=0}^{N} g(U_i) = \widehat{I} \tag{9.42}$$

for the Monte Carlo estimator of the integral. In summary, the estimator for the integral is implemented by evaluating the function $g(x)$ at N uniformly distributed random points and averaging. The process can be applied to any proper integral. By applying a simple change of variables, proper integrals having arbitrary limits may be evaluated using Monte Carlo techniques. For example, the integral

$$I = \int_a^b f(x)\,dx \tag{9.43}$$

can be placed in the standard form using the change of variable $y = (x-a)/(b-a)$ to yield

$$I = (b-a)\int_0^1 f[a+(b-a)y]\,dy \tag{9.44}$$

Example 9.5. In order to estimate the value of π using Monte Carlo integration it is necessary to find only a definite integral whose value is a known function of π. An integral that quickly comes to mind is

$$I = \int_0^1 \frac{dx}{1+x^2} = \frac{\pi}{4} \tag{9.45}$$

Thus, we evaluate the integral I using the algorithm defined by (9.42) and multiply the result by 4. Obviously the best that we can do is to use a large but finite value of N. In this case we will obtain not π but rather an approximation to π. Thus, the value of the integral, and consequently the estimated value of π, is a random variable. The results are shown in Figure 9.12 for five estimates of π, with each estimate based on 500 trials. The five estimates were

$$\widehat{\pi} = \begin{bmatrix} 3.1418 & 3.1529 & 3.1517 & 3.1040 & 3.1220 \end{bmatrix} \tag{9.46}$$

If these five results are averaged, we obtain

$$\widehat{\pi} = 3.1345 \tag{9.47}$$

The MATLAB program for estimating π using Monte Carlo integration follows:

```
% File:  c9_example5.m
M=5;                % Number of experiments
N=500;              % Trials per experiment
u = rand(N,M);      % Generate random numbers
uu = 1./(1+u.*u);   % Define function
data = zeros(N,M);  % Initialize array
% The following four lines of code determine
% M estimates as a function of j, 0<j<=N.
data(1,:)  = 4*uu(1,:);
for j=2:N
 data(j,:)=4*sum(uu(1:j,:))/j;
```

```
end
est = data(N,:)     % M estimates of pi
est1 = sum(est)/M   % Average estimate
plot(data,'k')      % Plot results
xlabel('Number of Trials')
ylabel('Estimate of pi')
% End of script file.
```

■

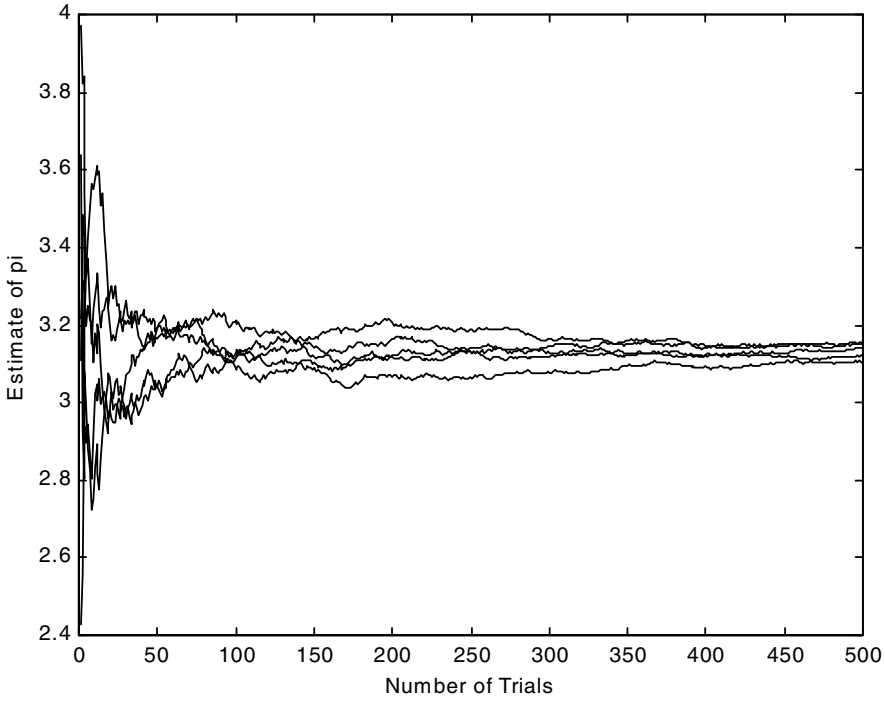


Figure 9.12 Estimation of π using Monte Carlo integration.

9.3.2 Convergence

Assume that the value of an integral, I, is to be estimated and that N random observations or samples, denoted X_i, are available. We form the estimator of I as

$$\widehat{I} = \frac{1}{N}\sum_{i=1}^{N} X_i \tag{9.48}$$

We assume that the N observations, X_i, are *independent and identically distributed* (IID). The arithmetic mean of the samples is given by

$$E\left\{\frac{1}{N}\sum_{i=1}^{N} X_i\right\} = \frac{1}{N}\sum_{i=1}^{N} E\{X_i\} = \frac{NI}{N} = I \tag{9.49}$$

so that the estimate $\widehat{I}$ is unbiased. Since the observations are assumed independent, the sample variance is

$$\sigma_{\widehat{I}}^2 = \frac{1}{N^2}\sum_{i=1}^{N}\sigma_x^2 = \frac{N\sigma_x^2}{N^2} = \frac{\sigma_x^2}{N} \tag{9.50}$$

which shows that the integral estimator is consistent.

Assuming that $X_i = g(U_i)$, the variance of the samples, denoted σ_x^2, is given by [3]

$$\sigma_x^2 = \int_0^1 g^2(u)\,du - \left[\int_0^1 g(u)\,du\right]^2 \tag{9.51}$$

Thus, given the integrand $g(u)$, the required value of N for a given error variance can be determined. Since the estimator is consistent, it follows that accurate estimates of the integral will be obtained if N is sufficiently large. It also follows that accurate estimates of I will be obtained if the samples of $g(u_i)$ have a small variance. Thus, Monte Carlo estimates of an integral will be very accurate, for a given value of N, if $g(u)$ is approximately constant (smooth) over the range of integration. As a matter of fact, it follows that if $g(u)$ is constant over the range of integration, the estimate $\widehat{I}$ is exact for $N = 1$.

9.3.3 Confidence Intervals

The quality of an estimator $\widehat{I}$ is often expressed in terms of the confidence interval, which gives the *probability* $(1-\alpha)$ that estimates fall within a given *range* $(\pm\beta\sigma_{\widehat{I}})$ of values. There are, therefore, two parameters of interest: the probability, which is fixed by α, and the range, which is fixed by β. In equation form, the confidence interval is defined by the expression

$$\Pr\left\{I - \beta\sigma_{\widehat{I}} \le \widehat{I} \le I + \beta\sigma_{\widehat{I}}\right\} = 1 - \alpha \tag{9.52}$$

as shown in Figure 9.13. We refer to the interval $I \pm \beta\sigma_{\widehat{I}}$ as the $1-\alpha$ confidence interval. We now consider the value of the parameter β. Equation (9.52) can be written in terms of the error $\widehat{I} - I$. This gives

$$\Pr\left\{-\beta\sigma_{\widehat{I}} \le \widehat{I} - I \le \beta\sigma_{\widehat{I}}\right\} = 1 - \alpha \tag{9.53}$$

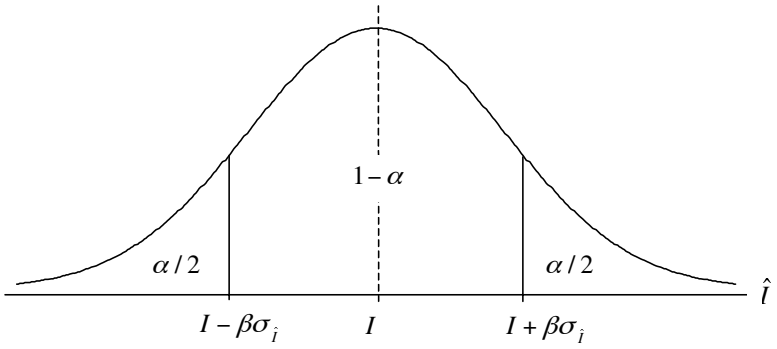


Figure 9.13 Confidence interval.

where $\sigma_{\widehat{I}} = \sigma_x/\sqrt{N}$ with σ_x determined from (9.51). Assuming that the error $\widehat{I} - I$ is a Gaussian random variable, which is a reasonable assumption for large N, the probability density function of $\widehat{I} - I$ is approximated by

$$\frac{1}{\sqrt{2\pi}\sigma_{\widehat{I}}} \exp\left(-\frac{(\widehat{I} - I)^2}{2\sigma_{\widehat{I}}^2}\right)$$

Note that $\widehat{I} - I$ is a zero-mean random variable, since the estimate of the integral is unbiased. It follows that

$$\Pr\left\{\widehat{I} - I \geq \beta\sigma_{\widehat{I}}\right\} = \frac{1}{\sqrt{2\pi}\sigma_{\widehat{I}}} \int_{\beta\sigma_{\widehat{I}}}^{\infty} \exp\left(-\frac{t^2}{2\sigma_{\widehat{I}}^2}\right) dt \tag{9.54}$$

With the change of variable $y = t/\sigma_{\widehat{I}}$ we have

$$\Pr\left\{\widehat{I} - I \geq \beta\sigma_{\widehat{I}}\right\} = \frac{1}{\sqrt{2\pi}} \int_{\beta}^{\infty} \exp(y^2/2)\, dy = Q(\beta) \tag{9.55}$$

where $Q(\cdot)$ represents the Gaussian Q-function.

It follows from Figure 9.13 that

$$\Pr\left\{\widehat{I} - I \geq \beta\sigma_{\widehat{I}}\right\} = Q(\beta) = \frac{\alpha}{2} \tag{9.56}$$

so that

$$\beta = Q^{-1}\left(\frac{\alpha}{2}\right) \tag{9.57}$$

Thus, as shown in Figure 9.13, the probability that the estimate of I falls in the interval $I \pm Q^{-1}(\alpha/2)\,\sigma_x/\sqrt{N}$ is $1 - \alpha$, where σ_x is given by (9.51). The quantities

$\pm Q^{-1}(\alpha/2)\,\sigma_x/\sqrt{N}$ determine the upper and lower confidence bounds. In order to evaluate these quantities σ_x must be determined.

Example 9.6. In order to illustrate the previous concepts, consider the integral

$$I = \int_0^1 \exp\left(-t^2\right)\,dt \tag{9.58}$$

Using numerical integration (e.g., the MATLAB function `quad`) the value of I is

$$I \approx 0.7468 \tag{9.59}$$

In like manner

$$I_2 = \int_0^1 \left[\exp\left(-t^2\right)\right]^2\,dt \approx 0.5981 \tag{9.60}$$

Substitution of (9.59) and (9.60) into (9.51) yields

$$\sigma_x^2 = 0.5981 - (0.7468)^2 = 0.0404 \tag{9.61}$$

Thus, the standard deviation of the estimate of the integral is

$$\sigma_{\widehat{I}} = \frac{\sigma_x}{\sqrt{N}} = \frac{0.2010}{\sqrt{N}} \tag{9.62}$$

and the upper and lower confidence limits are given by

$$I \pm \beta\sigma_{\widehat{I}} = 0.7468 \pm Q\left(\frac{\alpha}{2}\right)\left(\frac{0.2010}{\sqrt{N}}\right) \tag{9.63}$$

The results are illustrated in Figure 9.14. The MATLAB program used to generate the results illustrated in Figure 9.14 follows:

```
Figure:  c9_example6.m
mean = sqrt(pi)*(0.5-q((sqrt(2))));        % result
int2 = sqrt(pi/2)*(0.5-q(2));              % 2nd integral
varx = int2-mean*mean;                     % estimate variance
stdx = sqrt(varx);                         % standard deviation
alpha = 0.1;                               % 90% conf.  level
%
nsum = 0;                                  % initialize nsum
nppseg = 1000;                             % samples per segment
nseg = 100;                                % number of segments
est = zeros(1,nseg);                       % initialize vector
%
for j=1:nseg                               % increment segment
    ui = rand(1,nppseg);                   % uniform samples
    gui = sum(exp(-ui.*ui));               % integrand samples
    nsum = nsum+gui;                       % sum samples
```

```
    est(j) = nsum/(j*nppseg);                % normalize
end                                          % end loop
%
nn = nppseg*(1:nseg);                        % sample index
ub = mean+stdx*qinv(alpha/2)./sqrt(nn);      % upper bound
lb = mean-stdx*qinv(alpha/2)./sqrt(nn);      % lower bound
meanv = mean*ones(1,nseg);                   % exact result
si = 1:nseg;                                 % seg.  index for plot
plot(si,est,'k-',si,meanv,'k--',si,ub,'k:',si,lb,'k:')
xlabel('Number of Segments')                 % x axis label
ylabel('Estimate of Integral')               % y axis label
legend('estimated value','true value',...
    'upperbound','lower bound');
% End of script file.
```

There is, of course, a significant problem with this example. The upper and lower bounds of the confidence interval depend on the exact result, which in this example is known. In general this information will not be known and other approaches will be necessary. A common approach is to approximate the exact value of the integral by an estimated value derived using a long simulation to ensure a reasonable level of accuracy. ■

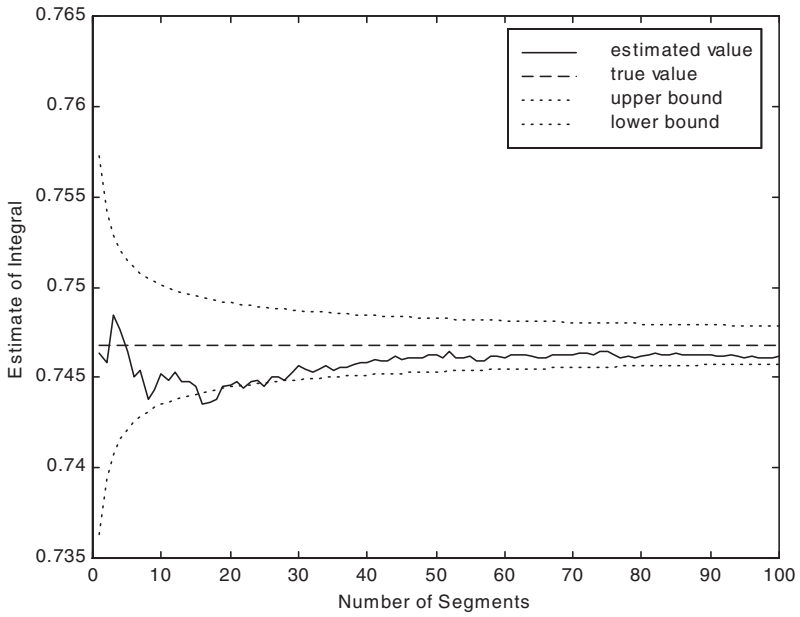


Figure 9.14 Monte Carlo estimate and 90% confidence interval.

9.4 Summary

This chapter addressed the subject of Monte Carlo estimation and simulation using a number of simple examples. We saw that Monte Carlo techniques are based on the performance of stochastic, or random, experiments. An event of interest is identified and the underlying random experiment is replicated a large number of times. The ratio of the number of occurrences of the event of interest to the total number of replications of the random experiment gives the relative frequency of the event of interest. The relative frequency, which is a random variable, is an estimator of the probability of the event of interest. For unbiased and consistant estimators, the relative frequency converges to the probability of the event of interest as the number of replications gets large.

The results presented in this chapter illustrate the very important distinction between stochastic simulation and traditional mathematical analysis. When traditional mathematical analysis is used to determine the value of a parameter, the result is most often a number. For example, analysis of a digital communication system may yield the result $P_E = 1.7638(10^{-3})$ which, of course, is a number. However, the use of Monte Carlo techniques typically provides a result that is a random variable. The properties of this random variable, such as the mean, variance, probability density function, etc., tell us much about the quality of the simulation result.

9.5 Further Reading

A number of books consider the general principles of Monte Carlo simulation. Examples are:

S. M. Ross, *A Course in Simulation*, New York: Macmillan, 1990.

B. D. Ripley, *Stochastic Simulation*, New York: Wiley, 1987.

R. Y. Rubenstein, *Simulation and the Monte Carlo Method*, New York: Wiley, 1981.

The application of Monte Carlo techniques to communcation systems can be found in the following books:

M. C. Jeruchim, P. Balaban, and K. S. Shanmugan, *Simulation of Communication Systems*, 2nd ed., New York: Kluwer Academic/Plenum Publishers, 2000.

F. M. Gardner and J. D. Baker, *Simulation Techniques*, New York: Wiley, 1997.

J. G. Proakis and M. Salehi, *Contemporary Communication Systems Using MATLAB*, Boston: PWS, 1998.

9.6 References

1. R. E. Ziemer and W. H. Tranter, *Principles of Communications: Systems, Modulation and Noise*, 5th ed., New York: Wiley, 2002.

2. P. Beckman, *A History of π (PI)*, New York: Barnes and Noble, 1993.

3. J. M. Wozencraft and I. M. Jacobs, *Principles of Communication Engineering*, New York: Wiley, 1965.

4. S. M. Ross, *A Course in Simulation*, New York: Macmillan, 1990.

5. G. B. Borse, *Numerical Methods with MATLAB*, Boston: PWS Publishing Company, 1997.

6. A. Papoulis, *Probability and Statistics*, Upper Saddle River, NJ: Prentice Hall, 1990.

7. R. Y. Rubenstein, *Simulation and the Monte Carlo Method*, New York: Wiley, 1981.

8. J. W. Craig, "A New, Simple, and Exact Result for Calculating the Probability of Error for Two-Dimensional Signal Constellations," *Proceedings of the 1991 IEEE Milcom Conference*, pp. 571–575.

9.7 Problems

9.1 Repeat Example 9.1 using the same bounding box as was used in Example 9.1, but let $R = 0.5$. Discuss the convergence properties observed in this example and compare with the $R = 1$ result.

9.2 Repeat Example 9.1 using a bounding box centered on the origin and two units on each side. In other words, let $A_{box} = 4$. Define a circle having radius $R = 1$ so that $A_{circle} = \pi$. The circle is also centered on the origin.

9.3 Repeat Example 9.1 using a bounding box centered on the origin and 4 units on each side. In other words, let $A_{box} = 16$, as in the previous problem. Define a circle of radius $R = 1$ so that $A_c = \pi$. Discuss the convergence properties observed with $A_{box} = 16$ and compare with the results of the previous problem for which $A_{box} = 4$.

9.4 Repeat Example 9.1 using a bounding box centered on the origin and 1.6 units on a side. In other words, let $A_{box} = (1.6)^2$. Also let $R = 1$ so that $A_{circle} = \pi$. Compare the rate of convergence with that found in Example 9.1 and in Problem 9.1. Note that, in this case, the area of the bounding box is less than the quantity to be estimated. Is the estimate biased? Why or why not? (This problem hints at a modification of Monte Carlo simulation that can be used to advantage when excessive simulation run times are encountered. This strategy leads to *importance sampling*, an important technique that will be explored in detail in a later chapter.)

9.5 A Woerneroid[6] is defined as the function

$$r = \left(\cos\left(10\pi\left(\frac{\theta}{2\pi}\right)^2\right)\right)^4, \qquad -\pi \le \theta < \pi$$

(a) Using the area sampling technique as depicted in Figure 9.3, develop a Monte Carlo simulation determine the area of the Woerneroid.

(b) Verify the result.

9.6 Repeat the preceding problem for θ defined on the range $0 \le \theta < 2\pi$.

9.7 Prove (9.14) and (9.15).

9.8 Example 9.3 presents a simulation program for a binary PSK communication system. A number of simplifying conditions were assumed. Under these assumptions, the conditional error probability given that $d[n] = 0$ and the conditional error probability given that $d[n] = 1$ are equal, and we may evaluate system performance using a simulation with $d[n] = 0$ all n. Modify the simulation given in Example 9.3 by letting $d[n] = 0$, all n. Compare the results with those given in Example 9.3.

9.9 Repeat the preceding problem assuming FSK modulation as was done in Example 9.4.

9.10 Simulate a binary PSK system assuming that

$$x_d[n] = \cos\left(\frac{\pi}{6}\right), \qquad d[n] = 0$$

and

$$x_q[n] = \sin\left(\frac{\pi}{6}\right), \qquad d[n] = 1$$

Compare the results with those given in Example 9.3.

9.11 Modify the binary PSK simulation given in Example 9.3 so that each BER estimate is based on 20 errors. Why would one wish to do this? How would you use the `waitbar` in this simulation?

9.12 Develop a MATLAB program to estimate the value of $\ln 3$ using Monte Carlo integration. Plot the estimated value as a function of the number of samples, N.

9.13 Use Monte Carlo integration to estimate the value of the integral

$$I = \int_{1.5}^{4} 4e^{-x/2}\,dx$$

Compare the Monte Carlo result with the true value of the integral for $N =$ 100, 500, and 1,000 trials.

[6] Thanks to Brian Woerner of Virginia Tech for this problem.

9.14 Repeat the preceding problem for

$$I = \int_0^1 \sqrt{1 - x^2}\, dx$$

9.15 A certain random variable, X, is known to be Gaussian with mean $m_x = 5$ and variance $\sigma_x^2 = 3$.

(a) Determine the probability $\Pr\{-1 < X < 6\}$. Express this probability in terms of one or more Gaussian Q-functions. Evaluate the result using an appropriate numerical approximation to the Q-function(s).

(b) Write a MATLAB program that uses Monte Carlo integration to estimate the probability found in (a). Plot the estimate of $Q(y)$ as a function of N, the number of trials. What can you say about this estimate?

9.16 The Gaussian Q-function is defined by

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty \exp\left(-y^2/2\right) dy$$

which is not a proper integral, since the support region is infinite. Another definition of the Gaussian Q-function is [8]

$$Q_1(x) = \frac{1}{\pi} \int_0^{\pi/2} \exp\left(-\frac{x^2}{2\sin^2\theta}\right) d\theta$$

This is a proper integral and is therefore better suited to estimation using Monte Carlo techniques than the classical definition.

(a) Use Monte Carlo integration to evaluate $Q_1(3)$ using $N = 500$ sample points and compare to an accurate approximation of $Q(3)$.

(b) Repeat for $N = 100$ and $N = 1{,}000$.

(c) Compare the results.

Chapter 10

MONTE CARLO SIMULATION OF COMMUNICATION SYSTEMS

This chapter extends the basic material on Monte Carlo (MC) techniques explored in the preceding chapter. In Section 10.1 we consider two example simulations of communications systems. The first of these systems, a phase-shift key (PSK) digital communications system, although very simple, serves as a building block for simulations developed later in this book. This is followed by a more complicated simulation of a differential QPSK system in which the effects of phase and symbol synchronization errors are considered. In Section 10.2 we turn our attention to the semianalytic (SA) technique, which combines MC simulation and analysis.

The two methodologies explored in this chapter are quite different. Monte Carlo simulations require very little mathematical analysis and can be applied to any communication system for which the signal-processing algorithm required to represent each functional block in the block diagram of the system is known. Monte Carlo simulation is therefore a very general tool, but is applied at the expense of very long simulation run times, since, as we saw in the preceding chapter, a basic tradeoff ex-

ists between simulation accuracy and the time required to execute the simulation. Semianalytic simulation requires a higher level of analysis, but the payoff is a significantly reduced run time. In addition, execution of an MC simulation yields an estimate of the bit error rate (BER) at a single value of E_b/N_0, while an SA simulation provides a complete curve of BER as a function of E_b/N_0. We will see, however, that SA simulation is not a methodology that can be universally applied, since it is applicable to a restricted class of systems. For most applications, an SA simulation consumes a trivial amount of computer time and, therefore, is the preferred methodology when it can be applied.

10.1 Two Monte Carlo Examples

As we saw in the previous chapter, the Monte Carlo technique, applied to the estimation of the BER of a digital communication system, is implemented by passing N data symbols through a simulation model of the system and counting the number of errors that occur. Assuming that passing N symbols through the simulation model results in N_e errors, the estimate of the BER is

$$\widehat{P}_E = \frac{N_e}{N} \tag{10.1}$$

We learned in the previous chapter that $\widehat{P}_E$ is a random variable, and accurate estimation of the BER requires that the estimator $\widehat{P}_E$ be unbiased and have small variance. Small variance requires that N be large and this in turn results in long computer run times. In the work to follow, the Monte Carlo technique is illustrated by giving two simple examples. Other examples are contained in the remainder of this book.

Example 10.1. (PSK). For our first example consider the basic system illustrated in Figure 10.1. We assume binary PSK modulation with both signal points in the signal constellation lying in the direct (in-phase) channel. (Recall Example 9.3.) With this assumption, we can eliminate the quadrature channel from the simulation. The filter at the output of the modulator, which is assumed to be a third-order Butterworth filter with a bandwidth equal to the bit rate ($BW = r_b$), leads to intersymbol interference (ISI). The purpose of the simulation is to determine the increase in BER resulting from the filter-induced ISI. The program for simulating the system is given in Appendix A. A block-serial approach is used in which blocks of 1,000 symbols are processed iteratively until N total symbols are processed. This was primarily done so that the MATLAB routine `filter`, which is a built-in MATLAB function implementing a time-domain convolution, could be used. As a built-in function it is very efficient and results in a significant reduction in the simulation run time. Note that one must ensure that the filter output is continuous from block to block. This is accomplished by using the initial condition parameter provided in `filter`.

The first problem is to determine the value of `delay`. There are a number of ways in which this can be accomplished. The most elegant way is to crosscorrelate the modulator output and the receiver output, as was done in Chapter 8 in the

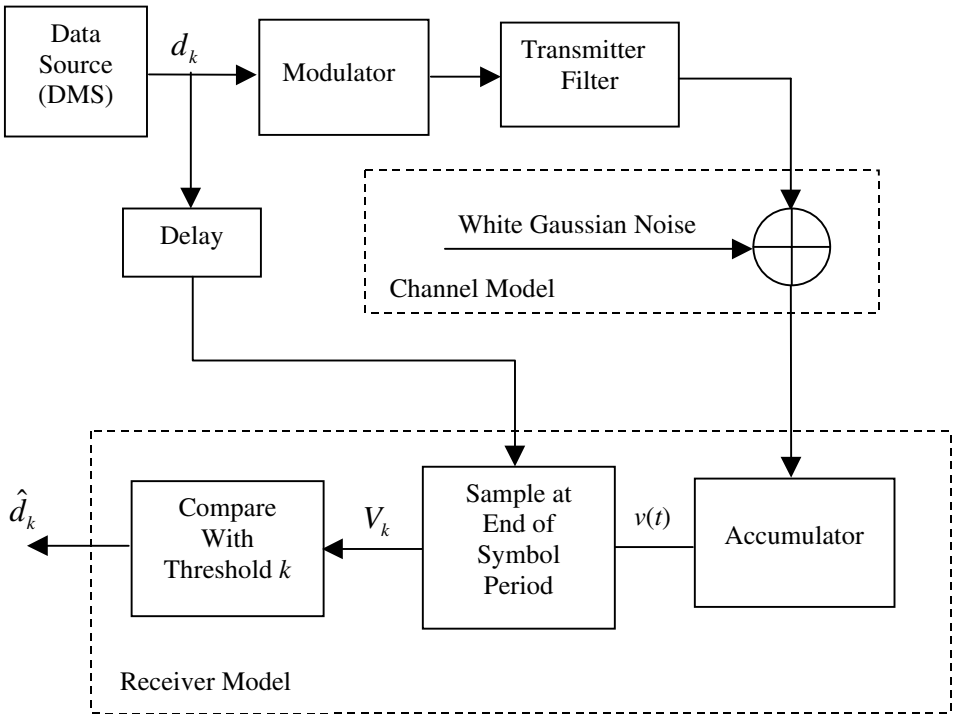


Figure 10.1 Basic communications system.

signal-to-noise ratio estimator. This will be the method used when we consider SA simulation. In order to illustrate the importance of correctly choosing the value of delay, we will use a different technique in this example. Specifically, we will choose a value of E_b/N_0, simulate the system using different values of `delay`, and observe the results. The MATLAB routine for accomplishing this follows:

```
% File:  c10_MCBPSKdelay.m
EbNodB = 6;                                % Eb/No (dB) value
z = 10.^(EbNodB/10);                       % convert to linear scale
delay = 0:8;                               % delay vector
BER = zeros(1,length(delay));              % initialize BER vector
Errors = zeros(1,length(delay));           % initialize Errors vector
BER_T = q(sqrt(2*z))*ones(1,length(delay));% theoretical BER vector
N = round(100./BER_T);      % 100 errors for ideal (zero ISI) system
FilterSwitch = 1;           % set filter switch (in=1 or out=0)
for k=1:length(delay)
    [BER(k),Errors(k)] = c10_MCBPSKrun(N(k),z,delay(k),FilterSwitch)
end
semilogy(delay,BER,'o',delay,BER_T,'-'); grid;
```

```
xlabel('Delay'); ylabel('Bit Error Rate');
% End of script file.
```

Note that the preceding MATLAB script is essentially a combined preprocessor and postprocessor. (The simulation engine is the MATLAB function given in Appendix A.) The assumed value of E_b/N_0 is 6 dB and delay is iterated from 0 to 8 samples. Since the sampling frequency is 10 samples per symbol, the step size of `delay` is $0.1T_s$, where T_s is the symbol duration. The value of N is chosen so that a sufficient number of errors occur to ensure that the estimator variance is suitably small. In this case, we set N to $100/P_T$, where P_T is the theoretical error probability for the additive, white, Gaussian noise (AWGN) case. The presence of ISI and other disturbances will of course increase the number of errors that occur for a given E_b/N_0 over the average value of 100. Note that for each value of `delay`, both the value of the BER and the number of errors used to compute the BER are displayed. This allows the assumption of "a sufficient number of errors to form a reliable BER estimate" to be verified.

Executing the simulation yields the result illustrated in Figure 10.2. The various simulation results are indicated by the small circles, and the performance of the ideal

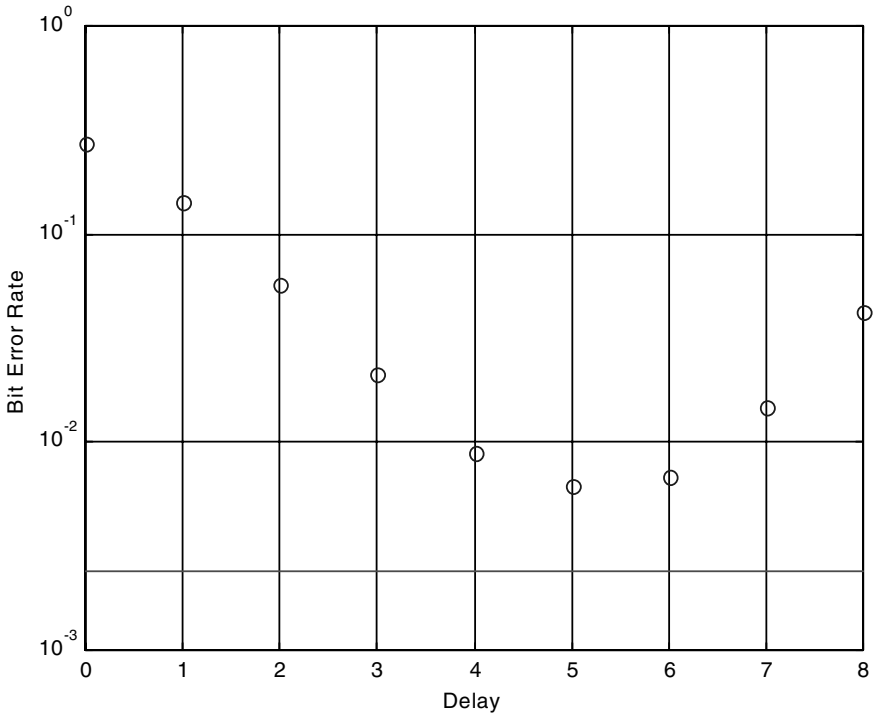


Figure 10.2 Preliminary simulation used to determine delay.

(zero ISI) system operating in an AWGN environment with $E_b/N_0 = 6$ dB is given by the solid line for reference. An incorrectly chosen value of `delay` clearly results in a value of BER that is too large. Since the value of the BER is a minimum at a delay of 5 samples, one might assume that 5 samples is the appropriate value of `delay`. However, since the delay is quantized to an integer number of sample periods, the value of 5 may not be precisely correct. Observation of Figure 10.2 implies that the corect delay value is most likely between 5 and 6 sampling periods. A more precise estimate of the correct value of `delay` can be determined by executing the simulation again with a higher sampling frequency (smaller sampling periods). (See Problem 10.3.) It should also be remembered that the estimator defined in (10.1) is a random variable and, as a result, any given value of BER may be too high or too low. The transmitter filter causes ISI, and the effect of ISI will prevent the BER from achieving the zero-ISI limit for any value of `delay`. If the transmitter filter is removed (`FilterSwitch=0`), the zero-ISI limit can be achieved.

Now that we know the appropriate value of `delay`, we can execute the simulation and determine the value of $\widehat{P}_E$ as a function of E_b/N_0. The MATLAB script follows:

```
% File:  c10_MCBPSKber.m
EbNodB = 0:8;                % vector of Eb/No (dB) values
z = 10.^(EbNodB/10);         % convert to linear scale
delay = 5;                   % enter delay value (samples)
BER = zeros(1,length(z));    % initialize BER vector
Errors = zeros(1,length(z)); % initialize Errors vector
BER_T = q(sqrt(2*z));        % theoretical (AWGN) BER vector
N = round(20./BER_T);        % 20 errors for ideal (zero ISI) system
FilterSwitch = 1;            % Tx filter out=0 or in=1
for k=1:length(z)
     N(k) = max(1000,N(k));  % ensure at least one block processed
    [BER(k),Errors(k)] = c10_MCBPSKrun(N(k),z(k),delay,FilterSwitch)
end
semilogy(EbNodB,BER,'o',EbNodB,BER_T)
xlabel('E_b/N_o - dB'); ylabel('Bit Error Rate'); grid
legend('System Under Study','AWGN Reference',0)
% End of script file.
```

Note that E_b/N_0 is stepped in 1 dB steps from 0 dB to 8 dB.

When executing a Monte Carlo simulation over a range of E_b/N_0 values, using the same value of N for each value of E_b/N_0 will result in an estimated value of BER that is based on a decreasing number of observed errors as E_b/N_0 increases. As a result, the BER estimate at large values of E_b/N_0 will be less reliable than the BER estimate at smaller values of E_b/N_0. This problem can partially be overcome by setting N, the number of samples processed, to K/P_T, where P_T is the error probability for the AWGN case. Since system impairments such as ISI and synchronization errors will result in simulated values of $\widehat{P}_E$ that exceed P_T, one will typically observe more than K errors in a given simulation run. The preceding MATLAB code uses $K = 20$. If N is determined in this fashion, values of N less

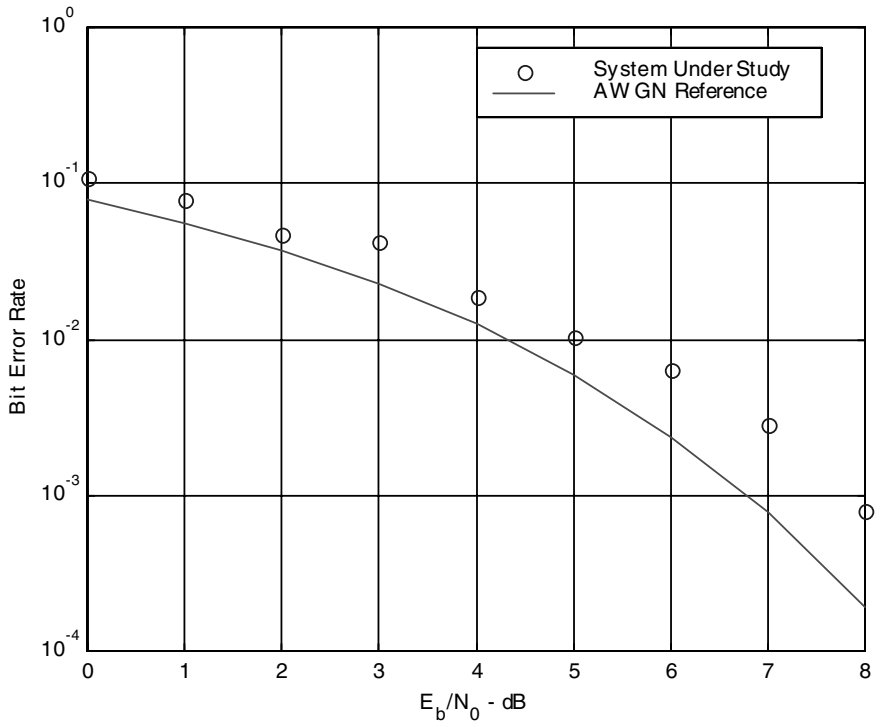


Figure 10.3 Binary PSK simulation for system with ISI.

than 1,000 are possible for sufficiently small values of E_b/N_0. Since the simulation is based on the processing of sequential blocks of samples, with a block size of 1,000 symbols (10,000 samples), we must ensure that $N > 1,000$ so that at least one complete block is processed by the simulation. If $N < 1,000$, incorrect results will occur.

Executing the simulation provides the results illustrated in Figure 10.3. The simulation results are given by the small circles, and the BER of the ideal (zero ISI) results are given by the solid line. The increased BER resulting from the ISI caused by the filter is evident.

The block-serial technique used in this example will be used again in Chapter 18 when we examine a simulation of a simplified CDMA system. This simulation, although simple, will serve as a building block for simulations presented later in this book. ■

Example 10.2. (QPSK). The previous example on BPSK modulation made a number of simplifying assumptions to keep the simulation code compact and the analysis tractable. This example of a QPSK system models some new sources of error, and includes some new parameters that may make it easier to relate the sim-

ulation results to those obtained from a physical communications system. Note for example that the channel attenuation, accounting for the propagation loss between transmitter and receiver, is included as a simulation parameter. Real (nonscaled) values for the symbol rate and the sampling frequency are also included in this simulation. The block diagram for this system is shown in Figure 10.4, and the code is given in Appendix B. Note that the transmitter filter, although illustrated in Figure 10.4, is not used in the simulations presented here, in order to reduce simulation execution time. Provision for including the transmitter filter is included in the simulation code given in Appendix B.

In coherent radio frequency (RF) systems, the receiver must provide carrier and symbol synchronization capabilities. Noise and distortion in the channel will make it impossible for the carrier and symbol synchronizers to operate perfectly. Incorrect carrier synchronization will result in a phase error, or phase rotation, of the received signal relative to the transmitted signal. The simulation provided in this example allows the user to simulate this phase error as a stochastic process. Symbol synchronization errors will result in the integrate-and-dump detector processing the received signal over the incorrect time interval. The simulation in this example also allows the user to examine the errors resulting from this effect.

As we know from basic communication theory, QPSK systems suffer from a problem called phase ambiguity. Since the channel introduces an unknown time delay to the signal, it will be impossible for the receiver to determine the absolute phase of the transmitted signal. For example, in a QPSK system, a transmitter may send the phase sequence 45°, 135°, 45°, and -45°. Suppose the channel introduces a time delay equal to 100.75 cycles of the RF carrier. The receiver will now mistakenly detect the original 45 degree signal to be -45 degrees, and make similar errors on the remaining symbols to produce the received sequence of -45°, -135°, 135°. If the information bits are contained in the absolute phase of the transmitted signal, the receiver will make a large number of errors. The solution to this problem involves encoding the information not in the absolute phase, but in the phase difference between symbols. For example, if the transmitter phase increases 90°, from 45° to 135° between the first and second symbols, the receiver will detect these two signals as -135° and 45°, which still shows a phase increase of 90°. Differential encoding is implemented in the MATLAB code given in Appendix B, which is the main simulation code for the QPSK system. All simulations presented in this example use repeated calls to this code.

As in the previous example, the time delay through the system must be determined. The following MATLAB program determines the optimal time delay to be used at the receiver to account for the signal propagation delay through the system:

```
% File:  c10_MCQPSKdelay.m
Eb = 23; No = -50;              % Eb (dBm) and No (dBm/Hz)
ChannelAttenuation = 70;        % channel attenuation in dB
N = 1000;
delay = -0.1:0.1:0.5;
EbNo = 10.^(((Eb-ChannelAttenuation)-No)/10);
BER_MC = zeros(size(delay));
```

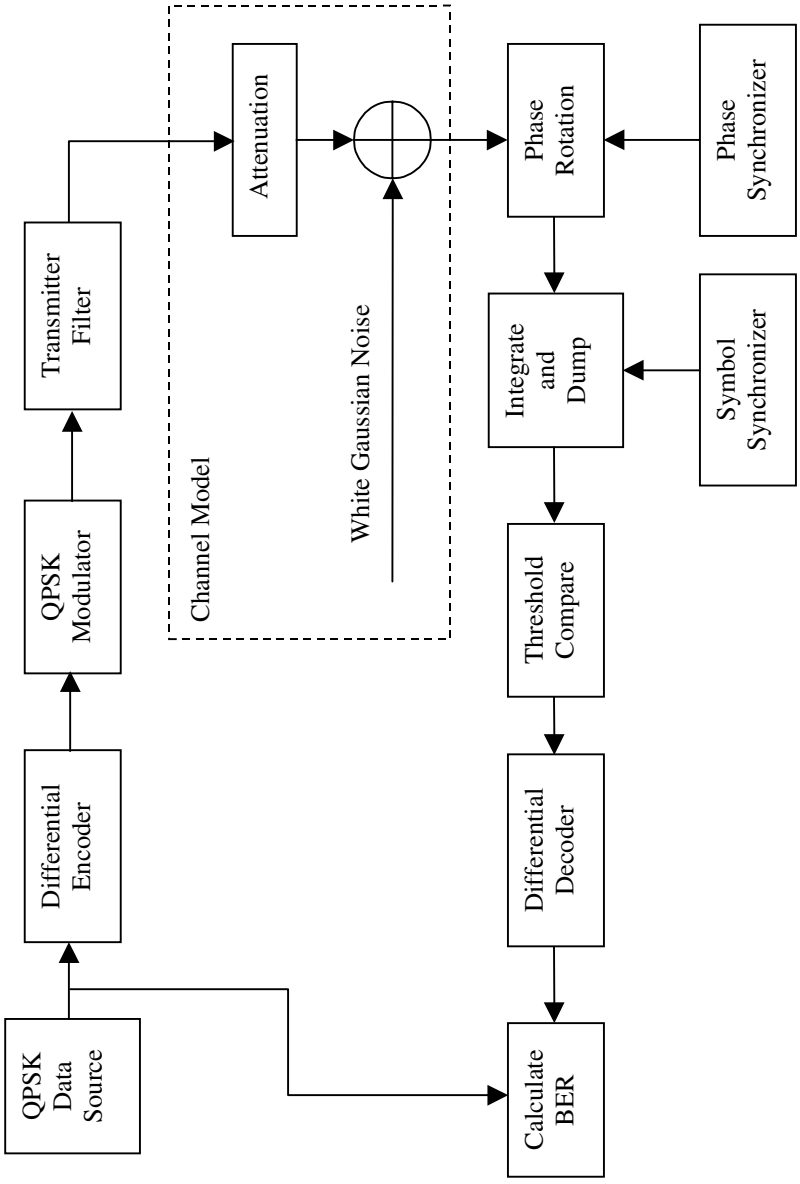


Figure 10.4 QPSK communications system.

```
for k=1:length(delay)
     BER_MC(k) = c10_MCQPSKrun(N,Eb,No,ChannelAttenuation,...
        delay(k),0,0,0);
     disp(['Simulation ',num2str(k*100/length(delay)),'% Complete']);
end
BER_T = 0.5*erfc(sqrt(EbNo))*ones(size(delay));   % Theoretical BER
semilogy(delay,BER_MC,'o',delay,2*BER_T,'-')      % Plot BER vs Delay
xlabel('Delay (symbols)'); ylabel('Bit Error Rate');
legend('MC BER Estimate','Theoretical BER'); grid;
% End of script file.
```

Since no channel filter was used, the optimal delay is zero symbols, as shown in Figure 10.5. Note that we have measured delay with respect to the symbol period rather than in samples, as was done in the previous example.

Now that we know the delay, we will measure the sensitivity of the BER to static synchronization phase error. The phase error is measured from 0 to 90 degrees in 10-degree increments. The code for accomplishing this follows:

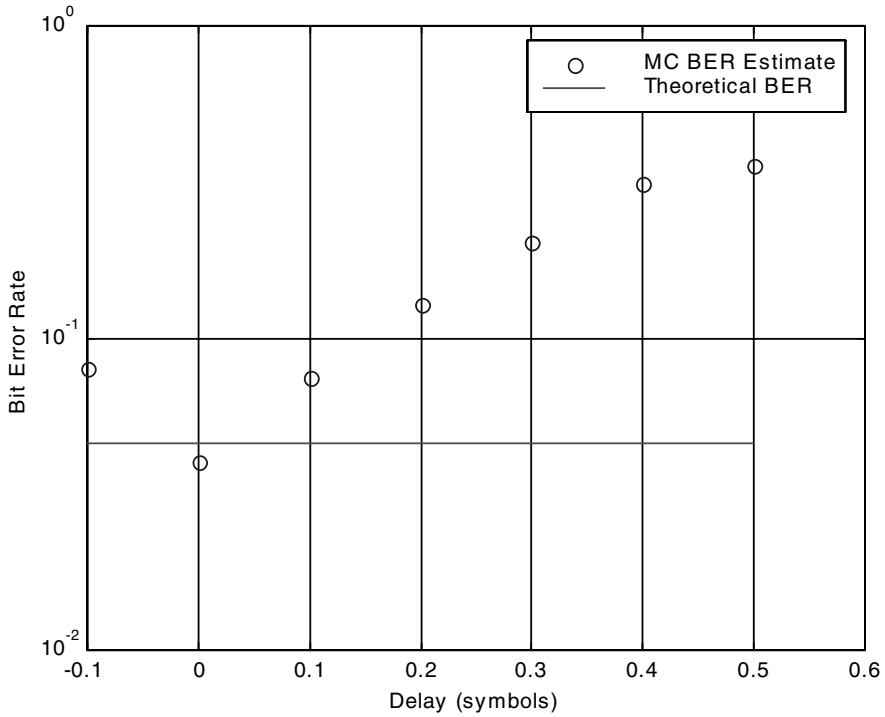


Figure 10.5 Preliminary simulation used to determine delay.

```
% File:  c10_MCQPSKphasesync.m
PhaseError = 0:10:90;            % Phase Error at Receiver
Eb = 24; No = -50;               % Eb (dBm) and No (dBm/Hz)
ChannelAttenuation = 70;         % dB
EbNo = 10.^((Eb-ChannelAttenuation-No)/10);
BER_T = 0.5*erfc(sqrt(EbNo)*ones(size(PhaseError)));
N = round(100./BER_T);
BER_MC = zeros(size(PhaseError));
for k=1:length(PhaseError)
     BER_MC(k) = c10_MCQPSKrun(N(k),Eb,No,ChannelAttenuation,0,0,...
         PhaseError(k),0);
     disp(['Simulation ',num2str(k*100/length(PhaseError)),'...
        % Complete']);
end
semilogy(PhaseError,BER_MC,'o',PhaseError,2*BER_T,'-')
xlabel('Phase Error (Degrees)');
ylabel('Bit Error Rate');
legend('MC BER Estimate','Theoretical BER'); grid;
% End of script file.
```

Executing the simulation yields the result illustrated in Figure 10.6. Figure 10.6 shows that the BER, as determined by the simulation, reaches a maximum at a phase error of 45 degrees and decreases back to the optimal value (value for zero synchronization phase error) for a phase error of 0 or 90 degrees. This behavior is due to the differential encoder.

Now that we know the optimal phase shift and time delay for the channel, we can measure the BER as a function of the signal-to-noise ratio (SNR). The MATLAB code for accomplishing this follows:

```
% File:  c10_MCQPSKber.m
Eb = 22:0.5:26; No = -50;            % Eb (dBm) and No (dBm/Hz)
ChannelAttenuation = 70;             % Channel attenuation in dB
EbNodB = (Eb-ChannelAttenuation)-No; % Eb/No in dB
EbNo = 10.^(EbNodB./10);             % Eb/No in linear units
BER_T = 0.5*erfc(sqrt(EbNo));        % BER (theoretical)
N = round(100./BER_T);               % Symbols to transmit
BER_MC = zeros(size(Eb));            % Initialize BER vector
for k=1:length(Eb)                   % Main Loop
     BER_MC(k) = c10_MCQPSKrun(N(k),Eb(k),No,ChannelAttenuation,...
        0,0,0,0);
     disp(['Simulation ',num2str(k*100/length(Eb)),'% Complete']);
end
semilogy(EbNodB,BER_MC,'o',EbNodB,2*BER_T,'-')
xlabel('Eb/No (dB)'); ylabel('Bit Error Rate');
legend('MC BER Estimate','Theoretical BER'); grid;
% End of script file.
```

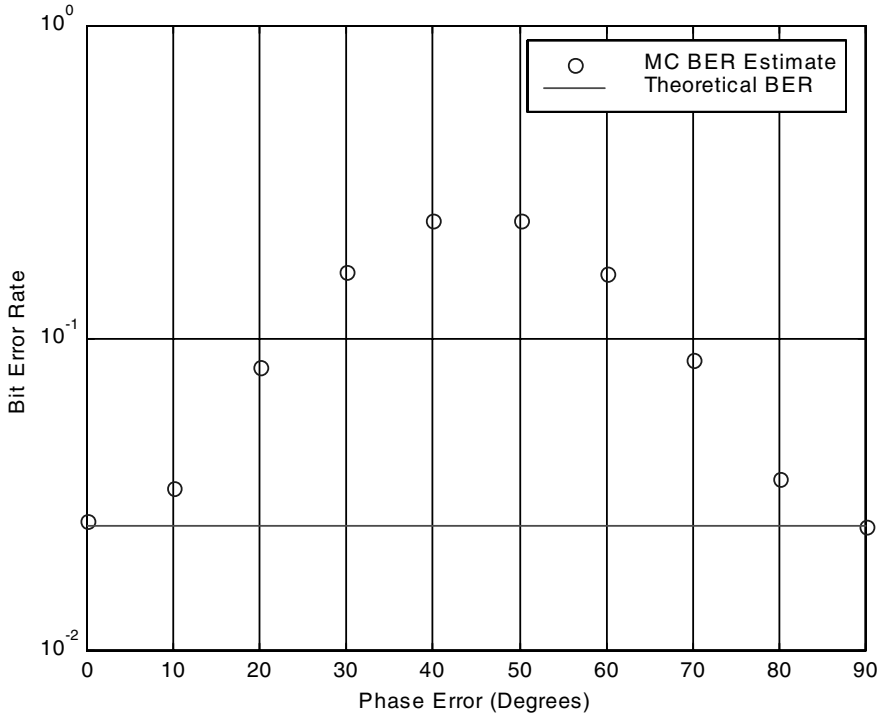


Figure 10.6 Sensitivity of BER to static phase errors.

Executing the simulation yields the plot shown in Figure 10.7. We see that the simulated result is very close to the theoretical AWGN result. This provides at least a partial sanity check on the simulation.

Next we examine the impact of phase jitter on the system BER. The phase error process is modeled as white Gaussian noise. The MATLAB code for the simulation follows:

```
% File:  c10_MCQPSKPhaseJitter.m
PhaseBias = 0; PhaseJitter = 0:2:30;
Eb = 24; No = -50;                    % Eb (dBm) and No (dBm/Hz)
ChannelAttenuation = 70;              % dB
EbNo = 10.^((Eb-ChannelAttenuation-No)/10);
BER_T = 0.5*erfc(sqrt(EbNo)*ones(size(PhaseJitter)));
N=round(100./BER_T);
BER_MC = zeros(size(PhaseJitter));
for k=1:length(PhaseJitter)
     BER_MC(k) = c10_MCQPSKrun(N(k),Eb,No,ChannelAttenuation,0,0,...
         PhaseBias,PhaseJitter(k));
```

```
    disp(['Simulation ',num2str(k*100/length(PhaseJitter)),'...
       % Complete']);
end
semilogy(PhaseJitter,BER_MC,'o',PhaseJitter,2*BER_T,'-')
xlabel('Phase Error Std.  Dev.  (Degrees)');
ylabel('Bit Error Rate');
legend('MC BER Estimate','Theoretical BER'); grid;
% End of script file.
```

Executing the simulation yields the result illustrated in Figure 10.8. As expected, the BER increases as the standard deviation of the phase jitter increases. In many system simulations it is not appropriate to model phase jitter as a white-noise process. Should this be the case, a finite impulse response (FIR) filter can be designed to realize the required power spectral density (PSD) of the phase jitter process.

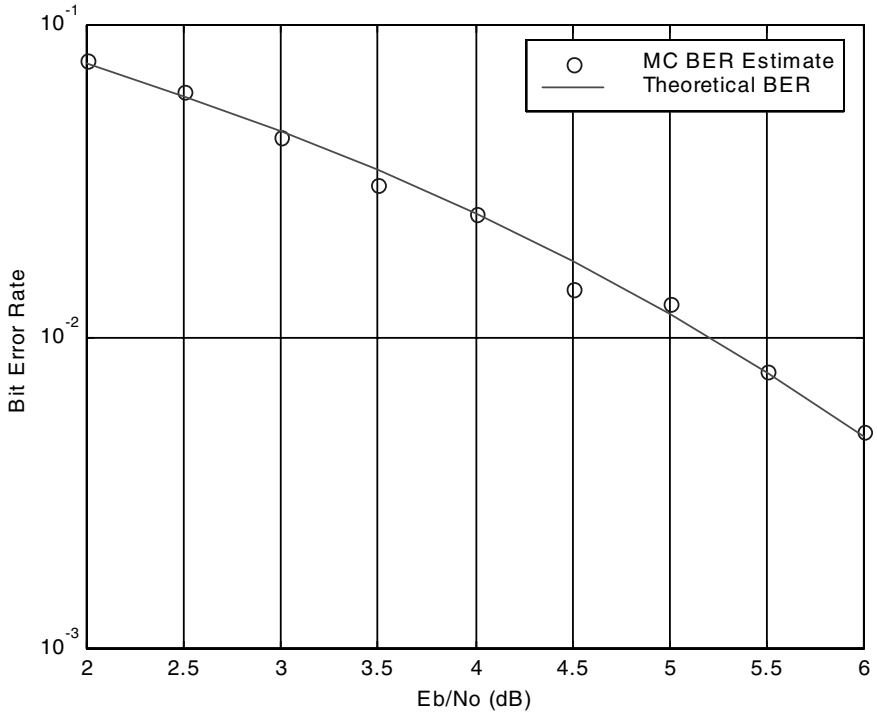


Figure 10.7 Simulation and theoretical results for a QPSK system operating in a AWGN environment.

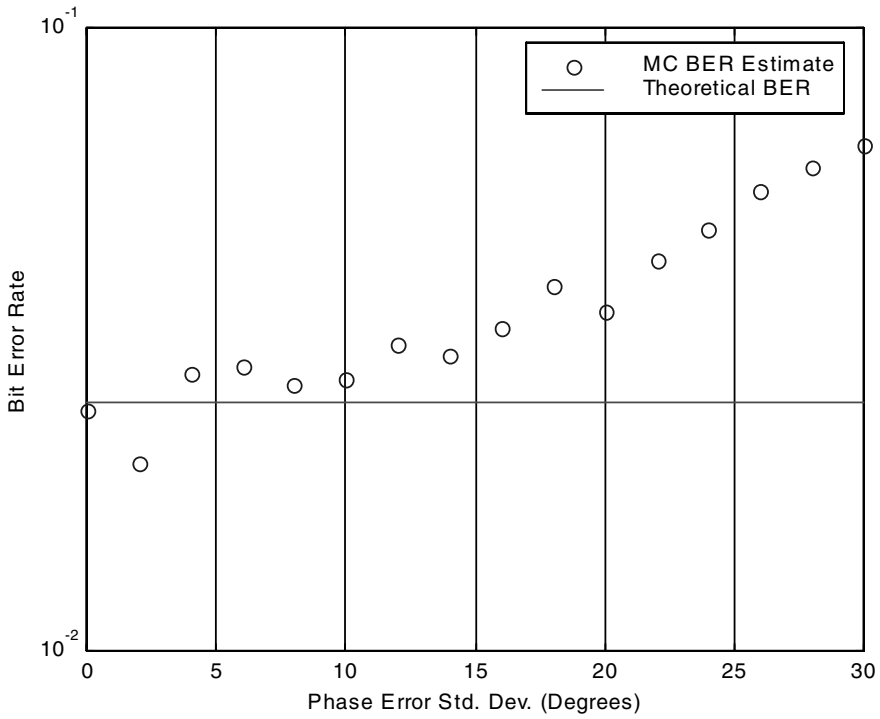


Figure 10.8 Simulation illustrating the sensitivity of QPSK to phase jitter.

The final simulation in this sequence examines the sensitivity of BER to symbol timing error. The MATLAB code follows:

```
% File:  c10_MCQPSKSymJitter.m
SymJitter = 0:0.02:0.2;
Eb = 24; No = -50;              % Eb (dBm) and No (dBm/Hz)
ChannelAttenuation = 70;        % channel attenuation in dB
EbNo = 10.^((Eb-ChannelAttenuation-No)/10);
BER_T = 0.5*erfc(sqrt(EbNo)*ones(size(SymJitter)));
N=round(100./BER_T);
BER_MC = zeros(size(SymJitter));
for k=1:length(SymJitter)
     BER_MC(k) = c10_MCQPSKrun(N(k),Eb,...
         No,ChannelAttenuation,0,SymJitter(k),0,0);
     disp(['Simulation ',num2str(k*100/length(SymJitter)),...
        '% Complete']);
end
semilogy(SymJitter,BER_MC,'o',SymJitter,2*BER_T,'-')
```

```
xlabel('Symbol Timing Error Std.  Dev.  (Symbols)');
ylabel('Bit Error Rate');
legend('MC BER Estimate','Theoretical BER'); grid;
% End of script file.
```

The result of executing the simulation is shown in Figure 10.9. Just as in the phase jitter case, the symbol synchronization error is modeled as a white Gaussian stochastic process. Once again, if memory effects in the symbol jitter process must be accurately modeled, an FIR filter can be designed to realize the required PSD.

In addition, in this simulation the transmitted symbols are crosscorrelated (note the use of the function `vxcorr`) in order to ensure that the transmitted and received symbols are properly aligned so that the BER is correctly determined. In future simulations, including Examples 10.3 and 10.4 to follow, the crosscorrelation technique will be used to calculate the appropriate value of delay and a separate simulation to determine this parameter will not be required. A separate simulation program was used here to illustrate the sensitivity of the simulation results to this important parameter. This simulation will be encountered again in Chapter 16 when we consider importance sampling. ■

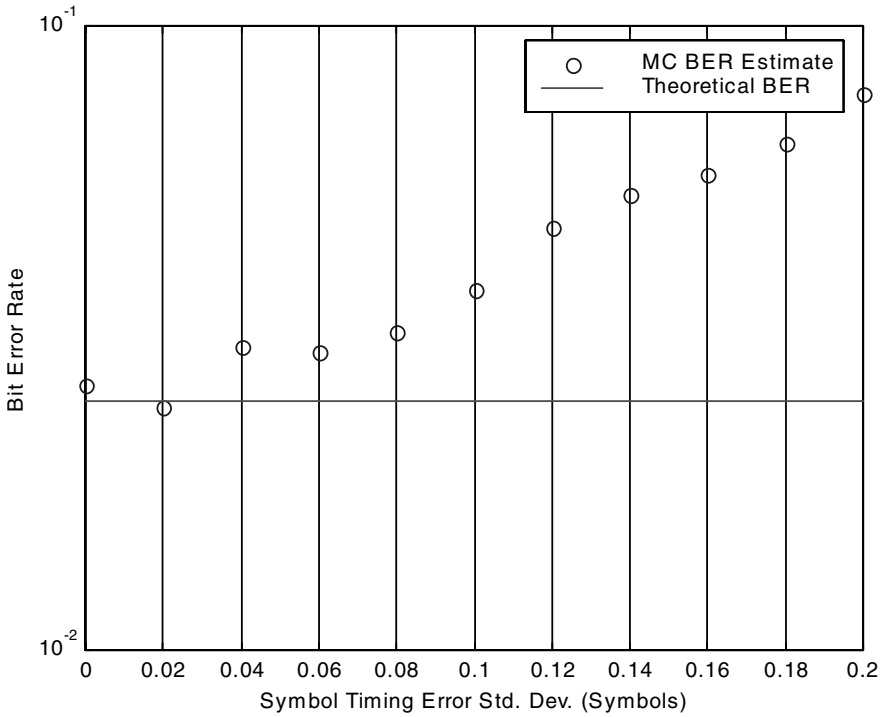


Figure 10.9 Simulation result illustrating the impact of symbol jitter.

10.2 Semianalytic Techniques

As we have seen, the Monte Carlo simulation method is completely general and may be applied to any system for which simulation models of the various system building blocks can be defined, or at least approximated, in terms of a numerical (digital signal processing, DSP) algorithm. No analytical knowledge, outside of that required to implement the subsystem models, is required. The price paid for using Monte Carlo methods is the run time required for executing the simulation. If the system and the channel model are complicated, and the BER is low, the required run time is sometimes so long that the use of Monte Carlo techniques becomes impractical for all but the most important simulations.

In the work to follow we stress the estimation of the BER, since the BER is the most common measure used to evaluate the performance of digital communication systems. In executing a simulation to estimate the BER, information is collected that allows other items of interest to be determined. These include waveforms, eye diagrams, signal constellations, and PSD estimates at various points in the system of interest.

Fortunately there are alternatives to the pure Monte Carlo method. One of the most powerful of these alternatives is the semianalytic method, in which analytical and simulation techniques are used together in a way that yields very rapid estimation of the BER. The SA simulation method, like all rapid simulation techniques, allows analytical knowledge to be traded off against simulation run time.

The block diagram of a simple system, to which the SA simulation technique is applicable, is illustrated in Figure 10.10. The k^{th} transmitted symbol is denoted

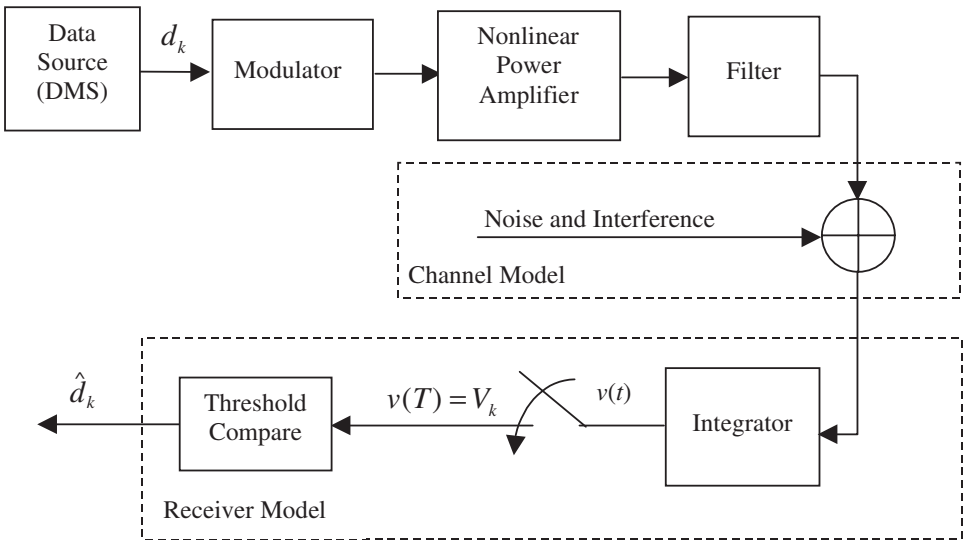


Figure 10.10 Example system to illustrate the SA simulation technique.

d_k, and the corresponding symbol at the receiver output is $\widehat{d}_k$. The transmitted symbol is received correctly if $\widehat{d}_k = d_k$, and an error is made if $\widehat{d}_k \neq d_k$. As we know from basic communication theory, the quantity V_k is the decision statistic for the k^{th} transmitted symbol, and the receiver makes a decision by comparing the value of V_k with a threshold T.

The decision statistic V_k, shown in Figure 10.10, is a function of three components:

$$V_k = f(S_k, D_k, N_k) \tag{10.2}$$

where S_k is the component of V_k due to the transmitted signal; D_k results from system-induced distortion such as ISI due to filtering or multipath, and N_k is due to the channel disturbances such as noise and interference. In applying semianalytic simulation, the combined effects of S_k and D_k are determined by an MC simulation, and the effects of noise, represented by N_k, are treated analytically. The SA simulation technique is applicable to any system in which the probability density function of the noise component of V_k can be analytically determined. A simple case is for the additive Gaussian-noise channel in which the system is linear from the point where noise is injected to the point at which the decision statistic V_k is defined. This follows since any linear transformation of a Gaussian random process yields a Gaussian process. Referring to Figure 10.10, we see that if the channel noise is Gaussian, the decision statistic V_k is a Gaussian random variable with the mean determined by S_k and D_k. Thus, semianalytic simulation is a combination of MC simulation and analysis.

10.2.1 Basic Considerations

As an example, consider a binary PSK (BPSK) system operating an AWGN (additive, white, Gaussian noise) environment. For the moment we neglect the transmitter filter and assume full-response signaling.[1] The pdf of V_k, conditioned on transmission of a binary 1 or binary 0, is Gaussian, as illustrated in Figure 10.11. The probability density functions (pdfs) of the decision statistic, V_k, conditioned on $d_k = 0$ and $d_k = 1$, are given by

$$f_V(v|d_k = 0) = \frac{1}{\sqrt{2\pi}\sigma_v} \exp\left[-\frac{(v - v_1)^2}{2\sigma_v^2}\right] \tag{10.3}$$

and

$$f_V(v|d_k = 1) = \frac{1}{\sqrt{2\pi}\sigma_v} \exp\left[-\frac{(v - v_2)^2}{2\sigma_v^2}\right] \tag{10.4}$$

where v_1 and v_2 are the means of the random variable V_k conditioned on $d_k = 0$ and $d_k = 1$, respectively. The probability of error conditioned on $d_k = 0$ is

$$\Pr\{Error|d_k = 0\} = \int_T^\infty f_V(v|d_k = 0)dv \tag{10.5}$$

[1] Recall that a full-response system is one in which the signal energy at the receiver is constrained to the symbol interval so that the matched-filter receiver, which integrates the received signal over a symbol period, captures all of the transmitted symbol energy.

where T is the decision threshold, and the probability of error conditioned on $d_k = 1$ is

$$\Pr\{Error|d_k = 1\} = \int_{-\infty}^{T} f_V(v|d_k = 1)dv \tag{10.6}$$

If the binary symbols $d_k = 0$ and $d_k = 1$ are transmitted with equal probability, the optimum threshold T is the point at which the two conditional pdfs are equal. For this case the two conditional error probabilities are equal and the overall error probability is

$$P_E = \frac{1}{2}\Pr\{Error|d_k = 0\} + \frac{1}{2}\Pr\{Error|d_k = 1\} \tag{10.7}$$

which is

$$P_E = \Pr\{Error|d_k = 0\} = \int_{T}^{\infty} f_V(v|d_k = 0)dv \tag{10.8}$$

The probability for an AWGN environment is usually expressed in terms of the Gaussian Q function. This gives

$$P_E = Q\left(\frac{v_1}{\sigma_v}\right) \tag{10.9}$$

where

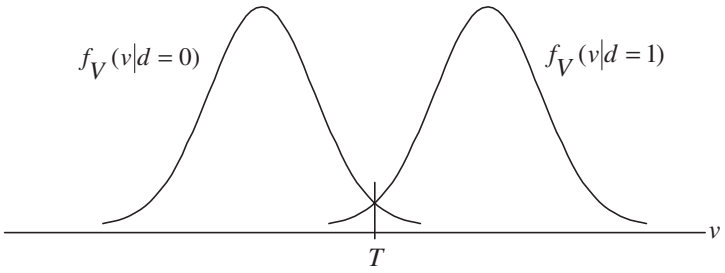
$$Q(x) = \frac{1}{\sqrt{2\pi}}\int_{x}^{\infty} \exp\left(-\frac{t^2}{2}\right) dt \tag{10.10}$$

defines the Gaussian Q function. Note that with equal energy signals the threshold T is zero and therefore $v_2 = -v_1$. Thus, determination of v_1 and σ_n completely determine system BER. In order to determine the error probability we need only to develop a simulation for estimating v_1 and σ_n. The Monte Carlo technique of counting errors is not required.

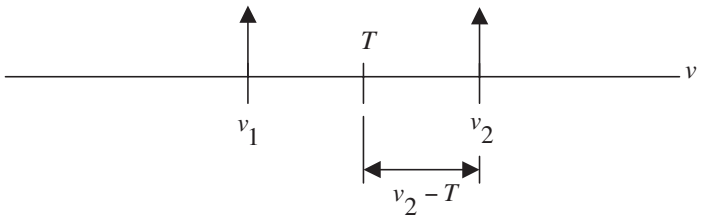
The value of v_1 can be determined by executing a noiseless simulation. If the channel noise is removed, the two conditional pfds shown in Figure 10.11(a) collapse to impulse functions ($\sigma_n = 0$) as shown in Figure 10.11(b). Each of these impulse functions has unity area, and the locations of the impulse functions define v_1 and v_2.

The value of σ_n is determined by executing a simple simulation of that portion of the system through which the noise passes. For the system being considered this consists of the receiver, which is modeled as an integrate-and-dump symbol detector. Assume that this portion of the system has transfer function $H(f)$. If white (delta-correlated) noise having a two-sided power spectral density of $N_0/2$ is input to the matched-filter receiver, the variance of the random variable V_k is

$$\sigma_v^2 = \frac{N_0}{2}\int_{-\infty}^{\infty} |H(f)|^2\, df = N_0\int_{0}^{\infty} |H(f)|^2\, df \tag{10.11}$$



(a) Probability density function of V with noise present



(b) Probability density function of V with noise absent

Figure 10.11 Binary decision process.

As we saw in Chapter 7, the equivalent noise bandwidth is defined as

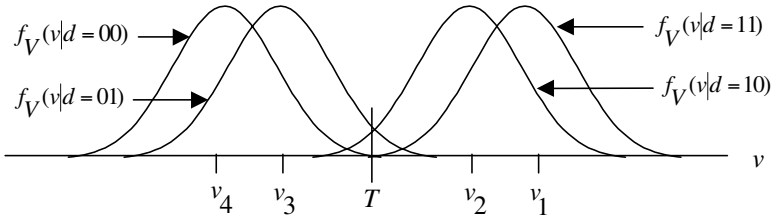
$$B_N = \int_0^\infty |H(f)|^2 \, df \tag{10.12}$$

and is the equivalent noise bandwidth of the receiver. It follows that the probability of error is given by

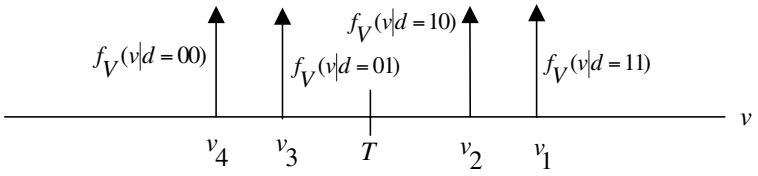
$$P_E = Q\left(\frac{v_1}{\sqrt{N_0 B_N}}\right) \tag{10.13}$$

Note that even though the system is AWGN, the nonlinear amplifier may affect system performance, since the nonlinear amplifier will affect the shape of the transmitted signals and this, in turn, will affect the value of v_1.

We now consider the presence of the transmitter filter. The effect of this filter is to spread, in time, the energy associated with the transmitted symbols beyond the symbol period giving rise to intersymbol interference. If the memory length of this filter is two symbols, the error probability associated with the transmission symbol will depend not only on the transmitted symbol but also on the previously transmitted symbol. As a result, the calculation of the probability of error will involve four conditional probability density functions rather than two conditional probability density functions as shown in Figure 10.11. This is illustrated in Figure 10.12(a).



(a) Probability density function of V with noise present



(b) Probability density function of V with noise absent

Figure 10.12 Conditional pdfs with a memory length of two.

As before, execution of a noiseless simulation will yield the values of v_1, v_2, v_3, and v_4. The system probability of error becomes

$$P_E = \frac{1}{4}\sum_{i=1}^{4} Q\left(\frac{v_i}{\sigma_v}\right) \tag{10.14}$$

The extension to a memory length of M symbols is obvious.

10.2.2 Equivalent Noise Sources

In applying the semianalytic technique we make use of the idea of an equivalent noise source. We have seen that the decision statistic V_k is a function of three components. In other words, $V_k = f(S_k, D_k, N_k)$ where S_k is due to the signal; D_k results from system-induced distortion such as ISI, and N_k is due to noise. The effects of S_k and D_k are determined by an MC simulation, and the effects of noise, represented by N_k, are treated, as we have seen, analytically. If a noise-free simulation is executed, the resulting sufficient statistic, which is denoted $V_{k,nf}$, will be a function of only S_k and D_k. To this statistic is added a random variable N_k having the variance defined by (10.11). Thus

$$V_k = V_{k,nf} + N_k \tag{10.15}$$

The random variable N_k may be viewed as a sample from an equivalent noise source $n_e(t)$ as shown in Figure 10.13. This equivalent noise source contains the combined

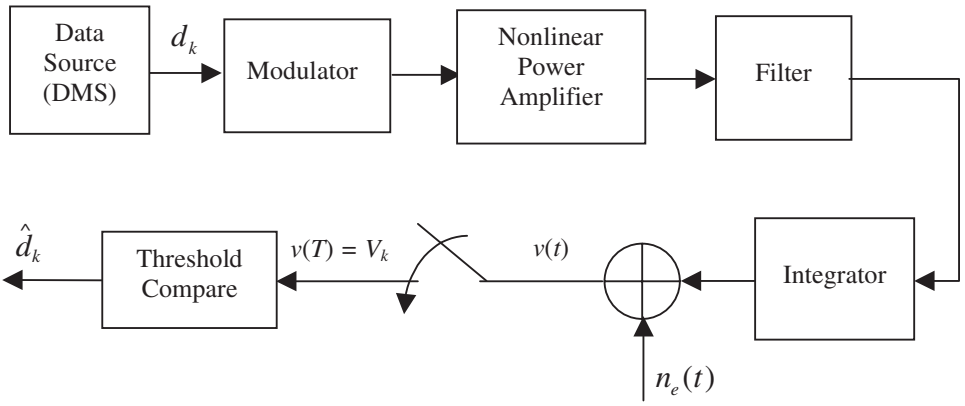


Figure 10.13 Equivalent noise source for semianalytic simulation.

effects of themal noise, interference, and other channel impairments reflected to the integrator output of the integrate-and-dump detector. If the channel noise is white, the impulse response, or equivalently the transfer function, defined in (10.11) is used to transform the channel noise to the integrator output.

10.2.3 Semianalytic BER Estimation for PSK

We now briefly consider the development of an algorithm for the determination of the BER in a binary PSK system using semianalytic simulation. We do this in a way that is easily extendable to QPSK. Consider the signal constellation illustrated in Figure 10.14. The transmitted signal points are denoted S_1 and S_2 and the corresponding decision regions are denoted D_1 and D_2. A correct decision is made at the receiver if S_i is transmitted and the received signal falls in region D_i; otherwise

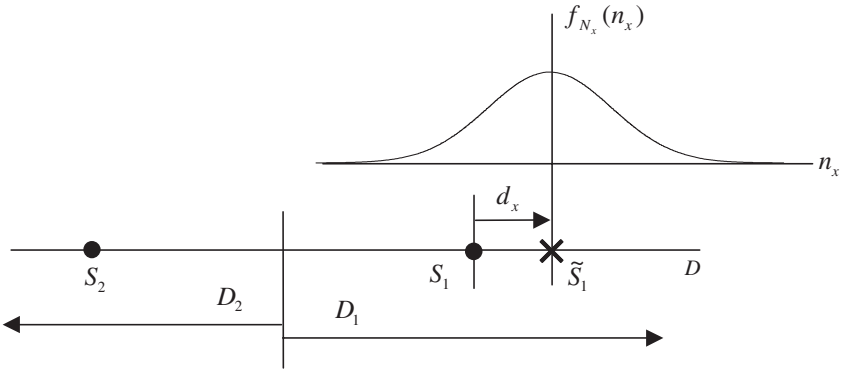


Figure 10.14 Semianalytic BER estimation for PSK.

an error occurs. In Figure 10.14 we assume that S_1 is transmitted and $\widetilde{S}_1$ is received. As discussed in the previous section, S_1 and $\widetilde{S}_1$ differ because of intersymbol interference, nonlinear distortion, of other signal-degrading effects. The difference between S_1 and $\widetilde{S}_1$ is denoted d_x. The *conditional* error probability, conditioned on the transmission of S_1 is

$$\Pr\{Error|S_1\} = \int_{\widetilde{S}_1+n\notin D_1} \frac{1}{\sqrt{2\pi}\sigma_n} \exp\left(-\frac{\left(n-\widetilde{S}_1\right)^2}{2\sigma_n^2}\right) dn \tag{10.16}$$

which is

$$\Pr\{Error|S_1\} = \int_{-\infty}^{0} \frac{1}{\sqrt{2\pi}\sigma_n} \exp\left(-\frac{\left(n-\widetilde{S}_1\right)^2}{2\sigma_n^2}\right) dn \tag{10.17}$$

In terms of the Gaussian Q-function, the preceding equation becomes

$$\Pr\{Error|S_1\} = Q\left(\frac{\widetilde{S}_1}{\sigma_n}\right) \tag{10.18}$$

Thus, knowledge of $\widetilde{S}_1$, determined using MC simulation, and σ_n, allows the conditional BER to be determined. In determining σ_n the value of B_N is found from the simulated impulse response $h[n]$.

Assume that S_k is the k^{th} transmitted bit in a simulated sequence of N bits. For each value of k, $1 \le k \le N$, S_k will be S_1 or S_2. The conditional BER is

$$\Pr\{Error|S_k\} = Q\left(\frac{\widetilde{S}_k}{\sigma_n}\right) \tag{10.19}$$

The overall BER, obtained by averaging over the entire sequence of N bits, is given by

$$P_E = \frac{1}{N}\sum_{k=1}^{N} Q\left(\frac{\widetilde{S}_k}{\sigma_n}\right) \tag{10.20}$$

Example 10.3. (PSK). The MATLAB code for executing a semianalytic simulation of a PSK system is given in Appendix C. The methodology used is that presented in the preceding paragraphs. Due to symmetry, the received symbols are rotated to positive values. The bandwidth of the transmitter filter, which gives rise to ISI, is equal to the bit rate. The increase in the BER resulting from ISI is clearly seen in Figure 10.15. ■

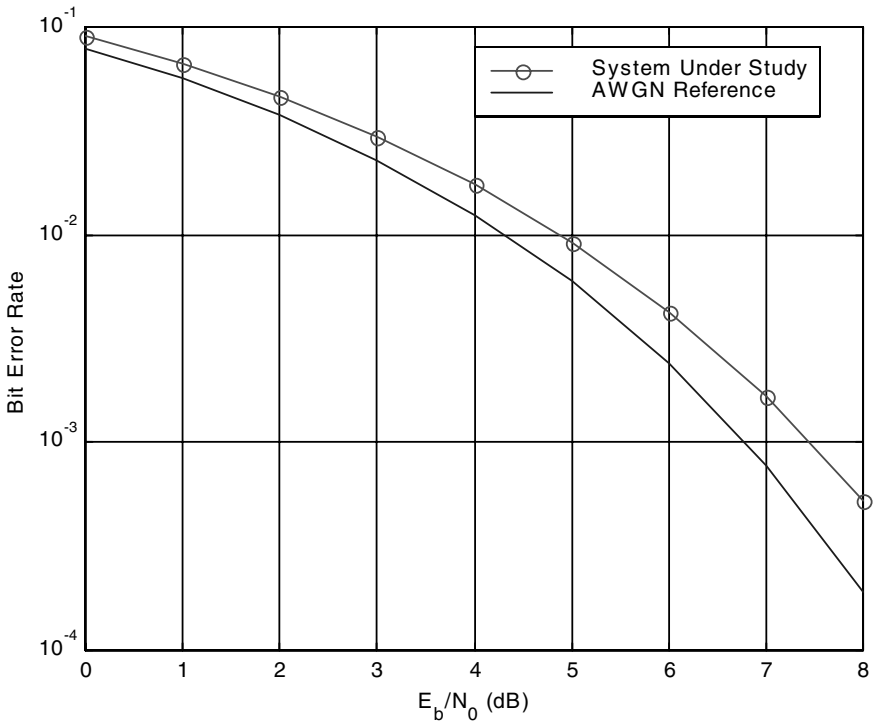


Figure 10.15 Semianalytic BER estimation for binary PSK.

10.2.4 Semianalytic BER Estimation for QPSK

We now consider a semianalytic estimator for the *symbol* error probability P_S in a QPSK system.[2] Since a QPSK signal constellation has four signal points rather than two, and since the signal space has two dimensions rather than one, the semianalytic estimator for QPSK is different from the estimator for PSK in that a dimension must be added for the quadrature channel.

Consider the signal constellation illustrated in Figure 10.16. The transmitted signal points are denoted S_i, $i = 1, 2, 3, 4$, and the decision regions are denoted D_i, $i = 1, 2, 3, 4$. As in the preceding section, a correct decision is made at the receiver if S_i is transmitted and the received signal falls in decision region D_i; otherwise an error occurs. In Figure 10.16 it is assumed that S_1 is transmitted, and the noiseless received signal is denoted $\widetilde{S}_1$. As a result of intersymbol interference and distortion, $\widetilde{S}_1 \neq S_1$. It is $\widetilde{S}_1$ rather than S_1 that is determined by the semianalytic simulation,

[2]Note that for QPSK we use semianalytic simulation to compute the symbol error probability, since QPSK points in signal space are defined by symbols rather than bits. Once the symbol error probability is determined, the symbol error probability can be converted to the bit error probability using analysis. For binary PSK, the symbol error probability and the bit error probability are, of course, equivalent.

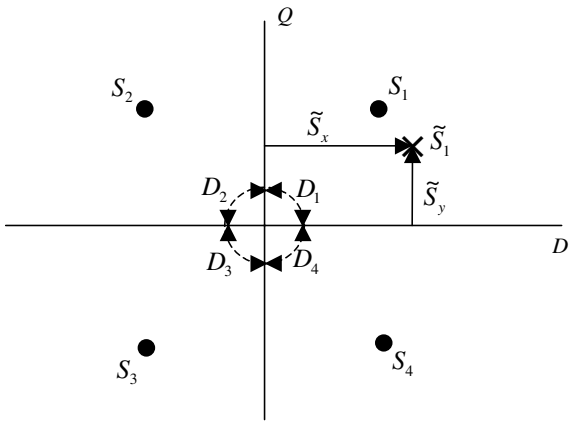


Figure 10.16 Semianalytic BER estimation for QPSK.

since the simulation will account for the effects of intersymbol interference but not the effects of noise. The direct and quadrature components of $\widetilde{S}_1$ are denoted $\widetilde{S}_x$ and $\widetilde{S}_y$, respectively, where $\widetilde{S}_x = \mathrm{Re}\left(\widetilde{S}_1\right)$ and $\widetilde{S}_y = \mathrm{Im}\left(\widetilde{S}_1\right)$. When noise is considered, by adding n_x and n_y to $\widetilde{S}_x$ and $\widetilde{S}_y$, respectively, a correct decision is made, conditioned on S_1 transmitted, if $(\widetilde{S}_x + n_x, \widetilde{S}_y + n_y) \in D_1$. An error is made if $(\widetilde{S}_x + n_x, \widetilde{S}_y + n_y) \notin D_1$. Keep in mind that since we are developing a semianalytic estimator, the impact of noise is treated analytically and does not appear in Figure 10.16.

The problem is to determine the noise components n_x and n_y that will result in an error given the received (noiseless) point in signal space $\widetilde{S}_1$. The problem is very similar to the PSK example just considered. The essential difference is that we are working in two dimensions rather than one. We assume that the direct and quadrature additive noise components are uncorrelated and jointly Gaussian. Thus, given that S_1 is transmitted and $\widetilde{S}_1$ is received, an error is made if

$$\Pr\{Error|S_1\} = \int\int_{(\widetilde{S}_x+n_x,\widetilde{S}_y+n_y)\notin D_1} \frac{1}{2\pi\sigma_n\sigma_n} \cdot \exp\left(-\frac{\left(n_x-\widetilde{S}_x\right)^2}{2\sigma_n^2} - \frac{\left(n_y-\widetilde{S}_y\right)^2}{2\sigma_n^2}\right) dn_x dn_y \qquad (10.21)$$

where n_x and n_y are the direct and quadrature noise components, and σ_n^2 represents the noise variance. In order to simplify the notation let

$$f_{N_x}(n_x|\widetilde{S}_x,\sigma_n) = \frac{1}{\sqrt{2\pi}\sigma_n}\exp\left(-\frac{\left(n_x-\widetilde{S}_x\right)^2}{2\sigma_n^2}\right) \tag{10.22}$$

and

$$f_{N_y}(n_y|\widetilde{S}_y,\sigma_n) = \frac{1}{\sqrt{2\pi}\sigma_n}\exp\left(-\frac{\left(n_y-\widetilde{S}_y\right)^2}{2\sigma_n^2}\right) \tag{10.23}$$

With these changes (10.21) becomes

$$\begin{aligned}\Pr\{Error|S_1\} = \int\int_{(\widetilde{S}_x+n_x,\widetilde{S}_y+n_y)\notin D_1} \\ \cdot f_{N_x}(n_x|\widetilde{S}_x,\sigma_n)f_{N_y}(n_y|\widetilde{S}_y,\sigma_n)\,dn_x dn_y\end{aligned} \tag{10.24}$$

This can be bounded by the expression

$$\begin{aligned}\Pr\{Error|S_1\} < \int\int_{(\widetilde{S}_x+n_x,\widetilde{S}_y+n_y)\in(D_2\cup D_3)} \\ \cdot f_{N_x}(n_x|\widetilde{S}_x,\sigma_n)f_{N_y}(n_y|\widetilde{S}_y,\sigma_n)\,dn_x dn_y \\ + \int\int_{(\widetilde{S}_x+n_x,\widetilde{S}_y+n_y)\in(D_3\cup D_4)} f_{N_x}(n_x)f_{N_y}(n_y)\,dn_x dn_y\end{aligned} \tag{10.25}$$

where the bound occurs since the decision region D_3 appears twice in (10.25). From the definition of the decision regions we can write

$$\begin{aligned}\Pr\{Error|S_1\} < \int_{-\infty}^{0} f_{N_x}(n_x|\widetilde{S}_x,\sigma_n)\,dx \int_{-\infty}^{\infty} f_{N_y}(n_y|\widetilde{S}_y,\sigma_n)\,dy \\ + \int_{-\infty}^{\infty} f_{N_x}|\widetilde{S}_x,\sigma_n(n_x)\,dn_x \int_{-\infty}^{0} f_{N_y}(n_y|\widetilde{S}_y,\sigma_n)\,dn_y\end{aligned} \tag{10.26}$$

Recognizing that two of the four integrals in (10.26) are equal to one yields

$$\Pr\{Error|S_1\} < \int_{-\infty}^{0} f_{N_x}(n_x|\widetilde{S}_x,\sigma_n)\,dn_x + \int_{-\infty}^{0} f_{N_y}(n_y|\widetilde{S}_y,\sigma_n)\,dn_y \tag{10.27}$$

Substituting (10.22) and (10.23) in the preceding expression, and using the definitions of $\widetilde{S}_x$ and $\widetilde{S}_y$, yields the bound on the conditional error probability. This conditional error probability bound is

$$\Pr\{Error|S_1\} < Q\left(\frac{\mathrm{Re}\left\{\widetilde{S}_1\right\}}{\sigma_n}\right) + Q\left(\frac{\mathrm{Im}\left\{\widetilde{S}_1\right\}}{\sigma_n}\right) \tag{10.28}$$

where, as always, $Q(\cdot)$ is the Gaussian Q function. By symmetry, the conditional error probability is the same for any of the four possible transmitted symbols.

As with PSK assume that S_k is the k^{th} transmitted symbol in a simulated sequence of N symbols. For each value of k, $1 \leq k \leq N$, S_k will be S_1, S_2, S_3, or S_4. The bound on the conditional symbol error rate is, from (10.28):

$$\Pr\{Error|S_k\} < Q\left(\frac{\mathrm{Re}\left\{\widetilde{S}_k\right\}}{\sigma_n}\right) + Q\left(\frac{\mathrm{Im}\left\{\widetilde{S}_k\right\}}{\sigma_n}\right) \tag{10.29}$$

The overall symbol error rate, obtained by averaging the conditional symbol error probability over the entire sequence of N symbols, is given by

$$P_S < \frac{1}{N}\sum_{k=1}^{N}\left[Q\left(\frac{\mathrm{Re}\left\{\widetilde{S}_k\right\}}{\sigma_n}\right) + Q\left(\frac{\mathrm{Im}\left\{\widetilde{S}_k\right\}}{\sigma_n}\right)\right] \tag{10.30}$$

The bit error rate, P_E, is $P_S/2$. Note that in the PSK case we obtained an exact solution, whereas in the case of QPSK we have a bound. The technique used here to develop a semianalytic estimator is easily extended to MPSK and QAM [1].

The estimator developed here will be used throughout the remainder of this book for evaluating the performance of a number of systems. Included will be examples illustrating the effect of multipath and fading in a wireless system and the effect of nonlinear distortion in a frequency multiplexed satellite communications system.

Example 10.4. (QPSK). The MATLAB code for executing a semianalytic simulation of a QPSK system is given in Appendix D. The simulation is run to examine the effect of the ISI resulting from transmitter filtering. The filter bandwidth is set equal to the symbol rate (one-half the bit rate, i.e., $BW = r_b/2$). Since the signal constellation is symmetric, all received signal points are rotated to the first quadrant as discussed in the preceding paragraphs.

Executing the simulation yields the signal constellation and BER illustrated in Figure 10.17. The received signal constellation is shown in the left-hand pane of Figure 10.17. Note that the received signal constellation no longer consists of 4 points, as is the case for ideal QPSK, but now consists of 16 points. In order to understand the reason for this, assume that the signal point in the first quadrant represents the data bit 00 and that the system memory, as a result of ISI, is two symbols (the current and the previous transmitted symbols). As a result, four signal points will result from the transmission of 00. These four signal points correspond to 00|00, 00|01, 00|10, and 00|11, where the vertical bar delineates the current symbol (00) and the previously transmitted symbols. Note also that each of the four points in the first quadrant are composed of points that are slightly scattered. This scattering results from the fact that the system exhibits a memory length that exceeds two symbols, although the effect of this additional memory is small. The left-hand pane of Figure 10.17 illustrates the BER of the system with transmitter filtering. The AWGN result is also illustrated for reference. The increase in the BER resulting from the ISI is clearly seen. ■

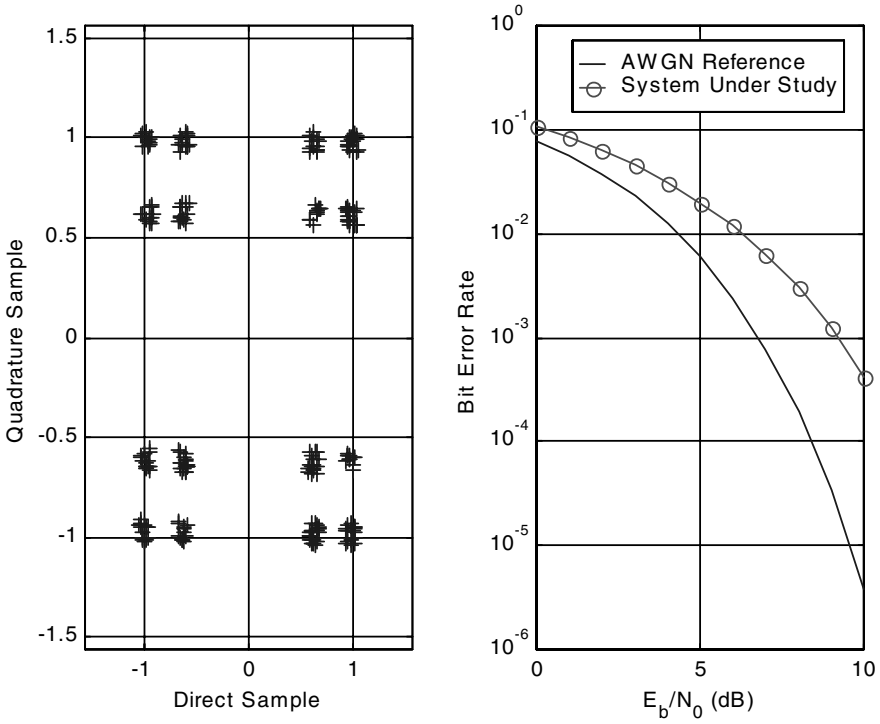


Figure 10.17 QPSK semianalytic simulation results.

10.2.5 Choice of Data Sequence

In applying semianalytic techniques to a system having memory, it is important to use a data source that generates sequences exhibiting all possible combinations of data symbols for the given memory length of the system. For example, if the memory length is three (current symbol plus the two preceding symbols) the symbol error probability is given by

$$P_S = \frac{1}{N}\sum_{k=1}^{N} \Pr\{Error|S_k, S_{k-1}, S_{k-2}\} \tag{10.31}$$

The error probability will, in general, be different for each (S_k, S_{k-1}, S_{k-2}) sequence. Thus, all combinations of S_k, S_{k-1}, and S_{k-2}, must appear an equal number of times to properly account for the memory efficts. In general, if a binary system exhibits significant memory spanning N symbols, then all binary sequences of length N should be generated by the data source an equal number of times in the simulation. For a binary system there are 2^N sequences of length N. There are three popular ways to accomplish, or at leat approximate, this requirement as follows:

1. If N is reasonably large one may wish to use a PN Sequence for the data source. As discussed in Chapter 7, the number of sequences generated will not be 2^N as desired but will be $L = 2^N - 1$, since the sequence of N consecutive zeros will not occur. The result will be an unbalanced sequence having $\frac{L}{2}$ ones and $\frac{L}{2} - 1$ zeros. If N is large, this effect is negligible. Note that one is free to select N greater than the memory length in order to mitigate this effect. Choosing N larger than necessary will, however, result in a longer simulation execution time.

2. If a perfectly balanced sequence is desired, a deBruijn sequence [2] may be used. As briefly discussed in Chapter 7, a deBruijn sequence is formed by adding an extra zero at the point where there are $N - 1$ zeros in the output of a PN sequence generator.

3. One may of course simply execute the semianalytic sequence using random data. If this sequence is sufficiently long, all data symbol combinations will occur approximately the same number of times. This is the approach taken in Examples 10.3 and 10.4.

10.3 Summary

In this chapter, simulation examples of binary PSK and differential QPSK communication systems were presented. Strict Monte Carlo simulations were first developed. These simulations were easily developed using the concepts presented in the previous chapter. The PSK system was very simple and served to illustrate the basic concepts. The only degrading effects were intersymbol interference and additive channel noise. The differential QPSK example considered the simulation of a much more realistic system.

We next considered semianalytic simulation. Both PSK and QPSK illustrated that the semianalytic estimators for BER are different for the PSK and QPSK cases. Thus, a unique procedure for conducting a semianalytic simulation does not exist. While the estimators are quite different, the methodologies are the same, in that the semianalytic simulation captures all deterministic system perturbations, such as intersymbol interference and distortion due to nonlinearities through a conventional Monte Carlo simulation. The effects of noise and other stochastic effects are dealt with analytically. This requires that the pdf of the samples upon which a bit or symbol decision is made is known. The simplest case, and the case most often used, assumes that the noise is Gaussian and that the system is linear from the point at which the noise enters the system to the point where bit or symbol decisions are made is linear. In this case, the pdf of the decision statistic is Gaussian, and the Monte Carlo simulation is executed to establish the mean of the decision statistic. We saw that for those cases in which the semianalytic method can be used, very fast simulations result.

10.4 References

1. M. C. Jeruchim, P. Balaban, and K. S. Shanmugan, *Simulation of Communication Systems*, 2nd ed., New York: Kluwer Academic/Plenum Publishers, 2000.

2. S. Golomb, *Shift Register Sequences*, Laguna Hills, CA: Aegean Press, 1982.

10.5 Problems

10.1 Verify that the MATLAB code segment used in Appendix A

```
[Btr,Atr]=butter(5,0.2)
```

with a sampling frequency of 10 samples/symbol yields a filter bandwidth equal to the symbol rate.

10.2 Rerun the simulation described in Example 10.1 using filter bandwidths of one-half the symbol rate and double the symbol rate. Compare the result of these two simulations with the result given in Example 10.1.

10.3 Rerun the simulation described in Example 10.1 using a sampling rate of 20 samples per symbol. Does this result in an improved estimate of delay? Explain. Estimate the BER using 20 samples/symbol and compare the result with that obtained in Example 10.1 using 10 samples/symbol.

10.4 Using the appropriate MATLAB routines, compare the required execution times for the Monte Carlo and semianalytic simulations of the binary PSK system given in Examples 10.1 and 10.3, respectively.

10.5 The semianalytic BER estimator for the PSK system (Appendix C) contains the line of code

```
nbwideal=1/(2*tb)
```

Explain the purpose of this line of code and verify that it is correct. The equivalent line of code for the BER estimator for the QPSK system (Appendix D) is

```
nbwideal=1/(2*tb*2)
```

Verify the correctness of this line of code.

10.6 Modify the simulation given in Example 10.1 so that the PSK system is simulated using a symbol-by-symbol approach rather than by using a block serial approach. In other words, the random binary data source will generate binary bits (0 or 1), and the waveform samples corresponding to these binary

symbols will be repeated the required number of times to satisfy a given samples/symbol specification. (You may wish to review Chapter 5 in order to derive an efficient filter simulation using MATLAB to establish the necessary numerator and denominator polynomials for the filter transfer function and use them in a sample-by-sample simulation.)

10.7 Example 10.2 examines the Monte Carlo simulation of a differential QPSK system. Rewrite this simulation for QPSK rather than for differential QPSK. Sanity check the simulation result by comparing it with the theoretical result for QPSK.

10.6 Appendix A: Simulation Code for Example 10.1

10.6.1 Main Program

```
% File: c10_MCBPSKrun.m
function [BER,Errors]=MCBPSKrun(N,EbNo,delay,FilterSwitch)
SamplesPerSymbol = 10;                          % samples per symbol
BlockSize = 1000;                               % block size
NoiseSigma = sqrt(SamplesPerSymbol/(2*EbNo)); % scale noise level
DetectedSymbols = zeros(1,BlockSize);           % initialize vector
NumberOfBlocks = floor(N/BlockSize);            % number of blocks
                                                %     processed
[BTx,ATx] = butter(5,2/SamplesPerSymbol);       % compute filter
                                                %     parameters
[TxOutput,TxFilterState] = filter(BTx,ATx,0); % initialize state
                                                %     vector
BRx = ones(1,SamplesPerSymbol); ARx=1;          % matched filter
                                                %     parameters
Errors = 0;                                     % initialize error
                                                %     counter
%
% Simulation loop begins here.
%
for Block=1:NumberOfBlocks
    %
    % Generate transmitted symbols
    %
    [SymbolSamples,TxSymbols] = random_binary(BlockSize,...
        SamplesPerSymbol);
    %
    % Transmitter filter if desired.
    %
    if FilterSwitch==0
        TxOutput = SymbolSamples;
    else
        [TxOutput,TxFilterState] = filter(BTx,ATx,SymbolSamples,...
        TxFilterState);
    end
    %
    % Generate channel noise.
    %
    NoiseSamples = NoiseSigma*randn(size(TxOutput));
    %
    % Add signal and noise.
    %
    RxInput = TxOutput + NoiseSamples;
```

```
    %
    % Pass Received signal through matched filter.
    %
    IntegratorOutput = filter(BRx,ARx,RxInput);
    %
    % Sample matched filter output every SamplesPerSymbol samples,
    % compare to transmitted bit, and count errors.
    %
    for k=1:BlockSize,
        m = k*SamplesPerSymbol+delay;
        if (m < length(IntegratorOutput))
            DetectedSymbols(k)=(1-sign(IntegratorOutput(m)))/2;
            if (DetectedSymbols(k) ~= TxSymbols(k))
                Errors = Errors + 1;
            end
        end
    end
end
BER = Errors/(BlockSize*NumberOfBlocks);      % calculate BER
% End of function file.
```

10.6.2 Supporting Program: random_binary.m

```
% file: random_binary.m
function [x, bits] = random_binary(nbits,nsamples)
% This function genrates a random binary waveform of length nbits
% sampled at a rate of nsamples/bit.
x = zeros(1,nbits*nsamples);
bits = round(rand(1,nbits));
for m=1:nbits
    for n=1:nsamples
        index = (m-1)*nsamples + n;
        x(1,index) = (-1)^bits(m);
    end
end
% End of function file.
```

10.7 Appendix B: Simulation Code for Example 10.2

10.7.1 Main Program

```
% file c10_MCQPSKrun.m
function BER_MC=MCQPSKrun(N,Eb,No,ChanAtt,...
    TimingBias,TimingJitter,PhaseBias,PhaseJitter)
fs = 1e+6;                          % sampling Rate (samples/second)
SymRate = 1e+5;                     % symbol rate (symbols/second)
Ts = 1/fs;                          % sampling period
TSym = 1/SymRate;                   % symbol period
SymToSend = N;                      % symbols to be transmitted
ChanBW = 4.99e+5;                   % bandwidth of channel (Hz)
MeanCarrierPhaseError = PhaseBias;  % mean of carrier phase
StdCarrierPhaseError = PhaseJitter; % stdev of phase error
MeanSymbolSyncError = TimingBias;   % mean of symbol sync error
StdSymbolSyncError = TimingJitter;  % stdev of symbol sync error
ChanGain = 10^(-ChanAtt/20);        % channel gain (linear units)
TxBitClock = Ts/2;                  % transmitter bit clock
RxBitClock = Ts/2;                  % receiver bit clock
%
% Standard deviation of noise and signal amplitude at receiver input.
%
RxNoiseStd = sqrt((10^((No-30)/10))*(fs/2));      % stdev of noise
TxSigAmp = sqrt(10^((Eb-30)/10)*SymRate);         % signal amplitude
%
% Allocate some memory for probes.
%
SampPerSym = fs/SymRate;
probe1 = zeros((SymToSend+1)*SampPerSym,1);
probe1counter = 1;
probe2 = zeros((SymToSend+1)*SampPerSym,1);
probe2counter = 1;
%
% Counters to keep track of how many symbols have have been sent.
%
TxSymSent = 1;
RxSymDemod = 0;
%
% Buffers that contain the transmitted and received data.
%
[unused,SourceBitsI] = random_binary(SymToSend,1);
[unused,SourceBitsQ] = random_binary(SymToSend,1);
%
% Differentially encode the transmitted data.
%
```

```
TxBitsI = SourceBitsI*0;                          % set first I bit
TxBitsQ = SourceBitsQ*0;                          % set first Q bit
for k=2:length(TxBitsI)
    TxBitsI(k) = or(and(not(xor(SourceBitsI(k),SourceBitsQ(k))),...
        xor(SourceBitsI(k),TxBitsI(k-1))), ...
        and(xor(SourceBitsI(k),SourceBitsQ(k)),...
        xor(SourceBitsQ(k),TxBitsQ(k-1))));
    TxBitsQ(k) = or(and(not(xor(SourceBitsI(k),SourceBitsQ(k))),...
        xor(SourceBitsQ(k),TxBitsQ(k-1))), ...
        and(xor(SourceBitsI(k),SourceBitsQ(k)),...
        xor(SourceBitsI(k),TxBitsI(k-1))));
end
%
% Make a complex data stream of the I and Q bits.
%
TxBits = ((TxBitsI*2)-1)+(sqrt(-1)*((TxBitsQ*2)-1));
%
RxIntegrator = 0;                  % initialize receiver integrator
TxBitClock = 2*TSym;               % initialize transmitter
%
% Design the channel filter, and create the filter state array.
%
[b,a] = butter(2,ChanBW/(fs/2));
b = [1]; a = [1];                                 % filter bypassed
[junk,FilterState] = filter(b,a,0);
%
% Begin simulation loop.
%
while TxSymSent < SymToSend
    %
    % Update the transmitter's clock, and see
    % if it is time to get new data bits
    %
    TxBitClock = TxBitClock+Ts;
    if TxBitClock > TSym
        %
        % Time to get new bits
        %
        TxSymSent = TxSymSent+1;
        %
        % We don't want the clock to increase to infinity,
        % so subtract off an integer number of Tb seconds.
        %
        TxBitClock = mod(TxBitClock,TSym);
        %
```

```
        % Get the new bit, and scale it up appropriately.
        %
        TxOutput = TxBits(TxSymSent)*TxSigAmp;
        end
    %
    % Pass the transmitted signal through the channel filter.
    %
    [Rx,FilterState] = filter(b,a,TxOutput,FilterState);
    %
    % Add white Gaussian noise to the signal.
    %
    Rx = (ChanGain*Rx)+(RxNoiseStd*(randn(1,1)+sqrt(-1)*randn(1,1)));
    %
    % Phase rotation due to receiver carrier synchronization error.
    %
    PhaseRotation = exp(sqrt(-1)*2*pi*...
        (MeanCarrierPhaseError+(randn(1,1)*StdCarrierPhaseError))...
    /360);
    Rx = Rx*PhaseRotation;
    probe1(probe1counter) = Rx; probe1counter=probe1counter+1;
    %
    % Update the Integrate and Dump Filter at the receiver.
    %
    RxIntegrator = RxIntegrator+Rx;
    probe2(probe2counter) = RxIntegrator;
    probe2counter=probe2counter+1;
    %
    % Update the receiver clock, to see if it is time to
    % sample and dump the integrator.
    %
    RxBitClock = RxBitClock+Ts;
    xTSym = TSym*(1+MeanSymbolSyncError+...
        (StdSymbolSyncError*randn(1,1)));
    if RxBitClock > RxTSym               % time to demodulate symbol
        RxSymDemod = RxSymDemod+1;
        RxBitsI(RxSymDemod) = round(sign(real(RxIntegrator))+1)/2;
        RxBitsQ(RxSymDemod) = round(sign(imag(RxIntegrator))+1)/2;
        RxBitClock = RxBitClock - TSym;   % reset receive clock
        RxIntegrator = 0;                 % reset integrator
    end
end
%
% Differential decoder.
%
SinkBitsI = SourceBitsI*0;                   % set first I sink bit
```

```
SinkBitsQ = SourceBitsQ*0;                    % set first Q sink bit
%
for k=2:RxSymDemod
    SinkBitsI(k) = or(and(not(xor(RxBitsI(k),RxBitsQ(k))),...
        xor(RxBitsI(k),RxBitsI(k-1))),...
        and(xor(RxBitsI(k),RxBitsQ(k)),...
        xor(RxBitsQ(k),RxBitsQ(k-1))));
    SinkBitsQ(k) = or(and(not(xor(RxBitsI(k),RxBitsQ(k))),...
        xor(RxBitsQ(k),RxBitsQ(k-1))),...
        and(xor(RxBitsI(k),RxBitsQ(k)),...
        xor(RxBitsI(k),RxBitsI(k-1))));
end;
%
% Look for best time delay between input and output for 100 bits.
%
[C,Lags] = vxcorr(SourceBitsI(10:110),SinkBitsI(10:110));
[MaxC,LocMaxC] = max(C);
BestLag = Lags(LocMaxC);
%
% Adjust time delay to match best lag
%
if BestLag > 0
    SourceBitsI = SourceBitsI(BestLag+1:length(SourceBitsI));
    SourceBitsQ = SourceBitsQ(BestLag+1:length(SourceBitsQ));
elseif BestLag < 0
    SinkBitsI = SinkBitsI(-BestLag+1:length(SinkBitsI));
    SinkBitsQ = SinkBitsQ(-BestLag+1:length(SinkBitsQ));
end
%
% Make all arrays the same length.
%
TotalBits = min(length(SourceBitsI),length(SinkBitsI));
TotalBits = TotalBits-20;
SourceBitsI = SourceBitsI(10:TotalBits);
SourceBitsQ = SourceBitsQ(10:TotalBits);
SinkBitsI = SinkBitsI(10:TotalBits);
SinkBitsQ = SinkBitsQ(10:TotalBits);
%
% Find the number of errors and the BER.
%
Errors = sum(SourceBitsI ~= SinkBitsI) + sum(SourceBitsQ ~=...
    SinkBitsQ);
BER_MC = Errors/(2*length(SourceBitsI));
% End of function file.
```

10.7.2 Supporting Programs

Program `random_binary.m` is defined in Appendix A of this chapter.

10.7.3 vxcorr.m

```
% File: vxcorr.m
function [c,lags] = vxcorr(a,b)
% This function calculates the unscaled cross-correlation of 2
% vectors of the same length. The output length(c) is
% length(a)+length(b)-1. It is a simplified function of xcorr
% function in matlabR12 using the definition:
% c(m) = E[a(n+m)*conj(b(n))] = E[a(n)*conj(b(n-m))]
%
a = a(:);                                % convert a to column vector
b = b(:);                                % convert b to column vector
M = length(a);                           % same as length(b)
maxlag = M-1;                            % maximum value of lag
lags = [-maxlag:maxlag]';                % vector of lags
A = fft(a,2^nextpow2(2*M-1));            % fft of A
B = fft(b,2^nextpow2(2*M-1));            % fft of B
c = ifft(A.*conj(B));                    % crosscorrelation
%
% Move negative lags before positive lags.
%
c = [c(end-maxlag+1:end,1);c(1:maxlag+1,1)];
%
% Return row vector if a, b are row vectors.
%
[nr nc]=size(a);
if(nr>nc)
    c=c.';
    lags=lags.';
end
% End of function file.
```

10.8 Appendix C: Simulation Code for Example 10.3

10.8.1 Main Program: c10_PSKSA.m

```
% File: c10_PSKSA.m
NN = 256;                              % number of symbols
tb = 1;                                % bit file
p0 = 1;                                % power
fs = 16;                               % samples/symbol
ebn0db = [0:1:8];                      % Eb/No vector in dB
[bt,at] = butter(5,2/fs);              % transmitter filter parameters
x = random_binary(NN,fs);              % establish PSK signal
y1 = x;                                % save signal
y2a = y1*sqrt(p0);                     % scale amplitude
y2 = filter(bt,at,y2a);                % transmitter output
br = ones(1,fs); br = br/fs; ar = 1; % matched filter parameters
y = filter(br,ar,y2);                  % matched filter output
%
% End of simulation.
%
% The following code sets up the semianalytic estimator. Find the
% max. magnitude of the cross correlation and the corresponding lag.
%
[cor lags] = vxcorr(x,y);              % compute crosscorrelation
[cmax nmax] = max(abs(cor));           % maximum of crosscorrelation
timelag = lags(nmax);                  % lag at max crosscorrelation
theta = angle(cor(nmax));              % determine angle
y = y*exp(-i*theta);                   % derotate
%
% Noise BW calibration.
%
hh = impz(br,ar);                      % receiver impulse response
nbw = (fs/2)*sum(hh.^2);               % noise bandwidth
%
% Delay the input and do BER estimation on the NN-20+1 128 bits.
% Use middle sample. Make sure the index does not exceed number
% of input points. Eb should be computed at the receiver input.
%
index = (10*fs+8:fs:(NN-10)*fs+8);
xx = x(index);
yy = y(index-timelag+1);
eb = tb*sum(abs(y2).^2)/length(y2);
eb = eb/2;
[peideal,pesystem] = psk_berest(xx,yy,ebn0db,eb,tb,nbw);
semilogy(ebn0db,pesystem,'ro-',ebn0db,peideal); grid;
xlabel('E_b/N_0 (dB)'); ylabel('Bit Error Rate')
```

```
legend('System Under Study','AWGN Reference',0)
% End of script file.
```

10.8.2 Supporting Programs

Program `random_binary.m` is defined in Appendix A of this chapter. Program `vxcorr.m` is defined in Appendix B of this chapter.

psk_berest

```
% File: psk_berest.m
function [peideal,pesystem] = psk_berest(xx,yy,ebn0db,eb,tb,nbw)
% ebn0db is an array of Eb/No values in db (specified at the
% receiver input); tb is the bit duration and nbw is the noise BW
% xx is the reference (ideal) input; yy is the filtered output;
%
nx = length(xx);
%
% For comparision purposes, set the noise BW of the ideal
% receiver (integrate and dump) to be equal to rs/2.
%
nbwideal = 1/(2*tb);                              % noise bandwidth
for m=1:length(ebn0db)
    peideal(m) = 0.0; pesystem(m) = 0.0;            % initialize
    %
    % Find n0 and the variance of the noise.
    %
    ebn0(m) = 10^(ebn0db(m)/10);                  % dB to linear
    n0 = eb/ebn0(m);                              % noise power
    sigma = sqrt(n0*nbw*2);                       % variance
    sigma1 = sqrt(n0*nbwideal*2);                 % variance of ideal
    %
    % Multiply the input constellation/signal by a scale factor so
    % that input constellation and the constellations/signal at the
    % input to receive filter have the same ave power
    % a = sqrt(2*eb/(2*tb)).
    %
    b = sqrt(2*eb/tb)/sqrt(sum(abs(xx).^2)/nx);
    d1 = b*abs(xx);
    d3 = abs(yy);
    peideal(m) = sum(q(d1/sigma1));
    pesystem(m) = sum(q(d3/sigma));
end
peideal = peideal/nx;
pesystem = pesystem/nx;
% End of function file.
```

q.m

```
% File: q.m
function out=q(x)
out=0.5*erfc(x/sqrt(2));
% End of function file.
```

10.9 Appendix D: Simulation Code for Example 10.4

```
% File: c14_QPSKSA.m
%
% Default parameters
%
NN = 256;                               % number of symbols
tb = 0.5;                               % bit time
p0 = 1;                                 % power
fs = 16;                                % samples/symbol
ebn0db = [0:1:10];                      % Eb/N0 vector
[b,a] = butter(5,1/16);                 % transmitter filter parameters
%
% Establish QPSK signals
%
x = random_binary(NN,fs)+i*random_binary(NN,fs); % QPSK signal
y1 = x;                                 % save signal
y2a = y1*sqrt(p0);                      % scale amplitude
%
% Transmitter filter
%
y2 = filter(b,a,y2a);                   % filtered signal
%
% Matched filter
%
b = ones(1,fs); b = b/fs; a = 1;        % matched filter parameters
y = filter(b,a,y2);                     % matched filter output
%
% End of simulation
%
% Use the semianalytic BER estimator. The following sets
% up the semi analytic estimator. Find the maximum magnitude
% of the cross correlation and the corresponding lag.
%
[cor lags] = vxcorr(x,y);
cmax = max(abs(cor));
nmax = find(abs(cor)==cmax);
timelag = lags(nmax);
theta = angle(cor(nmax));
y = y*exp(-i*theta);                    % derotate
%
% Noise BW calibration
%
hh = impz(b,a);                         % receiver impulse response
nbw = (fs/2)*sum(hh.^2);                % noise bandwidth
```

```
%
% Delay the input, and do BER estimation on the last 128 bits.
% Use middle sample. Make sure the index does not exceed number
% of input points. Eb should be computed at the receiver input.
%
index = (10*fs+8:fs:(NN-10)*fs+8);
xx = x(index);
yy = y(index-timelag+1);
[n1 n2] = size(y2); ny2 = n1*n2;
eb = tb*sum(sum(abs(y2).^2))/ny2;
eb = eb/2;
[peideal,pesystem] = qpsk_berest(xx,yy,ebn0db,eb,tb,nbw);
subplot(1,2,1)
yscale = 1.5*max(real(yy));
plot(yy,'+')
xlabel('Direct Sample'); ylabel('Quadrature Sample'); grid;
axis([-yscale yscale -yscale yscale])
subplot(1,2,2)
semilogy(ebn0db,peideal,ebn0db,pesystem,'ro-'); grid;
xlabel('E_b/N_0 (dB)'); ylabel('Bit Error Rate')
legend('AWGN Reference','System Under Study')
% End of script file.
```

10.9.1 Supporting Programs

Program `random_binary.m` is defined in Appendix A of this chapter. Program `vxcorr.m` is defined in Appendix B of this chapter. Program `q.m` is defined in Appendix C of this chapter.

qpsk_berest

```
% File: qpsk_berest.m
function [peideal,pesystem] = qpsk_berest(xx,yy,ebn0db,eb,tb,nbw)
% ebn0db is an array of Eb/No values in db (specified at the
% receiver input); tb is the bit duration and nbw is the noise BW
% xx is the reference (ideal) input; yy is the distorted output;
%
[n1 n2] = size(xx); nx = n1*n2;
[n3 n4] = size(yy); ny = n3*n4;
[n5 n6] = size(ebn0db); neb = n5*n6;
%
% For comparision purposes, set the noise BW of the ideal
% receiver (integrate and dump) to be equal to rs/2.
%
nbwideal = 1/(2*tb*2);
for m=1:neb
```

```
    peideal(m) = 0.0; pesystem(m) = 0.0;          % initialize
    %
    % Find n0 and the variance of the noise.
    %
    string1 = ['Eb/No = ',num2str(ebn0db(m))];
    disp(string1)                                 % track execution
    ebn0(m) = 10^(ebn0db(m)/10);                  % dB to linear
    n0 = eb/ebn0(m);                              % noise power
    sigma = sqrt(n0*nbw*2);                       % variance
    sigma1 = sqrt(n0*nbwideal*2);                 % variance of ideal
    %
    % Multiply the input constellation/signal by a scale factor so
    % that input constellation and the constellations/signal at the
    % input to receive filter have the same ave power
    % a=sqrt(2*eb/(2*tb)).
    %
    b = sqrt(2*eb/tb)/sqrt(sum(abs(xx).^2)/nx);
    for n=1:nx
    theta = angle(xx(n));
    if (theta<0)
    theta = theta+2*pi;
    end
    %
    % Rotate x and y to the first quadrant and compute BER.
    %
    xxx(n) = b*xx(n)*exp(-i*(theta-(pi/4)));
    yyy(n) = yy(n)*exp(-i*(theta-(pi/4)));
    d1 = real(xxx(n)); d2 = imag(xxx(n));         % reference
    d3 = real(yyy(n)); d4 = imag(yyy(n));         % system
    pe1 = q(d1/sigma1)+q(d2/sigma1);              % reference
    pe2 = q(d3/sigma)+q(d4/sigma);                % system
    peideal(m) = peideal(m)+pe1;                  % SER of reference
    pesystem(m) = pesystem(m)+pe2;                % SER of system
    end
end
peideal = (1/2)*peideal./nx;                      % convert to BER
pesystem = (1/2)*pesystem./nx;                    % convert to BER
% End of function file.
```

Chapter 11

METHODOLOGY FOR SIMULATING A WIRELESS SYSTEM

In this chapter we illustrate various aspects of methodology as they apply to the problem of estimating the performance of a wireless digital communication system operating over a slowly fading channel. We start with a block diagram of the system, which is shown in Figure 11.1.

We will assume that the design of the system is nearly complete and the following aspects of the design have been finalized:

1. The voice signal is source encoded using linear predictive coding to produce an output bit rate of 9,600 bits per second.

2. Error control coding is accomplished through the use of a rate 1/3 convolutional encoder with hard decision decoding (or soft decision decoding with 8 levels of quantizing).

3. The filters used in the system are 50% square root raised cosine (SQRC).

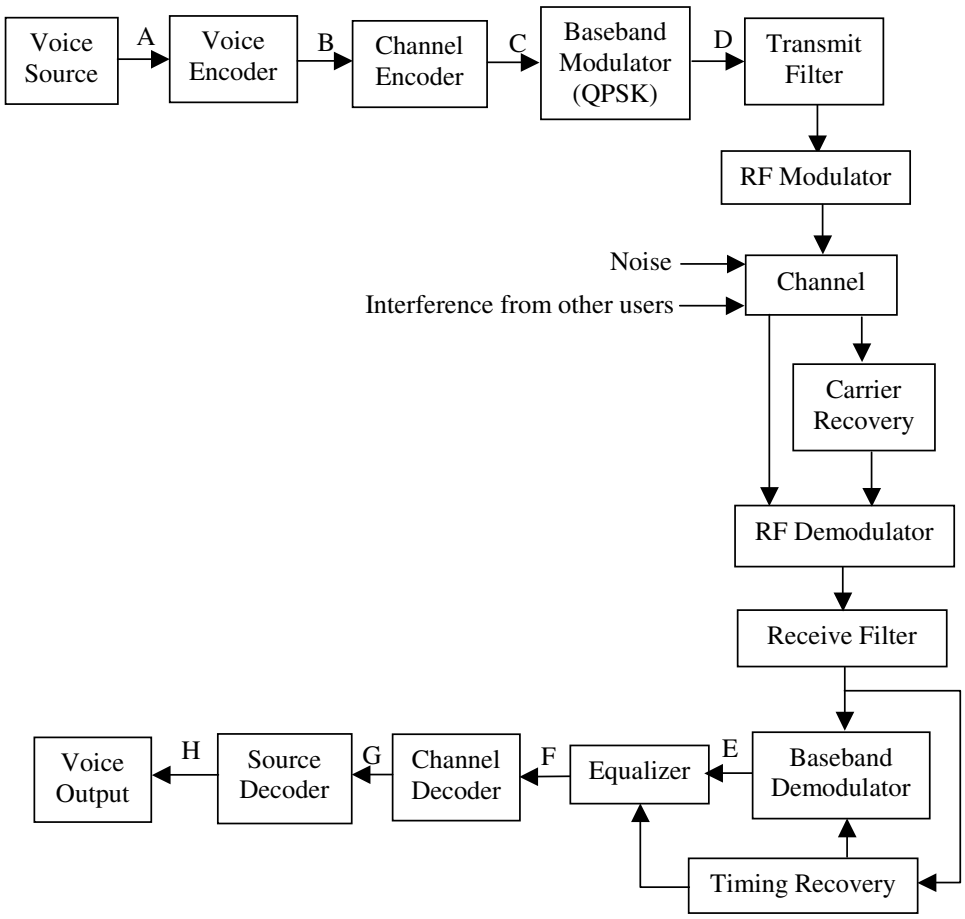


Figure 11.1 System-level simulation model.

4. The equalizer is a 9 tap synchronously spaced linear mean square (LMS) equalizer.

5. Modulation is QPSK with coherent demodulation at the receiver.

The channel over which this system is assumed to operate is characterized as a "two-ray" multipath channel with slow fading (slow compared to the symbol rate so that the channel can be treated as quasi-static). The input-output relationship of the channel is given by

$$\widetilde{y}(t) = \widetilde{a}_1(t)\,\widetilde{x}(t - \tau_1(t)) - \widetilde{a}_2(t)\,\widetilde{x}(t - \tau_2(t)) \tag{11.1}$$

where $\widetilde{x}(t)$ and $\widetilde{y}(t)$ are the complex (lowpass) input and output, respectively, and $\widetilde{a}_1(t)$ and $\widetilde{a}_2(t)$ represent the complex attenuation of the two multipath components

with delays $\tau_1(t)$ and $\tau_2(t)$. The complex attenuations are modeled as independent stationary processes, and the bandwidths of these processes (and hence their rate of change) are assumed to be small compared to the symbol rate. The complex attenuations are modeled by two independent complex Gaussian processes (Rayleigh envelope), and the delays are assumed to have uniform distributions. (This channel will be developed in detail in Chapter 14.)

The channel characteristics are randomly changing as a function of time. Therefore, the received signal power and the amount of signal distortion introduced by the channel, which will impact system performance, will also be changing over time. When the signal loss and distortion are small, the system performance will be very good, but when the signal loss and distortion are severe the system performance will degrade significantly. The overall performance metric of interest in this system is the output voice quality, which is obtained from listening tests. In these tests the output of the voice decoder is recorded and played back to a number of human subjects who rate the voice quality from 1 to 5, with 1 being the poorest quality and 5 being the highest quality. The average of the individual scores from a set of subjects is used as the voice-quality metric, and the overall goal of the system design is to guarantee a voice-quality metric greater than or equal to 3 at least 98% of the time. If the voice-quality metric is less than 3, the communication link is declared to be unusable and out of service.

The objective of this simulation exercise is to evaluate the system performance, as measured by a voice-quality metric V as a function of E_b/N_0, and compute the value of E_b/N_0 needed to maintain an outage probability less than 2% at a voice quality metric threshold of 3. We now present the details of the overall approach that can be used to estimate the outage probability as a function of E_b/N_0.

11.1 System-Level Simplifications and Sampling Rate Considerations

The slow-fading assumption leads to the following immediate simplifications of the simulation model that will be used for performance estimation.

1. *Synchronization*: For the purposes of performance estimation it can be assumed that synchronization is ideal, since fading is slow and hence the timing and phase recovery subsystems can establish near ideal timing and phase references. These subsystems can be omitted from the simulation model for performance estimation.

2. *Static channel*: The slow-fading assumption also implies that the channel can be treated as quasi-static and snapshots of the channel can be used during performance estimation. The channel model now reduces to

$$\widetilde{y}(t) = \widetilde{a}_1 \widetilde{x}(t - \tau_1) - \widetilde{a}_2 \widetilde{x}(t - \tau_2) \tag{11.2}$$

   where $\widetilde{a}_1, \widetilde{a}_2, \tau_1$, and τ_2 are now random variables whose values remain fixed during each performance estimation simulation. It is common practice to

assume (normalize) $\tau_1 = 0$, and $\widetilde{a}_1 = 1$, which results in the input-output relationship

$$\widetilde{y}(t) = \widetilde{x}(t) - \widetilde{a}\widetilde{x}(t-\tau) \tag{11.3}$$

and the channel transfer function

$$H_c(f) = 1 - \widetilde{a}\exp(-j2\pi f\tau) \tag{11.4}$$

In this model, the channel is characterized by two random variables $\widetilde{a}$ and τ, where $\widetilde{a}$ has a Rayleigh pdf and τ has a uniform pdf.

3. *Radio frequency (RF) modulator and demodulator*: These two blocks can be assumed to perform ideal frequency translations and hence they can be eliminated from the simulation model. The entire system can then be simulated using complex lowpass equivalent representations.

Sampling Rate

One other important simulation parameter that can be established at the outset is the overall sampling rate. The voice source, the source encoder, the error control encoder blocks on the transmit side, and the error control decoder and the source decoder blocks on the receive side, operate on symbol sequences and should be simulated using one sample per symbol (i.e., they are processed at the appropriate symbol or bit rate). From the output of the QPSK modulator to the output of the equalizer we are dealing with waveform representations and hence the signals, and components in this portion of the overall system should be simulated at a sampling rate consistent with the bandwidths of the signals and components such as filters. Since there are no nonlinearities and time-varying components in the system, we need not be concerned about any bandwidth expansion. Also, there is no need to consider multirate sampling for this portion of the system, since we are not dealing with multiple signals with widely differing bandwidths. The sampling rate for the "analog" portion of the system can be set at 16 times the bandwidth of the QPSK signal. This can be truncated to the bandwidth of the raised cosine filter, which is 0.75 times the symbol rate (0.5R + 50% of (0.5R) = 0.75R). With the QPSK symbol rate of $R = (9600 \times 3)/2 = 14{,}400$ symbols/second, we can use a sampling rate of $16 \times 0.75 \times 14{,}400 = 172{,}800$ samples/second. This is equivalent to 12 samples/QPSK symbol.

11.2 Overall Methodology

In outage probability estimation we are interested in determining the fraction of time that the channel conditions degrade the system performance below some acceptable threshold. Since the channel parameters are random variables, we can use the Monte Carlo approach to determine the outage probability induced by the channel. The Monte Carlo approach will involve drawing random numbers from the distributions of the channel parameters $\widetilde{a}$ and τ and computing the system performance for each pair of values for $\widetilde{a}$ and τ. The outage probability is estimated as the

percentage of channels simulated that yield a performance metric that is below the acceptable (threshold) level. Note that this part of the Monte Carlo simulation is different from the Monte Carlo simulation used for performance estimation for each channel condition. Monte Carlo simulation for performance estimation will involve the generation of sampled values of one or more random processes that represent the signals and noise.

The flowchart of the procedure for estimating outage probability is shown in Figure 11.2. To estimate outage probability, we need to compute system performance as measured by the output voice-quality metric for each value of E_b/N_0 and for thousands of snapshots of the channel. We typically define the channel in terms of $\widetilde{a}$ and τ (the amplitude-delay profile). Some of these channel conditions will produce significant performance degradation that might lead to a voice-quality metric below the desired threshold level (3 in this example). The outage probability for a given value of E_b/N_0 is estimated by the ratio of the number of simulated channels that produce a voice-quality metric lower than the threshold divided by the total number of channels simulated. As E_b/N_0 is increased the outage probability will decrease, and by repeating the simulations for different values of E_b/N_0 we can find

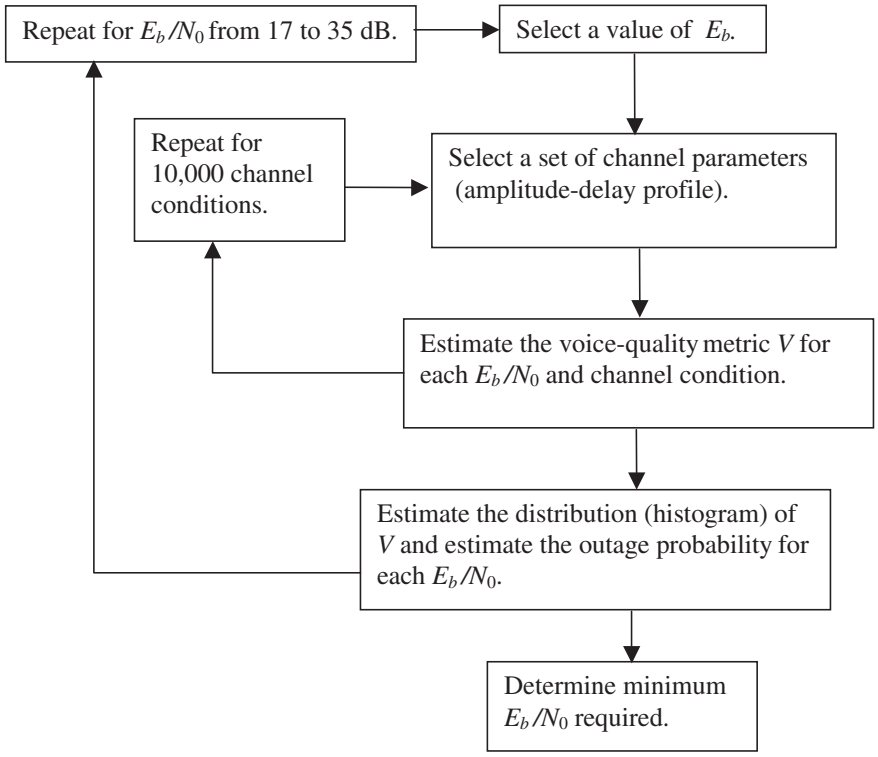


Figure 11.2 Flowchart for estimating outage probability.

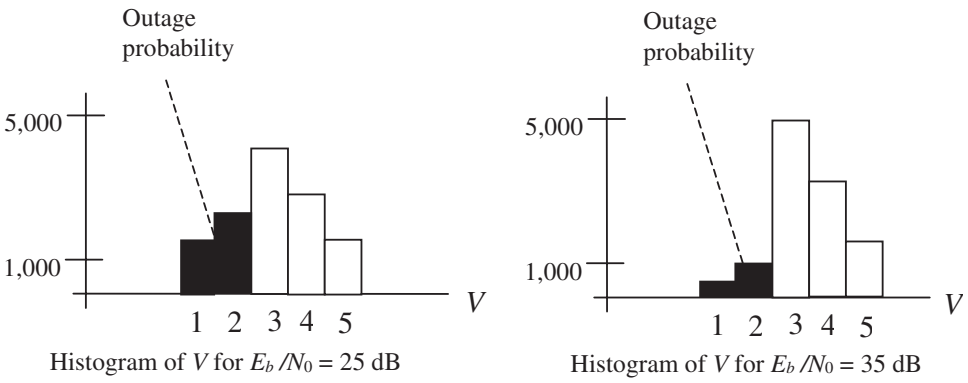


Figure 11.3 Typical histogram in outage probability calculation.

the (minimum) value of E_b/N_0 that guarantees an outage probability less than 2%. Let us assume that the range of values for E_b/N_0 to be considered is given as 17 to 35 dB in increments of 2 dB. Let us also assume that for each E_b/N_0 we need to simulate 10,000 channel conditions in order to obtain the distribution (histogram) of V and the outage probability. Typical results are illustrated in Figure 11.3. It can be seen from the histograms that the outage probability is less for $E_b/N_0 = 35$ dB than for $E_b/N_0 = 25$ dB.

For a given channel condition and E_b/N_0 we can estimate the voice-quality metric using a brute-force Monte Carlo approach in which we use sampled and digitized voice as the input, record the simulated output of the voice decoder, and play the recorded output to a set of human subjects and determine the voice-quality metric based on their scores. While this approach mimics reality, it is not practical to repeat this for thousands of channel conditions and many values of E_b/N_0, for even if we have the computer resources to do the simulations, this approach will require each listener to score thousands of voice segments.

A better approach is be to divide (partition) the problem into parts and simulate the parts separately. In order to arrive at an efficient partitioning scheme, let us consider the influence of different portions of the communication system on the overall performance as measured by the voice-quality metric. With respect to Figure 11.4, the waveform "processing" part of the system accepts a binary sequence at point C and produces a binary sequence for hard decision decoding (or quantized values for soft decision decoding) at point F. The probability of error q (or the transition probabilities for soft decision decoding) for this analog portion of the system, which we will call "the waveform channel," depends on the channel parameters and E_b/N_0. This probability of error (or a set of transition probabilities for soft decision decoding) can be estimated via a Monte Carlo or semianalytic technique with a random binary sequence as the input. It is not necessary to drive this part of the simulation with encoded voice bits.

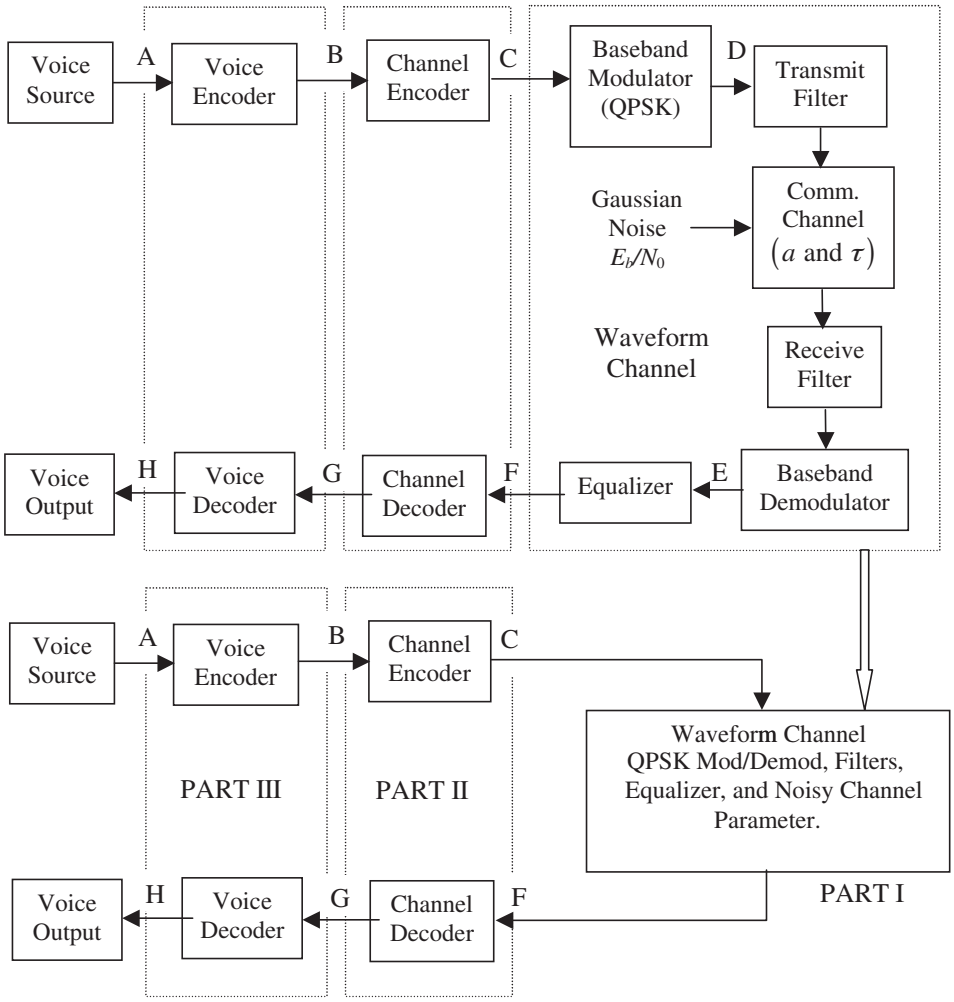


Figure 11.4 Simplified and partitioned simulation model.

The next segment of the system includes the error control encoder and decoder, which accepts a sequence of binary digits at point B and produces a sequence of binary digits at point G. The probability of error between points B and G will strictly be a function of the probability of error q (or the set of transition probabilities) in the waveform channel. Indeed, since the errors in the waveform channel are the result of additive, white, Gaussian noise (AWGN), we can assume the error pattern is an independent sequence and hence, as far as the evaluation of the coded bit error probability between points B and G are concerned, the waveform channel can be replaced by a binary random number generator that produces 1's and 0's with

probabilities q and $1-q$ with 1 representing a transmission error in the waveform channel. The coded error probability P_E can be evaluated via a Monte Carlo simulation in which the input to the encoder is a random binary sequence and the entire waveform channel is replaced by a binary random number generator. There is no need to drive this part of the simulation with encoded voice bits.

The performance of the error control coding can also be evaluated using a semi-analytic approach that maps the uncoded error probability q to the coded error probability P_E. The technique for accomplishing this was explored for both block and convolutional codes as discussed in Chapter 8. With this approach we can map the distribution of q as a function of E_b/N_0 to the distribution of P_E as a function of E_b/N_0.

The final step in the estimation of outage probability is the estimation of the distributions of the voice-quality metric V for various values of E_b/N_0. The voice-quality metric will depend on P_E (which itself depends on E_b/N_0) and the distribution of V as a function of P_E. Hence the distribution of V as a function of E_b/N_0 can be obtained by evaluating the voice-quality metric for different values of P_E. This evaluation can be done independent of the first two parts of the problem; all we have to do is to evaluate the performance of the voice coder and encoder for different values of the error probability P_E. This is best done using the actual voice encoder decoder chip set, running digitized voice through them, and evaluating the voice quality at the output of the encoder as a function of P_E. The effect of the entire system between points B and G is emulated by injecting random errors at the rate of P_E between the output of the voice encoder and the input of the voice decoder. This part of the voice-quality metric evaluation has to be done for only about a dozen values of P_E, say, from 10^{-1} to 10^{-7}, and the listener has to score the voice quality for each of these dozen values of P_E, which is much simpler than having to score the voice quality for thousands of channel conditions in the direct Monte Carlo simulation of the entire system.

Given the estimate of the voice-quality metric V as a function of P_E obtained from Part III, the distribution of P_E as a function of q in Parts II and III, and the distribution of q as a function of E_b/N_0 obtained in Part I, we can obtain V as a function of E_b/N_0. From

$$V\left(E_b/N_0\right) = V\left(P_e(q\left(E_b/N_0\right))\right) \tag{11.5}$$

we can estimate the *distribution* of V and the outage probability for each value of E_b/N_0. From a plot of the outage probability versus E_b/N_0 we can determine the minimum value of E_b/N_0 needed to assure an outage probability less than 2% at a voice quality less than 3. The overall approach for outage probability estimation is summarized in Figure 11.5.

We now turn our attention to the details of each of the three parts of performance evaluation starting with the estimation of the error probability for the waveform channel. This is the most computationally intensive part, since it has to be repeated for 10 values of E_b/N_0 and 10,000 channel conditions. The other two parts dealing with the mapping of the error probability q for the waveform channel to the coded

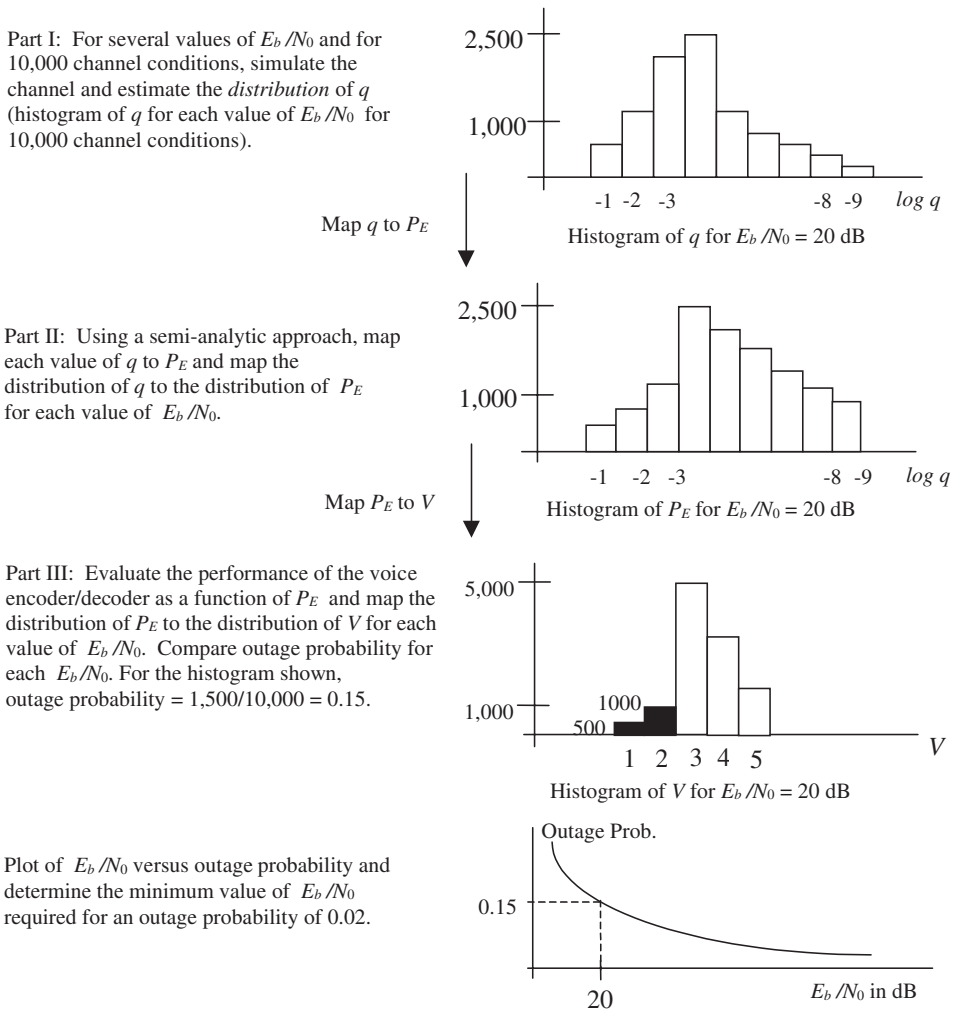


Figure 11.5 Partitioned methodology for outage probability estimation.

error probability P_E and the voice-quality metric V are repeated only once for each of approximately 10 values of q (10 values of E_b/N_0).

11.2.1 Methodology for Simulation of the Analog Portion of the System

The simulation model for the waveform channel (analog portion of the system) is shown in Figure 11.6. The main objective of the simulation is to obtain the distribution (histogram) of the probability of error q for 10 different values of E_b/N_0.

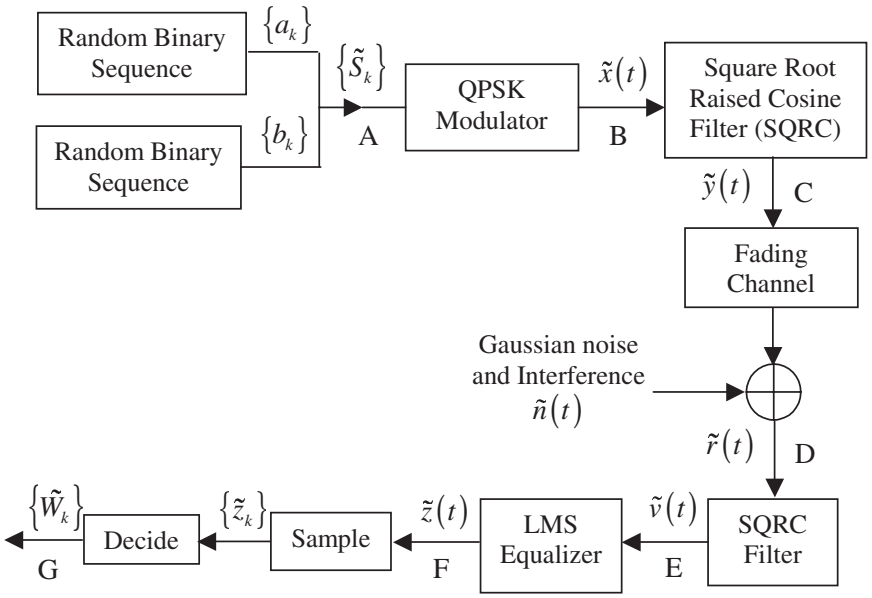


Figure 11.6 Simulation model for analog portion of the system.

For each value of E_b/N_0, 10,000 snapshot conditions of the channel are simulated and the histogram of q is obtained from the estimated BER for each channel. During each simulation, the channel condition remains fixed.

We describe the salient features of the simulation model before presenting computationally efficient techniques for simulating this portion of the system.

Details of the Simulation Model

Input: The input to the system consists of two random binary sources each with a bit rate of 14,400 bits/second (combined rate of 28,800 bits per second), which represents the bit stream coming from the error control encoder. The two bit sequences a_k and b_k are combined to produce a complex QPSK symbol sequence $\widetilde{S}_k = A_k + jB_k$, where A_k and B_k are mappings of the binary sequences a_k and b_k into amplitude sequences of +1 or - 1. The QPSK symbol sequence is mapped to a complex valued QPSK waveform

$$\widetilde{x}(t) = \sum_{k=-\infty}^{\infty} A_k p(t - kT) + j \sum_{k=-\infty}^{\infty} B_k p(t - kT) \tag{11.6}$$

which is sampled at a rate of 12 samples per symbol to create a sampled version of the QPSK waveform

$$\widetilde{x}\left(mT_s\right) = \sum_{k=-\infty}^{\infty} A_k p\left(mT_s - kT\right) + j \sum_{k=-\infty}^{\infty} B_k p\left(mT_s - kT\right) \tag{11.7}$$

where T is the duration of the QPSK symbol, $p(t)$ is a rectangular pulse of unit amplitude, and duration T. We assume

$$p\left(t\right) = \begin{cases} 1, & 0 < t \leq T \\ 0, & \text{elsewhere} \end{cases} \tag{11.8}$$

(The two random binary sources in Figure 11.6 may be combined into a single random binary source operating at the combined rate and the two separate sequences can be obtained at the output by taking the odd and even numbered bits of the output.)

Transmit and Receive Filters: The transmit and receive filters are square root raised cosine (SQRC) filters with the transfer function

$$H_T\left(f\right) = H_R\left(f\right) = \begin{cases} \sqrt{T}, & |f| \leq \frac{1-\beta}{2T} \\ \sqrt{\frac{T}{2}\left[1 - \sin\frac{\pi T}{\beta}\left(f - \frac{1}{2T}\right)\right]}, & \frac{1-\beta}{2T} \leq |f| \leq \frac{1+\beta}{2T} \\ 0, & |f| \geq \frac{1+\beta}{2T} \end{cases} \tag{11.9}$$

where $\beta = 0.5$ for a 50% roll-off [1]. These filters are optimum in the sense that they produce a finite bandwidth waveform with zero ISI and also produce optimum BER performance over AWGN channels. It is customary to include a $1/sinc$ function in the transfer function of the transmit filter in order to compensate for the fact that the QPSK waveform at filter input is a rectangular non-return-to-zero (NRZ) pulse waveform rather than an impulse waveform. The filter transfer function given in the preceding equation will produce a response with zero intersymbol interference (ISI) only when the input is a sequence of impulses. Instead of including a $1/sinc$ function, we can use an impulse sequence representation of the QPSK waveform; in this case, only the first of the 12 samples for the k^{th} QPSK symbol has the value $A_k + jB_k$ and the remaining 11 samples are zeros.

The SQRC filters are implemented as finite duration impulse response (FIR) filters, since an infinite duration impulse response (IIR) implementation will be very difficult since the transfer function is not given in pole-zero form in the s (Laplace transform) domain as discussed in Chapter 5. We will assume that an impulse invariant transformation with time-domain convolution is used for each filter. The impulse response of the SQRC filter is given by [1]

$$h_T(t) = h_R(t) = 8\beta \frac{\cos\left[\left(R + 2\beta\right)\pi t\right] + \sin\left[\left(R - 2\beta\right)\pi t\right]\left(8\beta t\right)^{-1}}{\pi\sqrt{R}\left[1 - \left(8\beta t\right)^2\right]},$$
$$-\infty < t < \infty \tag{11.10}$$

Since this is clearly a noncausal filter, the impulse response is truncated to a length of four symbols on either side of zero yielding a truncated duration of eight symbols. Shifting the resulting impulse response by four symbols then yields a causal time function.

Channel: The quasi-static channel model defined in (11.2) is characterized by two random variables $\widetilde{a}$ and τ. Each simulation is executed with fixed values of $\widetilde{a}$ and τ drawn from a Raleigh and uniform distribution, respectively. The value of τ is rounded off to an integer number of samples, say, r, and the simulation model for the channel consists of a direct path and a delayed path with a delay of r samples and attenuation $\widetilde{a}$. This model is trivial to implement.

Equalizer: The SQRC filters will produce zero ISI only when the channel transfer function is ideal over the bandwidth of the signal (which in this example is 0.75 times the symbol rate). Since the channel in this case is nonideal, some residual ISI will be present in the system and this residual ISI can be minimized by the use of an equalizer in the receiver. While a wide variety of equalizers are available, we chose to include a 9 tap, synchronously spaced LMS (linear minimum mean squared error) equalizer to illustrate several aspects of methodology.

A gradient algorithm is normally used for iteratively adjusting the equalizer weights. If the equalizer convergence is simulated, this has to be done via a Monte Carlo simulation using a training sequence for the input and with noise samples injected during the simulation. Since the LMS equalizer is a linear filter, and the noise at the *input* to the receiver is AWGN, the noise at the *output* of the equalizer will be additive and Gaussian, and hence a semianalytic technique can be used for error probability estimation. For BER estimation, we need to simulate only the effects of ISI distortion, and the effects of additive white Gaussian noise can be handled analytically without having to do a Monte Carlo simulation with noise samples.

We could consider two approaches for handling the equalizer when using a semianalytic technique for BER estimation. We can run a short Monte Carlo simulation in the beginning, with noise samples included, wait until the equalizer weights converge, and then "freeze" the equalizer weights and execute the performance estimation simulation with the noise source turned off.

The second approach that could be used for the equalizer is based on the well-known fact that the equalizer weights will converge to a weight vector whose value can be computed analytically according to

$$\widetilde{W} = \Gamma^{-1} R^* \tag{11.11}$$

where $\widetilde{W}$ is the weight vector, Γ is the "channel covariance matrix," and R is a vector of sampled values of the unequalized impulse response of the system from the input to the transmit filter to the output of the receive filter [2]. This unequalized impulse response, sampled at the symbol rate, can be obtained from a calibration run in which a unit impulse is applied at the transmit filter input and the impulse response is recorded at the output of the receive filter. The sampled values of the impulse response are used to compute the autocorrelation function of the unequalized impulse response and the entries in the matrix Γ are obtained from

the values of the autocorrelation function. Diagonal entries in Γ will include the autocorrelation value at zero lag plus the noise variance at the input to the equalizer, which can be computed knowing the input noise PSD and the noise bandwidth of the receive filter. With this approach, the equalizer weights can be computed prior to the simulation for BER estimation as part of the "calibration process" and the equalizer can be treated as an FIR filter during the BER simulations.

When we use a direct Monte Carlo simulation for performance estimation, the noise source will be "on," and hence the iterative (gradient) method is used at the beginning to let the equalizer weights converge. The weights are then frozen during performance estimation. (If the semianalytical method is used for BER estimation, the noise source will be turned off during the semianalytical BER estimation phase.)

Pure Monte Carlo Approach to Performance Estimation

In the direct Monte Carlo approach, the input and noise processes are explicitly simulated. An estimate of the error rate for each E_b/N_0 and channel condition is obtained by counting the number of errors between the symbol sequence at the input to the modulator $\widetilde{S}_k$ and at the output of the decision device $\widetilde{W}_k$. While the equalizer can provide amplitude normalization (which is not necessary with QPSK modulation) and can also compensate for phase offsets, a calibration run must be executed at the beginning to establish a timing reference for the equalizer and for lining up the input and output symbol sequences. Also, an initial training sequence might have to be used to aid in the equalizer convergence and the error rate estimation should start only after the equalizer weights have converged and are frozen. The essential steps in the Monte Carlo simulation are as follows:

1. Draw a set of $\widetilde{a}$ and τ and start with the initial value of E_b/N_0.
2. Execute a calibration run to establish a timing reference for the equalizer and for lining up the input and output for error counting.
3. Train the equalizer and freeze the weights (noise source turned on with the variance value computed from E_b/N_0).
4. Start the Monte Carlo simulation for performance estimation and run the simulation until about 50 errors are counted.
5. Repeat for all values of E_b/N_0 and 10,000 channel conditions.
6. Compute the histogram of q for each value of E_b/N_0.

While the direct Monte Carlo approach is simple to implement in principle, it does require long simulation runs for each value of E_b/N_0 and channel condition. Even if each simulation takes only a few seconds of CPU time, the total effort required to repeat the simulations for 10,000 channel conditions and 10 values of E_b/N_0 might be overwhelming.

Since the receiver is linear (an LMS equalizer is an FIR filter), and the noise is additive and Gaussian at the receiver input, the noise at the output will also be

additive and Gaussian. Hence we can use the semianalytic approach for performance estimation.

Semianalytic Approach to Performance Estimation

The BER in the system will be a function of intersymbol interference and additive Gaussian noise, the effects of which can be handled analytically. Hence only the ISI produced by the cascade of the transmit filter, the channel, the receive filter, and the equalizer is simulated. The BER is estimated (assuming that the transmitted signal constellation point is (1,1), which maps to (A,A) at the equalizer output as shown in Figure 11.7) using the semianalytic approach described in the previous chapter. This gives

$$\begin{aligned}\widehat{q} &= \frac{1}{M}\sum_{i=1}^{M}\{1-\{[1-Q\left(A+d_{xi}/\sigma_x\right)]\left[1-Q\left(A+d_{yi}/\sigma y\right)\right]\}\} \\ &\approx Q\left(A+d_{xi}/\sigma_x\right)+Q\left(A+d_{yi}/\sigma_y\right) \end{aligned} \tag{11.12}$$

where d_{xi} and d_{yi} are the direct and quadrature components of ISI associated with the i^{th} simulated symbol, σ_x^2 and σ_y^2 are the variances of the direct and quadrature components of the noise at the output of the equalizer, and M is the number of symbols simulated.

The values of σ_x^2 and σ_y^2 are computed using

$$\sigma_x^2 = \sigma_y^2 = N_0 B_N \tag{11.13}$$

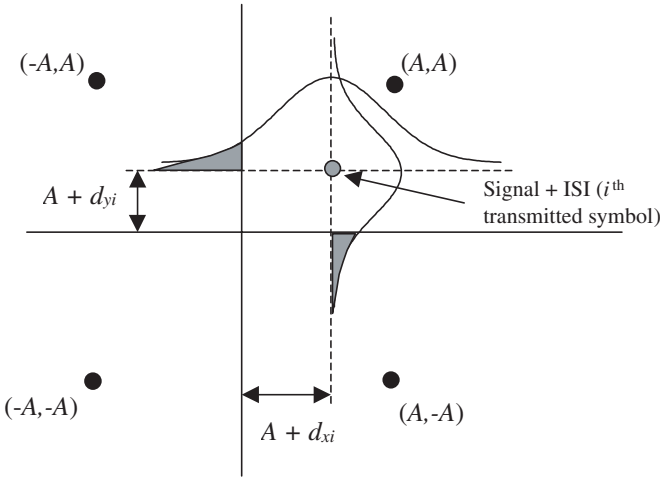


Figure 11.7 Semianalytic BER estimation for the QPSK signal.

where $N_0/2$ is the PSD of the two-sided bandpass noise at the input to the receive filter and B_N is the noise bandwidth of the receive filter and the equalizer together. The noise bandwidth is computed from a calibration run as outlined earlier.

The steps in applying semianalytic techniques for performance estimation are as follows:

1. Initialization: Chose an initial value for E_b/N_0 and the channel parameters.
2. Calibrations and equalizer weight determination:
   - Establish a timing reference for the equalizer and the overall time delay.
   - Obtain the unequalized impulse response via simulation by injecting an impulse at the input A and compute the weight vector for the equalizer using (11.11).
   - Compute the noise bandwidth of the receive filter and the equalizer and calibrate the variance of the noise at the output using (11.13).
3. Simulation: Simulate M symbols and estimate the BER according to (11.12).
4. Repeat for 10,000 channels and 10 values of E_b/N_0 and compute the histogram of q.

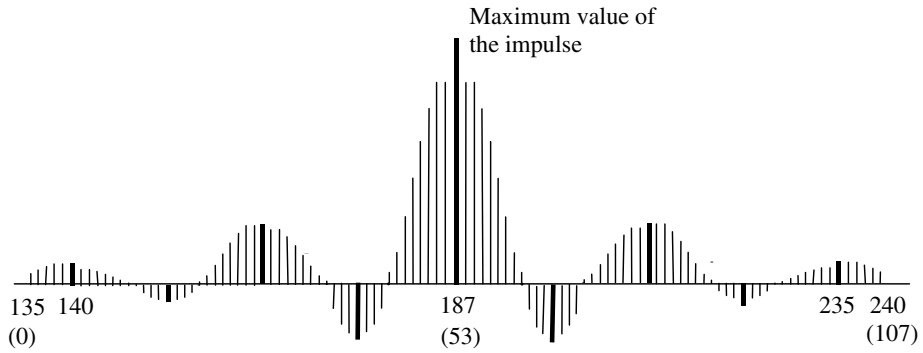
Faster Semianalytic Technique

The semianalytic error rate estimation can be speeded up considerably by combining all the blocks, the transmit filter, the channel, the receive filter, and the equalizer (after weights have been computed and set) into one single block, since all of them are linear time-invariant components. Since no noise samples are injected, these components of the system, which are in cascade, process the QSPK waveform signal in a pipeline fashion. From a performance estimation point of view we are simply interested in the waveform at the output of the equalizer. Since we are not interested in the waveforms at the outputs of the other blocks in the system, there is really no need for processing the input waveform through each individual block. By combining all the blocks into one and processing the input waveform, the equivalent representation will be computationally very efficient.

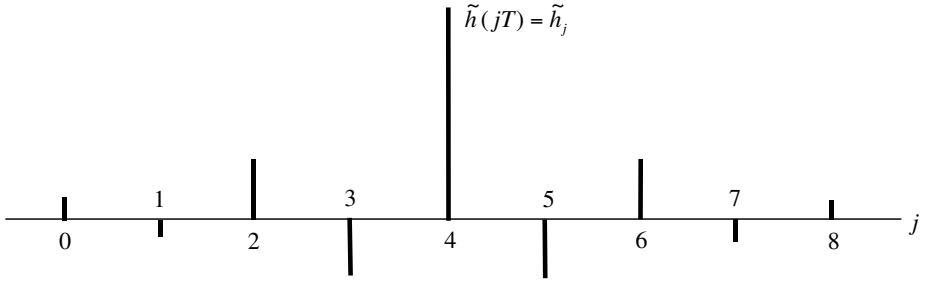
The overall impulse response of the system is

$$\widetilde{h}\left(kT_s\right) = \widetilde{h}_T\left(kT_s\right) * \widetilde{h}_c\left(kT_s\right) * \widetilde{h}_R\left(kT_s\right) * \widetilde{h}_{eq}\left(kT_s\right) \tag{11.14}$$

and this response can actually be obtained via simulation by injecting an impulse at point A and measuring the impulse response at the output of the equalizer (point F in Figure 11.4). The overall impulse response can be truncated and the entire system can be simulated as a single FIR filter. An example of the overall impulse response is shown in Figure 11.8. Note the delay through the system is approximately 135 samples and the impulse response can be truncated to 108 samples from sample number 135 to 242. The impulse response is assumed to be zero outside this interval. The time index for the nonzero values of the impulse response are renumbered from 0 to 107 for notational convenience.



(a) Truncated impulse response sampled at 178,200 samples per second (12 samples/sym)



(b) Truncated impulse response sampled at the *symbol* rate (1 sample/sym)

Figure 11.8 Truncated impulse response sampled at 12 samples per symbol and 1 sample per symbol.

A detailed waveform level simulation of the model will be executed at the sampling rate of $r_s = 172,800$ samples per second according to the equation

$$\widetilde{z}(mT_s) = \sum_{p=0}^{107} \widetilde{h}(pT_s)\,\widetilde{x}((m-p)\,T_s) \tag{11.15}$$

where $\widetilde{x}(mT_s)$ are the sampled values of the QPSK waveform at the input to the transmit filter, $\widetilde{z}(mT_s)$ is the output of the equalizer, and $\widetilde{h}(pT_s)$, $p = 0$ to 107 are the truncated values of the overall impulse response. The output of the equalizer is sampled starting at sample number 187 (why) and once every 12 samples afterward to produce the decision metric $\widetilde{z}_k$, and a decision $\widetilde{W}_k$ (estimate of the transmitted symbol) is made based on the value of the decision metric. As far as performance estimation is concerned, we are interested only in every 12^{th} sample (one every symbol) of the equalizer output, corresponding to decision times, and the intervening samples are of no interest or use. Since the equalizer operates with a tap spacing of

12 samples (or one symbol duration T), we can write an expression for the decision metric using every 12^{th} sample of the impulse response (see Figure 11.8) as

$$\widetilde{z}_i = \sum_{j=0}^{8} \widetilde{h}_j \widetilde{S}_{i-j} \tag{11.16}$$

Note that (11.16) is the entire simulation model for semianalytic error rate estimation; we simply generate a sequence of QPSK symbols and process them through (11.16). This requires only eight operations per symbol to produce the values of the decision metric, which represents the input symbol with additive ISI. The semianalytic error computation given in (11.12) is applied to the sequence $\widetilde{z}_i$.

It is easy to see that the model given in (11.16) will be about two orders of magnitude faster than the model given in (11.15), since we now compute the output only once every 12^{th} sample and each output sample requires only 8 multiply and add operations, as opposed to 108 multiply and add operations for the model given in (11.15). Compared to processing the QPSK waveform though each block, the combined model in (11.15) will be a factor about three to four times faster, since we have combined four blocks. Thus, the overall computational savings of the simulation model given in (11.16) could be of the order of 1,000 compared to the direct approach wherein we simulate the evolution of waveforms through each block in the simulation model on a sample by sample basis.

The simulation model given in (11.16), coupled with the semianalytic approach for BER estimation, will be computationally very efficient. The overall memory length of the system is nine symbols and hence a PN sequence, having a period of $2^9 = 512$ symbols, will be adequate to produce all possible ISI values except, of course, the all-zero sequence. Thus, after calibration, each performance simulation run consists of generating 512 QPSK symbols, computing the 512 output samples according to (11.16), and applying the semianalytic estimation given in (11.16). The essential steps in the fast analytic error rate estimation can be summarized as follows:

1. Initialization: Chose an initial value for E_b/N_0 and the channel parameters.

2. Calibrations and equalizer weight determination:

   - Establish a timing reference for the equalizer and the overall time delay.
   - Obtain the unequalized impulse response via simulation by injecting an impulse at the input (point A) and compute the weight vector for the equalizer using (11.11).
   - Compute the noise bandwidth of the receive filter and the equalizer and calibrate the variance of the noise at the output using (11.13).
   - Obtain the equalized pulse response sampled at the symbol rate by injecting an impulse at the input (point A) and sampling the impulse response at the equalizer output once every symbol (Figure 11.8).

3. Simulation: Generate $M = 512$ QPSK symbols, process them according to (11.16), and estimate the error probability according to (11.12).
4. Repeat for 10,000 channels and 10 values of E_b/N_0 and compute the histogram of q.

Moment Method for BER Estimation

While the fast semianalytic technique described in the preceding section is computationally very efficient, the computational load increases significantly if the memory length of the system and/or the order of the modulation scheme increases. The increase in memory length results, since the number of symbols that need to be simulated for all possible ISI values increase according to $M = m^L$, where L is the memory length of the system and m is the alphabet size (2 in the binary case). When m and L are large, the simulation length will be very long. In such cases we can use another method to reduce the computational burden.

The basic computation performed in (11.12) can be expressed

$$\widehat{q} = E\left\{Q\left(A + D_x/\sigma_x\right)\right\} + E\left\{Q\left(A + D_y/\sigma_y\right)\right\} \tag{11.17}$$

where the expectations are taken with respect to D_x and D_y, which are the random variables that represent the direct and quadrature phase components of the ISI. Rather than estimating this expected value by averaging over simulated values of ISI, we can compute the moments of the ISI, approximate the distribution of D_x and D_y using the computed moments of the ISI, and then perform the expected value operation using the approximate distribution of ISI.

Let us consider approximating the distribution of D_x using its moments. From (11.16), we can write the ISI term D_x as

$$D_x = \sum_{\substack{j=0 \\ j\neq 0}}^{8} \alpha_j A_j - \sum_{j=0}^{8} \beta_j B_j \tag{11.18}$$

where α_j and β_j are the real and imaginary components of the impulse response (i.e., $\widetilde{h}_k = \alpha_k + j\beta_k$), and $A_j = \pm 1$, $B_j = \pm 1$ are the real and imaginary components of the QPSK symbol sequence. The moments of D_x can be computed according to

$$E\left\{D_x^k\right\} = E\left\{\left[\sum_{\substack{j=0 \\ j\neq 0}}^{8} \alpha_j A_j - \sum_{j=0}^{8} \beta_j B_j\right]^k\right\}, \qquad k = 1, 2, 3, \ldots \tag{11.19}$$

Note that in the preceding equation the values of the impulse response are constants whose values are known and A_j and B_j are independent binary random variables with values ± 1. The odd moments of A_j and B_j are zero and even moments are 1. Hence the computation of the moments of D_x involves a simple expansion of the binomial sum in (11.19). We compute the expected values term by term, and add them.

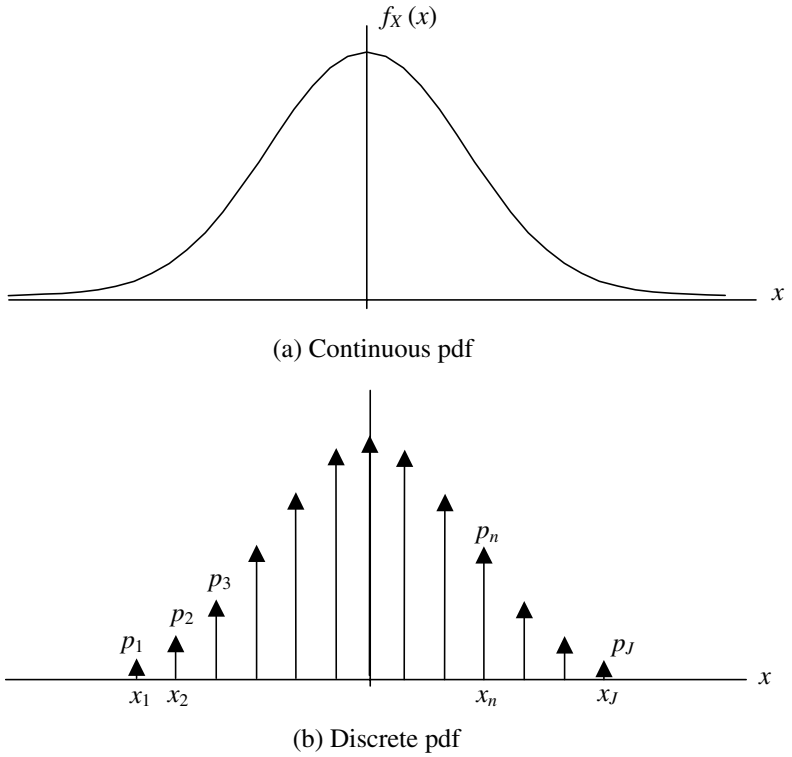
From the moments of D_x we can obtain a discrete approximation to the distribution of D_x. In this approximation, D_x is treated as a discrete random variable with J values $d_1, ..., d_J$, with probabilities $p_1, ..., p_J$. As a simple example of the principle, consider a discrete approximation of a Gaussian pdf as shown in Figure 11.9.

The abscissas $x_1, ..., x_J$ and the ordinates $p_1, ..., p_J$ are chosen such that the continuous distribution and the discrete approximation yield the same moments

$$E\left(X^n\right) = \mu_n = \sum_{k=1}^{J} x_k^n p_k \tag{11.20}$$

Given the first $2J$ moments $\mu_1, ..., \mu_{2J}$ of X, we can solve the set of $2J$ nonlinear equations

$$\mu_n = \sum_{k=1}^{J} x_k^n p_k, \qquad n = 1, 2, ..., 2J \tag{11.21}$$



(a) Continuous pdf

(b) Discrete pdf

Figure 11.9 Example of a discrete approximation of a Gaussian pdf.

for the values of x_k and p_k, $k = 1, 2, \cdots, J$. Details of the techniques used to derive the discrete approximation of a distribution using its moments may be found in [3, 4].

Using the discrete approximation for the ISI distribution we can compute $E\{Q(A + D_x/\sigma_x)\}$ as

$$E\{Q(A + D_x/\sigma_x)\} \approx \sum_{j=1}^{J} Q(A + d_j/\sigma_x)\, p_j \tag{11.22}$$

The second term in (11.17) can be computed using a similar procedure.

Note that, in this method, there is no Monte Carlo simulation at all for performance estimation! It is entirely analytic except for two *single event simulations*; one for obtaining the unequalized pulse response from which the equalizer weights are computed and a second pulse response simulation to obtain the equalized pulse response sampled at the symbol rate. The moments of the ISI distribution are *computed* using (11.19) and the discrete approximation of the distribution of ISI and the probability of error are also *computed*. In addition to the two single event simulations, calibration runs have to be executed for establishing the timing references for sampling the unequalized and equalized pulses and for calculating the noise bandwidth of the receive filter.

The computational efficiency of the moment method will depend on the number of moments needed to obtain a good approximation of the ISI distribution and the computational load associated with computing the moments and the moment-based approximation. The latter will be a function of the length of the impulse response, which in many cases can be truncated to 10 or so symbols. Good approximations of the ISI distribution can be obtained from the first six or eight moments.

The moment method is very useful if higher-order modulation schemes such as 256 QAM are used. In this case the direct and quadrature phase waveforms have 16 amplitude levels, and if the memory length is 10, then we need a 16-ary PN sequence of length 16^{10} symbols to simulate all possible ISI values. Hence the semianalytic method will require a long simulation, whereas the moment method in this case will be computationally much more efficient. The key steps in applying the moment method are summarized below:

1. Initialization: Chose an initial value for E_bN_0 and the channel parameters.
2. Calibration and equalizer weight determination:
   - Establish a timing reference for the equalizer and the overall time delay.
   - Obtain the unequalized impulse response via simulation by injecting an impulse at the input A and compute the weight vector for the equalizer [Equation (11.11)].
   - Compute the noise bandwidth of the receive filter and the equalizer and calibrate the variance of the noise at the output using (11.13).
   - Obtain the equalized pulse response sampled at the symbol rate by injecting an impulse at the input (point A in Figure 11.4) and sampling the impulse response at the equalizer output once every symbol (Figure 11.8).

3. BER *Computation*:
   - Compute the moments and moment based approximation of the distribution of ISI.
   - Compute the BER according to (11.22).
4. Repeat for 10,000 channels and 10 values of E_b/N_0 and compute the histogram of q.

11.2.2 Summary of Methodology for Simulating the Analog Portion of the System

In this section we illustrated many important aspects of the methodology for simulating the waveform processing portion of wireless communication systems operating over slowly fading channels. Several approaches to simplifying the simulation problem were discussed and a number of performance estimation techniques were also presented. These techniques range from pure Monte Carlo, to partial Monte Carlo, to a totally analytic method.

In any performance estimation simulation, a considerable amount of up-front effort must be expended for simplifying the simulation model and for examining and evaluating the various approaches that are possible. These efforts will lead to tremendous computational savings during performance estimation simulations. While many of the details discussed in the context of the example presented in this section may not be directly applicable to other problems, the overall methodology presented in this example should suggest the range of factors that should be considered before reaching the final choice of a performance estimation procedure.

It should be obvious by now that calibration and single-event simulations for measuring pulse and impulse responses play an important role prior to any performance estimation. All of these calibrations do not have to be repeated for each situation. For example, if the channel condition is fixed, the unequalized pulse response does not have to be measured for each value of E_b/N_0.

11.2.3 Estimation of the Coded BER

The next step in outage probability estimation is to obtain the coded bit error probability $P_E(E_b/N_0)$ from the uncoded bit error probability $q(E_b/N_0)$. This can be accomplished semianalytically using the transfer function bound for the convolutional code used for error correction. This technique was discussed in Chapter 8.

11.2.4 Estimation of Voice-Quality Metric

The last step in outage probability estimation is the mapping of the coded error probability to the voice-quality metric. This is accomplished by injecting binary errors between the output of the voice coder and the input to the decoder and scoring the quality of the resulting voice output. By repeating the listening tests for various values of the injected error rate, we can establish the relationship between the error rate P_E and the voice-quality metric V. An example is given in Table 11.1.

Table 11.1 Relationship Between Error Probability and Voice Quality

P_E	Voice Quality V
$> 10^{-1}$	1
10^{-1} to 5×10^{-3}	2
5×10^{-3} to 10^{-4}	3
10^{-4} to 10^{-6}	4
$< 10^{-6}$	5

Note that for this part of the performance evaluation process, the derivation of the relationship between P_E and V can be carried out independent of the previous steps. Also, this table could have been obtained from the manufacturer of the voice coder/decoder chip set.

From the preceding table it is clear that the outage probability $P(V < 3)$ can be expressed in terms of the distribution of P_E as

$$\Pr(V < 3) = \Pr\left\{P_e > 5 \times 10^{-3}\right\} \tag{11.23}$$

From the analytic bounds which relate P_E to q we can establish the value of q, say, q_0, which yields $P_E > 5 \times 10^{-3}$. The system outage probability then is equal to $P(q > q_0)$, which can be obtained for different values of E_b/N_0 from the distribution of the BER q for the analog portion of the channel. The outage probability $P(q > q_0)$, will decrease as E_b/N_0 is increased, and by plotting the outage probability as a function of E_b/N_0, we can determine he minimum value of E_b/N_0 needed to maintain $P(q > q_0) < 0.02$.

11.2.5 Summary of Overall Methodology

We now summarize the major steps in the methodology used for estimating outage probability in a wireless communication system:

1. Determine the relationship between the coded error probability P_E and the voice-quality metric V by simulating the voice encoder/decoder for different values of P_E.

2. Compute the bounds that relate the uncoded probability of error q to the coded probability of error P_E.

3. Simulate the analog portion of the system for various values of E_b/N_0 and 10,000 channel conditions and obtain the distribution of q for different values of E_b/N_0.

4. Map the distribution of q to the distribution of V and for different values of E_b/N_0 and determine the value of E_b/N_0 needed to maintain the specified outage probability.

11.3 Summary

In this chapter we described the methodology used for estimating system-level performance metrics such as outage probabilities in a typical wireless communication system operating over a slowly fading channel. This problem requires extensive simulations for several thousand channel conditions, and brute-force Monte Carlo simulation is simply not feasible because of the computational burden involved.

Given the computational burden required for simulating the system for many thousands of channel conditions, we need to carefully look at simplifying the simulation model, as well as at alternate techniques for estimating the error rate in the system. Several approaches were discussed in this chapter for reducing the computational burden associated with this problem, such as

1. Using a hierarchical approach and partitioning the problem so that the complexity of simulations grow linearly rather than in a multiplicative manner for various combinations of parameter values.

2. Simplifying the simulation model by combining various functional blocks and looking at alternate approaches for simulating the behavior of individual components (e.g., the channel covariance inversion technique for simulating the equalizer).

3. Using semianaytical techniques when possible.

4. Running the simulations at symbol rate rather than at a rate of 8 to 16 samples/symbol.

5. Using moment based approximations for some of the pdfs involved in the semianalytical procedure.

6. Reducing the entire simulation run to a small number of single-event simulations and using analytical computations instead of Monte Carlo simulations for estimating performance metrics.

7. Using a semianalytical technique for obtaining P_E from q.

While the techniques discussed in this chapter may not be directly applicable to a different simulation problem, the overall methodology clearly illustrates the need to think through and carefully design the simulation experiment. By doing so, it is possible to reduce the computational burden significantly.

11.4 Further Reading

An excellent example of the application of the methodology discussed here can be found in

M. C. Jeruchim, P. Balaban, and K. S. Shanmugan, *Simulation of Communication Systems*, 2nd ed., New York: Kluwer Academic/Plenum Publishers, 2000, Chap. 12.

11.5 References

1. G. L. Stuber, *Principles of Mobile Communication*, Boston: Kluwer Academic Publishers, 1996.

2. J. Proakis, *Digital Communications*, 3rd ed., New York: McGraw-Hill, 2000.

3. M. Abramowitz and I. Stegun, *Handbook of Mathematical Functions*, Mineola, NY: Dover, 1972.

4. A. H. Stroud and D. Secrest, *Gaussian Quadrature Formulas*, Englewood Cliffs, NJ: Prentice Hall, 1966.

11.6 Problems

11.1 Consider a binary baseband system in which the received signal at sampling instants consists of $y_k = A_k + D_k + n_k$ where $\{A_k\}$ is the transmitted symbol sequence, D_k is the intersymbol interference, and n_k is the noise. Assume the symbol sequence to be independent and $A_k = \pm 1$. The ISI term is given by

$$D = D_k = \sum_{j=1}^{5} h_j A_{j-k}$$

where the sampled values of the overall impulse response of the system that contribute to ISI are tabulated below:

h_1	h_2	h_3	h_4	h_5
0.20	0.10	0.08	0.07	0.05

(a) Find the exact distribution of the ISI values.

(b) Find first 10 moments of the ISI.

(c) Find a moment matching approximation to the ISI distribution using three values of ISI in the approximation.

(d) Compare the probability of error using the exact distribution of the ISI values and the moment matching approximation.

11.2 Repeat Problem 11.1 for a QPSK signal where the I and Q symbol sequences $\{A_k, B_k\}$ are made up of independent binary values ± 1 and the overall complex lowpass equivalent response of the system is given in the table below:

$\widetilde{h}_1$	$\widetilde{h}_2$	$\widetilde{h}_3$	$\widetilde{h}_4$	$\widetilde{h}_5$
$0.20 + j0.08$	$0.10 + j0.06$	0.08	0.07	0.05

where

$$\widetilde{y}_k = \widetilde{S}_k + \widetilde{D}_k + \widetilde{n}_k$$

$$\widetilde{S}_k = A_k + B_k$$

and

$$\widetilde{D}_k = \sum_{j=1}^{5} \widetilde{h}_j S_{k-j}$$

11.3 Develop a recursive formula for computing the moments of the ISI. In other words, do an expansion of

$$D^n = \left[\sum_{j=1}^{5} h_j A_{j-k} \right]^n$$

and arrange it such that the n^{th} moment can be computed by successively adding the contributions coming from $h_1, h_2, h_3, \cdots$.

11.4 Use the approximations for a Gaussian pdf given in reference [3] with 20 points and compare the first eight moments of the actual Gaussian pdf with the moments of the approximation.

11.5 Simulate a 5 tap LMS equalizer for a binary system with the impulse response values given in Problem 11.1 using ordinary Monte Carlo simulations (gradient algorithm) with noise samples injected and compare the mean square error results with the weight vectors obtained from the channel covariance inversion method (Figure 11.10).

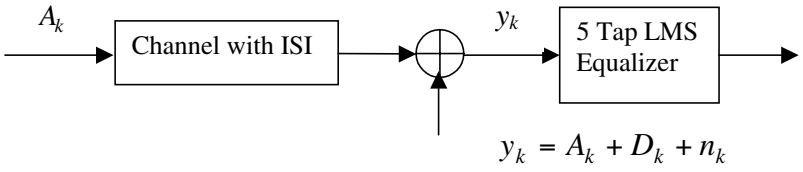


Figure 11.10 Simulation setup for Problem 11.5. $E_b N_0 = 6$ dB.

PART III

Advanced Models and Simulation Techniques

Chapter 12

MODELING AND SIMULATION OF NONLINEARITIES

Simulation models of functional blocks in communication systems fall into three broad categories as follows:

1. Linear time-invariant causal systems such as filters and static communication channels
2. Linear time-varying systems such as fading channels and linear equalizers
3. Nonlinear time-invariant systems such as high-power amplifiers, limiters, and phase-locked loops (PLLs)

While most of the functional blocks in a communication system are either linear or can be approximated by linear behavior, there are many functional blocks that are nonlinear. Some of these components are inherently nonlinear and are intentionally included in the system to improve performance. A limiter, for example, is a nonlinear element that may be included at the front end of a receiver to improve

performance in the presence of impulsive noise. Another example of a nonlinear device that produces better performance over its linear counterpart is the decision feedback equalizer (DFE). Other functional blocks, such as a high-power amplifier, however, may exhibit unintentional nonlinear behavior.

12.1 Introduction

There are many functional blocks in communications systems whose intended or desired behavior is linear but the physical devices that are used to implement these functions may produce nonlinear effects over certain ranges of operation. One example of such a device is a high-power amplifier, which might exhibit limiting and saturation when the input amplitude or power is very large.

Mathematical analysis of the effects of nonlinearities is, in general, very difficult. Even in the case of a simple third-order memoryless nonlinearity such as

$$y(t) = x(t) - 0.2x^3(t) \tag{12.1}$$

it is very difficult in general to compute the probability density function (pdf) and the autocorrelation function of the output $y(t)$ given the pdf and the autocorrelation function of the input $x(t)$. Using simulation, however, it is very easy to generate sampled values of the of the input $\{x(kT_s)\}$ and then use (12.1) to generate sampled values of the output $\{y(kT_s)\}$. From the two sequences $\{x(kT_s)\}$ and $\{y(kT_s)\}$ one may estimate a number of quantities of interest including the pdf and autocorrelation function of $x(t)$, the pdf and autocorrelation function of $y(t)$, and the crosscorrelation of $x(t)$ and $y(t)$. Indeed, simulation may be the only method available for the analysis and design of communication systems containing nonlinearities, as well as nonideal filters and non-Gaussian noise. This chapter focuses on methods of modeling and simulating nonlinear components in communication systems.

12.1.1 Types of Nonlinearities and Models

Nonlinearities in communication systems may be either baseband or bandpass. For example, a limiter is an example of a baseband nonlinearity whereas a radio frequency (RF) amplifier is a bandpass nonlinearity. The input to a bandpass nonlinearity will be centered at some frequency f_c and the spectral components of the output will lie in the neighborhood of f_c. Harmonic terms near $2f_c$, $3f_c \cdots$, will not be of interest in most cases, since the functional blocks "downstream" from the nonlinearity will usually reject the harmonic terms. (One example of where the harmonic terms might be of interest is in the analysis of "spurs," or spurious components, in the output of mixers.) The most commonly used model for a bandpass nonlinearity is a baseband nonlinearity followed by a "zonal filter" that passes only those components that lie within on near the edge of the band of frequencies occupied by the input to the nonlinearity.

Another classification of nonlinearity is based on whether the nonlinearity is, or is not, memoryless. The output of a memoryless nonlinearity at time t will depend

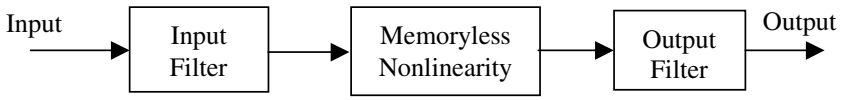


Figure 12.1 Model for a frequency selective nonlinearity with memory.

only on the instantaneous value of the input at time t, whereas a nonlinearity with memory will generate an output at time t that is a function of the present and past values of the input. Devices with memory will exhibit frequency-selective behavior, and they are usually modeled by a memoryless nonlinearity "sandwiched" between two filters as shown in Figure 12.1. The filters account for the frequency-selective properties of the nonlinearity with memory.

Bandpass nonlinearities can be modeled and simulated using the real-valued bandpass version of the signals. As we will see later on, the behavior of most bandpass nonlinearities can also be modeled and simulated in terms of the lowpass complex envelopes of the bandpass signals. The lowpass complex envelope representation leads to significant computational savings and hence is the preferred simulation technique.

Nonlinear devices may also be described by nonlinear differential equations. In this case, the simulation may be carried out in the form of a recursive solution of the nonlinear differential equations. An alternate approach is to represent the nonlinear system in block diagram form (if possible) and simulate the block diagram model. This approach is called an assembled block diagram method.

The assembled block diagram method was treated in some detail in Chapter 6 when we studied the phase-locked loop, but to briefly review consider a subsystem described by the differential equation

$$\frac{d^2y(t)}{dt^2} + t^2\frac{dy(t)}{dt} + \ln\left(|y(t)|\right) = x(t) \tag{12.2}$$

This differential equation can be rearranged as

$$\frac{d^2y(t)}{dt} = -t^2\frac{dy(t)}{dt} - \ln\left(|y(t)|\right) + x(t) \tag{12.3}$$

which can be represented in a block diagram form as shown in Figure 12.2. As we saw in Chapter 6, the simulation immediately follows by representing the continuous-time integrations by an appropriate algorithm for discrete-time integration.

The nonlinear differential equation given in (12.2) can also be simulated directly using recursive numerical integration techniques. This method is computationally very efficient, but requires some up-front effort to derive the model. For this reason, it is usually reserved for very complex nonlinear devices.

12.1.2 Simulation of Nonlinearities—Factors to Consider

There are a number of factors to be considered when one attempts to simulate the behavior of a nonlinearity. The simulation of nonlinearities is almost always performed in the time domain except for filters included in the model to account

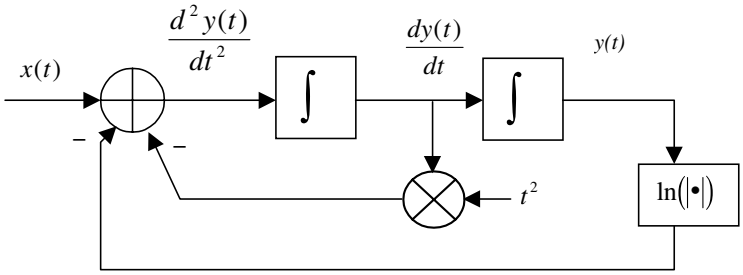


Figure 12.2 Assembled block diagram model for a nonlinearity.

for frequency-selective behavior. Filters, of course, can be simulated in the time domain or in the frequency domain.

To illustrate some of the factors that must be considered in the process of modeling or simulating a nonlinearity, assume that the model takes the form of either the zero-memory model described by (12.1) or the frequency-selective model shown in Figure 12.1.

Sampling Rate

The first factor we have to consider is the sampling rate. For a linear system we typically set the sampling rate at 8 to 16 times the bandwidth of the input signal. In the case of a nonlinearity of the form

$$y(t) = x(t) - 0.2x^3(t) \tag{12.4}$$

where the input $x(t)$ is a deterministic finite energy signal, the transform $Y(f)$ of the output $y(t)$ will be given by

$$Y(f) = X(f) - 0.2X(f) \circledast X(f) \circledast X(f) \tag{12.5}$$

where $\circledast$ denotes convolution. The triple convolution will lead to a threefold increase in the bandwidth of $Y(f)$ over the bandwidth of $X(f)$. This effect is called spectral spreading and is an effect of the nonlinearity. If $y(t)$ is to be represented adequately without excessive aliasing error, the sampling rate must be set on the basis of the bandwidth of $y(t)$, which has much higher bandwidth than $x(t)$. Thus, in setting the sampling rate for simulating a nonlinearity, we must take spectral spreading into account and set an appropriately high sampling rate. However, the sampling frequency actually required for simulation will not be as high as that indicated in this example (see Section 12.2.2 on the zonal bandpass model).

Cascading

Another factor that we have to consider is the effect of cascading linear and nonlinear blocks as in Figure 12.1. If we are using, for example, the overlap and add technique for simulating the filters, we have to exercise caution. We cannot process blocks

of data through the first filter, then through the nonlinearity and the second filter, and perform overlap and add at the output of the second filter. This operation is incorrect, since the principle of superposition does not apply for nonlinearities. A correct processing technique is to apply overlap and add at the output of the first filter, compute the time-domain samples representing the first filter output, process these samples on a sample-by-sample basis through the memoryless nonlinearity, and apply the overlap and add technique to the second filter. Note that this problem does not occur in the overlap and save method, since superposition is not applied.

Nonlinear Feedback Loops

Feedback loops might require the insertion of a one-sample delay in the loop in order to avoid computational deadlocks (recall, from Chapter 6, the feedback path in a PLL simulation). A small delay in a linear feedback loop may not adversely affect the simulation results. In a nonlinear system, however, a small delay in a feedback loop might not only significantly degrade the simulation results but might even lead to unstable behavior. In order to avoid these effects, the sampling rate must be increased, which, in effect, decreases the delay.

Variable Sampling Rate and Interpolation

If the model is a nonlinear differential equation that is solved using numerical integration techniques, many numerical integration algorithms included in software packages such as SIMULINK will use a variable integration step size. The step size will be determined automatically at each step depending on the behavior of the solution. If the solution is well behaved locally, then a large step size might be used. In order to avoid aliasing problems in downstream blocks, it might be necessary to interpolate the output and produce uniformly spaced samples of the output signal.

12.2 Modeling and Simulation of Memoryless Nonlinearities

The input-output relationship of a memoryless nonlinearity is given by

$$y(t) = F(x(t - t_0)) \tag{12.6}$$

where $F(\cdot)$ is a nonlinear function and t_0 may be assumed to be zero. If $x(t)$ is a baseband signal, then $y(t)$ will also be treated as a baseband signal. On the other hand, if $x(t)$ is a bandpass signal, then $y(t)$ will in general be a bandpass signal, but $y(t)$ may also contain a dc term as well as harmonics of the input signal. In practical cases, we might be interested in only those output terms whose spectral contents lie near the input frequencies. We can select these components of interest by adding a zonal bandpass filter at the output of the nonlinearity. Thus, the terminology lowpass versus bandpass applies to the frequency content of the input and the output. The nonlinearity itself is modeled as a simple instantaneous device.

In Chapter 4 the lowpass complex envelope representation for modeling and simulating bandpass signals through linear systems was developed. It is desirable to have models for bandpass nonlinearities also in terms of complex lowpass envelopes

of the input and output. It turns out that complex lowpass envelope models for many nonlinear devices including limiters (recall the example in Chapter 4), broadband RF amplifiers, phase-locked loops, decision feedback equalizers, and timing recovery systems can be derived.

12.2.1 Baseband Nonlinearities

The input to a baseband nonlinearity is a real-valued signal $x(t)$ and its output is also a real-valued signal, $y(t)$. The nonlinearity is modeled as $y(t) = F(x(t))$. The most commonly used models of baseband nonlinearities are the power series model and the limiter model. The power series model is defined by

$$y(t) = \sum_{k=0}^{N} a_k x^k(t) \tag{12.7}$$

and the general limiter model has the form [1]

$$y(t) = \frac{M\text{sgn}(x(t))}{[1 + (m/\,|x(t)|)^s]^{1/s}} \tag{12.8}$$

In (12.8), M is the limiting value of the output, m is the input limiting value, and s is the "shape" parameter. The normalized input-output relationship for a limiter is given in Figure 12.3 for different values of s. Note that $s = \infty$ corresponds to a "soft" limiter and $m = 0$ corresponds to a hard limiter. Also, note that with $m = 0$, the value of s has no effect on the characteristic of the nonlinearity described by (12.8). Figure 12.3 can be generated with the following MATLAB program:

```
% File: c12_limiter.m
x = -10:0.1:10;             % input voltage vector
M = 1;                      % output limiting value
m = 1;                      % input limiting value
s = [0.5 1.0 10.0];         % vector of shape factors
for k = 1:201
    xx=x(k);
        num = M*sign(xx);
        for kk = 1:3
        xxx=(m/abs(xx))^s(kk);
        den = (1+xxx)^(1/s(kk));
       y(k,kk) = num/den;
    end
end
plot(x,y(:,1),'k-',x,y(:,2),'k:',x,y(:,3),'k--');
grid
xlabel('Input Voltage')
ylabel('Output Voltage')
legend('s=0.5','s=1.0','s=10.0',2)
% End of script file.
```

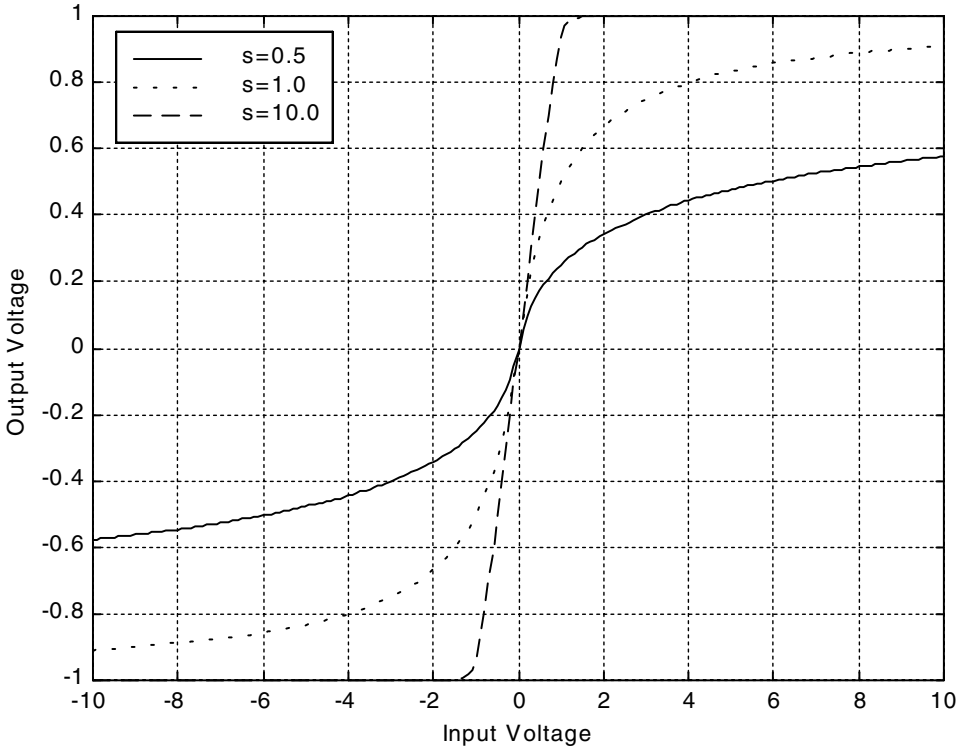


Figure 12.3 Limiter characteristics.

The student is encouraged to examine the behavior of the model for various values of m and s.

Simulation of the models given in (12.7) and (12.8) is trivial and it is always implemented in the time domain. We simply generate sample values of $x(t)$, process these samples through (12.7) or (12.8) to generate samples of $y(t)$. The samples of $y(t)$ are then used as input to the downstream blocks. The properties of $y(t)$, when the input $x(t)$ is a random process, can be estimated from the samples $\{y(kT_s)\}$. We do need to pay careful attention to the sampling rate requirements as discussed at the beginning of this section.

12.2.2 Bandpass Nonlinearities—Zonal Bandpass Model

Memoryless bandpass models are used to characterize a variety of narrowband nonlinear bandpass devices encountered in communications systems. The word *memoryless* implies not only an instantaneous relationship between input and output, but also implies that the device does not exhibit frequency-selective behavior over the bandwidth of operation. The bandwidth of the nonlinear device and the bandwidth of the signal are both assumed to be much less than f_c, where f_c is the carrier frequency.

When the bandwidth becomes wider, the nonlinearity may exhibit frequency-selective behavior, and we use appropriate frequency-selective models for such devices. Frequency selectivity is synonymous with memory, and the most commonly used model for frequency-selective nonlinearities (i.e., nonlinearities with memory) consists of a memoryless nonlinearity sandwiched between two filters as illustrated in Figure 12.1. We will focus on bandpass nonlinearities without memory in this section and deal with nonlinearities with memory in the next section.

Consider a memoryless nonlinearity of the form

$$y(t) = x(t) - 0.2x^3(t) \tag{12.9}$$

Assume that the input is a bandpass random signal of the form

$$x(t) = A(t)\cos[2\pi f_c t + \phi(t)] \tag{12.10}$$

where the amplitude $A(t)$ and the phase deviation $\phi(t)$ are lowpass random processes having bandwidth $B \ll f_c$ (the narrowband assumption). From (12.9) and (12.10) we obtain the output of the nonlinearity $Y(t)$ as

$$\begin{aligned} y(t) &= A(t)\cos[2\pi f_c t + \phi(t)] - 0.2\{A(t)\cos[2\pi f_c t + \phi(t)]\}^3 \\ &= A(t)\cos[2\pi f_c t + \phi(t)] - \frac{0.2}{4}A^3(t)\{\cos[6\pi f_c t + 3\phi(t)] + 3\cos[2\pi f_c t + \phi(t)]\} \\ &= [A(t) - \frac{0.6}{4}A^3(t)]\cos[2\pi f_c t + \phi(t)] - \frac{0.2}{4}A^3(t)\cos[6\pi f_c t + 3\phi(t)] \end{aligned} \tag{12.11}$$

In the preceding equation, the first term is at the center or carrier frequency f_c and the last term is at the third harmonic of the carrier frequency $3f_c$. The bandwidth of the third harmonic will be of the order of $3B$. Since our assumption is that $f_c \gg B$, the second term will be well outside the bandwidth of interest. We can therefore approximate the first zone output of the nonlinearity as

$$z(t) \approx [A(t) - \frac{0.6}{4}A^3(t)]\cos[2\pi f_c t + \phi(t)] \tag{12.12}$$

or

$$z(t) \approx f(A(t))\cos[2\pi f_c t + \phi(t)] \tag{12.13}$$

where

$$f(A(t)) = [A(t) - \frac{0.6}{4}A^3(t)] \tag{12.14}$$

Thus, the model for a narrowband, memoryless nonlinearity is a memoryless nonlinearity followed by a "zonal" bandpass filter that passes only the "first zone" output near f_c. The model is illustrated in Figure 12.4, in which $x(t)$ and $y(t)$ represent the bandpass input and output of the model, respectively, and the center frequency of the zonal bandpass filter is f_c. The memoryless nonlinearity itself does not respond differently to the baseband or bandpass input, and it is also not sensitive to the carrier frequency. It is the zonal bandpass filter that turns the baseband model into a bandpass model at f_c.

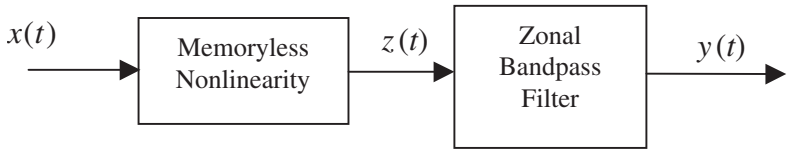


Figure 12.4 Zonal Bandpass Model for a Memoryless, Narrowband Nonlinearity.

Source: M. C. Jeruchim, P. Balaban, and K. S. Shanmugan, *Simulation of Communications Systems*, 2nd ed., New York: Kluwer Academic/Plenum Publishers, 2000.

Note that for the power series model, the bandpass output $y(t)$ has the same form as the input with the output amplitude related to the input amplitude via $f(A(t))$ and the output phase is the same as the input phase. The function $f(A(t))$ is referred to as the input amplitude to output amplitude, or the AM-to-AM transfer characteristic, of the nonlinearity. A limiter or power series model affects only the amplitude of the input signal. The phase is unaffected by the model.

In terms of the complex envelopes of the input $x(t)$ and the output $z(t)$, the lowpass equivalent model for the power series nonlinearity is

$$\widetilde{x}(t) = A(t) \exp\left[j\phi(t)\right] \tag{12.15}$$

and

$$\widetilde{z}(t) = f\left[A(t)\right] \exp\left[j\phi(t)\right] \tag{12.16}$$

The power series model can be simulated using the zonal bandpass model given in (12.13) and Figure 12.4 or the lowpass equivalent model given in (12.16). Simulating the bandpass model requires a much higher sampling rate and computational complexity than the lowpass equivalent model. Fortunately it is possible to derive lowpass equivalent models for most memoryless nonlinearities either analytically or from measurements as shown in the following sections.

12.2.3 Lowpass Complex Envelope (AM-to-AM and AM-to-PM) Models

Devices such as bandpass amplifiers, which respond to bandpass inputs having spectra centered on the carrier frequency, produce outputs that are bandpass in nature. The spectral components of these bandpass outputs are centered on the carrier frequency but perhaps have larger bandwidths than the input signal. Such devices can be modeled using the complex envelope representations of the input and output signals.

Suppose the input to a memoryless bandpass nonlinearity is of the form

$$x(t) = A(t)\cos[2\pi f_c t + \phi(t)] = A\cos(\alpha) \tag{12.17}$$

where

$$\alpha = 2\pi f_c t + \phi(t) \tag{12.18}$$

The output of the memoryless nonlinearity $y(t) = F(x(t))$ can be expressed as

$$y(t) = F(A\cos(\alpha)) \tag{12.19}$$

Since $A\cos(\alpha)$ is periodic in α, $y(t)$ will also be periodic in α. Therefore, $y(t)$ can be expanded in a Fourier series:

$$y(t) = a_0 + \sum_{k=1}^{\infty}(a_k\cos(k\alpha) + b_k\sin(k\alpha)) \tag{12.20}$$

where a_k and b_k are the Fourier coefficients given by

$$a_k = \frac{1}{\pi}\int_0^{2\pi} F(A\cos(\alpha))\cos(k\alpha)d\alpha \tag{12.21}$$

and

$$b_k = -\frac{1}{\pi}\int_0^{2\pi} F(A\cos(\alpha))\sin(k\alpha)d\alpha \tag{12.22}$$

The first zone output $z(t)$ in the vicinity of f_c is given by the $k = 1$ term in the Fourier series. This gives

$$z(t) = f_1(A(t))\cos[2\pi f_c t + \phi(t)] - f_2(A(t))\sin[2\pi f_c t + \phi(t)] \tag{12.23}$$

where

$$f_1(A) = a_1 = \frac{1}{\pi}\int_0^{2\pi} F(A\cos(\alpha))\cos(\alpha)d\alpha \tag{12.24}$$

and

$$f_2(A) = -b_1 = -\frac{1}{\pi}\int_0^{2\pi} F(A\cos(\alpha))\sin(\alpha)d\alpha \tag{12.25}$$

The function $f_1(A)$ is sometimes called the first-order Chebyshev transform and the complex function $[f_1(A) - jf_2(A)]$ is called the describing function of the nonlinearity.

The model given by (12.23) can also be expressed as

$$z(t) = f(A(t))\cos[2\pi f_c t + \phi(t) + g(A(t))] \tag{12.26}$$

where

$$f(A(t)) = \sqrt{f_1^2(A) + f_2^2(A)} \tag{12.27}$$

and

$$g(A(t)) = \arctan(f_2(t)/f_1(t)) \tag{12.28}$$

In polar coordinates we have

$$f(A)\exp(jg(A)) = f_1(A) + jf_2(A) \tag{12.29}$$

The functions $f(A)$ and $g(A)$ are the amplitude-to-amplitude (AM-to-AM) and amplitude-to-phase (AM-to-PM) transfer characteristics of the nonlinearity.

The model given in (12.26) is a generalization of the input-output relationship given in (12.12). Whereas the model given in (12.12) accounts for amplitude distortion only, the model given in (12.26) includes both the amplitude distortion and the phase distortion introduced by the nonlinearity, and it can be expressed in terms of the complex lowpass equivalents of the input and output as

$$\widetilde{x}(t) = A(t)\exp[j\phi(t)] \tag{12.30}$$

and

$$\widetilde{z}(t) = f(A(t))\exp(jg(A))\exp(j\phi(t)) \tag{12.31}$$

The model given in the preceding equation is in polar form, and it can be converted to rectangular form

$$z(t) = s_d(t)\cos[2\pi f_c t + \phi(t)] - s_q(t)\sin[2\pi f_c t + \phi(t)] \tag{12.32}$$

where the complex envelope can be expressed in terms of the direct or in-phase component and the quadrature components of $s_d(t)$ and $s_q(t)$ as

$$\widetilde{z}(t) = [s_d(t) + js_q(t)]\exp(j\phi(t)) \tag{12.33}$$

in which

$$s_d(t) = f(A(t))\cos(g(A(t)) \tag{12.34}$$

and

$$s_q(t) = f(A(t))\sin(g(A(t)) \tag{12.35}$$

Analytical Derivation of AM–to-AM and AM-to-PM Characteristics

The AM-to-AM and AM-to-PM transfer characteristics can be derived for a zonal bandpass nonlinearity using (12.23). For a hard-limiter type nonlinearity [see (12.8) with $m = 0$] it can be easily shown that

$$f(A) = \frac{4}{\pi}M \tag{12.36}$$

and

$$g(A) = 0 \tag{12.37}$$

The value of $f(A)$ is simply the amplitude of the fundamental sinusoidal component (the "first zone output") present in the square wave output of the limiter.

For the so-called soft-limiter type nonlinearity [see (12.8) with $s = \infty$], $f(A)$ and $g(A)$ can be shown to be [1]

$$\frac{f(A)}{A} = \begin{cases} \frac{M}{m}, & A < m \\ \frac{2}{M}\left[\arcsin\left(\frac{m}{A}\right) + \left(\frac{m}{A}\right)\left(1 - \frac{m^2}{A^2}\right)^{1/2}\right], & A \geq m \end{cases} \tag{12.38}$$

and

$$g(A) = 0 \tag{12.39}$$

which illustrates that the limiter does not introduce any phase distortion.

For any arbitrary memoryless zonal bandpass nonlinearity, we can derive $f(A)$ and $g(A)$ analytically if the transfer characteristics of the nonlinearity are given and if the integrals in (12.21) and (12.22) can be evaluated in closed form. In some cases we might be able to derive the lowpass equivalent model directly.

Consider for example a power series nonlinearity of the form

$$y(t) = \sum_{k=1}^{N} a_k x^k(t) \tag{12.40}$$

The bandpass input to the nonlinearity $x(t)$ can be expressed in terms of its lowpass complex envelope as

$$\begin{aligned} x(t) &= \mathrm{Re}\left\{\widetilde{x}(t)\exp(j2\pi f_0 t)\right\} \\ &= \frac{1}{2}\left[\widetilde{x}(t)\exp(j2\pi f_0 t) + \widetilde{x}^*(t)\exp(-j2\pi f_0 t)\right] \end{aligned} \tag{12.41}$$

In terms of the complex envelope, the n^{th} power of $x(t)$ can be expressed as

$$x^n(t) = \frac{1}{2^2}\sum_{k=0}^{n}\binom{n}{k}\left[\widetilde{x}(t)\right]^k\left[\widetilde{x}^*(t)\right]^{n-k}\exp\left(-j2\pi f_0(2k-n)t\right) \tag{12.42}$$

Only terms with n odd and $2k - n = \pm 1$ in $x^n(t)$ will produce a first-zone output. Hence the complex envelope of the first-zone output of the power series nonlinearity is

$$\widetilde{y}(t) = \widetilde{x}(t)\sum_{m=0}^{(N-1)/2}\frac{a_{2m+1}}{2^{2m}}\binom{2m+1}{m+1}\left|\widetilde{x}^*(t)\right|^{2m} \tag{12.43}$$

Equation (12.43) describes the complex lowpass equivalent model for a power series type of nonlinearity. [The third-order power series nonlinearity discussed in Section 12.2.2 is a special case of the model given in (12.43).] Note that in this model there is no phase distortion, that is, $g(A) = 0$.

Measurement of AM-to-AM and AM-to-PM Characteristics

The AM-to-AM and AM-to-PM characteristics of bandpass amplifiers and many other devices are normally determined experimentally from measurements and are not derived analytically. The input to the amplifier will be an unmodulated carrier of amplitude A and the output amplitude $f(A)$ and the output phase $g(A)$ are measured for different values of A. Such measurements are called "swept-power measurements" and the results of such measurements are usually included in the data sheets for the amplifiers.

The AM-to-AM and AM-to-PM characteristics will be displayed in decibel units of power. The output axis will be normalized with respect to the maximum output power of the device at saturation and the input axis will be normalized with respect to the input power that produces the maximum output. These normalized powers are referred to as output backoff (OBO) and input backoff (IBO), respectively. If the average input power is very small compared to the input power required to produce the maximum output power, the amplifier will behave linearly. As the input power is increased from this low level, the device will begin to exhibit nonlinear behavior. Sometimes the "operating point" of the amplifier will be specified as so many dBs below the input power required to produce the maximum output power. It is also a common practice to normalize the gain of the amplifier to one. It is trivial to add an ideal amplifier to account for the nonunity gain.

Measurements are typically made at a few input power levels, or equivalently, at various levels of $|A(t)|$, and the measured characteristics will be given as a table. During simulation it might be necessary to interpolate the values in the table for a given input power level. Also, it should be noted that the model given in (12.30) and (12.31) is in terms of voltage (or current) levels for $x(t)$, $A(t)$, etc., where the AM-to-AM measurements are usually given in terms of power levels. In such cases it will be necessary to convert the power gain or attenuation to voltage (or current) gain or attenuation. An example of typical AM-to-AM and AM-to-PM characteristics of a bandpass amplifier is shown in Figure 12.5. Also illustrated in Figure 12.5 is the operating point and the backoff. The backoff is measured with respect to the peak value of $f(|A(t)|)$ and is specified with respect to the input level, input backoff, or with respect to the output level, output backoff.

Analytical Forms of AM-to-AM and AM-to-PM Characteristics

It is often a common practice to approximate the measured AM-to-AM and AM-to-PM characteristics by analytical forms and to use the analytical forms rather than numerical interpolation for obtaining the output amplitude and phase values. Two functional forms widely used to model the measured characteristics of RF amplifiers [10] are given by

$$S_p(A) = \frac{\alpha_p A}{1 + \beta_p A^2}; \qquad S_q(A) = \frac{\alpha_q A^3}{(1 + \beta_q A^2)^2} \tag{12.44}$$

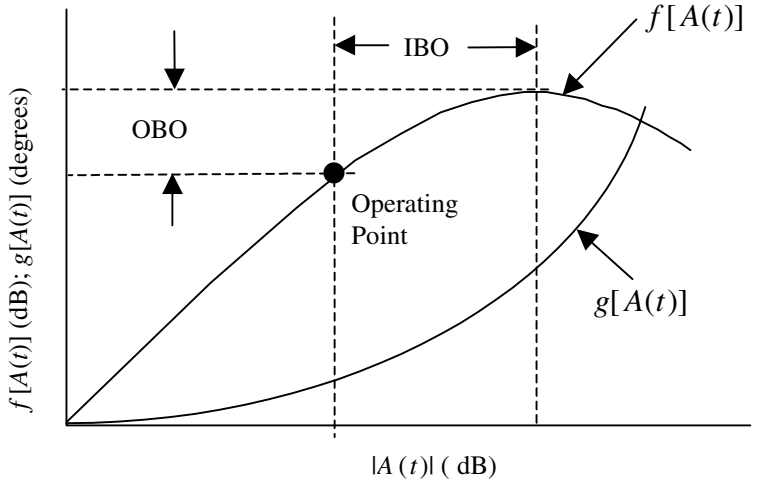


Figure 12.5 AM-to-AM and AM-to-PM characteristics.

or

$$f(A) = \frac{\alpha_f A}{1 + \beta_f A^2}; \qquad g(A) = \frac{\alpha_g A^2}{1 + \beta_g A^2} \tag{12.45}$$

The coefficients of the model, α_p, α_q, β_p, and β_q, or α_f, β_f, α_g, and β_g, are obtained from data using numerical curve-fitting techniques.

Example 12.1. The following MATLAB code illustrates the generation of the AM-to-AM and AM-to-PM characteristics using Saleh's model as defined through (12.45). The code for Saleh's model is given in Appendix A in which the model parameters are defined as $\alpha_f = 1.1587$, $\beta_f = 1.15$, $\alpha_g = 4.0$, and $\beta_g = 2.1$.

```
% File:  c12_example1.m
x = 0.1:0.1:2;                      % input power vector
n = length(x);                      % length of x
backoff = 0.0;                      % backoff
y = salehs_model(x,backoff,n);      % nonlinearity model
subplot(2,1,1)
pin = 10*log10(abs(x));             % input power in dB
pout = 10*log10(abs(y));            % output power in dB
plot(pin,pout); grid;
xlabel('Input power - dB')
ylabel('Output power - dB')
subplot(2,1,2)
plot(pin,(180/pi)*unwrap(angle(y))); grid;
xlabel('Input power - dB')
```

```
ylabel('Phase shift - degrees')
% End of script file.
```

Executing the program results in the output illustrated in Figure 12.6. ■

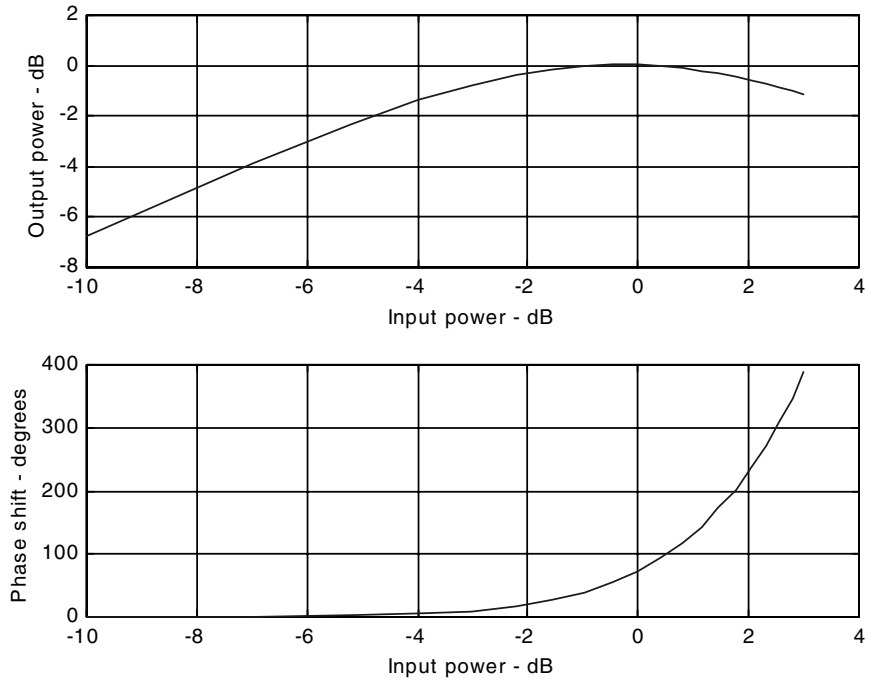
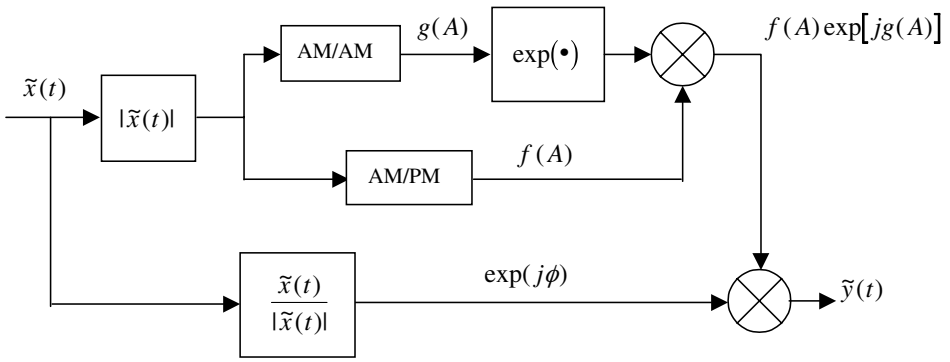


Figure 12.6 AM-to-PM characteristics.

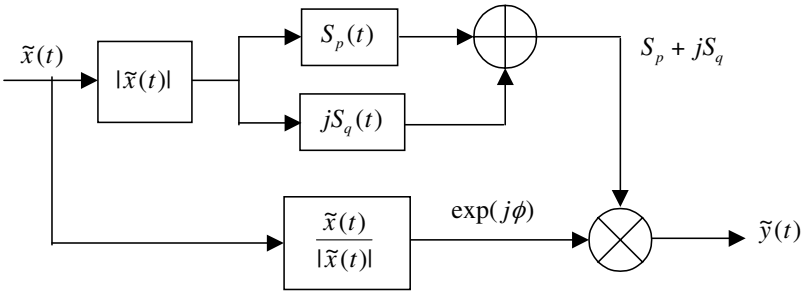
12.2.4 Simulation of Complex Envelope Models

Complex envelope models can be simulated in the time domain in either polar form, as defined in (12.30) and (12.31), or in quadrature form as defined in (12.33), (12.34), and (12.35). The corresponding block diagrams are shown in Figure 12.7. In polar form, the simulation procedure consists of the following steps:

1. Generate sampled values of the complex envelope of the input $\{\widetilde{x}(kT_s)\}$.
2. Compute the input amplitude $A(kT_s) = |\widetilde{x}(kT_s)|$, and phase $\phi(kT_s) = \arg\left(\frac{\widetilde{x}(kT_s)}{|\widetilde{x}(kT_s)|}.\right)$.
3. Find the output amplitude using the AM-to-AM characteristic $f(A)$. (Note: This step might require interpolation of AM-to-AM values and conversion of power gain to voltage or current gain.)



(a) Polar model



(b) Quadrature model

Figure 12.7 Complex lowpass envelope models of a zonal bandpass nonlinearity.

Source: M. C. Jeruchim, P. Balaban, and K. S. Shanmugan, *Simulation of Communications Systems*, 2nd ed., New York: Kluwer Academic/Plenum Publishers, 2000.

4. Find the output phase offset using the AM-to-PM characteristic $g(A)$. (Note: This step might require interpolation of AM-to-PM values.)

5. Compute sampled values of the complex output $\{\tilde{z}(kT_s)\}$ according to (12.31).

A similar procedure is used for implementing the quadrature model shown in Figure 12.7. Computationally, the two procedures are identical.

12.2.5 The Multicarrier Case

In our development of the AM-to-AM and AM-to-PM models we have thus far implicitly assumed that the input to the nonlinear device consists of a single carrier with amplitude modulation $A(t)$ and frequency or phase modulation $\phi(t)$. While

this might be the case in wideband single-carrier TDMA systems, the input signal in a FDMA system might consist of many individually modulated carriers and the sum of many such carriers might be amplified by a single-power amplifier. It is rather straightforward to extend the AM-to-AM and AM-to-PM models to the multicarrier case.

The Multicarrier Model

Suppose that the input to a nonlinearity consists of the sum of m modulated carriers

$$x(t) = \sum_{k=1}^{m} A_k(t) \cos\left[2\pi\left(f_c + f_k\right)t + \phi_k(t)\right], \quad -\frac{B}{2} \leq f_k \leq \frac{B}{2} \tag{12.46}$$

where f_k is the frequency offset of the k^{th} carrier, f_c is the nominal center frequency, and $A_k(t)$ and $\phi_k(t)$ represent the amplitude and phase modulation associated with the k^{th} carrier. We can express the composite multicarrier signal in lowpass complex envelope form as (recall Section 4.3)

$$x(t) = \text{Re}\left\{\widetilde{x}(t)\exp(j2\pi f_c t)\right\} \tag{12.47}$$

where the lowpass complex envelope $\widetilde{x}(t)$ is given by

$$x(t) = \sum_{k=1}^{m} A_k(t)\exp\left[j2\pi f_k t + j\phi_k(t)\right] = A(t)\exp\left[j\phi(t)\right] \tag{12.48}$$

In (12.48) the amplitude of the complex envelope is

$$A(t) = \sqrt{x_i^2(t) + x_q^2(t)} \tag{12.49}$$

and the phase is

$$\phi(t) = \arctan\left[x_q(t)/x_i(t)\right] \tag{12.50}$$

where

$$x_i(t) = \sum_{k=1}^{m} A_k(t)\cos\left[j2\pi f_k t + j\phi_k(t)\right] \tag{12.51}$$

is the direct component and

$$x_q(t) = \sum_{k=1}^{m} A_k(t)\sin\left[j2\pi f_k t + j\phi_k(t)\right] \tag{12.52}$$

is the quadrature component.

The lowpass complex envelope of the output, $\widetilde{z}(t)$, and the actual bandpass output signal $z(t)$ can be obtained from

$$\widetilde{z}(t) = f(A(t))\exp\left[jg(A(t))\right]\exp\left[j\phi(t)\right] \tag{12.53}$$

and

$$z(t) = \text{Re}\left\{\widetilde{z}(t)\exp\left[j2\pi f_c t\right]\right\} \tag{12.54}$$

respectively.

Intermodulation Distortion in Multicarrier Systems

Consider a power series nonlinearity of the form

$$y(t) = x(t) - a_3 x^3(t) \tag{12.55}$$

followed by a zonal filter. Suppose that the input to the nonlinearity is the sum of two modulated tones

$$x(t) = A_1 \cos\left[2\pi\left(f_c + f_1\right)t\right] + A_2 \cos\left[2\pi\left(f_c + f_2\right)t\right] \tag{12.56}$$

where f_c is the carrier frequency, f_1 and f_2 are the offset frequencies, and $f_i < B \ll f_0, i = 1, 2$. It can be shown that the first zone output (terms in the vicinity of f_c) is given by

$$\begin{aligned} z(t) = &\left\{a_1 A_1(t) - \left[\frac{3}{4}a_3 A_1^3(t) + \frac{3}{2}a_3 A_2^2(t) A_1(t)\right]\right\} \cos\left[2\pi\left(f_c + f_1\right)t\right] \\ &+ \left\{a_1 A_2(t) - \left[\frac{3}{4}a_3 A_2^3(t) + \frac{3}{2}a_3 A_1^2(t) A_2(t)\right]\right\} \cos\left[2\pi\left(f_c + f_2\right)t\right] \\ &- \frac{3}{4}A_1^2(t) A_2(t) \cos\left[2\pi\left(f_c + 2f_1 - f_2\right)t\right] \\ &- \frac{3}{4}A_2^2(t) A_1(t) \cos\left[2\pi\left(f_c + 2f_2 - f_1\right)t\right] \end{aligned} \tag{12.57}$$

The preceding expression for the output consists of distorted terms at the input frequencies as well as cross-modulated terms at $f_c + 2f_1 - f_2$ and $f_c + 2f_2 - f_1$, which are usually referred to as "intermodulation" distortion terms. These intermodulation distortion terms degrade the overall signal quality and might also produce adjacent channel interference if the frequency of these terms causes them to fall outside the bandwidth of the signal of interest but inside the bandwidth that might be occupied by an adjacent signal. (The reader can verify that even-powered terms in a power series type nonlinearity do not produce any distortion terms in the vicinity of the input frequency.)

It can be easily seen that, when the input consists of a large number of carriers or when the nonlinearity has many higher-order nonlinear terms, the output will contain a very large number of terms at various linear combinations of the input frequencies. While it is possible to develop an algorithm for the number of intermodulation terms, and the frequencies of these terms, it is very difficult to characterize and analyze the effects of the distortion introduced by the nonlinearity analytically when the input consists of a large number of arbitrarily modulated carriers and/or when the nonlinearity is severe. The communication literature is full of techniques for approximating the effects of intermodulation distortion. However, an exact characterization of intermodulation in a multicarrier digital system with arbitrary modulation schemes is difficult. Simulation is very useful in such applications, as will be seen in the following section.

Example 12.2. Consider a memoryless third-order nonlinearity of the form

$$y(t) = x(t) - 0.3x^3(t) \tag{12.58}$$

with a two-tone bandpass input $x(t)$ at frequencies 11 and 14 Hz

$$x(t) = \cos\left[2\pi\left(11\right)t\right] + 0.707\cos\left[2\pi\left(14\right)t\right] \tag{12.59}$$

The bandpass versions of the input and output are shown in Figure 12.8. The intermodulation terms generated by the nonlinearity lie at 8Hz and 17 Hz and the third harmonics are around 33 and 42Hz.

The lowpass equivalent version of this example, with a reference frequency $f_0 = 12$, has the form

$$\widetilde{x}(t) = \exp\left[-j2\pi\left(1\right)t\right] + 0.707\exp\left[j2\pi\left(2\right)t\right] \tag{12.60}$$

and

$$\widetilde{y}(t) = \widetilde{x}(t) - 0.75(0.3)\left|\widetilde{x}(t)\right|^2\widetilde{x}(t) \tag{12.61}$$

The intermodulation distortion terms now appear at -4 and $+5$ Hz, respectively (with respect to the reference frequency $f_0 = 12$ Hz), as illustrated in Figure 12.9. No third harmonic terms are accounted for in the lowpass equivalent model. The MATLAB code used to generate the results illustrated in this example is contained in Appendix B. ■

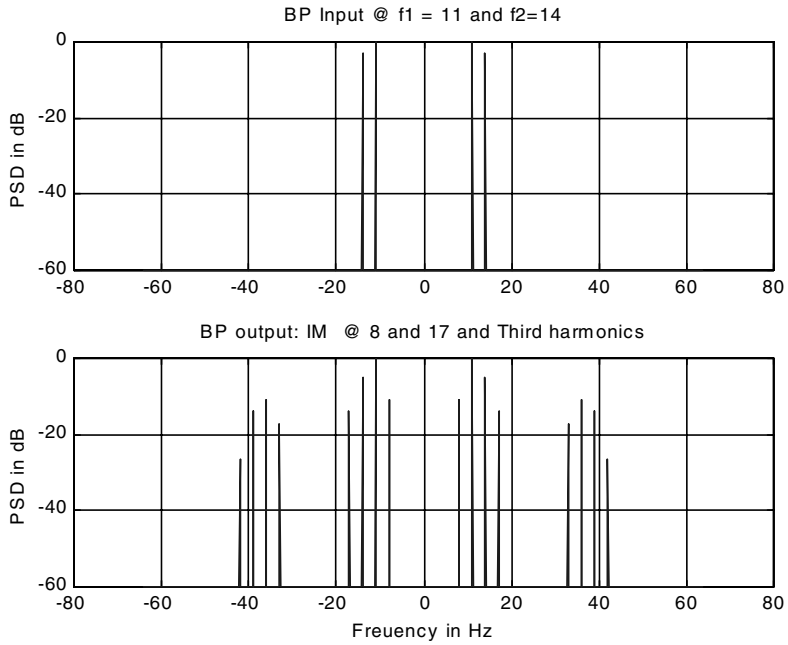


Figure 12.8 Nonlinearity input and output based on bandpass model.

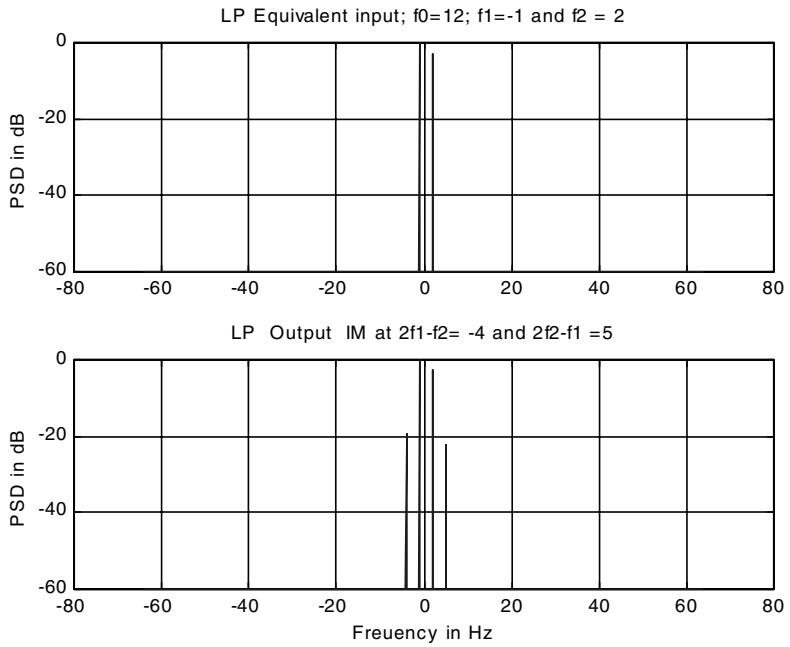


Figure 12.9 Nonlinearity input and output based on lowpass model.

Example 12.3. We now consider the nonlinearity to be based on the AM-to-AM and AM-to-PM characteristic rather than on the power series model as we did in the preceding example. The AM-to-AM and AM-to-PM model used in this example is based on Saleh's model as defined through (12.45). The AM-to-AM and AM-to-PM characteristics were determined in Example 12.1 and are illustrated in Figure 12.6. The parameter values are given in Example 12.1, the MATLAB code for simulating Saleh's model is given in Appendix A, and `log_psd.m` is given in Appendix A of Chapter 7. The lowpass equivalent signal model, which is the sum of two complex tones of the preceding example, is used for the simulation so that the results can be compared. The MATLAB code follows:

```
% File:  c12_example3.m
backoff = input('Enter backoff in dB > ');
f1 = -1.0; f2 = 2.0; ts = 1.0/128; n = 1024;
for k=1:n
    t(k) = (k-1)*ts;
    x(k) = exp(i*2*pi*f1*t(k))+0.707*exp(i*2*pi*f2*t(k));
    y(k) = salehs_model(x(k),-1*backoff,1);
end
```

```
[psdx,freq] = log_psd(x,n,ts);
[psdy,freq] = log_psd(y,n,ts);
subplot(2,1,1)
plot(freq,psdx); grid; title('Input to the NL');
ylabel('PSD in dB');
subplot(2,1,2)
plot(freq,psdy); grid; title('Output of the NL');
ylabel('PSD in dB'); xlabel('Frequency in Hz');
% End of script file.
```

Executing the code yields the result illustrated in Figure 12.10 for 5 dB backoff. (Note: In the MATLAB program we enter backoff as a positive quantity, since this seems more natural. Note that this is converted to a negative quantity in the call to the model, since the signal level is, in this case, −5 dB relative to the peak value.) ■

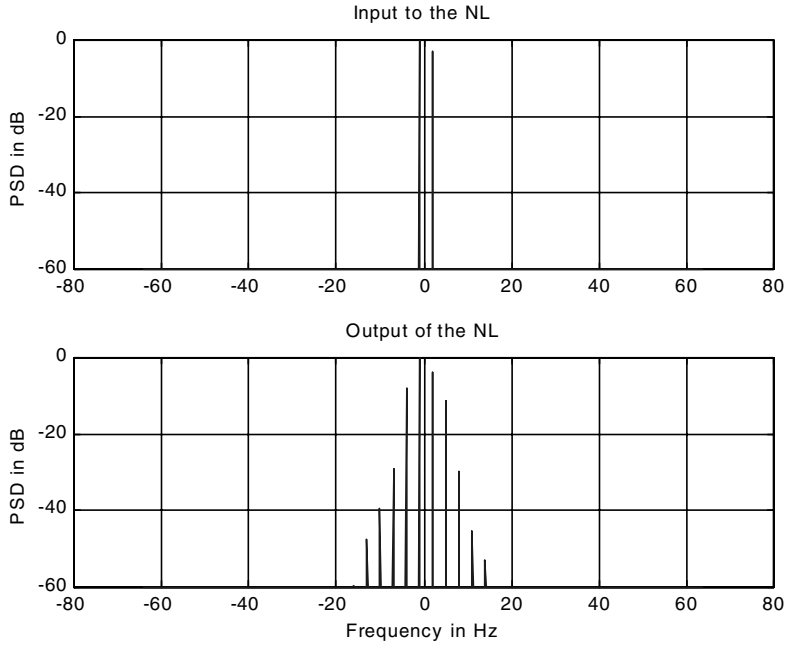


Figure 12.10 Simulation results for 5 dB backoff.

Example 12.4. We now consider a complex lowpass signal model for a 16-QAM signal. As in the previous example Saleh's model for the AM-to-AM and AM-to-PM characteristics is used. The input and output signal constellation is computed for a backoff of 10 dB. The purpose of the simulation is to determine the effect of the nonlinearity on the signal constellation. The MATLAB code for implementing the

simulation follows:

```
% File:  c12_example4.m
%
% Create input constellation
backoff = input('Enter backoff in dB > ');
N = 1024;                        % number of points
x1 = 2*fix(4*rand(1,N))-3;       % direct components
x2 = 2*fix(4*rand(1,N))-3;       % quadrature components
y = x1+i*x2;                     % signal space points
%
% Run it thru Saleh's model
z = salehs_model(y,-1*backoff,1024);
subplot(1,2,1)
plot(real(y),imag(y));grid; title('Input Constellation');
xlabel('direct'); ylabel('quadrature')
axis equal
subplot(1,2,2)
plot(real(z),imag(z));grid; title('Output Constellation');
xlabel('direct'); ylabel('quadrature')
axis equal
% End of script file.
```

Executing the MATLAB program yields the results illustrated in Figure 12.11 for 10 dB backoff. Note that the effect of the nonlinearity is to move a number of the signal points closer together. As we noted in Chapter 4, for an additive, white, Gaussian noise (AWGN) channel the pairwise error probability is a monotonic function of the Euclidean distance between a pair of points in signal space with the error probability increasing as the points in signal space move closer together. We therefore conclude that the nonlinearity degrades the probability of error of the communications system. Like the eye diagram, the signal constellation gives us a qualitative measure of system performance. Thus, viewing the change in the signal constellation as system parameters are varied gives us considerable insight into system performance. This is especially true for nonlinear systems. The student should therefore simulate this system a number of times and observe the result of varying the backoff. ■

12.3 Modeling and Simulation of Nonlinearities with Memory

If the output of a nonlinear device depends on the present and past values of the input signal, the device is classified as a nonlinearity with memory. Memory, or dependence on past input values, is modeled in linear systems by the impulse response and the convolution integral in the time domain. Linear systems are modeled in the frequency domain by the transfer function, which implies that the sinusoidal

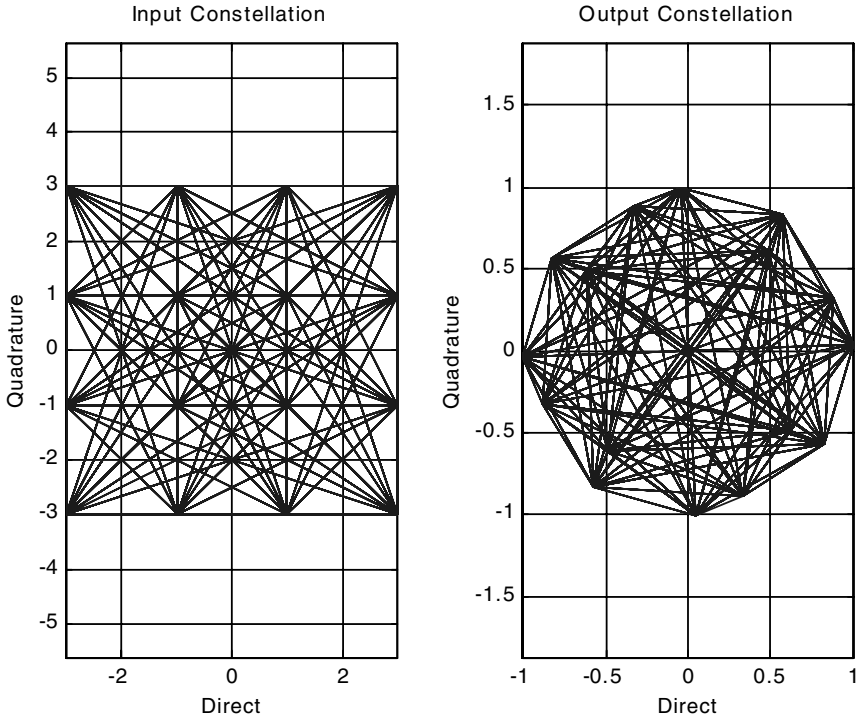


Figure 12.11 Input and output signal constellations for 10 dB backoff.

steady-state system response is dependent on the input frequency. Thus, memory and frequency selective behavior are synonymous.

Many nonlinear devices, such as wideband amplifiers, exhibit frequency-selective behavior. Such behavior will be evident when the response of the device is measured at different power levels and at different frequencies. If the input-output relationship does not depend on the frequency of the tone used in the measurements, the device is memoryless. Otherwise, the device exhibits frequency-selective behavior and therefore has memory.

Suppose that the input to an AM-to-AM nonlinearity is an unmodulated tone at some frequency $f_c + f_i$ where f_c is the center frequency of the device and f_i is the offset frequency with respect to the center frequency. The complex envelope of the input signal is

$$\widetilde{x}(t) = A \exp\left[j2\pi f_i t\right] \tag{12.62}$$

from which

$$|\widetilde{x}(t)| = A \tag{12.63}$$

If the nonlinearity has no memory, and is therefore not frequency selective, the output is given by

$$\widetilde{y}(t) = f(A)\exp\left[j2\pi f_i t\right] \tag{12.64}$$

Note that the output amplitude is independent of frequency.

If measurements indicate that the nonlinearity is frequency selective, we can attempt to account for the frequency selectivity by modifying the AM-to-AM function to include the input frequency and write the response as

$$\widetilde{y}(t) = f(AH(f_i))\exp\left[j2\pi f_i t\right] \tag{12.65}$$

As we vary f_i over the bandwidth of the device, the function $H(f_i)$ accounts for the frequency dependency of the nonlinearity. Note that $H(f_i)$ may be viewed as the transfer function of a filter that precedes the AM-to-AM nonlinearity. The filter produces a response $AH(f_i)\exp\left[j2\pi f_i t\right]$ when the input is $\widetilde{x}(t) = A\exp\left[j2\pi f_i t\right]$. Thus, the model for the AM-to-AM part of a frequency-selective nonlinearity consists of a filter $H(f_i)$ followed by a memoryless AM-to-AM nonlinearity $f(A)$.

This approach can be extended to account for frequency-selective AM-to-PM transfer characteristics by including another filter in the model. Indeed, the most commonly used model for frequency-selective nonlinearities consists of a memoryless nonlinearity sandwiched between two filters. We now describe two of these models.

12.3.1 Empirical Models Based on Swept Tone Measurements

These models are derived from "swept tone" and "stepped power" measurements made with an input that is an unmodulated tone of the form

$$x(t) = A_i\cos\left[2\pi\left(f_i + f_c\right)t\right] \tag{12.66}$$

The output amplitude and phase offset are measured for different values of the input amplitude A_i and frequency f_i producing a set of plots as shown in Figure 12.12(a). Note that these plots clearly show the frequency dependent nature of the response of the nonlinear device.

Two models that attempt to take into account the frequency-dependent nature of nonlinear devices are Poza's model and Saleh's model. Both of these models try to reproduce as accurately as possible the swept tone and stepped power measurements.

Poza's Model

A simple simulation model that characterizes measurements like the ones shown in Figure 12.12(a) has been derived by Poza et al. [11] based on the following assumptions:

1. AM-to-AM: Curves of AM-to-AM for different frequencies are similar in shape, with all curves being a combination of vertical and horizontal translations of one another.

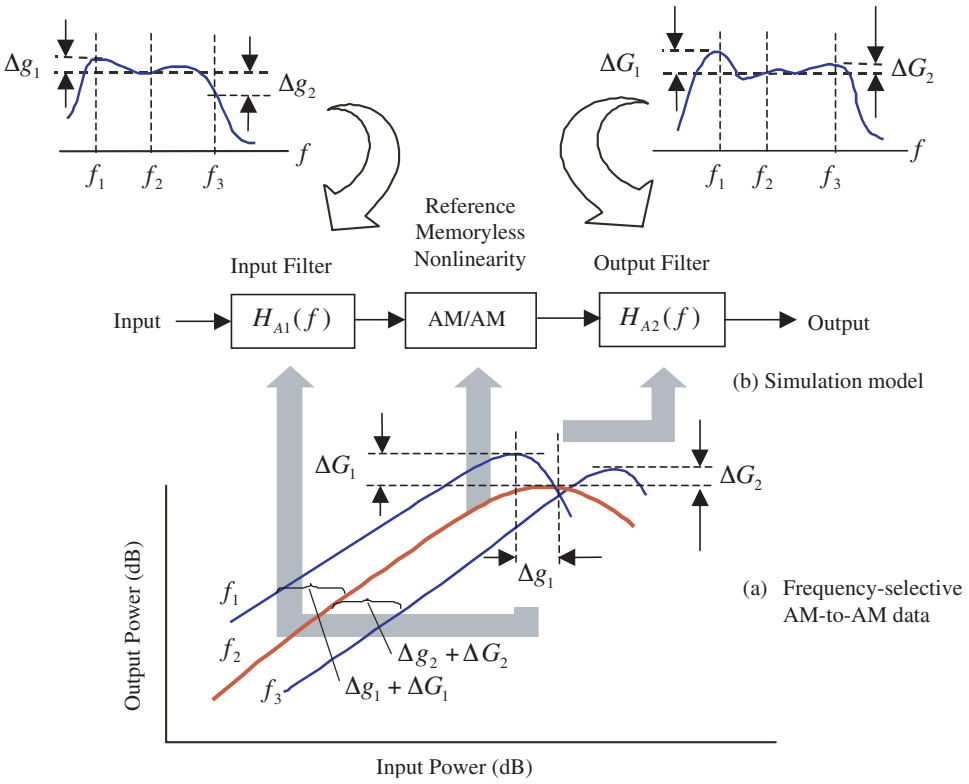


Figure 12.12 Example of frequency-selective AM-to-AM model.

Source: M. C. Jeruchim, P. Balaban, and K. S. Shanmugan, *Simulation of Communications Systems*, 2nd ed., New York: Kluwer Academic/Plenum Publishers, 2000.

2. AM-to-PM: These curves for different frequencies are also similar in shape and all curves are a combination of horizontal and vertical translations of each other.

This model requires a complex curve-fitting procedure to fit a family of curves for the AM-to-AM and AM-to-PM data such that each member of the family is a translated version of the other member (translation of both horizontal and vertical axes) of the same family.

Let us first consider how to implement the frequency-selective AM-to-AM simulation model. The first step in the implementation is to select one of the curves within the AM-to-AM family as the "reference" nonlinearity at the reference frequency f_c. With respect to this curve, the AM-to-AM response at another frequency f_1 can be obtained through a horizontal translation $\Delta G_1 + \Delta g_1$. The horizontal translation corresponds to an attenuation or gain of the input signal with respect to the reference nonlinearity at frequency f_1, and the vertical translation ΔG_1

corresponds to an attenuation or gain of the output of the reference nonlinearity at frequency f_1. These can be implemented by two FIR filters, one preceding and one following the reference nonlinearity, with amplitude responses $H_{AM1}(f)$ and $H_{AM2}(f)$, respectively. Thus, the AM-to-AM model can be implemented as shown in Figure 12.12.

A similar model and implementation can be derived for the AM-to-PM responses. This model will consist of an input filter with an amplitude response $H_{PM}(f)$ preceding the AM-to-PM reference nonlinearity and a phase response $\exp[j\theta_{PM}(f)]$ following the output. The AM-to-AM and AM-to-PM models can be combined into one model as shown in Figure 12.13. The AM-to-PM part precedes the AM-to-AM part, and the input (amplitude) offset introduced by the AM-to-PM model has to be "taken out" prior to the AM-to-AM model so that the power level at the input to the AM-to-AM part of the model is the same as the power in the input signal $\widetilde{x}(t)$.

Saleh's Model

A slightly different approach to modeling nonlinearities with memory has been proposed by Saleh [10]. This model is obtained from the memoryless quadrature model given in (12.44) and (12.45) by modifying the coefficients to be frequency dependent. This gives

$$S_d(A) = \frac{\alpha_p(f)A}{1 + \beta_p(f)A^2} \tag{12.67}$$

and

$$S_q(A) = \frac{\alpha_q(f)A^3}{[1 + \beta_q(f)A^2]^2} \tag{12.68}$$

The coefficients are obtained from the measurements made at various frequencies f using a least squares fit. The actual implementation of the model is given in Figure 12.14. The functions illustrated in Figure 12.14 are defined as follows: $\phi_0(f)$ is the small signal phase response, $H_p(f) = \sqrt{\beta_p(f)}$, $H_q(f) = \sqrt{\beta_q(f)}$, $P_0 = A/(1 + A^2)$, $G_p(f) = \alpha_p(f)/\sqrt{\beta_p(f)}$, $G_q(f) = \alpha_q(f)/\sqrt{\beta_q^{3/2}(f)}$, and $Q_0(A) = A^3/[1 + A^2]^2 T$. The details of this derivation are left as an exercise for the reader.

Note that Saleh's model and Poza's model start out with different assumptions and that the two models are topologically different. They are both "block diagram models" designed to reproduce a specific set of measurements. Both of these models have more complex structures than models for memoryless nonlinearities. There is also an added degree of complexity due to the empirical procedure used for fitting model parameters to measured data.

12.3.2 Other Models

In spite of the complexity, the models described in the preceding section are not really capable of capturing the behavior of a nonlinearity when the input consists

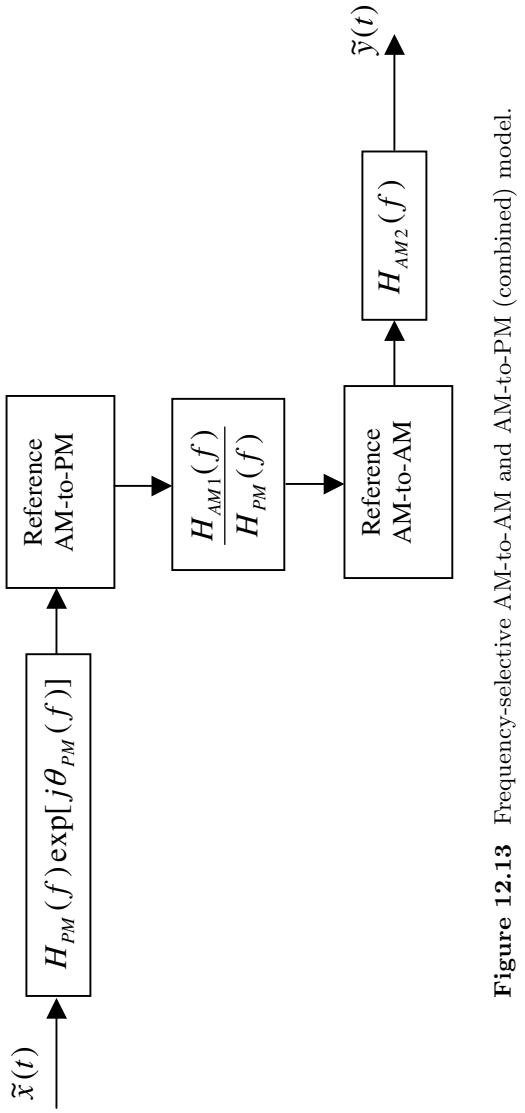


Figure 12.13 Frequency-selective AM-to-AM and AM-to-PM (combined) model.

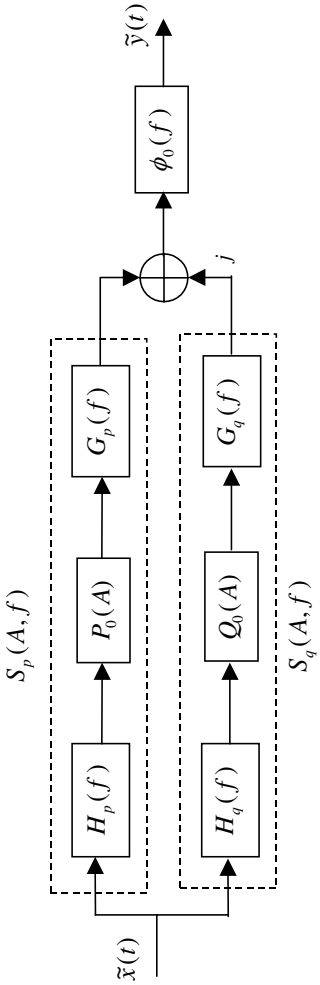


Figure 12.14 Saleh's quadrature model for a nonlinearity with memory.

Source: M. C. Jeruchim, P. Balaban, and K. S. Shanmugan, *Simulation of Communications Systems*, 2nd ed., New York: Kluwer Academic/Plenum Publishers, 2000.

of a sum of many modulated carriers, since the models were derived on the basis of a single, constant envelope, carrier (tone) measurements. Since superposition does not hold for nonlinear systems, it is not possible to characterize the behavior with multitone input based on single-tone measurements. It is, of course, very difficult to make multitone measurements for many different combinations of input frequencies and power levels. The number of combinations of frequencies and power levels will be overwhelmingly large when the number of carriers is large.

Some alternate approaches have been suggested, one of which includes approximating the behavior of the nonlinearity using a carrier modulated with a PN sequence as the input [12]. The input now approximates the sum of a large number of carriers with a more or less uniform power spectral density at the input. (Note that, with this approach, the input spectral components are generated by the modulating signal.) By changing the total power in the input, measurements are taken to characterize the behavior of the nonlinearity under input conditions that produce a reasonable approximation of a multicarrier input. The resulting model has a transfer function of the form $H(f, P)$, where P is the input power level. The details of this approach may be found in [12].

Another more general approach that has been suggested for modeling and analyzing nonlinearities with memory uses the Volterra series. The input-output relationship in the time domain is given by

$$y(t) = \sum_{k=1}^{\infty} y_k(t) \tag{12.69}$$

where

$$y_k(t) = \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} h(\tau_1, \cdots, \tau_k) x(t-\tau_1) \cdots x(t-\tau_k) d\tau_1 \cdots d\tau_k \tag{12.70}$$

The time response model given in (12.70) assumes that the system has no dc response or constant in the output. Note that the model looks like a power series model. However, each term in the Volterra series model is a k-fold convolution (and not a k^{th} power) of the input signal with a k-fold impulse response kernel $h(\tau_1, \cdots, \tau_k)$. A lowpass equivalent version of the model can be derived but it is exceedingly complex. The interested reader is referred to [13].

While the Volterra series model is analytically very elegant, the higher-order kernels are difficult to characterize and measure. Hence the model is of limited use for simulation except in cases where the order of the model is small. As in the case of power series models, only odd terms contribute to the first zone output.

12.4 Techniques for Solving Nonlinear Differential Equations

We have seen that a nonlinear system with memory may be modeled by a nonlinear differential equation (NLDE) and the simulation of the system can take place by substituting a discrete-time integration for the continuous-time integration. We have also seen there are two approaches to simulating a system described by a

differential equation. There is the *assembled block diagram approach*, in which the system block diagram is constructed using basic building blocks including some simpler memoryless nonlinear blocks and integrators, and the solution is obtained by simulating the block diagram model of the nonlinear system. An alternate approach is to derive the nonlinear differential equation that governs the dynamic behavior of the PLL and solve the NLDE using recursive procedures. This method, referred to as the direct or s*tand-alone method*, will be superior in terms of stability and accuracy for a given sample time. However, some effort will be required to develop the model and implement this approach, and the effort has to be repeated for each new nonlinear system that needs to be simulated. With the block diagram assembly method, a core set of building blocks can be used to construct models of new systems rather quickly.

All approaches yield the same results when the sample time, or simulation step time, is small. However, if we desire a large time step in order to reduce the computational burden, the stand-alone model using a variable step-size solution technique will be very efficient. The step size is usually chosen automatically by the solution method depending on the behavior of the underlying NLDE. In regions where the solution behaves well, it is possible to use very large time steps resulting in significant savings in simulation time. Some interpolation, however, will be required if succeeding blocks require uniformly spaced samples.

When a PLL is simulated by itself, the time step required will be determined by the so-called "loop bandwidth." If the PLL is simulated along with other parts a receiver, the sampling rate for the remainder of the receiver will be governed by the data rate, R. In general, R will be much larger than the loop bandwidth, and hence the time step commensurate with R will be much smaller than that needed for accurate simulation of the PLL itself. In this situation, it might be advantageous to use multirate simulations in which different time steps are used for different parts of the system with appropriate decimation and interpolation in between.

The focus of the remainder of this chapter is on the simulation of nonlinear systems with memory using a recursive solution of the underlying NLDE. We will use the PLL, which we studied extensively in Chapter 6, as an example to illustrate the methodology.

12.4.1 State Vector Form of the NLDE

The literature on numerical methods is full of techniques for solving nonlinear differential equations [14]. From a simulation point of view, techniques that can be implemented in recursive form are most attractive both from a structural as well a computational point of view. For linear differential equations of the m^{th} order, the recursive solution in time domain often makes use of the state variable method where the system equations are represented in the form

$$\dot{\mathbf{X}}(t) = \mathbf{A}\mathbf{X}(t) + \mathbf{B}\mathbf{U}(t) \tag{12.71}$$

and

$$\mathbf{Y}(t) = \mathbf{D}\mathbf{X}(t) \tag{12.72}$$

where $\mathbf{X}$ is the state vector of size $m \times 1$, $\mathbf{U}$ is the input vector, $\mathbf{Y}$ is the output vector, $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{D}$ are matrices of constants which define the system, and m is the order of the system.

To apply this technique to the PLL, consider the relation between $e_d(t)$, as identified in Figure 12.15 and the VCO phase $\theta(t)$. This is the linear part of the system and it follows from Figure 12.15 that the relationship between $\theta(t)$ and $e_d(t)$ is defined by

$$\theta(t) = \int c_3 \left[\dot{x}(t) + c_2 \ddot{x}(t)\right] dt = c_3 x(t) + c_2 c_3 \dot{x}(t) \tag{12.73}$$

and

$$\ddot{x}(t) = e_d(t) + c_1 \dot{x}(t) \tag{12.74}$$

From the definition of $e_d(t)$ and (12.73)

$$e_d(t) = \sin\left[\phi(t) - \theta(t)\right] = \sin\left[\phi(t) - c_2 c_3 \dot{x}(t) - c_3 x(t)\right] \tag{12.75}$$

Substitution into (12.74) gives the differential equation

$$\ddot{x}(t) = \sin\left[\phi(t) - c_2 c_3 \dot{x}(t) - c_3 x(t)\right] + c_1 \dot{x}(t) \tag{12.76}$$

From Figure 6.3 and (6.25) we know that

$$\Theta(s) = G \frac{1}{s} \frac{s+a}{s+\lambda a} E_d(s) \tag{12.77}$$

and from Figure 12.15 we have

$$\Theta(s) = \frac{c_3}{s} \frac{s c_2 + 1}{s - c_1} E_d(s) = c_2 c_3 \frac{1}{s} \frac{s + 1/c_2}{s - c_1} E_d(s) \tag{12.78}$$

From which

$$c_1 = -\lambda a, \quad c_2 = \frac{1}{a}, \quad c_2 c_3 = G, \quad c_3 = aG \tag{12.79}$$

Now, if we define the state variables as

$$\begin{aligned} x_1(t) &= x(t) \\ x_2(t) &= \dot{x}_1(t) \end{aligned} \tag{12.80}$$

we can, from (12.76) write the NLDE in the matrix form

$$\begin{aligned} \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} &= \begin{bmatrix} x_2 \\ \sin[\phi(t) - c_2 c_3 x_2(t) - c_3 x_1(t)] + c_1 x_2(t) \end{bmatrix} \\ &= \begin{bmatrix} f_1(x_1, x_2, \phi) \\ f_2(x_1, x_2, \phi) \end{bmatrix} \end{aligned} \tag{12.81}$$

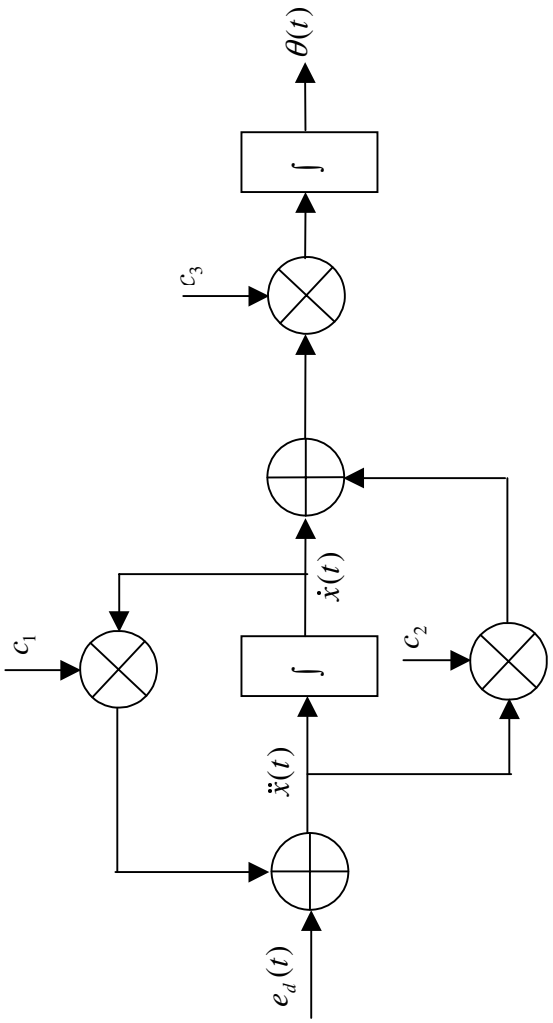


Figure 12.15 Representation of the loop filter and the definition of the state variables for the NLDE model of the PLL.

Note that the NLDE can also be expressed as

$$\dot{X} = F(X, t, U) \tag{12.82}$$

where X is a vector of size m, and F is a nonlinear vector function with m components and U is the input vector that in this example is the scalar $\phi(t)$.

In general, it is possible to convert an m^{th} order NLDE into a set of m simultaneous first-order NLDE that are similar in form to the state variable form of m^{th} order linear differential equations. We now look at methods of solving simultaneous (i.e., vector) first-order differential equations, starting with a scalar first-order differential equation.

12.4.2 Recursive Solutions of NLDE-Scalar Case

Explicit Techniques

Consider a first-order differential equation of the form

$$\dot{x}(t) = f(x, t, u) \tag{12.83}$$

with initial condition $x(t_0)$. Let x_n denote the solution obtained via numerical integration at time step t_n [we will use the notation $x(t_n)$ for the actual solution, which is not known, and use x_n to denote the approximate solution obtained via numerical integration].

Most of the numerical integration methods for solving nonlinear (or linear) differential equations are based on Taylor series expansions. For example, consider the Taylor series (or some other) expansion of the form

$$x(t_{n+1}) = x(t_n + h_n) = x(t_n) + h_n\dot{x}(t_n) + T(h_n) \tag{12.84}$$

where h_n is the time step $t_{n+1} - t_n$, and T is the local error or remainder given by

$$T(h_n) = h_n^2\ddot{x}(\xi)/2, \quad t_n \le \xi \le t_{n+1} \tag{12.85}$$

If h_n is sufficiently small, then we can obtain a recursive solution for the first-order differential equation as

$$x_{n+1} = x_n + h_n\dot{x}_n = x_n + h_n f_n \tag{12.86}$$

$$f_n = f(x_n, t_n, u(t_n)) \tag{12.87}$$

The recursion can be started with the initial condition $x(t_0) = x_0$. A simpler version of the recursive solution can be derived with a fixed step size h as

$$x_{n+1} = x_n + h\dot{x}_n = x_n + hf(x_n, t_n, u(t_n)) \tag{12.88}$$

$$t_{n+1} = t_n + h \tag{12.89}$$

Equations (12.88) and (12.89) define Euler's integration method.

By including the higher derivatives in the Taylor series, we can derive more accurate integration formulas. For example, a "second-order" integration rule may be derived by starting from

$$x_{n+1} = x_n + h\dot{x}_n + \frac{1}{2}h^2\ddot{x}_n \tag{12.90}$$

Using the approximation for the second derivative

$$\begin{aligned}\ddot{x}_n &\approx \frac{\dot{x}_n - \dot{x}_{n-1}}{h} = \frac{f(x_n, t_n, u(t_n)) - f(x_{n-1}, t_{n-1}, u(t_{n-1}))}{h} \\ &= \frac{f_n - f_{n-1}}{h}\end{aligned} \tag{12.91}$$

we obtain the two-step formula called the second-order Adam-Bashforth integration rule

$$x_{n+1} = x_n + hf_n + h^2\frac{f_n - f_{n-1}}{2h} = x_n + \frac{h(3f_n - f_{n-1})}{2} \tag{12.92}$$

Another similar class of integration rules are the various Runge-Kutta (R-K) methods. One of the most widely used of the R-K formula is the classical four-stage formula,

$$x_{n+1} = x_n + \frac{h(k_1 + k_2 + k_3 + k_4)}{6} \tag{12.93}$$

where

$$k_1 = f(x_n, t_n, u(t_n)) \tag{12.94}$$

$$k_2 = f(x_n + \frac{hk_1}{2}, t_n + \frac{h}{2}, u(t_n + \frac{h}{2})) \tag{12.95}$$

$$k_3 = f(x_n + \frac{hk_2}{2}, t_n + \frac{h}{2}, u(t_n + \frac{h}{2})) \tag{12.96}$$

$$k_4 = f(x_n + \frac{hk_3}{2}, t_n + \frac{h}{2}, u(t_n + \frac{h}{2})) \tag{12.97}$$

All three methods described above are called explicit methods, since the recursion is carried out explicitly, using the solution values and derivative values obtained at the preceding time step.

Implicit Techniques

There is another class of solution techniques, called implicit techniques, that yield better accuracy and more stable solutions at the expense of increased computational burden. In implicit techniques, the solution at time step t_{n+1} will involve not only the quantities computed at the previous time step t_n but also the quantities to be computed at the current time step t_{n+1}. This requires the solution of a nonlinear algebraic equation at each time step.

A very simple and very popular implicit technique is the Trapezoidal integration rule given by

$$\begin{aligned} x_{n+1} &= x_n + \frac{h}{2}\left[\dot{x}_n + \dot{x}_{n+1}\right] \\ &= x_n + \frac{h}{2}\left[f(x_n, t_n, u(t_n)) + f(x_{n+1}, t_{n+1}, u(t_{n+1}))\right] \end{aligned} \tag{12.98}$$

which is simply a trapezoidal approximation of the areas in the Reimann sum of an integral. (Recall our discussion of trapezoidal integration in Chapter 5.) It is clear that the preceding equation gives x_{n+1} only implicitly, and we have to solve the nonlinear equation given by (12.98) to obtain x_{n+1}, whereas in the explicit methods x_{n+1} does not appear within the nonlinear $f(\cdot)$ on the right-hand side, and hence the solution, as we have seen, is rather straightforward.

There are many other implicit methods besides the Trapezoidal rule. As another example of an implicit technique, let us consider the so-called third-order Adams-Moulton (A-M) method based on the Taylor series expansion

$$x(t_{n+1}) \approx x_n + h\dot{x}_n + \frac{h^2\ddot{x}_n}{2} + \frac{h^3 x_n}{6} \tag{12.99}$$

with a symmetric approximation for the derivatives

$$\dot{x}_n = f(x_n, t_n, u(t_n)) = f_n \tag{12.100}$$

$$\ddot{x}_n = \frac{\dot{x}_{n+1} - \dot{x}_{n-1}}{2h} \tag{12.101}$$

$$x_n = \frac{\dot{x}_{n+1} - 2\dot{x}_n + x_{n-1}}{h^2} \tag{12.102}$$

Substituting these derivatives in (12.99), we obtain the "two-step" Adams-Moulton integration formula

$$x_{n+1} = x_n + \frac{h(5f_{n+1} + 8f_n - f_{n-1})}{12} \tag{12.103}$$

The A-M integration formula given above points out two major problems with higher-order implicit techniques: initial conditions, and solving a nonlinear equation at each time step. When the recursion in (12.103) is applied to finding x_1 , we need $x_{0,}$ and x_{-1}. While the initial condition x_0 will be given, x_{-1} is not usually available. To avoid this problem we start the iterations at $n+1 = 2$, by using the value of x_1, which may be computed using an explicit method such as the R-K method.

The second difficulty arises because x_{n+1} appears on the right-hand side inside the (nonlinear) function f, and hence the solution for x_{n+1} will require the numerical solution of the implicit equation given in (12.103). We present below two iterative methods for accomplishing this.

Implicit Solution Using the Predictor-Corrector Method

The idea behind the predictor-corrector (P-C) method is to first obtain a predicted value for x_{n+1} using an explicit technique such as the R-K or the A-B method

(of the same order as the A-M method) and use this predicted (or the estimated) value of x_{n+1} in the right-hand side of (12.103) to obtain a corrected (or improved) value of x_{n+1}. We then we proceed to $n+2$. To improve the accuracy of the solution, one could use an interactive technique at each time step and repeat the predictor-corrector method many times. Now, (12.103) is modified as

$$x_{n+1}^{r+1} = x_n + \frac{h(5f_{n+1}^r + 8f_n - f_{n-1})}{12}, r = 1, 2, 3.. \tag{12.104}$$

where r is the iteration index. The iteration is started using a predicted value x_{n+1}^1 of x_{n+1} (obtained via an explicit method) on the right-hand side of (12.104) to obtain the next value in the iteration x_{n+1}^2, which is then used in the right-hand side again to obtain the next improved value and so on until the iterations converge. [Note: x_{n+1}^{r+1} is the $(r+1)^{\text{st}}$ iterated value of x_{n+1}, not the $(r+1)^{\text{st}}$ power of x_{n+1}!]

Implicit Solution Using Newton-Raphson Method

A variety of other techniques, such as the Newton-Raphson (N-R) method, can also be used for solving the implicit expression defined by (12.103). The N-R method is based on an iterative technique for finding a solution to $y = g(x) = 0$, that is, find the value of x that yields $g(x) = 0$.

An iterative solution to the "root-finding" problem can be obtained as follows. With reference to Figure 12.16, the slope of the curve $y = g(x)$ at x^0 is given by

$$\dot{g}(x^0) = \frac{AC}{BC} = \frac{g(x^0)}{x^0 - x^1} \tag{12.105}$$

We can rearrange the preceding equation as

$$x^1 = x^0 - \frac{g(x^0)}{\dot{g}(x^0)} \tag{12.106}$$

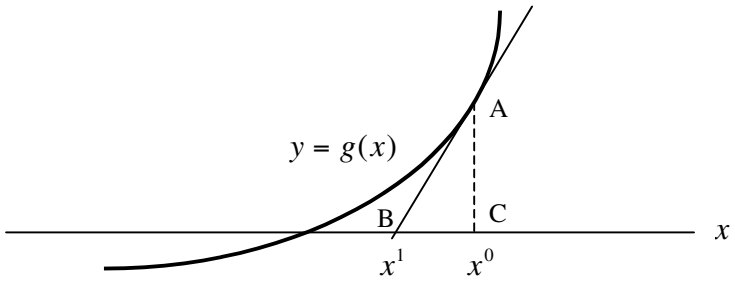


Figure 12.16 Iterative technique for solving $g(x) = 0$.

If this procedure is repeated to generate a sequence of values $x^1, x^2, x^3, \cdots$, by the recurrence relation

$$x^{r+1} = x^r - \frac{g(x^r)}{\dot{g}(x^r)} \tag{12.107}$$

the x^r converges to the zero of $g(x)$ under quite simple conditions.

This technique could be applied to solving (12.103), by first writing it as

$$g(x_{n+1}) = x_{n+1} - x_n - \frac{h(5f_{n+1} + 8f_n - f_{n-1})}{12} = 0 \tag{12.108}$$

and applying the recursion given in (12.107) as

$$x_{n+1}^{r+1} = x_{n+1}^r - \frac{g(x_{n+1}^r)}{\dot{g}(x_{n+1}^r)} \tag{12.109}$$

The iterations given in the preceding equation are repeated until the difference between successive values is small. The starting value x_{n+1}^1 of x_{n+1} is obtained using an explicit method.

12.4.3 General Form of Multistep Methods

The general form of the recursive solution of NLDE can be expressed as

$$x_{n+1} = \sum_{i=0}^{p} a_i x_{n-i} + h \sum_{j=-1}^{p} b_j f_{n-j} \tag{12.110}$$

which is referred to as a p-step integration rule and it is explicit or implicit depending on whether $b_{-1} = 0$ (explicit) or $b_{-1} \neq 0$ (implicit). Table 12.1 summarizes the different integration formulas that are commonly used in simulation. The truncation error given in the table is obtained from the truncation error associated with terminating the Taylor series expansion [see (12.84) and (12.85)].

12.4.4 Accuracy and Stability of Numerical Integration Methods

Accuracy

The truncation error at a given time step is proportional to the higher-order derivatives and the step size h. If it is possible to estimate the local truncation error at each step, then the step size can be adjusted up or down. In general, higher-order implicit methods are better, since they yield smaller error. However, implicit methods require solving a nonlinear equation at each step. Reducing step size reduces local truncation error, and it will also speed up the convergence of the iterative solution of the nonlinear equation at each time step. Reducing the time step, however, will increase the overall computational burden.

The truncation error contributes, in a cumulative fashion, to the overall (global) error. While it is possible to estimate the local truncation error, unfortunately no general procedures are available for controlling the possible growth of global error. Therefore, most of the methods rely on estimating the local truncation error and reducing the step size when necessary.

Table 12.1 Integration Functions Commonly Used in Simulation

Method	**Coefficients**	**Truncation Error**
Forward Euler	$p = 0; a_0 = 1; b_{-1} = 0; b_0 = 1$ $x_{n+1} = x_n + hf_n$	$\frac{1}{2}h^2\ddot{x}(\xi), \quad t_n \leq \xi \leq t_n + h$
Backward Euler	$p = 0; a_0 = 1; b_{-1} = 1; b_0 = 1$ $x_{n+1} = x_n + hf_{n+1}$	$\frac{1}{2}h^2\ddot{x}(\xi), \quad t_n \leq \xi \leq t_n + h$
Trapezoidal	$p = 0; a_0 = 1; b_{-1} = \frac{1}{2}; b_0 = \frac{1}{2}$ $x_{n+1} = x_n + \frac{h}{2}[f_n + f_{n+1}]$	$\frac{1}{12}h^3\dddot{x}(\xi)$
Adam-Bash-2	$p = 0; a_0 = 1;$ $b_{-1} = 0; b_0 = \frac{3}{2}; b_1 = -\frac{1}{2}$ $x_{n+1} = x_n + \frac{h}{2}[3f_n - f_{n+1}]$	$\frac{5}{12}h^3\dddot{x}(\xi)$
Adam-Moulton-2	$p = 1; a_0 = 1;$ $b_{-1} = \frac{5}{12}; b_0 = \frac{8}{12}; b_1 = -\frac{1}{12}$ $x_{n+1} = x_n + \frac{h}{12}[5f_{n+1} + 8f_n - f_{n-1}]$	$\frac{1}{24}h^4x^{(4)}(\xi)$

Stability

Another measure of the "goodness" of an integration method is stability, which is a measure of the degree to which the recursive solution converges to the true solution. Stability depends on the specific problem and the integration method, and it is common practice to investigate the stability of different integration methods using a simple test problem like

$$\dot{x} = \lambda x$$

for which the solution is known to be

$$x(t) = \exp(\lambda t)$$

If $\mathrm{Re}(\lambda) < 0$, the real solution tends to zero as $t \to \infty$. With this test case, an integration rule is said to be stable if the recursive solution converges to zero as $n \to \infty$.

While it is easy to investigate the stability of the integration methods for the simple test case, it is difficult to draw general conclusions about stability of a method when applied to an arbitrary nonlinear differential equation. A general rule of thumb is that the implicit methods have better stability properties than explicit methods.

It is difficult to track truncation error and stability of the solution of a NLDE, since tracking the truncation error requires knowledge of the higher-order derivatives, and analysis of stability requires knowledge of the roots of the characteristic equation of the linearized version of the NLDE in the vicinity of t_n. During simulations, an often-used heuristic method consists of comparing the solutions obtained with time steps h and $h/2$. If the estimated error associated with both time steps is within specified limits, then the solution is accepted. If not, the time step is halved again and the procedure is repeated. This method controls both stability and truncation error.

Among the multitude of integration techniques that are available, it is impossible to say which method is the "best" because the answer depends to a large extent on the problem being considered, the accuracy desired, the type of output needed, and the computational burden imposed by the integration technique. Trapezoidal rule, fourth-order Runge-Kutta method, and the Adam-Moulton method (with predictor-corrector) are the most commonly used methods offering a good tradeoff between computational complexity and stability and accuracy (with the Trapezoidal method being the least complex). For a reasonably well behaved nonlinear components or subsystems system like the PLL, any one of these methods with a small step size (of the order of eight samples/Hz of bandwidth) will provide a stable and accurate solution.

12.4.5 Solution of Higher-Order NLDE-Vector Case

If the NLDE model is m^{th} order, then it can be converted to m simultaneous first-order differential equations (as described in Section 12.4.1), in vector form as

$$\dot{\mathbf{X}} = F(\mathbf{X}, t, \mathbf{U})$$

where $\mathbf{X}$ and $\mathbf{U}$ are vectors of size $m \times 1$. The multistep iterative solution of the vector NLDE is of the same form as the scalar case, and is given by

$$X_{n+1} = \sum_{i=0}^{p} a_i X_{n-i} + \sum_{j=-1}^{p} b_j \dot{X}_{n-j} = \sum_{i=0}^{p} a_i X_{n-i} + \sum_{j=-1}^{p} b_j F_{n-j}$$

where a_i and b_j are the same scalar constants given in Table 12.1.

For the implicit methods, a set of simultaneous nonlinear equations have to be solved in each step. Either the predictor-corrector method or the Newton-Raphson method can be used for this. The vector form of the Newton-Raphson method is given by

$$G(X_{n+1}) = X_{n+1} - \sum_{i=0}^{p} a_i X_{n-i} + \sum_{j=-1}^{p} b_j F_{n-j} \tag{12.111}$$

and

$$X_{n+1}^{r+1} = X_{n+1}^{r} - J^{-1}(X_{n+1}^{r}) G(X_{n+1}^{r}), \quad r = 1, 2, 3, \ldots \tag{12.112}$$

where the $(i,j)^{\text{th}}$ entry in the $m \times m$ Jacobian matrix, $J(\cdot)$, is defined as

$$[J(X_{n+1}^{r})]_{i,j} = \frac{\partial g_i(X_{n+1}^{r})}{\partial x_j} \tag{12.113}$$

The starting value X_{n+1}^{1} of X_{n+1} is obtained via an explicit method. At each time step, t_{n+1}, the iterations are carried out (the iteration index is r) until subsequent iterated solutions differ by a small amount. Then the time index is advanced to t_{n+2}, an initial value X_{n+2}^{1} of X_{n+2} is obtained using an explicit method, and an iterated solution for X_{n+2} is obtained. The procedure is repeated until the time index reaches the simulation stop time.

12.5 PLL Example

We now return our original PLL problem introduced in Chapter 6 and illustrate the simulation of the PLL using several of the integration techniques discussed in the previous section.

12.5.1 Integration Methods

The techniques to be used are defined in terms of the PLL problem in the following sections.

Forward Euler (Explicit Method)

The defining equations are as follows:

$$t_{n+1} = t_n + h \tag{12.114}$$

$$x_{1,n+1} = x_{1,n} + hx_{2,n} \tag{12.115}$$

and

$$x_{2,n+1} = x_{2,n} + h\left[\sin\left(\phi_n - c_1 x_{2,n} - c_2 x_{1,n}\right) + c_3 x_{2,n}\right] \tag{12.116}$$

Backward Euler (Implicit Method with Predictor-Corrector)

The predictor is defined by

$$x^1_{1,n+1} = x_{1,n} + hx_{2,n} \tag{12.117}$$

and

$$x^1_{2,n+1} = x_{2,n} + h\left[\sin\left(\phi_n - c_1 x_{2,n} - c_2 x_{1,n}\right) + c_3 x_{2,n}\right] \tag{12.118}$$

and the corrector is defined by

$$x^{r+1}_{1,n+1} = x^r_{1,n} + hx^r_{2,n}, \quad r = 1, 2, \cdots \tag{12.119}$$

and

$$x^{r+1}_{2,n+1} = x^r_{2,n} + h\left[\sin\left(\phi_{n+1} - c_1 x^r_{2,n+1} - c_2 x^r_{1,n+1}\right) + c_3 x^r_{2,n+1}\right], \quad r = 1, 2, \cdots \tag{12.120}$$

Backward Euler (Implicit Method with N-R Iterations)

The predictor is the same as with the previous backward Euler [see (12.117) and (12.118)]. The N-R iterations for X_{n+1} are defined by

$$X^{r+1}_{n+1} = X^r_{n+1} - J^{-1}\left(X^r_{n+1}\right) G\left(X^r_{n+1}\right) \tag{12.121}$$

where

$$\begin{aligned} G\left(X^r_{n+1}\right) &= \begin{bmatrix} g_1\left(X^r_{n+1}\right) \\ g_2\left(X^r_{n+1}\right) \end{bmatrix} \\ &= \begin{bmatrix} x^r_{1,n+1} - x^r_{1,n} - hx^r_{2,n+1} \\ x^r_{2,n+1} - x^r_{2,n} + h\left[\sin\left(\phi_n - c_1 x^r_{2,n} - c_2 x^r_{1,n}\right) + c_3 x^r_{2,n}\right] \end{bmatrix} \end{aligned} \tag{12.122}$$

and $J^{-1}\left(X^r_{n+1}\right)$ is the 2×2 matrix

$$J^{-1}\left(X^r_{n+1}\right) = \begin{bmatrix} e_{11} & e_{12} \\ e_{21} & e_{22} \end{bmatrix} \tag{12.123}$$

with elements

$$e_{11} = 1 \tag{12.124}$$

$$e_{12} = -h \tag{12.125}$$

$$e_{21} = -c_2 h\left[\cos\left(\phi_{n+1} - c_1 x^r_{2,n+1} - c_2 x^r_{1,n+1}\right) + c_3 x^r_{2,n+1}\right] \tag{12.126}$$

$$e_{22} = 1 - h\left[c_1 \cos\left(\phi_{n+1} - c_1 x^r_{2,n+1} - c_2 x^r_{1,n+1}\right) + c_3\right] \tag{12.127}$$

It is left as an exercise for the reader to derive similar formulas for the A-M method with A-B as a predictor and corrector iterations or N-R iterations.

12.6 Summary

Simulation plays an important role in the analysis of nonlinear components and their impact on communication system performance. While the performance of linear systems can, in principle, be attacked by analytical means, the analysis of the impact of nonlinear components in a system context is by and large intractable. Simulation may be the only approach for tackling these problems without actually building the systems and testing them.

In this chapter we developed modeling and simulation techniques for nonlinear components in communication systems such as power amplifiers and limiters. For bandpass nonlinearities it was shown that the simulation can be carried out either with bandpass or with lowpass equivalent models. The complex lowpass equivalent model will be computationally more efficient. With either model, simulation of nonlinear components is usually carried out on a sample-by-sample basis in time domain.

Bandpass nonlinearities may be frequency nonselective (memoryless) or frequency selective (with memory). The lowpass equivalent model for memoryless bandpass nonlinear components in communication systems can be derived analytically or obtained from swept power measurements. For devices such as bandpass limiters, the lowpass equivalent model can be analytically derived using the Fourier integrals and the resulting model is expressed in terms of the AM-to-AM $[f(A)]$ and AM-to-PM $[g(A)]$ transfer characteristics. For devices such as high-power amplifiers, the AM-to-AM and AM-to-PM characteristics are directly obtained from measurements. These transfer characteristics can be stored in empirical form in tables or can be approximated by functional forms such as the ones given in (12.44) and (12.45). Simulation of the AM-to-AM and AM-to-PM models consist of generating sampled values of the input complex envelope and obtaining the sampled values of the output complex envelope as shown in Figure 12.7.

For bandpass nonlinear devices operating over wide bandwidths, it may be necessary to use frequency-selective models. The preferred simulation model for frequency-selective nonlinearities consists of two filters with a memoryless nonlinearity sandwiched in between. The transfer function of the filters and the transfer characteristics of the memoryless nonlinearity can be obtained from swept power/swept frequency measurements. Once the structure and parameters of the model are specified, simulation is rather straightforward.

An alternate approach for simulating nonlinearities is based on representing the dynamic behavior of the system by a set of nonlinear differential equations and solving them using numerical integration techniques. This method is very useful for simulating such devices as PLLs and is very efficient computationally.

12.7 Further Reading

An excellent and detailed treatment of the topics covered in this chapter and additional material on simulation of nonlinear components may be found in

M. C. Jeruchim, P. Balaban, and K. S. Shanmugan, *Simulation of Communication Systems*, 2nd ed., Kluwer Academic/Plenum Publishers, 2000, Chap. 5.

The literature on nonlinear systems is rather broad and is also very application dependent. The literature on this topic can be divided into the following categories:

1. Mathematical and numerical techniques for solving nonlinear differential equations
2. Nonlinear circuit analysis techniques
3. Nonlinear control systems
4. Modeling and simulation of nonlinear components in communication systems

A selected set of papers from the last group are given in the list of references.

12.8 References

1. R. Deutsch, *Nonlinear Transformations of Random Signals*, New York: Prentice Hall, 1962.
2. E. Bedrosian and S. O. Rice, "The Output Properties of Volterra Systems (Nonlinear Systems with Memory) Driven by Harmonic and Gaussian Inputs," *Proceedings of the IEEE*, Vol. 59, December 1971, pp. 1688–1707.
3. N. M. Blachman, "Bandpass Nonlinearities," *IEEE Transactions on Information Theory*, Vol. IT-10, April 1974, pp. 162–164.
4. N. M. Blachman, "Detectors, Bandpass Nontinearities and the Chebyshev Transform," *IEEE Transactions on Information Theory*, Vol. IT-17, No. 4, July 1971, pp. 398–404.
5. G. L. Wise, A. P. Traganitis, and J. B. Thomas, "The Effect of a Memoryless Nonlinearity on the Spectrum of a Random Process," *IEEE Transactions on Information Theory*, Vol. IT-23, No. 1, January 1977, pp. 84–89.
6. J. S. Bendat, *Nonlinear Systems Analysis and Identification*, New York: Wiley, 1990.
7. O. Shimbo, *Transmission Analysis in Communication Systems*, Vols. 1 and 2, Rockville, MD: Computer Science Press, 1988.
8. L. C. Palmer and S. Lebowitz, "Computer Simulation of Solid-State Amplifiers," *COMSAT Technical Review*, Vol. 8, No. 2, Fall 1978, pp. 371–404.
9. K. W. Schneider and W. H. Tranter, "Efficient Simulation of Multi-Carrier Digital Communication Systems in Nonlinear Channel Environments," *Proceedings of the IEEE 1992 Milcom Conference*, San Diego, CA, October 11–14, 1992.

10. A. A. M. Saleh, "Frequency-Independent and Frequency-Dependent Nonlinear Models of TWT Amplifiers," *IEEE Transactions on Communications*, Vol. C0M-29, No. 11, November 1981, pp. 1715–1720.

11. H. B. Poza, Z. A. Sarkozy, and H. L. Berger, "A Wideband Data Link Computer Simulation Model," *Proceedings of the 1975 NAECON Conference*, Dayton, OH, June 10–12, 1975.

12. R. Blum and M. C. Jeruchim, "Modeling Nonlinear Amplifiers for Communication Simulation," *Proceedings of the 1989 IEEE International Conference on Communications*, Boston, MA, June 1989.

13. W. Bosch and G. Gatti, "Measurement and Simulation of Memory Effects in Predistortion Linearizers," *IEEE Transactions on Microwave Theory and Techniques*, Vol. 37, No. 12, December 1989 , pp. 1885–1890.

14. D. Quinney, *An Introduction to the Numerical Solution of Differential Equations*, New York: Wiley, 1987.

12.9 Problems

12.1 Derive the complex lowpass equivalent (CPE) model for the cubic nonlinearity of the form

$$y(t) = x(t) - 0.25x^3(t)$$

For a two-tone input of equal amplitudes at $f_1 = 13$ Hz and $f_2 = 15$ Hz, find the first-zone output for the bandpass case analytically and compute the power levels of the various output components. Simulate the bandpass and the LPE models and compare the simulated power levels with the computed power levels. Do this for at least three different sampling rates and compare the results.

12.2 Show that a second-order nonlinearity does not produce any first-zone intermodulation products (use a two-tone input as in Problem 12.1).

12.3 Show that $f_2(A) = 0$ in the LPE model for an arbitrary memoryless nonlinearity $y(t) = g(x(t))$.

12.4 Derive the LPE model for a bandpass soft limiter and simulate the bandpass and LPE models with the same input as in Problem 12.1 and compare the results.

12.5 Derive the LPE model for the nonlinearity $y(t) = |x(t)|$.

12.6 Derive the LPE model for the limiter given in (12.8) and plot A versus $f(A)$. Note that the Fourier integral will have to be evaluated using numerical integration procedures.

12.7 Experimental tests performed on a TWT amplifier results in the data given in Table 12.2.

(a) Implement a simulation model for a TWT amplifier specified by the AM-to-AM and AM-to-PM data given in Table 12.2.

(b) Simulate this model with a 64 QAM input for backoffs of −10 and −20 dB and compare the distortion in the signal constellations.

(c) Repeat the simulations for the two-tone input described in Problem 12.1 and observe the intermodulation components and their power levels as a function of backoff.

Table 12.2 Experimental TWT Data

Input Power	Output Power	Phase (degrees)	Input Power	Output Power	Phase (degrees)
0	0	32.8510	0.2230	0.6570	17.3930
0.0110	0.0500	32.8510	0.2510	0.6970	16.0620
0.0120	0.0560	32.7570	0.2820	0.7370	14.8880
0.0140	0.0630	32.6540	0.3160	0.7770	13.5740
0.0150	0.0700	32.4840	0.3540	0.8130	12.2540
0.0170	0.0790	32.3210	0.3980	0.8480	11.0110
0.0190	0.0880	32.1930	0.4460	0.8770	9.7070
0.0220	0.0980	31.9740	0.5010	0.9050	8.3830
0.0250	0.1100	31.5650	0.5620	0.9320	6.9150
0.0280	0.1230	31.2160	0.6300	0.9510	5.4690
0.0310	0.1370	30.8170	0.7070	0.9700	41.1550
0.0350	0.1520	30.4400	0.7950	0.9810	2.8770
0.0390	0.1690	30.0840	0.8910	0.9920	1.4660
0.0440	0.1890	29.6240	1.0000	1.0000	0
0.0500	0.2100	29.1450	1.1220	0.9980	-1.5710
0.0560	0.2330	28.6350	1.2580	0.9980	-3.0480
0.0630	0.2570	28.0410	1.4120	0.9890	-4.7500
0.0700	0.2840	27.3180	1.5840	0.9820	-6.5670
0.0790	0.3130	26.3990	1.7780	0.9760	-8.3850
0.0890	0.3440	25.6150	1.9950	0.9700	-10.2020
0.1000	0.3770	24.8700	2.2480	0.9630	-12.0190
0.1120	0.4130	23.9820	2.5110	0.9570	-13.8360
0.1250	0.4510	23.0500	2.8120	0.9510	-15.6530
0.1410	0.4910	22.0380	3.1620	0.9450	-17.4710
0.1580	0.5320	21.0060	3.5480	0.9380	-19.2880
0.1770	0.5730	19.9700	3.9810	0.9320	-21.1050
0.1990	0.6150	18.6400	4.4660	0.9260	-22.9220

12.8 Set up a four-carrier QPSK input into the TWTA model specified by Table 12.2. The symbol rate for each QPSK signal is 1 symbol/sec and the lowpass equivalent carrier frequencies are $f_1 = -2.5$ Hz, $f_2 = -1.25$ Hz, $f_3 = 1.25$ Hz, and $f_4 = 2.5$ Hz. Assume that each QPSK signal is filtered by an SQRC filter with a roll off factor of 25%. Plot the power spectral density at the output of the TWT and observe the spectral regrowth in the vicinity of the tails and at $f = 0$ as the backoff is varied. Explain the spectral regrowth due to the nonlinearity in general and the intermodulation products in particular. Where would you expect the IM to be worst? Repeat for different carrier spacing and backoffs.

12.9 Fit a Saleh's functional form to the AM-to-AM and AM-to-PM data given in Table 12.2.

12.10 Implement a simulation model for the second-order PLL using the various numerical integration techniques given in Table 12.1. Simulate a second-order PLL using the same parameter values as the example in Chapter 6 and compare the results.

12.10 Appendix A: Saleh's Model

```
% File: salehs_model.m
function [y]=salehs_model(x,backoff,n)
% This function implements Saleh's model
% x is the complex input vector of size n;
% Back-off is in db; the input amplitude is scaled by
% c=10^(backoff/20);
% The maximum normalized input power should be less than 3 dB
% i.e., 20 log10(a*abs(x)) < 3 dB

y = zeros(1,n)*(1.0+i*1.0);          % initialize output array
a1=2.1587; b1=1.15;                  % model parameters
a2=4.0; b2=2.1;                      % model parameters
c=10^(backoff/20);                   % backoff in dB
for k=1:n
    ain = c*abs(x(k));
    thetain(k) = angle(x(k));
    aout = a1*ain/(1+b1*ain^2);
    thetapm(k) = a2*ain^4/(1+b2*ain^2);
    thetaout(k) = thetain(k)+thetapm(k);
    y(k) = aout*exp(i*thetaout(k));
end;
% End of function file.
```

12.11 Appendix B: MATLAB Code for Example 12.2

```
% File: c12_example2.m
% This example implements both Lowpass and bandpass versions of a
% power series nonlinearity of the form y(t) = x(t) - a3*x(t)^3
% For the BP Model f1=11Hz; f2=14Hz; IM @ 8 and 17 Hz
% The 3rd harmonics are at 33 and 42 Hz
%
% For the LPE model the ref freq is f0 =10Hz
% Hence f1=-1 and f2=2 Hz; IM @ -4 and 5Hz
% Input parameters: None;
% Plots: BP input; BP output; LPE input; LPE Output
f1=11.0; f2=14.0; ts=1.0/128; n=1024; a3=0.3;
% Generate input samples
for k=1:n
 t(k)=(k-1)*ts;
 x(k) = cos(2*pi*f1*t(k))+0.707*cos(2*pi*f2*t(k));
end
% Generate output samples
for k=1:n
 y(k)=x(k)-a3*x(k)^3;
end;
% Plot the results
[psdx,freq]=log_psd(x,n,ts); [psdy,freq]=log_psd(y,n,ts);
subplot(2,1,1)
plot (freq,psdx,'b'); grid;
ylabel('PSD in dB')
title('BP Input @ f1 = 11 and f2=14');
subplot(2,1,2)
plot (freq,psdy,'b'); grid;
xlabel('Frequency in Hz')
ylabel('PSD in dB')
title('BP output: IM @ 8 and 17 and Third harmonics')
%
% This Section of the model implements the LPE model for the 3-rd
% order power series nonlinearity.
% Baseband model: y(t) = x(t) - a3*x(t)^3.
% LPE Model: y(k)=x(k)+0.75*a3*(abs(x(k))^2)*x(k);
% Generate LPE of the input signals using 12Hz as the ref frequency
% And generate output samples using the LPE model
f1=-1.0; f2=2.0;
for k=1:n
 t(k)=(k-1)*ts;
 x(k) = exp(i*2*pi*f1*t(k))+0.707*exp(i*2*pi*f2*t(k));
 y(k)=x(k)+0.75*a3*(abs(x(k))^2)*x(k);
```

```
end
% Plot the results
[psdx,freq]=log_psd(x,n,ts); [psdy,freq]=log_psd(y,n,ts);
figure;
subplot(2,1,1)
plot(freq,psdx); grid;
ylabel('PSD in dB')
title('LP Equivalent input; f0=12; f1=-1 and f2 = 2');
subplot(2,1,2)
plot(freq,psdy); grid;
xlabel('Frequency in Hz')
ylabel('PSD in dB')
title('LP Output IM at 2f1-f2= -4 and 2f2-f1 =5')
% End of script file.
```

12.11.1 Supporting Routines

Program `log_psd.m` is defined in Appendix A of Chapter 7.

Chapter 13

MODELING AND SIMULATION OF TIME-VARYING SYSTEMS

Up to this point in our studies, all components in the system under study, and the resulting models for those components, have been fixed or time-invariant. Linear time-invariant (LTIV) models, and their application to the modeling and simulation of system elements such as filters, were discussed in detail in the preceding chapters. Linear elements in a communication system can be time invariant or time varying in nature. We now remove the restriction of time invariance and turn our attention to the modeling and simulation of time-varying systems. As we will see, there are many examples of time-varying systems. Of particular interest later in our studies, will be the time-varying channel that is encountered in the mobile radio system.

13.1 Introduction

The assumption of time invariance implies that the properties of the system being modeled do not change over time. If we are modeling a time-invariant system by

a transfer function, time invariance requires that the transfer function (both magnitude and phase) remain fixed as a function of time. Consider, for example, the model of a third-order Butterworth filter with a 3-dB bandwidth of 1 MHz. Time invariance implies that neither the order nor the bandwidth of the filter changes as a function of time. The filter is only time invariant if the values of all physical components, such as the resistors, capacitors, and the parameters of active components, such as the gain of an operational amplifier, do not change with time. While the time-invariance assumption may hold over a shorter periods of time, component values do change over longer periods of time due to aging. (Recall the role of end-of-life performance predictions discussed in Chapter 1.) From a practical point of view, whether or not a system can be considered time invariant depends not only on the system but also on the nature of the problem being solved.

Whether to use a time-invariant or time-varying system model is usually determined by the rate at which the characteristics of the communication system being modeled are changing in comparison to other parameters of the communication system such as the symbol rate. As an example, if the time constant associated with the system time variations is very large compared to the symbol rate, then a time-invariant model may be justified. On the other hand, if the system parameters are changing at a rate approaching the symbol rate, a time-varying model is appropriate. Thus, there is a notion of "slow" or "fast" variations, compared to the symbol rate, or some other attribute of the system that influences the choice of a time-varying or time-invariant model. We further examine this point using two examples.

13.1.1 Examples of Time-Varying Systems

Consider a microwave radio communication system in which the transmitting and receiving antennas are located on fixed microwave towers. The "channel" between these antennas is the atmosphere, and the changes in the channel characteristics are due to changes in the atmospheric conditions, which typically have time constants on the order of minutes or hours. If the communication link is operating at a symbol rate of 100 Mbit/s, the time constant associated with the channel variations is very long compared to the symbol time of 10^{-8} s. Indeed, the channel will remain in nearly the same state while billions of symbols flow over the link. If the objective of a system simulation is estimation of the bit error rate (BER), the channel can be assumed to be in a "static" state and a time-invariant model can be used during the time interval over which the BER is estimated. The resulting BER estimate, of course, is valid only for the particular channel state used in the simulation. The long-term behavior of the channel, and its impact on long-term system performance, can be evaluated by analyzing system performance over a series of "snapshots" of static channel conditions, using a different time-invariant model for each snapshot. This is illustrated in Figure 13.1, where three simulations are developed for the three values of channel attenuation shown. From such simulations, one can obtain performance measures such as "outage probabilities," which describe the portion of time during which the channel performance might fall below some BER threshold. If $10,000$ channel snapshots are simulated and 100 of these channel conditions produce a BER

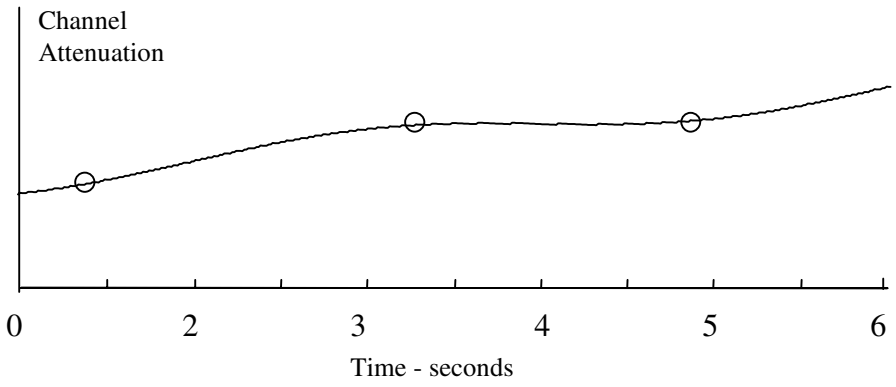


Figure 13.1 Snapshots of a slowly varying time-varying channel.

corresponding to unsatisfactory system performance, BER$> 10^{-3}$, for example, then the outage probability is $(100/10,000) = 0.01$ for this specific BER threshold.

As a second example, consider a mobile communication system consisting of a fixed base station and a mobile user. The characteristics of the communication channel between the transmitter and the receiver will be time varying, since the parameters of the channel, such as the attenuation and delay, are changing due to relative motion between the base station and the mobile user. In addition, changes in atmospheric conditions will also contribute to the time-varying nature of the channel. If the mobile user is rapidly moving and if the symbol rate is of the order of $10,000$ symbols per second, the rate at which channel conditions are changing might be comparable to the symbol rate. In this case a time-varying channel model would be required. While a time-varying model may or may not be needed for BER estimation, such a model will be necessary to study the behavior of receiver subsystems such as synchronizers and equalizers.

13.1.2 Modeling and Simulation Approach

As with LTIV systems, LTV systems can be modeled and simulated in either the time domain or in the frequency domain. The time-domain approach leads to a model consisting of a tapped delay-line structure with time-varying tap gains. This model is very easy to implement for simulation purposes and is computationally very efficient if the time-varying impulse response is relatively short.

Many of the modeling and simulation concepts previously discussed for LTIV systems apply to LTV systems, but with some important differences. Particular attention must be paid to the sampling rate used in the simulation, since an increase in the sampling rate will be required because of bandwidth expansion resulting from underlying time variations. One source of bandwidth expansion is the "doppler" spreading in a mobile communications system. In addition, caution

must be exercised in simplifying the block diagrams of LTV systems, since LTV blocks do not obey commutative properties and hence the order of computations cannot be interchanged between LTV blocks as with LTIV blocks.

However, as long as the time-varying system is linear in nature, superposition and convolution apply, and many of the time-domain and frequency-domain analysis techniques developed for LTIV systems can be used, with slight modifications, to model and simulate LTV systems. We will also use equivalent lowpass signal and system representations as we develop simulation models and techniques for time-varying systems.

13.2 Models for LTV Systems

In the time domain, a linear time-invariant system is described by a complex envelope impulse response $\widetilde{h}(\tau)$, where $\widetilde{h}(\tau)$ is defined as the response of the system at time τ to an impulse applied at the input at time $t = 0$. The variable τ represents the "elapsed time," which is the difference between the time at which the impulse response is measured and the time at which the impulse is applied at the system input. The complex envelope input-output relationship for a LTIV system is given by the familiar convolution integral

$$\widetilde{y}(t) = \int_{-\infty}^{\infty} \widetilde{h}(\tau)\widetilde{x}(t-\tau)\,d\tau \tag{13.1}$$

where $\widetilde{x}(t)$ and $\widetilde{y}(t)$ represent the complex envelopes of the system input and output, respectively.

Taking the Fourier transform of (13.1) gives the input-output relationship for the LTIV system in the frequency domain. This is

$$\widetilde{Y}(f) = \widetilde{X}(f)\widetilde{H}(f) \tag{13.2}$$

where $\widetilde{H}(f)$ is the transfer function of the system, and $\widetilde{X}(f)$ and $\widetilde{Y}(f)$ are the Fourier transforms of the input and output, respectively. The output $\widetilde{y}(t)$ is obtained by taking the inverse transform of $\widetilde{Y}(f)$. This gives

$$\begin{aligned}\widetilde{y}(t) &= \int_{-\infty}^{\infty} \widetilde{Y}(f)\exp(j2\pi ft)\,df \\ &= \int_{-\infty}^{\infty} \widetilde{X}(f)\widetilde{H}(f)\exp(j2\pi ft)\,df\end{aligned} \tag{13.3}$$

The preceding expressions serve as the starting point for deriving models for time-varying systems.

13.2.1 Time-Domain Description for LTV System

Time-varying systems are also characterized in the time domain by an impulse response. For time-varying systems the impulse takes the form $\widetilde{h}(\tau, t)$, which is defined as the response of the system measured at time t, to an impulse applied at

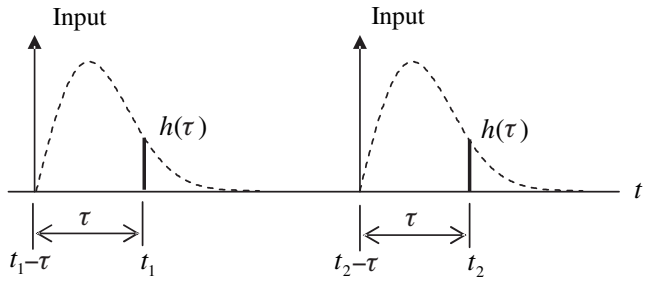
the input τ seconds earlier. In other words, the impulse is applied at the input at time $t-\tau$ and the response is measured at time t, after an "elapsed time" of τ. Since the system is time-varying, the impulse response will change as a function of both the time at which the impulse is applied, $t-\tau$, and the time at which the output is measured, t. For a time-invariant system, the impulse response will strictly be a function of the elapsed time τ and it is therefore represented by $\widetilde{h}(\tau)$. Note that the impulse response of a time-varying system is a function of two arguments, while the impulse response of a time-invariant system has one argument. Figure 13.2 depicts the impulse response of both time-invariant and time-varying systems.

While the impulse response of a LTIV system maintains the same functional form irrespective of when the impulse is applied at the input, the impulse response of a LTV system depends on when the input is applied. In other words, for an LTIV system

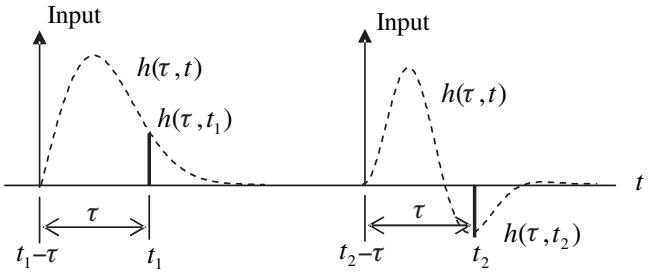
$$\widetilde{h}(\tau, t_1) = \widetilde{h}(\tau, t_2) = \widetilde{h}(\tau) \tag{13.4}$$

but for an LTV system

$$\widetilde{h}(\tau, t_1) \neq \widetilde{h}(\tau, t_2) \tag{13.5}$$



(a) Impulse response of time-invariant system.



(b) Impulse response of time-varying system.

Figure 13.2 Impulse response of a time-invariant and a time-varying system.

In addition, the response of an LTIV system to an arbitrary input remains the same irrespective of when the input is applied to the system except for a time delay. If the input $\widetilde{x}(t)$ produces an output $\widetilde{y}(t)$, then the same input applied t_0 seconds later, $\widetilde{x}(t-t_0)$, will produce a delayed version, $\widetilde{y}(t-t_0)$, of $\widetilde{y}(t)$ at the system output. In a time-varying system, this will not be the case, and the responses due to identical inputs may be entirely different if identical inputs are applied to the system at different times.

Note that the important difference between the impulse response of an LTIV system and an LTV system is that the impulse response for a time-invariant system is strictly a function of the elapsed time and not the time at which the input is applied or the time at which the output is observed. The impulse response of a time-varying system, on the other hand, is a function of both the elapsed time, τ, and the observation time, t.

The two time variables τ and t in $\widetilde{h}(\tau,t)$ characterize two different aspects of the system. The variable τ has the same role as the τ variable in the impulse response $\widetilde{h}(\tau)$ of time-invariant systems for which the Fourier transform associated with this variable carries the notions of transfer function, frequency response, and bandwidth. For an LTV system, one can develop the notion of a transfer function, although it is a time-varying transfer function $\widetilde{H}(f,t)$, by simply taking the Fourier transform of $\widetilde{h}(\tau,t)$ with respect to τ as

$$\widetilde{H}(f,t)=\int_{-\infty}^{\infty}\widetilde{h}(\tau,t)\exp(-j2\pi f\tau)\,d\tau \tag{13.6}$$

If the system is "slowly time-varying," then the concepts of frequency response and bandwidth can be applied to $\widetilde{H}(f,t)$. Whereas the LTIV system is characterized by a single impulse response function and a single transfer function, the LTV system is characterized by a family of impulse response functions and transfer functions, with one function for each value of t. If $\widetilde{h}(\tau,t)=\widetilde{h}(\tau)$ or equivalently $\widetilde{H}(f,t)=\widetilde{H}(f)$, then the system is time invariant.

The variable t in $\widetilde{h}(\tau,t)$ and $\widetilde{H}(f,t)$ describes the time-varying nature of the system. Strong dependence on t, and fast changes associated with t, indicate a rapidly time-varying system. Usually, the time-varying nature of the system is modeled as a random phenomenon, and $\widetilde{h}(\tau,t)$ is treated as a random process in t. If the process is stationary, then the time variations can be modeled by an appropriate autocorrelation function in the time domain or by a corresponding power spectral density in the frequency domain. The time constant of the autocorrelation function or the bandwidth of the power spectral density are key parameters that describe whether $\widetilde{h}(\tau,t)$ is slowly or rapidly time-varying.

From the definition of the impulse response of LTV systems, it is easy to see that the input-output relationship can be expressed by the convolution integral

$$\begin{aligned}\widetilde{y}(t)&=\int_{-\infty}^{\infty}\widetilde{h}(\tau,t)\widetilde{x}(t-\tau)\,d\tau\\&=\int_{-\infty}^{\infty}\widetilde{h}(t-\tau,t)\widetilde{x}(\tau)\,d\tau\end{aligned} \tag{13.7}$$

The frequency domain version of the input-output relationship is somewhat more complicated, as shown in the following section.

13.2.2 Frequency Domain Description of LTV Systems

As a starting point for the frequency domain description of LTV systems, let us define a two-dimensional Fourier transform of $\widetilde{h}(\tau, t)$ as

$$\widetilde{H}(f_1, f_2) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \widetilde{h}(\tau, t) \exp(-j2\pi f_1 \tau - j2\pi f_2 t)\, d\tau\, dt \tag{13.8}$$

In defining this two-dimensional Fourier transform, the usual finite energy assumption is made about the impulse response function to ensure existence of the Fourier transform. However, we will see later that for channel modeling, $\widetilde{h}(\tau, t)$ will be treated as a stationary random process in t in which case the Fourier transform of $\widetilde{h}(\tau, t)$ may not exist with respect to t. The autocorrelation function, of course, will exist and the appropriate procedure is to define the autocorrelation of $\widetilde{h}(\tau, t)$ with respect to the t variable, and then take the Fourier transform of the autocorrelation function to obtain the frequency domain representation. The result is the power spectral density of the random process. It should also be noted that if the system is time invariant, then $\widetilde{h}(\tau, t) = \widetilde{h}(\tau)$, and $\widetilde{H}(f_{1,} f_2) = \widetilde{H}(f_1)\delta(f_2)$.

Taking the inverse transform of $\widetilde{H}(f_1, f_2)$ gives

$$\widetilde{h}(\tau, t) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \widetilde{H}(f_{1,} f_2) \exp(j2\pi f_1 \tau + j2\pi f_2 t)\, df_1\, df_2 \tag{13.9}$$

Substituting $\widetilde{h}(\tau, t)$ in (13.7), shows that

$$\widetilde{Y}(f) = \int_{-\infty}^{\infty} \widetilde{H}(f_1, f - f_1) \widetilde{X}(f_1)\, df_1 \tag{13.10}$$

and

$$\widetilde{y}(t) = \int_{-\infty}^{\infty} \widetilde{Y}(f) \exp(j2\pi f t)\, df \tag{13.11}$$

In the two-dimensional "transfer function" defined in (13.8), the frequency variable $f_1$is associated with the time variable, and it may be viewed as analogous to the frequency variable f in the transfer function $H(f)$ of linear time-invariant systems. However, the input-output relationship for LTV systems in the frequency domain, given in (13.10), involves a convolution in the frequency domain in the second variable of the transfer function $\widetilde{H}(f_1, f_2)$. This convolution accounts for the effect of the time-varying nature of the system in the frequency domain.

If the input to an LTIV system is a tone at $f_c + f_0$

$$x(t) = A \cos[2\pi(f_c + f_0)t] \tag{13.12}$$

for which the complex envelope is

$$\widetilde{x}(t) = A\exp(j2\pi f_0 t) \tag{13.13}$$

the input-output relationship in the frequency domain is given by

$$\widetilde{X}(f) = A\delta(f - f_0) \tag{13.14}$$

and

$$\widetilde{Y}(f) = \widetilde{H}(f)\widetilde{X}(f) = \widetilde{H}(f_0)A\delta(f - f_0) \tag{13.15}$$

The system complex envelope output, in the time domain, is defined as

$$\widetilde{y}(t) = A\left|\widetilde{H}(f_0)\right|\exp[j2\pi f_0 t + \angle\widetilde{H}(f_0)] \tag{13.16}$$

and the time-domain bandpass signal is given by

$$y(t) = A\left|\widetilde{H}(f_0)\right|\cos[2\pi(f_c + f_0)t + \angle\widetilde{H}(f_0)] \tag{13.17}$$

The relationship given in (13.13) and (13.17) for LTIV systems show that when the input to the system is a complex tone at frequency f_0, the system produces an output tone at the same frequency. The amplitude and phase of the output tone are affected by the amplitude and phase response of the system at frequency f_0. This is illustrated in Figure 13.3.

We now consider the same situation for a time-varying system. With the same input, (13.13), the output of an LTV system in the frequency domain is

$$\begin{aligned}\widetilde{Y}(f) &= \int_{-\infty}^{\infty}\widetilde{H}(f_1, f - f_1)\widetilde{X}(f_1)\,df_1 \\ &= \int_{-\infty}^{\infty}\widetilde{H}(f_1, f - f_1)A\delta(f_1 - f_0)\,df_1 \\ &= \widetilde{H}(f_0, f - f_0)A\end{aligned} \tag{13.18}$$

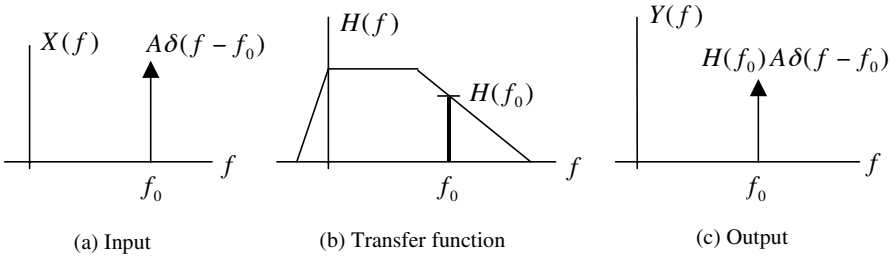


Figure 13.3 Response of an LTIV system to a tone input.

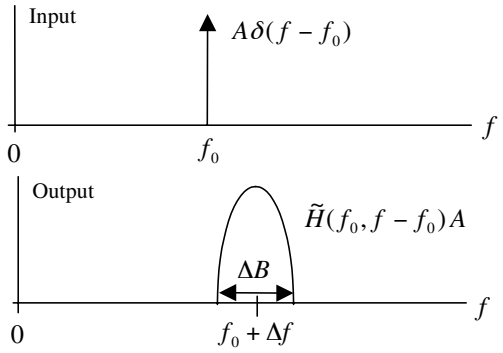


Figure 13.4 Response of an LTV system to a tone input (note the spectral "shift" Δf and "spreading" ΔB).

Source: M. C. Jeruchim, P. Balaban, and K. S. Shanmugan, *Simulation of Communications Systems*, 2nd ed., New York: Kluwer Academic/Plenum Publishers, 2000.

The preceding equation shows that the output does not consist of a single tone but possibly a continuum of tones at all frequencies as dictated by the behavior of the two-dimensional transfer function in f_2. An example is shown in Figure 13.4. [Note that if the system is time invariant, then $\widetilde{h}(\tau, t) = \widetilde{h}(\tau)$, $\widetilde{H}(f_1, f_2) = \widetilde{H}(f_1)\delta(f_2)$, and (13.18) produces $\widetilde{Y}(f) = \widetilde{H}(f_0)A\delta(f - f_0)$ which is the same as (13.15).]

In general, the output of an LTV system responding to an input tone at frequency f_0 might be shifted in frequency and also might be spread. In the context of mobile communication channels, this is referred to as the doppler shift and doppler spreading respectively, and is produced by the relative motion (velocity and acceleration) between the transmitting and receiving antennas or by other changes in the channel.

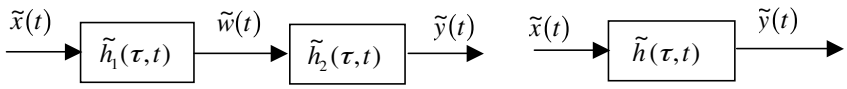
13.2.3 Properties of LTV Systems

Several properties of LTIV systems are useful in simplifying the simulation model of communication systems. Examples include the combining of the transfer functions of blocks in parallel, blocks in series, and combinations of blocks in series and parallel. These operations are based on the associative, distributive, and commutative properties of LTIV systems. These properties are now examined for LTV systems.

Associative Property

To examine the associative property in detail, consider the system shown in Figure 13.5(a). The output of the first block in Figure 13.5(a) is given by

$$\widetilde{w}(t) = \int_{-\infty}^{\infty} \widetilde{h}_1(\tau_1, t)\widetilde{x}(t - \tau_1)\, d\tau_1 = \int_{-\infty}^{\infty} \widetilde{h}_1(t - \tau_1, t)\widetilde{x}(\tau_1)\, d\tau_1 \tag{13.19}$$



(a) Series connection (b) Equivalent system

Figure 13.5 Series connection and equivalent system.

The output of the second block is

$$\tilde{y}(t) = \int_{-\infty}^{\infty} \tilde{h}_2(\tau_2, t)\tilde{w}(t-\tau_2)\, d\tau_2 \tag{13.20}$$

Substituting for $\tilde{w}(t)$ gives

$$\tilde{y}(t) = \int_{-\infty}^{\infty} \tilde{h}_2(\tau_2, t) \left[\int_{-\infty}^{\infty} \tilde{h}_1(t-\tau_2-\tau_1, t-\tau_2)\tilde{x}(\tau_1)\, d\tau_1\right] d\tau_2 \tag{13.21}$$

which can be written

$$\tilde{y}(t) = \int_{-\infty}^{\infty} \tilde{x}(\tau_1) \left[\int_{-\infty}^{\infty} \tilde{h}_1(t-\tau_2-\tau_1, t-\tau_2)\tilde{h}_2(\tau_2, t)\, d\tau_2\right] d\tau_1 \tag{13.22}$$

The quantity inside the brackets is the overall impulse response of the system $\tilde{h}(t-\tau_1, t)$. Substituting $\tau = t-\tau_1$, we obtain the overall impulse response of two cascaded blocks as

$$\tilde{h}(\tau, t) = \int_{-\infty}^{\infty} \tilde{h}_1(\tau-\tau_2, t-\tau_2)\tilde{h}_2(\tau_2, t)\, d\tau_2 \tag{13.23}$$

In simpler appearing form we have

$$\tilde{h}(\tau, t) = \int_{-\infty}^{\infty} \tilde{h}_1(\tau-\alpha, t-\alpha)\tilde{h}_2(\alpha, t)\, d\alpha \tag{13.24}$$

Using the preceding equation, we can define the associative property for an LTV system as

$$\begin{aligned}\tilde{x}(\tau) \circledast \tilde{h}_1(\tau) \circledast \tilde{h}_2(\tau) &= [\tilde{x}(\tau) \circledast \tilde{h}_1(\tau)] \circledast \tilde{h}_2(\tau) \\ &= \tilde{x}(\tau) \circledast [\tilde{h}_1(\tau) \circledast \tilde{h}_2(\tau)]\end{aligned} \tag{13.25}$$

The overall impulse response is

$$\tilde{h}(\tau, t) = \tilde{h}_1(\tau, t) \circledast \tilde{h}_2(\tau, t) \tag{13.26}$$

where

$$\tilde{h}(\tau, t) = \int_{-\infty}^{\infty} \tilde{h}_1(\tau-\alpha, t-\alpha)\tilde{h}_2(\alpha, t) d\alpha \tag{13.27}$$

as shown in Figure 13.5(b).

Commutative Property

While it is possible to combine the response of two LTV blocks in series according to (13.27), one cannot interchange the order of LTV blocks. Interchanging the two blocks yields the overall impulse response as the convolution of the individual impulse responses. Thus

$$\begin{aligned}\widetilde{g}(\tau,t) &= \widetilde{h}_2(\tau,t) \circledast \widetilde{h}_1(\tau,t) \\ &= \int_{-\infty}^{\infty} \widetilde{h}_2(\tau-\alpha,t-\alpha)\widetilde{h}_1(\alpha,t)d\alpha \end{aligned} \tag{13.28}$$

In general, $\widetilde{h}(\tau,t) \neq \widetilde{g}(\tau,t)$. Hence, interchanging the order of LTV is not, in general, a valid operation. The example given in Figure 13.6 provides a specific example for which the commutative property does not hold for LTV blocks, that is,

$$\widetilde{h}_1(\tau,t) \circledast \widetilde{h}_2(\tau,t) \neq \widetilde{h}_2(\tau,t) \circledast \widetilde{h}_1(\tau,t) \tag{13.29}$$

for LTV systems. However, it should be noted that the commutative property does hold for LTIV blocks, that is,

$$\widetilde{h}_1(\tau) \circledast \widetilde{h}_2(\tau) = \widetilde{h}_2(\tau) \circledast \widetilde{h}_1(\tau) \tag{13.30}$$

for LTIV systems.

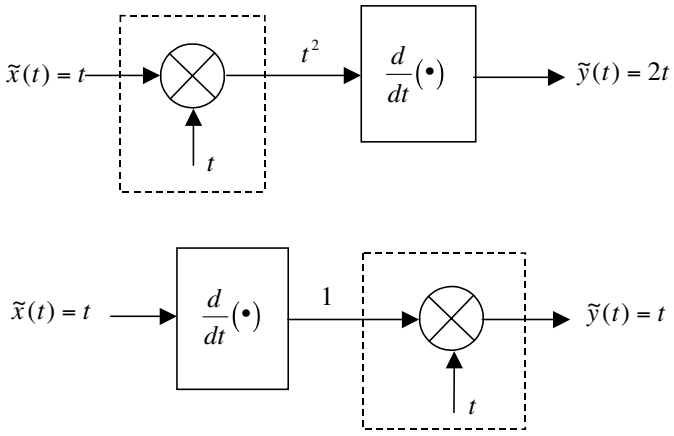


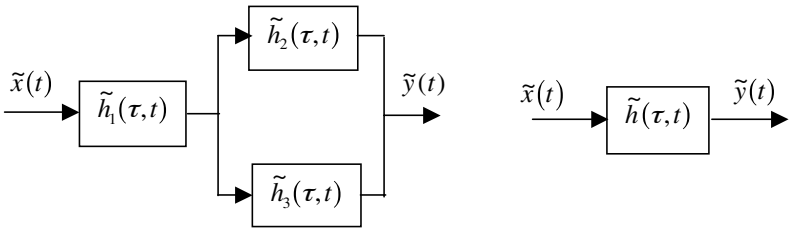
Figure 13.6 Example on interchanging LTV blocks.

Source: M. C. Jeruchim, P. Balaban, and K. S. Shanmugan, *Simulation of Communications Systems*, 2nd ed., New York: Kluwer Academic/Plenum Publishers, 2000.

Distributive Property

The reader can verify that the distributive property

$$\widetilde{x}(\tau) \circledast [\widetilde{h}_1(\tau,t) + \widetilde{h}_2(\tau,t)] = \widetilde{x}(\tau) \circledast \widetilde{h}_1(\tau,t) + \widetilde{x}(\tau) \circledast \widetilde{h}_2(\tau,t) \tag{13.31}$$



(a) Series and parallel connections (b) Equivalent system

Figure 13.7 Series/parallel connection and equivalent system.

or, equivalently

$$\widetilde{h}_1(\tau,t) \circledast [\widetilde{h}_2(\tau,t) + \widetilde{h}_3(\tau,t)] = \widetilde{h}_1(\tau,t) \circledast \widetilde{h}_2(\tau,t) + \widetilde{h}_1(\tau,t) \circledast \widetilde{h}_3(\tau,t) \tag{13.32}$$

holds for LTV systems. An example of the simplification of simulation block diagrams of LTV systems, which result from the associative and distributive properties, is shown in Figure 13.7, where

$$\widetilde{h}(\tau,t) = \widetilde{h}_1(\tau,t) \circledast \widetilde{h}_2(\tau,t) + \widetilde{h}_1(\tau,t) \circledast \widetilde{h}_3(\tau,t) \tag{13.33}$$

in Figure 13.7(b).

Unfortunately, the one simplification that results in significant computational efficiency in LTIV blocks does not apply directly for LTV blocks. Block diagrams involving feedback cannot be simplified.

Example 13.1. Assume that the input to a linear time-varying system is defined by a unit amplitude complex exponential having a frequency of 1 kHz. In other words

$$\widetilde{x}(t) = \exp(j2000\pi t) \tag{13.34}$$

which, in the frequency domain, is

$$\widetilde{X}(f) = \delta(f - 1000) \tag{13.35}$$

The system, defined by $\widetilde{h}(\tau,t)$, is assumed to have a time-varying impulse response

$$\widetilde{h}(\tau,t) = a_1(t)\,\delta(\tau) + a_2(t)\,\delta(\tau - T) + a_3(t)\,\delta(\tau - 2T) \tag{13.36}$$

where T is 1 ms and the time-varying attenuations, $a_i(t)$, are given by

$$a_1(t) = 1\cos(100\pi t) \tag{13.37}$$

$$a_2(t) = 0.7\cos(200\pi t) \tag{13.38}$$

and

$$a_3(t) = 0.3\cos(300\pi t) \tag{13.39}$$

Figure 13.8 illustrates ten "snapshots" of the impulse response $\widetilde{h}(\tau, t)$ taken every 2 ms starting at $t = 0$ s. We can see that the system is a function of both the elapsed time, τ, and the observation time, t. Diagrams of this type are frequently used to describe the time-varying channel typical of wireless communications systems.

The time-varying frequency response of the channel is found by taking the two-dimensional Fourier transform defined by (13.8). Substituting the impulse response defined by (13.36) into (13.8) gives

$$\begin{aligned}\widetilde{H}(f_1, f_2) = &\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} 1\cos(100\pi t)\,\delta(\tau) \\ &\cdot \exp(-j2\pi f_1\tau)\exp(-j2\pi f_2 t)\,d\tau dt \\ &+ \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} 0.7\cos(200\pi t)\,\delta(\tau - T) \\ &\cdot \exp(-j2\pi f_1\tau)\exp(-j2\pi f_2 t)\,d\tau dt \\ &+ \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} 0.3\cos(300\pi t)\,\delta(\tau - 2T) \\ &\cdot \exp(-j2\pi f_1\tau)\exp(-j2\pi f_2 t)\,d\tau dt\end{aligned} \tag{13.40}$$

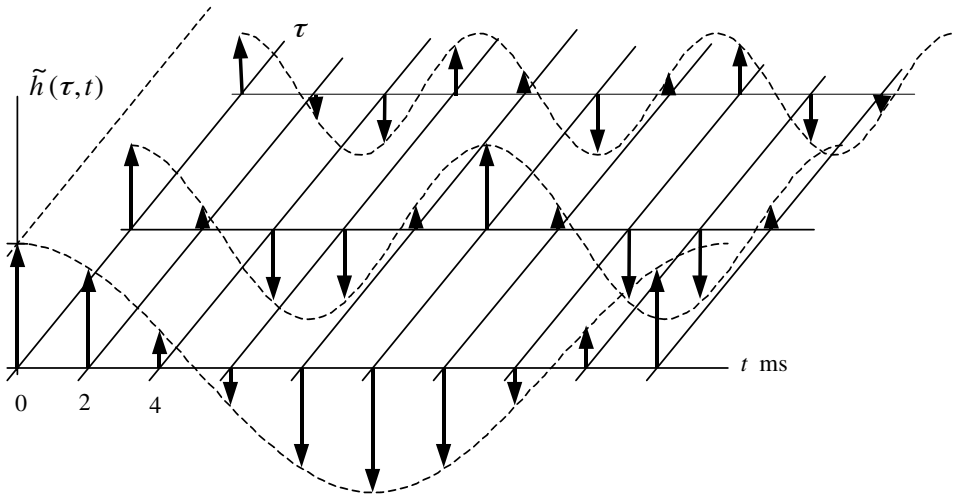


Figure 13.8 $\widetilde{h}(\tau, t)$ as a function of both τ and t.

and then applying the sifting and modulation Fourier transform theorems, we find the transfer function

$$\widetilde{H}(f_1, f_2) = \frac{1}{2}\delta(f_2 \pm 50) + \frac{0.7}{2}\delta(f_2 \pm 100)\exp(-j2\pi f_1 T) + \frac{0.3}{2}\delta(f_2 \pm 150)\exp(-j4\pi f_1 T) \tag{13.41}$$

whose magnitude response is plotted in Figure 13.9. Note that, in contrast to an LTIV system, the transfer function is dependent on both f_1 and f_2.

The system output is defined by (13.10). Substitution of (13.35) and (13.41) into (13.10) gives

$$\begin{aligned}\widetilde{Y}(f) = &\frac{1}{2}\int_{-\infty}^{\infty}\delta[(f-f_1)\pm 50]\,\delta(f_1-1000)\,df_1 \\ &+\frac{0.7}{2}\int_{-\infty}^{\infty}\delta[(f-f_1)\pm 100]\,\delta(f_1-1000)\exp(-j2\pi f_1 T)\,df_1 \\ &+\frac{0.3}{2}\int_{-\infty}^{\infty}\delta[(f-f_1)\pm 150]\,\delta(f_1-1000)\exp(-j4\pi f_1 T)\,df_1\end{aligned} \tag{13.42}$$

which, using the shifting property of the delta function, integrates to

$$\begin{aligned}\widetilde{Y}(f) = &\frac{1}{2}\delta(f-1000\pm 50) \\ &+\frac{0.7}{2}\delta(f-1000\pm 100)\exp(-j2000\pi T) \\ &+\frac{0.3}{2}\delta(f-1000\pm 150)\exp(-j4000\pi T)\end{aligned} \tag{13.43}$$

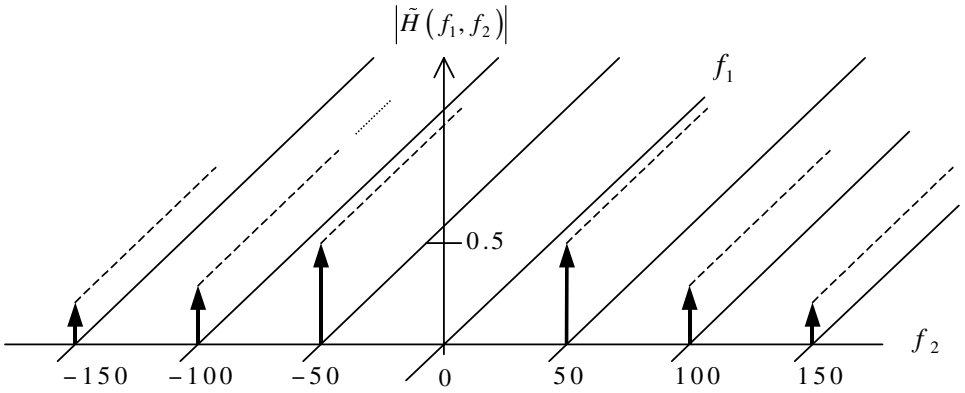


Figure 13.9 Fourier transform of time-varying impulse response.

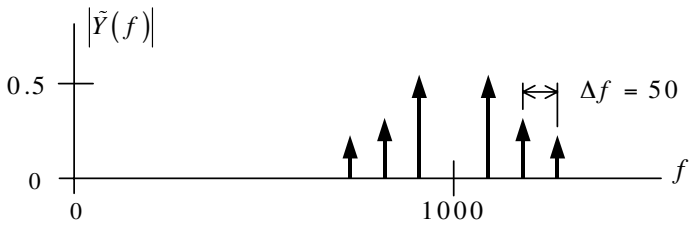


Figure 13.10 System output.

The response of the time-varying system to the assumed input is illustrated in Figure 13.10. We see that the input, which is a single tone at 1,000 Hz, has been shifted, as well as spread in frequency, due to the time-varying channel impulse response. The spreading effect will be explained in the following section. ■

13.3 Random Process Models

In many time-varying systems, the characteristics of the system change as a function of time in a random manner. Examples include changes in hardware characteristics due to aging of the components and changes in the characteristics of a wireless channel due to changes in the atmospheric conditions. These variations in time are usually modeled as random processes. If the underlying model is linear, the random variations of the system characteristics as a function of time can be handled by treating the impulse response $\widetilde{h}(\tau, t)$ as a random process in t. Such an approach is extensively used to model mobile wireless communication channels. While the details of different types of channel models will be considered in the next chapter, we provide a brief introduction here, in order to develop a generic simulation model for randomly time-varying systems. (In this section, we will use lower-case functions of time to denote random processes even though it is customary to use upper-case letters to denote random variables and random processes. Upper-case letters will be used to denote Fourier transforms.)

As a simple example, consider a system in which the output is an attenuated and delayed version of the input, where the attenuation is randomly changing as a function of time. The system is characterized by an impulse response

$$\widetilde{h}(\tau, t) = a(t)\delta(\tau - t_0) \tag{13.44}$$

where $a(t)$ is the time-varying attenuation and t_0 is the delay. The input-output relationship is given by

$$\widetilde{y}(t) = \int_{-\infty}^{\infty} \widetilde{h}(\tau, t)\widetilde{x}(t - \tau)\, d\tau = a(t)\widetilde{x}(t - t_0) \tag{13.45}$$

The attenuation $a(t)$ can be modeled as a stationary or nonstationary random process. The preferred model is a stationary random process, since this leads to the

autocorrelation function and power spectral density. Modeling $a(t)$ as a stationary process still permits a time-varying impulse response model.

If the input $\widetilde{x}(t)$ to the system is a stationary random process, the autocorrelation and the power spectral density of the output can be obtained as

$$R_{\widetilde{y}\widetilde{y}}(\alpha) = E\left\{\widetilde{y}^*(t)\widetilde{y}(t+\alpha)\right\} \tag{13.46}$$

which is

$$R_{\widetilde{y}\widetilde{y}}(\alpha) = E\left\{\tilde{a}^*(t)\widetilde{x}^*(t-t_0)\tilde{a}(t+\alpha)\widetilde{x}(t-t_0+\alpha)\right\} \tag{13.47}$$

Assuming that the input process $\widetilde{x}(t)$ and the attenuation $\widetilde{a}(t)$ are uncorrelated, which is a reasonable assumption, the output autocorrelation reduces to the simple form

$$R_{\widetilde{y}\widetilde{y}}(\alpha) = R_{\widetilde{a}\widetilde{a}}(\alpha)R_{\widetilde{x}\widetilde{x}}(\alpha) \tag{13.48}$$

By taking the Fourier transform of the preceding equation, we can obtain the power spectral density (PSD) of the output. The result is

$$S_{\widetilde{y}\widetilde{y}}(f) = S_{\widetilde{a}\widetilde{a}}(f) \circledast S_{\widetilde{x}\widetilde{x}}(f) \tag{13.49}$$

The convolution in the preceding equation could lead to a spectral shifting and spreading. If the input is a randomly phased complex exponential:

$$\widetilde{x}(t) = A\exp(j2\pi f_0 t + j\theta), \quad \theta \text{ Uniform in } [-\pi, \pi]$$

then

$$R_{\widetilde{x}\widetilde{x}}(\alpha) = A^2\exp(j2\pi f_0\alpha) \tag{13.50}$$

from which the input PSD is

$$S_{\widetilde{x}\widetilde{x}}(f) = A^2\delta(f - f_0) \tag{13.51}$$

The output PSD is

$$S_{\widetilde{y}\widetilde{y}}(f) = A^2 S_{\widetilde{h}\widetilde{h}}(f - f_0) \tag{13.52}$$

as illustrated in Figure 13.11. An LTIV system will produce an output power spectral density of the form

$$S_{\widetilde{y}\widetilde{y}}(f) = A^2\delta(f - f_0)\left|H(f_0)\right|^2 \tag{13.53}$$

The essential difference between the LTV system and the LTIV system is the spectral spreading caused by the LTV system.

One manifestation of spectral spreading in simulation is that the sampling rate has to be increased appropriately at least by an amount equal to twice the bandwidth expansion due to the spreading. A rule of thumb is to arrive at a sampling

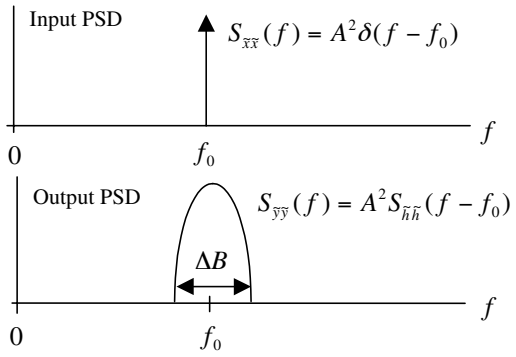


Figure 13.11 Spectral spreading in a time-varying system.

rate treating the system as time invariant and increasing the sampling rate to accommodate the "excess" bandwidth, which is equal to the bandwidth of the random process modeling the time-varying nature of the system.

A more general version of the time-varying model treats $\widetilde{h}(\tau, t)$ as a stationary random process in t with an autocorrelation function

$$R_{\widetilde{h}\widetilde{h}}(\tau_1, \tau_2, \alpha) = E\left\{\widetilde{h}^*(\tau_1, t)\widetilde{h}(\tau_2, t + \alpha)\right\} \tag{13.54}$$

The most commonly used model for $\widetilde{h}(\tau, t)$ is a zero mean stationary Gaussian process that leads to a Rayleigh probability density function for $\left|\widetilde{h}(\tau, t)\right|$. In this model, it is usually assumed that $\widetilde{h}(\tau_1, t)$ and $\widetilde{h}(\tau_2, t)$ are uncorrelated for $\tau_1 \neq \tau_2$. In other words:

$$\begin{aligned} R_{\widetilde{h}\widetilde{h}}(\tau_1, \tau_2, \alpha) &= E\left\{\widetilde{h}^*(\tau_1, t)\widetilde{h}(\tau_2, t + \alpha)\right\} \\ &= R_{\widetilde{h}\widetilde{h}}(\tau_1, \alpha)\delta(\tau_1 - \tau_2) \end{aligned} \tag{13.55}$$

For this case, the autocorrelation of the output of the system can be obtained from

$$R_{\widetilde{y}\widetilde{y}}(\alpha) = E\left\{\widetilde{y}^*(t)\widetilde{y}(t + \alpha)\right\} \tag{13.56}$$

Substituting for $\widetilde{y}(t)$ yields

$$\begin{aligned} &R_{\widetilde{y}\widetilde{y}}(\alpha) = \\ &\quad E\left\{\int_{-\infty}^{\infty} \widetilde{h}^*(\tau_1, t)\widetilde{x}^*(t - \tau_1)\, d\tau_1 \int_{-\infty}^{\infty} \widetilde{h}(\tau_2, t + \alpha)\widetilde{x}(t + \alpha - \tau_2)\, d\tau_2\right\} \end{aligned} \tag{13.57}$$

Interchanging orders of expectation and integration results in

$$R_{\widetilde{y}\widetilde{y}}(\alpha) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} E\left\{\widetilde{h}^*(\tau_1,t)\widetilde{h}(\tau_2,t+\alpha)\widetilde{x}^*(t-\tau_1)\widetilde{x}(t+\alpha-\tau_2)\right\} d\tau_1\, d\tau_2 \quad (13.58)$$

Assuming $\widetilde{x}(t)$ and $\widetilde{h}(t)$ independent, which is certainly reasonable, yields

$$R_{\widetilde{y}\widetilde{y}}(\alpha) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} E\left\{\widetilde{h}^*(\tau_1,t)\widetilde{h}(\tau_2,t+\alpha)\right\} \cdot E\left\{\widetilde{x}^*(t-\tau_1)\widetilde{x}(t+\alpha-\tau_2)\right\} d\tau_1\, d\tau_2 \quad (13.59)$$

Recognizing that the two expectations are autocorrelation functions and invoking (13.55) provides the simplification

$$R_{\widetilde{y}\widetilde{y}}(\alpha) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} R_{\widetilde{h}\widetilde{h}}(\tau_1,\alpha)\delta(\tau_1-\tau_2)R_{\widetilde{x}\widetilde{x}}(\alpha+\tau_1-\tau_2)\, d\tau_1\, d\tau_2 \quad (13.60)$$

Performing the integration on τ_2 using the sifting property gives

$$R_{\widetilde{y}\widetilde{y}}(\alpha) = R_{\widetilde{x}\widetilde{x}}(\alpha)\int_{-\infty}^{\infty} R_{\widetilde{h}\widetilde{h}}(\tau_1,\alpha)\, d\tau_1 \quad (13.61)$$

which can be expressed by

$$R_{\widetilde{y}\widetilde{y}}(\alpha) = R_{\widetilde{x}\widetilde{x}}(\alpha)\bar{R}_{\widetilde{h}\widetilde{h}}(\alpha) \quad (13.62)$$

where

$$\bar{R}_{\widetilde{h}\widetilde{h}}(\alpha) = \int_{-\infty}^{\infty} R_{\widetilde{h}\widetilde{h}}(\tau_1,\alpha)\, d\tau_1 \quad (13.63)$$

The power spectral density of the output can be obtained by taking the Fourier transform of (13.62), which leads to the convolution

$$S_{\widetilde{y}\widetilde{y}}(f) = S_{\widetilde{x}\widetilde{x}}(f) \circledast \bar{S}_{\widetilde{h}\widetilde{h}}(f) \quad (13.64)$$

where $\circledast$, as always, denotes convolution. Note that $\bar{S}_{\widetilde{h}\widetilde{h}}(f)$ is "averaged" power spectral density and is the Fourier transform of the "averaged" autocorrelation function defined in (13.63).

Note that the output power spectral density is a convolution of the input power spectral density and the "averaged" power spectral density of the random process that models the time variations. In contrast, the power spectral density relationship for an LTIV system is given by

$$S_{\widetilde{y}\widetilde{y}}(f) = S_{\widetilde{x}\widetilde{x}}(f)\,|H(f)|^2 \quad (13.65)$$

Once again, when the input is a tone, an LTIV system produces an output tone at the same frequency, whereas the output of an LTV system could be shifted and spread in frequency.

13.4 Simulation Models for LTV Systems

Given a description of an LTV system in the form of its impulse response $\widetilde{h}(\tau, t)$, a simulation model can be derived using the sampling theorem assuming that the input to the channel is bandlimited [1]. We start from the input-output relationship given by the convolution integral

$$\widetilde{y}(t) = \int_{-\infty}^{\infty} \widetilde{h}(\tau, t)\widetilde{x}(t-\tau)\, d\tau$$

and use the sampling theorem to represent the input in terms of its sampled values.

From the sampling theorem we know that a lowpass signal $\widetilde{w}(\tau)$ bandlimited to B Hz can be represented in terms of its sampled values as

$$\widetilde{w}(\tau) = \sum_{n=-\infty}^{\infty} \widetilde{w}(nT)\frac{\sin(2\pi B(\tau - nT))}{2\pi B(\tau - nT)} \tag{13.66}$$

where $1/T$ is the sampling rate, which is set equal to the Nyquist rate $2B$. The minimum sampling rate of $2B$ is chosen to minimize the computational burden of the simulation model. Using the representation given above, with $\widetilde{x}(t-\tau) = \widetilde{w}(\tau)$, we can replace $\widetilde{x}(t-\tau)$ in the convolution integral by

$$\widetilde{x}(t-\tau) = \sum_{n=-\infty}^{\infty} \widetilde{x}(t-nT)\frac{\sin(2\pi B(\tau - nT))}{2\pi B(\tau - nT)} \tag{13.67}$$

which leads to

$$\begin{aligned}\widetilde{y}(\tau) &= \int_{-\infty}^{\infty} \widetilde{h}(\tau, t)\left\{\sum_{n=-\infty}^{\infty} \widetilde{x}(t-nT)\frac{\sin(2\pi B(\tau - nT))}{2\pi B(\tau - nT)}\right\} d\tau \\ &= \sum_{n=-\infty}^{\infty} \widetilde{x}(t-nT)\int_{-\infty}^{\infty} \widetilde{h}(\tau, t)\left\{\frac{\sin(2\pi B(\tau - nT))}{2\pi B(\tau - nT)}\right\} d\tau \end{aligned} \tag{13.68}$$

We can therefore write

$$\widetilde{y}(t) = \sum_{n=-\infty}^{\infty} \widetilde{x}(t-nT)\widetilde{g}_n(t) \tag{13.69}$$

where

$$\widetilde{g}_n(t) = \int_{-\infty}^{\infty} \widetilde{h}(\tau, t)\left\{\frac{\sin(2\pi B(\tau - nT))}{2\pi B(\tau - nT)}\right\} d\tau \tag{13.70}$$

Equations (13.69) and (13.70) define a simulation model for time-varying systems.

13.4.1 Tapped Delay Line Model

The model given in (13.69) can be implemented in the form of a tapped delay line (TDL) as shown in Figure 13.12, with the tap gain functions specified by (13.70). In general, the tap gain functions will themselves be random processes and they will be correlated, that is, $\widetilde{g}_n(t)$ and $\widetilde{g}_m(t\ )$ will be correlated.

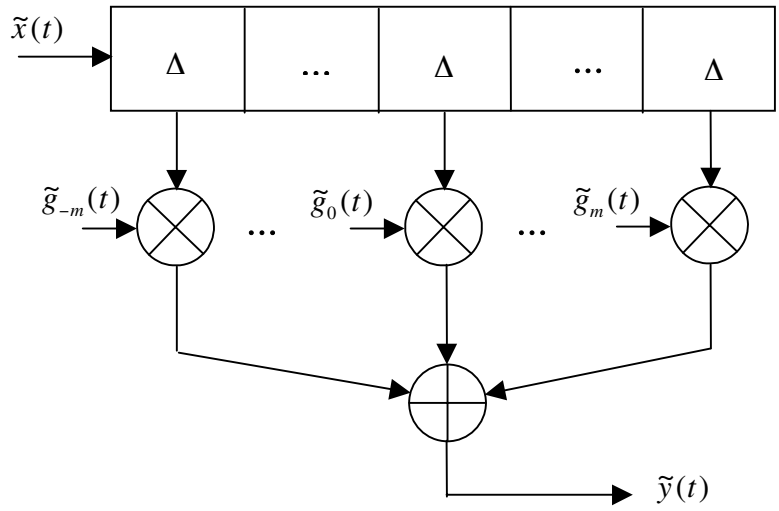


Figure 13.12 Tapped delay-line model.

Source: M. C. Jeruchim, P. Balaban, and K. S. Shanmugan, *Simulation of Communications Systems*, 2nd ed., New York: Kluwer Academic/Plenum Publishers, 2000.

Simplification of the TDL Model

The model shown above can be simplified in many ways using several approximations and assumptions. First, it is often assumed that the tap gain processes are uncorrelated. They are also approximated by

$$\widetilde{g}_n(t) \approx T\widetilde{h}(nT, t) \tag{13.71}$$

In this approximation, the tap gain functions represent the sampled values of the time-varying impulse response $\widetilde{h}(\tau, t)$, where the sampling is done in the impulse response variable.

The second approximation involves truncation of the impulse response. If

$$E\left\{\left|\widetilde{h}(\tau, t)\right|^2\right\} \to 0, \quad |\tau| \geq mT$$

then the summation in (13.69) can be truncated to $2m + 1$ terms as

$$\sum_{n=-\infty}^{\infty} \widetilde{x}(t - nT)\widetilde{g}_n(t) \approx \sum_{n=-m}^{m} \widetilde{x}(t - nT)\widetilde{g}_n(t) \tag{13.72}$$

and the tapped delay line model has only a finite number of taps as shown in Figure 13.12. The total number of taps should be kept to a minimum in order to maximize the computational efficiency of the model.

Finally, if the system is time invariant, then $\widetilde{h}(\tau, t) = \widetilde{h}(\tau)$, and the tap gains become constants

$$\widetilde{g}_n(t) = \widetilde{g}_n \approx T\widetilde{h}(nT)$$

In other words, the gains are simply the sampled values of the impulse response of the LTIV system, and the tapped delay line model reduces to an impulse invariant model, or an FIR filter performing time domain convolution.

One other aspect of the TDL also deserves additional attention. The TDL model shown in Figure 13.12 has continuous time input $\widetilde{x}(t)$ and continuous time output $\widetilde{y}(t)$. However, in simulation we will use sampled values of $\widetilde{x}(t)$ and $\widetilde{y}(t)$. Sampling should normally be carried out using a sampling frequency 8 to 16 times the bandwidth, where the bandwidth includes the effect of spreading due to the time-varying nature of the system. This effect can be seen in (13.52) and (13.64). Note that the Nyquist rate of $2B$ was used to derive the TDL model, and the tap spacing is $T = 1/2B$, which will be $>> T_s$, where T_s is the sampling time for the input and output signals. It is of course possible to derive a TDL model with a smaller tap spacing of T_s, but such a model will be computationally inefficient and does not necessarily improve the accuracy.

Generation of Tap Gain Processes

The tap gain processes are stationary random processes with a given probability density function and power spectral density. The simplest model for the tap gain processes assumes them to be uncorrelated, complex, zero mean Gaussian processes with different variances but identical power spectral densities. In this case, the tap gain processes can be generated by filtering white Gaussian processes, as shown in Figure 13.13.

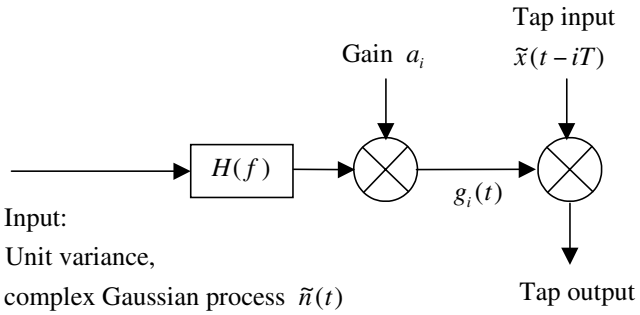


Figure 13.13 Generation of the i^{th} tap gain process, $t = kT_s$.

Source: M. C. Jeruchim, P. Balaban, and K. S. Shanmugan, *Simulation of Communications Systems*, 2nd ed., New York: Kluwer Academic/Plenum Publishers, 2000.

The filter transfer function is chosen such that it produces the desired power spectral density, that is, $H(f)$ is chosen such that

$$S_{\widetilde{g}\widetilde{g}}(f) = S_{\widetilde{n}\widetilde{n}}(f)\left|\widetilde{H}(f)\right|^2 = \left|\widetilde{H}(f)\right|^2 \tag{13.73}$$

where $S_{\widetilde{n}\widetilde{n}}(f)$ is the power spectral density of the input noise process, which can be set equal to 1, and $S_{\widetilde{g}\widetilde{g}}(f)$ is the specified power spectral density of the tap gain processes. The static gain, a_i, in Figure 13.13 accounts for the different power levels or variances for the different taps. If the power spectral density of the tap gains are different, then different filters will be used for the different taps.

There are several ways of implementing the spectral shaping filter (also referred to as the doppler filter in channel models). An FIR filter in the time domain is the most common implementation, since the (doppler) power spectral densities do not lend themselves to spectral factoring and implementation in recursive form.

In generating the tap gain processes, it should be noted that the bandwidth of the tap gain processes of slowly varying systems will be small compared to the bandwidth of the system and the signals that flow through it. In this case, the tap gain filter should be designed and executed at a slower sampling rate. Interpolation can be used at the output of the filter to produce denser samples at a rate consistent with the sampling rate of the signal coming into the tap. Designing the filter at the higher sampling rate will lead to computational inefficiencies as well as stability problems.

13.5 MATLAB Examples

We conclude this chapter with two MATLAB examples illustrating the concepts presented in this chapter.

13.5.1 MATLAB Example 1

This example illustrates the spectral spreading that takes place in a time-varying system as shown in (13.64). The system simulated is a simple one in which the bandpass input to the system is a "tone" of the form $x(t) = \cos[2\pi(f_0 + f_1)t]$, which corresponds to a lowpass equivalent signal

$$\widetilde{x}(t) = \exp(j2\pi f_1 t) \tag{13.74}$$

The lowpass equivalent impulse response of the system is assumed to be of the form

$$\widetilde{h}(\tau, t) = \widetilde{a}(t)\delta(t - \tau_0) \tag{13.75}$$

which is an allpass channel with a delay of τ_0 and a complex, time-varying attenuation of $\widetilde{a}(t)$. The attenuation is modeled as a zero-mean Gaussian random process with a power spectral density

$$S_{\widetilde{a}\widetilde{a}}(f) = \frac{1}{(j2\pi f)^2 + B^2} \tag{13.76}$$

where B is the doppler bandwidth. The complex envelope of the output of the time-varying system can be shown to be

$$\widetilde{y}(t) = \widetilde{a}(t)\widetilde{x}(t - \tau_0) \tag{13.77}$$

Simulation of this model involves generating sampled values of the input tone and multiplying with a filtered complex Gaussian process, where the filter used is chosen to have a transfer function that yields the power spectral density given by (13.76). It is left as an exercise for the reader (Problem 13.6) to show that the required filter has the transfer function

$$H(s) = \frac{1}{s + B} \tag{13.78}$$

The simulation results illustrated are for parameter values $f_1 = 512$ Hz, $B = 64$ Hz, $\tau_0 = 0$, $f_s = 8{,}192$ samples/sec and simulation length = 8,192 samples.

Results of the simulation are shown in Figures 13.14 and 13.15. The power spectral density of the filtered Gaussian process and the time-domain values of the correlated Gaussian sequence generated by the filter are shown in the top and bottom frames of Figure 13.14, respectively. The input tone expressed by (13.74) and

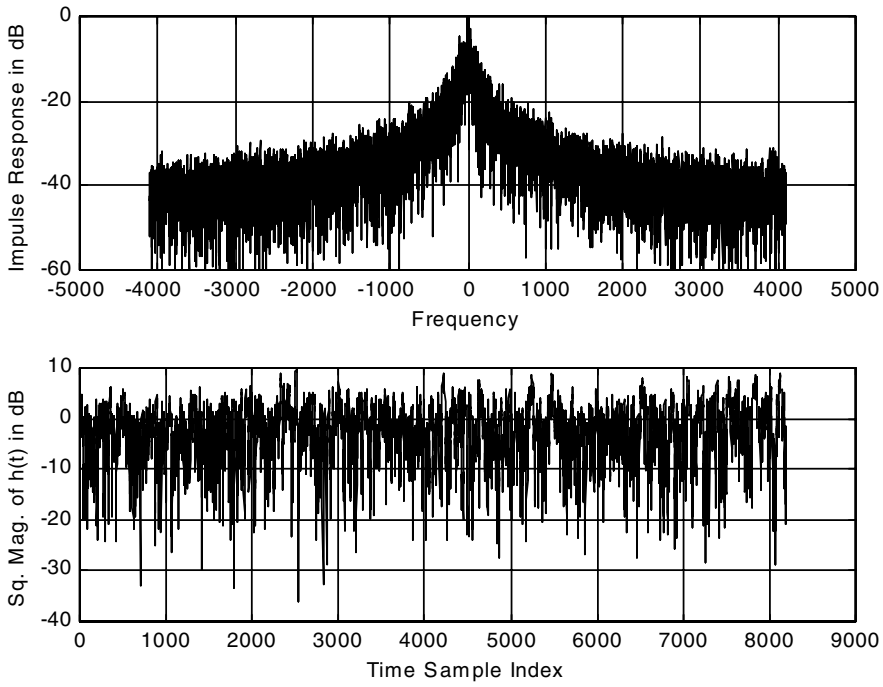


Figure 13.14 Frequency domain (top) and time domain (bottom) representation of the filtered Gaussian process.

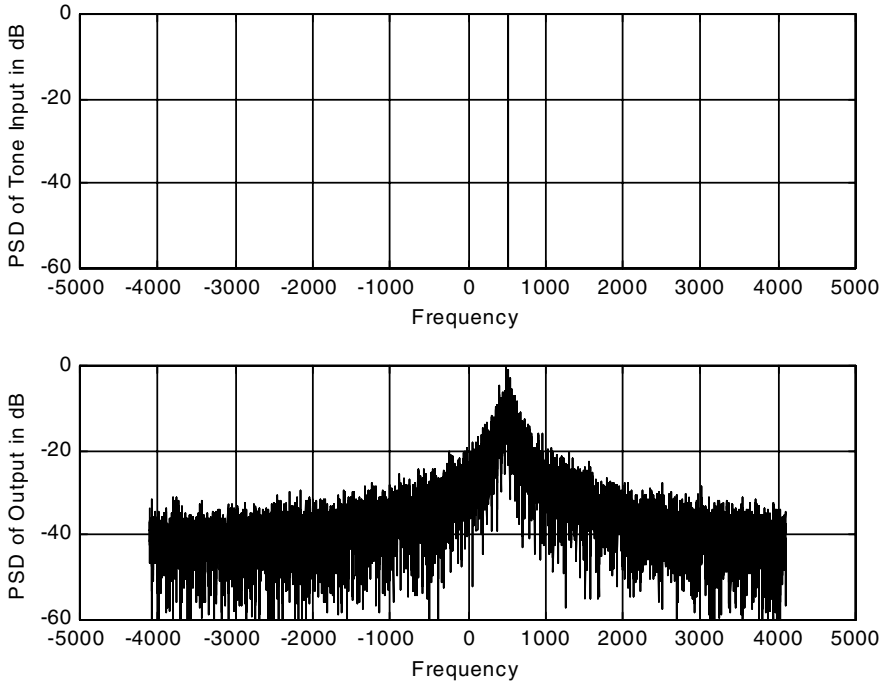


Figure 13.15 PSD of input tone and spread system output.

output of the time-varying system, both in the frequency domain, are shown in the top and bottom frames of Figure 13.15, respectively. The spectral spreading due to the time-varying nature of the system can be seen in the bottom frame of Figure 13.15. The MATLAB code used to develop Figures 13.14 and 13.15 is given in Appendix A. (Note: The MATLAB code given in Appendix A yields four different plots with titles rather than two plots per figure as was done here to save space. The student should experiment with the simulation parameters, especially the tone frequency f_1, and observe the results of the parameter changes.)

13.5.2 MATLAB Example 2

In this example we illustrate two aspects of the time-varying channel: frequency selectivity and time variation. In the previous example the channel was allpass or frequency nonselective (sometimes also referred to as a flat channel). We now modify the system model so that the complex lowpass equivalent impulse response has the form

$$\widetilde{h}(\tau,t) = \widetilde{a}_1(t)\delta(\tau-\tau_1) + \widetilde{a}_2(t)\delta(\tau-\tau_2) \tag{13.79}$$

It can be shown that this model is frequency selective by considering a time-invariant version with

$$\widetilde{h}(\tau, t) = \widetilde{a}_1 \delta(\tau - \tau_1) + \widetilde{a}_2 \delta(\tau - \tau_2) \tag{13.80}$$

which leads to the transfer function, with τ_1 assumed equal to zero, given by

$$H(f) = \widetilde{a}_1 + \widetilde{a}_2 \exp(-j2\pi f \tau_2) \tag{13.81}$$

The transfer function given above has different values at different frequencies and hence is frequency selective.

The complex lowpass equivalent output of the time-varying system given by (13.79) is

$$\widetilde{y}(t) = \widetilde{a}_1(t)\widetilde{x}(t - \tau_1) + \widetilde{a}_2(t - \tau_2)\widetilde{x}(\tau - \tau_2) \tag{13.82}$$

If the input to the system is a BPSK signal, the time-varying aspect of the system attenuates and phase rotates the BPSK signal as a function of time. The frequency selective nature of the channel, which is due to the two delayed components of the output [Equations (13.79) and (13.81)], manifests itself in the form of intersymbol interference. Both of these effects are clearly illustrated in the simulation results shown in Figure 13.16. These simulation results were obtained by executing the simulation given in Appendix B with the following parameters: BPSK symbol rate = 512 bits/s, sampling rate = 16 samples per bit, doppler bandwidth for $\widetilde{a}_1(t)$,

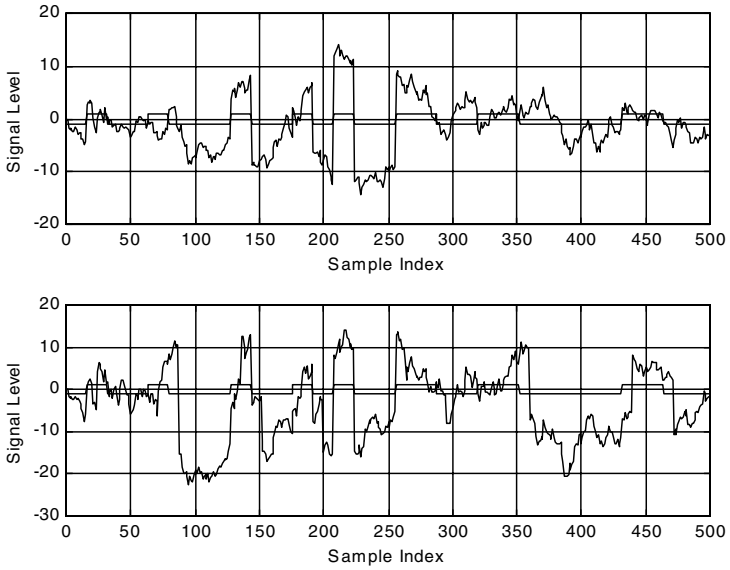


Figure 13.16 Effect of one (top) and two (bottom) time-varying components.

$\widetilde{a}_2(t) = 16$Hz, $\tau_1 = 0$, and $\tau_2 = 8$ samples (half the bit duration). In Figure 13.16 the top frame illustrates the effect of one time-varying component, and the bottom frame illustrates the effect of two components. Only the first 500 samples are shown. (Note: The program given in Appendix B also generates and displays the PSD of the first component impules response. As in the previous example, the MATLAB code given in Appendix B yields four different plots with titles rather than two plots per figure as was done here to save space. The student should experiment with the simulation parameters and observe the impact of parameter changes.)

13.6 Summary

Techniques for modeling and simulating linear time-varying systems were discussed in this chapter. These techniques will be used in the next chapter for simulating mobile communication channels.

Linear time-varying systems are usually characterized by a time-varying impulse response $\widetilde{h}(\tau, t)$, which is the response of the system measured at time t to an input applied at time $t - \tau$. The input-output relationship will be a convolution integral in the time domain

$$\widetilde{y}(t) = \int_{-\infty}^{\infty} \widetilde{h}(\tau, t)\widetilde{x}(t - \tau)\, d\tau \tag{13.83}$$

In the frequency domain the input-output relationship is given by

$$\widetilde{Y}(f) = \int_{-\infty}^{\infty} H(f_1, f - f_1)\, df_1 \tag{13.84}$$

which explains the spectral spreading (sometimes referred to as the doppler spread) that takes place in a time-varying system such as a mobile communication channel.

The convolution integral given in (13.83) can be approximated by a finite convolution, which leads to an FIR simulation model

$$\widetilde{y}(kT_s) \cong T_s \sum_{n=-m}^{m} \widetilde{h}(nT, kT_s)\widetilde{x}(kT_s - nT) \tag{13.85}$$

with time-varying tap gains functions. In the context of mobile communication channels, the time variations will be modeled by complex Gaussian processes with a given autocorrelation or power spectral density function. Sampled values of correlated Gaussian process can be generated by filtering an uncorrelated Gaussian sequence with an appropriate filter.

Linear time-varying components introduce two different forms of distortion. The first one is due to the time-varying nature of the response, and the second one is due to the frequency-selective aspects of the system. Two simulation examples were presented and illustrate the modeling and simulation approaches as well as the effects of time-varying components on two different types of input signals: an unmodulated tone to show the spectral spreading and a BPSK signal to show the effects of the time-varying and frequency-selective aspects.

13.7 Further Reading

The tapped delay line model for time-varying systems based on the sampling principle was first derived by Kailath [1]. A general reference for time-varying systems is the book by D'Angelo [2]. Additional details and examples of simulation models may be found in [3].

13.8 References

1. T. Kailath, "Channel Characterization: Time-Variant Dispersive Channels," *Lectures on Communication Systems*, E. J. Baghdady, Editor, New York: McGraw-Hill, 1961.

2. H. D'Angelo, *Linear Time Varying Systems: Analysis and Synthesis*, Boston: Allyn and Bacon, 1970.

3. M. C. Jeruchim, P. B. Balaban, and K. S. Shanmugan, *Simulation of Communication Systems*, 2nd ed., New York: Kluwer Academic/Plenum Publishers, 2000.

13.9 Problems

13.1 Develop an approach (i.e., write a short paper outlining an approach) for simulating a time-varying system whose behavior is described in the form of an n^{th} order differential equation whose coefficients are functions of time. For example

$$\frac{d^2y}{dt^2} + g_1(t)\left(\frac{dy}{dt}\right)^2 + g_2\,|y(t)| = x(t)$$

13.2 Develop an approach (as in Problem 13.1 write a short paper outlining an approach) for simulating a time-varying system in the frequency domain using two-dimensional DFTs—assuming that the system can be represented in the sampled time domain as an FIR approximation given in (13.70).

(a) First consider a deterministic model.

(b) How would you modify the simulation model if $g_n(t)$ are random processes?

13.3 The transmitted signal in a mobile communication system is an unmodulated tone $x(t) = \cos(2\pi f t)$ where $f = f_0 + f_i$. Let $f_0 = 1$ GHz be the nominal carrier and let $f_i = 512$ Hz. Consider simulating the received signal (and its spectrum) at the mobile over a time interval of 2 seconds during which the mobile is moving toward the base station at a constant velocity of 40 miles per hour, starting from a distance of 1 mile from the base station. Note: The received signal can be written as

$$y(t) = a(t)\cos\left[2\pi\left(f_c + f_i\right)\left(t - \Delta(t)\right)\right]$$

where

$$\Delta(t) = d(t)/c$$

and

$$a(t) = k/d^2(t)$$

In the preceding expressions $d(t)$ is the distance between the base station and the mobile at time t, c is the velocity of light, $\Delta(t)$ is the propagation delay, and $a(t)$ is the attenuation of the signal at time t, which can be assumed constant over a 2-second interval. Carry out this simulation using the complex lowpass equivalent model. (The output "spectrum" is the magnitude squared of the DFT.)

13.4 In Problem 13.3 assume now that the mobile accelerates steadily from 40 miles per hour to 50 miles per hour over the 2-second interval. What happens to the output spectrum?

13.5 Rework Problem 13.3 assuming that the mobile is on a busy highway and its speed, measured at time intervals of 0.01 second, can increase or decrease randomly from 50 mph by 1/10 mph with a probability of 1/2.

13.6 Fill in the details of the IIR filter for synthesizing the power spectral density given in (13.76).

13.7 In many simulation problems involving time-varying systems, the power spectral density for the random process model will be assumed to be uniform (white) over the doppler bandwidth. Synthesize an FIR filter for generating a complex Gaussian process with a flat power spectral density over a doppler bandwidth of 64 Hz and repeat the simulations given in MATLAB Example 1.

13.8 Repeat Problem 13.7 with a doppler power spectral density that has a Gaussian shape defined by

$$S_{\tilde{h}\tilde{h}}(f) = \exp\left(-f^2/\sigma^2\right)$$

where $\sigma = 64$ Hz.

13.9 Resimulate MATLAB Example 1 with the two-tone input

$$\widetilde{x}(t) = \exp\left[j2\pi(128)t\right] + \exp\left[j2\pi(512)t\right]$$

13.10 The system described in MATLAB Example 2 is frequency selective. Replace the random binary input in this example with an input of the form

$$\widetilde{x}(t) = \exp\left[j2\pi(128)t\right] + \exp\left[j2\pi(512)t\right]$$

and show via simulation that the two input tones are affected differently by the time-varying system, thus demonstrating the frequency-selective nature of the system. Compare the results with Problem 13.9.

13.10 Appendix A: Code for MATLAB Example 1

```
% File: c13_tiv1.m
%
% Set default parameters}
f1 = 512;                       % default signal frequency
bdoppler = 64;                  % default doppler sampling
fs = 8192;                      % default sampling frequency
tduration = 1;                  % default duration
%
ts = 1.0/fs;                    % sampling period
n = tduration*fs;               % number of samples
t = ts*(0:n-1);                 % time vector
x1 = exp(i*2*pi*f1*t);          % complex signal
zz = zeros(1,n);
%
% Generate Uncorrelated seq of Complex Gaussian Samples
z = randn(1,n) + i*randn(1,n);
%
% Filter the uncorrelated samples to generate correlated samples
coefft = exp(-bdoppler*2*pi*ts);
h = waitbar(0,'Filtering Loop in Progress');
for k =2:n
    zz(k) = (ts*z(k)) + coefft*zz(k-1);
    waitbar(k/n)
end
close(h)
y1 = x1.*zz;                    % filtered output of LTV system
%
% Plot the results in time domain and frequency domain
[psdzz,freq]=log_psd(zz,n,ts);
figure;
plot(freq,psdzz); grid;
ylabel('Impulse Response in dB')
xlabel('Frequency')
title('PSD of the Impulse Response');
zzz=abs(zz.^2)/(mean(abs(zz.^2)));
figure;
plot(10*log10(zzz)); grid;
ylabel('Sq. Mag. of h(t) in dB')
xlabel('Time Sample Index')
title('Normalized Magnitude Square of the Impulse Response in dB');
[psdx1,freq]=log_psd(x1,n,ts);
figure;
plot(freq,psdx1); grid;
```

```
ylabel('PSD of Tone Input in dB')
xlabel('Frequency')
title('PSD of Tone Input to the LTV System');
[psdy1,freq]=log_psd(y1,n,ts);
figure;
plot(freq,psdy1); grid;
ylabel('PSD of Output in dB')
xlabel('Frequency')
title('Spread Output of the LTV System');
% End of script file.
```

13.10.1 Supporting Program

Program `log_psd.m` is defined in Appendix A of Chapter 7.

13.11 Appendix B: Code for MATLAB Example 2

```
% File: c13_tiv2.m
%
% Set default parameters
symrate = 512;
nsamples = 16;
nsymbols = 128;
bdoppler = 16;
ndelay = 8;
%
n = nsymbols*nsamples;
ts = 1.0/(symrate*nsamples);
%
% Generate two uncorrelated seq of Complex Gaussian Samples
z1 = randn(1,n) + i*randn(1,n);
z2 = randn(1,n) + i*randn(1,n);
%
% Filter the two uncorrelated samples to generate correlated sequences
coefft = exp(-bdoppler*2*pi*ts);
zz1 = zeros(1,n);
zz2 = zeros(1,n);
for k = 2:n
    zz1(k) = z1(k)+coefft*zz1(k-1);
    zz2(k) = z2(k)+coefft*zz2(k-1);
end
%
% Generate a BPSK (random binry wavefrom and compute the output)
M = 2; % binary case
x1 = mpsk_pulses(M,nsymbols,nsamples);
y1 = x1.*zz1; % first output component
y2 = x1.*zz2; % second output component
y(1:ndelay) = y1(1:ndelay);
y(ndelay+1:n) = y1(ndelay+1:n)+y2(1:n-ndelay);
%
% Plot the results
[psdzz1,freq] = log_psd(zz1,n,ts);
figure; plot(freq,psdzz1); grid;
title('PSD of the First Component Impulse Response');
nn = 0:255;
figure; plot(nn,imag(x1(1:256)),nn,real(y1(1:256))); grid;
title('Input and the First Component of the Output');
xlabel('Sample Index')
ylabel('Signal Level')
figure; plot(nn,imag(x1(1:256)),nn,real(y(1:256))); grid;
```

```
title('Input and the Total Output')
xlabel('Sample Index')
ylabel('Signal Level')
% End of function file.
```

13.11.1 Supporting Routines

Program `log_psd.m` is defined in Appendix A of Chapter 7.

13.11.2 mpsk_pulses.m

```
% File: mpsk_pulses.m
function [x] = mpsk_pulses(M,nsymbols,nsamples)
% This function genrates a random MPSK complex NRZ waveform of
% length nsymbols; Each symbol is sampled at a rate of nsamples/bit
%
u = rand(1,nsymbols);
rinteger= round ((M*u)+0.5);
phase = pi/M+((rinteger-1)*(2*pi/M));
for m = 1:nsymbols
    for n = 1:nsamples
        index = (m-1)*nsamples + n;
        x(1,index) = exp(i*phase(m));
    end
end
% End of function file.
```

Chapter 14

MODELING AND SIMULATION OF WAVEFORM CHANNELS

14.1 Introduction

Modern communication systems operate over a broad range of communication channels including twisted pairs of wires, coaxial cable, optical fibers, and wireless channels. All practical channels introduce some distortion, noise, and interference. Appropriate modulation, coding, and other signal-processing functions such as equalization, are used to mitigate the degradation induced by the channel and to produce a system that satisfies the throughput and quality of service objectives while meeting the constraints on power, bandwidth, complexity, and cost. If the channel is relatively benign (e.g., does not significantly degrade the signal), or is well characterized, the design of the communication system is relatively straightforward.

What complicates the design is that many communication channels, such as the mobile radio channel, introduce significant levels of interference, distortion, and noise. The mobile radio channel is also time varying and undergoes fading. In addition, some channels are so variable that they are difficult to characterize. Furthermore, wireless communication systems, such as next-generation PCS, must be designed to operate over radio channels all over the world, in a variety of envi-

ronments from urban areas to hilly terrains, and under a wide variety of weather conditions. While it is possible to build prototypes of a proposed system and field-test the prototype in many locations around the globe, such an approach will be very expensive and will not be feasible in the early stages of the system design process when a number of candidate designs must be explored. The only feasible approach is to create appropriate models for the channel, and base the *initial design* on those models.

Given either deterministic or statistical models for communications channels, it might be possible, at least in the initial stages of communication system design, to use analytical approaches for evaluating the performance of a given design. For example, if we can assume that the "fading" in a channel has a Rayleigh amplitude probability density function, and the noise is additive Gaussian, the probability of error for a binary communication system operating over this channel can be expressed as

$$P_e = 1/2\overline{\gamma}_b \tag{14.1}$$

where $\overline{\gamma}_b$ is the "average" value of the signal-to-noise ratio (SNR) at the receiver input. This expression can then be used to determine such things as the transmitter power required to ensure a given error probability. However, when the system is actually built, implementation effects such as nonideal filters and nonlinear amplifiers must be considered. These effects are difficult to characterize analytically and, in most cases, one must resort to simulation or to a combination of simulation and analytical analysis. Thus, modeling and simulation play a central role in the design of communication systems. These two topics are covered in this chapter with an emphasis on simulation approaches and methodologies for wireless communication channels.

14.1.1 Models of Communication Channels

While a communication channel represents a physical medium between the transmitter and the receiver, the "channel model" is a representation of the input-output relationship of the channel in mathematical or algorithmic form. This model may be derived from measurements, or based on the theory of the physical propagation phenomena. Measurement-based models lead to an empirical characterization of the channel in the time or frequency domain, and often involve statistical descriptions in the form of random variables or random processes. The parameters of the underlying distributions and power spectral densities are usually estimated from measured data. While measurement-based models instill a high degree of confidence in their validity, and are often the most useful models for successful design, the resulting empirical models often prove unwieldy and difficult to generalize unless extensive measurements are collected over the appropriate environments. For example, it is very difficult to use measurements taken in one urban location to characterize a model for another urban location unless a substantial amount of data is collected over a wide variety of urban locations, and the necessary underlying theory is available to justify extrapolating the model to the new location.

Developing mathematical models for the propagation of signals over a transmission medium requires a good understanding of the underlying physical phenomena. For example, to develop a model for an ionospheric radio channel, one must understand the physics of radio-wave propagation. Similarly, a fundamental understanding of optical sciences is needed to develop models for single mode and multimode optical fibers. Communication engineers rely on experts in the physical sciences to provide the fundamental models for different types of physical channels.

One of the challenges in channel modeling is the translation of a detailed physical propagation model into a form that is suitable for simulation. Mathematical models, from a physical perspective, might often be extremely detailed and may not be in a form suitable for simulation. For example, the mathematical model for a radio channel may take the form of Maxwell's equations. While accurate, this model must be simplified and converted to a convenient form, such as a transfer function or impulse response, prior to using it for simulation. Fortunately, this is a somewhat easier process than deriving fundamental physical models and specifying the parameters of such models. Once a physical model has been derived, and the parameter values specified, translating the physical model into a simulation model (algorithm) is usually straightforward.

14.1.2 Simulation of Communication Channels

Physical communication channels such as wires, wave guides, free space, and optical fibers often behave linearly. Some channels, such as the mobile radio channel, while linear, may behave in a random time-varying manner. The simulation model of these channels falls into one of the following two categories:

1. Transfer function models for time-invariant channels. Examples are wires, free-space propagation, and optical fibers. In such models, the channel is assumed to be static in nature (i.e., the channel has a time-invariant impulse response), which provides a particular frequency response due to the fixed delays within the channel. The transfer function of the time-invariant channel is said to be "flat" if the applied message source has a bandwidth for which the channel has a constant gain response. The channel is said to be "frequency selective" if the applied modulated message source has a bandwidth over which the channel has a significant gain variation.

2. Tapped delay line (TDL) models for time-varying channels. An important example is the mobile radio channel. For these channel models, the channel is assumed to vary over time. If the channel changes during the smallest time interval of interest for an applied signal, the channel is said to be "fast fading." If the channel remains static for a large number of consecutive symbols of the applied source, the channel is said to be "slow fading" and the channel can be treated as in (1) above over the particular span of time for which the channel is static.

Transfer function models can be simulated in either the time domain or frequency domain using finite impulse response (FIR) or infinite impulse response (IIR) filters.

Empirical models in the form of measured or synthesized impulse or frequency responses are usually simulated using FIR techniques. Analytical expressions for the transfer function are easier to simulate using IIR techniques. IIR and FIR filters were discussed in detail in Chapter 5.

Simulation models for randomly time-varying (fading) channels take the form of TDLs with tap gains and delays that are random processes. Given the random process model for the underlying time variations (fading), the properties of the tap gain process can be derived and simulated using the techniques discussed in Chapter 13. If the channel is assumed to be slowly time varying, so that channel conditions do not change over many transmitted symbols, then we can use a snapshot (i.e., static impulse response) of the channel for simulation. This may be repeated as channel conditions change. By repeating the simulations for a large number of channel conditions, we can infer system performance over longer periods of time using performance measures, such as outage probabilities, as discussed in Chapter 11.

14.1.3 Discrete Channel Models

The focus of this chapter is on waveform-level channel models, which are used to represent the physical interactions between a transmitted waveform and the channel. Waveform channel models are sampled at an appropriate sampling frequency. The resulting samples are processed through the simulation model. Another technique, which is often more efficient for some applications, is to represent the channel by a finite number of states. As time evolves, the channel state changes in accordance with a set of transition probabilities. The channel can then be defined by a Markov chain. The resulting channel model most often takes the form of a hidden Markov model (HMM). Assuming that the HMM is constructed correctly, simulations based on the HMM allow the performance of a communication system to be accurately characterized with minimum computational burden. Discrete channel models and HMMs are the subject of the following chapter.

14.1.4 Methodology for Simulating Communication System Performance

Simulating the performance of a communication system operating over a time-invariant (fixed) channel is rather straightforward. The channel is simply treated as another linear time-invariant (LTIV) block in the system. Time-varying channels, on the other hand, require a number of special considerations. The methodology used will depend on the objective of the simulation and whether the channel is varying slowly or rapidly with respect to the signals and subsystems that are being simulated. Another important factor is the relationship between the bandwidth of the applied signal and the bandwidth of the channel. The complexity of a useful channel model is a function of both the time and frequency characteristics of both the source and the channel.

14.1.5 Outline of Chapter

The first part of this chapter is devoted to the development of models for communication channels, starting with simple transfer function models for "wired" or "guided" channels. These channels include twisted pairs, cables, waveguides, and optical fibers. These channels are linear and time invariant and, therefore, a transfer function or static impulse response model is sufficient. We then consider models for free space radio channels that are linear but may be time varying.

The second part of this chapter deals with the simulation of communication channels with the emphasis on the implemantation of TDL (tapped delay line) models for randomly time-varying channels. Three different TDL models of increasing complexity and capabilities are developed.

We conclude the chapter with the description of a methodology for simulating the performance of communication systems operating over fading channels. Throughout the chapter, near-earth and mobile communication channels will be emphasized, since these channels present most of the challenges in the modeling and simulation of channels, and also because of the current high level of interest in wireless communications.

14.2 Wired and Guided Wave Channels

Electrical communication systems use a variety of conducting media such as twisted pairs of wires and coaxial cable. These channels can be adequately characterized by RLC circuit models, and the input-output signal transfer characteristics can be modeled by a transfer function. Cable manufacturers often provide impedance characteristics of the transmission line models for the cables, and it is easy to derive transfer function models from this data. The transfer function is then used as a simulation model. It is also easy to measure the frequency response of varying lengths of cable and derive a transfer function model based on the resulting measurements. In a large cable network it might be necessary to define the channel using a number of random variables that characterize the parameters of a resulting transfer function or static impulse response. The channel, in that instance, may be treated as time invariant and, therefore, a time-varying model is not needed.

Waveguides and optical fibers can also be included in the broad category of guided wave transmission media. While the mode of propagation might vary, channels in this category can be modeled as time-invariant linear systems characterized by transfer functions.

Guided lightwave communication systems use optical fibers, while free-space optical communication systems transmit light through the air. The most common type of lightwave communication system uses either a single-mode or multimode fiber cable as the channel, and has a binary digital source and a receiver that makes a decision based on the energy received during each bit interval.

Besides attenuating the transmitted pulses, the optical fiber distorts or spreads the transmitted pulses. There are two different distortion mechanisms: chromatic dispersion and intermodal dispersion. Chromatic dispersion is a result of the differences in the propagation velocities of different transmitted spectral components.

Intermodal dispersion is seen in multimode fibers and results from a large number of propagation paths traveling along the fiber and arriving at the detector input with different delays. This is a multipath effect. Joints and splices in a fiber network cause reflections that can be approximated as additional intermodal dispersion. The multipath channel model was briefly introduced in Chapter 4 and will be studied in more detail in Section 14.4. While the emphasis in Section 14.4 is on the fading radio channel, the material to be presented is applicable to a wide variety of channels, including cables and optical fibers.

The relationship between the input and the output of a fiber can be described by the lowpass equivalent transfer function [1, 2]

$$H(f) = \int_{-\infty}^{\infty} S(\lambda)G(\lambda)H_{im}(\lambda)H_c(\lambda, f)d\lambda \tag{14.2}$$

where $S(\lambda)$ is the source spectrum as a function of wavelength λ, $G(\lambda)$ is the frequency-selective gain of the fiber, $H_{im}(\lambda)$ is the intermodal dispersion, and $H_c(\lambda, f)$ is the chromatic dispersion [2]. The intermodal dispersion is

$$H_{im}(f) = \frac{1}{\sigma_{im}\sqrt{2\pi}} \exp\left[\left(-\sigma_{im}^2(2\pi f)^2/2\right) - j2\pi f t_d\right] \tag{14.3}$$

where σ_{im} is the rms impulse response width and t_d is the fiber time delay. The chromatic dispersion is

$$H_c(\lambda, f) = \exp\left[-j2\pi f l T(\lambda)\right] \tag{14.4}$$

where l is the fiber length and $T(\lambda)$ is the group delay of the fiber [2].

The source spectrum $S(\lambda)$, the dispersion characteristics $T(\lambda)$, and the loss $L(\lambda)$ are obtained from the manufacturer's data sheets for the source and the fiber, and are used to compute the transfer function by substituting them in (14.2) and carrying out the integration numerically for different values of f. Several approximations for $S(\lambda)$ and $T(\lambda)$ are used to simplify the computation of the transfer function [1, 2, 3]. For example, the source spectrum can be assumed to be a frequency impulse for ideal sources. A Gaussian approximation with mean λ_0 can be used for most practical sources. The group delay function is often approximated by a parabolic function in $\lambda - \lambda_0$. Once the integral in (14.2) is evaluated, it is stored in tabular form, and the simulations are carried out using an FIR implementation for the channel.

The model given in (14.2) is an input power to output power transfer function model for the fiber, and is valid for use in direct detection lightwave communication systems in which the source spectrum is very narrow compared to the modulation bandwidth. For wideband systems, and for coherent optical communication systems, the model is not valid. The reader is referred to the lightwave communications literature for appropriate transfer function models for these systems [1, 2, 3].

14.3 Radio Channels

Radio channels have been used for long-distance communications since the early days of electrical communications starting with Marconi's experiments in radio

telegraphy. The propagation of radio waves through the atmosphere, including the ionosphere, which extends several hundred kilometers above the surface of the earth, is an extremely complex phenomenon. Atmospheric propagation takes on a wide range of behaviors depending on many factors including the frequency and bandwidth of the signal, the types of antennas used, the terrain between the transmit and receive antennas (rural, urban, indoor, outdoor, etc.), and weather conditions (clear air, rain, fog, etc.). Atmospheric scientists have devoted considerable effort to the understanding and development of models that describe radio-wave propagation through the atmosphere. Also, many measurement programs were carried out over the past several decades to gather empirical propagation data for HF to microwave. All of these efforts have led to a somewhat better understanding of how to model radio-wave propagation through the atmosphere, and how to use these models to aid in the analysis, design, and simulation of modern communication systems. The literature on modeling radio channels is vast and any effort to summarize this literature in a few pages would be inadequate. Nevertheless, we will attempt to provide the reader with a sampling of the various approaches to modeling and simulating communication systems.

From a communication systems designer's point of view, propagation models fall into two categories: those that aid in the calculation of path losses and those that aid in the modeling of signal distortion that may be due to multipath effects or random variations in the propagation characteristics of the channel. While the first category of models is used to establish the link power budgets and coverage analysis during initial design, it is the latter class of models that aid in the detailed design of communication systems. Hence, our focus will be on the second category of models, with an emphasis on approaches to simulating them efficiently.

We begin our discussion of channel models with an "almost" free-space channel that treats the region between the transmit and receive antennas as being free of all objects that might absorb or reflect RF energy. It is also assumed that the atmosphere behaves as a uniform and nonabsorbing medium, and that the earth is infinitely far away from the propagation path. Such a model is, for example, appropriate for satellite links.

In this idealized model, the channel simply attenuates the signal, and waveform distortion does not occur. The attenuation is computed according to the *free-space propagation model* defined by

$$L_f = \left(\frac{4\pi d}{\lambda}\right)^2 \tag{14.5}$$

where λ is the wavelength of the transmitted signal and d is the distance between the transmitter and receiving antennas, both of which are assumed to be omnidirectional. The transmitter and receiver antenna gains are taken into account while calculating the actual received power.

For most practical channels in which the signal propagates through the atmosphere and near the ground, the free-space propagation channel assumption is not adequate. The first effect that must be included is the atmosphere, which causes absorption, refraction, and scattering. Absorption due to the atmosphere,

when considered over narrow bandwidths, results in additional attenuation. However, over larger bandwidths, absorption is frequency dependent and can usually be modeled by a transfer function. This filtering effect can be considered time invariant, or at least quasi-static, since the channel is very slowly changing with respect to the signal. Other atmospheric phenomena, such as phase distortion introduced by the ionosphere, can also be modeled by a phase response that is slowly varying or time invariant. Several examples of transfer function models used to characterize certain types of atmospheric channels are described in the following paragraphs.

Other atmospheric effects (other than absorption) and the presence of ground and other objects near the transmission path often lead to what is known as multipath propagation. Multipath propagation is the arrival of a signal over multiple reflected and/or refracted paths from the transmitter to the receiver. These effects can also be time varying due to changes in atmospheric conditions or relative motion of the transmitting and receiving antennas, as is the case in mobile communications. The term *scintillation,* which originated in radio astronomy, is used to describe time variations in channel characteristics due to physical changes in the propagation medium, such as variations in the density of ions in the ionosphere that reflect high frequency (HF or shortwave) radio waves. Multipath fading is the terminology used in mobile communications to describe changes in channel conditions and the resulting changes in the received signal characteristics. Models for multipath fading channels will be covered in a later section of this chapter.

14.3.1 Tropospheric Channel

Tropospheric (non-ionospheric) communications use VHF (30 to 300 MHz) and UHF (300 MHz to 3 GHz) frequency bands for communications over distances up to several hundred kilometers. In these frequency bands, the oxygen and water vapor present in the atmosphere absorb RF energy. The loss due to absorption is dependent on the frequency of the RF wave as well as the atmospheric conditions, particularly the relative humidity. A typical set of characteristics for propagation losses due to atmospheric absorption is given in [4].

The frequency selective absorption characteristics of the atmosphere can be approximated by a transfer function of the form [4]

$$H(f) = H_0 \exp\{j0.02096f \left[10^6 + N(f)\right] l\} \tag{14.6}$$

where $N(f)$ is the complex refractivity of the atmosphere in parts per million, and is given by

$$N(f) = N_0 + D(f) + jN''(f) \tag{14.7}$$

In (14.6) and (14.7) H_0 is a constant, N_0 is the frequency dependent refractivity, $D(f)$ is the refractive absorption, $N''(f)$ is the absorption, and l is the distance in km. These parameters are dependent on frequency and atmospheric conditions such as temperature, barometric pressure, and relative humidity. Typical values are tabulated in [4].

Given the atmospheric conditions and the bandwidth occupied by the transmitted signal, the transfer function can be computed empirically for various values of frequency using (14.6) and (14.7). The lowpass equivalent transfer function can be obtained by frequency translation, and the resulting channel model can be simulated using FIR techniques.

14.3.2 Rain Effects on Radio Channels

Rain has a significant impact on microwave propagation at higher frequencies (greater than 10 GHz), since the size of the rain drops is on the order of the wavelength of the transmitted signal. Various techniques have been proposed in the literature for modeling the effects of rain [5, 6]. The attenuation due to rainfall is a function of the rate of rainfall and frequency. At higher frequencies and rain rate, rain-induced attenuation, as well as depolarization, is much more significant. Thus, attenuation increases as both rain rate and frequency increase. In addition, there are resonant peaks in the attenuation characteristic that result in significantly greater attenuation in the neighborhood of these peaks. Substantial resonant peaks occur at 22 GHz and 60 GHz. The peak at 22 GHz is due to water vapor, and the peak at 60 GHz is due to molecular oxygen. These effects are well documented [5].

Attenuation curves due to rainfall are usually computed for a given geographic location using the statistics of the rain rates for that location. For satellite communications, the attenuation is computed as a function of the elevation angle of the ground station antenna (with respect to the horizon) and frequency. Lower elevation means that there is more rain water in the transmission path and hence the attenuation is higher. The effect of rainfall is typically computed for a given outage probability, which is the fraction of the time that the link BER will exceed an acceptable threshold value (usually 10^{-3} for voice communications and 10^{-6} for data links).

Over relatively small bandwidths, the effects of rain can be accounted for by simply including an additional attenuation term in the channel model. However, as the bandwidth of the signal becomes larger, the attenuation varies over the bandwidth, and a transfer function type model is required. The amplitude response of the transfer function has a linear tilt [on a log (dB) scale] and the phase can be assumed linear.

Free-space propagation channels at higher frequencies generally use highly directional antennas that have a particular polarization characteristic. When the carrier frequency is such that the wavelength is much greater than the size of atmospheric particles, and when there are no physical obstructions to induce multipath, it becomes possible to use antenna polarization to isolate channels. In communication systems that use multiple orthogonal polarizations for different signals, the depolarizing effect of rain must be considered. Depolarization means that energy in one polarization leaks into, or couples with, the energy in the orthogonal polarization. This produces cross-talk [6, 7, 8]. If the two signals that are transmitted on orthogonal polarizations are

$$\widetilde{s}_i(t) = A_i(t) \exp\left[j\phi_i(t)\right], \quad i = 1, 2 \tag{14.8}$$

the simplest model for the two received signals with depolarization is

$$\begin{aligned} \widetilde{r}_1(t) &= \alpha_{11}\widetilde{s}_1(t) + \alpha_{12}\widetilde{s}_2(t) \\ \widetilde{r}_2(t) &= \alpha_{21}\widetilde{s}_1(t) + \alpha_{22}\widetilde{s}_2(t) \end{aligned} \tag{14.9}$$

where the ratio $20\log(\alpha_{11}/\alpha_{21})$ is a measure of the cross-polarization interference (or XPI) on signal 1 from signal 2. While a variety of approximations are available in the literature for analyzing the effects of XPI on analog and digital communication systems, the need for simulation increases as the system departs from the ideal.

14.4 Multipath Fading Channels

14.4.1 Introduction

We now turn our attention to the modeling and simulation of multipath and motion-induced fading, which are two of the most severe performance-limiting phenomena that occur in wireless radio channels. In any wireless communication channel there can be more than one path in which the signal can travel between the transmitter and receiver antennas. The presence of multiple paths may be due to atmospheric reflection or refraction, or reflections from buildings and other objects. Multipath and/or fading may occur in all radio communication systems. These effects were first observed and analyzed for HF troposcatter systems in the 1950s and 1960s [9]. Much of the current interest is in the modeling and simulation of multipath fading in mobile and indoor wireless communications in the $1 - 60$ GHz frequency range. Although the fading mechanisms may be different, the concepts of modeling, analysis, and simulation are the same.

14.4.2 Example of a Multipath Fading Channel

To illustrate the basic approach to modeling fading channels, let us consider a mobile communication channel in which there are two distinct paths (or rays) from the mobile unit to a fixed base station, as illustrated in Figure 14.1. Although Figure 14.1 shows only two paths, it is easily generalized to N paths. For the N-path case the channel output (the input signal to the mobile receiver) is

$$y(t) = \sum_{n=1}^{N} a_n(t)x(t - \tau_n(t)) \tag{14.10}$$

where $a_n(t)$ and $\tau_n(t)$ represent the attenuation and the propagation delay associated with the n^{th} multipath component, respectively. Note that the delays and attenuations are shown as functions of time to indicate that, as the automobile moves, the attenuations and delays, as well as the number of multipath components, generally change as a function of time. In (14.10) the additional multipath components are assumed to be caused by reflections from both natural features, such as mountains, and manmade features, such as additional buildings. Furthermore, each multipath component or ray may be subjected to local scattering in the vicinity of the mobile due the presence of objects such as signs, road surfaces,

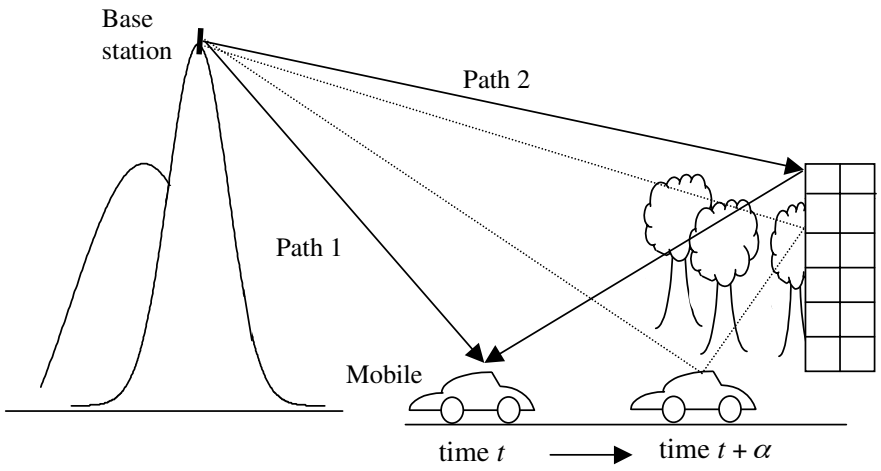


Figure 14.1 Example of a multipath fading channel.

Source: M. C. Jeruchim, P. Balaban, and K. S. Shanmugan, *Simulation of Communications Systems*, 2nd ed., New York: Kluwer Academic/Plenum Publishers, 2000.

and trees located near the mobile. The total signal that arrives at the receiver is made up of the sum of a large number of scattered components. These components add vectorially with random phases and hence the resulting complex envelope can be modeled as a complex Gaussian process by virtue of the central limit theorem. Movement over small distances of the order of $\lambda/2$ (about 15 cm at 1 GHz) can result in significant phase changes in the scattered components within a ray and cause components that add constructively at one location to add destructively at a location just a short distance away. This results in rapid fluctuations in the received signal amplitude/power and this phenomenon is called small scale or fast fading.

It should be noted that the small-scale fading is caused by changes in phase rather than by path attenuation, since the path lengths change by only a small amount over small distances. On the other hand, if the mobile moves over a larger distance and the path length increases from 1 km to 2 km, the received signal strength will drop, since the attenuation will change significantly. Movement over larger distances ($\gg \lambda$) and changes in terrain features affect attenuation and received signal power slowly. This phenomenon is called large-scale or slow fading and is modeled separately as discussed in the following sections of this chapter.

We have seen that the complex envelope of the receiver input due to a large number of scattered components is a complex Gaussian process. For the case in which this process is zero mean, the magnitude of the process is Rayleigh. If a line-of-sight (LOS) component is present, the process becomes Ricean. The effect of this will be demonstrated in Example 14.1.

The definition of fast fading and slow fading are, to some extent, in the eye of the beholder. However, when speaking about fast and slow fading we usually have an underlying symbol rate in mind. Slow-fading channels are typically defined as channels in which the received signal level is essentially constant over many symbols or data frames. Fast fading typically means that the received signal strength changes significantly over time intervals on the order of a symbol time. The definition of fast fading and slow fading therefore depends upon the underlying symbol rate.

We now determine the complex envelope of the received signal. Assume that the channel input (the transmitted signal) is a modulated signal of the form

$$x(t) = A(t)\cos(2\pi f_c t + \phi(t)) \tag{14.11}$$

Since waveform simulation is usually accomplished using complex envelope signals, we now determine the complex envelope for both $x(t)$ and $y(t)$.

The complex envelope of the transmitted signal is, by inspection,

$$\widetilde{x}(t) = A(t)\exp[\phi(t)] \tag{14.12}$$

Substituting (14.11) for $x(t)$ in (14.10) gives

$$y(t) = \sum_{n=1}^{N} a_n(t)A(t-\tau_n(t))\cos[2\pi f_c(t-\tau_n(t)) + \phi(t-\tau_n(t))] \tag{14.13}$$

which can be written

$$\begin{aligned} y(t) = \sum_{n=1}^{N} & a_n(t)A(t-\tau_n(t)) \\ & \cdot \mathrm{Re}\left\{\exp[j\phi(t-\tau_n(t))]\exp[-j2\pi f_c\tau_n(t)]\exp(j2\pi f_c t)\right\} \end{aligned} \tag{14.14}$$

Since $a_n(t)$ and $A(t)$ are both real, (14.14) can be written

$$\begin{aligned} y(t) = \mathrm{Re}\Bigg\{ & \sum_{n=1}^{N} a_n(t)A(t-\tau_n(t))\exp[j\phi(t-\tau_n(t))] \\ & \cdot \exp[-j2\pi f_c\tau_n(t)]\exp(j2\pi f_c t)\Bigg\} \end{aligned} \tag{14.15}$$

From (14.12) we recognize that

$$A(t-\tau_n(t))\exp[j\phi(t-\tau_n(t))] = \widetilde{x}(t-\tau_n(t)) \tag{14.16}$$

so that

$$y(t) = \mathrm{Re}\left\{\sum_{n=1}^{N} a_n(t)\widetilde{x}(t-\tau_n(t))\exp[-j2\pi f_c\tau_n(t)]\exp(j2\pi f_c t)\right\} \tag{14.17}$$

The complex path attenuation is defined as

$$\widetilde{a}_n(t) = a_n(t) \exp\left[-j2\pi f_c \tau_n(t)\right] \tag{14.18}$$

so that

$$y(t) = \mathrm{Re}\left\{\sum_{n=1}^{N} \widetilde{a}_n(t)\widetilde{x}(t-\tau_n(t)) \exp(j2\pi f_c t)\right\} \tag{14.19}$$

Thus, the complex envelope of the receiver input is

$$\widetilde{y}(t) = \sum_{n=1}^{N} \widetilde{a}_n(t)\widetilde{x}(t-\tau_n(t)) \tag{14.20}$$

The channel input-output relationship defined by (14.20) corresponds to a linear time-varying (LTV) system with an impulse response

$$\widetilde{h}(\tau, t) = \sum_{n=1}^{N} \widetilde{a}_n(t)\delta(t-\tau_n(t)) \tag{14.21}$$

In (14.21), $\widetilde{h}(t,\tau)$ is the impulse response of the channel measured at time t assuming that the impulse is applied at time $t-\tau$. Thus, τ represents the *elapsed time* or the propagation delay. In the absence of movement or other changes in the transmission medium, the input-output relationship is time invariant even though multipath is present. In this case, the transmission delay associated with the n^{th} propagation path and the path attenuation are constant (the channel is fixed) and

$$\widetilde{y}(t) = \sum_{n=1}^{N} \widetilde{a}_n \widetilde{x}(t-\tau_n) \tag{14.22}$$

For the fixed-channel case, the channel can be represented in the time domain by an impulse response of the form

$$\widetilde{h}(\tau) = \sum_{n=1}^{N} \widetilde{a}_n \delta(\tau-\tau_n) \tag{14.23}$$

The corresponding representation in the frequency domain is

$$H(f) = \sum_{n=1}^{N} \widetilde{a}_n \exp(-j2\pi f\tau_n) \tag{14.24}$$

We see that for the time-invariant channel case, the channel simply acts as a filter on the transmitted signal.

Example 14.1. In this example we simulate the BER performance of a QPSK system operating over a fixed 3-ray multipath channel with AWGN, and compare the BER performance with an identical system operating over an ideal AWGN channel (no multipath). In order to simplify the simulation model we will make the following assumptions:

1. The channel has three paths consisting of an unfaded LOS path and two Rayleigh components. The received power levels associated with each path, and the differential delays between the three paths, are simulation parameters.

2. The Rayleigh fading in the channel affects only the amplitude of the transmitted signal. The instantaneous phase is not affected.

3. The magnitude of the attenuation of each multipath component is constant over a symbol interval and has independent values over adjacent intervals (no doppler spectral shaping required).

4. No transmitter filtering is used, and the receiver model is an ideal integrate-and-dump receiver.

The received signal for this example can be written as

$$\widetilde{y}(t) = \underbrace{a_0\widetilde{x}(t)}_{\text{LOS}} + \underbrace{a_1R_1\widetilde{x}(t)}_{\text{Rayleigh}} + \underbrace{a_2R_2\widetilde{x}(t-\tau)}_{\text{Delayed Rayleigh}} \tag{14.25}$$

where R_1 and R_2 are two independent Rayleigh random variables representing the attenuation of the two Rayleigh paths, and τ is the relative delay between the two Rayleigh components. The Fourier transform of (14.25) is

$$\widetilde{Y}(f) = a_0\widetilde{X}(f) + a_1R_1\widetilde{X}(f) + a_2R_2\widetilde{X}(f)\exp(-j2\pi f\tau) \tag{14.26}$$

which leads to the channel transfer function

$$\widetilde{H}(f) = a_0 + a_1R_1 + a_2R_2\exp(-j2\pi f\tau) \tag{14.27}$$

Clearly, if the product $f\tau$ is not negligible over the range of frequencies occupied by the signal, the channel is frequency selective, which leads to delay spread and ISI. The values of a_0, a_1, and a_2 determine the relative power levels P_0, P_1, and P_2 of the three multipath components.

Simulations were conducted for each of the six sets of parameter values given in Table 14.1. For each scenario, the BER is evaluated using semianalytic estimation. In Table 14.1, the delay is expressed in terms of the sampling period. Since the simulation sampling frequency is 16 samples per symbol, $\tau = 8$ corresponds to a delay of one-half the sample period. (See Appendix for code.)

The simulation results for Scenarios 1 and 2 are illustrated in Figure 14.2. In Scenario 1, only a line-of-sight component is present. There is no multipath for Scenario 1 and this result provides the semianalytic estimation of the BER for a QPSK system operating in an AWGN environment. This simulation serves to verify the simulation methodology and provide baseline results representing an ideal QPSK system. For comparison purposes, this result is displayed along with the BER results for all five of the remaining scenarios. Table 14.1 shows that Scenario 2 results by adding a Rayleigh fading component to the LOS component of Scenario 1. This gives rise to a Ricean fading channel. Since $\tau = 0$, Scenario 2 is flat fading (not

Table 14.1 Scenarios for Fading Example

Scenario	P_0	P_1	P_2	τ **(samples)**	**Comments**
1	1.0	0	0	0	Validation
2	1.0	0.2	0	0	Ricean flat fading
3	1.0	0	0.2	0	Ricean flat fading
4	1.0	0	0.2	8	Ricean frequency selective fading
5	0	1.0	0.2	0	Rayleigh flat fading
6	0	1.0	0.2	8	Rayleigh frequency selective fading

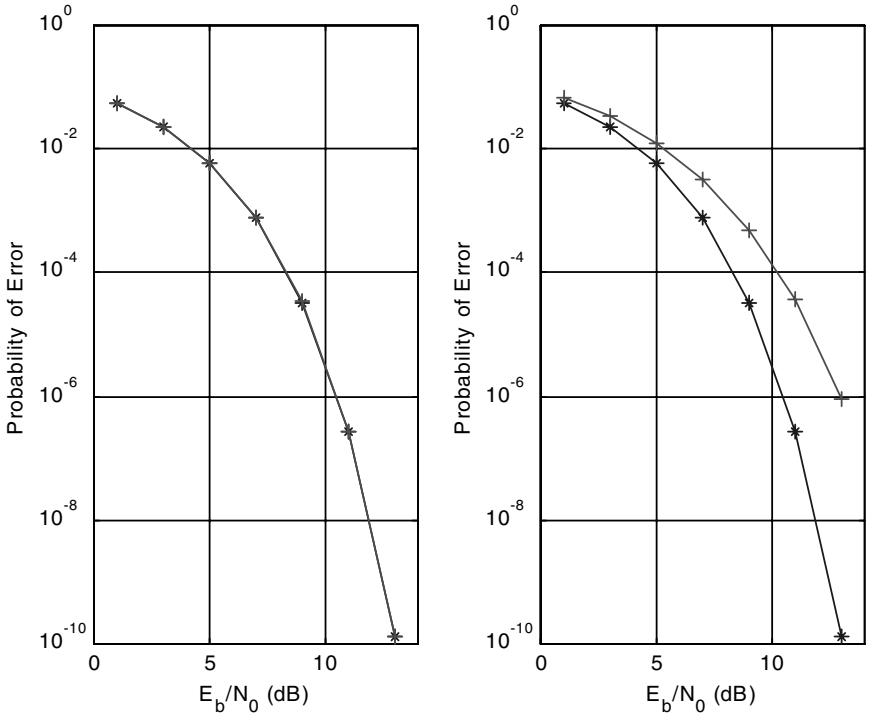


Figure 14.2 Scenario 1 (left-hand pane) and Scenario 2 (right-hand pane) illustrating the calibration run and Ricean flat fading.

frequency selective). Note the increase in BER compared to the baseline (no fading) result given in Scenario 1.

The simulation results for Scenarios 3 and 4 are illustrated in Figure 14.3. Scenario 3 is essentially equivalent to Scenario 2. The small difference is due to the fact that the fading process is different from that used in Scenario 2 due to a different initialization of the underlying random number generator. Scenario 4 is the same as Scenario 3 except that the fading is now frequency selective. Note that system performance is further degraded.

The simulation results for Scenarios 5 and 6 are illustrated in Figure 14.4. Note that for both of these scenarios there is no line-of-sight component present at the receiver input. Comparison of the Scenario 5 result with the preceding four results shows that, even for the flat-fading scenario (left-hand pane), the performance is worse than with any of the scenarios in which a line-of-sight component is present. Scenario 6 is the same as Scenario 5 except that the fading is now frequency selective. Note that system performace is further degraded. Rayleigh and Ricean channels will be explored in greater detail in the following sections. ■

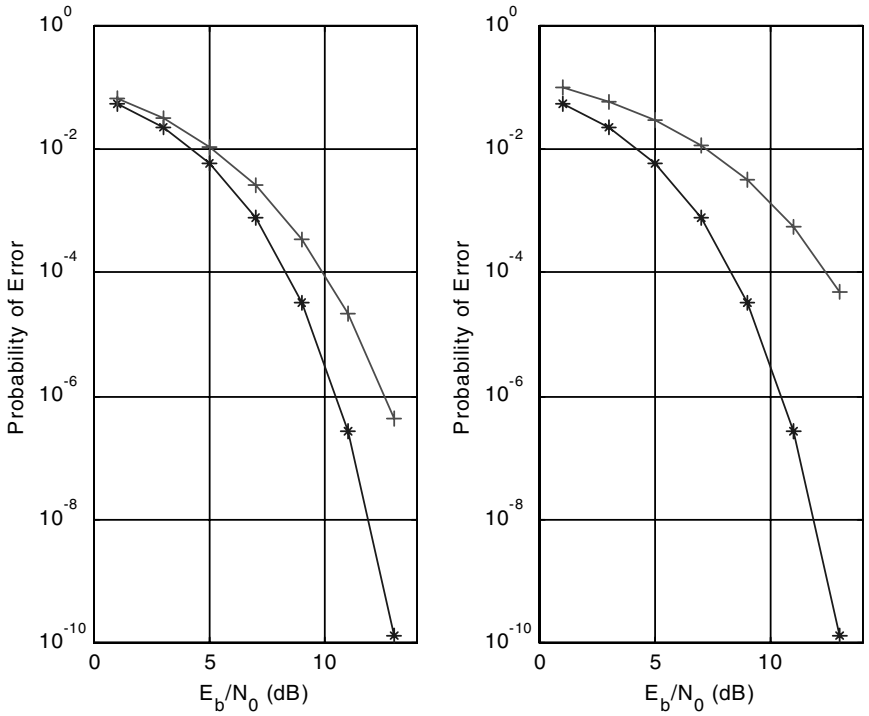


Figure 14.3 Scenario 3 (left-hand pane) and Scenario 4 (right-hand pane) illustrating Ricean flat fading and frequency selective fading.

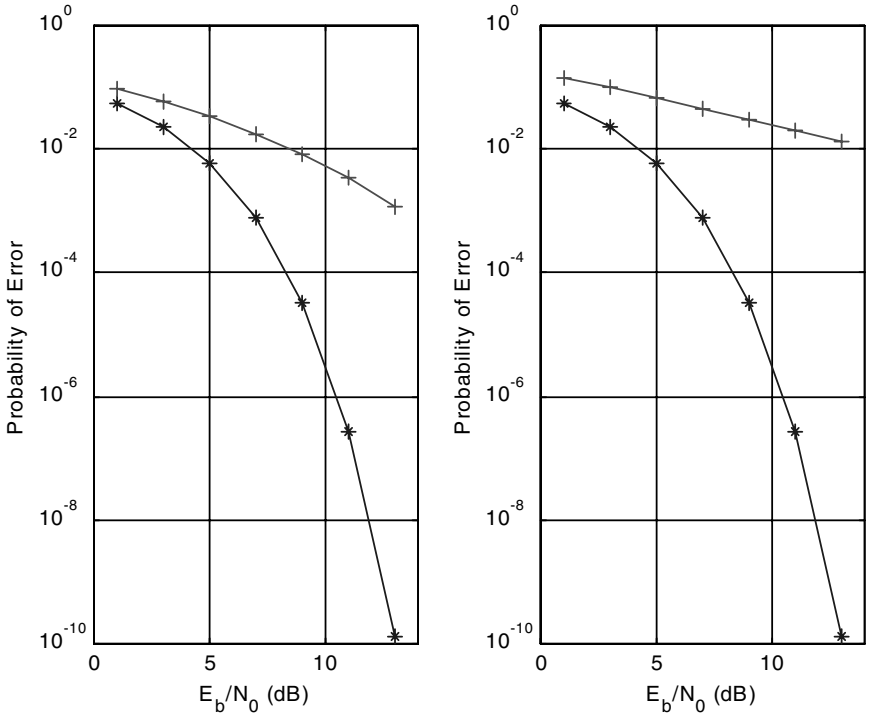


Figure 14.4 Scenario 5 (left-hand pane) and Scenario 6 (right-hand pane) illustrating Rayleigh flat fading and frequency selective fading.

Source: M. C. Jeruchim, P. Balaban, and K. S. Shanmugan, *Simulation of Communications Systems*, 2nd ed., New York: Kluwer Academic/Plenum Publishers, 2000.

14.4.3 Discrete Versus Diffused Multipath

The number of multipath components will vary depending on the type of channel. In microwave communication links between fixed microwave towers using large directional antennas (narrow beams), the number of multipath components will be small, whereas in an urban mobile communication system using omnidirectional antennas, there may be a large number of multipath components caused by reflections from buildings. The same will be true for indoor wireless communications where signals can bounce off walls, furniture, and other surfaces.

There are some situations like troposcatter channels, or some mobile radio channels, where it is more appropriate to view the received signal as consisting of a continuum of multipath components rather than as a collection of discrete components. This situation is called *diffused* multipath. We will see later on in this chapter that the diffused multipath channel can be approximated by a (sampled version of) discrete multipath channel for simulation purposes.

14.5 Modeling Multipath Fading Channels

The recent literature on communication systems contains a vast quantity of articles dealing with the modeling and analysis of *multipath fading channels*, particularly indoor wireless and outdoor mobile channels [10–15]. While a complete review of the literature is outside the scope of this chapter, we will provide a brief review of the modeling of outdoor mobile wireless channels leading to the development of simulation techniques. These modeling and simulation techniques can be applied to other multipath fading channels.

Modeling an outdoor mobile channel is usually carried out as a two-step process which represents large-scale (macro) and small-scale (mirco) effects of multipath and fading. As previously mentioned, large-scale fading represents attenuation or path loss over a large area, and this phenomenon is affected by prominent terrain features like hills, buildings, etc., between the transmitter and the receiver. The receiver is often hidden or shadowed by such terrain features, and the statistics of large-scale fading provide a way of computing the estimated signal power or path loss as a function of distance. Small-scale fading deals with large dynamic variations in the received signal amplitude and phase as a result of very small changes in the spatial separation between the transmitter and the receiver.

There are three mechanisms that affect the quality of the received signal in a mobile channel [13]: reflection, refraction, and scattering. Reflection occurs when the radio wave impinges upon a large, smooth surface (water or large metallic surfaces). Diffraction takes place when there is an obstruction in the radio path between the transmitter and receiver causing secondary radio waves to form behind the obstruction. This is called *shadowing,* and this phenomenon accounts for radio waves reaching the receive antenna even though there is no direct or line-of-sight path between the transmitter and the receiver. The third effect, scattering, results from rough surfaces whose dimensions are of the order of the wavelength, which causes the reflected energy to scatter in all directions.

While electromagnetic theory offers very complex models for these phenomena, it is possible to use simpler statistical models for the input-output relationship in a mobile channel. Specifically, the lowpass equivalent response of a mobile channel can be modeled by a complex impulse response [12] having the form

$$\widetilde{h}(\tau,t)=\left\{\left[\frac{k}{d^n}g_{sh}\left(p(t)\right)\right]^{1/2}\right\}\widetilde{c}(\tau,p(t)),\qquad d>1\text{ km} \tag{14.28}$$

where the term in braces models the large-scale fading, and $\tilde{c}(\tau,p(t))$ accounts for the small-scale fading as a function of the position of $p(t)$ at time t. The constant $K=-10\log_{10}(k)$ is the median dB loss at a distance of 1 km. Since the reference distance is 1 km, (14.28) is only valid for $d>1$ km. Typically, K is of the order of 87 dB at 900 MHz, d is the distance in meters between the transmitter and the receiver, and the path loss exponent n has a value of 2 for free space (for most mobile channels its value will range from 2 to 4, with higher values applying to obstructed paths). The factor $g_{sh}(p(t))$ accounts for shadowing due to buildings, tunnels, and other obstructions at a given location $p(t)$, and $G=10\log_{10}(g_{sh}(p(t)))$ is usually

modeled as a Gaussian variable with a mean of 0 dB and a standard deviation of 6 to 12 dB depending upon the environment (this model is called the lognormal shadowing model (see [13] for more details). It is a common practice to express the path loss [the term in braces in (14.28)] as

$$L(d)_{dB} = L(1 \text{ km})_{dB} + 10n \log(d) + X_\sigma \tag{14.29}$$

where X_σ is a zero mean Gaussian variable with a standard deviation of 6 to 12 dB.

In (14.28), $\tilde{c}(\tau, p(t))$ represents the complex lowpass equivalent impulse response of the channel at position $p(t)$, and the local multipath and fading that will result from small spatial displacements around the location $p(t)$. The path loss associated with large-scale fading, represented by the term in braces in (14.28), as well as fading due to shadowing, changes very slowly as a function of time at normal vehicular speeds compared to the rate of change of $\tilde{c}(\tau, p(t))$. Hence the channel attenuation due to large-scale fading and shadowing may be treated as a constant within a small local area, and the large-scale effect on system performance is reflected in the *average* received signal. The dynamic behavior of receiver subsystems such as tracking loops and equalizers, as well as the bit error rate of the system, will be affected significantly by the small-scale behavior modeled by $\tilde{c}(\tau, p(t))$. Hence much of the effort in the modeling and simulation of mobile wireless channels is focused on $\tilde{c}(\tau, p(t))$. In the following discussion we will use $\tilde{c}(\tau, t)$ as a shorter notation for $\tilde{c}(\tau, p(t))$.

14.6 Random Process Models

A variety of models have been proposed for characterizing multipath fading channels, and almost all of them involve using random process models to characterize fading (see [15] for an example). There are two classes of models for describing multipath, the discrete multipath model (finite number of multipath components), and the diffused multipath model (continuum of multipath components). In mobile radio communications, the first model is often used for waveform-level simulation of mobile radio channels, while the second model is used for troposcatter channels having narrowband modulation. In both of these cases, the channel is modeled as a linear time-varying system with a complex lowpass equivalent response $\tilde{c}(\tau, t)$. If there are N discrete multipath components, the output of the channel consists of the sum of N delayed and attenuated versions of the input. Thus

$$\tilde{y}(t) = \sum_{k=1}^{N(t)} \widetilde{a}_k(t)\, \widetilde{x}\,(t - \tau_k(t)) \tag{14.30}$$

The impulse response $\tilde{c}(\tau, t)$ is

$$\tilde{c}(\tau, t) = \sum_{k=1}^{N(t)} \tilde{a}_k(t) \delta(\tau - \tau_k(t)) \tag{14.31}$$

where $N(t)$ is the number of multipath components, and $\tilde{a}_k(t)$ and $\tau_k(t)$ are the complex attenuation and the delay of the k^{th} multipath at time t.

As previously mentioned, a multipath channel may be time invariant. However, for all practical channels of interest, the channel may be characterized as time varying (fading). Time variations arise for two reasons:

1. The environment is changing even though the transmitter and receiver are fixed; examples are changes in the ionosphere, movement of foliage, and movement of reflectors and scatterers.

2. The transmitter and the receiver are mobile even though the environment might be static. Hence, in practical multipath channels, N, a_k, and τ_k may all be randomly time varying. An example is illustrated in Figure 14.5.

Random fluctuations in the received signal due to fading can be modeled by treating $\tilde{c}(\tau, t)$ as a random process in t. If the received signal is made up of the sum of a large number of scattered components in each path, the central limit theorem leads to a model in which $\tilde{c}(\tau, t)$ can be represented as a complex Gaussian process in t. At any time t, the probability density function of the real and imaginary parts are Gaussian. This model implies that for each τ or τ_k, the ray is composed of a large number of unresolvable components. Hence, $\tilde{c}(\tau, t)$ and $\tilde{a}_k(t)$ are both complex Gaussian processes in t.

If $\tilde{c}(\tau, t)$ has a zero mean, the envelope $R = |\tilde{c}(\tau, t)|$ has a Rayleigh probability density function of the form

$$f_R(r) = \frac{r}{\sigma^2} \exp\left(-\frac{r^2}{2\sigma^2}\right), \qquad r > 0 \tag{14.32}$$

where σ^2 is the variance of the real and imaginary parts of $\tilde{c}(\tau, t)$.

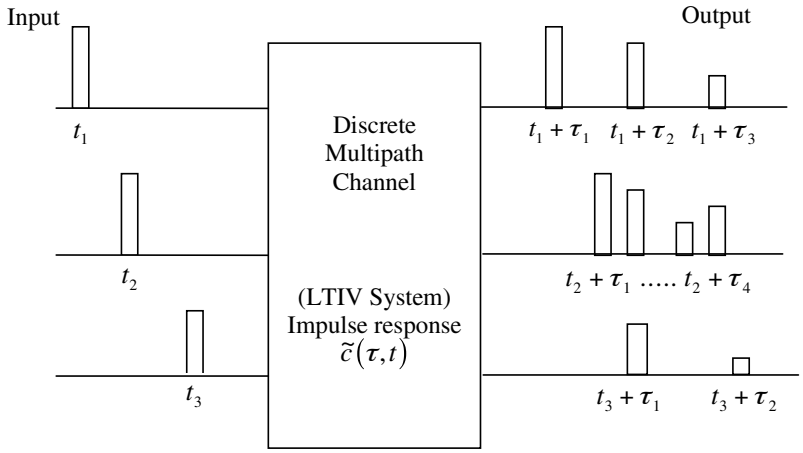


Figure 14.5 Example of a discrete multipath fading channel.

If $\tilde{c}(\tau, t)$ has a nonzero mean, which implies the presence of a line-of-sight nonfaded path (referred to as a specular component), then $R = |\tilde{c}(\tau, t)|$ has a Ricean probability density function of the form

$$f_R(r) = \frac{r}{\sigma^2} I_0\left(\frac{Ar}{\sigma^2}\right) \exp\left(-\frac{r^2 + A^2}{2\sigma^2}\right), \qquad r > 0 \tag{14.33}$$

where A is the nonzero mean of $\tilde{c}(\tau, t)$, and $I_0(z)$ is the modified Bessel function defined by

$$I_0(z) = \frac{1}{2\pi} \int_0^{2\pi} \exp(z\cos(u))\, du \tag{14.34}$$

The ratio $K = A^2/\sigma^2$, referred to as the Ricean factor, is an indicator of the relative power in the unfaded and faded components. Values of $K \gg 1$ indicate less severe fading, whereas $K \ll 1$ indicates severe fading.

The channel is called a Rayleigh fading channel or a Ricean fading channel depending on the pdf of $|\ \tilde{c}(\tau, t)\ |$. Other distributions for $|\ \tilde{c}(\tau, t)\ |$ such as Nakagami and Weibul are also possible [12]. Generalized probability density functions describing envelope statistics for a finite number of specular components, together with diffuse multipath, have recently been developed [16]. In these results, Ricean and Rayleigh fading are special cases. For discrete multipath channels, these pdfs apply to $|\ \tilde{a}_k(t)\ |$. While the pdf of $|\ \tilde{c}(\tau, t)\ |$ describes the instantaneous value of the complex impulse response, the temporal variations are modeled by either an appropriate autocorrelation function or power spectral density of the random process in the t variable. We describe these models now.

14.6.1 Models for Temporal Variations in the Channel Response (Fading)

The time-varying nature of the channel is mathematically modeled by treating $\tilde{c}(\tau, t)$ as a *wide sense stationary* (WSS) random process in t with an autocorrelation function

$$R_{\tilde{c}\tilde{c}}(\tau_1, \tau_2, \alpha) = E\left\{\tilde{c}^*(\tau_1, t)\tilde{c}(\tau_2, t + \alpha)\right\} \tag{14.35}$$

In most multipath channels, the attenuation and phase shift associated with different delays (i.e., paths) are assumed uncorrelated. This *uncorrelated scattering* (US) assumption leads to

$$R_{\tilde{c}\tilde{c}}(\tau_1, \tau_2, \alpha) = R_{\tilde{c}\tilde{c}}(\tau_1, \alpha)\delta(\tau_1 - \tau_2) \tag{14.36}$$

Equation (14.36) embodies both the wide sense stationary and uncorrelated scattering assumptions. It is often referred to as the WSSUS model for fading, and was originally proposed by Bello [9]. This autocorrelation function is denoted by $R_{\tilde{c}\tilde{c}}(\tau, \alpha)$ and is given by

$$R_{\tilde{c}\tilde{c}}(\tau, \alpha) = E\left\{\tilde{c}^*(\tau, t)\tilde{c}(\tau, t + \alpha)\right\} \tag{14.37}$$

By Fourier transforming the autocorrelation function we can obtain a frequency domain model for fading in the form of a power spectral density as

$$S(\tau,\lambda) = F\left\{(R_{\tilde{c}\tilde{c}}(\tau,\alpha)\right\} = \int_{-\infty}^{\infty} R_{\tilde{c}\tilde{c}}(\tau,\alpha)\exp(-j2\pi\lambda\alpha)\,d\alpha \tag{14.38}$$

The quantity $S(\tau,\lambda)$ is called the scattering function of the channel, and is a function of two variables, a time domain variable (delay) and a frequency domain variable, which is called the doppler frequency variable. The scattering function provides a single measure of the average power output of the channel as a function of delay and doppler frequency.

From the scattering function we can obtain the most important parameters of the channel which impact the performance of a communication system operating over the channel. We start with the "multipath intensity" profile, defined as

$$p(\tau) = R_{\tilde{c}\tilde{c}}(\tau,0) = E\left\{|\tilde{c}(\tau,t)|^2\right\} \tag{14.39}$$

which represents the average received power as a function of delay. Equation (14.39) is commonly referred to as the power-delay profile [13]. It can be shown that $p(\tau)$ is related to the scattering function via

$$p(\tau) = \int_{-\infty}^{\infty} S(\tau,\lambda)\,d\lambda \tag{14.40}$$

Another function that is useful for characterizing fading is the doppler power spectrum, which is derived from the scattering function according to

$$S_d(\lambda) = \int_{-\infty}^{\infty} S(\tau,\lambda)\,d\tau \tag{14.41}$$

The relationships between these functions are shown in Figure 14.6.

The multipath intensity profile is usually measured by probing the channel with a wideband RF waveform where the modulating signal is a high-rate PN sequence. By crosscorrelating the receiver output against delayed versions of the PN sequence and measuring the average value of the correlator output, one can obtain the power versus delay profile. Where measurements for mobile radio applications with a fixed base station and mobile user are concerned, the power delay profile is measured in short distance increments of fractions of a wavelength. The recorded power profile is then averaged over 10 to 20 wavelengths in order to average out the effects of Rayleigh fading. The correlation measurements made as a function of position, i.e., the spatial autocorrelation function, can be converted to a temporal correlation function by noting that $\Delta X = v\Delta t$, where ΔX is the incremental spatial movement of the mobile and v is the speed. Thus, the doppler spectrum can be obtained by transforming the temporal correlation function for any vehicle speed.

14.6.2 Important Parameters

The scattering function, the multipath intensity profile, and the doppler spectrum describe various aspects of a fading channel in detail. The two most important

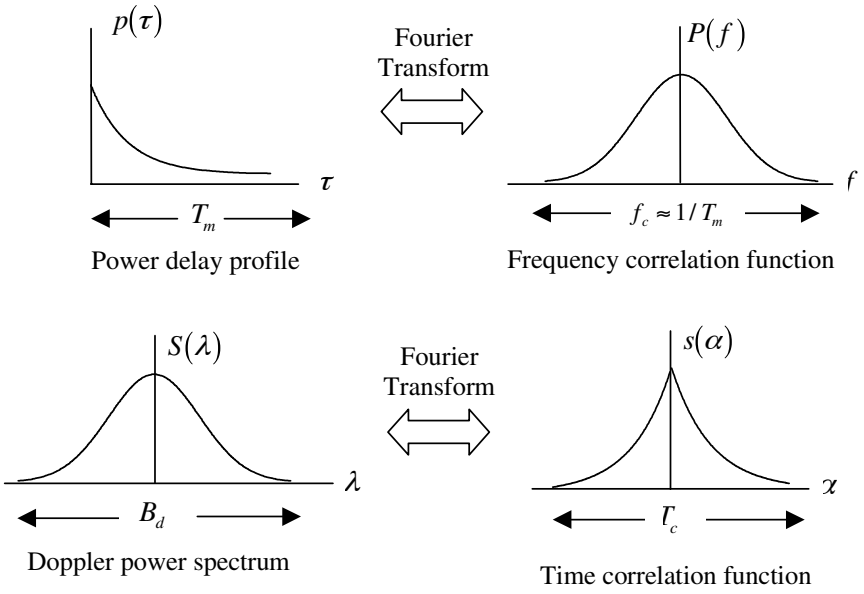


Figure 14.6 Relationship between various parts of the scattering function.

parameters, however, for simulating a fading channel are the multipath spread and the doppler bandwidth.

Multipath Spread

Important indicators of the severity of the multipath effect are the maximum delay spread and the rms delay spread. The (maximum) delay spread which represents the value $T_{\max}$ of the delay beyond which the received power $p(\tau)$ is very small, and the *rms delay spread* σ_τ, is defined as

$$\sigma_\tau = [< \tau^2 > - < \tau >^2]^{1/2} \tag{14.42}$$

where $< x >$ denotes the time-average value of x and

$$< \tau^k >= \frac{\int \tau^k p(\tau)\, d\tau}{\int p(\tau)\, d\tau} \tag{14.43}$$

When the delay spread is of the order of, or greater than, the symbol duration in a digital communication system, the delayed multipath components will arrive in different symbol intervals and cause intersymbol interference, which can adversely impact the BER performance. This is equivalent to the time-varying transfer function of a channel having a bandwidth less than the signal bandwidth. In this case, the channel behaves as a bandlimiting filter and is said to be frequency selective.

For a channel that is not frequency selective, the maximum delay spread is much smaller than the symbol duration T_s

$$T_{\max} \ll T_s \quad \text{or} \quad \sigma_T < 0.1T_s \tag{14.44}$$

In the nonfrequency-selective case, all of the delayed multipath components arrive within a short fraction of a symbol time. In this case, the channel can be modeled by a single ray, and the input-output relationship can be expressed as a multiplication. In other words

$$\widetilde{y}(t) = \widetilde{a}(t)\,\widetilde{x}(t) \tag{14.45}$$

For a frequency-selective channel

$$T_{\max} \gg T_s \quad \text{or} \quad \sigma_T > 0.1T_s \tag{14.46}$$

and the input-output relationship is the convolution

$$\widetilde{y}(t) = \widetilde{c}(\tau, t) \circledast \widetilde{x}(t) \tag{14.47}$$

where $\circledast$, as always, denotes convolution. While the delay *spread* (maximum or rms) has a significant impact on the performance of a communications system, it has been observed that the system performance is not very sensitive to the *shape* of the multipath intensity profile $p(\tau)$. The most commonly assumed forms for $p(\tau)$ are uniform and exponential.

Doppler Bandwidth

The doppler bandwidth, or the doppler spread, B_d, is the bandwidth of the doppler spectrum $S_d(\lambda)$ as defined by (14.41), and is an indicator of how fast the channel characteristics are changing (fading) as a function of time. If B_d is of the order of the signal bandwidth B_s ($\approx 1/T_s$), the channel characteristics are changing (fading) at a rate comparable to the symbol rate, and the channel is said to be fast fading. Otherwise the channel is said to be slow fading. Thus

$$\begin{aligned} B_d &\ll B_s \approx 1/T_s \quad \text{(Slow fading channel)} \\ B_d &\gg B_s \approx 1/T_s \quad \text{(Fast fading channel)} \end{aligned} \tag{14.48}$$

If the channel is slow fading, then a snapshot approach can be used to simulate the channel for performance estimation. Otherwise, the dynamic changes in the channel conditions must be explicitly simulated.

14.7 Simulation Methodology

We now turn our attention to the simulation of multipath fading channels. We will assume that either a discrete or diffused multipath model is specified and that the models are WSSUS. The distributions, delay profile, and the doppler spectrum, are assumed to be given. Furthermore, we will assume the fading to be Rayleigh or Ricean, with an emphasis on Rayleigh fading, since the Ricean model can be obtained from the Rayleigh model by adding a nonzero mean. We begin with the diffused multipath channel.

14.7.1 Simulation of Diffused Multipath Fading Channels

The diffused multipath channel is a linear time-varying system that is characterized by a continuous, rather than discrete, time-varying impulse response $\tilde{c}(\tau, t)$. The simulation model for an LTV system was derived in Chapter 13 and we repeat only the essential steps here. Since the lowpass input to the channel can be assumed to be bandlimited to a bandwidth B of the order $r/2$, where r is the symbol rate ($B \approx r$ for the bandpass case), we can represent the lowpass input in terms of its sampled values using the minimum sampling rate of r samples per second as

$$\tilde{x}(t-\tau) = \sum_{n=-\infty}^{\infty} \tilde{x}(t-nT)\frac{\sin(2\pi B(\tau - nT))}{2\pi B(\tau - nT)} \tag{14.49}$$

where $T = 1/r$ is the time between samples. Substituting the above representation of $\tilde{x}(t-\tau)$ in the convolution integral

$$\tilde{y}(t) = \int_{-\infty}^{\infty} \tilde{c}(\tau, t)\tilde{x}(t-\tau)\, d\tau \tag{14.50}$$

we obtain

$$\begin{aligned}\tilde{y}(\tau) &= \int_{-\infty}^{\infty} \tilde{c}(\tau, t)\left\{\sum_{n=-\infty}^{\infty} \tilde{x}(t-nT)\frac{\sin(2\pi B(\tau - nT))}{2\pi B(\tau - nT)}\right\} d\tau \\ &= \sum_{n=-\infty}^{\infty} \tilde{x}(t-nT)\int_{-\infty}^{\infty} \tilde{c}(\tau, t)\left\{\frac{\sin(2\pi B(\tau - nT))}{2\pi B(\tau - nT)}\right\} d\tau \end{aligned} \tag{14.51}$$

Thus

$$\tilde{y}(t) = \sum_{n=-\infty}^{\infty} \tilde{x}(t-nT)\tilde{g}_n(t) \tag{14.52}$$

where

$$\tilde{g}_n(t) = \int_{-\infty}^{\infty} \tilde{c}(\tau, t)\left\{\frac{\sin(2\pi B(\tau - nT))}{2\pi B(\tau - nT)}\right\} d\tau \tag{14.53}$$

Simulation models for diffused multipath fading channels are derived from (14.52) using two approximations. Truncating the sum in (14.52) so that only the terms for which $|n| \leq m$ are included and approximating the integral in (14.53) as

$$\tilde{g}_n(t) \approx T\,\tilde{c}(nT, t) \tag{14.54}$$

leads to the computationally efficient form

$$\begin{aligned}\tilde{y}(t) &= \sum_{n=-\infty}^{\infty} \tilde{x}(t-nT)\tilde{g}_n(t) \approx \sum_{n=-m}^{m} \tilde{x}(t-nT)\tilde{g}_n(t) \\ &\approx T\sum_{n=-m}^{m} \tilde{x}(t-nT)\tilde{c}(nT, t)\end{aligned} \tag{14.55}$$

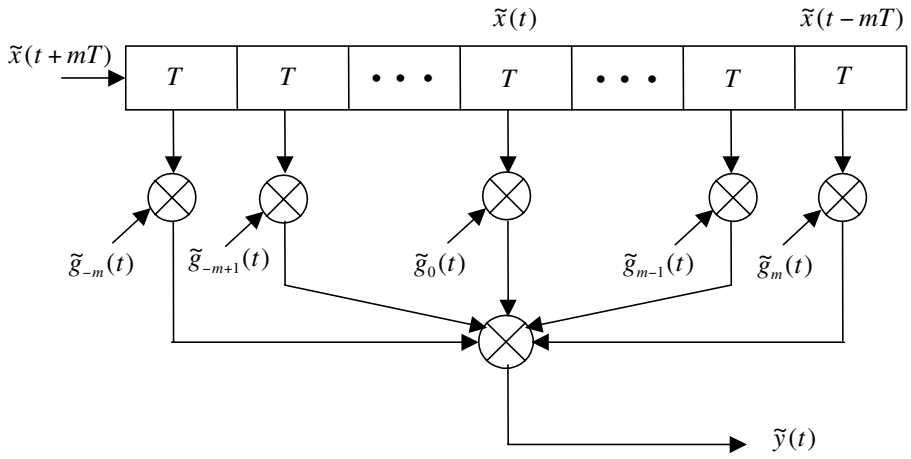


Figure 14.7 TDL model for a diffused multipath channel with $\tilde{g}n(t) = T\tilde{c}(nT, T)$.

Source: M. C. Jeruchim, P. Balaban, and K. S. Shanmugan, *Simulation of Communications Systems*, 2nd ed., New York: Kluwer Academic/Plenum Publishers, 2000.

Equation (14.55) can be implemented using a tapped delay line as shown in Figure 14.7.

For a Rayleigh fading channel, the tap gain processes $\tilde{g}_n(t) \approx T\,\tilde{c}(nT, t)$ are zero mean complex Gaussian processes. They will be uncorrelated because of the WSSUS assumption. The power spectral density of each tap gain process is specified by the doppler spectrum, and the variance σ_n^2 of the n^{th} tap gain process is given by

$$E\left\{|\tilde{g}_n(t)|^2\right\} \approx \sigma_n^2 = T^2 E\{|\tilde{c}(nT, t)|^2\} = T^2 p(nT) \tag{14.56}$$

and is obtained from the sampled values of the multipath intensity profile $p(\tau)$, an example of which is shown in Figure 14.8, where the total number of taps is $T_{\max}/T$.

Special Cases

If the channel is time invariant, then $\tilde{c}(\tau, t) = \tilde{c}(\tau)$, and the tap gains become constants. Therefore

$$\tilde{g}_n(t) = \tilde{g}_n \approx T\,\tilde{c}(nT) \tag{14.57}$$

In other words, the tap gains are sampled values of the impulse response of the LTIV system, and the tapped delay line model reduces to an FIR filter performing time-domain convolution. If the channel is frequency nonselective, then there is only one tap in the model, and $\tilde{y}(t) = \tilde{x}(t)\tilde{g}(t)$, where $\tilde{g}(t)$ is either a Rayleigh or Ricean process.

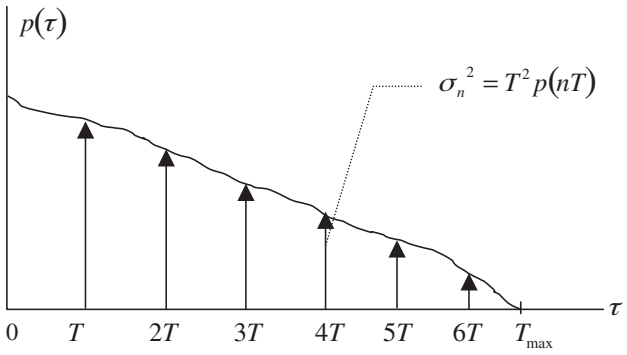


Figure 14.8 Sampled values of the power delay profile.

Source: M. C. Jeruchim, P. Balaban, and K. S. Shanmugan, *Simulation of Communications Systems*, 2nd ed., New York: Kluwer Academic/Plenum Publishers, 2000.

Sampling

An important aspect of the TDL model that deserves additional attention is the sampling rate for simulations. The TDL model shown in Figure 6.8 was derived with continuous time input $\tilde{x}(t)$ and output $\tilde{y}(t)$. However, in simulation we use sampled values of $\tilde{x}(t)$ and output $\tilde{y}(t)$ which should be sampled at 8 to 32 times the bandwidth, where the bandwidth includes the effect of spreading due to the time-varying nature of the system as defined in Chapter 13. Note that the Nyquist rate of $2B$, $B = r/2$ was used to derive the TDL model, and the tap spacing of $T = 1/r$ will be much greater than T_s, where T_s is the sampling time for the input and output waveforms. It is of course possible to derive a TDL model with a smaller tap spacing (i.e., more samples per symbols), but such a model will be computationally inefficient and does not necessarily improve the accuracy of the simulation.

Generation of Tap Gain Processes

The tap gain processes are stationary random processes with Gaussian probability density functions and arbitrary power spectral density functions. The simplest model for the tap gain processes assume them to be uncorrelated, complex, zero mean Gaussian processes with different variances but identical power spectral densities. In this case, the tap gain processes can be generated by filtering white Gaussian processes, as shown in Figure 14.9.

The filter transfer function is chosen such that it produces the desired doppler power spectral density. In other words, $H(f)$ is chosen such that

$$S_{\tilde{g}\tilde{g}}(f) = S_d(f) = S_{\tilde{w}\tilde{w}}(f) \left|\tilde{H}(f)\right|^2 = \left|\tilde{H}(f)\right|^2 \tag{14.58}$$

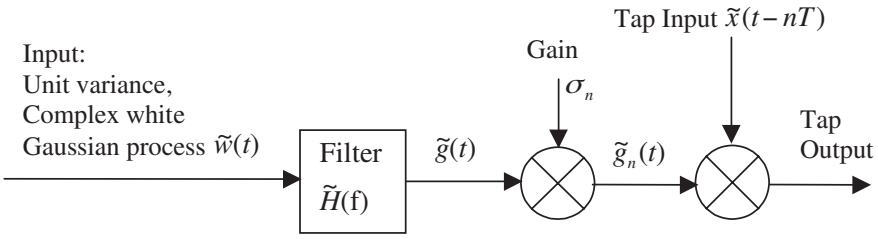


Figure 14.9 Generation of the n^{th} tap gain process.

Source: M. C. Jeruchim, P. Balaban, and K. S. Shanmugan, *Simulation of Communications Systems*, 2nd ed., New York: Kluwer Academic/Plenum Publishers, 2000.

where $S_{\tilde{w}\tilde{w}}(f)$ is the power spectral density of the input white noise process, which can be set equal to 1, and $S_{\tilde{g}\tilde{g}}(f)$ is the specified doppler power spectral density of the tap gain processes. The filter gain is chosen such that $\tilde{g}(t)$ has a normalized power of 1. The static gain σ_n in Figure 14.9 accounts for the different power levels or variances for the different taps. If the power spectral density of the tap gains are different, then different filters will be used for different taps.

Delay Power Profiles and Doppler Power Spectral Densities

As previously mentioned, the BER performance of a communication system is more sensitive to the values of the rms and maximum delay spreads than to the shape of the power delay profile. Therefore, simple profiles such as uniform or exponential can be used for simulation. The delay profiles are normalized to have unit area (i.e., total normalized power, or the area under the locally averaged power delay profile, is set equal to one). Thus

$$\int_0^{T_m} p(\tau)dt = 1 \tag{14.59}$$

Typical rms delay spreads are given in Table 14.2.

The most commonly used models for doppler power spectral densities for mobile applications assume that there are many multipath components, each having different delays, and that all components have the same doppler spectrum. Each

Table 14.2 Typical rms Delay Spreads

Link Type	Link Distance	rms Delay Spread
Troposcatter	100 Km	milliseconds (10^{-3})
Outdoor Mobile	1 Km	microseconds (10^{-6})
Indoor	10 m	nanoseconds (10^{-9})

multipath component (ray) is actually made up of a large number of simultaneously arriving unresolvable multipath components, having angle of arrival with a uniform angular distribution at the receive antenna. This channel model was used by Jakes and others at Bell Laboratories to derive the first comprehensive mobile radio channel model for both doppler effects and amplitude fading effects [11]. The classical Jakes' doppler spectrum has the form, which was initially simulated in Chapter 7 (see Example 7.11),

$$S_d(f) = S_{\tilde{g}_n \tilde{g}_n}(f) = \frac{K}{\sqrt{1-(f/f_d)^2}}, \qquad -f_d \le f \le f_d \tag{14.60}$$

where $f_d = v/\lambda$ is the maximum doppler shift, v is the vehicle speed in meters per second, and λ is the wavelength of the carrier. While the doppler spectrum defined by (14.60) is appropriate for dense scattering environments like urban areas, a "Ricean spectrum" is recommended for rural environments in which there is one strong direct line-of-sight path and hence Ricean fading. The Ricean doppler spectrum has the form

$$S_d(f) = S_{\tilde{g}_n \tilde{g}_n}(f) = \frac{0.41}{\sqrt{1-(f/f_d)^2}} + 0.91\delta(f \pm 0.7 f_d), \qquad -f_d \le f \le f_d \tag{14.61}$$

and is shown in Figure 14.10. Other spectral shapes used for the doppler power spectral densities include Gaussian and uniform. Typical doppler bandwidths in mobile applications at 1 GHz will range from 10 to 200 Hz.

There are several ways of implementing the doppler spectral shaping filter needed to generate the tap gain processes in the TDL model for the channel when using the model assumed by Jakes. An FIR filter in time domain is the most common implementation, since doppler power spectral densities do not lend themselves easily to implementation in recursive form. The generation of a Jakes spectrum using FIR filtering techniques was illustrated in Chapter 7. A block processing model based on frequency domain techniques is discussed in [13].

In generating the tap gain processes it should be noted that the bandwidth of the tap gain processes for slowly time-varying channels will be very small compared

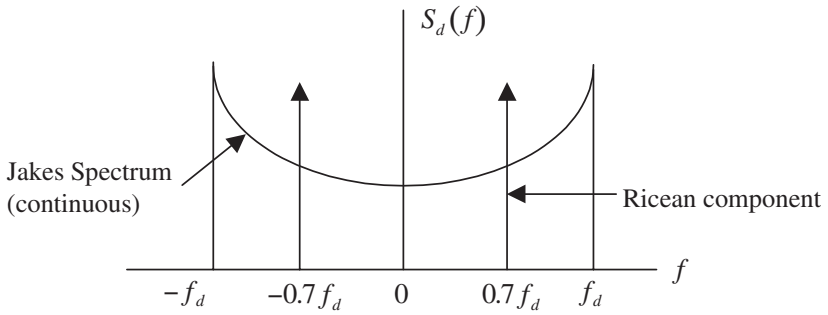


Figure 14.10 Example of doppler power spectral densities.

to the bandwidth of the signals that flow through them. In this case, the tap gain filter should be designed and executed at a slower sampling rate. Interpolation can be used at the output of the filter to produce denser samples at a rate consistent with the sampling rate of the signal coming into the tap. Designing the filter at the higher rate will lead to computational inefficiencies as well as stability problems.

Correlated Tap Gain Model

The approximation of the tap gain processes given in (14.53) and (14.54) by

$$\widetilde{g}_n(t) = \int_{-\infty}^{\infty} \widetilde{c}(\tau, t) \left\{ \frac{\sin(2\pi B(\tau - nT))}{2\pi B(\tau - nT)} \right\} d\tau \approx T\widetilde{c}(nT, t) \tag{14.62}$$

leads to uncorrelated tap gain functions. Without the approximation, the tap gain functions will be correlated. It can be shown that the correlation between $g_n(t)$ and $g_m(t)$ is given by

$$\begin{aligned} R_{m,n}(\eta) &= E\left\{g_m^*(t) g_n(t+\eta)\right\} \\ &= \int R_{\tilde{c}\tilde{c}}(\tau, \eta) \operatorname{sinc}(2B\tau - m) \operatorname{sinc}(2B\tau - n) d\tau \end{aligned} \tag{14.63}$$

where $T = 1/2B$ is the tap spacing.

Generating a set of correlated random processes with arbitrary power spectral density functions is very difficult. An approximation that simplifies this problem somewhat makes the reasonable assumption that all tap gain functions have the same power spectral density. Therefore, we assume that

$$S(\tau, \lambda) = m(\tau) S_d(\lambda) \tag{14.64}$$

where $S(\tau, \lambda)$ is the scattering function, $m(\tau)$ is the normalized power delay power profile, and $S_d(\lambda)$ is the doppler spectrum. The solution to this case may be found in [17].

An approach to solving the general problem has been recently proposed [18]. This method is based on fitting a vector ARMA model to the tap gain processes and deriving the vector ARMA model from the given correlations and power spectral densities. The procedure for fitting the vector ARMA model is very complex, and it is not clear whether the extra work required can be justified in terms of the improvement in accuracy.

14.7.2 Simulation of Discrete Multipath Fading Channels

Compared to the diffused multipath model, simulation of the discrete multipath model is rather straightforward, at least conceptually. We must keep in mind that since the channel is dynamic in both space and time, care must be used to avoid aliasing [19]. The input-output relationship of a discrete multipath model is given by

$$\tilde{y}(t) = \sum_{k=1}^{N(t)} \widetilde{a}_k(t) \widetilde{x}(t - \tau_k(t)) \tag{14.65}$$

where $\widetilde{a}_k(t)$ is the complex path attenuation as discussed in Section 14.4. In (14.65) it can be assumed that the number of multipath components and the delay structure will vary slowly compared to the variations in $\widetilde{a}_k(t)$. Hence the delays $\tau_k(t)$ can be treated as constants over the duration of a simulation, and the preceding equation can be written as

$$\tilde{y}(t) = \sum_{k=1}^{N(t)} \widetilde{a}_k(t)\, \widetilde{x}(t - \tau_k) \tag{14.66}$$

and implemented in block diagram form as shown in Figure 14.11.

In order to illustrate the basic approach for simulating discrete channel models we assume that the model is specified in terms of probability distributions for the number of components N, the delays, and the complex attenuations as a function of the delays. A representation (snapshot) of the channel is then obtained as follows:

1. Draw a random number N to obtain the number of delays.
2. Draw a set of N random numbers from the distribution for delay values.
3. Draw a set of N attenuations based on the delay values.

This set of $3N$ random numbers represents a snapshot of the channel, which is implemented as shown in Figure 14.11. In Figure 14.11 the initial delay is $\Delta_1 = \tau_1$. The remaining delays Δ_n, $2 \le n \le N$, are differential delays defined by

$$\Delta_n = \tau_n - \tau_{n-1}, \qquad 2 \le n \le N \tag{14.67}$$

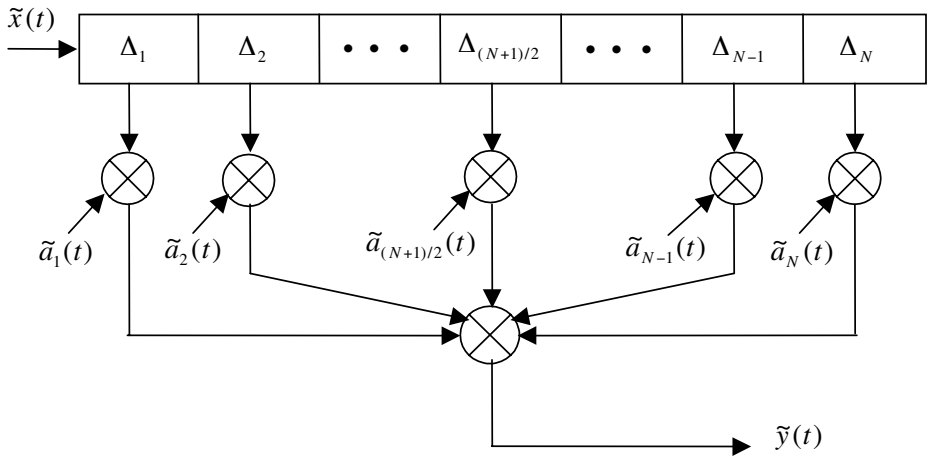


Figure 14.11 A variable delay TDL model for discrete multipath channels.

Source: M. C. Jeruchim, P. Balaban, and K. S. Shanmugan, *Simulation of Communications Systems*, 2nd ed., New York: Kluwer Academic/Plenum Publishers, 2000.

While the implementation shown in Figure 14.11 is rather straightforward, it poses a problem when the delays differ by very small time offsets. Since everything will be sampled, the tap spacings (i.e., the differential delays $\tau_n - \tau_{n-1}$) must be expressed in terms of an integer number of sampling periods for simulation. Hence, the sample time must be very small, smaller than the smallest differential delay. This might lead to excessive sampling rates and an unacceptable computational burden. We can avoid this problem by developing a TDL model with uniform tap spacing following the approach used in the simulation of diffused multipath channels in Section 14.7.1.

Uniformly Spaced TDL Model for Discrete Multipath Fading Channels

The tap gains of a uniformly spaced TDL model are given in (14.53) as

$$\tilde{g}_n(t) = \int_{-\infty}^{\infty} \tilde{c}(\tau, t) \left\{ \frac{\sin(2\pi B(\tau - nT))}{2\pi B(\tau - nT)} \right\} d\tau \tag{14.68}$$

Substituting the impulse response of the discrete multipath channel, given by

$$\tilde{c}(\tau, t) = \sum_{k=1}^{N} \widetilde{a}_k(t)\, \delta(\tau - \tau_k) \tag{14.69}$$

in the preceding equation, we obtain the tap gains as

$$\tilde{g}_n(t) = \sum_{k=1}^{N} \widetilde{a}_k(t) \operatorname{sinc}\left(\frac{\tau_k}{T} - n\right) = \sum_{k=1}^{N} \widetilde{a}_k(t)\ \alpha(k, n) \tag{14.70}$$

In (14.70)

$$\alpha(k, n) = \operatorname{sinc}\left(\frac{\tau_k}{T} - n\right) \tag{14.71}$$

Note that the envelope of $\alpha(k, n)$ decreases as $|n|$ increases. Hence the number of taps can be truncated to $|n| \leq m$, where m is chosen to satisfy $m \gg T_{\max}T$. For the case where the maximum delay spread $T_{\max}$ will not exceed 3 or 4 symbol times, the number of taps need not be greater than about 20 ($-m < n < m$, $m = 10$). The model now takes the form previously derived for the approximate diffused multipath model illustrated in Figure 14.7.

The generation of the tap gains is illustrated in Figure 14.12. Note that the generation of the tap gain processes for the discrete multipath model is straightforward compared to the generation of the tap gain processes for the diffused case. We start with a set of N independent, zero-mean complex Gaussian white noise processes, which are filtered to produce the appropriate doppler spectrum. These are then scaled to produce the desired power profile, and are finally transformed according to (14.70) to produce the tap gain processes. (Note that only two of the N paths are shown in Figure 14.12.)

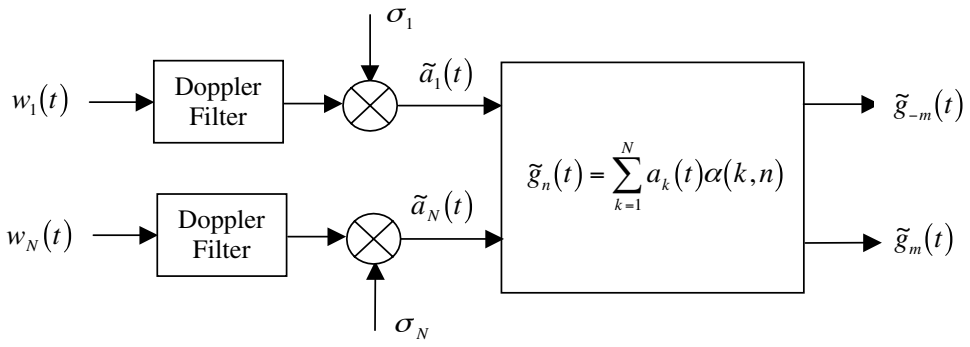


Figure 14.12 Generation of the tap gain processes.

Source: M. C. Jeruchim, P. Balaban, and K. S. Shanmugan, *Simulation of Communications Systems*, 2nd ed., New York: Kluwer Academic/Plenum Publishers, 2000.

To illustrate the calculation of the tap gain functions let us assume that

$$\Delta\tau = \frac{\tau_2 - \tau_1}{T} = 0.5 \tag{14.72}$$

The tap gain functions in this case are obtained by filtering two uncorrelated white Gaussian noise processes and then transforming them to tap gain processes according to (14.70) as

$$\begin{bmatrix} \widetilde{g}_{-4}(t) \\ \widetilde{g}_{-3}(t) \\ \widetilde{g}_{-2}(t) \\ \widetilde{g}_{-1}(t) \\ \widetilde{g}_0(t) \\ \widetilde{g}_1(t) \\ \widetilde{g}_2(t) \\ \widetilde{g}_3(t) \\ \widetilde{g}_4(t) \end{bmatrix} = \begin{bmatrix} \operatorname{sinc}(0.0+4) & \operatorname{sinc}(0.5+3) \\ \operatorname{sinc}(0.0+3) & \operatorname{sinc}(0.5+3) \\ \operatorname{sinc}(0.0+2) & \operatorname{sinc}(0.5+2) \\ \operatorname{sinc}(0.0+1) & \operatorname{sinc}(0.5+1) \\ \operatorname{sinc}(0.0) & \operatorname{sinc}(0.5) \\ \operatorname{sinc}(0.0-1) & \operatorname{sinc}(0.5-1) \\ \operatorname{sinc}(0.0-2) & \operatorname{sinc}(0.5-2) \\ \operatorname{sinc}(0.0-3) & \operatorname{sinc}(0.5-3) \\ \operatorname{sinc}(0.0-4) & \operatorname{sinc}(0.5-4) \end{bmatrix} \begin{bmatrix} \widetilde{a}_1(t) \\ \widetilde{a}_2(t) \end{bmatrix} \tag{14.73}$$

which is

$$\begin{bmatrix} \widetilde{g}_{-4}(t) \\ \widetilde{g}_{-3}(t) \\ \widetilde{g}_{-2}(t) \\ \widetilde{g}_{-1}(t) \\ \widetilde{g}_0(t) \\ \widetilde{g}_1(t) \\ \widetilde{g}_2(t) \\ \widetilde{g}_3(t) \\ \widetilde{g}_4(t) \end{bmatrix} = \begin{bmatrix} 0.0 & 0.0707 \\ 0.0 & -0.0910 \\ 0.0 & 0.1273 \\ 0.0 & -0.2122 \\ 1.0 & 0.6366 \\ 0.0 & 0.6366 \\ 0.0 & -0.2122 \\ 0.0 & 0.1273 \\ 0.0 & -0.0909 \end{bmatrix} \begin{bmatrix} \widetilde{a}_1(t) \\ \widetilde{a}_2(t) \end{bmatrix} \tag{14.74}$$

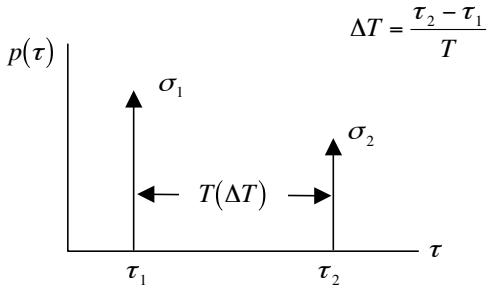


Figure 14.13 Simple two-ray model. (Note that $\Delta\tau$ is normalized).

Source: M. C. Jeruchim, P. Balaban, and K. S. Shanmugan, *Simulation of Communications Systems*, 2nd ed., New York: Kluwer Academic/Plenum Publishers, 2000.

where $\widetilde{a}_1$ and $\widetilde{a}_2$ are defined in Figure 14.12. The preceding equation shows the coefficients of the transformation for only 9 taps. We see that these coefficients will be negligible for higher-order tap gains and, as a result, they can be ignored. The TDL model is therefore truncated to 9 taps.

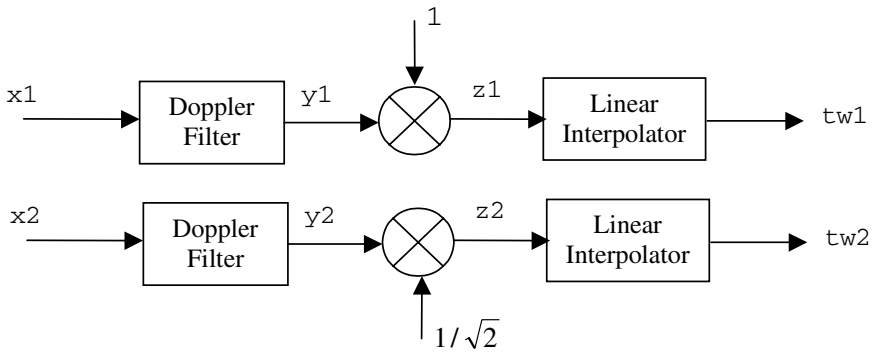
A simple two-ray model is often used to make preliminary performance predictions for fading channels. Consider the power-delay profile illustrated in Figure 14.13. Parametric performance predictions can be made by varying the ratio of the normalized delay spread $\Delta\tau = (\tau_2 - \tau_1)/T$, where T is the symbol duration and the ratio of relative powers in the two paths $(\sigma_1/\sigma_2)^2$. If $\Delta\tau \ll 0.1$, then the two paths can be combined and the model can be treated as frequency nonselective. If $\Delta\tau > 0.1$, there will be considerable intersymbol interference in the channel and it is treated as frequency selective.

Example 14.2. In this example we consider the effect of fading due to doppler on the transmission of a QPSK signal on a discrete multipath channel. The block diagram is illustrated in Figure 14.14. The generation of the tap weights is shown in Figure 14.14(a). The doppler filter is realized using the Jakes model defined by (14.60) with $K = 1$ and $f_d = 100$ Hz. The tap gain processes are uncorrelated and Gaussian. The tap spacing is based on an RF bandwidth of 20 kHz (lowpass equivalent bandwidth of 10 kHz). The tap weights are denoted `tw1` and `tw2`. The complex signal is multiplied by the complex tap weights. Both the complex QPSK signal and the complex carrier are used as inputs. The carrier is defined by

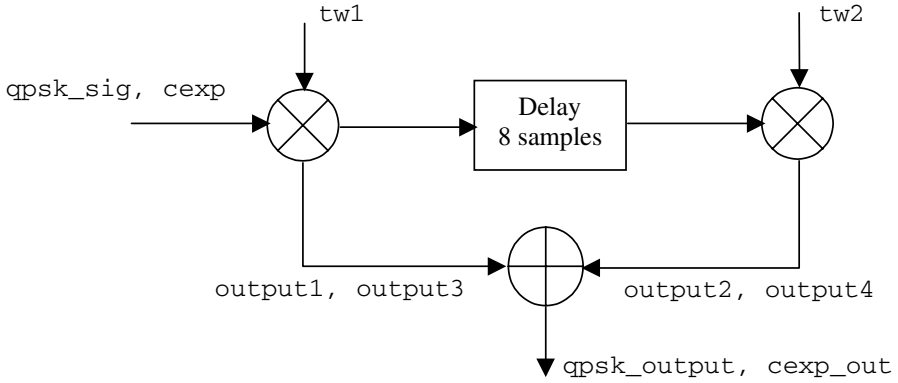
$$c(t) = \exp\left[j2\pi(1000)t\right] \tag{14.75}$$

The delay of 8 samples corresponds to one-half of the symbol time. Additional details are included in the MATLAB code for this example, which is given in Appendix B.

The simulation length is determined from a number of considerations. In order to observe the spectra of the input and output for the complex exponential case,



(a) Generation of tap weights.



(b) Processing of QPSK signal and carrier.

Figure 14.14 Block diagrams of simulated systems.

10 to 20 cycles of the complex exponential are needed. At the same time, in order to capture the effects of the time-varying channel, we need to simulate the fading process for about 5 to 10 times the reciprocal of the doppler bandwidth. These two considerations lead to a simulation length of 1/20 second, or about 8,000 samples.

Executing the MATLAB program given in Appendix B generates the results illustrated in Figures 14.15, 14.16, and 14.17. The input and output carrier signals are shown in Figure 14.15. The input (top pane) is the tone at 1,000 Hz. The output (bottom pane) illustrates the spectral spreading due to doppler. The direct channel input and output are illustrated in Figure 14.16. The input signal (top pane) has two levels as expected. The output signal (bottom pane) has more than

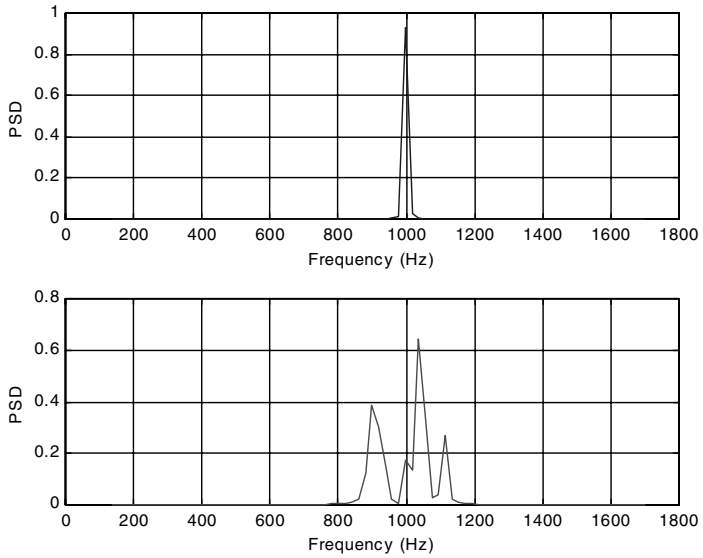


Figure 14.15 Input (top pane) and output (bottom pane) power spectral densities. The spectral spreading due to doppler is evident in the bottom pane.

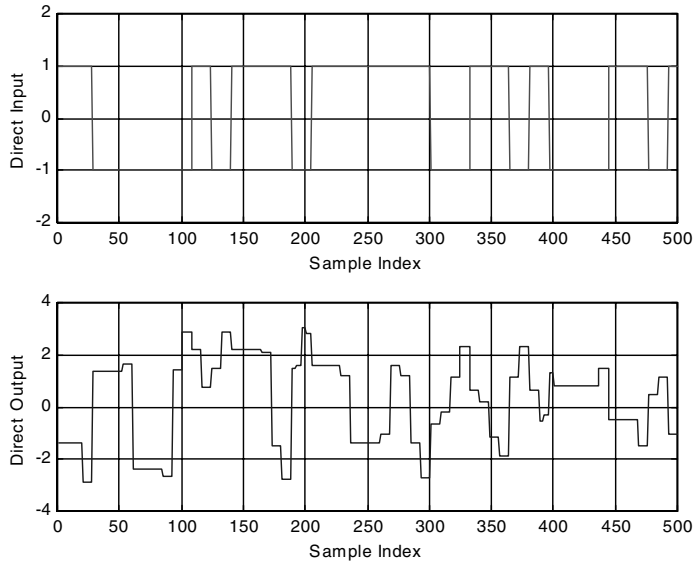


Figure 14.16 Direct channel QPSK input and output.

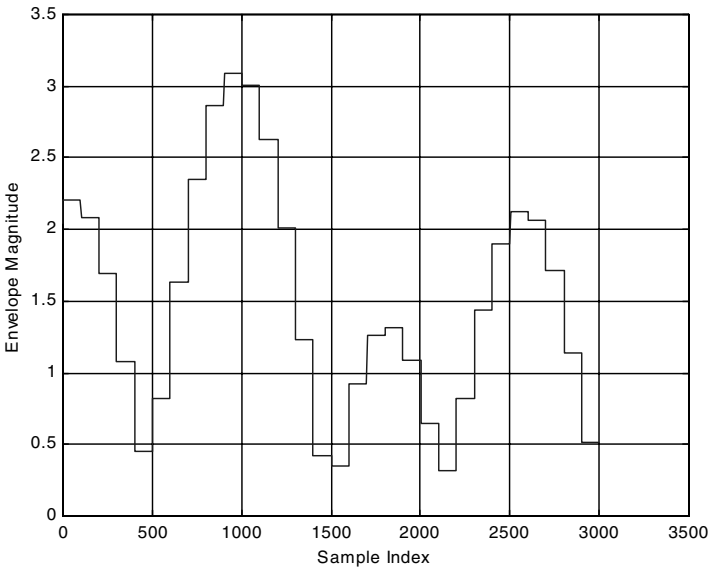


Figure 14.17 Envelope of the complex exponential output.

two levels because of intersymbol interference. The envelope of the QPSK output signal is illustrated in Figure 14.17. ■

14.7.3 Examples of Discrete Multipath Fading Channel Models

In this section we present a number of examples of discrete multipath models that are used to simulate the performance of wireless communication systems. The first model that we present is the so-called Rummler's model for terrestrial microwave communication links between fixed antenna towers. This is a line-of-sight radio channel with a very small number of multipath components because of the larger directional antennas used in the system and the very benign properties of the tropospheric channel used by LOS microwave radio. Larger antennas mean that the field of view of the antenna is limited at very small angles of arrival which yields a smaller number of multipath components. Also, since the antennas are fixed, the only time variations in the channel characteristics are due to changes in the atmospheric conditions. These variations can be considered very slow compared to the channel bandwidths which will be of the order of tens of MHz. Hence Rummler's model is a multipath model with very slow fading.

The second set of examples that we present are for mobile radio channels. These channels typically have a larger number of multipath components because of the use of omnidirectional antennas which pick up a large number of reflections with widely varying propagation delays, especially in urban areas. They will also experience faster fading due to the the large number of multipath components that experience

large carrier phase shifts over small distance changes and thus can combine destructively or constructively over small distances.

Rummler's Model for LOS Terrestrial Microwave Channels

One of the most widely used models for terrestrial microwave links operating in the frequency range of $4-6$ GHz between fixed towers, was developed by Rummler [20]. This model is based on a set of assumptions, and measured data is used to obtain numerical values of the model parameters. Given the geometry of the link and antenna parameters, Rummler hypothesized a three-ray model of the form

$$y(t) = x(t) + \alpha\, x(t-\tau_1) + \beta\, x(t-\tau_2) \tag{14.76}$$

where $x(t)$ and $y(t)$ are the bandpass input and output, respectively. In terms of the complex envelopes, the model takes the form

$$\tilde{y}(t) = \tilde{x}(t) + \alpha\, \exp(-j2\pi f_c\tau_1)\tilde{x}(t-\tau_1) + \beta \exp(-j2\pi f_c\tau_2)\, \tilde{x}(t-\tau_2) \tag{14.77}$$

and the lowpass equivalent transfer function of Rummler's channel is given by

$$H(f) = 1 + \alpha \exp(-j2\pi(f_c-f)\tau_1) + \beta \exp(-j2\pi(f_c-f)\tau_2) \tag{14.78}$$

The first simplification of the model is based on the assumption that over the bandwidth of interest $(f_c - f)\,\tau_1 << 1$, and hence $\exp(-j2\pi(f_c-f)\tau_1) \approx 1$ and

$$H(f) \approx 1 + \alpha + \beta \exp(-j2\pi(f_c-f)\tau_2) \tag{14.79}$$

The next step is to assume that the "notch" frequency, where the magnitude of the response is minimum, is $f_c + f_0$ in the bandpass case, and at f_0 in the lowpass model, so that the final form of the lowpass equivalent transfer function can be written as

$$H(f) \approx a[1 - b \exp(-j2\pi(f_0-f)\tau_2)] \tag{14.80}$$

where $a = 1+\alpha$ is the overall attenuation, and $b = -\beta/\,(1+\alpha)$ is a shape parameter. The value of the delay parameter τ_2, chosen to fit the measured data, has a value of $\tau_2 = \tau = 6.3$ ns. Note that this small time delay is only on the order of 2 meters of propagation delay, which is physically plausable and corresponds to typical refractive path differences observed over tropospheric channels for LOS microwave radio at 2–6 GHz.

The amplitude response of the Rummler model is

$$|H(f)|^2 = a^2[1 + b^2 - 2b\cos(2\pi(f-f_o)\tau)] \tag{14.81}$$

and an example of the magnitude response is shown in Figure 14.18.

The parameters a and b are normalized and expressed in dB units as illustrated in Table 14.3. Analysis of channel data yields exponential distributions for B_1

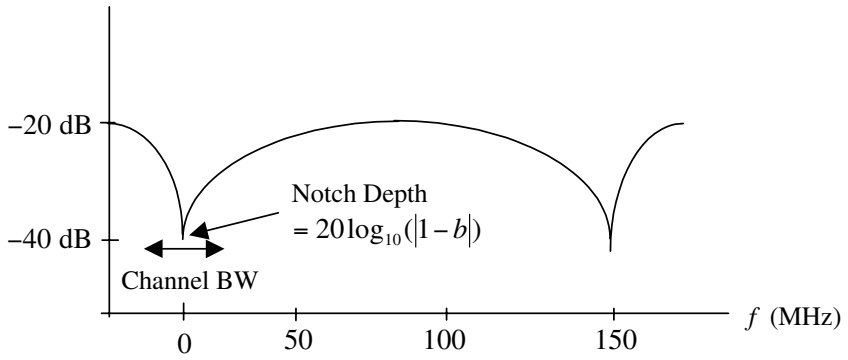


Figure 14.18 Example of the magnitude response of the Rummler channel.

Table 14.3 Parameters for Rummler Model

Minimum Phase Case $(b < 1)$	Nonminimum Phase Case $(b > 1)$
$A_1 = 20\log_{10}(a)$	$A_2 = -20\log_{10}(ab)$
$B_1 = 20\log_{10}(1-b)$	$B_2 = 10\log_{10}(1-1/b)$

and B_2 with means of 3.8 dB. Likewise, A_1 and A_2 are Gaussian with standard deviations of 5 dB. The means are

$$\mu = 24.6\left(\frac{B^4+500}{B^4+800}\right)$$

where $B = B_1$ for A_1 and $B = B_2$ for A_2. The probability density function of $\theta = 2\pi f_0 \tau$ is shown in Figure 14.19.

In order to simulate a snapshot of the Rummler channel, we draw the following

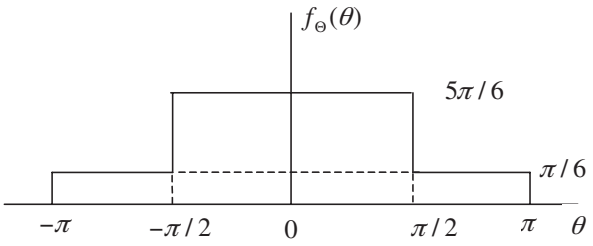


Figure 14.19 Probability density function of θ.

set of random numbers:

1. Draw a number U uniformly distributed in [0,1]. If $U > 0.5$, assume minimum phase. If $U < 0.5$, assume nonminimum phase. (Minimum phase and nonminimum phase fades are assumed equally likely.)

2. Draw an exponentially distributed random number for B_1 or B_2.

3. Draw a Gaussian random number for A_1 or A_2 using the value of B_1 or B_2.

4. Draw a random number for θ and set the notch frequency at $f_0 = \theta/2\pi\tau$, $\tau = 6.3ns$.

These parameters define a snapshot of the Rummler channel. Since the channel is assumed to be slowly varying with respect to the symbol rate, a series of snapshots of the channel is adequate for performance evaluation using a Monte Carlo simulation for each snapshot produced by the model.

Models for Mobile Channels

Discrete channel models are also widely used for indoor and outdoor wireless channels. Many models are based on emperical data collected over a wide range of environments [21, 22, 23]. Given the large number of both mathematical and empirical models that have been proposed recently, the designer of a communication system is faced with the difficult problem of choosing a representative set of channel models that will represent the channels over which the communication system is to operate satisfactorily. Fortunately, some guidance on the choice of which models to use has been provided by international standards bodies that specify a set of "representative" channels for analyzing and simulating the performance of different types of communication systems. We present two examples below.

Discrete Channel Models for GSM Applications The Global System for Mobile Communications (GSM) is a standard for mobile communications in the frequency band from 1 to 2 GHz and uses 200 kHz RF channels for time-division multiplexed communications [13, 24]. The symbol time in GSM is of the order of a few microseconds.

The recommended GSM models are discrete models consisting of 12 rays (paths), and are specified for three different scenarios: rural, hilly, and urban. For each scenario, two models are specified. In addition to the 12-ray models, a simpler set of models with 6 rays (paths) are also defined. The 12-ray and 6-ray models for urban areas are given in Table 14.4 and Table 14.5, respectively. In addition to these models, there is also a model specified for testing the performance of the Viterbi equalizer used in GSM systems. This model is given in Table 14.6. All of the relative powers are in dB, and (1) and (2) designate the two equivalent models.

It should be noted that the symbol time in the system is of the order of a few microseconds, and some of the differential delays are of the order of $0.1\mu s$, which means that a sampling rate of 10 M samples/sec should be used in order to represent these small delays. Another approach, as outlined in the preceding

Table 14.4 Typical Profile for Urban Areas (12-ray model)

Ray	Relative Time (1) s	Relative Time (2) s	Average Power (1) dB	Average Power (2) dB	Dopp. Spect.
1	0.0	0.0	-4.0	-4.0	Jakes
2	0.1	0.2	-3.0	-3.0	Jakes
3	0.3	0.4	0.0	0.0	Jakes
4	0.5	0.6	-2.6	-2.0	Jakes
5	0.8	0.8	-3.0	-3.0	Jakes
6	1.1	1.2	-5.0	-5.0	Jakes
7	1.3	1.4	-7.0	-7.0	Jakes
8	1.7	1.8	-5.0	-5.0	Jakes
9	2.3	2.4	-6.5	-6.0	Jakes
10	3.1	3.0	-8.6	-9.0	Jakes
11	3.2	3.2	-11.0	-11.0	Jakes
12	5.0	5.0	-10.0	-10.0	Jakes

Table 14.5 Reduced Profile for Urban Areas (6-ray model)

Ray	Relative Time (1) s	Relative Time (2) s	Average Power (1) dB	Average Power (2) dB	Dopp. Spect.
1	0.0s	0.0s	-3.0	-3.0	Jakes
2	0.2	0.2	0.5	0.0	Jakes
3	0.5	0.6	-2.0	-2.0	Jakes
4	1.6	1.6	-6.0	-6.0	Jakes
5	2.3	2.4	-8.0	-8.0	Jakes
6	5.0	5.0	-10.0	-10.0	Jakes

Table 14.6 Profile for Equalization Test

Ray	Relative Time	Average Power	Doppler Spectrum
1	0.0s	0.0dB	Jakes
2	3.2	0.0	Jakes
3	6.4	0.0	Jakes
4	9.6	0.0	Jakes
5	12.8	0.0	Jakes
6	16.0	0.0	Jakes

Table 14.7 Parameters for a 3-Ray Outdoor Model for PCS

Environment	τ_1 (ns)	τ_2 (ns)	τ_3 (ns)	Doppler	Doppler BW
Pedestrian	0	1,500	14,500	Flat	12Hz
Wireless Loop	0	1,500	14,500	Gaussian	12Hz
Vehicular	0	1500	15,500	Jakes	180Hz

Table 14.8 Ray Strengths

Ray	Power(dB)
1	0
2	-3
3	-6

Table 14.9 Parameters of PCS Indoor Model

Environment	Tap Spacing	Number of Taps	Doppler Spectrum	Doppler BW
Residential	50(ns)	2	Gaussian	3Hz
Office	50	4	Gaussian	3
Commercial	50	12	Flat	30

section, uses a symbol time spaced TDL (correlated tap gain functions) to reduce the computational load.

Discrete Models for PCS Applications For PCS communication systems operating in the 2-GHz band, the standards bodies have agreed on a set of discrete models for typical operating environments [25]. These models are summarized in Tables 14.7, 14.8, and 14.9. For the model given in Table 14.7, the ray strengths $E\left\{|\tilde{a}|^2\right\}$ are given in Table 14.8.

It should be noted again that the differential delays of 50 ns (indoor model) are very small compared to the symbol time of proposed PCS systems. Hence, either the fading should be treated as frequency nonselective, or a bandlimited TDL model with symbol time spacing should be used for simulations.

Discrete Multipath Channel Models for 3G Wideband CDMA Systems Cellular communication systems are in their third generation of evolution, and the third generation systems (3G) will use wideband CDMA operating around 2 GHz. Examples of the discrete channel models proposed for 3G systems are shown in Table 14.10 (Case 1: Indoor, Case 2: Indoor or Pedestrian, Case 3: Vehicular) [26].

Table 14.10 Parameters for 3G Wideband CDMA Channels

Case 1	(3 km/h)	Case 2	(3 km/h)	Case 3	(120 km/h)
Delay (ns)	Power (dB)	Delay (ns)	Power (dB)	Delay (ns)	Power (dB)
0	0.0	0	0.0	0	0.0
244	-9.6	244	-12.5	244	-2.4
488	-35.5	488	-24.7	488	-6.5
				732	-9.4
				936	-12.7
				1220	-13.3
				1708	-15.4
				1953	-25.4

14.7.4 Models for Indoor Wireless Channels

Fading characteristics of indoor wireless channels are very different from those of vehicular channels due to differences in physical environments (dimensions, materials, etc.) and propagation mechanisms. Outdoor vehicular environments are characterized by larger cells of the order of kilometers and a smaller number of multipath components. Indoor environments, on the other hand, are characterized by smaller dimensions (tens of meters) and a large number of multipath components due to reflections from walls, tables, and other flat work surfaces. There are a number of statistical models for indoor channels derived from measurements and, by and large, the indoor models can be categorized as dense discrete multipath models with an rms delay spread in the range of 30 to 300 ns with each component having Ricean envelope statistics [23]. The path loss index typically varies from 1.8 to 4. Additional details of the indoor channel characteristics may be found in the references [27–31]. The simulation techniques for indoor channels are the same as those we have seen for other multipath channels. However, the small differential delays encountered in indoor situations might require the conversion of nonuniformly spaced TDL models to uniformly spaced models as discussed in Section 14.7.2.

14.8 Summary

The overall performance of a communication system is significantly impacted by the distortion, noise, and interference introduced by the communication channels over which they operate. To assess communication system performance, and to design and optimize the signal-processing operations in the transmitter and receiver, we need simulation models for communication channels.

The simplest simulation model for a communication channel is the transfer function model, which can be used for time-invariant communication channels such as optical fibers and electrical cables. Radio communication channels, on the other hand, require more complex models to account for the multipath effect and the time variations (fading) in the channel characteristics, especially in mobile channels.

The simulation model for multipath fading channels has the structural form of a tapped delay line with time-varying tap gains, which are modeled as stationary random processes over observation intervals for which the stationarity assumption applies. For mobile applications, fading in the communication channel is characterized by complex Gaussian processes with appropriate power spectral density functions. Sampled values of tap gain processes in the tapped delay line model are generated by filtering uncorrelated Gaussian sequences with FIR filters which shape the power spectral densities.

For most applications, the tap gains can be assumed to be uncorrelated. However, in some simulation cases, the tap gain processes in the simulation models will be correlated. Generating a set of correlated tap gain processes is, in general, a difficult problem. If the processes involved are Gaussian and have the same power spectral densities, this problem is easily handled.

The literature on measurements of mobile and other radio channels is vast and varied. For simulation purposes we often rely on statistical models derived from measurements. Many examples of the models used for designing and evaluating the performance of second- and third-generation mobile communication systems were presented in this chapter. The reader can find additional models and details in the references.

14.9 Further Reading

A vast amount of material has been published on the characterization and modeling of wireless channels and only the most fundamental material is included in this chapter. Almost every issue of the *IEEE Transactions on Wireless Communications*, the *IEEE Transactions on Communications*, and the *IEEE Transactions on Antennas and Propagation* contain new research results in this area. Good collections of papers are given in the double issue of the *IEEE Journal on Selected Areas in Communications* cited below.

L. J. Greenstein et al., eds., "Channel and Propagation Models for Wireless System Design I and II," *IEEE Journal on Selected Areas in Communications*, Vol. 20, Nos. 3 and 6, April 2002 and August 2002.

The interested student is also referred to the recent book

H. L. Bertoni, *Radio Propagation for Modern Wireless Systems*, Upper Saddle River, NJ: Prentice Hall PTR, 2000.

14.10 References

1. A. F. Elrefaie, J. K. Townsend, M. B. Romeiser, and K. S. Shanmugan, "Computer Simulation of Digital Lightwave Links," *IEEE Journal on Selected Areas in Communications*, Vol. 6, No. 1, January 1984, pp. 94–106.

2. D. G. Duff, "Computer-Aided Design of Digital Lightwave Systems," *IEEE Journal on Selected Areas in Communications,* Vol. 2, No. 1, January 1984, pp. 171–185.

3. P. K. Cheo, *Fiber Optic Devices and Systems*, New York: Prentice Hall, 1985.

4. H. Liebe, "Modeling the Attenuation and Phase of Radio Waves in Air at Frequencies Below 1000GHz," *Radio Science*, Vol. 16, No. 6, 1981, pp. 1183–1199.

5. R. K. Crane, "Prediction of Attenuation by Rain," *IEEE Transactions on Communications*, Vol. 28, No. 9, September 1980, pp. 1717–1773.

6. L. J. Ippolito, *Radio Wave Propagation in Satellite Communications*, New York: Van Nostrand, 1986.

7. W. L. Flock, "Propagation Effects in Satellite Communications," NASA Reference 1108, December 1983.

8. L. J. Ippolito et al., "Propagation Effects Handbook for Satellite Systems," NASA Reference 1082, June 1983.

9. P. A. Bello, "Characterization of Randomly Time-Variant Linear Channels," *IEEE Transactions on Communication Systems*, Vol. 11, No. 4, December 1963, pp. 360–393.

10. W. C. Y. Lee, *Mobile Cellular Communications*, New York: McGraw-Hill, 1989.

11. W. C. Jakes, ed., *Microwave Mobile Communications*, New York: Wiley, 1974.

12. B. Glance and L. J. Greenstein, "Frequency Selective Fading Effects in Digital Mobile Radio with Diversity Combining," *IEEE Transactions on Communications*, Vol. 31, No. 9, September 1993, pp. 1085–1094.

13. T. S. Rappaport, *Wireless Communications*, 2nd ed., New York: Prentice Hall, 2002.

14. K. Phalaven and A. H. Leveque, *Mobile Wireless Networks*, New York: Wiley, 1995.

15. B. Sklar, "Rayleigh Fading Channels in Mobile Digital Communications," Parts I and II, *IEEE Communications Magazine*, Vol. 35, July 1997, pp. 90–110.

16. G. D. Durgin, T. S. Rappaport, and D. A. deWolf, "New Analytical Models and Probability Density Functions for Fading in Wireless Communications," *IEEE Transactions on Communications*, Vol. 50, No. 6, June 2002, pp. 1001–1015.

17. S. A. Fetchel and H. Meyer, "A Novel Approach to Modeling and Efficient Simulation of Fading Radio Channel," Proceedings of the International Conference on Communications, Geneva, May 1991, pp. 302–308.

18. W. Escalante, "Simulation of Fading Channels With Arbitrary Scattering Functions," M.S. Thesis, University of Kansas, 1996.

19. V. Fung, T. S. Rappaport, and B. Thoma, "Bit Error Simulation for pi/4 DQPSK Mobile Radio Communication Using Two-Ray and Measurement-Based Impulse Response Models," *IEEE Journal on Selected Areas in Communications*, Vol. 11, No. 3, April 1993, pp. 393–405.

20. W. D. Rummler, R. P. Counts, and M. Lineger, "Multipath Fading Models for Microwave Digital Radio," *IEEE Communications Magazine*, Vol. 24, No. 11, 1986, pp. 30–42.

21. G. L. Turin et al., "A Statistical Model of Urban Multipath Propagation," *IEEE Transactions on Vehicular Technology*, Vol. 21, February 1972, pp. 1–9.

22. H. Hashemi, "The Indoor Radio Propagation Channel," *Proceedings of the IEEE*, Vol. 81, No. 7, July 1993, pp. 943–968.

23. T. S. Rappaport, S. Y. Seidel, and K. Takamizawa, "Statistical Channel Impulse Response Models for Factory and Open Plan Building Radio Communcations System Design," *IEEE Transactions on Communications*, Vol. 39, No. 5, May 1991, pp. 794–806.

24. ETSI, "GSM Recommendations 05.05, Radio Transmission and Reception," Annex 3, 13–16, November, 1988.

25. ANSI J-STD-008, "Personal Station-Base Station Compatibility Requirements for 1.8 to 2.0 GHz CDMA PCS," March 1995.

26. 3GPP Website: A full set of specifications for UMTS release 99 is found on the website www.3gpp.org; ftp://ftp.3gpp.org/Specs/December_99/21_series/.

27. A. A. M. Saleh and R. A. Valenzuela, "A Statistical Model for Indoor Multipath Propagation," *IEEE Journal on Selected Areas in Communication*, Vol. 54, 1987, pp. 128–137.

28. T. S. Rappaport, "Characterization of UHF Multipath Radio Channels in Factory Buildings," *IEEE Transactions on Antennas and Propagation*, Vol. 37, 1989, pp. 1058–1069.

29. R. Ganesh and K. Phalavan, "Statistical Modeling and Computer Simulation of Indoor Radio Channel," *Proceedings of the IEEE*, Vol. 138, 1991, pp. 153–161.

30. J. B. Anderson, T. S. Rappaport, and S. Yoshida, "Propagation Measurements and Models for Wireless Communications," *IEEE Communications Magazine*, Vol. 33, January 1995, pp. 42–49.

31. S. C. Kim, H. L. Bertoni, and M. Stern, "Pulse Propagation Characteristics at 2.4 GHz Inside Buildings," *IEEE Transactions on Vehicular Technology*, Vol. 45, August 1996, pp. 579–592.

14.11 Problems

14.1 The lowpass transfer function models used for many time-invariant communication channels have a linear tilt in the amplitude response (in dB)

$$|H(f)| = k_1 + k_2 f \quad \text{dB} \tag{14.82}$$

and a quadratic phase response of the form

$$\angle H(f) = g_1 f + g_2 f^2 \tag{14.83}$$

Develop a MATLAB FIR filter model for this transfer function. The tilt in dB/Hz, and the maximum linear and quadratic phase offsets at the band edge are parameters of the model.

14.2 In simulating multipath fading channels it is important to calibrate the simulations. It is a common practice to normalize the power profile $p(\tau)$ and the doppler spectrum $S(\lambda)$ in order to have unit areas.

(a) Find the value of a for normalizing an exponential power profile of the form

$$p(\tau) = a \exp\left(-a\tau^2\right) \tag{14.84}$$

(b) Find the value of K needed for normalizing the Jakes doppler spectrum.

(c) Find the area under the Ricean doppler spectrum defined by (14.61).

14.3 Simulate the impact of the linear amplitude tilt and the quadratic phase distortion on a QPSK (LPE) signal with the following parameters: Symbol rate = 1MS/sec, linear tilt = 2dB/MHz, parabolic phase shift = $\pi/8$ radians at 1Mhz. No transmit filter; receive filter is an ideal integrate and dump detector.

14.4 In order to validate the results of simulating the performance of a communication system operating over fading channels, we often compare the performance of the simulated systems against similar systems operating over ideal AWGN channels and/or over a Rayleigh fading channels with ideal phase references. Compare the BER versus E_b/N_0 performance of a QPSK system operating over an AWGN channel with an integrate-and-dump receiver with a differential QPSK system over a Rayleigh fading channel with AWGN. The received signal (bandpass case) is of the form

$$y(t) = R_k \cos\left(2\pi f_c t + \phi_k + \theta_k\right) + n(t), \qquad kT_s \le t \le (k+1)T_s \tag{14.85}$$

where T_s is the symbol duration, $n(t)$ represents the AWGN, and ϕ_k represents the differential QPSK modulation. In addition, R_k and θ_k represent the amplitude and phase associated with the Rayleigh fading. Assume that R_k and θ_k change slowly with respect to the symbol rate.

14.5 Create a MATLAB simulation model for any two of the GSM models given in Section 14.8. Run BER simulations using the appropriate parameters of the GSM system for vehicle speeds of 25 and 100 MPH. Assume ideal SQRC filtering in the transmitter and receiver and ideal synchronization.

14.6 Develop an approach for generating sampled values of a set of correlated Gaussian processes each having a different PSD.

14.7 Rerun the MATLAB simulation given in Example 14.1 for different power levels and differential delays and compare the results.

14.8 Extend the simulation given in Example 14.1 to a 6-ray model, and run BER simulations for different power profiles as follows (Assume flat fading.)

(a) Uniform power over the 6 rays

(b) Exponentially decreasing power profile over the 6 taps with the last tap at 10 dB below the first ray

14.12 Appendix A: MATLAB Code for Example 14.1

14.12.1 Main Program

```
% File: c14_threeray.m
%
% Default parameters
%
NN = 256;                          % number of symbols
tb = 0.5;                          % bit time
fs = 16;                           % samples/symbol
ebn0db = [1:2:14];                 % Eb/N0 vector
%
% Establish QPSK signals
%
x = random_binary(NN,fs)+i*random_binary(NN,fs); % QPSK signal
%
% Input powers and delays
%
p0 = input('Enter P0 > ');
p1 = input('Enter P1 > ');
p2 = input('Enter P2 > ');
delay = input('Enter tau > ');
delay0 = 0; delay1 = 0; delay2 = delay;
%
% Set up the Complex Gaussian (Rayleigh) gains
%
gain1 = sqrt(p1)*abs(randn(1,NN) + i*randn(1,NN));
gain2 = sqrt(p2)*abs(randn(1,NN) + i*randn(1,NN));
for k = 1:NN
    for kk=1:fs
    index=(k-1)*fs+kk;
    ggain1(1,index)=gain1(1,k);
    ggain2(1,index)=gain2(1,k);
    end
end
y1 = x;
for k=1:delay2
    y2(1,k) = y1(1,k)*sqrt(p0);
end
for k=(delay2+1):(NN*fs)
    y2(1,k)= y1(1,k)*sqrt(p0) + ...
    y1(1,k-delay1)*ggain1(1,k)+...
    y1(1,k-delay2)*ggain2(1,k);
end
%
```

```
% Matched filter
%
b = -ones(1,fs); b = b/fs; a = 1;
y = filter(b,a,y2);
%
% End of simulation
%
% Use the semianalytic BER estimator. The following sets
% up the semi analytic estimator. Find the maximum magnitude
% of the cross correlation and the corresponding lag.
%
[cor lags] = vxcorr(x,y);
cmax = max(max(abs(cor)));
nmax = find(abs(cor)==cmax);
timelag = lags(nmax);
corrmag = cmax;
theta = angle(cor(nmax))
y = y*exp(-i*theta); % derotate
%
% Noise BW calibration
%
hh = impz(b,a); ts = 1/16; nbw = (fs/2)*sum(hh.^2);
%
% Delay the input, and do BER estimation on the last 128 bits.
% Use middle sample. Make sure the index does not exceed number
% of input points. Eb should be computed at the receiver input.
%
index = (10*fs+8:fs:(NN-10)*fs+8);
xx = x(index);
yy = y(index-timelag+1);
[n1 n2] = size(y2); ny2=n1*n2;
eb = tb*sum(sum(abs(y2).^2))/ny2;
eb = eb/2;
[peideal,pesystem] = qpsk_berest(xx,yy,ebn0db,eb,tb,nbw);
figure
semilogy(ebn0db,peideal,'b*-',ebn0db,pesystem,'r+-')
xlabel('E_b/N_0 (dB)'); ylabel('Probability of Error'); grid
axis([0 14 10^(-10) 1])
% End of script file.
```

14.12.2 Supporting Functions

A number of the supporting functions for this exmple appeared previously and are not given here. These are:

`qpsk_berest.m`	Given in Chapter 10, Appendix C.

`vxcorr.m` Given in Chapter 10, Appendix B.

`random_binary.m` Given in Chapter 10, Appendix A.

14.13 Appendix B: MATLAB Code for Example 14.2

14.13.1 Main Program

```
% File: c14_Jakes.m
% This program builds up a two-tap TDL model and computes the output
% for the two inpput signal of interest.
% Generate tapweights
%
fd = 100; impw = jakes_filter(fd);
%
% Generate tap input processes and Run through doppler filter.
%
x1 = randn(1,256)+i*randn(1,256); y1 = filter(impw,1,x1);
x2 = randn(1,256)+i*randn(1,256); y2 = filter(impw,1,x2);
%
% Discard the first 128 points since the FIR filter transient
% Scale them for power and Interpolate weight values
% Interpolation factor =100 for the QPSK sampling rate of 160000/sec;
%
z1(1:128) = y1(129:256); z2(1:128) = y2(129:256);
z2 = sqrt(0.5)*z2; m = 100;
tw1 = linear_interp(z1,m); tw2 = linear_interp(z2,m);
%
% Generate QPSK signal and filter it.
%
nbits = 512; nsamples = 16; ntotal = 8192;
qpsk_sig = random_binary(nbits,nsamples)+i*random_binary...
    (nbits,nsamples);
%
%Generate output of tap1 (size the vectors first).
%
input1 = qpsk_sig(1:8184); output1 = tw1(1:8184).*input1;
%
% Delay the input by eight samples (this is the delay specified
% in term of number of samples at the sampling rate of
% 16,000 samples/sec and generate the output of tap 2.
%
input2 = qpsk_sig(9:8192); output2 = tw2(9:8192).*input2;
%
% Add the two outptus and genrate overall output.
%
qpsk_output = output1+output2;
%
% Generate the 1000 Hz complex exponential and run it through the TDL
% model. This could be done at the higher sampling rate of 16,0000
```

```
% samples per sec or at a lower rate. At the lower rate the tap
% spacing must be recomputed in number of samples at the lower rate.
% Also the interpolation of the tap gain functions must now be at
% the lower rate. In this example we will use the higher sampling rate.
%
ts = 1/160000; time = (ts:ts:8200*ts);
cexp = exp(2*pi*i*1000*time);
input1 = cexp(1:8184); output3 = tw1(1:8184).*input1;
input2 = cexp(9:8192); output4 = tw2(9:8192).*input2;
%
% Add the two outputs and genrate overall output.
%
cexp_out = output3+output4;
[psdcexp,freq,ptotal,pmax] = linear_psd(cexp(1:8184),8184,ts);
[psdcexp_out,freq,ptotal,pmax] = linear_psd(cexp_out(1:8184),8184,ts);
%
subplot(2,1,1)
plot(freq(4100:4180), psdcexp(4100:4180)); grid;
xlabel('Frequency (Hz)'); ylabel('PSD')
subplot(2,1,2)
plot(freq(4100:4180), psdcexp_out(4100:4180),'r'); grid;
xlabel('Frequency (Hz)'); ylabel('PSD')
figure; subplot(2,1,1)
plot(real(qpsk_sig(501:1000)),'r'); grid;
xlabel('Sample Index'); ylabel('Direct Input');
axis([0 500 -2 2])
subplot(2,1,2)
plot(real(qpsk_output(501:1000)));grid;
xlabel('Sample Index'); ylabel('Direct Output');
figure;
plot(abs(output3(3000:6000))); grid
xlabel('Sample Index'); ylabel('Envelope Magnitude')
% End script file.
```

14.13.2 Supporting Functions

jakes_filter.m

```
% File: Jakes_filter.m
function [impw] = jakes_filter(fd)
% FIR implementation of the Jakes filter (128 points)
n = 512; nn = 2*n;                % nn is FFT block size
fs = 0:fd/64:fd;                  % sampling frequency = 16*fd
H = zeros(1,n);                   % initialize H(f)
for k=1:(n/8+1)                   % psd for k=1:65
    jpsd(k)=1/((1-((fs(k))/fd)^2)^0.5);
```

```
    if(jpsd(k)) > 1000
        jpsd(k)=1000;
    end
    H(k)=jpsd(k)^0.5; % first 65 points of H
end
for k=1:n                          % generate negative frequencies
    H(n+k) = H(n+1-k);
end
[inv,time] = linear_fft(H,nn,fd/64);    % inverse FFT
imp = real(inv(450:577));               % middle 128 points
impw = imp.*hanning(128)';              % apply hanning window
energy = sum(impw.^2);                  % compute energy
impw = impw/(energy^0.5);               % normalize
% End of function file.
```

linear_psd.m

```
% File: linear_psd.m
function [psd,freq,ptotal,pmax] = linear_psd(x,n,ts)
% This function takes the n time domain samples (real or complex)
% and finds the psd by taking (fft/n)^2. The two sided spectrum is
% produced by shifting the psd.
% NOTE: n must be an even number, preferably a power of 2.
for k=1:n
    y(k) = 0.;
end
for k=1:n
    freq (k) =( k-1-(n/2))/(n*ts);
    y(k) = x(k)*((-1.0)^k);
end;
v = fft(y)/n; psd = abs (v).^2;
pmax = max(psd); ptotal = sum(psd)
% End of function file.
```

Chapter 15

DISCRETE CHANNEL MODELS

In this chapter we take a distinctly different approach to the problem of channel modeling. In our previous work, the channel was defined in terms of the noise, interference, and other disturbances that combine with the transmitted signal to produce a distorted and noisy waveform at the input to the receiver. The transmitted signal, as well as the noise, interference, and other channel disturbances, were all represented by samples of waveforms. The result was a waveform-level simulation that processed data on a sample-by-sample basis. We now consider partitioning the system in a way that eliminates much of the need for simulating at the waveform level. The result is a discrete channel model in which simulations operate on a symbol-by-symbol basis. The motivation for replacing a waveform-level channel model by a discrete channel model is speed of simulation. We will see that the discrete channel model is an abstraction of the physical (waveform) channel in which the channel is completely characterized by a small set of parameters. Determining these parameters constitutes an important part of the modeling process, and must be accomplished through measurements made on the physical channel or through the use of a single waveform-level simulation.

15.1 Introduction

The basic model of a communication system, illustrated in Figure 15.1, typically consists of a discrete data source, a channel coder for error control, a modulator and transmitter, a channel, a receiver, and a decoder. Depending on the application for the system and the detail of the simulation, other elements such as equalizers, interleavers, deinterleavers, carrier synchronizers, and symbol synchronizers, may be part of the system model. As we know, the modulator maps a symbol, or a sequence of symbols, at the modulator input onto a waveform at the modulator output. The waveform is subjected to a number of degrading effects in the channel. Among these effects are noise, bandlimiting, interference, and fading. All of these effects can be characterized by waveforms. The input to the receiver is also a waveform, which is the transmitted waveform combined with the degrading effects of the channel. The role of the receiver is to observe the input waveform over a symbol period, or over a sequence of symbols, and determine the transmitted symbols. The channel in this case is referred to as a waveform channel, since both the channel input and the channel output are characterized as waveforms. It is important to note that all system elements, with the exception of the channel, are characterized by deterministic mappings. The channel performs a stochastic, or random, mapping of the channel input to the channel output.

The terminology "discrete channel model" (DCM) is used to denote all the elements of a communication system that lie between the two points **A** and **B** in the system, where the input at point **A** is a vector of discrete symbols (input sequence), denoted $\mathbf{X} = [x_1, x_2, \cdots, x_k, \cdots]$, and the output at **B** is another vector of discrete symbols (output sequence), denoted $\mathbf{Y} = [y_1, y_2, \cdots, y_k, \cdots]$. Usually, point **A** will

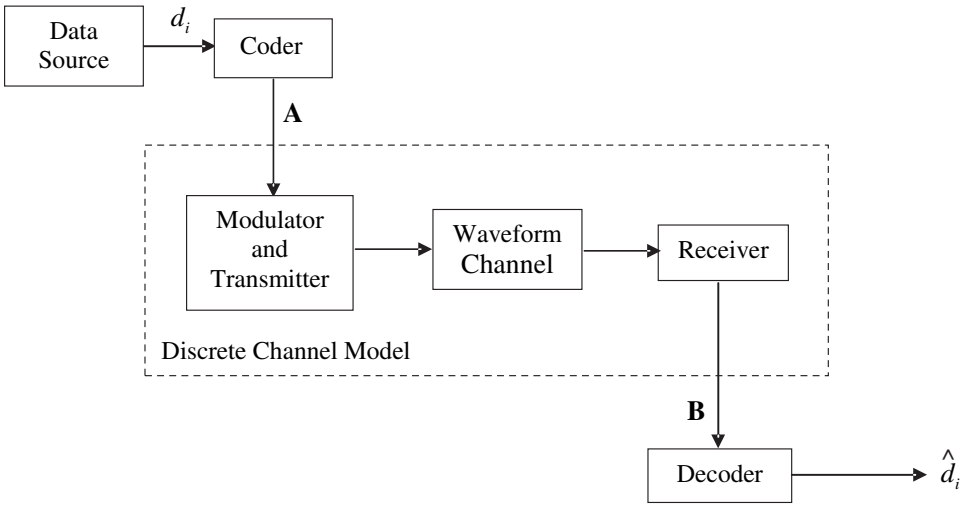


Figure 15.1 Communication system with discrete channel model.

be the output of the channel encoder or, equivalently, the modulator input, and point **B** will be the input to the decoder as shown in Figure 15.1. Note that, with the partitioning shown in Figure 15.1, the modulator and transmitter, the waveform channel, and the receiver are part of the discrete channel model. The relationship between the discrete channel input vector **X** and the output vector **Y** will be affected by the system components, such as filters, and by the random disturbances induced in the physical (waveform) channel. In a binary communication system with hard decision receivers, the elements of both **X** and **Y** will be binary sequences. In the case of soft-decision receivers, the elements of **Y** will be from an M-ary alphabet. Transmission errors resulting from imperfections in the system elements between points **A** and **B** in Figure 15.1, including the physical channel, will cause elements of **X** and **Y** to be different, at least occasionally.

Discrete channel models describe the error-generation mechanism probabilistically. These models fall into two categories. The first class of these models, referred to as "memoryless" channel models, are used to model the transmission errors or the transitions from the channel input to the channel output under the assumption that there is no temporal correlation in the transition mechanism. That is, the input-output transition probabilities for the n^{th} channel input symbol are not affected by what happened to any other input symbol. Such models are applicable to channels in which there is no inter-symbol interference (ISI) or fading, and the noise is AWGN. In a hard decision binary system, this assumption implies that bit errors are uncorrelated. Discrete channel models for memoryless channels are usually trivial to derive.

The second and more interesting class of discrete models apply to situations in which the transitions from the input symbols to the output symbols are temporally correlated. In this case the probability of error for the n^{th} symbol depends on whether or not an error occurred in the transmission of previous symbols. The fading channel commonly encountered in wireless communications is a good example of a channel having correlated errors. Errors will tend to occur in bursts when the radio channel is in a deep fade, and these channels are referred to as burst error channels or channels with memory.

Discrete channel models are probabilistic models that are computationally more efficient than waveform channel models. The increased efficiency comes from two factors. Discrete models are simulated at the symbol rate, whereas waveform level models are typically simulated at 8 to 16 times the symbol rate. This alone reduces the computational burden by roughly an order of magnitude. While each individual block is simulated in detail in a waveform-level model, there is a high level of abstraction in the discrete channel model. This level of abstraction, as we will see, further reduces the computational burden. These two factors may well contribute to several orders of magnitude savings in the time required to execute a simulation.

In simple cases, discrete channel models can be derived analytically from the models of the underlying components between the discrete channel input **A** and the discrete channel output **B**. However, in most cases, discrete channel models are derived from simulated or measured error patterns between the points **A** and **B**. Discrete channel models are used to design and analyze the impact of components

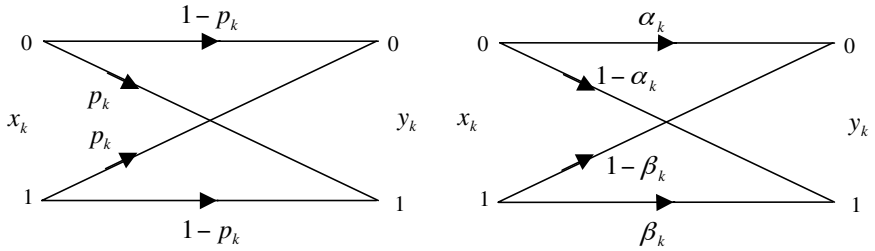
outside the portion of the system between points **A** and **B**. This includes, for example, error control coders, source encoders, and interleavers.

Modeling discrete memoryless channels is a straightforward process. For example, in the case of a discrete memoryless symmetric channel with binary input and binary output, all we need to know to characterize the channel is a single number, namely, the bit error probability. Simulation of this channel involves drawing a single random number and comparing that number to a threshold in order to decide whether or not a given bit will suffer a transmission error or be received without error. Thus, in order to simulate the transmission of a million bits through a given channel, all we have to do is generate a million independent uniformly distributed random numbers and perform the required threshold comparisons. The entire system represented by this channel is therefore simulated efficiently. (Compare this with a waveform-level simulation, which involves generating millions of samples representing the waveforms present in the system being simulated, and processing these samples though all of the functional blocks in the system.)

Discrete channels with memory are more difficult to model. This temporally correlated error generation mechanism is usually modeled by a discrete-time Markov sequence [1, 2, 3, 4] in which a state model is used to characterize the various states of the channel and a set of transition probabilities are used to capture the progression of the channel states. Each state will also have associated with it a set of input-to-output symbol transition probabilities. Thus, the model is now more complex and requires more parameters than the uncorrelated channel. Model structure and parameter values are estimated from either simulated or measured error patterns. Simulation of the discrete channel consists of generating a random number prior to the transmission of each symbol to determine the channel state, and then drawing another random number to determine the input-to-output transition. While the model and parameter estimation procedures are complex, simulation of the Markov model is very efficient. We will elaborate on these ideas in the following sections.

15.2 Discrete Memoryless Channel Models

In a discrete memoryless channel, the mapping of input to output is instantaneous and is described by a set of transition probabilities. The simplest discrete memoryless channel model is the binary symmetric channel (BSC), which is illustrated in Figure 15.2(a). The input to the discrete channel is a sequence of binary symbols, denoted by the vector $\mathbf{X}$. The k^{th} component of this vector, x_k, corresponds to the k^{th} channel input, and p_k corresponds to the probability of error on the k^{th} symbol transmission. For a memoryless channel, p_k is independent of k, so the error on all symbols is effected by the channel in the same manner. (If the channel has memory, p_k will, in general, be a function of p_{k-1}.) A sequence of symbols of length M is processed through the memoryless channel by invoking the channel model M times in succession. For the k^{th} symbol, a binary input of "0" is received correctly as "0" with a probability $1 - p_k$, and incorrectly as "1" with an error probability of p_k. The k^{th} channel output is denoted by y_k and the M-symbol sequence of outputs is



(a) Binary symmetric channel

(b) General binary channel

Figure 15.2 Binary channel models.

denoted by the vector $\mathbf{Y}$. The channel is symmetric in that zeroes and ones are affected by the channel in the same manner. Note that one consequence of symmetry is that the error probability is independent of the transmitted symbol so that the error source can be simulated separately from the information (data) source.

For a binary channel, the input-output relationship can be expressed as

$$\mathbf{Y} = \mathbf{X} \oplus \mathbf{E} \tag{15.1}$$

in which $\mathbf{X}$ and $\mathbf{Y}$ are the input and output data vectors, respectively, $\oplus$ denotes the XOR operation, and $\mathbf{E}$ is the error vector. Specifically, $\mathbf{E} = \{e_1, e_2, e_3, \cdots\}$ is a binary vector or sequence having elements $\{0, 1\}$ in which $e_k = 0$ denotes that the k^{th} element of $\mathbf{X}$, x_k, is received correctly ($y_k = x_k$), and $e_k = 1$ denotes that the k^{th} element of $\mathbf{X}$ is received in error ($y_k \neq x_k$). The parameter defining the performance of the binary symmetric channel model is the probability of error P_E, which can be easily estimated from measurements or from a system simulation performed at the waveform level. Note that, for a binary system, the Monte Carlo estimate of the error probability is the Hamming weight of the error vector $\mathbf{E}$ divided by N, the number of elements in the vector $\mathbf{E}$.[1]

Note that once p_k is known, the error sequence corresponding to N transmitted symbols can be generated by making N calls to a uniform random number generator and comparing the random number to the threshold p. Specifically:

$$e_k = \begin{cases} 1, & U_k \leq p_k \\ 0, & U_k > p_k \end{cases} \tag{15.2}$$

where U_k denotes the number obtained from the k^{th} call to the random number generator. It is important to realize that the discrete channel model will not reproduce the original error pattern used to originally determine p. It will, however, reproduce a statistically equivalent error pattern. The simplicity of (15.2) illustrates

[1] Recall that the Hamming weight of a binary vector, consisting only of the elements "0" and "1," is equal to the number of binary ones in the vector.

why substitution of a DCM for the portion of the system between points **A** and **B** in Figure 15.1 results in a significant savings in computational complexity.

A slightly more complicated model for the binary channel is the binary nonsymmetric channel model in which the probability of error may be different for zeroes and ones, as in some optical communication systems. This model is illustrated in Figure 15.2(b). This model is characterized by a set of four input-to-output transition probabilities, but only two of these probabilities, α_k and β_k, are independent. The channel is nonsymmetric if $\alpha_k \neq \beta_k$.

Extension of this model to the case in which the input and output belong to an M-ary alphabet is straightforward and is shown in Figure 15.3 for $M = 3$. The channel is characterized by a set of probabilities α_{ij}, $i, j = 0, 1, 2$, in which α_{ij} represents the conditional probability

$$\alpha_{ij} = \Pr\{\text{output} = j \,|\, \text{input} = i\} \tag{15.3}$$

If the set of probabilities α_{ij} are constant, and therefore independent of k, the k^{th} channel output depends only on the k^{th} channel input. For this case, the model shown in Figure 15.3 is memoryless. The transition probabilities are estimated using a detailed waveform-level simulation in which the transitions from the i^{th} input to the j^{th} output are counted. In these models, the transition probabilities are estimated from simulated or measured data as

$$\alpha_{ij} = \frac{n_{ij}}{N_i} \tag{15.4}$$

where N_i is the number of occurrences of the i^{th} input symbol, and n_{ij} is the number of times the i^{th} input symbol transitions to the j^{th} output. Another example, shown in Figure 15.4, is used for memoryless channels in which the output of the channel

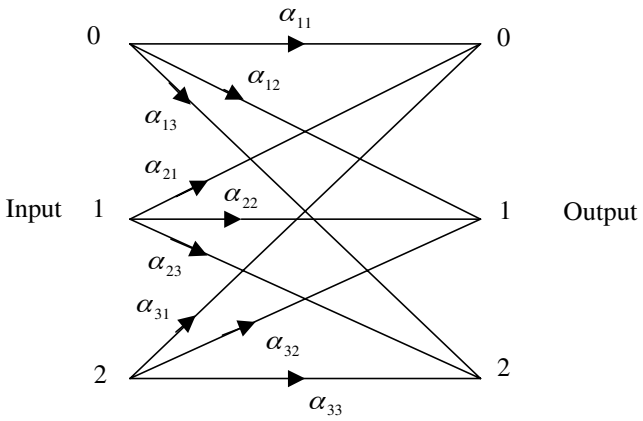


Figure 15.3 Discrete channel model with three inputs and three outputs.

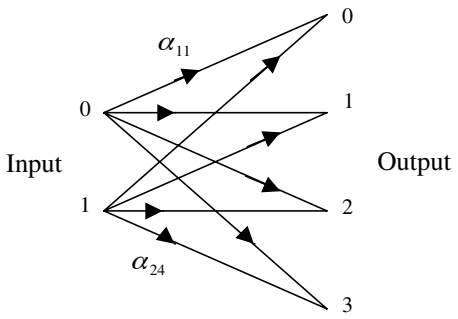


Figure 15.4 Discrete channel model with two inputs and four outputs.

Source: M. C. Jeruchim, P. Balaban, and K. S. Shanmugan, *Simulation of Communications Systems*, 2nd ed., New York: Kluwer Academic/Plenum Publishers, 2000.

is quantized to a different number of levels from the number of input levels, as in soft decision decoding.

15.3 Markov Models for Discrete Channels with Memory

For channels with memory, the most commonly used model is the discrete-time, finite-state Markov model (MM). There are several reasons for the popularity of Markov models. These models are analytically tractable, their theory is well established in the statistical literature, and they have been applied successfully to a variety of important communication problems. Markov models have been used to model the output of discrete information sources such as English text. (The occurrence of a sequence of letters of the English alphabet in a typical passage can be modeled as a Markov sequence.) Similarly, the sampled values of an audio or video waveform can also be modeled by an MM. Markov modeling techniques are directly applicable to the modeling and analysis of discrete communication channels [1]. Also, Markov models can be used to evaluate the capacity of a discrete channel and for the design of optimal error control coding techniques. Markov models have been successfully used to characterize fading channels in wireless communication systems [5–10]. Most importantly, computationally efficient techniques are available for estimating the parameters of Markov sequences from simulated or measured error patterns. As an introduction to the Markov Model, we first consider a simple two-state model. The results are then generalized to an N-state model.

15.3.1 Two-State Model

To set the stage for Markov channel models, consider a fading channel in which the received signal strength is above an acceptable performance threshold part of the time, and below the threshold during a deep fade. If we have interest only in the

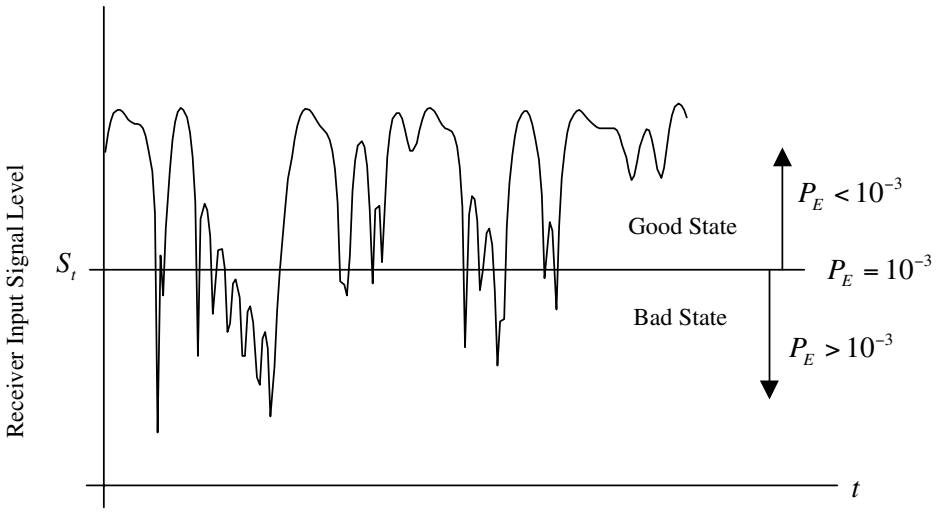


Figure 15.5 Assumed receiver input signal level and two-state model.

"above threshold" or "below threshold" conditions, we can model the channel to be in either one of the two states as shown in Figure 15.5. In Figure 15.5 we have a good state, g, in which the system performance is acceptable ($P_E < 10^{-3}$) and a bad state, b, in which the received signal level is so low that the probability of error is unacceptably high ($P_E > 10^{-3}$).

Thus, in the simple two-state model shown in Figure 15.5, the states can be represented by the set

$$S = \{g, b\} \tag{15.5}$$

As time progresses, the channel goes from good state to bad state, and vice versa. The rate of transition and the length of stay in each of the two states will depend on the temporal correlation of the fading process. If time is measured in increments of a symbol (bit) time, then we can construct a discrete channel model as follows.

At the beginning of each symbol (bit) interval, the channel is in one of the two states. If the channel is in a good state, the probability of transmission error is negligible. On the other hand, if the channel is in a bad state, the probability of transmission error is unacceptably high. Prior to the transmission of each new bit, the channel may change state or remain in the current state. This transition between states takes place with a set of transition probabilities, a_{ij}. These are conditional probabilities, and for the two-state model there are four probabilities of interest. Let S_t and S_{t+1} denote the state of the channel at time t and at time $t+1$, respectively. (Note that $t+1$ represents time one *increment* greater than t, and not

one second greater than t, so that t can be viewed as a discrete time index.) With this notation, we define the four transition probabilities to be

$$\begin{aligned} a_{gg}(t) &= \Pr\{S_{t+1} = g \,|\, S_t = g\} \\ a_{gb}(t) &= \Pr\{S_{t+1} = b \,|\, S_t = g\} \\ a_{bg}(t) &= \Pr\{S_{t+1} = g \,|\, S_t = b\} \\ a_{bb}(t) &= \Pr\{S_{t+1} = b \,|\, S_t = b\} \end{aligned} \tag{15.6}$$

This can be represented by the state transition matrix

$$A(t) = \begin{bmatrix} a_{gg}(t) & a_{gb}(t) \\ a_{bg}(t) & a_{bb}(t) \end{bmatrix} \tag{15.7}$$

Note that since we are considering a two-state model, given that $S_t = g$, we must have $S_{t+1} = g$ (the channel remains in the good state) or $S_{t+1} = b$ (the channel transitions to the bad state). Thus, each row of the state transition matrix must sum to 1. A graphical representation of the state transition diagram of the two-state model is shown in Figure 15.6.

In the work to follow we will assume that the channel model is stationary and therefore the state state transition matrix $A(t)$ is fixed so that $A(t) = A$. However, if a simulation is performed in which the initial state probability distribution Π_0 is different from the steady-state distribution, some time is required for the state distribution to evolve to the steady-state value Π_{ss}. The probability of finding the model in a given state is of interest. Define Π_t as the state probability distribution at time t. Specifically:

$$\Pi_t = \begin{bmatrix} \pi_{t,g} & \pi_{t,b} \end{bmatrix} \tag{15.8}$$

in which $\pi_{t,g}$ and $\pi_{t,b}$ represent the probabilities of finding the channel in the good state or in the bad state at time t, respectively.[2] By definition of the state transition matrix, the state distribution at time $t+1$ is given by

$$\Pi_{t+1} = \Pi_t A \tag{15.9}$$

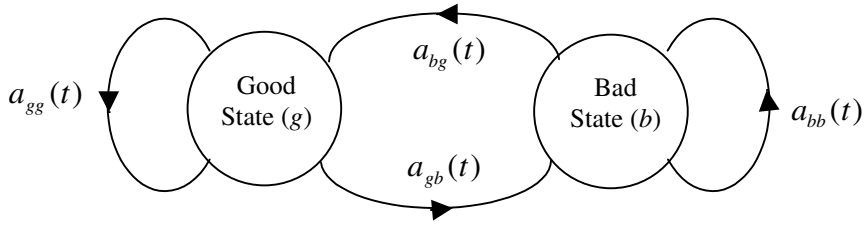


Figure 15.6 Two-state Markov model.

[2]Note that in this formulation we are not using t as a continuous variable but as a time index that ranges over a set of integers.

In a similar manner, the state distribution at time $t+2$ is

$$\Pi_{t+2} = \Pi_{t+1}A = (\Pi_t A)A = \Pi_t A^2 \tag{15.10}$$

Thus, in general

$$\Pi_{t+k} = \Pi_t A^k \tag{15.11}$$

where A^k represents the k step transition matrix.

Most, but not all, Markov processes settle to a steady-state probability distribution as time evolves.[3] Assuming that the Markov process converges to a steady-state value distribution $\Pi_{t+k} = \Pi_t$ for sufficiently large t and arbitrary k, it follows that

$$\Pi_{ss} = \Pi_{ss}A^k = \begin{bmatrix} \pi_g & \pi_b \end{bmatrix} \tag{15.12}$$

for arbitrary k. It is easily shown that for sufficiently large values of k, the rows of A^k give Π_{ss} [1]. The convergence of Π_0 to Π_{ss} is illustrated in the following example.

Example 15.1. As a simple example let

$$A = \begin{bmatrix} 0.98 & 0.02 \\ 0.05 & 0.95 \end{bmatrix} \tag{15.13}$$

with the initial state distribution

$$\Pi_0 = \begin{bmatrix} 0.50 & 0.50 \end{bmatrix} \tag{15.14}$$

The following MATLAB program illustrates the convergence of Π_0 to Π_{ss}:

```
% File:  c15_MMtransient.m
N = 100;
pie = zeros(N,2);
A = [0.98 0.02; 0.05 0.95];
pie(1,:)  = [0.50 0.50];
for k=2:N
    pie(k,:)  = pie(k-1,:)*A;
end
kk = 1:N;
plot(kk,pie(:,1),'k-',kk,pie(:,2),'k:')
xlabel('Iteration')
ylabel('Probability')
text1 = ['The steady-state probabilities are ',...
       num2str(pie(N,1)),' and ' ,num2str(pie(N,2)),'.'];
legend('State 1','State 2',2)
disp(text1)
```

[3]We have interest only in those process for which the state distribution converges to a steady-state value.

```
disp(' ')
disp('The value of A^N is'); A^N
% End of script file.
```

Executing the program results in the script

```
≫ c15_MMtransient
The steady-state probabilities are 0.71412 and 0.28588.

The value of A^N is
ans =
0.7145 0.2855
0.7138 0.2862
```

We see that calculating the value of Π_{ss} iteratively using (15.9) and by raising A to a high value gives, as expected, consistent results. The manner in which the probability distribution converges to the steady-state value is shown in Figure 15.7. ■

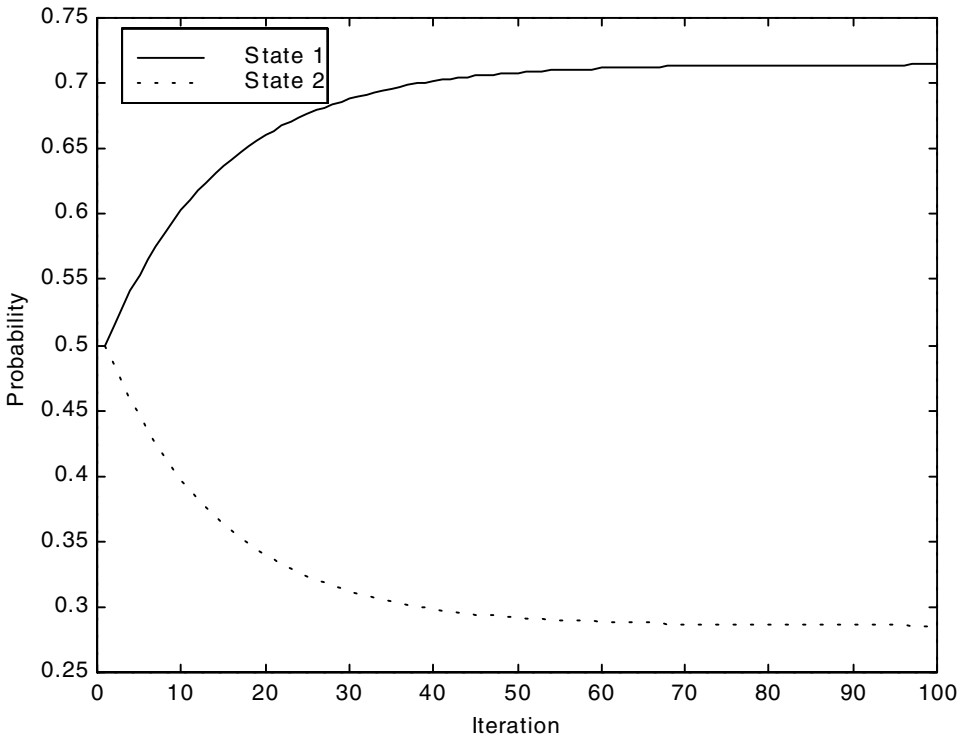


Figure 15.7 Convergence of the state probability distribution to the steady-state values.

Before leaving the two-state model, we need to discuss the error generation matrix, defined as

$$B = \begin{bmatrix} \Pr\{C|g\} & \Pr\{C|b\} \\ \Pr\{E|g\} & \Pr\{E|b\} \end{bmatrix} \tag{15.15}$$

in which "C" denotes that a correct decision is made and "E" denotes that an error is made. By simple matrix multiplication it follows that the unconditional probability of a correct decision P_C and error P_E are given by

$$\begin{bmatrix} P_C & P_E \end{bmatrix} = \Pi_{ss} B^T \tag{15.16}$$

where Π_{ss} is the steady-state state distribution matrix and B^T is the transpose of the error generation matrix B.

Note that, if all elements of B are nonzero, either state can produce an error although the error probabilities may be quite different for different states. Thus, upon observation of an error, the state that produced the error cannot be identified. It is for this reason that we call the model "hidden." These models are referred to as hidden Markov models (HMMs).

Example 15.2. In this example, the simulation of the channel using the Markov model is demonstrated and the system error probability is computed. Assume that errors can be produced in either state where, obviously, the probability of error in the good state will be much less than the error probability in the bad state. Specifically we define the conditional error probabilities as

$$\Pr\{E|g\} = 0.0005 \tag{15.17}$$

and

$$\Pr\{E|b\} = 0.1000 \tag{15.18}$$

For these error probabilities, the error generation matrix B takes the form

$$B = \begin{bmatrix} 0.9995 & 0.9000 \\ 0.0005 & 0.1000 \end{bmatrix} \tag{15.19}$$

In addition, the Markov chain is defined by the transition matrix

$$A = \begin{bmatrix} 0.98 & 0.02 \\ 0.05 & 0.95 \end{bmatrix} \tag{15.20}$$

as in the previous example. The MATLAB program for simulating the channel is

```
% File:  c15_hmm2.m
N = 100000;                          % number of iterations
state = 'Good';                      % initial state
A = [0.98 0.02; 0.05 0.95];          % state transition matrix
B = [0.0005 0.1000];                 % second row of B
```

```
out = zeros(1,N);                    % initialize matrix
errors = 0;
for i=1:N
    error = 0;                       % initialize error counter
    y = rand(1);                     % RV for state transition
    err = rand(1);                   % RV for error given state
    if state==‘Good’                 % test for Good state
        if y<A(1,1)
            state=‘Good’;            % remain in Good state
            if err<B(1);             % test for error
                error = 1;           % record an error
            end
        else
            state=‘Bad ’;            % transition to Bad state
            if err<B(2);             % test for error
                error = 1;           % record an error
            end
        end
    else                             % state = Bad
        if y<A(2,2);
            state=‘Bad ’;            % remain in Bad state
            if err<B(2);             % test for error
                error = 1;           % record an error
            end
        else
            state=‘Good’;            % transition to Good state
            if err<B(1);             % test for error
                error = 1;           % record an error
            end
        end
    end
    errors = errors + error;         % increment error counter
end
PE = errors/N                        % calculate error probability
% End of script file.
```

Executing the program yields the following MATLAB result:

```
≫ c15_hmm2
PE =
0.0285
```

Thus, we have

$$P_E = 0.0285 \tag{15.21}$$

Note that P_E computed in this manner is a random variable. We now "sanity-check" this simulation result.

From the previous example, the steady-state probabilities are

$$\Pi_{ss} = \begin{bmatrix} 0.7141 & 0.2859 \end{bmatrix} \tag{15.22}$$

From (15.16) we have

$$\begin{bmatrix} P_C & P_E \end{bmatrix} = \begin{bmatrix} 0.7141 & 0.2859 \end{bmatrix} \begin{bmatrix} 0.9995 & 0.0005 \\ 0.9000 & 0.1000 \end{bmatrix} \tag{15.23}$$

This gives

$$\begin{bmatrix} P_C & P_E \end{bmatrix} = \begin{bmatrix} 0.9711 & 0.0289 \end{bmatrix} \tag{15.24}$$

which is very close ($P_E = 0.0289$) to the value given by the simulation. Thus, the simulation result appears reasonable. ■

A more general version of the two-state Markov model can be constructed with a larger number of states, with many good and bad states with varying degrees of "goodness" or "badness" corresponding to increasingly more severity of fading and or noise. The result is the N-state Markov model, which is a generalization of the two-state model considered in this section.

15.3.2 N-state Markov Model

An N-state Markov (processes) model for a discrete communication channel is defined by a number of parameters. The set of states S is denoted by

$$S = \{1, 2, 3, \cdots, N\} \tag{15.25}$$

and S_t, as before, denotes the state at time t. Thus, S_t ranges over the set S. We have interest in the probability, $\Pi_{t,i}$, that the Markov model will be in state i at time t. We denote this as

$$\Pi_{t,i} = \Pr\left[S_t = i\right], \qquad i = 1, 2, 3, \cdots, N \tag{15.26}$$

The transition probabilities, a_{ij}, denote the probability of going from state i at time t ($S_t = i$) to state j at time $t + 1$ ($S_{t+1} = j$). In equation form

$$a_{ij} = \Pr\left[S_{t+1} = j \,|\, S_t = i\right], \qquad i, j = 1, 2, 3, \cdots N \tag{15.27}$$

This transition normally takes place in a time increment equal to a bit or symbol duration. Finally we have the input-to-output transition (error symbol) probabilities. The error symbols for an M-ary symbol alphabet are denoted by the set

$$E = \{e_1, e_2, e_3, \cdots, e_M\} \tag{15.28}$$

For the familiar binary case

$$E = \{0, 1\} \tag{15.29}$$

where 0 denotes no error and 1 denotes an error. The probability that error symbol e_k occurs given that the model is in state i is denoted $b_i(e_k)$. In equation form we have

$$b_i(e_k) = \Pr\left[e_k \,|\, S_t = i\right] \tag{15.30}$$

In binary hard decision applications, the parameters $b_i(e_k)$ are represented by a matrix having two rows and N columns, where N represents the number of states in the model. Thus, the error generation matrix takes the form

$$B = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1i} & \cdots & b_{1N} \\ b_{21} & b_{22} & \cdots & b_{2i} & \cdots & b_{2N} \end{bmatrix} \tag{15.31}$$

where b_{1i} denotes the probability of a correct decision given that the model is in state i, and b_{2i} represents the probability of error given that the system is in state i. With this formulation, note that the columns of B must sum to one. For nonbinary applications, or binary soft-decision applications, the error generation matrix B will have more than two rows.

These parameters define a discrete-time Markov process operating at a transition rate equal to the symbol rate of the communication system, and the output of the process consists of two sequences: the sequence of states $\{S_t\}$, and a sequence of error symbols $\{E_t\}$, where t is the time index that can be indexed over the integer set $\{0, 1, 2, \cdots\}$. Normally, only the input and the output of the channel and hence the error sequence can be observed. The state sequence itself cannot be observed. Hence the state sequence is "hidden" or not visible from external observations and the Markov model is labeled as a hidden Markov model.

15.3.3 First-Order Markov Process

The Markov process of order m (or memory m), defined as the probability of the state at time $t+1$, S_{t+1}, is given by

$$\Pr[S_{t+1} |\, S_t, S_{t-1}, ...] = \Pr[S_{t+1} |\, S_t, S_{t-1}, ...S_{t-m-1}] \tag{15.32}$$

which states that the probability is dependent upon the previous m states. For a first-order Markov process, the probability of the state at time $t+1$ is a function only of the previous state. In other words:

$$\Pr[S_{t+1} |\, S_t, S_{t-1}, ...] = \Pr[S_{t+1} |\, S_t] \tag{15.33}$$

For channel models we will use a first-order Markov model. For modeling data sources, higher-order models may be required. As an example, modeling the symbol sequences in an English-language text may require a second-order or third-order Markov model, since the probability of occurrence of a given letter may depend on the previous two or three letters.

15.3.4 Stationarity

Since stationarity is usually assumed in modeling the analog portion of the communication channel, it is common to assume stationarity of the Markov model for

discrete channels also. Stationarity implies that the parameters of the model, that is, the probabilities $\Pi_{t,i}$, $a_{ij}(t)$, and $b_i(e_k)$ do not depend on t. In this case, we have

$$\Pr[S_{t+1}=i]=\Pi_i=\sum_{k=1}^{N}\Pr[S_t=k]\Pr[S_{t+1}=i|S_t=k]=\sum_{k=1}^{N}\pi_k a_{ki}, \quad i=1,\cdots,N \tag{15.34}$$

where the terms a_{ki} are the elements in the i^{th} column of the state transition matrix

$$A=\begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1i} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2i} & \cdots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \ldots & a_{Ni} & \cdots & a_{NN} \end{bmatrix} \tag{15.35}$$

and Π_k are the elements of the state probability distribution vector

$$\Pi=\begin{bmatrix} \pi_1 & \pi_2 & \cdots & \pi_N \end{bmatrix} \tag{15.36}$$

As in the case of a two-state model, the n-stage transition matrix is given by A^n. As was demonstrated in Example 15.1 for a two-state model, (15.34) along with the constraint that

$$\sum_{i=1}^{N}\pi_i=1 \tag{15.37}$$

implies that the state probabilities, π_i, are uniquely determined from the transition probabilities, a_{ij}. Hence the Markov model is completely defined by the matrix A of the state transition probabilities and B, where B is the matrix of the input-to-output symbol transition (i.e., error) probabilities

$$B=\begin{bmatrix} b_{11} & \cdots & b_{1N} \\ \vdots & \ddots & \vdots \\ b_{M1} & \cdots & b_{MN} \end{bmatrix} \tag{15.38}$$

as previously discussed.

15.3.5 Simulation of the Markov Model

Once the discrete Markov model is derived, simulation of the model is relatively easy. We saw this in Example 15.2 for a two-state model, and the technique used in Example 15.2 is now generalized for an N-state model.

Assume that the model is in state k at time t and that both the next state and the corresponding component of the error vector, e_{k+1}, is to be simulated. The first step is to generate two uniformly distributed random variables, U_1 and U_2. The first random variable, U_1, is used to determine the new state. Given that the model

is in state k, the transition probabilities are defined by the k^{th} row of the state transition matrix A. The k^{th} row of A is designated by the vector A_k where

$$A_k = \begin{bmatrix} a_{k1} & a_{k2} & \cdots & a_{ki} & a_{k,i+1} & \cdots & a_{kN} \end{bmatrix} \tag{15.39}$$

The cumulative probabilities associated with A_k are denoted β_k and are given by

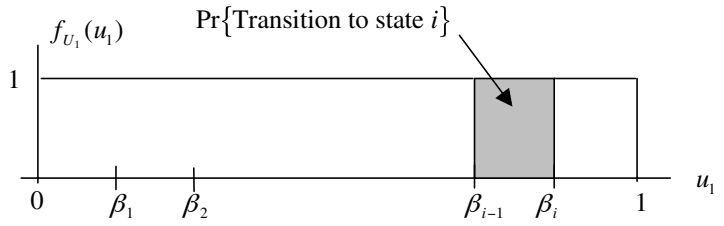
$$\beta_k = \sum_{j=1}^{k} a_{kj} \tag{15.40}$$

Note that the MATLAB library function `cumsum` maps the vector A_k into the a vector having elements β_k. The probability of making a transition from state k to state i is, by definition, a_{ki}, and is given by

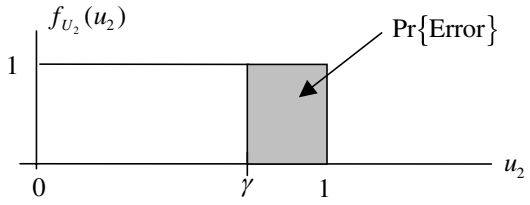
$$a_{ki} = \beta_i - \beta_{i-1} \tag{15.41}$$

This probability is represented by the shaded area in Figure 15.8(a).

After the new state is established, the next step is to determine whether or not an error is made in the new state. In order to accomplish this, we compare the second uniform random variable U_2 with the threshold γ shown in Figure 15.8(b). The probability of a correct decision is $\Pr\{U_2 < \gamma\}$, and the probability of error is $\Pr\{U_2 > \gamma\}$, which is the shaded region in Figure 15.8(b). Since the current state is i, the value of γ is given by the element b_{1i} (row 1, column i) in the state observation matrix B.



(a) Determination of transition probability



(b) Determination of error probability

Figure 15.8 Steps in the simulation of a Markov model.

The implementation of the simulation procedure is realized using the following MATLAB code:

```
u1 = rand(1);                       % get uniform RV 1
cum_sum = [0 cumsum(A(state,:))];
for i=1:total_states                % loop to determine new state
    if u1 > = cum_sum(i) & u1 < cum_sum(i+1);
        state = i;                  % assign new state
    end
end
state_seq(t) = state;               % new state
u2 = rand(1);                       % get uniform RV 2
if u2 > B(1,state)
    out(t) = 1;                     % record error
end
```

We now demonstrate this process in the following example.

Example 15.3. In this example, a binary sequence of length $N = 20,000$ is generated representing symbol errors on a channel. This error sequence is generated using a three-state Markov model. If a given element in the sequence is a binary "one," a channel symbol error is recorded for the position corresponding to the given element. A binary "zero" denotes that an error did not occur in the corresponding position.

We assume that errors can occur in any of the three states. Specifically we assume

$$\Pr\{E|S_1\} = 0.0010 \tag{15.42}$$

$$\Pr\{E|S_2\} = 0.0500 \tag{15.43}$$

and

$$\Pr\{E|S_3\} = 0.0100 \tag{15.44}$$

which corresponds to the error probability matrix

$$B = \begin{bmatrix} 0.9990 & 0.9500 & 0.9900 \\ 0.0010 & 0.0500 & 0.0100 \end{bmatrix} \tag{15.45}$$

The state transition matrix for the Markov model is assumed to be

$$A = \begin{bmatrix} 0.80 & 0.10 & 0.10 \\ 0.20 & 0.60 & 0.20 \\ 0.02 & 0.08 & 0.90 \end{bmatrix} \tag{15.46}$$

We assume that the model is initially in state 1.

MATLAB code `c15_errvector.m`, for generating the error sequence, is given in Appendix A. Execution of the program `c15_errvector` requires as input data the

parameters N and the matrices B and A. The default values are $N = 20,000$, and the matrices B and A are defined by (15.45) and (15.46), respectively. The program produces two outputs. These are the error vector `out`, which gives the error positions in a sequence of N transmissions, and the state sequence vector `state_seq`, which details the state transitions. The probability of finding the model in a given state can be computed from the state sequence vector, and the simulated error probability can be computed from the error vector. These calculations are carried out in the MATLAB program `c15_hmmtest.m`. Execution of these two programs using the default values produces the following output, in which the dialog relating to the acceptance or rejection of default values is suppressed:

```
≫ c15_errvector
≫ c15_hmmtest
Simulation results:
The probability of State 1 is 0.2432.
The probability of State 2 is 0.1634.
The probability of State 3 is 0.5934.
The error probability is 0.01445.
```

Note that the simulated values given for the state probabilities and the error probability are random variables. The variance of these random variables is reduced by increasing the length of the error vector.

A "sanity check" on the above values can be obtained by calculating the state probabilities and the error probability directly from the matrices B and A. The simple MATLAB code, together with the output, is as follows:

```
≫ A100 = A^100
A100 =
0.2353 0.1765 0.5882
0.2353 0.1765 0.5882
0.2353 0.1765 0.5882
≫ PE = B*A100'
PE =
0.9851 0.9851 0.9851
0.0149 0.0149 0.0149
```

Note that the theoretical state probabilities S_1, S_2, and S_3 are 0.2353, 0.1765, and 0.5882, respectively, and that the theoretical error probability is 0.0149. These obviously differ slightly from the simulated values, which, as previously mentioned, are random variables. The variance of these random variables can obviously be reduced, as the student should verify, by choosing a larger value of N. ■

15.4 Example HMMs—Gilbert and Fritchman Models

The characterization of discrete channels using discrete-time, finite-state Markov sequences has been proposed in the past by Gilbert, Fritchman, and others [11, 12]. The Gilbert model is a two-state model with a good error-free state, and a bad

state with an error probability of p. This model was used by Gilbert to calculate the capacity of a channel with burst errors. Parameters of the model can be estimated from measured or simulated data using the procedures that will be explored in the following section.

The Fritchman model, first proposed in 1967 [12], is now receiving considerable attention, since the Fritchman model appears to be suitable for modeling burst errors in mobile radio channels and also because it is relatively easy to estimate the parameters of the Fritchman model from burst error distributions. For binary channels, Fritchman's framework divides the state space into k good states and $N - k$ bad states. The good states represent error-free transmissions, and the bad states always produce a transmission error. Hence the entries in the B matrix are zeros and ones and they need not be estimated. A Fritchman model with three good states and one bad state is illustrated in Figure 15.9. This model has the state transition matrix

$$A = \begin{bmatrix} a_{11} & 0 & 0 & a_{14} \\ 0 & a_{22} & 0 & a_{24} \\ 0 & 0 & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \tag{15.47}$$

The B matrix for this case takes the very simple form

$$B = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{15.48}$$

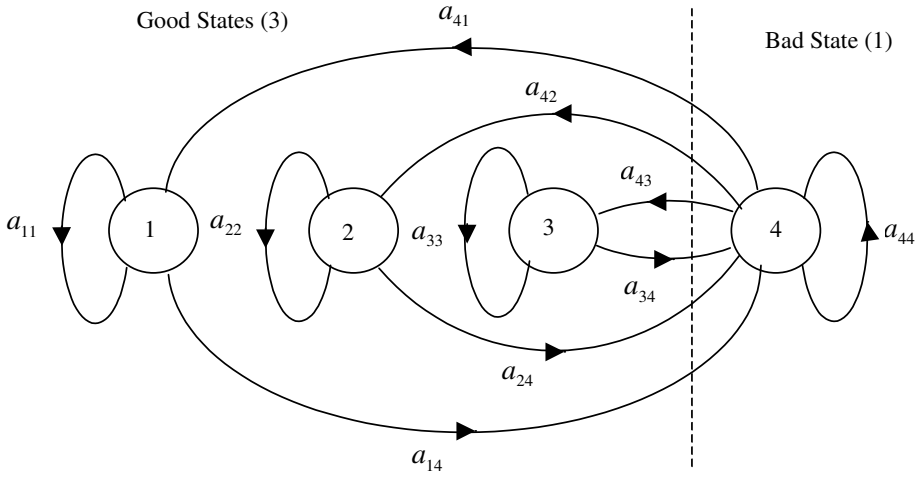


Figure 15.9 Fritchman model with three good states and one bad state.

Source: M. C. Jeruchim, P. Balaban, and K. S. Shanmugan, *Simulation of Communications Systems*, 2nd ed., New York: Kluwer Academic/Plenum Publishers, 2000.

In general, the state transition matrix A for the Fritchmann model can be partitioned as

$$A = \begin{bmatrix} A_{gg} & A_{gb} \\ A_{bg} & A_{bb} \end{bmatrix} \tag{15.49}$$

where the submatrices represent the transition probabilities between various good and bad states. For this model, it is possible to derive analytically the expressions for burst error distributions in terms of the model parameters and use these expressions to estimate the parameters of the model from empirical (measured or simulated) burst error distributions.

Let $\overline{O} = \{O_1, O_2, \cdots, O_T\}$ be an error sequence with $O_k = 1$ indicating that the k^{th} transmitted bit suffered a transmission error, and $O_k = 0$ indicating error-free transmission of the k^{th} symbol. Also, let the notation $(0^m|1)$ denote the event of observing m or more consecutive error-free transmissions following an error, and $(1^m|0)$ represent the event of observing m or more consecutive errors following an error-free transmission. Fritchman showed that the probabilities of occurrence of these two events are given by the sum of weighted exponentials

$$\Pr(0^m|1) = \sum_{i=1}^{k} f_i \lambda_i^{m-1} \tag{15.50}$$

and

$$\Pr(1^m|0) = \sum_{i=k+1}^{N} f_i \lambda_i^{m-1} \tag{15.51}$$

where $\lambda_i,\ i = 1, 2, \cdots, k$, and $\lambda_i,\ i = k+1, k+2, \cdots, N$, are the eigenvalues of A_{GG} and A_{BB}, respectively, and the corresponding values of f_i are functions of a_{ij}. From (15.50) and (15.51) we can obtain the probability of obtaining exactly m zeros (or ones) as

$$\Pr(0^{m-1}|1) - \Pr(0^m|1) = \sum_{i=1}^{k} f_i \lambda_i^{m-2}(1 - \lambda_i) \tag{15.52}$$

and

$$\Pr(1^{m-1}|0) - \Pr(1^m|0) = \sum_{i=k+1}^{N} f_i \lambda_i^{m-2}(1 - \lambda_i) \tag{15.53}$$

The Fritchman model can be interpreted as equivalent to a Markov process with a state transition probability matrix

$$\tilde{A} = \begin{bmatrix} \Lambda_{gg} & A_{gb} \\ A_{bg} & \Lambda_{bb} \end{bmatrix} \tag{15.54}$$

where Λ_{gg} and Λ_{bb} are diagonal matrices given by

$$\Lambda_{gg} = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_k \end{bmatrix} \tag{15.55}$$

and

$$\Lambda_{bb} = \begin{bmatrix} \lambda_{k+1} & 0 & \cdots & 0 \\ 0 & \lambda_{k+2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_N \end{bmatrix} \tag{15.56}$$

Note that in this equivalent model there are no transitions within the set of k good states and no transitions within the set of $N-k$ bad states. Such transitions are indistinguishable from the observed error sequence, since a transition from one good state to another good state still produces no errors and the transition is not observable from the output. This structure will be revisited when semi-Markov models are considered in a later section.

The Fritchman model is not unique except when there is only one bad state. This is so because, in the general case, the error-free run distribution $\Pr(0^m|1)$ and the error burst distribution $\Pr(1^m|0)$ do not specify the statistical dependence of the error-free runs and the error bursts. In the case of a single error state model, as in Figure 15.9, Fritchman showed that [12]

$$\Pr(0^m|1) = \sum_{k=1}^{N-1} \frac{a_{Nk}(a_{kk})^m}{a_{kk}} \tag{15.57}$$

From (15.57) it can be seen that, in the case of a single error state model, the error-free run uniquely specifies the $2(N-1)$ model parameters. An empirical procedure is used to fit an $N-1$ mixture of exponentials as in (15.50) and (15.51) to the (simulated or measured) error-free run distribution.

While Fritchman's model is applicable to discrete channels with simple burst error distributions, it may not be adequate to characterize very complex burst error patterns that will require more than one error state in the model. In such cases it will be very difficult to estimate the model parameters from the burst error distributions alone. Fortunately, it is possible to find a maximum likelihood estimate of the parameters using iterative techniques.

15.5 Estimation of Markov Model Parameters

The Markov model for a discrete channel is described by the $N \times N$ state transition matrix A and the $M \times N$ error probability generation matrix B. An iterative procedure for estimating these parameters $\Gamma = \{A, B\}$ from a given error sequence,

obtained through simulation or measurement, $\overline{O} = \{ O_1, , ..., O_t, ...O_T\}$ is based on the Baum-Welch algorithm [13]. This iterative algorithm is designed to converge to the maximum likelihood estimator of $\Gamma = \{A, B\}$ that maximizes $\Pr(\overline{O}|\Gamma)$.

The goal is to compute estimates of the elements of the state-transition matrix A. These are given by

$$\hat{a}_{i\,j} = \frac{\text{expected number of transitions from } i \text{ to } j}{\text{expected number of transitions from } i} \tag{15.58}$$

We also require the estimates of the elements of the error generation matrix, which are given by

$$\hat{b}_j(e_k) = \frac{\text{expected number of times } e_k \text{ is emitted from state } j}{\text{expected number of visits to state } j} \tag{15.59}$$

The calculations required to implement the Baum-Welch algorithm are described in the following steps. In addition, the MATLAB program for implementing the Baum-Welch algorithm is developed. Two versions of the MATLAB code for implementing the detailed calculations are given. The first version, denoted the "basic code," contains a number of looping operations and is therefore inefficient. It is, however, easier to relate the basic code to the defining equations. The second version, denoted the "more efficient version," eliminates some or all of the looping operations in order to take advantage of MATLAB's ability to implement vector operations. It should be noted that in several instances the code is not fully vectorized. This was done in order to have more readable code. The version of the Baum-Welch algorithm given in Appendix B was developed by combining the vectorized code segments.

Step 0: Start with an initial (assumed) model $\Gamma = \{A, B\}$.

Step 1: With $\Gamma = \{A, B\}$ as the model, we first compute the "forward variables"

$$\alpha_t(i) = \Pr[O_1, O_2, ...O_t, s_t = i \,|\, \Gamma] \tag{15.60}$$

and the "backward variables"

$$\beta_t \;=\; \Pr[O_{t+1}, O_{t+2}\,, ..., O_T \;|\; s_t = i, \Gamma] \tag{15.61}$$

for $t = 1, 2, ..., T$ and $i = 1, 2, .., N$. Details of these calculations follow.

Forward variables Calculation of the forward variables involves three steps: initialization, induction, and termination.

Initialization:

$$\alpha_1(i) = \pi_i b_i(O_1), \quad i = 1, 2, ...N \tag{15.62}$$

Induction:

$$\alpha_{\,t+1}(j) \;=\; \left[\sum_{i=1}^{N} \alpha_{\,t}(i) a_{ij}\right] b_j(O_{t+1}),$$

$$1 \le t \le T-1, \quad 1 \le j \le N \tag{15.63}$$

Termination:

$$\Pr[\overline{O} \,|\, \Gamma] = \sum_{i=1}^{N} \alpha_T(i)\beta_T(i) \tag{15.64}$$

Note that

$$\sum_{i=1}^{N} \alpha_T(i) = \sum_{i=1}^{N} \Pr[O_1, ..., O_T, s_T = i \,|\, \Gamma] = \Pr[\overline{O} \,|\, \Gamma] \tag{15.65}$$

The basic MATLAB code for performing these calculations follows. (Note the use of the scaling factor. The scaling factor is described in the following section.)

```
alpha = zeros(len,states);            % memory allocation
for column = 1:states
    alpha(1,column) = pye(1, column)*b(1, column);
hspace*50em% initialization
end
scale(1) = sum(alpha(1,:));           % normalizing factor
alpha(1,:)  = alpha(1,:)/scale(1,:);  % normalization
sum1 = 0;
for t = 1:(len - 1)                   % induction
    for j = 1:states
        for i = 1:states
            inner_sum = alpha(t,i)*p(i,j);
            sum1 = sum1 + inner_sum;
        end
        alpha(t+1,j) = sum1*b(out(t+1)+1,j);
        sum1 = 0;
    end
    scale(t+1) = sum(alpha(t+1,:));   % normalizing factor
    alpha(t+1,:)  = alpha(t+1,:)/scale(t+1); % normalization
end
```

More efficient MATLAB code results by eliminating several of the looping operations. This yields

```
alpha = zeros(len,states);            % memory allocation
alpha(1,:)  = pye.*b(1,:);            % initialization
scale(1) = sum(alpha(1,:));           % normalizing factor
alpha(1,:)  = alpha(1,:)/scale(1);    % normalization
for t = 1:len-1                       % induction
    alpha(t+1,:)  = (alpha(t,:)*p).*b(out(t+1)+1,:);
    scale(t+1) = sum(alpha(t+1,:));   % normalizing factor
    alpha(t+1,:)  = alpha(t+1,:)/scale(t+1); % normalization
end
```

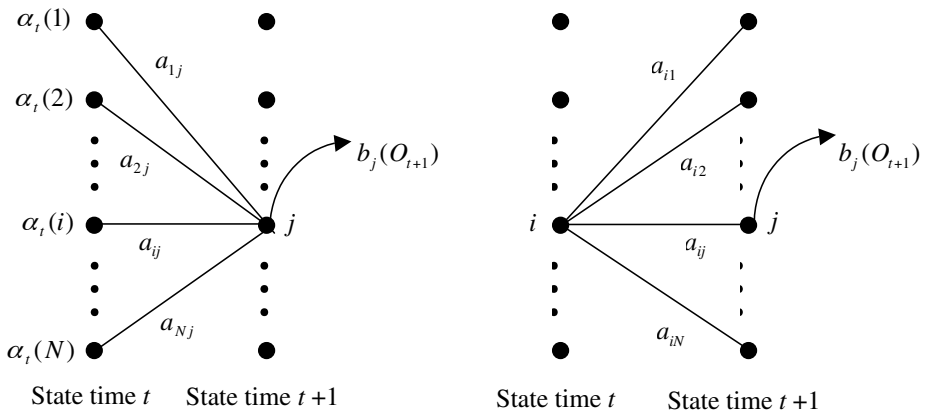


Figure 15.10 Parameter estimation for the HMM.

Source: M. C. Jeruchim, P. Balaban, and K. S. Shanmugan, *Simulation of Communications Systems*, 2nd ed., New York: Kluwer Academic/Plenum Publishers, 2000.

Backward variables Calculation of the backward variables involves two steps: initialization and induction.

Initialization:

$$\beta_T(i) = 1, \quad i = 1, 2, ..., N \tag{15.66}$$

Induction:

$$\beta_t(i) = \sum_{j=1}^{N} \beta_{t+1}(j) b_j(O_{t+1}) a_{ij}, \qquad 1 \le t \le T-1, \quad 1 \le j \le N \tag{15.67}$$

The details of these calculations are illustrated in Figure 15.10. The basic MATLAB code for this segment follows:

```
beta = zeros(len, states);      % memory allocation
beta(len,:)  = 1/scale(len);    % initialization and scaling
for t = (len-1):-1:1            % induction
    for i = 1:states
        for j = 1:states
            inner_sum = p(i,j)...
                * b(out(t+1)+1,j) * beta(t+1,j);
            sum2 = sum2 + inner_sum;
        end
        beta(t,i) = sum2;
```

```
        sum2 = 0;
    end
    beta(t,:)  = beta(t,:)/scale(t);   % scaling
end
```

The more efficient (partially vectorized) version follows:

```
beta = zeros(len, states);              % memory allocation
beta(len,:)  = 1/scale(len);            % initialization
for t = len-1:-1:1                      % induction
    beta(t,:)  = (beta(t+1,:).*b(out(t+1)+1,:))...
        *(p')/scale(t);
end
```

Step 2: The next step is to compute $\gamma_t(i)$ according to

$$\gamma_t(i) = \Pr[s_t = i \,|\, \overline{O}, \Gamma] = \frac{\alpha_t(i)\beta_t(i)}{\Pr[\overline{O} \,|\, \Gamma]}; \qquad i = 1, 2, ..., N \tag{15.68}$$

The basic MATLAB code for implementing this calculation follows:

```
gamma = zeros(len,states);              % memory allocation
for i = 1:len
for j = 1:states
gamma(i,j) = alpha(i,j)*beta(i,j); % calculation of gamma
                                   %    variable
end
gamma(i,:  = gamma(i,:)/sum(gamma(i.:));
end
```

We observe that we don't need to keep all of these values in memory, since they can be summed for the time indices. The following MATLAB code explains this further:

```
gamma_sum = zeros(1,states);             % memory allocation
for t = 1:len
gamma_sum = gamma_sum + alpha(t,:).*beta(t,:);
end
```

The quantity $\xi_t(i,j)$ is defined by

$$\xi_t(i,j) = \Pr[s_t = i, s_{t+1} = j \,|\, \overline{O}, \Gamma] = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{\Pr[\overline{O}|\Gamma]} \tag{15.69}$$

which can be calculated as follows:

```
prob_model_given_seq = zeros(1,len); % memory allocation
                                     %    for model
eta = zeros(states,states,len);      % memory allocation for eta
for t = 1:len                        % start the loop
    for i = 1:states
        prob_model_given_seq = zeros(1,len); % memory allocation
        temp(i) = alpha(t,i)*beta(t,i);
    end
    prob_model_given_seq(t) = sum(temp); % probability of model
end
for i = 1:states
    for j = 1:states
        for t = 1:(len-1)
            eta(i,j,t)=((alpha(t,i)*p(i,j)...
                *b(out(t+1)+1,j)*beta(t+1,j)));
        end
    end
end
```

Careful MATLAB programming indicates that we don't need to save this variable for all the time indices. Rather we can compute

```
eta = zeros(states,states);        % memory allocation
sum_eta = zeros(states,states)     % memory allocation
for t = 1:(len-1)
    for i = 1:states
        eta(i,:)= ((alpha(t,i)*(p(i,:).  ...
            *(b(out(t+1)+1,:))).*beta(t+1,:)));
    end
    sum_eta = sum_eta + eta; % adds the values of eta
end
```

We now determine the new state state transition probability $\widehat{a}_{ij}$ according to

$$\hat{a}_{i\,j} = \frac{\text{expected number of transitions from } i \text{ to } j}{\text{expected number of transitions from } i} = \frac{\sum\limits_{t=1}^{T-1} \xi_t(i,j)}{\sum\limits_{t=1}^{T-1} \gamma_t(i)} \tag{15.70}$$

The basic MATLAB code is

```
for i = 1:states
    for j = 1:states
        p_estimate(i,j) = sum(eta(i,j,:))/(sum(gamma(:,i))
            -gamma(len,i));
    end
```

```
    p_estimate(i,:)  = p_estimate(i,:)/sum...
        (p_estimate(i,:)); % normalization
end
```

Alternatively:

```
for i = 1:states
    for j = 1:states
        p_estimate(i,j) = sum_eta(i,j)...
            /(gamma_sum(i)-alpha(len,i).*beta(len,i)...
            /(sum(alpha(len,:).*beta(len,:))));
    end
    p_estimate(i,:)  = p_estimate(i,:)/sum(p_estimate(i,:));
end
```

Next, $\widehat{b}_j(e_k)$, defined by

$$\begin{aligned}\hat{b}_j(e_k) &= \frac{\text{expected number of times } e_k \text{ is emitted from state } j}{\text{expected number of visits to state } j} \\ &= \frac{\sum\limits_{t=1|\, O_t=e_k}^{T} \gamma_t(j)}{\sum\limits_{t=1}^{T} \gamma_t(j)}\end{aligned} \tag{15.71}$$

is computed. The basic MATLAB code is

```
for j = 1:states
    i = find(out==0); % find indices of correct bits
    for k = 1:length(i)
        sum_gamma = sum_gamma +gamma(i(k),j);
    end
    b(1,j) = sum_gamma/sum(gamma(:,j));
    sum_gamma = 0;
end
for j = 1:states
    ii = find(out==1); % find the indices of errors
    for k = 1:length(ii)
        sum_gamma = sum_gamma +gamma(ii(k),j);
    end
    b(2,j) = sum_gamma/sum(gamma(:,j));
    sum_gamma = 0;
end
for i = 1:states
    b(:,i) = b(:,i)/sum(b(:,i));
end
```

Using more efficient computations, the output symbol probability matrix can be estimated as

```
out_0 = find(out == 0);              % find correct bits indices
out_1 = find(out == 1);              % find indices of errors
sum_0 = zeros(1,states);
sum_1 = zeros(1,states);
gamma_sum = sum(gamma);
for i = 1:length(out_0)
    sum_0 = sum_0 + gamma(out_0(i),:);   % adds correct bits
end
for i = 1:length(out_1)
    sum_1 = sum_1 + gamma(out_1(i),:);   % adds error bits
end
for i = 1:states
    for j = 1:2
        if j == 1
            b(j,i) = sum_0(i)/gamma_sum(i);
                                     % elements b correct bits
        end
        if j == 2
            b(j,i) = sum_1(i)/gamma_sum(i);
                                     % elements b error bits
        end
    end
end
for i = 1:states
    b(:,i) = b(:,i)/sum(b(:,i));   % normalize the b matrix
end
```

We can also compute

$$\begin{aligned}\widehat{\pi}_i &= (\text{expected number of times in state } S_i \text{ at time } t = 1) \\ &= \alpha_1(i)\beta_1(i)\end{aligned} \tag{15.72}$$

Step 3: Go back to Step 1 with the new values of $\widehat{\Gamma} = \left\{\widehat{A}, \widehat{B}, \widehat{\pi}\right\}$, or equivalently $\widehat{\Gamma} = \Gamma$, obtained in Step 2 and repeat until the desired level of convergence, as discussed in a section to follow, is reached.

15.5.1 Scaling

The forward and backward vectors tend to zero exponentially for large data size and must be scaled properly in order to prevent numerical underflow. The scaling

constant, the use of which is implemented in the MATLAB code given in Appendix B, is first defined as

$$C_t = \sum_{i=1}^{N} \alpha_t(i) \tag{15.73}$$

The scaled values of $\alpha_t(j)$, denoted $\overline{\alpha_t}(j)$, are given by

$$\overline{\alpha_t}(j) = \alpha_t(j)/C_t \tag{15.74}$$

This, of course, implies that

$$\sum_{i=1}^{N} \overline{\alpha_t}(i) = 1 \tag{15.75}$$

The values of C_t are saved and used to scale the backward variables. The scaled values of $\beta_t(i)$, denoted $\overline{\beta_t}$, are given by

$$\overline{\beta_t}(i) = \beta_i(i)/C_t \tag{15.76}$$

with the initialization

$$\overline{\beta_T} = \frac{\mathbf{1}}{C_T}$$

where $\mathbf{1}$ denotes the column vector containing all ones. The gamma variable can also be normalized if desired, but scaling the gamma variables is not necessary. Turin [1] discusses scaling in more detail.

15.5.2 Convergence and Stopping Criteria

Since the Baum-Welch algorithm is iterative, the number of iterations to be performed for a required level of model accuracy must be determnined. Perhaps the best way to accomplish this is to display the estimates of A and B as the algorithm is executing. If one desires each element of A and B to be accurate to a given number of significant figures, execution of the algorithm is allowed to continue until the elements of A and B no longer change, within the given accuracy, from iteration to iteration. The algorithm is then terminated manually. This technique has considerable appeal, since the level of accuracy is known. Also, based on previous knowledge, one may simply perform a given number of iterations.

Another commonly used method for determining convergence is to continue the iterations until successive values of $\Pr\left[\overline{O}\,|\,\Gamma\right]$ differ very little. (The Baum-Welch algorithm is guaranteed to converge to a maximum likelihood solution. A proof of this statement is given in [1].) The value of $\Pr\left[\overline{O}\,|\,\Gamma\right]$ is determined in terms of the scaling constant C_t in (15.73). Specifically

$$\Pr\left[\overline{O}\,|\,\Gamma\right] = \prod_{t=1}^{T} C_t \tag{15.77}$$

For T large, this number will be very small and is usually expressed as

$$\log_{10} \Pr\left[\overline{O} \,|\, \Gamma\right] = \sum_{t=1}^{T} \log_{10} C_t \tag{15.78}$$

This is referred to as the log-likelihood ratio and is illustrated in Example 15.4.

It should be pointed out that the estimates of A and B, for a given set of data, are not unique unless the initial estimates of A, B, and Π are specified. Since the error vector is a function of both A and B, various combinations may produce statistically equivalent results and a specific result will depend on the initial conditions.

15.5.3 Block Equivalent Markov Models

The Baum-Welch algorithm is one of many reestimation algorithms available for the computation of model parameters based on an error vector. This error sequence could involve many thousands, or even millions, of symbols. Note that long observation periods are required to accurately estimate small values of a_{ij}. In communication systems, these small values are often critical for estimating system performance. The computational burden associated with the Baum-Welch algorithm is quite high when the error vector is long, since the forward and backward variables are computed for each symbol in the given error sequence. In addition, convergence is sometimes slow, which adds to the computational burden. For low error probability applications, the error vector has long runs of zeros. In addition, the error vector must contain a significant number of error events in order for the error vector to accurately define the model. In these situations the Baum-Welch algorithm is especially inefficient and faster algorithms are needed.

One method, proposed by Turin [14], for dealing with this problem, involves the computation of the forward and backward variables using a block matrix version of the form

$$A = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \tag{15.79}$$

The relationship between the terms in the matrix defined by (15.79) are illustrated in Figure 15.11. Note that the powers of submatrices involved in the computations can be precomputed and reused in order to reduce the overall computational burden.

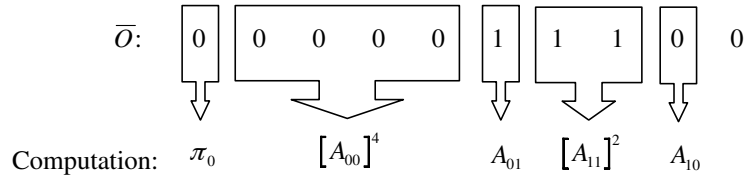


Figure 15.11 Computations for the sequence 0000011100 (Version 1).

Another modification proposed by Sivaprakasam and Shanmugan [6] is based on the fact that for a general Markov model there is a statistically equivalent Fitchman-like model with k good states and $N-k$ bad states and an A matrix having the form

$$A = \begin{bmatrix} \Lambda_{00} & A_{01} \\ A_{10} & \Lambda_{11} \end{bmatrix} \tag{15.80}$$

where Λ_{00} and Λ_{11} are diagonal matrices. These models are often referred to as block diagonal Markov models. With this model, the channel remains in the same state during an error burst and changes state only at the end of a burst. As a result, all variables are computed only at time steps involving the change of error symbol. In other words, variables are computed at the beginning of each burst of errors rather than once every symbol. With this modification, the computations take the form illustrated in Figure 15.12. Note that the computations during long bursts of ones and zeros now involve raising the power of a diagonal matrix rather than raising the power of arbitrary matrices, and matrix multiplications occur only at the transition from one burst to another. Thus, the computations are very efficient, and long error bursts can be processed without excessive storage and computational requirements.

Using this model we have interest only in the run lengths of zeros and ones. Thus, the error vector can be placed in a very compact form. As a simple illustration, consider an error sequence

$$E = [0000011100000000000000011000000000000000000000100] \tag{15.81}$$

The run-length vector corresponding to this error sequence can be written as

$$V = [0^5 1^3 0^{14} 1^2 0^{21} 10^2] \tag{15.82}$$

The MATLAB routine `c15_seglength.m`, given in Appendix D, generates the run-length vector defined by (15.83) from the error vector defined by (15.81).

The development of reestimation algorithms based on block equivalent Markov models will not be discussed here. They have, however, been extensively studied in the literature [1, 14, 15]. A MATLAB program, based on the work of Sivaprakasam and Shanmugan [15] is given in Appendix C. An example illustrating the use of this algorithm is presented in a following section.

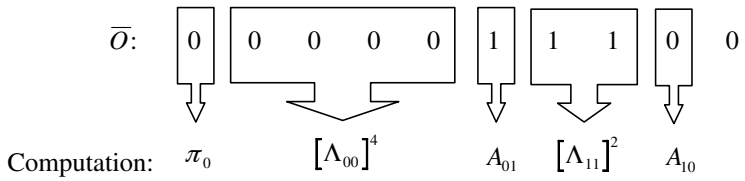


Figure 15.12 Computations for the sequence 0000011100 (Version 2).

Irrespective of the algorithms used, the Markov model for a discrete channel must be computed for different values of the parameters of the underlying physical channel. If the underlying physical channel is changed in any way, the discrete channel model must be reestimated. For example, if a discrete channel model is developed for a given channel, then the Markov model must be developed from measurements or waveform-level simulations for different values of the parameters of the channel including S/N. Thus a parametrized set of Markov models for the underlying channel can be developed and used for the design and analysis of error control coders, interleavers, etc.

15.6 Two Examples

We conclude this chapter with two examples that demonstrate the determination of a Markov channel and the determination of a semi-Markov model. The flow of these examples is as follows:

1. In the first example (Example 15.4), the error vector generated in Example 15.3 is used as input to the Baum-Welch algorithm. The result is an estimated channel model. The error probability and the run-length distribution resulting from this model are determined and are then compared to the error probability and run-length distribution of the original sequence generated in the first example.

2. The second example (Example 15.5) is similar to the first example. The essential difference is that a block equivalent semi-Markov model is used rather than a Markov model. The motivation for using a semi-Markov model is that the semi-Markov model leads to a substantial reduction in the time required to derive the model from measured or simulated data.

Example 15.4. In this example we generate an error sequence of length $N =$ 20,000 with a known two-state model defined by

$$A = \begin{bmatrix} 0.95 & 0.05 \\ 0.10 & 0.90 \end{bmatrix} \tag{15.83}$$

and

$$B = \begin{bmatrix} 0.95 & 0.50 \\ 0.05 & 0.50 \end{bmatrix} \tag{15.84}$$

The Baum-Welch algorithm is then used to estimate the model using the generated error sequence. Note that we assume a three-state model when the Baum-Welch model is applied. This may at first seem strange but, given simulation or measurement data, the channel model that produced the data is not known. The MATLAB dialog, with some of the nonessential dialog eliminated, is as follows:

```
≫ c15_errvector
Accept default values?
Enter y for yes or n for no > n
    Enter N, the number of points to be generated > 20000
    Enter A, the state transition matrix > [0.95 0.05; 0.1 0.9]
    Enter B, the error distribution matrx > [0.95 0.5; 0.05 0.5]
≫ out1 = out;
≫ c15_bwa(20,3,out)
Enter the initial state transition matrix P >
        [0.9 0.05 0.05; 0.1 0.8 0.1; 0.1 0.2 0.7]
Enter the initial state probability vector pye > [0.3 0.3 0.4]
Enter the initial output symbol probability matrix...
    B > [0.9 0.8.7; 0.1 0.2 0.3]
```

After one iteration we have[4]

$$\widehat{A}_1 = \begin{bmatrix} 0.9051 & 0.0469 & 0.0481 \\ 0.0991 & 0.7931 & 0.1078 \\ 0.0895 & 0.1856 & 0.7249 \end{bmatrix} \qquad \widehat{B}_1 = \begin{bmatrix} 0.9206 & 0.7462 & 0.5848 \\ 0.0794 & 0.2538 & 0.4152 \end{bmatrix}$$

Also, after 10 iterations, the estimated state transition matrix is

$$\widehat{A}_{10} = \begin{bmatrix} 0.9415 & 0.0300 & 0.0285 \\ 0.1157 & 0.7504 & 0.1340 \\ 0.0669 & 0.1382 & 0.7949 \end{bmatrix} \qquad \widehat{B}_{10} = \begin{bmatrix} 0.9547 & 0.6745 & 0.4257 \\ 0.0453 & 0.3255 & 0.5743 \end{bmatrix}$$

At the 20^{th} iteration, which is the termination point, the estimated state transition matrix is

$$\widehat{A}_{20} = \begin{bmatrix} 0.9437 & 0.0299 & 0.0263 \\ 0.1173 & 0.7425 & 0.1402 \\ 0.0663 & 0.1324 & 0.8013 \end{bmatrix} \qquad \widehat{B}_{20} = \begin{bmatrix} 0.9522 & 0.6637 & 0.4313 \\ 0.0478 & 0.3363 & 0.5687 \end{bmatrix}$$

The log likelihood function is illustrated in Figure 15.13. It can be seen that the log likelihood function converges in about 10 iterations. Note that this conclusion is consistent with the preceding computations of A_k and B_k for $k = 10$ and $k = 20$. Note also that, as discussed previously, the likelihood numbers are very small.

We now generate a second sequence, `out2`, using the model estimated by the Baum-Welch algorithm. This gives, with the dialog dealing with default values suppressed:

[4]It should be mentioned that rerunning this program may result in estimates for A and B that are different from the results given here, even though the same parameters and initial conditions are used. The reason for this lies in the fact that the error vector, upon which the estimates of A and B are based, is a sample function of a random process. In addition, the rows of $\widehat{A}$ and the columns of $\widehat{B}$ may not sum to one when the elements of $\widehat{A}$ and $\widehat{B}$ are represented with finite precision.

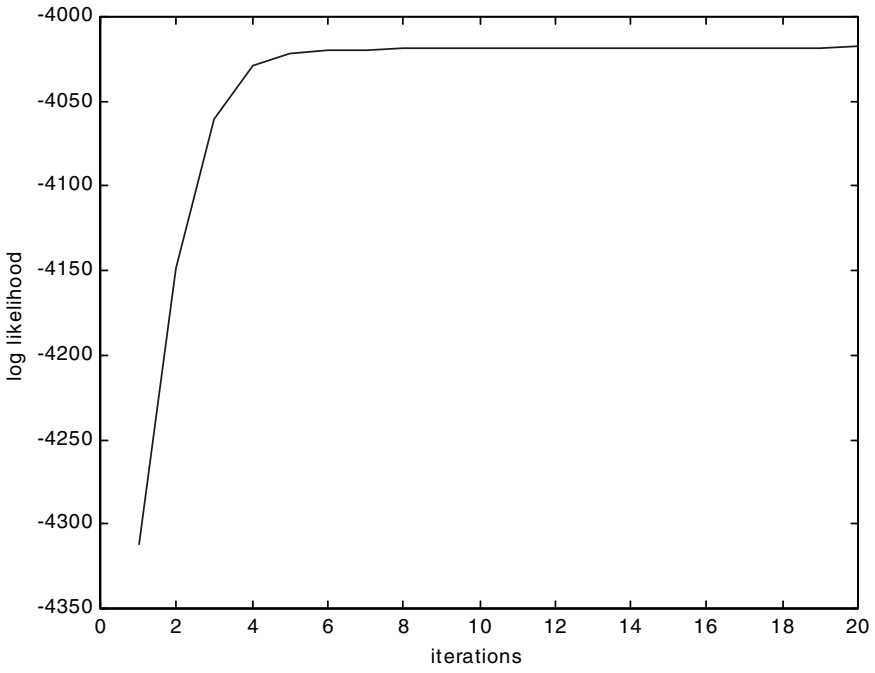


Figure 15.13 Log likelihood function for Example 15.4.

```
≫ a = [0.9437 0.0299 0.0263; 0.1173 0.7425 0.1402;...
    0.0663 0.1324 0.8013];
≫ b = [0.9522 0.6637 0.4313; 0.0478 0.3363 0.5687];
≫ c15_errvector
Accept default values?
Enter y for yes or n for no > n
    Enter N, the number of points to be generated > 20000
    Enter A, the state transition matrix > a
    Enter B, the error distribution matrix > b
≫ out2 = out;
```

In order to calculate and plot $\Pr\{0^m|1\}$ for both `out1` and `out2`, we execute the following lines of MATLAB code:

```
≫ runcode1=c15_seglength(out1);
≫ runcode2=c15_seglength(out2);
≫ c15_intervals2(runcode1,runcode2)
```

This produces the plots illustrated in Figure 15.14. The top frame illustrates $\Pr\{0^m|1\}$ for the original data and the bottom frame illustrates $\Pr\{0^m|1\}$ using the data generated by the model estimated by the Baum-Welch algorithm.

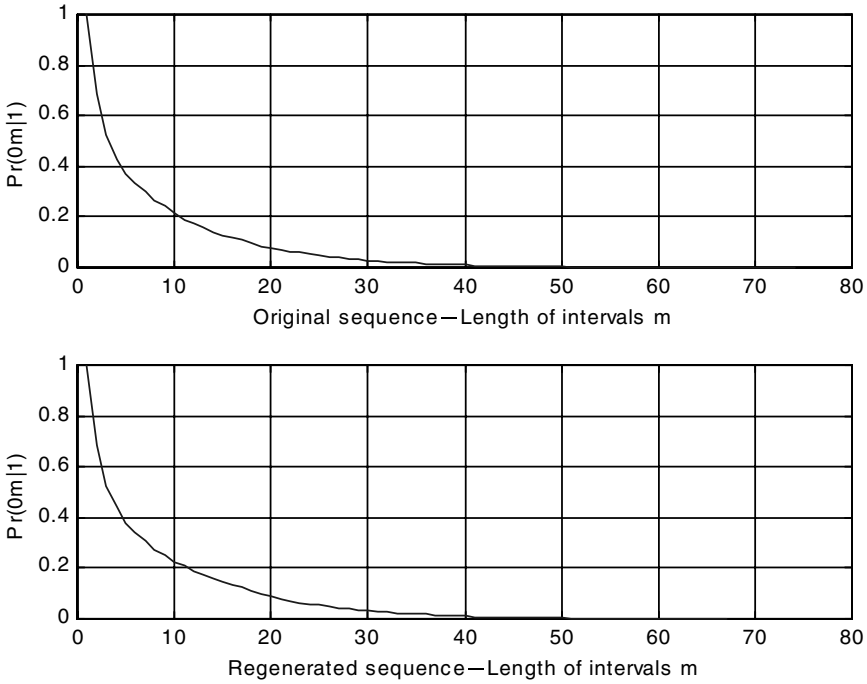


Figure 15.14 $\Pr(0^m|1)$ for the original data and the data resulting from the estimated model.

There are a few obvious differences between the two plots illustrated in Figure 15.14, especially in the neighborhood of $m = 10$. Note, however, that the error probabilities are given by

```
≫ p1 = sum(out1)/N
p1 =
0.0038
≫ p2 = sum(out2)/N
p2 =
0.0035
```

■

Example 15.5. In this final example, we consider a test case used by Sivaprakasam and Shanmugan [6] to illustrate the block diagonal Markov model. Assume a three-state model having two good states and one bad state, defined by[5]

[5]Note that A represents a physical channel and does not have a block diagonal form. The estimated state transition matrix $\widehat{A}$ will have a block diagonal form.

$$A = \begin{bmatrix} 0.90 & 0.02 & 0.08 \\ 0.10 & 0.50 & 0.40 \\ 0.10 & 0.20 & 0.70 \end{bmatrix} \tag{15.85}$$

Since the good states are error-free and errors are always produced in the bad state, the error observation matrix is defined by

$$B = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{15.86}$$

Note that this model is semi-hidden. If an error occurs, we know that the error was generated by state 3. If, however, no error is generated, the state cannot be identified.

First, we generate an error vector, having length 20,000, for the state transition matrix A and the error observation matrix B. The appropriate MATLAB commands, with the nonessential dialog supressed, follows:

```
≫ a1 = [0.90 0.02 0.08; 0.10 0.50 0.40; 0.10 0.20 0.70];
≫ b1 = [1 1 0; 0 0 1];
≫ c15_errvector
Accept default values?
Enter y for yes or n for no > n
Enter N, the number of points to be generated > 20000
Enter A, the state transition matrix > a1
Enter B, the error distribution matrix > b1
≫ out1 = out;
≫ runcode1 = c15_seglength(out1);
≫ [A_matrix, pi_est] = c15_semiMarkov(runcode1,100,[2 1])
```

Note that we must run `c15_seglength` prior to running `c15_semiMarkov` in order to generate the run-length vector.

After 50 iterations, the estimate of the state transition matrix is

$$\widehat{A} = \begin{bmatrix} 0.9043 & 0.0 & 0.0957 \\ 0.0 & 0.4911 & 0.5089 \\ 0.1402 & 0.1515 & 0.7083 \end{bmatrix} \tag{15.87}$$

Note that this is close to, but not exactly equal to, the result obtained by Sivaprakasam and Shanmugan [6]. There are several explanations for the differences. First, the error vector upon which the estimated model is based is a sample function of a stochastic process and, for practical purposes, no two error vectors will be identical. Also, error vectors of different lengths will result in slightly different models. Finally, we fixed the number of iterations, and Sivaprakasam and Shanmugan used a different stopping criteria.

The next step is to generate a second error vector based on the estimated model and plot $\Pr\{0^m|1\}$ for both the original error vector and for the error vector generated by the estimated model. This is accomplished using the following MATLAB code:

```
≫ a2 = [0.9043 0.0 0.0957; 0.0 0.4911 0.45089; 0.1402 0.1515 0.7083];
≫ b2 = [1 1 0; 0 0 1];
≫ c15_errvector
Accept default values?
Enter y for yes or n for no > n
Enter N, the number of points to be generated > 20000
Enter A, the state transition matrix > a2
Enter B, the error distribution matrix > b2
≫ out2 = out;
≫ runcode2 = c15_seglength(out2);
≫ intervals2(runcode1,runcode2);
```

The results are illustrated in Figure 15.15. Note that the results are in close agreement even though the original model, defined by A, and the estimated model, defined by $\widehat{A}$, are quite different. Obviously we knew from the start of this problem that A and $\widehat{A}$ would be different, since $\widehat{A}$ is constrained to have a block diagonal form.

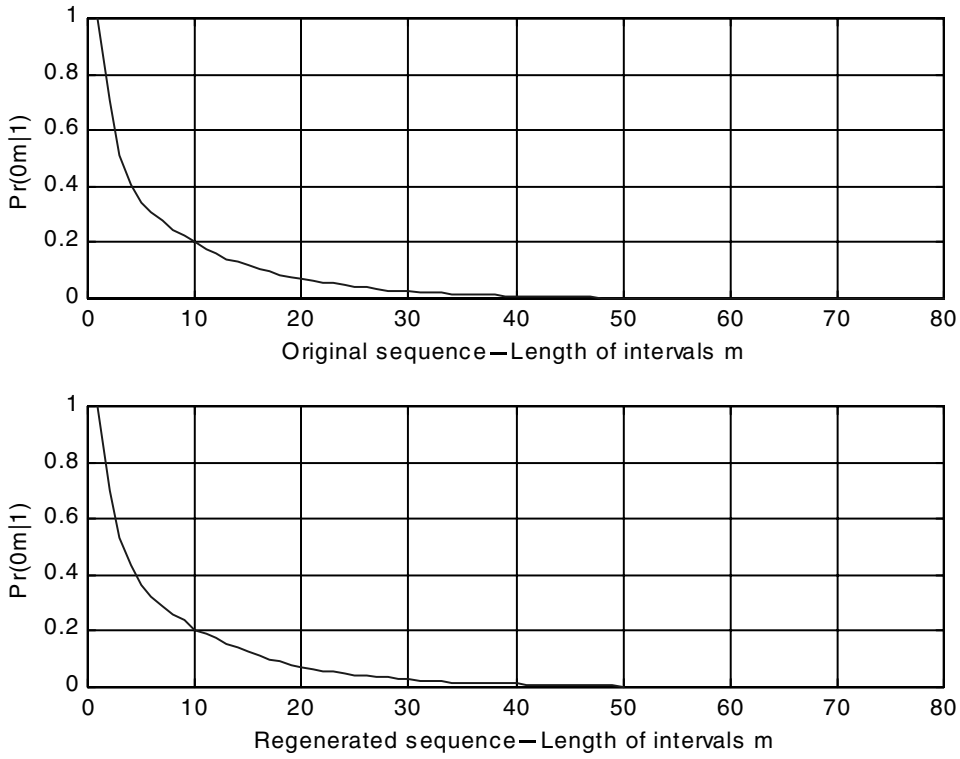


Figure 15.15 $\Pr\{0^m|1\}$ for Example 15.5.

It is also instructive to check the error probabilities. In order to accomplish this we compute

```
≫ pe1 = sum(out1)/20000
pe1 =
0.3617
≫ pe2 = sum(out2)/20000
pe2 =
0.3538
```

Once again we have close agreement. ■

15.7 Summary

The focus of this chapter was the development of discrete channel models. Discrete channel models are attractive, since their use greatly reduces the computational complexity associated with the execution of a simulation. The parameters of the model may be determined from either measured data or from the results of a waveform-level simulation. Discrete channel models have found widespread use in wireless communication systems, since they can be used to effectively model fading channels. Discrete channel models are an abstraction of waveform-level models in that they characterize the input-output characteristics of the channel but do not model the physical functionality of the channel. It is through this abstraction that the computational burden is reduced.

A two-state model was initially considered in order to establish the concept of a state-transition matrix, the state distribution vector, and the error generation matrix. Several different techniques for determining the steady-state distribution matrix were considered. These concepts were then extended to the N-state model including the development of a technique for efficiently simulating a channel based on a discrete channel model.

We next considered the two-state Gilbert model and the N-state Fritchman model as examples of a hidden Markov model. The Fritchman model is attractive from a computational point of view, since the state-transition matrix contains a large number of zeros (relatively sparse). The Fritchman model has also been shown to effectively model the fading channel environments encountered in wireless communication systems.

The heart of this chapter dealt with the development of discrete channel models based on measured data or data obtained from a waveform-level simulation. The basic tool used for parameter estimation is the Baum-Welch algorithm. The development of the Baum-Welch algorithm was briefly outlined and the development of a MATLAB implementation of the Baum-Welch algorithm was presented. Scaling to prevent underflow was discussed, as was the block equivalent Markov model, which can be estimated more efficiently than the general Markov model.

The chapter concluded with three examples summarizing techniques for estimating both Markov models and block-equivalent Markov models. The first example focuses on the generation of an error vector given a Markov model. Validation of

the result is based on the probability of state occupancy and the probability of error. The second example uses the Baum-Welch algorithm to estimate a channel model. Validation includes a comparison of the run-length statistic $\Pr\{0^m|1\}$ for both the original error vector and the error vector produced by the estimated model. The third example is similar to the second example except that a block equivalent Markov model is used. These three examples should serve as a guide to the use of the tools developed in this chapter.

15.8 Further Reading

The development and use of Markov models for burst error channels (channels with memory) are currently an active area of research and new papers appear frequently. The interested reader may find recent articles on this topic in *IEEE Transactions on Communications*, *IEEE Transactions on Vehicular Technology*, and other related journals and conference proceedings. Markov models are also being developed for burst errors at the higher layers of communication networks. For example, packet errors at the transport layer in a network. The book by Turin [1] is recommended as a comprehensive treatment of the subject.

15.9 References

1. W. Turin, *Digital Transmission Systems: Performance, Analysis and Modeling*, 2nd ed., New York: McGraw-Hill, 1999.

2. L. R. Rabiner and B. H. Juang, "An Introduction to Hidden Markov Models," *IEEE ASSP Magazine*, January 1986, pp. 4–16.

3. R. A. Howard, *Dynamic Probabilistic Systems, Vol. I: Markov Models*, New York: Wiley, 1971.

4. R. A. Howard, *Dynamic Probabilistic Systems, Vol. II: SemiMarkov and Decision Processes*, New York: Wiley, 1971.

5. H. S. Wang and N. Moayeri, "Finite-State Markov Channels—A Useful Model for Radio Communications Channels," *IEEE Transactions on Vehicular Technology*, Vol. 44, February 1995, pp. 163–171.

6. S. Sivaprakasam and K. S. Shanmugan, "An Equivalent Model for Burst Errors in Digital Channels," *IEEE Transactions on Communications*, Vol. 43, No. 2/3/4, February/March/April 1995, pp. 1347–1355.

7. H. S. Wang and P.-C. Chang, "On Verifying the First-Order Markovian Assumption for a Rayleigh Fading Channel Model," *IEEE Transactions on Vehicular Technology*, Vol. 45, May 1996, pp. 353–357.

8. W. Turin and R. van Nobelen, "Hidden Markov Modeling of Flat Fading Channels," *IEEE Journal on Selected Areas in Communications*, Vol. 16, No. 9, December 1998, pp. 1809–1817.

9. F. Babich, O. E. Kelly, and G. Lombardi, "Generalized Markov Modeling for Flat Fading," *IEEE Transactions on Communications*, Vol. 48, No. 4, April 2000, pp. 347–351.

10. M. Zorzi, R. R. Rao, and L. B. Milstein, "A Markov Model for Block Errors on Fading Channels," *PIMRC '96 Conference Record*, October 1996.

11. E. N. Gilbert, "Capacity of a Burst-Noise Channel," *Bell System Technical Journal*, Vol. 39, September 1960, pp. 1253–1266.

12. B. D. Fritchman, "A Binary Characterization Using Partitioned Markov Chains," *IEEE Transactions on Information Theory*, Vol. IT-13, No. 2, April 1967, pp. 221–227.

13. L. E. Baum, T. Petrie, G. Soules, and N. Weiss, "A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains," *Annals of Mathematical Statistics*, Vol. 41, Issue 1, February 1970, pp. 164–171.

14. W. Turin and M. M. Sondhi, "Modeling Error Sources in Digital Channels," *IEEE Journal on Selected Areas in Communications*, Vol. 11, No. 3, April 1993, pp. 340–347.

15.10 Problems

15.1 Assume that a binary symmetric channel is defined by the error probability $p = 10^{-2}$. Using the technique defined by (15.2), with p_k independent of k, simulate the transmission of N symbols through the channel for $N = 100$, $N = 1{,}000$ and $N = 10{,}000$. In each case count the number of errors that actually occur and calculate the BER resulting from the simulation. Repeat this experiment ten times for each value of N. What do you conclude?

15.2 Rework Example 15.1 assuming that

$$\Pi_0 = \begin{bmatrix} 0.1 & 0.9 \end{bmatrix}$$

15.3 Repeat the preceding problem assuming that the initial state probability distribution

$$\Pi_0 = \begin{bmatrix} 0.4 & 0.3 & 0.3 \end{bmatrix}$$

and the state transition matrix

$$A = \begin{bmatrix} 0.8 & 0.15 & 0.05 \\ 0.25 & 0.7 & 0.05 \\ 0.1 & 0.1 & 0.8 \end{bmatrix}$$

15.4 A Markov process has the state transition matrix

$$A = \begin{bmatrix} 0.8 & 0.15 & 0.05 \\ 0.2 & 0.7 & 0.1 \\ 0 & 0.1 & 0.9 \end{bmatrix}$$

Determine the steady-state distribution Π_{ss} using the technique illustrated in Example 15.1. Verify the result by raising A to a sufficiently high power.

15.5 We know that Π_{ss} can be determined by raising the state transition probability matrix to a high power. As yet another method for solving this problem, recognize that $\Pi_{ss} = \Pi_{ss}A$ implies that Π_{ss} is an eignvector of A. Using this observation, develop a MATLAB program for determining Π_{ss} given the state transition matrix A. Apply this technique to

$$A = \begin{bmatrix} 0.8 & 0.1 & 0.1 \\ 0.2 & 0.6 & 0.2 \\ 0.1 & 0.2 & 0.7 \end{bmatrix}$$

15.6 Develop a MATLAB program to generate a Markov sequence based on the state transition matrix

$$A = \begin{bmatrix} 0.8 & 0.15 & 0.05 \\ 0.2 & 0.7 & 0.1 \\ 0 & 0.1 & 0.9 \end{bmatrix}$$

Assuming that the system is initially in state 1, generate 200 state transitions and develop a figure that illustrates the state of the channel for each time step. Next generate 40,000 state transitions and, from the simulation results, compute the steady-state probabilities of each state. Verify the result by computing A^k for sufficiently large k.

15.7 Consider the channel model illustrated in Figure 15.16. The probability of error given that the channel is in the good state is 0 and the probability of error given that the channel is in the bad state is 1.

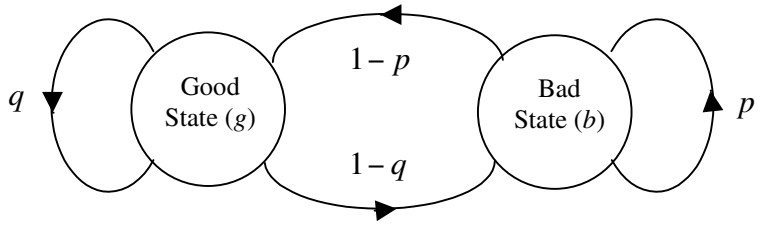


Figure 15.16 Channel model for Problem 15.7.

(a) Generate a 10,000-bit sequence with $p = 0.01$ and $q = 0.99$. Construct a histogram of the burst length and the inter-error gap (error-free run) distributions.

(b) Discuss how p and q can be estimated emperically from a histogram of (simulated) burst length and inter-error gap distributions. Estimate p and q.

(c) Use the Baum-Welch algorithm to estimate p and q. Compare with the empirical estimates obtained from the histograms.

15.8 Develop a figure similar to Figure 15.9 for the Fritchman model

$$A = \begin{bmatrix} a_{11} & 0 & 0 & a_{14} & a_{15} \\ 0 & a_{22} & 0 & a_{24} & a_{25} \\ 0 & 0 & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & 0 \\ a_{51} & a_{52} & a_{53} & 0 & a_{55} \end{bmatrix}$$

15.9 Generalize `c15_hmmtest.m` for a N-state model.

15.10 Rework Example 15.4 by estimating a two-state model. Compare the estimated model with the original two-state model. Plot the estimated parameters as a function of the number of iterations up to 20 iterations. Discuss the convergence.

15.11 Rework Example 15.4 using, as initial conditions to the Baum-Welch algorithm, the matrices A and B that were used to generate the original error vector. Discuss the convergence.

15.12 Rework Example 15.4 with $N = 4$, 3, and 2, where N is the number of states in the stimated model. Compare the likelihood metric after convergence is reached. Use as input to the Baum-Welch algorithm the data generated by the two-state model used in Example 15.2. Work this problem several times using different initial conditions for the Baum-Welch algorithm and discuss the results.

15.13 Modify the Baum-Welch algorithm given in Appendix A (`c15_bwa.m`) so that one does not have to enter initial guesses for the matrices A, B, and π. Make the elements of A and B random but reasonable. (For example, the on-diagonal elements of the A matrix are typically larger than the off-diagonal terms.) For the initial π matrix make all values equal. Test your resulting program by reworking Example 15.4 several times. What do you observe?

15.14 Modify the MATLAB code for Example 15.4 so that the CPU time spent in the model estimation algorithm can be measured. After this is accomplished, rework Example 15.4 for error vectors having length $N = 20{,}000$ and 100,000. Compare the times required to estimate the model. Also compare the error probabilities.

15.15 Rework Example 15.5 assuming two good states and two bad states.

15.16 Assume that a discrete channel is defined by the state transition matrix

$$A = \begin{bmatrix} 0.990 & 0.001 & 0.003 & 0.006 \\ 0.120 & 0.600 & 0.180 & 0.100 \\ 0.150 & 0.200 & 0.500 & 0.150 \\ 0.040 & 0.030 & 0.130 & 0.800 \end{bmatrix}$$

Determine an equivalent block diagonal Markov model having two good states and two bad states. (Note: This channel model represents another test case presented by Sivaprakasam and Shanmugan [6]. The student should compare the result with that found by Sivaprakasam and Shanmugan.)

15.11 Appendix A: Error Vector Generation

15.11.1 Program: c15_errvector.m

```
% File: c15_errvector.m
disp(' ')
disp('Default values are:')
N = 20000                                          % default N
A = [0.8 0.1 0.1; 0.2 0.6 0.2; 0.02 0.08 0.90]  % default A
B = [0.999 0.95 0.99; 0.001 0.05 0.01]           % default B
disp(' ')
disp('Accept default values?')
dtype = input('Enter y for yes or n for no > ','s');
if dtype == 'n'
    N = input(' Enter N, the number of points to be generated > ');
    A = input(' Enter A, the state transition matrix > ');
    B = input(' Enter B, the error distribution matrix > ');
end
state = 1; % initial state
total_states = size(A,1);
out = zeros(1,N); % initialize error vector
state_seq = zeros(1,N); % initialize state sequence
h = waitbar(0,'Calculating Error Vector');
%
ran_b = rand(1);                        % get random number
if ran_b>B(1,state)                     % test for error
    out(1) = 1;                         % record error
end
state_seq(1) = state;                   % record state
for t=2:N
    u1 = rand(1);                       % get random number
    cum_sum = [0 cumsum(A(state,:))];
    for i=1:total_states                % loop to determine new state
        if u1 >= cum_sum(i) & u1 < cum_sum(i+1);
            state = i;                  % assign new state
        end
    end
    state_seq(t) = state;               % new record state
    u2 = rand(1);                       % get random number
   if u2 > B(1,state)}
        out(t) = 1;                     % record error
    end
 waitbar(t/N)
end
close(h)
% End of script file.
```

15.11.2 Program: c15_hmmtest.m

Note: This program is for a three-state model. It is easily generalized for an arbitrary number of states.

```
% File: c15_hmmtest.m
pe = sum(out)/N;
state_sum = zeros(1,total_states);
for k=1:N
    if state_seq(k)==1
        state_sum(1)=state_sum(1)+1;
    end
    if state_seq(k)==2
       state_sum(2)=state_sum(2)+1;
    end
    if state_seq(k)==3
        state_sum(3)=state_sum(3)+1;
    end
end
a = ['The probability of State 1 is ',num2str(state_sum(1)/N),'.'];
b = ['The probability of State 2 is ',num2str(state_sum(2)/N),'.'];
c = ['The probability of State 3 is ',num2str(state_sum(3)/N),'.'];
d = ['The error probability is ',num2str(pe),'.'];
disp('Simulation results:')
disp(a) % display probability of state 1
disp(b) % display probability of state 2
disp(c) % display probability of state 3
disp(d) % display error probability
% End script file.
```

15.12 Appendix B: The Baum-Welch Algorithm

```
% File: c15_bwa.m
function [p, pye, b] = c15_bwa(iterations,states,out);
len = length(out);
p = input('Enter the initial state transition matrix P > ');
pye = input('Enter the initial state probability vector pye >');
b = input('Enter the initial output symbol probability matrix B > ');
alpha=zeros(len,states); beta=zeros(len,states);
eta=zeros(states,states); gamma=zeros(1,states); scale=zeros(len,1);
log_likelihood = zeros(1,iterations);
iplot = 1;                                  % plot switch
%
p                                           % display initial p
pye                                         % display initial pye
%
pye_rec = zeros(states,1);
pye_rec(:,1) = pye';
sum_gamma = 0;
sum_eta = 0;
%
for cycle = 1:iterations
    cycle                                   % display iteration index
    %
    % alpha generation
    %
    alpha(1,:) = pye.*b(1,:);
    scale(1) = sum(alpha(1,:));
    alpha(1,:) = alpha(1,:)/scale(1);
    for t = 2:len
        alpha(t,:) = (alpha(t-1,:)*p).*b(out(t)+1,:);
        scale(t) = sum(alpha(t,:));
        alpha(t,:) = alpha(t,:)/scale(t);
    end
    %
    % beta generation
    %
    beta(len,:) = 1/scale(len);
    for t = len-1:-1:1
        beta(t,:) = (beta(t+1,:).*b(out(t+1)+1,:))*(p')/scale(t);
    end
    %
    % eta generation
    %
    sum_eta = zeros(states);
```

```
for t = 1:len-1
    for i = 1:states
        eta(i,:) = ...
            ((alpha(t,i)*(p(i,:).*(b(out(t+1)+1,:))). ...
                *beta(t+1,:)))...
            /sum(alpha(t,:).*beta(t,:));
    end
    sum_eta = sum_eta + eta;
end
%
% gamma generation
%
gamma_sum = zeros(1,states);
for t = 1:len
    gamma_sum = gamma_sum + alpha(t,:).*beta(t,:);
end
%
% calculate and display the log_likelihood function
%
loglikelihood = sum(log10(scale));
log_likelihood(cycle) = loglikelihood;
loglikelihood                                    % display result
%
% Re-estimation of the intial state probability vector pye
%
pye(1,:) = alpha(1,:).*beta(1,:)/sum(alpha(1,:).*beta(1,:));
%
pye % display pye
%
pye_rec(:,cycle+1) = pye';                      % Save for plot
%
% Re-estimation of the state transition matrix P
%
for i = 1:states
    for j = 1:states
        p_estimate(i,j) = ...
            sum_eta(i,j)/(gamma_sum(i)-alpha(len,i). ...
                *beta(len,i)...
            /(sum(alpha(len,:).*beta(len,:))));
        end
    p_estimate(i,:) = p_estimate(i,:)/sum(p_estimate(i,:));
end
%
p = p_estimate                                  % display p
%
```

```
    % Re-estimation of output symbol probability matrix B
    %
    out_0 = find(out == 0);
    out_1 = find(out == 1);
    sum_0 = zeros(1,states);
    sum_1 = zeros(1,states);
    for i = 1:length(out_0)
        sum_0 = sum_0 + alpha(out_0(i),:).*beta(out_0(i),:)...
            /sum(alpha(out_0(i),:).*beta(out_0(i),:));
    end
    for i = 1:length(out_1)
        sum_1 = sum_1 + alpha(out_1(i),:).*beta(out_1(i),:)...
            /sum(alpha(out_1(i),:).*beta(out_1(i),:));
    end
    for i = 1:states
        for j = 1:2
            if j == 1
                b(j,i) = sum_0(i)/gamma_sum(i);
            end
            if j == 2
                b(j,i) = sum_1(i)/gamma_sum(i);
            end
        end
    end
    for i = 1:states
        b(:,i) = b(:,i)/sum(b(:,i));
    end
    b                                                % display b
end
if iplot==1
    plot(1:iterations,log_likelihood)
    xlabel('iterations')
    ylabel('log likelihood')
end
% End of function file.
```

15.13 Appendix C: The Semi-Hidden Markov Model

```
% File: c15_semiMarkov.m
function [A_matrix, pi_est] = c15_semiMarkov(runlength,cycles,...
    partition)
% runlength = runlength code
% cycles = number of iterations
% partition = 1 by 2 vector = [good states, bad states]
%
[symbols len_symbols] = size(runlength); % gets size of runlength
                                         %     vector
m = runlength(1,:);      % the first bit received is error free
u = runlength(2,:);      % arbitrary number of elements
C = len_symbols;         % total length of runlength vector
%
A = cell(length(partition));   % a 2x2 array only 2 symbols
pye = rand(1,sum(partition));  % initialize the initial state vector
pye = pye(1,:)/sum(pye(1,:));  % normalize
pi_u1 = pye(1:partition(1));
%
% initialize A matrix
%
A = cell(partition);      % allocate memeory for the A matrix
A00 = diag(1 - abs(randn(partition(1),1)/1000));   % initialize A00
                                                   %     matrix
A10 = rand(partition(2),partition(1));             % initialize A10
                                                   %     matrix
A01 = rand(partition(1),partition(2));             % initialize the
                                                   %     A01 matrix
A11 = diag(1 - abs(randn(partition(2),1)/1000));   % initialize A11
                                                   %     matrix
A{1,1} = A00; A{1,2} = A01; A{2,1} = A10; A{2,2} = A11;
A_matrix = [A{1,1} A{1,2};A{2,1} A{2,2}];          % cell-array in
                                                   %     matrix form
%
for i = 1:sum(partition)
    A_matrix(i,:) = A_matrix(i,:)/sum(A_matrix(i,:)); % normalize the
                                                      %     A matrix
end
A_matrix;
%
A{1} = A_matrix(1:partition(1),1:partition(1));
A{2} = A_matrix(partition(1)+1:partition(1)+partition(2),1:...
    partition(1));
A{3} = A_matrix(1:partition(1),partition(1)+1:partition(1)...
```

```
    +partition(2));
A{4} = A_matrix(partition(1)+1:partition(1)+...
    partition(2),partition(1)+1:partition(1)+partition(2));
%
for iterations = 1:cycles
    %
    % alpha generation
    %
    alpha{1} = pi_u1*(A{u(1)+1,u(1)+1}.^(m(1)-1)); scale(1)= ...
        sum(alpha{1});
    alpha{1}= alpha{1}/scale(1); % normalization
    for c = 2:C
        alpha{c}= alpha{c-1}*A{u(c-1)+1,u(c)+1}*A{u(c)+1,...
            u(c)+1}^(m(c)-1);
        scale(c)= sum(alpha{c}); % scaling factor
        alpha{c}= alpha{c}/scale(c); % normalize alpha
    end;
    %
    % beta generation
    %
    beta{C}= ones(partition(u(C)+1),1)/scale(C); % last element of
                                                 %     beta
    for(c= C-1:-1:1)
        beta{c}= A{u(c)+1,u(c+1)+1}*(A{u(c+1)+1,u(c+1)+1}...
            ^(m(c+1)-1))*beta{c+1}/scale(c);
    end;
    %
    % gamma generation
    %
    Gamma{1} = alpha{1}.*beta{1}';
    Gamma{2} = alpha{2}.*beta{2}';
    %
    sum_Tii_00s = diag(zeros(partition(1),1));      % initialization
                                                    %     of A00
    sum_Tii_11s = diag(zeros(partition(2),1));      % initialization
                                                    %     of A11
    sum_Tij_01s = zeros(partition(1),partition(2)); % initialization
                                                    %     of A01
    sum_Tij_10s = zeros(partition(2),partition(1)); % initialization
                                                    %     of A10
    %
    % re-estimation for the A00 matrix
    %
    for c=1:2:C-1
        if ( c == 1)
```

```
        Tii_00s = diag((m(1)-1)*(pi_u1)'.*(diag(A{u(1)+1,...
            u(1)+1}).^(m(1)-1)).*beta{1});
    else
        Tii_00s = diag((m(c)-1)*((alpha{c-1}*A{u(c-1)+1,...
            u(c)+1})'.*(diag(A{u(c)+1,u(c)+1}^(m(c)-1)))). ...
                *beta{c});
    end
    sum_Tii_00s = sum_Tii_00s + Tii_00s;        % sum elements
                                                %     of A00
end
%
% re-estimation for the A11 matrix
%
for c=2:2:C-1
    Tii_11s = diag((m(c)-1)*((alpha{c-1}*A{u(c-1)+1,u(c)+1})'...
        .*(diag(A{u(c)+1,u(c)+1}^(m(c)-1)))).*beta{c});
    sum_Tii_11s = sum_Tii_11s + Tii_11s;        % sum elements
                                                %     of A11
end
%
% re-estimation for the A01 matrix
%
for c=1:2:C-1
    Tij_01s = (alpha{c}'*((A{u(c+1)+1,u(c+1)+1}^(m(c+1)-1))...
        *beta{c+1})').*A{u(c)+1,u(c+1)+1};
    sum_Tij_01s = sum_Tij_01s + Tij_01s;         % sum elements
                                                 %     of A01
end
%
% re-estimation for the A10 matrix
%
for c=2:2:C-1
Tij_10s = (alpha{c}'*((A{u(c+1)+1,u(c+1)+1}^(m(c+1)-1))...
        *beta{c+1})').*A{u(c)+1,u(c+1)+1};
    sum_Tij_10s = sum_Tij_10s + Tij_10s; % sums elements of A10
end
%
A_matrix = [sum_Tii_00s sum_Tij_01s; sum_Tij_10s sum_Tii_11s];
%
for i = 1:sum(partition)
    A_matrix(i,:) = A_matrix(i,:)/sum(A_matrix(i,:)); % normalize
                                                      % A
end
%
A{1} = A_matrix(1:partition(1),1:partition(1));
```

```
    A{2} = A_matrix(partition(1)+1:partition(1)+partition(2),1:...
        partition(1));
    A{3} = A_matrix(1:partition(1),partition(1)+1:partition(1)...
        +partition(2));
    A{4} = A_matrix(partition(1)+1:partition(1)+partition(2),...
        partition(1)+1:partition(1)+partition(2));
    %
    pi_est = [Gamma{1} Gamma{2}]; % re-estimated initial state vector
    pi_est = pi_est/sum(pi_est); % normalized initial state vector
    pi_rec(iterations,:) = pi_est;
    pi_u1 = pi_est(1:partition(1));
    iterations                          % display current iteration
    A_matrix                            % display estimated A matrix
end
% End of function file.
```

15.14 Appendix D: Run-Length Code Generation

```
% File: c15_seglength.m
function runcode=c15_seglength(errvect)
% Produces a two-row matrix of error intervals and error-free
% intervals. Row 1 specifies the interval length and row 2
% specifies the interval class (error(1) or no error(0)).
%
len = length(errvect);                  % length of input vector
j = 1;                                  % initialize index of m
count = 1;                              % initialize counter
for i=1:(len-1)
    if errvect(i+1) == errvect(i);      % compare elements
        count = count+1;                % on match increment count
    else
        m(j) = count;                   % record count
        j = j+1;                        % increment index of m
        count = 1;                      % reset counter
    end
end
%
runcode = zeros(2,length(m));           % allocate memory
runcode(1,:) = m;                       % assign counts to row 1
%
if errvect(1)==0
    runcode(2,2:2:length(m)) = 1;       % even index error count
else
    runcode(2,1:2:length(m)) = 1;       % odd index error count
end
% End of function file.
```

15.15 Appendix E: Determination of Error-Free Distribution

In this appendix two MATLAB programs are given for determing a plotting the error-free distribution. The first program is designed for a single data file, and the second program is for the comparison of two data files.

15.15.1 c15_intervals1.m

```
% File: c15_intervals1.m
function [] = c15_Intervals1(r1);
start = find(r1(2,:)==0);                        % index of first
                                                 %     interval
maxLength_1 = max(r1(1,start(1):2:length(r1))); % maximum length
                                                 %     of interval
interval_1 = r1(1,start(1):2:length(r1));        % get the intervals
for i = 1:maxLength_1
    rec_1(i) = length(find(interval_1>=i));      % record the
                                                 %     intervals
end
int1out = rec_1/max(rec_1);
figure;
plot(1:maxLength_1,int1out)
grid;
ylabel('Pr(0m|1)');
xlabel('Length of intervals m');
% End of function file.
```

15.15.2 c15_intervals2.m

```
% File: c15_intervals2.m
function [] = c15_intervals2(r1,r2);
start1 = find(r1(2,:)==0);                         % index of first
                                                   %     interval
maxLength_1 = max(r1(1,start1(1):2:length(r1)));   % maximum length
                                                   %     of interval
interval_1 = r1(1,start1(1):2:length(r1));         % get the
                                                   %     intervals
for i = 1:maxLength_1
    rec_1(i) = length(find(interval_1>=i));        % record the
                                                   %     intervals
end
start2 = find(r2(2,:)==0);                         % index of first
                                                   %     interval
maxLength_2 = max(r2(1,start2(1):2:length(r2)));   % maximum length
                                                   %     of interval
interval_2 = r2(1,start2(1):2:length(r2));         % get the
                                                   %     intervals
```

```
for i = 1:maxLength_2
    rec_2(i) = length(find(interval_2>=i));       % record the
                                                  %     intervals
end
subplot(2,1,1)
plot(1:maxLength_1,rec_1/max(rec_1))
v = axis;
grid;
ylabel('Pr(0m|1)');
xlabel('Original sequence - Length of intervals m');
subplot(2,1,2)
plot(1:maxLength_2,rec_2/max(rec_2))
axis([v])
grid;
ylabel('Pr(0m|1)');
xlabel('Regenerated sequence - Length of intervals m');
% End of function file.
```

Chapter 16

EFFICIENT SIMULATION TECHNIQUES

The basic Monte Carlo (MC) technique was examined in Chapter 9 and, in Chapter 10, the Monte Carlo technique was applied to several simple communications systems. In both Chapters 9 and 10 we saw that the Monte Carlo method, while applicable to all systems without regard to architecture or complexity, has one very significant drawback. The fundamental problem with the Monte Carlo method is that the time required to execute a simulation and obtain a reliable estimate of system performance is often very long. In some cases the required run time may be so long that use of the Monte Carlo method is not practical.

The semianalytic (SA) technique was introduced in Chapter 10. While the SA technique yields simulations that execute very rapidly, application of the SA technique is restricted to situations in which the pdf of the sufficient statistic, upon which symbol decisions are based, is known. In many simulations this statistic is not known, and the SA technique cannot be applied.

In this chapter we take a very brief look at simulation techniques aimed at overcoming the lengthy run-time requirements of the Monte Carlo method. The three methods considered here are quite different. The first of these methods, tail extrapolation, involves curve fitting to MC simulation results. The second technique is based on estimating the pdf of a decision metric through the application of moment methods. The third method to be addressed, importance sampling,

involves biasing the channel noise in a way that forces more decision errors to be made. Importance sampling is a variance reduction technique that provides an estimate of the BER to be obtained that has a smaller variance than the estimate provided by an MC simulation of equivalent execution time. The application of variance reduction techniques essentially involves a tradeoff between analysis and computer run time. A number of these techniques have been investigated and the search for efficient simulation techniques remains an area of active research.

Of the three techniques presented in this chapter, importance sampling has received the most attention and appears to be the most generally applicable. We therefore consider importance sampling in more detail than the other two simulation methods. The effective application of any of these methods (especially importance sampling) requires considerable analytical skill. This chapter should be considered only a brief introduction to these topics. The interested student should consult the research literature on this subject. A few key references are given at the end of this chapter.

16.1 Tail Extrapolation

The first technique we consider is tail extrapolation [1, 2]. This method is applied by executing a number of Monte Carlo simulations using E_b/N_0 values for which reliable simulation results can be obtained with reasonable run times. These results are then extrapolated to values of E_b/N_0 where Monte Carlo simulations are not practical. Of course, one must be extremely careful not to extrapolate into regions of E_b/N_0 where extrapolation is not justified. An example for an assumed system is illustrated in Figure 16.1. For $A < E_b/N_0 < B$, reliable values of the error probability can be computed with reasonable run times using Monte Carlo simulation. This yields points 1, 2, 3, and 4. For our system of interest, extrapolation is valid for $B < E_b/N_0 < C$, and therefore point 5, although extrapolated, is valid. Continuing extrapolation gives point 6. However, our example system exhibits a floor on the probability of error and, as a result, extrapolation of the simulated values into the region for which $E_b/N_0 > C$ is not valid. Therefore, point 6 is not valid. One must be very careful about extrapolating results into regions where experemental or simulated results are not available.

Suppose we know, or are willing to assume, that the pdf of the decision metric is a Gaussian random variable. For this case the probability of symbol error will be given by $Q\left(\sqrt{g\left(E_b/N_0\right)}\right)$, where $Q(x)$ is the Gaussian Q-function and $g(E_b/N_0)$ is a function of E_b/N_0 determined by system parameters such as the modulation format. We see that fitting simulated points to a Gaussian Q-function (such as points 1, 2, 3, and 4 in Figure 16.1) *may* allow for accurate performance extrapolation to higher values of E_b/N_0.

As a more general example, assume that the decision metric is not strictly Gaussian but may be approximated by a pdf of a generalized exponential class.

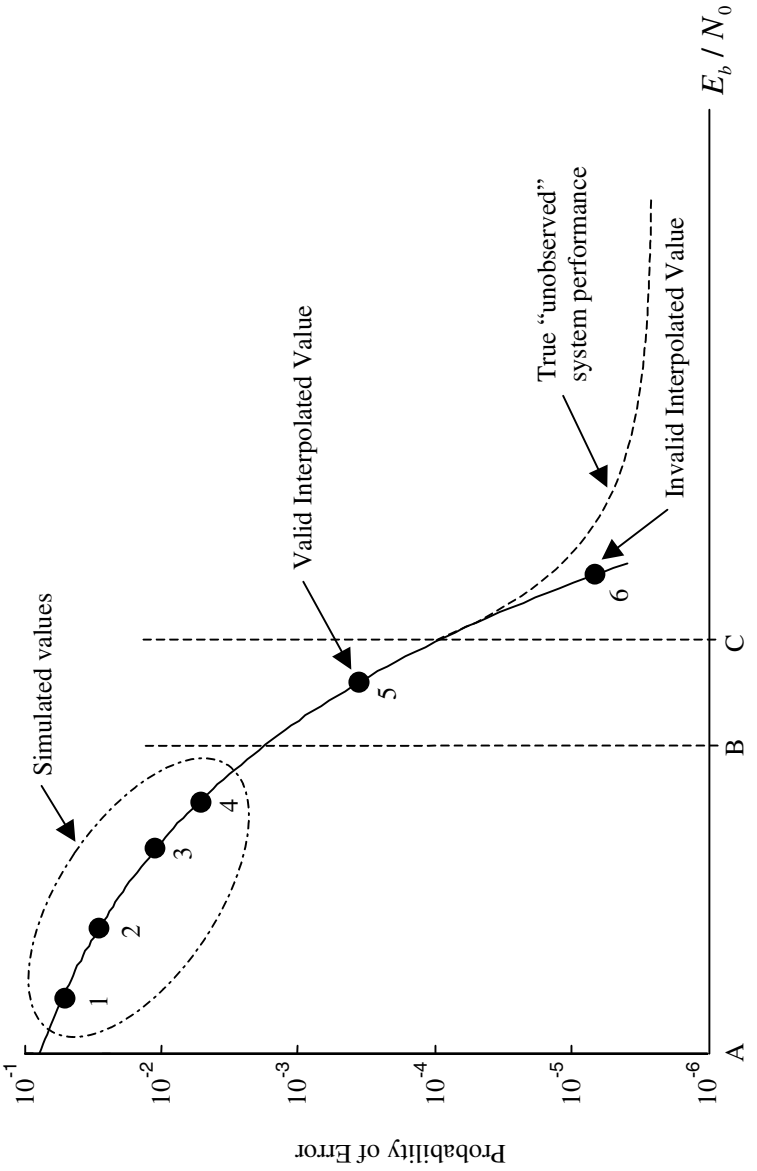


Figure 16.1 Valid and invalid uses of tail extrapolation.

This class of pdfs is defined by the expression

$$f_X(x|v,\sigma,m) = \frac{v}{\sigma\Gamma(1/v)\sqrt{8}} \exp\left[-\left|\frac{x-m}{\sigma\sqrt{2}}\right|^v\right] \tag{16.1}$$

where $\Gamma(\cdot)$ denotes the gamma function, m is the mean of X, v is the mode, and σ is a parameter related to the spreading of the pdf about the mean. The parameter σ is, of course, the standard deviation of X for $\nu = 2$, since a Gaussian pdf results for $v = 2$. A Laplacian pdf results for $v = 1$. If we assume that the mean is zero, one can show that at for large values of T (error threshold) [1]

$$\int_T^\infty f_X(x|v,\sigma)dx \approx \exp\left[-\left(\frac{T}{\sigma\sqrt{2}}\right)^v\right] \tag{16.2}$$

The left-hand side of (16.2) is the BER of the system when we use a threshold of T and, therefore we write the left side as $\widehat{P}_e(T)$. Taking the logarithm of (16.2) twice, we see that

$$\ln\left[-\ln\left(\widehat{P}_e(T)\right)\right] \approx v\ln\left(\frac{T}{\sigma\sqrt{2}}\right) \tag{16.3}$$

or

$$\ln\left[-\ln\left(\widehat{P}_e(T)\right)\right] \approx v\ln\left(\frac{T}{\sqrt{2}}\right) - v\ln(\sigma) \tag{16.4}$$

Now suppose we run M MC simulations using a variety of thresholds, $T_1, T_2 ... T_M$. The BER for each of these thresholds is estimated yielding $\widehat{P}_e(T_1), \widehat{P}_e(T_2), \cdots, \widehat{P}_e(T_M)$. The plot of $\ln\left[-\ln\left(\widehat{P}_e(T)\right)\right]$ as a function of $\ln\left(T/\sqrt{2}\right)$ should be a straight line, determined using linear regression, with y intercept $-v\ln(\sigma)$ and slope v. This is illustrated in Figure 16.2. The mode v is determined from the slope. Once v is known, σ is determined from the y intercept y_i according to

$$\sigma = \exp(-y_i/\sigma) \tag{16.5}$$

Once v and σ have been determined, (16.2) can be used as an estimator of the BER of the system. Tail extrapolation can be extended to other pdfs. This technique will be accurate if the analyst is skilled, or lucky, enough to select an appropriate family of pdfs for a given system.

16.2 pdf Estimators

We know that the error probability of a digital communications system can be written in the form

$$P_E = \int_T^\infty f_V(v)\,dv \tag{16.6}$$

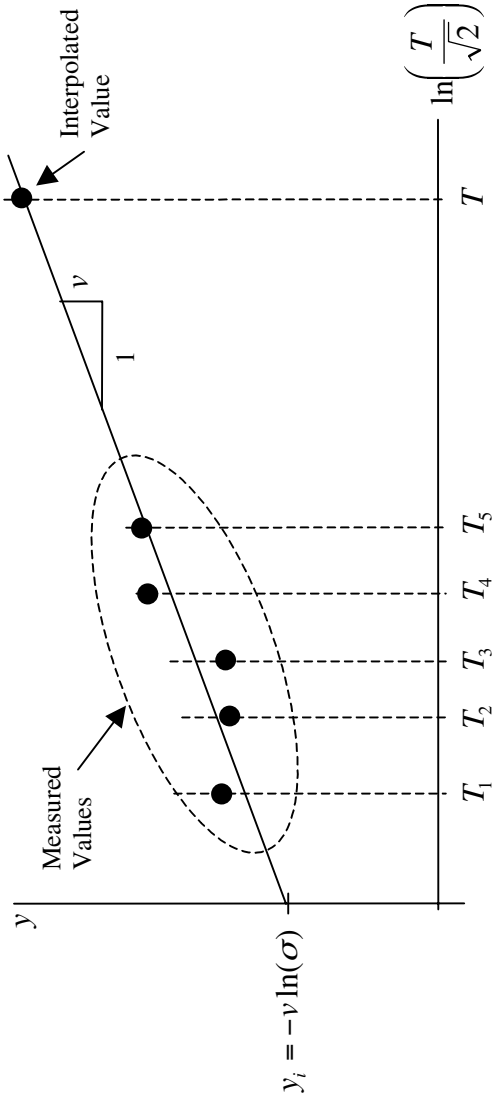


Figure 16.2 Tail extrapolation procedure.

where T is the error threshold (values of $X > T$ result in errors) and $f_V(v)$ is the pdf of the decision statistic. One technique for estimating the BER is to first estimate the pdf and then evaluate the integral defined in (16.6) using numerical integration. An intuitive way of accomplishing this is to form a histogram of the data representing the values of V. In order to be useful the histogram must have a sufficient number of bins for $V > T$ to allow the integration defined by (16.6) to be carried out with the required accuracy. In addition, as we saw in Chapter 8, the histogram is a biased estimator for finite data size N. Fortunately there are alternatives to the histogram method. The two most popular techniques for estimating the pdf from a data set $x[n]$ are the Gram-Charlier series [3, 4] and the Parzen series [4].

The Gram-Charlier series approximates a pdf in the form

$$\widehat{f}_Y(y) = \frac{1}{\sqrt{2\pi}\sigma_v} \exp\left[-\frac{y^2}{2}\right] \sum_{k=0}^{N} C_k H_k(y) \tag{16.7}$$

where $H_k(\cdot)$ represents the Chebyshev-Hermite polynomials and C_k denotes the series coefficients. In writing (16.7) we have made, for mathematical simplicity, the assumption that X is a zero-mean unit-variance random variable. The Chebyshev-Hermite polynomials are defined by the recursion relationship

$$H_k(y) = yH_{k-1}(y) - (k-1)H_{k-2}(y), \qquad k \geq 2 \tag{16.8}$$

in which $H_0(y) = 1$ and $H_1(y) = y$. Given the Chebyshev-Hermite polynomials, the coefficients can be calculated from

$$C_k = \frac{1}{k!} \int_{-\infty}^{\infty} H_k(y) f_Y(y) dy \tag{16.9}$$

Note that even though $f_Y(y)$ is unknown, (16.9) can be evaluated in terms of the moments of Y. If the random variable of interest Y is not a zero-mean unit-variance variable, we can apply the transformation

$$Z = \frac{Y - \mu_y}{\sigma_y} \tag{16.10}$$

where μ_y and σ_y represent the mean and standard deviation of Y, respectively. Under this transformation Z becomes a zero-mean unit variance random variable so that (16.7) can be applied directly with Y replaced by Z.

The Gram-Charlier series provides a good approximation to the target pdf in the neighborhood of the mean but usually provides a poor approximation in the tails of the pdf. Unfortunately, for BER estimation it is the tails of the pdf that are of interest. The Gram-Charlier series has a number of other difficulties. The Gram-Charlier approximation is not asymptotically unbiased and does not uniformly converge to the target pdf as more terms are added. Also, for a finite number of terms, N, the approximation defined by (16.7) is not a true pdf and may even be negative for particular values of y. Despite these shortcomings, the Gram-Charlier series is a useful pdf estimator for many applications.

The natural estimator for the moments of a random variable producing a data sequence $x[n]$ is given by

$$E\left\{X^k\right\} = \lim_{N\to\infty}\left[\frac{1}{N}\sum_{n=1}^{N} x^k[n]\right] \tag{16.11}$$

In practice of course, the data size N must be finite. For finite N, this estimator, unfortunately, has no optimality properties. It is, however, usually a consistent estimator and therefore satisfactory performance can be expected for sufficiently large N [5].

The shortcomings of the Gram-Charlier series can at least be partially overcome through the use of the Parzen pdf estimator. The Parzen estimator takes the form

$$\widehat{f}_V(v) = \frac{1}{Nh(N)}\sum_{k=1}^{N} g\left(\frac{v-v_k}{h(N)}\right) \tag{16.12}$$

The choices for $g(x)$ and $h(N)$ are somewhat arbitrary but reasonable choices that work well in many applications are [4]

$$g(x) = \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{x^2}{2}\right) \tag{16.13}$$

and

$$h(N) = \frac{1}{\sqrt{N}} \tag{16.14}$$

In (16.12) N is the size of the data set, the function $g(\cdot)$ is a weighting function, and $h(N)$ is a smoothing factor. Although the Parzen estimator is biased for finite N, it can be shown that if $h(N) \to 0$ as $N \to \infty$ and $Nh(N) \to \infty$ as $N \to \infty$, the Parzen estimator is asymptotically unbiased. Clearly, the smoothing function defined by (16.14) satisfies these requirements. It can also be shown that the Parzen estimator is consistent.

Both the Gram-Charlier and the Parzen techniques have been successfully applied to a number of problems. A recent paper explores the use of both of these techniques for estimating the BER of a digital communications system operating in a variety of environments. These include the AWGN channel, AWGN with cochannel interference, and AWGN plus multipath [6].

16.3 Importance Sampling

As we saw in Chapters 9 and 10, MC simulations perform a random experiment a large number of times, N, and count the number of outcomes, N_A, denoting some event, A, of interest. One can estimate the probability of event A using the equation

$$\Pr(A) = \lim_{N\to\infty}\frac{N_A}{N} \tag{16.15}$$

Assuming that N_A denotes the number of errors in the transmission of N symbols, high-performance communication systems yield values of N_A many orders of magnitude smaller than N. The brief discussion of confidence intervals in Chapter 9 showed that to generate a reliable estimator, we require N_A to be at least 10, and preferably 100. In many situations the requirement for large N_A can lead to very long simulation execution times. This is especially true when the BER of a communication system is low. If the BER is small, a simulated demodulation error is a rare (low probability) but important event. The goal of importance sampling is to alter the simulation in a controlled way, in order to increase the number of these important or rare events while still enabling the determination of the correct probability of demodulation error. This requires a change in (16.15). Importance sampling falls into a class of simulation methodologies known as variance reduction techniques. The goal of variance reduction techniques is to develop an unbiased estimator which exhibits a reduced variance and/or a reduced simulation execution time as compared to a Monte Carlo simulation.

Before beginning our discussion of importance sampling, it is important to point out that a detailed description of importance sampling is well beyond the scope of this introductory text. Rather than treating this subject in depth, we instead consider the estimation of the area of a geometrical shape in a way that makes use of the fundamental concepts of importance sampling. This is followed by an example of importance sampling applied to a communications system.

In Chapter 9 the subject of Monte Carlo simulation was introduced by developing an estimator for the area of a two-dimensional shape. (Recall that this led to an estimator for the value of π.) In this chapter we again consider an estimator for the area of a two-dimensional shape (an ellipse in this case) in a way that provides insight into the importance sampling simulation technique. At the end of this section a simple communications system simulation is presented. This introduction hopefully makes the basic literature on importance sampling more understandable to the reader. One wishing to apply importance sampling to a practical communications system will need to become familiar with the basic literature. A suggested list of papers is given at the end of this chapter.

16.3.1 Area of an Ellipse

Suppose one wishes to use Monte Carlo simulations to estimate the area of an ellipse in the (x, y) plane. In particular, let us examine the problem of finding the area of an ellipse which has a major axis in the x direction of length $2\sqrt{2}$, and a minor axis of length 2.

Monte Carlo Estimators Revisited

The specified ellipse contains all points (x, y) such that

$$x^2 + 2y^2 < 2 \tag{16.16}$$

The area of the ellipse, A_e, is

$$A_e = \int_{y=-1}^{1} \int_{x=-\sqrt{2-2y^2}}^{\sqrt{2-2y^2}} dxdy = \pi\sqrt{2} \cong 4.443 \tag{16.17}$$

By rewriting the integral, we can put the problem in a form better suited for Monte Carlo integration and for demonstrating importance sampling. The first step is to represent the limits of the integral by a function in the integrand. This is accomplished by defining an *indicator function*, $h_e(x, y)$, such that

$$h_e(x,y) = \begin{cases} 1, & x^2 + 2y^2 \leq 2 \quad \text{(inside the ellipse)} \\ 0, & x^2 + 2y^2 > 2 \quad \text{(outside the ellipse)} \end{cases} \tag{16.18}$$

This gives

$$A_e = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h_e(x,y)\, dxdy \tag{16.19}$$

We next define a bounding area that contains the ellipse and has an area easy to calculate. The area of this bounding area is denoted A_{bound}. Multiplying the integrand by unity in the form A_{bound}/A_{bound} gives

$$A_e = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h_e(x,y) A_{bound} \frac{1}{A_{bound}}\, dxdy \tag{16.20}$$

The next step is to define a pair of random variables (X, Y) that are uniformly distributed over A_{bound}. The pdf for this pair of random variables is

$$f_{XY}(x,y) = \begin{cases} 1/A_{bound}, & h_e(x,y) = 1 \\ \text{arbitrary}, & h_e(x,y) = 0 \end{cases} \tag{16.21}$$

Combining (16.20) and (16.21) gives

$$A_e = A_{bound} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h_e(x,y) f_{XY}(x,y)\, dxdy \tag{16.22}$$

As in Chapter 9, the estimator for the area of the ellipse is

$$\widehat{A}_e = A_{bound} \left(\frac{N_e}{N} \right) \tag{16.23}$$

where N denotes the number of points generated within the bounding area, and N_e denotes the number of points falling inside the ellipse.

Selecting Bounding Boxes for MC Simulations

One can show that the estimator defined by (16.23) is consistent and unbiased provided the bounding area fully encloses the ellipse. This is illustrated in the following example.

Example 16.1. As a simple example, $N = 500$ points in a square box, centered on the origin, are generated. The area of the ellipse is estimated using (16.23) with

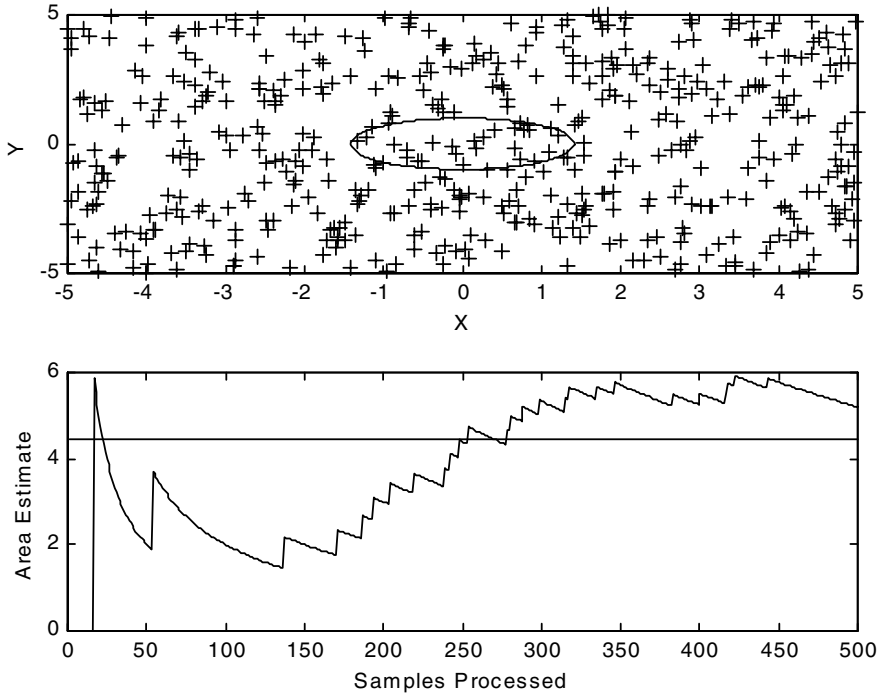


Figure 16.3 Area estimation process for $N = 100$.

three different box sizes; $A_{bound} = 4, 8$, and 100. The upper half of Figure 16.3 shows the sample points generated in this simulation with $A_{bound} = 100$ (10 units on a side), and the lower half of Figure 16.3 shows the running estimate of the area, along with the theoretically calculated value. It is clear from Figure 16.3 that we need more than 500 samples in this simulation to generate an accurate estimate of the ellipse area.

Next, let us examine the effect of changing the area of the bounding box. Setting $A_{boound} = 8$ dramatically improves the convergence properties of the estimator. As a result, estimated area is within a few percent of the correct value after only 100 samples. Based on this observation we are tempted to conclude that the smaller the bounding box, the faster the estimator will converge. This is correct, to a point. Note, however, that when $A_{bound} = 4$, the estimator quickly converges, but to an incorrect value. The estimator is biased, since the bounding box no longer includes the entire ellipse. If the bounding box were to continue to shrink, the rate of convergence would continue to improve at the cost of increased bias and ultimately we would simply be finding the area of the bounding box and not the area of the ellipse. The convergence properties for $A_{bound} = 100$, 8, and 4 are illustrated Figure 16.4.

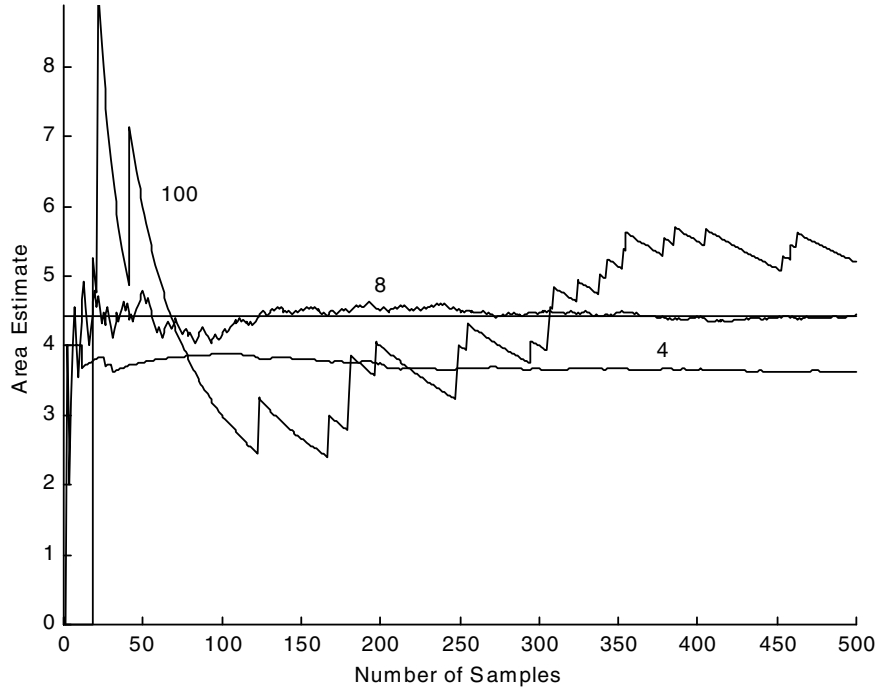


Figure 16.4 Demonstration of convergence for different bounding areas.

The effect of the bounding box size can be predicted from (16.23). The relative precision of the ratio N_e/N increases with N_e. In other words, we wish to have as many points as possible inside the ellipse, leading us to select small boxes. However, the MC estimator will become biased if the box fails to fully enclose the entire ellipse, leading to an optimal box size of $A_{bound} = 8$, which is the smallest square that encloses the ellipse. The challenge is to find estimators that both converge quickly and are unbiased. ■

Optimal Bounding Regions

We saw in the previous section that a two-dimensional MC simulation will generate random points over some region of the (x, y) plane. To make the estimator converge quickly, we would like the region used by the MC estimator to be as small as possible, while still ensuring that the area of interest is fully enclosed. For simplicity, we initially assume that the bounding region is a rectangle. However, there is nothing in the development of the estimation algorithm that requires this. The algorithm is applicable for any bounding region of known area. Consider, for example, a bounding rectangle. The smallest rectangle that includes the ellipse has an area of

$4\sqrt{2}$. Since this is less than the area of the smallest bounding square, the estimator based on a rectangular bounding region will converge faster than the estimator based on a square bounding region.

We now extend the search for a superior bounding box to higher-order polygons. We can find polygons that enclose the ellipse and have areas even smaller than $4\sqrt{2}$. It will become increasingly difficult to calculate the area of the bounding polygon analytically and will also become increasingly difficult to generate points uniformly distributed within the polygon. There is typically a point of diminishing returns, where the saving in computer execution time does not justify the additional analytical difficulties. The ultimate bounding region is simply the ellipse itself. In this case, $A_{bound} = A_e$, and every randomly generated point will fall inside the ellipse. Thus, $N_e = N$. Equation (16.23) indicates that such a simulation will be extremely efficient and will *produce the exact answer after a single random point has been generated*. Of course, in this limiting case there is no need to perform the simulation, since A_e must to be determined analytically before the simulation code can be developed. This, of course, eliminates the need for the simulation.

One of the dangers of making highly efficient simulations is that a small calculation error may cause part of the ellipse to fall outside the bounding region, generating a biased estimator. If one did not know the correct answer, it would be impossible to detect the bias by simply observing the simulation results.

In summary, when using a uniform pdf in MC simulations, one should:

1. Determine the area of the bounding region analytically.
2. Prove analytically that the bounding region fully encloses the area of interest.
3. Find an algorithm for efficiently generating uniformly distributed points within the bounding region.
4. Minimize the area of the bounding region, given constraints 1, 2, and 3.
5. Run the simulation long enough to observe a large number of samples in the ellipse.

These concerns and requirements are quite easy to understand for this simple problem. They will reappear in the following sections in more complicated simulation strategies, and the mathematical complexities will tend to make them more difficult to visualize and appreciate.

Nonuniform pdfs and Weighting Functions

To this point we have considered only the basic Monte Carlo algorithm. Recall from Chapter 9 that in Monte Carlo simulations we have two counters, a sample counter and an event counter. The sample counter is incremented each time the underlying random experiment, such as generating a sample and testing to see if it falls in an area of interest, is performed. The event counter is incremented each time an event of interest, such as a sample falling within a given area occurs. Counters are usually incremented by adding the number one to the counter contents. We

now show that, by adding a weight w_i to the event counter rather than the number one, the necessity for having a uniform distribution across the area of interest is removed. The concept of incrementing by weights rather than by integers is central to the application of importance sampling methodology to BER estimation.

Previously in (16.19) we multiplied the integrand by unity in the form A_{bound}/A_{bound} to yield (16.20). A more general estimator is formed by multiplying the integrand in (16.19) by a two-dimensional *weighting function* divided by itself, $w(x, y)/w(x, y)$. As before, we replace $1/w(x, y)$ by a pdf. Specifically

$$f_{XY}(x, y) = \begin{cases} 1/w(x, y), & h_e(x, y) = 1 \\ \text{arbitrary}, & h_e(x, y) = 0 \end{cases}$$

This gives

$$A_e = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h_e(x, y) w(x, y) f_{X,Y}(x, y)\, dxdy \tag{16.24}$$

This is the more general form of (16.22), where $f_{XY}(x, y)$ can be any pdf that is nonzero within the ellipse. The area estimator corresponding to (16.24) is

$$\widehat{A}_e = \frac{1}{N} \sum_{i=1}^{N} h_e(x_i, y_i) w(x_i, y_i) \tag{16.25}$$

where random vectors $(x_i,\ y_i)$ are IID (independent and identically distributed) with pdf $f_{XY}(x, y)$. This is a generalization of the previous result, since for the uniform case the reciprocal of the pdf, that is, the weighting function, was the constant A_{bound} for all points inside the ellipse. Equation (16.25) allows us to use nearly any pdf in our simulation and includes the uniform pdf as a special case. The variance and bias of the estimator will be a function of the pdf selected for the simulation.

To avoid estimator bias, the uniform pdf had to include the entire ellipse. In other words, the uniform pdf could not be zero over any finite area inside the ellipse. This same condition applies here. One can show that the area estimator will be unbiased provided $f_{XY}(x, y) > 0$ for all points inside the ellipse. Since many common pdfs are nonzero over the entire (x, y) plane, it is not difficult to generate unbiased estimators. Minimizing the variance is a more difficult problem since the convergence rate of the simulation is a rather complex function of the pdf. The following example provides insight into this problem.

Example 16.2. In this example we restrict our attention to Gaussian pdfs, and investigate the convergence rate for three different standard deviations, namely, $\sigma = 10$, $\sigma = 1$, and $\sigma = 0.2$. Figure 16.5 shows the result with $\sigma = 10$. (Note the original 5×5 box. The ellipse is obscured by the sampling points.) When σ is large, the pdf inside the ellipse is small but nearly constant. Thus, relatively large weight is assigned to each sample falling in the ellipse. Several large jumps followed by intervals with a hyperbolic $(1/N)$ decay rate can be seen. The area estimate *slowly* converges to the correct value, in much the same way as a uniform pdf and a large bounding area.

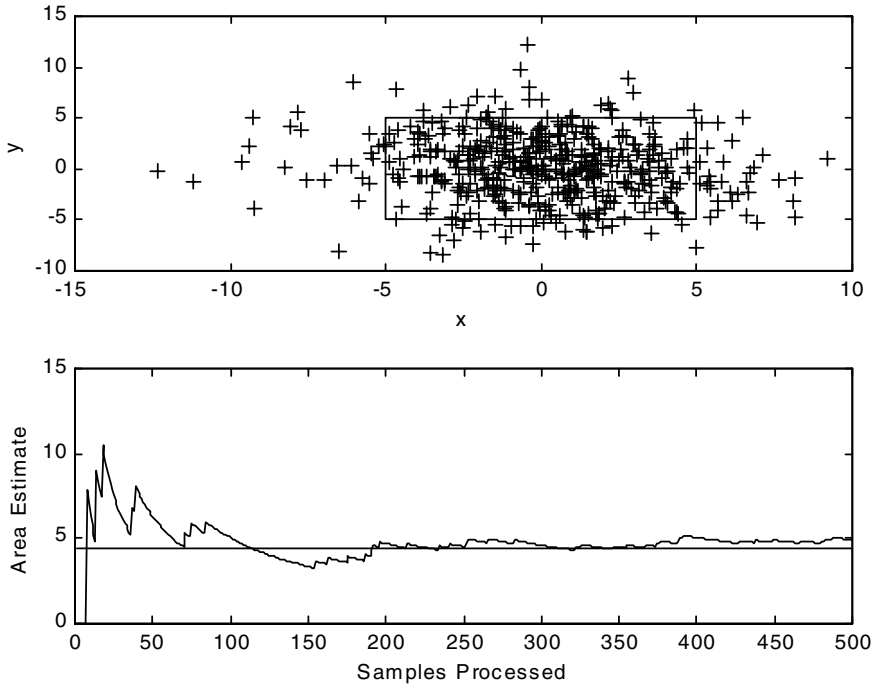


Figure 16.5 Estimator performance with Gaussian sampling and $\sigma = 10$.

We next consider the case in which $\sigma = 1$. For this case the standard deviation is close to the dimensions of the ellipse and, as a result, the convergence rate is much more rapid. A large number of samples fall inside the ellipse, and each is assigned a relatively small weight. This case roughly corresponds to the uniform pdf when the bounding area is just slightly larger than the ellipse. This result is shown in Figure 16.6.

Perhaps the most interesting case is when σ is small compared to the dimensions of the ellipse. Nearly all of the points in the sample simulation fall inside the ellipse, most very close to the origin. However, at sample 153, the Gaussian random number generator produces a sample near the edge of the ellipse. Because the pdf for this sample is so small, it is assigned a phenomenally large weight, 1.3874×10^{16} in this case, and the large weight causes the area estimate to suddenly increase by many orders of magnitude. The resulting behavior is shown in Figure 16.7. We call this point an *extreme event* because it is an extremely rare, but extremely important, event. Since the estimate is unbiased, the extreme event is required to offset the effect an initially low estimate. The only way that the estimator can be unbiased is to run the simulation long enough to see many extreme events. This will typically require far more processor time than any of the methods mentioned to

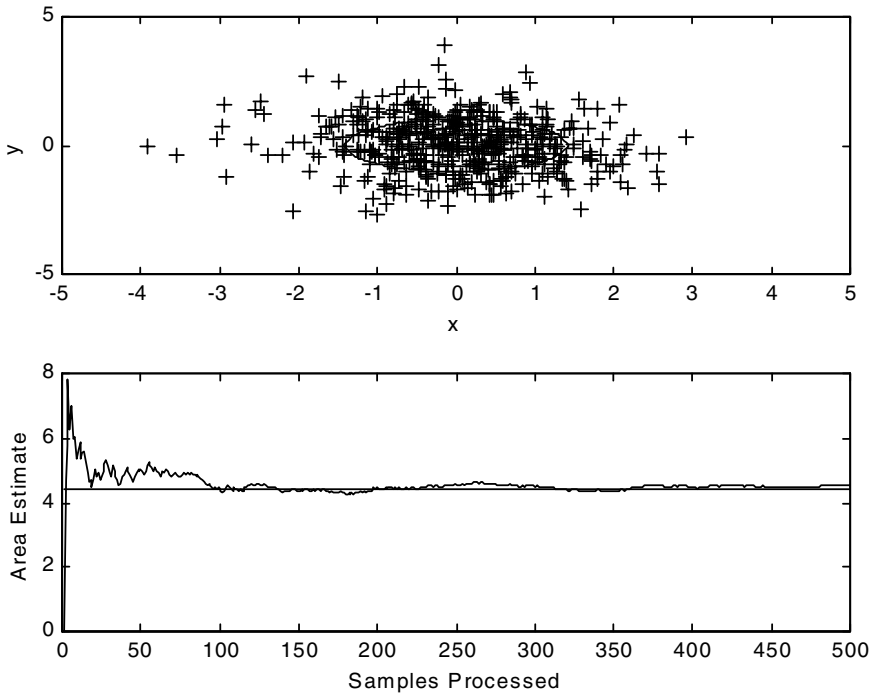


Figure 16.6 Estimator performance with Gaussian sampling and $\sigma = 1$.

date. This simulation is somewhat similar to the uniform case when the bounding area was smaller than the ellipse. In the uniform sampling case a biased estimate was produced. In this new case in which Gaussian sampling is used the estimator is unbiased, but only because of the presence of extreme events. If one terminates the simulation prior to the occurrence of the first extreme event or shortly after the occurrence of an early (small N) extreme event, very misleading results will be obtained from the simulation. Note that in all three cases the estimator N_e/N considered here is unbiased. However, it may require an unreasonable amount of computer time to average out the extreme events.

The effect of extreme event behavior is familiar to us from past studies. As a simple example, assume that a digital communications system has a BER of 10^{-6}. Also assume that an MC simulation is performed and the first 99 transmitted symbols are received without error. The BER estimator $\widehat{P}_E = N_e/N$ will produce zero for $1 \leq N \leq 99$, a value that is clearly too small. If symbol number 100 in the simulation is received in error (an extreme event), the value of $\widehat{P}_E$ jumps from zero to 10^{-2}, which is a value four orders of magnitude too large. If a long span of correct decisions follows this error, $\widehat{P}_E$ will decrease as $1/N$ for $N > 100$. This behavior will continue until the next error occurs. This is essentially the same behavior

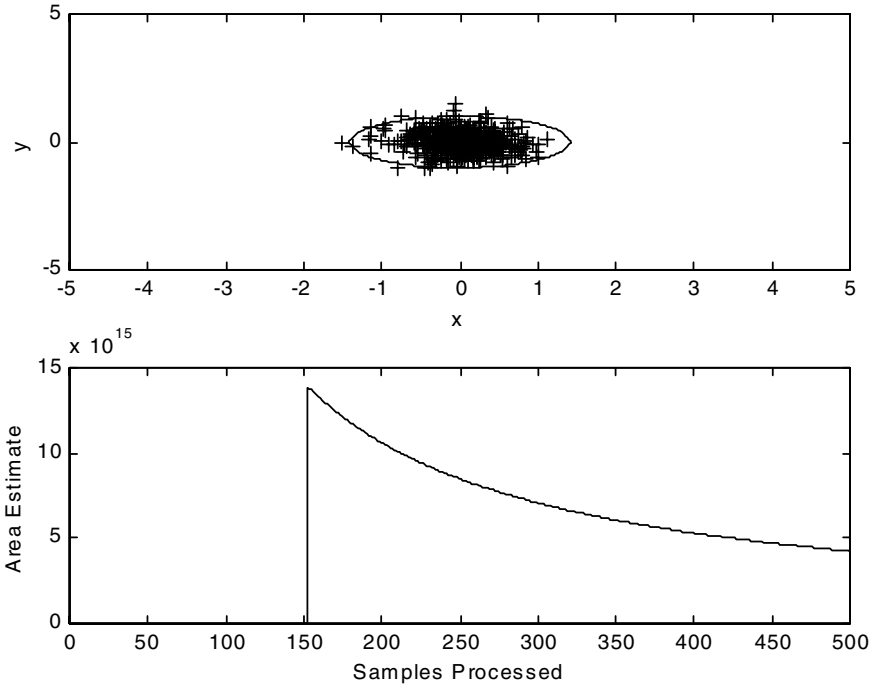


Figure 16.7 Estimator performance with Gaussian sampling and $\sigma = 0.15$.

observed in Figure 16.7, with the only difference being that the error in Figure 16.7 is multiplied by a large weight w_i. Note that extreme events occurring early in a simulation run have a more pronounced effect than extreme events occurring later in a simulation run.

All three simulations used the same random number seed. All three estimators are unbiased and consistent, but the mode of convergence varies widely. These modes are summarized in Table 16.1. ■

Table 16.1 Convergence Modes

Standard Deviation, σ	**Number of Samples in Ellipse**	**Mode of Convergence**
Too small	Nearly all	Slow (exhibits extreme events)
Appropriate	Between 10% and 90%	Rapid
Too large	Very few	Slow

In this very simple example, there are a variety of methods for predicting the mode of convergence. The scatter diagrams indicate when the pdf covers too little of the ellipse or too much of the plane. Another clear indicator is the weighting function. Ideally the weighting function should be small over the entire area of the ellipse. If enough data is collected, the estimates and samples of the error functions also clearly indicate the modes. Ideally, one would like to establish confidence intervals for these estimators. This is difficult because the estimates no longer have a binomial distribution. The Gaussian approximation can be used if a sufficiently large number of symbols are processed. The number of samples required for the approximation to be accurate will depend on the convergence mode. When the convergence is rapid, only a few hundred samples may be necessary. The slowly converging estimators are similar to the uniform case, and a few hundred points inside the ellipse are necessary. In the last case, many hundred extreme events would need to be observed before the Gaussian approximation could be applied.

16.3.2 Sensitivity to the pdf

Finding the area of an ellipse is a deterministic problem. The random variables used in the MC simulation are strictly artificial, and do not model any physical device or signal. Most physical communication systems are subject to random perturbations and process random data. The first inclination is to develop a simulation that uses the same noise source pdf as the physical system. However, as we will show, this is not necessary. The simulation can use a *biased* noise source, having a different pdf than the one used in the physical system. For example, suppose a physical system generates a jointly Gaussian pair of random variables (x, y) having the pdf

$$f_{phy}(x,y) = \frac{1}{2\pi\sigma_x\sigma_y\sqrt{1-r^2}} \exp\left[-\frac{1}{2(1-r^2)}\left(\frac{x^2}{\sigma_x^2} - \frac{2rxy}{\sigma_x\sigma_y} + \frac{x^2}{\sigma_y^2}\right)\right] \tag{16.26}$$

which σ_x and σ_y are the standard deviations of X and Y, respectively, and r represents the correlation coefficient. The probability that a pair of random variables falls inside the ellipse described earlier is

$$\begin{aligned} B_e &= \int_{y=-1}^{1} \int_{x=-\sqrt{2-2y^2}}^{\sqrt{2-2y^2}} f_{phy}(x,y)\,dxdy \\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h_e(x,y) f_{phy}(x,y)\,dxdy \end{aligned} \tag{16.27}$$

This expression is difficult to evaluate. A simple MC solution to this problem would be to use the sum

$$\widehat{B}_e = \frac{1}{N}\sum_{i=1}^{N} h_e(x_i, y_i) = \frac{N_e}{N} \tag{16.28}$$

where the random vectors (x_i, y_i) are IID with pdf $f_{phy}(x, y)$. Suppose we did not have a random number generator that would produce $f_{phy}(x, y)$, but we did have

a random number generator to produce samples with the distribution $f_{sim}(x, y)$. We can still solve the problem using Monte Carlo techniques. As in the previous section, we begin by multiplying the integrand of (16.27) by unity in the form of $w(x, y)/w(x, y)$. Unlike the earlier development, the weighting function is now defined as

$$w(x,y) = \begin{cases} f_{phy}(x,y)/f_{sim}(x,y), & h_e(x,y) = 1 \\ \text{arbitrary}, & h_e(x,y) = 0 \end{cases} \tag{16.29}$$

Substituting (16.29) into (16.27) gives

$$B_e = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h_e(x,y)w(x,y)f_{sim}(x,y)\,dxdy \tag{16.30}$$

which can be approximated by the sum

$$\widehat{B}_e = \frac{1}{N}\sum_{i=1}^{N} h_e(x_i,y_i)w(x,y) \tag{16.31}$$

The random vectors (x_i, y_i) are IID with pdf $f_{sim}(x, y)$. As with the previous problem, one must be concerned about the bias, consistency, and convergence mode of this estimator. It is easy to show that when $f_{sim}(x, y) > 0$ for all points where $h_e(x, y)f_{phy}(x, y) > 0$, the estimate will be unbiased. One can also show that this ensures that the estimate is consistent. As in the simpler problem, the mode of convergence may be slow, fast, or very slow because of the existence of exteme events. The mode will depend on the weighting function $w(x, y) = f_{phy}(x, y)/f_{sim}(x, y)$ inside the ellipse. Convergence rates are summarized in Table 16.2.

Table 16.2 Summary of Convergence Rates

PDFs Inside Ellipse	Weighting Function Inside Ellipse	Rate of Convergence
$f_{sim}(x,y) = f_{phy}(x,y)$	$w(x,y) = 1$	Same as simple MC
$f_{sim}(x,y) > f_{phy}(x,y)$	$w(x,y) < 1$	Faster than simple MC
$f_{sim}(x,y) < f_{phy}(x,y)$	$w(x,y) > 1$	Slower than simple MC

16.3.3 A Final Twist

Before applying the ellipse area estimation problem to a communications system, we need to add one final twist. The channel generates noise, but this noise may be altered by the receiver into a new distribution with an unknown pdf. We need to find the probability that samples generated by this new pdf fall in the region of interest. Working only with two-dimensional pdfs for now, assume that the channel generates the physical noise, (x, y), with pdf $f_{phy}(x, y)$. These random variables are fed to a receiver, which applies operator $g(x, y) = (\alpha, \beta)$. This new set of random

variables has pdf $f_{rec}(\alpha, \beta)$. The probability the random variables (α, β) falls into some region of interest, such as the ellipse, is

$$B_e = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h_e(\alpha, \beta) f_{rec}(\alpha, \beta)\, d\alpha d\beta \tag{16.32}$$

The complexity of g may make it impractical to calculate $f_{rec}(\alpha, \beta)$ from $f_{phy}(x, y)$. The MC simulation of this problem will be

$$\widehat{B}_e = \frac{1}{N} \sum_{i=1}^{N} h_e(\alpha_i, \beta_i) \tag{16.33}$$

where (α_i, β_i) has pdf $f_{rec}(\alpha, \beta)$. It is possible to run this simulation even though we do not know how to calculate $f_{rec}(\alpha, \beta)$. If we generate random vectors (x_i, y_i) with pdf $f_{phy}(x, y)$ and then pass each sample vector through function g, the resulting vector (α_i, β_i) will have pdf $f_{rec}(\alpha, \beta)$. A more descriptive way of writing (16.33) is

$$\widehat{B}_e = \frac{1}{N} \sum_{i=1}^{N} h_e(g(x_i, y_i)) \tag{16.34}$$

Suppose we now alter $f_{phy}(x, y)$ to $f_{bias}(x, y)$. This will cause $f_{rec}(\alpha, \beta)$ to change to $f_{rec-bias}(\alpha, \beta)$. To remove the effects of this bias from the MC estimate, we need to calculate a weighting function

$$w(\alpha_i, \beta_i) = \frac{f_{rec}(\alpha_i, \beta_i)}{f_{rec-bias}(\alpha_i, \beta_i)} = \frac{f_{phy}(x_i, y_i)}{f_{bias}(x_i, y_i)} \tag{16.35}$$

where $(\alpha_i, \beta_i) = g(x_i, y_i)$. This weighting function can be used to alter (16.34) to create

$$\widehat{B}_e = \frac{1}{N} \sum_{i=1}^{N} w(g(x_i, y_i)) h_e(g(x_i, y_i)) \tag{16.36}$$

which can be shown to be a consistent and unbiased estimate.

16.3.4 The Communication Problem

The geometric problem of finding the area of an ellipse can be mapped to the problem of finding the BER of a communication system. The two sets of random variables (x, y) and (α, β) represent the channel noise waveform and the decision metric, respectively. In the communications system problem, the channel noise often has far more than two dimensions, even though the decision metric may have only one dimension. These two pairs of random variables were related by operator $g(\cdot)$, which represents the receiver. The ellipse of the geometric problem now corresponds to the values of the decision metric which causes the receiver to have a demodulation error. Finding the BER of the system is equivalent to finding the probability of a

randomly generated sample falling inside the ellipse in the geometric problem. We will bias the channel noise in order to increase the frequency of error events, and calculate a weighting function

$$w_i = \frac{f_{phy}(n_i)}{f_{bias}(n_i)} \tag{16.37}$$

The BER estimator is

$$\hat{P}_E = \frac{1}{N}\sum_{i=1}^{N} w(n_i) h_e(g(x_i, y_i)) \tag{16.38}$$

In the geometric problem, we ensured an unbiased estimator by making sure that the bounding region included the entire ellipse. The equivalent requirement here is simply that all values of the decision metric can be generated. It is not difficult to satisfy this requirement. The rate of convergence in the geometric problem depends on how we select the variance of the noise source that establishes the sampling points. If a large number of samples fall outside the ellipse, the convergence rate is slow. If we generate almost no samples outside the ellipse, the convergence is again slow because of extreme events. We ideally desire a mix of samples both inside and outside the ellipse. In the communication system problem, we ideally want a mix of decision metric values both in the error region and in the error-free region. Too many samples in either region will cause the convergence rate to be very slow. Figure 16.8 illustrates the system with the additional functions required to implement importance sampling represented by heavy lines.

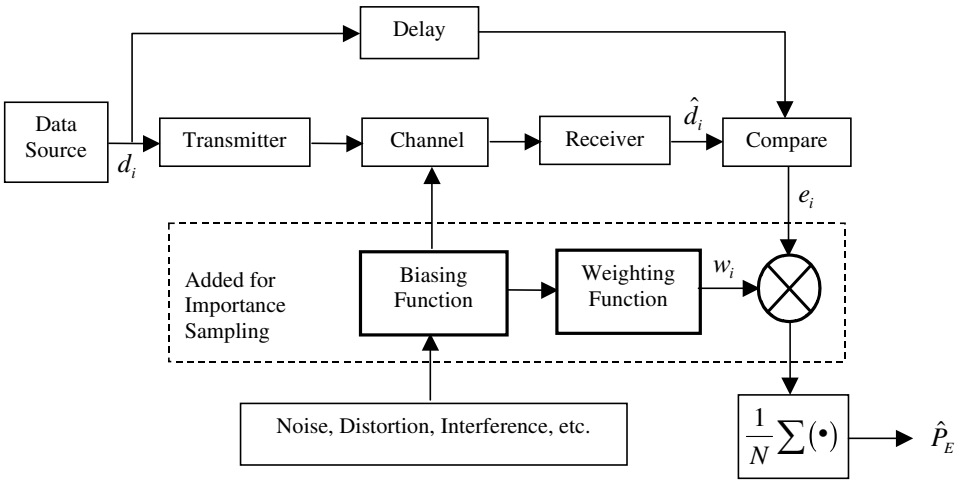


Figure 16.8 System simulation with importance sampling.

16.3.5 Conventional and Improved Importance Sampling

There are a variety of methods for biasing the noise when applying importance sampling. The two the most common methods are called Conventional Importance Sampling (CIS) and Improved Importance Sampling (IIS). In CIS one simply increases the variance of the channel noise, which is equivalent to operating the system at a lower signal-to-noise ratio (SNR). In IIS the mean of the noise is altered rather than the variance. For a comparison of these two techniques and examples illustrating their application to communications systems, the interested student is referred to the literature. (See the Further Reading section.)

Example 16.3. In this example we consider CIS applied to the differential QPSK system previously considered in Chapter 10. The MATLAB script for running a CIS simulation follows:

```
% File:  c16_CISQPSK.m
Eb = 20:2:32; No = -50;                  % Eb and Noin dBm
ChannelAttenuation = 70;                 % channel attenuation in dB
EbNodB = (Eb-ChannelAttenuation)-No;     % Eb/No in dB
EbNo = 10.(EbNodB./10);                  % Eb/No in linear units
BER_T = 0.5*erfc(sqrt(EbNo));            % BER (theoretical)
N = ones(size(BER_T))*2000;              % set N=2000 for all runs
CISBias = 1+(EbNo/20);                   % set CIS bias
BER_CIS = zeros(size(Eb));               % initialize BER vector
for k=1:length(Eb)                       % main loop starts here
    BER_CIS(k) = c16_CISQPSKrun(N(k),Eb(k),...
        No,ChannelAttenuation,0,0,0,0,CISBias(k));
    disp(['Simulation ',num2str(k*100/length(Eb)),'% Complete']);
end
semilogy(EbNodB,BER_CIS,'o',EbNodB,2*BER_T,'-')
xlabel('Eb/No (dB)'); ylabel('BER'); grid;
% End of script file.
```

This script sets the bias for the CIS simulation and then calls the function listed in Appendix A. This function is nearly identical to the Monte Carlo simulation of the QPSK system considered in Chapter 10. The only difference is that the channel noise is biased by increasing the noise variance.

The BER estimator in the example is defined by (16.38). A sample output from this simulation is shown in Figure 16.9. The circles in this figure represent the CIS estimates of the BER, while the solid curve is the theoretical performance. At first glance there appears to be some significant errors in the measurements at high SNR. However, these results are based only on 1,000 demodulated symbols. Using conventional MC simulations it would be impractical to measure any BER less than approximately 10^{-3}. However, from the results of the CIS simulation we are able to get some idea of the BER for error rates approaching 10^{-8}. ■

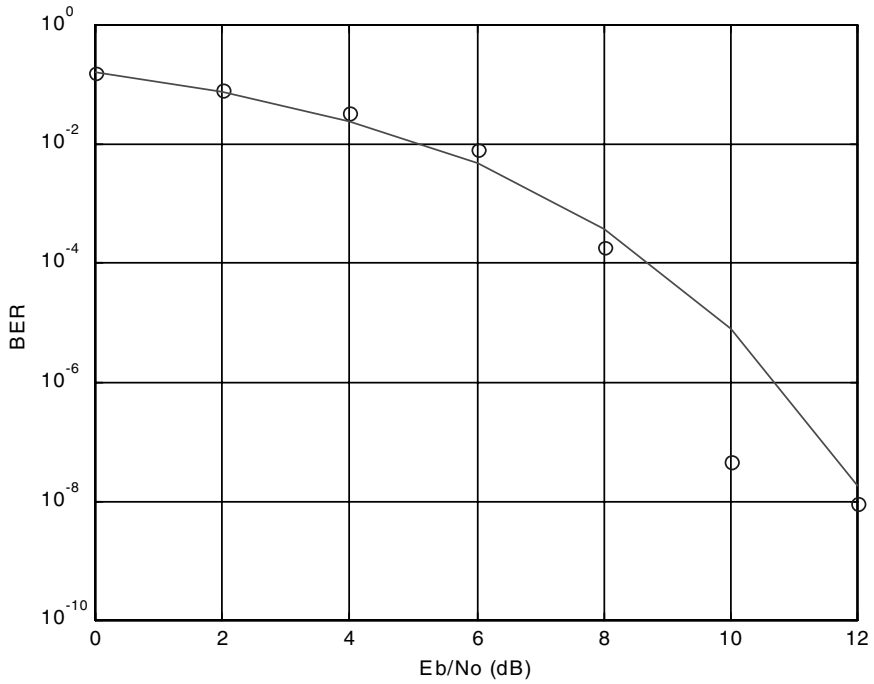


Figure 16.9 Sample CIS simulation result.

16.4 Summary

This chapter focused on alternatives to the Monte Carlo simulation method for the determination of the BER in a digital communications system. The search for alternative simulation methodologies is motivated by the lengthy execution times often encountered with Monte Carlo methods, especially when the BER is small. Three techniques were briefly discussed. These were tail extrapolation, pdf estimation, and importance sampling. The successful application of any of these techniques requires considerable analytical skill. The material presented in this chapter should be considered only a high-level overview and the student wishing to pursue an in-depth understanding of these methods will wish to consult the literature.

16.5 Further Reading

The topics covered in this chapter are also covered in

M. C. Jeruchim, P. Balaban, and K. S. Shanmugan, *Simulation of Communication Systems*, 2nd ed., New York: Kluwer Academic/Plenum Press, 2000.

A very large number of papers have been written on the subject of importance sampling. A partial reading list consists of the following:

P. Smith, M. Shafi, and G. Hongshen, "Quick Simulation: A Review of Importance Sampling Techniques in Communications Systems," *IEEE Journal on Selected Areas in Communications*, Vol. 15, No. 4, May 1997, pp. 597–613.

M. C. Jeruchim, "Techniques for Estimating the Bit Error Rate in the Simulation of Digital Communication Systems," *IEEE Journal on Selected Areas Communications*, Vol. 2, No. 1, January 1984, pp. 153–170.

R. J. Wolfe, M. C. Jeruchim, and P. M. Hahn, "On Optimum and Suboptimum Biasing Procedures for Importance Sampling in Communication Simulation," *IEEE Transactions on Communications*, Vol. 38, No. 5, May 1990, pp. 639–647.

K. S. Shanmugan and P. Balaban, "A Modified Monte Carlo Simulation Technique for the Evaluation of Error Rate in Digital Communication Systems," *IEEE Transactions on Communications*, Vol. 28, No. 11, November 1980, pp. 1916–1924.

M. C. Jeruchim, "On the Application of Importance Sampling to the Simulation of Digital Satellite Multihop Links," *IEEE Transactions on Communications*, Vol. 32, No. 10, October 1984, pp. 1088–1092.

B. R. Davis, "An Improved Importance Sampling Method for Digital Communication System Simulations," *IEEE Transactions on Communications*, Vol. 34, No. 7, July 1986, pp. 715–719.

P. M. Hahn and M. C. Jeruchim, "Development in the Theory and Application of Importance Sampling," *IEEE Transactions on Communications*, Vol. 35, No. 7, July 1987, pp. 706–714.

Q. Wang and V. Bhargava, "On the Application of Importance Sampling to BER Estimation in the Simulation of Digital Communication Systems," *IEEE Transactions on Communications*, Vol. 35, No. 11, November 1987, pp. 1231–1233.

D. Lu and K. Yao, "Improved Importance Sampling Technique for Efficient Simulation of Digital Communication Systems," *IEEE Journal on Selected Areas in Communications*, Vol. 6, No. 1, January 1988, pp. 67–75.

M. C. Jeruchim, P. M. Han, K. P. Smyntek, and R. T. Ray, "An Experimental Investigation of Conventional and Efficient Importance Sampling," *IEEE Transactions on Communications*, Vol. 37, No. 6, June 1989, pp. 578–587.

D. Remondo, R. Srinivasan, V. Nicola, and W. Van Etten, "Adaptive Importance Sampling for Performance Evaluation and Parameter Optimization of Communication Systems," *IEEE Transactions on Communications*, Vol. 48, No. 4, April 2000, pp. 557–565.

J. Porath and T. Aulin, "Improved Technique for Quick Error Rate Estimation of Multi-Dimensional Communication Schemes," *IEE Proceedings—Communications*, Vol. 146, No. 6, December 1999, pp. 343–346.

16.6 References

1. S. B. Weinstein, "Estimation of Small Probabilities by Linearization of the Tail of a Probability Distribution Function," *IEEE Transactions on Communications Technology*, Vol. 19, No. 6, 1971, pp. 1149–1155.

2. M. C. Jeruchim, P. Balaban, and K. S. Shanmugan, *Simulation of Communication Systems*, 2nd ed., New York: Kluwer Academic/Plenum Press, 2000.

3. K. S. Miller, *Engineering Mathematics,* New York: Dover Publications, 1956.

4. K. S. Shanmugan and A. M. Breipohl, *Random Signals: Detection, Estimation and Data Analysis*, New York: Wiley, 1988.

5. S. M. Kay, *Fundamentals of Statistical Signal Processing, Vol. 1: Estimation Theory*, Upper Saddle River, NJ: Prentice Hall, 1993.

6. J. D. Laster, J. H. Reed, and W. H. Tranter, "Bit Error Rate Estimators Using Probability Density Function Estimators," *IEEE Transactions on Vehicular Technology*, Vol. 52, No. 1, January 2003, pp. 260–267.

16.7 Problems

16.1 Assume that the conditional error probability, conditioned on the transmission of a binary 0, for a binary digital communications system can be expressed

$$P_{E|0} = \int_0^\infty \frac{v}{\sigma\Gamma(1/v)\sqrt{8}} \exp\left[-\left|\frac{x-m}{\sigma\sqrt{2}}\right|^v\right] dx$$

where we assume that m is negative. Show that $P_{E|0}$ can be expresssed in terms of a incomplete gamma function $\Gamma(a,x)$ where

$$\Gamma(a,x) = \int_x^\infty t^{a-1}\exp(-t)\,dt$$

By choosing m appropriately, and assuming that $P_{E|0} = P_{E|0}$, plot the system error probability for $v = 1.9$, 2, and 2.1. (Note: MATLAB contains a routine for computing the incomplete gamma function.)

16.2 Using the recursion relationship for the Chebyshev-Hermite polynomials, evaluate $H_k(y)$ for $2 \le k \le 6$. Using these results determine the Gram-Charlier series coefficients C_k for $0 \le k \le 6$.

16.3 A three-dimensional surface is defined by the equation

$$z^2 < -x^2 - 2y^2 + 16$$

find the volume enclosed by this surface using Monte Carlo simulation. For each iteration of the MC simulation, generate three independent, identically distributed, uniform random numbers, ranging from -10 to +10. How many points did you need to generate before you became 90% confident that your error was less than 1%?

16.4 Repeat Problem 16.3, but allow the x, y, and z points to cover different ranges. How many points did you need to generate before you became 90% confident that your error was less than 1%?

16.5 Repeat Problem 16.3, but use a three-dimensional, jointly Gaussian pdf for x, y, and z. What are the optimal values for mean and variance for each of the three marginal pdfs?

16.6 Suppose you wish to estimate the area of a two-dimensional ellipse, using Monte Carlo simulation and a jointly Gaussian pdf. Would you always select uncorrelated random variables, or would there be times when it would help to have them correlated?

16.7 Water dripping from a faucet hits a the bottom of a sink with a jointly Gaussian pdf (zero mean, standard deviation 0.1 in both directions, correlation coefficient is zero). There is a circular drain in the sink, centered at the origin, of radius 0.1. Using Monte Carlo simulation, estimate what fraction of the drops hit the drain.

16.8 Repeat Problem 16.7, but in the simulation use a jointly Gaussian pdf that has a standard deviation of 0.2 in both directions. The physical problem has not changed—you will be using a different pdf in the simulation than you observe in the physical world.

16.9 Write a simulation that implements an MSK transmitter, AWGN channel, and matched filter receiver. Perform a Monte Carlo simulation for this system to determine the SNR required to achieve a BER of 10^{-2}.

16.10 Repeat Problem 16.9, but use importance sampling. Bias the noise by altering it's variance. What is the optimal amount to alter the variance, and how much execution time does this save you?

16.11 Repeat Problem 16.10, but bias the noise by adding a constant. What is the best constant to add and how much simulation time does this save you?

16.12 Write a simulation that implements an OQPSK transmitter. Place a bandpass filter on the transmitter output to limit the transmitted signal to the main lobe. Create two adjacent channels, which are also OQPSK transmitters limited to the main lobe. Place the adjacent channels as close as possible to

the desired channel, without allowing the spectra to overlap. Pass the signals through an AWGN channel, and a matched filter receiver. If the interfering signals are not present, set the BER of the system to 10^{-3}. Use MC simulation to estimate the BER when the interfering signals are equal in power to the desired signal.

16.8 Appendix A: MATLAB Code for Example 16.3

```
% file: c16_CISQPSKrun.m
function BER_CIS=CISQPSKrun(N,Eb,No,ChanAtt,...
    TimingBias,TimingJitter,PhaseBias,PhaseJitter,CISBias)
fs = 1e+6;                          % sampling rate (samples/second)
SymRate = 1e+5;                     % symbol rate (symbols/second)
Ts = 1/fs;                          % sampling period
TSym = 1/SymRate;                   % symbol period
SampPerSym=fs/SymRate;              % samples per symbol
SymToSend = N;                      % symbols to be transmitted
ChanBW = 4.99e+5;                   % bandwidth of channel (Hz)
CISWeightIntegrator = 1;            % importance sampling weight
CISWeightIntegratorOld = 1;         % importance sampling weight
MeanCarrierPhaseError = PhaseBias;  % mean of carrier phase
StdCarrierPhaseError = PhaseJitter; % std dev of phase error
MeanSymbolSyncError = TimingBias;   % mean symbol sync error
StdSymbolSyncError = TimingJitter;  % std dev symbol sync error
ChanGain = 10^(-ChanAtt/20);        % channel gain (linear units)
TxBitClock = Ts/2;                  % Tx clock period
RxBitClock = Ts/2;                  % Rx clock period
TxSymSent = 1; RxSymDemod = 0;      % Tx and Rx symbol counters
%
RxNoiseStd = sqrt((10^((No-30)/10))*(fs/2));     % std dev of noise
TxSigAmp = sqrt(10^((Eb-30)/10)*SymRate);        % signal amplitude
probe1 = zeros((SymToSend+1)*SampPerSym,1);      % probe 1 memory
probe2 = zeros((SymToSend+1)*SampPerSym,1);      % probe 2 memory
probe1counter = 1; probe2counter = 1;            % initialize probes
%
% Buffers that contain the transmitted and received data.
%
[unused,SourceBitsI] = random_binary(SymToSend,1);
[unused,SourceBitsQ] = random_binary(SymToSend,1);
%
% Differentially encode the transmitted data.
%
TxBitsI = SourceBitsI*0; TxBitsQ = SourceBitsQ*0;
for k=2:length(TxBitsI)
    TxBitsI(k) = or(and(not(xor(SourceBitsI(k),SourceBitsQ(k))),...
        xor(SourceBitsI(k),TxBitsI(k-1))), ...
        and(xor(SourceBitsI(k),SourceBitsQ(k)),...
        xor(SourceBitsQ(k),TxBitsQ(k-1))));
    TxBitsQ(k) = or(and(not(xor(SourceBitsI(k),SourceBitsQ(k))),...
        xor(SourceBitsQ(k),TxBitsQ(k-1))), ...
        and(xor(SourceBitsI(k),SourceBitsQ(k)),...
```

```
        xor(SourceBitsI(k),TxBitsI(k-1))));
end;
%
% Make a complex data stream of the I and Q bits.
%
TxBits = ((TxBitsI*2)-1)+(sqrt(-1)*((TxBitsQ*2)-1));
%
% Initialize transmitter and the receiver integrate and dump filter.
%
RxIntegrator = 0; TxBitClock = 2*TSym;
%
% Design the channel filter and state array if needed.
%
[b,a] = butter(2,ChanBW/(fs/2));
b = [1]; a = [1];                                    % bypass filter
[junk,FilterState] = filter(b,a,0);
%
% Loop once for each sample.
%
while TxSymSent < SymToSend
    %
    % Update transmitter clock. Get new data bits if required.
    %
    TxBitClock = TxBitClock+Ts;
    if TxBitClock  > TSym
        TxSymSent = TxSymSent+1;                    % get new bit
        %
        % We don't want the clock to increase to infinity so
        % subtract off an integer number of Tb seconds.
        %
        TxBitClock = mod(TxBitClock,TSym);
        %
        % Get the new bit and appropriately.
        %
        TxOutput = TxBits(TxSymSent)*TxSigAmp;
    end
    [Rx,FilterState] = filter(b,a,TxOutput,FilterState);
    %
    % Add white Gaussian noise to the signal.
    % First create unbiased (Monte Carlo) noise and then bias.
    %
    UnbiasedNoise = RxNoiseStd*(randn(1,1)+sqrt(-1)*randn(1,1));
    BiasedNoise = CISBias*UnbiasedNoise;
    %
    % Calculate the CIS weight for this particular noise sample.
```

```
    %
    CISWeight = cgpdf(BiasedNoise,0,RxNoiseStd)./...
        cgpdf(BiasedNoise,0,CISBias*RxNoiseStd);
    %
    % Since we are using white noise, the total CIS weight will be
    % the product of the individuals CIS weights.
    %
    CISWeightIntegrator = CISWeightIntegrator*CISWeight;
    Rx = (ChanGain*Rx)+BiasedNoise;
    %
    % Phase rotation due to receiver carrier synchronization error.
    %
    PhaseRotation = exp(sqrt(-1)*2*pi*(MeanCarrierPhaseError+...
        (randn(1,1)*StdCarrierPhaseError))/360);
    Rx = Rx*PhaseRotation;
    probe1(probe1counter) = Rx; probe1counter = probe1counter+1;
    %
    % Update the Integrate and Dump Filter at the receiver.
    %
    RxIntegrator = RxIntegrator+Rx;
    probe2(probe2counter) = RxIntegrator;...
        probe2counter=probe2counter+1;
    %
    % Update the receiver clock, to see if it is time to
    % sample and dump the integrator.
    %
    RxBitClock = RxBitClock+Ts;
    RxTSym = TSym*(1+MeanSymbolSyncError+...
        (StdSymbolSyncError*randn(1,1)));
    if RxBitClock > RxTSym
        RxSymDemod = RxSymDemod+1;
        RxBitsI(RxSymDemod) = round(sign(real(RxIntegrator))+1)/2;
        RxBitsQ(RxSymDemod) = round(sign(imag(RxIntegrator))+1)/2;
        RxBitsCISWeight(RxSymDemod) = ...
            CISWeightIntegrator*CISWeightIntegratorOld;
        %
        % Reset clock and dump the integrator.
        %
        RxBitClock = RxBitClock-TSym; RxIntegrator = 0;
        CISWeightIntegratorOld = CISWeightIntegrator;
        CISWeightIntegrator = 1;
    end
end
%
% Implement differential decoder.
```

```
%
SinkBitsI = SourceBitsI*0;
SinkBitsQ = SourceBitsQ*0;
for k=2:RxSymDemod
    SinkBitsI(k) = or(and(not(xor(RxBitsI(k),RxBitsQ(k))),...
        xor(RxBitsI(k),RxBitsI(k-1))),...
        and(xor(RxBitsI(k),RxBitsQ(k)),...
        xor(RxBitsQ(k),RxBitsQ(k-1))));
    SinkBitsQ(k) = or(and(not(xor(RxBitsI(k),RxBitsQ(k))),...
        xor(RxBitsQ(k),RxBitsQ(k-1))),...
        and(xor(RxBitsI(k),RxBitsQ(k)),...
        xor(RxBitsI(k),RxBitsI(k-1))));
end;
%
% Look for best time delay between input and output, 100 bits.
%
[C,Lags] = vxcorr(SourceBitsI(10:110),SinkBitsI(10:110));
[MaxC,LocMaxC] = max(C);
BestLag = Lags(LocMaxC);
%
% Adjust time delay to match best lag.
%
if BestLag > 0
    SourceBitsI = SourceBitsI(BestLag+1:length(SourceBitsI));
    SourceBitsQ = SourceBitsQ(BestLag+1:length(SourceBitsQ));
    RxBitsCISWeight = ...
        RxBitsCISWeight(BestLag+1:length(RxBitsCISWeight));
elseif BestLag < 0
    SinkBitsI = SinkBitsI(-BestLag+1:length(SinkBitsI));
    SinkBitsQ = SinkBitsQ(-BestLag+1:length(SinkBitsQ));
    RxBitsCISWeight = ...
        RxBitsCISWeight(-BestLag+1:length(RxBitsCISWeight));
end
%
% Make all arrays the same length.
%
TotalBits = min(length(SourceBitsI),length(SinkBitsI));
TotalBits = TotalBits-20;
SourceBitsI = SourceBitsI(10:TotalBits);
SourceBitsQ = SourceBitsQ(10:TotalBits);
SinkBitsI = SinkBitsI(10:TotalBits);
SinkBitsQ = SinkBitsQ(10:TotalBits);
RxBitsCISWeight = RxBitsCISWeight(10:TotalBits);
%
% Find the number error events and the BER.
```

```
%
IErrors = SourceBitsI ^= SinkBitsI;
QErrors = SourceBitsQ ^= SinkBitsQ;
BER_CIS = ...
    sum(IErrors.*RxBitsCISWeight)+sum(QErrors.*RxBitsCISWeight);
BER_CIS = BER_CIS/(2*length(SourceBitsI));
% End of function file.
```

16.8.1 Supporting Routines

The program `random_binary.m` is given in Chapter 10, Appendix A. The program `vxcorr.m` is given in Chapter 10, Appendix B.

cgpdf.m

```
function value = cgpdf(x,mean,sigma)
variance=sigma.^2;
value=(exp((((real(x)-mean).^2)+((imag(x)-mean).^2))/...
    (-2*variance)))/(2*pi*variance);
% End of function file.
```

Chapter 17

CASE STUDY: SIMULATION OF A CELLULAR RADIO SYSTEM

17.1 Introduction

A wide variety of wireless communication systems have been developed to provide access to the communications infrastructure for mobile or fixed users in a myriad of operating environments. Most of today's wireless systems are based on the *cellular radio* concept. Cellular communication systems allow a large number of mobile users to seamlessly and simultaneously communicate to wireless modems at fixed base stations using a limited amount of radio frequency (RF) spectrum. The RF transmissions received at the base stations from each mobile are translated to baseband, or to a wideband microwave link, and relayed to mobile switching centers (MSCs), which connect the mobile transmissions with the Public Switched Telephone Network (PSTN). Similarly, communications from the PSTN are sent to the base station, where they are transmitted to the mobile. Cellular systems employ either frequency division multiple access (FDMA), time division multiple access (TDMA), code division multiple access (CDMA), or spatial division multiple access (SDMA) [1, 2].

Wireless communication links experience hostile physical channel characteristics, such as time-varying multipath and shadowing due to large objects in the propagation path. In addition, the performance of wireless cellular systems tends to be limited by interference from other users, and for that reason, it is important to have accurate techniques for modeling interference. These complex channel conditions are difficult to describe with a simple analytical model, although several models do provide analytical tractability with reasonable agreement to measured channel data [3, 4]. However, even when the channel is modeled in an analytically elegant manner, in the vast majority of situations it is still difficult or impossible to construct analytical solutions for link performance when error control coding, equalization, diversity, and network models are factored into the link model. Simulation approaches, therefore, are usually required when analyzing the performance of cellular communication links.

Like wireless links, the *system* performance of a cellular radio system is most effectively modeled using simulation, due to the difficulty in modeling a large number of random events over time and space. These random events, such as the location of users, the number of simultaneous users in the system, the propagation conditions, interference and power level settings of each user, and the traffic demands of each user, combine together to impact the overall performance seen by a typical user in the cellular system. The aforementioned variables are just a small sampling of the many key physical mechanisms that dictate the instantaneous performance of a particular user at any time within the system. The term *cellular radio system*, therefore, refers to the *entire population* of mobile users and base stations throughout the geographic service area, as opposed to a single link that connects a single mobile user to a single base station. To design for a particular system-level performance, such as the likelihood of a particular user having acceptable service throughout the system, it is necessary to consider the complexity of *multiple users* that are simultaneously using the system throughout the coverage area. Thus, simulation is needed to consider the multi-user effects upon any of the individual links between the mobile and the base station.

The link performance is a *small-scale* phenomenon, which deals with the instantaneous changes in the channel over a small local area, or small time duration, over which the average received power is assumed constant [1]. Such assumptions are sensible in the design of error control codes, equalizers, and other components that serve to mitigate the transient effects created by the channel. However, in order to determine the overall system performance of a large number of users spread over a wide geographic area, it is necessary to incorporate *large-scale* effects such as the statistical behavior of interference and signal levels experienced by individual users over large distances, while ignoring the transient channel characteristics. One may think of link-level simulation as being a vernier adjustment on the performance of a communication system, and the system-level simulation as being a coarse, yet important, approximation of the overall level of quality that any user could expect at any time.

Cellular systems achieve high capacity (e.g., serve a large number of users) by allowing the mobile stations to share, or *reuse* a communication channel in different

regions of the geographic service area. Channel reuse leads to *co-channel interference* among users sharing the same channel, which is recognized as one of the major limiting factors of performance and capacity of a cellular system. An appropriate understanding of the effects of co-channel interference on the capacity and performance is therefore required when deploying cellular systems, or when analyzing and designing system methodologies that mitigate the undesired effects of co-channel interference. These effects are strongly dependent on system aspects of the communication system, such as the number of users sharing the channel and their locations. Other aspects, more related to the propagation channel, such as path loss, shadow fading (or shadowing), and antenna radiation patterns are also important in the context of system performance, since these effects also vary with the locations of particular users. In this chapter, we will discuss the application of system-level simulation in the analysis of the performance of a cellular communication system under the effects of co-channel interference. We will analyze a simple multiple-user cellular system, including the antenna and propagation effects of a typical system. Despite the simplicity of the example system considered in this chapter, the analysis presented can easily be extended to include other features of a cellular system.

17.2 Cellular Radio System

17.2.1 System-Level Description

Cellular systems provide wireless coverage over a geographic service area by dividing the geographic area into segments called cells as shown in Figure 17.1. The available frequency spectrum is also divided into a number of channels with a group of channels assigned to each cell. Base stations located in each cell are equipped with wireless modems that can communicate with mobile users. Radio frequency channels used in the transmission direction from *the base station to the mobile* are referred to as *forward channels*, while channels used in the direction from *the mobile to the base station* are referred to as *reverse channels*. The forward and reverse

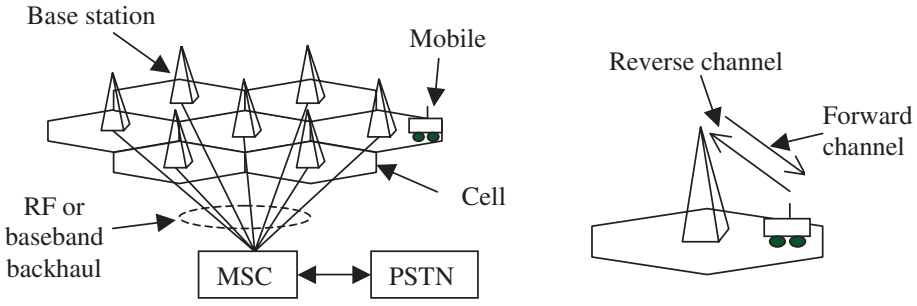


Figure 17.1 Basic architecture of a cellular communications system.

channels together identify a duplex cellular channel. When *frequency division duplex* (FDD) is used, the forward and reverse channels are split in frequency. Alternatively, when *time division duplex* (TDD) is used, the forward and reverse channels are on the same frequency, but use different time slots for transmission.

High-capacity cellular systems employ *frequency reuse* among cells. This requires that co-channel cells (cells sharing the same frequency) are sufficiently far apart from each other to mitigate co-channel interference. Channel reuse is implemented by covering the geographic service area with clusters of N cells, as shown in Figure 17.2, where N is known as the cluster size.

The RF spectrum available for the geographic service area is assigned to each cluster, such that cells within a cluster do not share any channel [1]. If M channels make up the entire spectrum available for the service area, and if the distribution of users is uniform over the service area, then each cell is assigned M/N channels. As the clusters are replicated over the service area, the reuse of channels leads to *tiers* of co-channel cells, and *co-channel interference* will result from the propagation of RF energy between co-channel base stations and mobile users. Co-channel interference in a cellular system occurs when, for example, a mobile simultaneously receives signals from the base station in its own cell, as well as from co-channel base stations in nearby cells from adjacent tiers. In this instance, one co-channel forward link (base station to mobile transmission) is the desired signal, and the other co-channel signals received by the mobile form the total co-channel interference at the receiver. The power level of the co-channel interference is closely related to the separation distances among co-channel cells. If we model the cells with a hexagonal shape, as

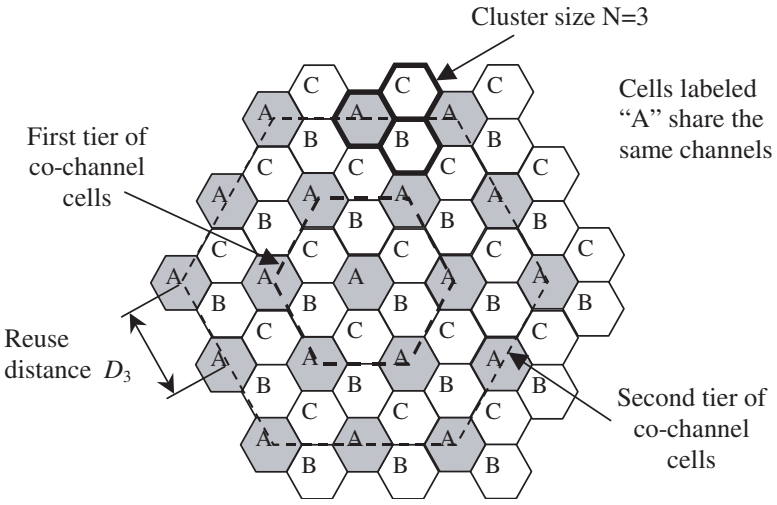


Figure 17.2 Cell clustering: Depiction of a three-cell reuse pattern.

in Figure 17.2, the minimum distance between the center of two co-channel cells, called the *reuse distance* D_N, is

$$D_N = \sqrt{3N}R \tag{17.1}$$

where R is the maximum radius of the cell (the hexagon is inscribed within the radius). Therefore, we can immediately see from Figure 17.2 that a small cluster size (small reuse distance D_N), leads to high interference among co-channel cells.

The level of co-channel interference received within a given cell is also dependent on the number of active co-channel cells at any instant of time. As mentioned before, co-channel cells are grouped into tiers with respect to a particular cell of interest. The number of co-channel cells in a given tier depends on the *tier order* and the geometry adopted to represent the shape of a cell (e.g., the coverage area of an individual base station). For the classic hexagonal shape, the closest co-channel cells are located in the first tier and there are six co-channel cells. The second tier consists of 12 co-channel cells, the third, 18, and so on. The total co-channel interference is, therefore, the sum of the co-channel interference signals transmitted from all co-channel cells of all tiers. However, co-channel cells belonging to the first tier have a stronger influence on the total interference, since they are closer to the cell where the interference is measured.

Co-channel interference is recognized as one of the major factors that limits the capacity and link quality of a wireless communications system and plays an important role in the tradeoff between system capacity (large-scale system issue) and link quality (small-scale issue). For example, one approach for achieving high capacity (large number of users), without increasing the bandwidth of the RF spectrum allocated to the system, is to reduce the channel reuse distance by reducing the cluster size N of a cellular system [1]. However, reduction in the cluster size increases co-channel interference, which degrades the link quality.

The level of interference within a cellular system at any time is random and must be simulated by modeling both the RF propagation environment between cells and the position location of the mobile users. In addition, the traffic statistics of each user and the type of channel allocation scheme at the base stations determine the instantaneous interference level and the capacity of the system.

The effects of co-channel interference can be estimated by the *signal-to-interference ratio* (*SIR*) of the communication link, defined as the ratio of the power of the desired signal S, to the power of the total interference signal, I. Since both power levels S and I are random variables due to RF propagation effects, user mobility and traffic variation, the *SIR* is also a random variable. Consequently, the severity of the effects of co-channel interference on system performance is frequently analyzed in terms of the system outage probability, defined in this particular case as the probability that *SIR* is below a given threshold SIR_0. This is

$$P_{outage} = \Pr[SIR < SIR_0] = \int_0^{SIR_0} p_{SIR}(x)\, dx \tag{17.2}$$

where $p_{SIR}(x)$ is the probability density function (pdf) of the *SIR*. Note the distinction between the definition of a *link outage probability,* that classifies an outage

based on a particular bit error rate (BER) or E_b/N_0 threshold for acceptable voice performance, and the *system outage probability* that considers a particular *SIR* threshold for acceptable mobile performance of a typical user.

Analytical approaches for estimating the outage probability in a cellular system, as discussed in Chapter 11, require tractable models for the RF propagation effects, user mobility, and traffic variation, in order to obtain an expression for $p_{SIR}(x)$. Unfortunately, it is very difficult to use analytical models for these effects, due to their complex relationship to the received signal level. Therefore, the estimation of the outage probability in a cellular system usually relies on simulation, which offers flexibility in the analysis. In this chapter, we present a simple example of a simulation of a cellular communication system, with the emphasis on the system aspects of the communication system, including multi-user performance, traffic engineering, and channel reuse. In order to conduct a system-level simulation, a number of aspects of the individual communication links must be considered. These include the channel model, the antenna radiation pattern, and the relationship between E_b/N_0 (e.g., the *SIR*) and the acceptable performance.

In order to simulate a cellular system, we must mathematically model the individual components of the system. In the next section, the models of the system components to be simulated will be discussed.

17.2.2 Modeling a Cellular Communication System

This section treats several aspects of a cellular communication system that will be useful when developing a simulation model for a system to be investigated.

Trunking and Grade of Service

We begin by discussing some important issues related to the capability of a cellular radio system to provide service to a large number of users. Like fixed telephone systems, cellular radio systems rely on the *trunking concept* to provide communication service for a large number of users, employing a limited resource that, in the cellular communications case, is the available RF spectrum or the number of available channels. The use of trunking techniques is possible due to the statistical behavior of users, described mainly by the following two aspects:

1. A single user accesses the system, that is, *requests a call*, on a random basis during a period of time, and the interval τ between two consecutive call requests from the same user, follows an exponential distribution. Thus the underlying pdf is

$$p_\tau(\tau) = \lambda_u \ \exp(-\lambda_u \tau) \tag{17.3}$$

where λ_u is the average number of call requests per unit time (calls per time) made by a single user. If we consider a population of U users, the distribution of the time interval between two consecutive call requests, made by any two users, is also exponential. The average number of call requests is $\lambda = U\lambda_u$.

2. The call duration is also a random variable that follows an exponential distribution, such that short calls are more likely to occur than long calls. Denoting the duration of a call by s, the pdf of s is

$$f_S(s) = \mu \, \exp(-\mu s) \tag{17.4}$$

where $1/\mu = H$ is the average call duration (units of time).

Based on this statistical behavior, a large number of users can share a relatively small number of channels in a pool of channels. For each base station in a cellular system, a pool of C trunked channels is made available to all users that are within the coverage area of the base station. Since a single user does not require access to the cellular system at all times, channels can be allocated to users on a per call basis. Once the call is terminated, the channel returns to the pool of available channels. However, one can intuitively expect that a single user may not always be allowed to establish radio communication with its serving base station because of a lack of available radio channels at the base station. In this case, all channels would be busy serving calls placed by other users, and the call request is *blocked*. Based on the statistical behavior of the users, the number of channels available in the pool, and certain characteristics of the trunked system, we can determine the probability that a user will have its call request blocked due to a lack of idle channels. This probability, usually called blocking probability, is a measure of the "grade of service" of a trunked system. The statistical behavior of a single user can be summarized by the traffic A_u generated by that user, given in Erlangs and defined as

$$A_u = \lambda_u \, H \tag{17.5}$$

For a system containing U users, the total offered traffic in Erlangs is

$$A = UA_u = \lambda \, H \tag{17.6}$$

An important characteristic of the trunked system that dictates the quality of the service offered to the users concerns how the system handles blocked calls. There are two basic strategies. In the first strategy, call requests that do not find available free channels are blocked and cleared. In this case the trunked system is referred to as *blocked calls cleared*. In the second strategy, blocked calls are held in a queue and served as soon as a channel becomes available. Trunked systems using this strategy are called *blocked calls delayed*. We will focus our attention on the blocked calls cleared trunked system, since this type of system is more often found in practice.

We therefore assume that blocked calls are cleared. In addition, the following assumptions are made:

- Call arrivals are memoryless. In other words, any user, including the users that had previous calls blocked, can request a call at any time.

- There are an infinite number of users.
- There are C trunked channels available in the pool.

Under these conditions the probability P_B that a call is blocked is given by the *Erlang B* formula. The Erlang B formula relates the number of trunked channels C, the blocking probability P_B and traffic A, which can be either the *offered* traffic or the *carried* traffic. In the former case, P_B is the blocking probability experienced by the population of users that generates the traffic A, *offered* to a trunked system with C channels. In the later case, A is the maximum *carried* traffic by a trunked system containing C channels at blocking probability P_B. The carried traffic by a trunked system is also a measure of the capacity of the system. The Erlang B formula is

$$P_B = \frac{A^C/C!}{\sum_{k=0}^{C} A^k/k!} \tag{17.7}$$

Given the offered traffic and the number of trunked channels, the blocking probability can be computed using the following MATLAB code

```
function erb = erlang_b(A,c)
% A = offered traffic in Erlangs.
% c = number of trunked channels.
num = A^c;
sum = 0;
for k=0:c
    kfact = prod(1:k);
    term = (A^k)/kfact;
    sum = sum + term;
end
cfact = prod(1:c);
den = cfact*sum;
erb = num/den;
```

Figure 17.3 shows the Erlang B chart with the number of trunked channels as a parameter. The number of trunked channels, C, is given across the top of the chart. The complete MATLAB program used to compute Figure 17.3 is given in Appendix A. The code listed in Appendix A makes repeated calls to the previously listed function `erlang_b(A,c)`. Given the Erlang B formula, the effects of the cluster size of a cellular radio system on the capacity of the system, in terms of the number of users, can be evaluated.

Example 17.1. Consider a cellular system where 400 pairs of forward and reverse channels are available for the entire system. Each cell has a radius of 5 km, and the base stations are equipped with omnidirectional antennas that are assumed to be located at the center of each cell. Assume that each user generates a traffic of 0.02 Erlang and that a cluster size $N = 7$ is used. Assuming that the distribution

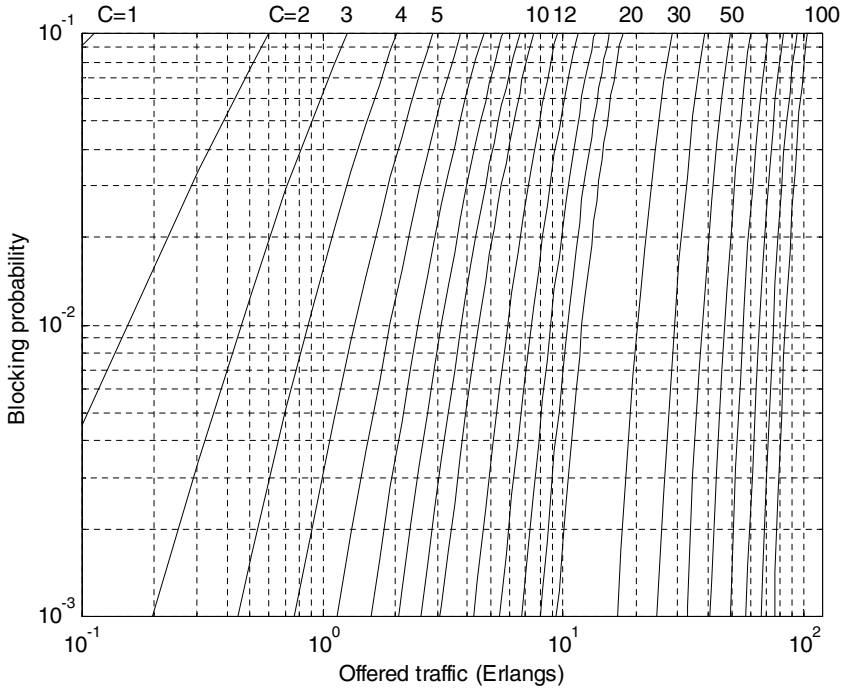


Figure 17.3 Blocking probability given by the Erlang B formula.

of users is uniform over the service area, each cell is allocated $N_C = 400/7 \approx 57$ channels. If we further assume that blocked calls are cleared and that a blocking probability $P_B = 0.02$ is acceptable, the maximum traffic A_C carried by each cell is given by Erlang B formula. The result is

$$A_C = 46.8 \quad \text{Erlangs per cell}$$

The number of users supported per cell is easily computed. Since $A_u = 0.02$ Erlang per user, the number of users supported per cell is

$$U = \frac{A_C}{A_u} = 2340 \quad \text{users per cell}$$

Now reduce the cluster size to $N = 3$ without changing the coverage area of each cell. Each cell will now be allocated $N_C = 400/3 = 133$ channels at $P_B = 0.02$. This gives

$$A_C = 120.1 \quad \text{Erlangs per cell}$$

for the maximum carried traffic per cell. This gives

$$U = \frac{A_C}{A_u} = \frac{120.1}{0.02} = 6005 \quad \text{users per cell}$$

We see that we achieve higher capacity by reducing the cluster size due to increased reuse and trunking efficiency. However, co-channel cells are closer to each other in cluster size $N = 3$ than in cluster size $N = 7$. Using (17.1) we have

$$D_3 = 15 \quad \text{km}$$

for $N = 3$ and

$$D_7 = 22.9 \quad \text{km}$$

for $N = 7$. We intuitively see that the level of interference will be higher in cluster size $N = 3$ than in cluster size $N = 7$. Therefore, cluster size $N = 3$ will present a lower link quality. ■

Channel Model

When analyzing the performance of a cellular system, it is very important to accurately model the effects of the radio propagation on the received signal, as those effects are very often among the major sources of system performance degradation. As discussed in Chapter 14, the statistical characterization of the received signals (both desired and interference signals) involves mainly two propagation effects: *small-scale fading*, induced by multipath over a local area, and *shadowing* (large-scale fading), induced by random attenuators of the local mean signal, such as trees, buildings, and terrain [1]. Measurements have shown that the local mean signal level in a wireless communications system [5, 6] can be accurately modeled as a lognormal random variable. When expressed in decibel units, the local mean signal level follows a normal variation and is characterized by an area mean value and standard deviation, both in dB. The area mean value is a function of the transmitter to receiver separation (T-R) distance, transmitter power levels, and antenna gains, while the shadowing standard deviation depends on the physical environment. In the general case of system design or simulation, the effects of small-scale fading and shadowing must be taken into consideration, although in some cases, shadowing of the desired and interference signals is the main source of performance degradation. For example, spatial diversity, spread spectrum, and coding and interleaving techniques have been extensively employed to combat the effects of small-scale fading [1], such that received signals are mainly dependent on large-scale channel variations. In the analysis presented here, we assume, for simplicity, that the small-scale fading effects are averaged out and only shadowing and path loss are considered. However, the effects of small-scale fading can be easily incorporated in the analysis.

Assuming that the effects of small-scale fading are averaged out, the local mean power level of the desired or individual interference signal, denoted generically here by ρ, undergoes lognormal variation. In dBW, the local mean power level can be modeled as [1]

$$X = 10 \log_{10} \rho = m_X + \chi \quad \text{dBW} \tag{17.8}$$

where m_X is the area mean power level in dBW (or, alternatively, average large-scale propagation path loss in dB) and χ is a zero-mean normally distributed random

variable in dB with standard deviation σ_X, also in dB, due to the shadowing caused by large obstacles [1]. The area mean power m_X is usually modeled as a function of the T-R separation d, path loss exponent γ, transmitted power P_T in dBW, and transmitter and receiver antenna gains G_T and G_R, both in dB. Specifically

$$m_X = P_T + G_T(\theta_T, \phi_T) + G_R(\theta_R, \phi_R) - 10K\gamma \log_{10} d \quad \text{dBW} \tag{17.9}$$

The constant K in (17.9) comprises all terms that do not change in the model. The angles θ_T and ϕ_T are the elevation and azimuth angles of departure of the signal transmitted toward the receiver, while θ_R and ϕ_R are the elevation and azimuth angles of arrival of the signals impinging on the receiving antenna. Angles θ_T, ϕ_T, θ_R, and ϕ_R depend on the relative locations of the base station and the mobile antennas.

Example 17.2. Consider that a mobile station is receiving a signal from a base station as depicted in Figure 17.4(a). The T-R separation is $d = 1{,}200$ meters. Let us assume that the communication link between the base station and mobile occurs through line of sight. Assuming the coordinate systems shown in the figure and

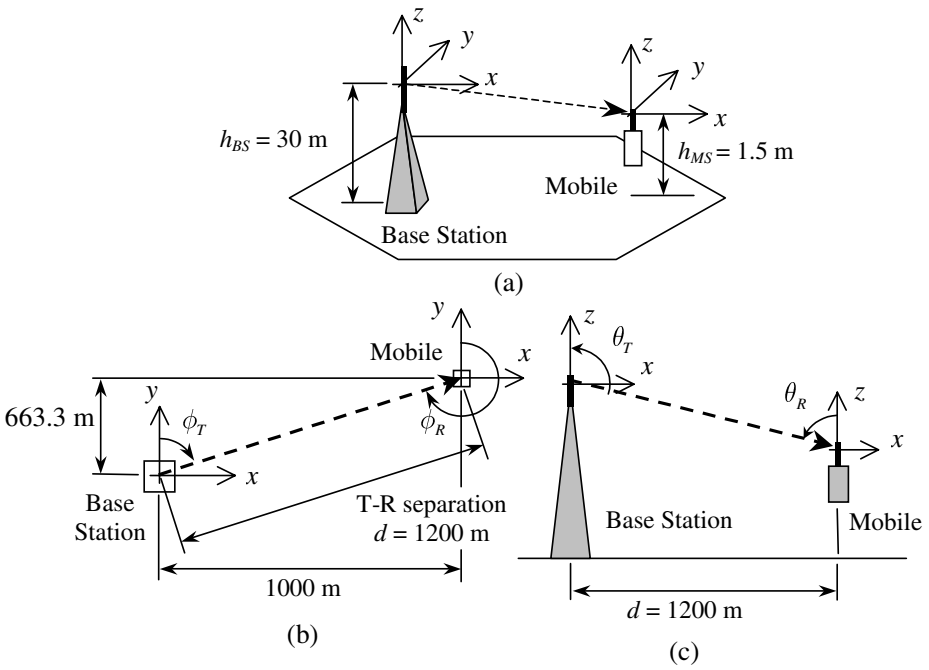


Figure 17.4 Angles of departure and arrival.

using simple geometry, we can determine the azimuth angle of the signal transmitted by the base station [see Figure 17.4(b)]:

$$\phi_T = \cos\left(\frac{663.3}{1200}\right) = 56.4^\circ$$

Likewise, the azimuth angle of the signal impinging on the mobile antenna, according to the coordinate system adopted, is

$$\phi_R = 236.4^\circ$$

Suppose that the base station antenna is 30 meters high, and that the user holds the mobile telephone 1.5 meters above the ground, as also shown in the figure. Using again simple geometry, the elevation angle of the signal transmitted by the base station toward the mobile is

$$\theta_T = 180^\circ - \tan\left(\frac{1200}{30 - 1.5}\right) = 91.6^\circ$$

Likewise for the elevation angle of the signal arriving at the mobile antenna, we have

$$\theta_R \approx 88.4^\circ$$

■

We see that the azimuth and elevation angles depend on the relative locations of the transmitter and receiver antennas, and the coordinate system used, as well. In the simulation of cellular systems, the coordinate systems adopted to specify all angles and distances must be carefully defined [11].

Another important conclusion from the preceding example is that, for macrocellular systems, where the cell radius is larger than 1 km and, consequently, T-R separations are much larger than the difference between the heights of the base station and mobile antennas, we can assume that

$$\theta_T \approx \theta_R \approx 90^\circ \tag{17.10}$$

and omit both θ_T and θ_R in (17.9). This gives

$$\begin{aligned} G_T(\theta_T, \phi_T) &\Rightarrow G_T(\phi_T) \\ G_R(\theta_R, \phi_R) &\Rightarrow G_R(\phi_R) \end{aligned} \tag{17.11}$$

Sectorized Cells

Cellular communication systems often use several sectorized antennas at the base station, in order to reduce the co-channel interference. Each sectorized antenna radiates within a specified sector of the cell, and each sector is allocated a subset of the set of channels available for the cell. Therefore, due to the directivity of the

base station antennas, the total co-channel interference impinging on the receiver antenna (at the base station or mobile) is reduced, as illustrated in the next example.

Example 17.3. Consider a cellular system with the base stations equipped with sectorized antennas on both links. Assume a beamwidth $BW = 120°$ and that the front-to-back ratio of the sectorized antennas is infinity, so that no power is radiated out of the beamwidth. Also assume that cluster size $N = 4$ is used. Considering first the forward link, the number of interfering base stations in the first tier reduces from six to only two (base stations at Cells 5 and 6), as shown in Figure 17.5, due to the directivity of the base station antennas. On the reverse link, the number of co-channel mobiles in the first tier interfering with the base station at the center cell also drops from six to two (mobiles at Cells 2 and 3). Therefore, sectoring is an effective way for reducing co-channel interference. It is clear that the amount of reduction in co-channel interference depends on the beamwidth of the sectorized antenna and the cluster size used. ■

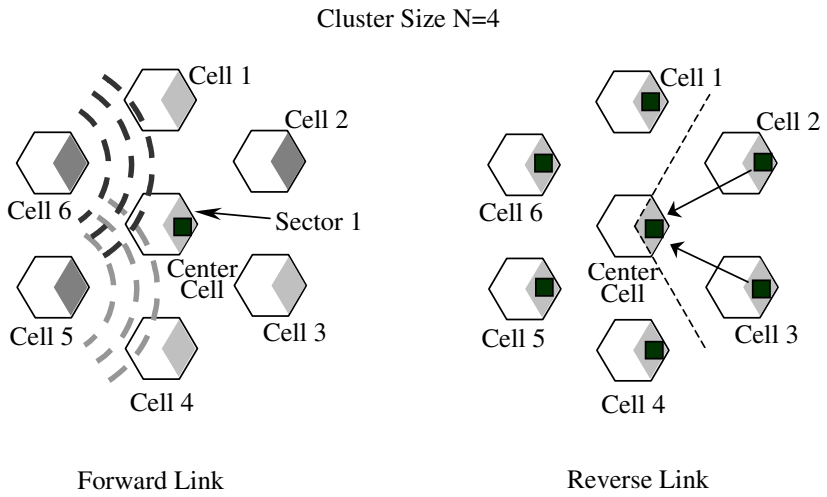


Figure 17.5 Sectorized antennas on both the forward and reverse links.

Actual sectorized antennas used in practice have a finite front-to-back ratio. As a result, the number of co-channel interference signals from co-channel cells in the first tier would still be six, but some of them will be very weak, since they are attenuated by the front-to-back ratio. Also, sectorized antennas commonly found in practice have beamwidths of 120° (three sectors), or 60° (six sectors).

Total Co-Channel Interference

Based on the assumption that the received individual interference signals are affected by shadowing and path loss only, the total co-channel interference is modeled

as a composition of interference signals, whose local mean power levels follow lognormal variations. It is usually assumed that the phase shift observed in each individual interference signal varies significantly due to scattering, such that we can assume that the phases are random and thus signals add incoherently (e.g., their powers add) when averaged over the local area. Therefore, the total co-channel interference I received at a given location is modeled as the *sum* of lognormally distributed signals. Thus

$$I = \sum_{i=1}^{k} I_i, \tag{17.12}$$

where I_i, when expressed in decibel units, is modeled as in (17.8).

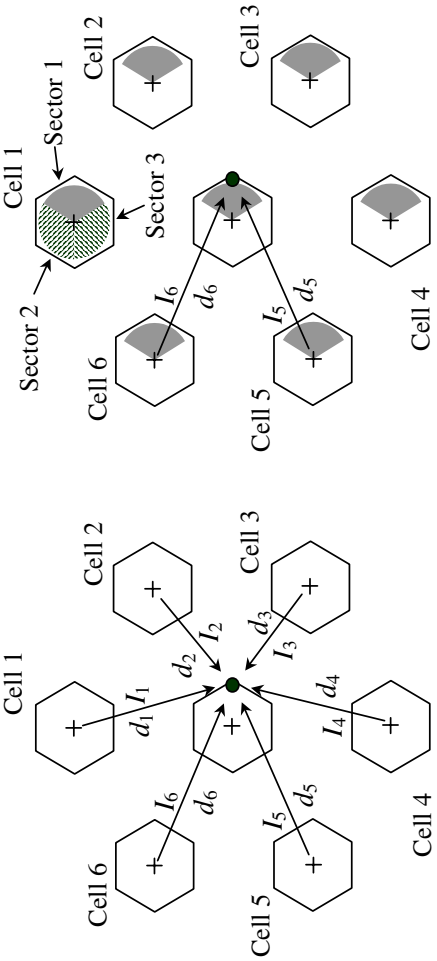
It is well accepted that the distribution of the sum of lognormal random variables can be approximated by another lognormal distribution [7, 8, 9], and several methods have been proposed for computing the mean value and standard deviation, in dB, of the resulting lognormal distribution. The two most popular techniques, as discussed in Appendix C, are Wilkinson's method [7] and Schwartz and Yeh's method [8]. Once we know the distributions of the individual co-channel interference signals I_i, or in other words, the means m_{I_i} and standard deviations σ_{I_i} of I_i, we can compute the mean m_I and standard deviation σ_I of the total interference I, by using Wilkinson's method or Schwartz and Yeh's method.

Effects of Sectoring

Since we now know how to compute the total co-channel interference, let us analyze more closely the effects of sectoring on capacity and link quality of a cellular system. We have seen that cell sectoring reduces co-channel interference, but at the expense of a reduction in trunking efficiency which, in turn, reduces the total traffic carried by the cell. Each sector of the cell will be allocated a subset of the set of channels allocated to the cell. From traffic theory, we know that when a pool of channels is partitioned into subsets of channels, the sum of the maximum traffic carried by the subsets is always lower than the maximum traffic carried by the whole pool of channels.

Example 17.4. Consider an AMPS (Advanced Mobile Phone System) cellular system using a cluster size of $N = 4$. In addition, assume that 395 pairs of forward and reverse traffic channels are available. In this example the maximum traffic carried by each cell and the co-channel interference will be estimated. Two different configurations will be considered: (1) omnidirectional antennas at the base station and (2) sectorized antennas at the base station. Only the forward link will be analyzed.

1. For onmidirectional antennas, base stations will radiate in all directions with equal strength. They will interfere with and will receive interference from all co-channel cells. Considering only the first tier of co-channel cells, the total co-channel interference received at a given mobile is the sum of all co-channel interference signals, as shown in Figure 17.6(a). Based on this figure, we can



(a) Omnidirectional cells (b) Sectorized cells

Figure 17.6 Cellular system with omnidirectional and sectorized base station antennas.

compute the area mean *SIR* at a mobile located at the boundary of the cell. This is a worst-case situation and is given by

$$SIR_{mean} = m_S - m_I \quad \text{dB} \tag{17.13}$$

where m_S is the area mean power of the desired signal and m_I is the area mean power of the total interference (note that both are expressed in dB which explains the substraction). In order to compute m_S and m_I, we will use (17.9) and (17.12), assuming for simplicity, that the mobile antenna is omnidirectical ($G_R(\theta_R, \phi_R) = 0$ dB), $K = 1$, and all base stations transmit the same power level $P_T = 0$ dBW. Therefore

$$m_S = -10 \log_{10} R \tag{17.14}$$

and

$$m_I = 10 \log_{10} \left(\sum_{i=1}^{6} 10^{(-10\gamma \log_{10} d_i)/10} \right) \tag{17.15}$$

where R is the cell radius, γ is the path loss exponent, and d_i, $i = 1, \cdots, 6$, are the T-R separations. Using the geometry of cluster size $N = 4$, assuming that $R = 1{,}000$ and $\gamma = 4$, we can show that

$$d_1 = d_4 = 3605.55 \quad \text{m} \tag{17.16}$$

$$d_2 = d_3 = 2645.75 \quad \text{m} \tag{17.17}$$

and

$$d_5 = d_6 = 4358.90 \quad \text{m} \tag{17.18}$$

Therefore

$$m_S = -120.0 \quad \text{dBW} \tag{17.19}$$

and

$$m_I = -132.35 \quad \text{dBW} \tag{17.20}$$

which gives the result

$$SIR_{mean}^{\text{omni}} = 12.35 \quad \text{dB} \tag{17.21}$$

For the traffic analysis, we again assume $N = 4$ and also assume that the distribution of users is uniform over the cell area. Each cell is allocated $N_C = 395/4 \cong 98$ channels. Assuming that blocked calls are cleared and that the blocking probability is $P_B = 0.02$, the maximum traffic carried per cell is, from the Erlang B formula

$$A_C^{\text{omni}} = 86.0 \quad \text{Erlangs per cell} \tag{17.22}$$

If a single user generates 0.02 Erlang of traffic, the omnidirectional cell can support up to $86.0/0.02 \cong 4{,}300$ users at a blocking probability of $P_B = 0.02$.

2. We now assume that 120° sectoring is used (three sectors per cell) as shown in Figure 17.6(b). In order to analyze the co-channel interference the same approach used in part 1 is applied. We assume an idealized antenna with an infinite front-to-back ratio so that only Cells 5 and 6 interfere with the center cell. Therefore, the number of interfering co-channel cells in this idealized example drops from six (all co-channel cells in the first tier) to only two, and

$$\begin{aligned} m_I &= 10\log_{10}\left(10^{(-10\gamma\log_{10} d_5)/10} + 10^{(-10\gamma\log_{10} d_6)/10}\right) \\ &= -142.57 \quad \text{dBW} \end{aligned} \tag{17.23}$$

and

$$m_S = -120.0 \text{ dBW} \tag{17.24}$$

Finally

$$SIR_{mean}^{\text{sect}} = 22.6 \quad \text{dB} \tag{17.25}$$

For the capacity traffic analysis, we recognize that each sector is allocated $N_S = 395/(4 \times 3) \cong 33$ channels. The maximum traffic carried by each sector is, assuming a blocking probability of $P_B = 0.02$, given by

$$A_S^{\text{sect}} = 23.7 \quad \text{Erlangs per sector} \tag{17.26}$$

For a three-sector cell this number is multiplied by three. Therefore:

$$A_C^{\text{sect}} = 71.1 \quad \text{Erlangs per cell} \tag{17.27}$$

Again assuming that each user generates 0.02 Erlang of traffic, the sectorized cell with three sectors can support up to $71.1/0.02 \cong 3{,}550$ users at a blocking probability of $P_B = 0.02$.

We see from these results that a link quality improvement of approximately 10 dB ($\Delta SIR = 22.56 - 12.35 = 10.21$ dB) is achieved by using sectoring, compared with the omnidirectional case. However, this link quality improvement is achieved at the expense of trunking efficiency, so that the total traffic carried per cell is reduced from 86.0 Erlangs per cell with the omnidirectional antenna to 71.1 Erlangs per cell with sectoring. After sectoring N can be reduced if desired. ■

In the preceding example, the signal-to-interference ratio was computed for the worst-case situation, since the mobile was assumed to be located at the cell boundary. It is obvious that at locations closer to the serving base station, the *SIR* will be higher. Another limitation of the simplified analysis presented in the preceding example is that the effects of shadowing were not considered. When we consider both the spatial distribution of mobiles and the effects of shadowing, *SIR* becomes a random variable, as we discussed before. The performance of the cellular system must then be measured through the *outage probability*, defined in (17.2) as the probability that *SIR* is below a minimum acceptable level SIR_0.

In the next section, the methodology for simulating a simple cellular system is presented, taking into consideration both the user spatial distribution and shadowing effects on the received signals. The results of the simulation will be the *SIR* statistics, that will give us an estimate of the performance of the system. Despite the simplicity of the simulation, the methodology presented can be viewed as the core part of more complex simulations, such as those described by Cardieri and Rappaport [10, 11].

17.3 Simulation Methodology

The simulation procedure consists of modeling a snapshot of the location of mobile stations. At each snapshot, the statistics (mean value and variance) of the *SIR* at a given base station (reverse link) and mobile station (forward link) are computed, taking into account user locations and propagation conditions. Several snapshots are simulated, in order to generate a sufficiently large sample set so that statistically valid results are achieved. Figure 17.7 shows the flowchart of the procedure. In the following sections, we describe some aspects of the simulation. Other aspects will be described subsequently, together with the steps required to simulate one snapshot of the system.

17.3.1 The Simulation

We will consider only the first tier of co-channel cells. The number of sectors in each cell is a simulation parameter and can be chosen among: (1) one, or omnidirectional antennas, (2) three, or 120° sectoring, and (3) six, or 60° sectoring. The gain of the sectorized antenna is assumed to be constant within the sector and equal to 0 dB, as shown in Figure 17.8. The sectorized antennas have a finite front-to-back ratio of B dB, which is a simulation parameter. For 120° sectoring, the radiation patterns of the sectorized antennas for Sector 1 are modeled as

$$G_T^{(1)}(\phi) = G_R^{(1)}(\phi) = \begin{cases} 0 \text{ dB}, & -\pi/3 < \phi \leq \pi/3 \\ B \text{ dB}, & \text{otherwise} \end{cases} \tag{17.28}$$

The radiation pattern for Sectors 2 and 3 are obtained by rotating $G_T^{(1)}(\phi)$ and $G_R^{(1)}(\phi)$ 120° and 240°, respectively. For 60° sectoring, we have

$$G_T^{(1)}(\phi) = G_R^{(1)}(\phi) = \begin{cases} 0 \text{ dB}, & 0 < \phi \leq \pi/3 \\ B \text{ dB}, & \text{otherwise} \end{cases} \tag{17.29}$$

The radiation pattern for Sectors 2 through 6 are obtained by appropriate rotation of $G_T^{(1)}(\phi)$ and $G_R^{(1)}(\phi)$ by 60°. Both forward link and reverse link base station antennas are assumed to be identical. Mobile antennas are assumed to be omnidirectional antennas on both forward and reverse links.

The channel model includes both path loss and lognormal shadowing. The path loss exponent and the standard deviation of the lognormal shadowing are simulation parameters. The simulation consists of the following two major steps:

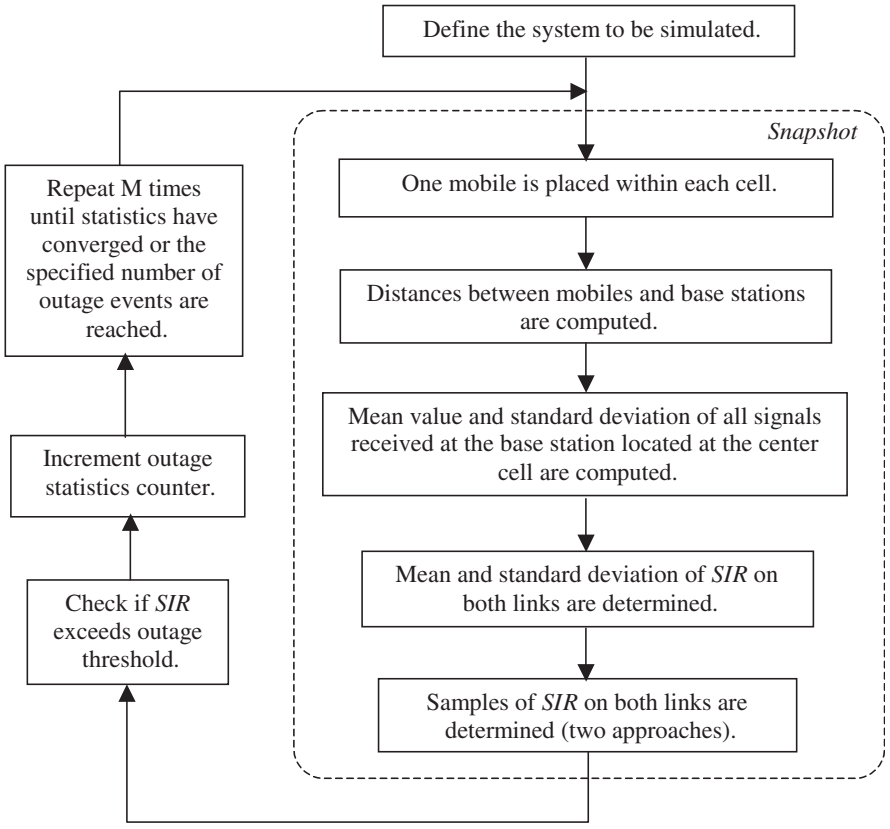


Figure 17.7 Flowchart to estimate the *SIR* and outage in a cellular system using Monte Carlo simulation.

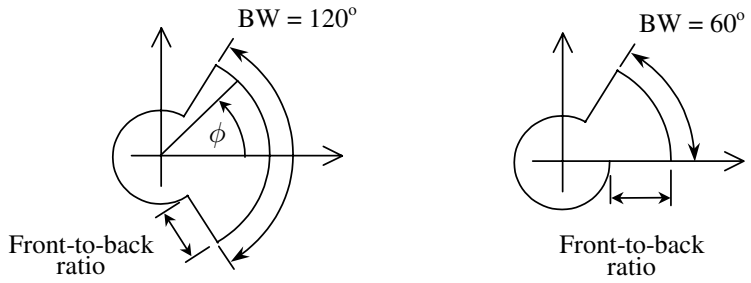


Figure 17.8 Model for sectorized antenna for 120° and 60° sectoring.

- Definition of the target system to be simulated
- Generation of snapshots of mobile locations and computation of the *SIR*

These are discussed in the following sections.

Definition of the Target System to Be Simulated

Here we define the propagation characteristics (channel parameters) and the location of the co-channel cells.

Propagation characteristics (channel parameters). The parameters that define the channel characteristics consist of the following:

- Cell radius R
- Path loss exponent (γ)
- Standard deviation in decibel units of lognormal shadowing (σ)
- Base station transmission power level ($P_{T,BS}$)
- Mobile station transmission power level ($P_{T,MS}$)
- Number of sectors per cell
- Front-to-back ratio of sectorized antennas
- Number of snapshots to be simulated (M)

This part of the simulation can be implemented using the MATLAB code given in Appendix B. The complete simulation program presented in this section can be assembled by appending the code segments appearing later in this section to the MATLAB code given in Appendix B.

Locations of co-channel cells. For convenience, we will adopt both the rectangular and polar coordinate systems to represent the locations of the base stations and mobiles in our simulation. The base stations will be located at the center of the corresponding cells. The base station at the center cell, where the co-channel interference will be measured, will be located at the center of the coordinate systems. The locations of the co-channel cells depend on the cluster size N of the cellular system and the cell radius R. For the first tier, all co-channel cells are located on a circumference of radius $D = \sqrt{3N}R$ centered at the center cell, and equally distant from each other, as indicated in Figure 17.9. Also indicated in this figure is the angle θ_N (see Table 17.1), which determines the angular location of the first co-channel cell. Using simple geometry, we can show that, for cluster size N, the location of the i^{th} co-channel cell, using vector notation, is

$$x_{BS_i}\,\hat{\mathbf{x}} + y_{BS_i}\,\hat{\mathbf{y}} = \sqrt{3N}\,R\,\{\cos[\theta_N + (i-1)\pi/6]\,\hat{\mathbf{x}} + \sin[\theta_N + (i-1)\pi/6]\,\hat{\mathbf{y}}\} \tag{17.30}$$

where $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ are the unit vectors in the direction of axes x and y, respectively. An example of the implementation of this part of the simulation is as follows:

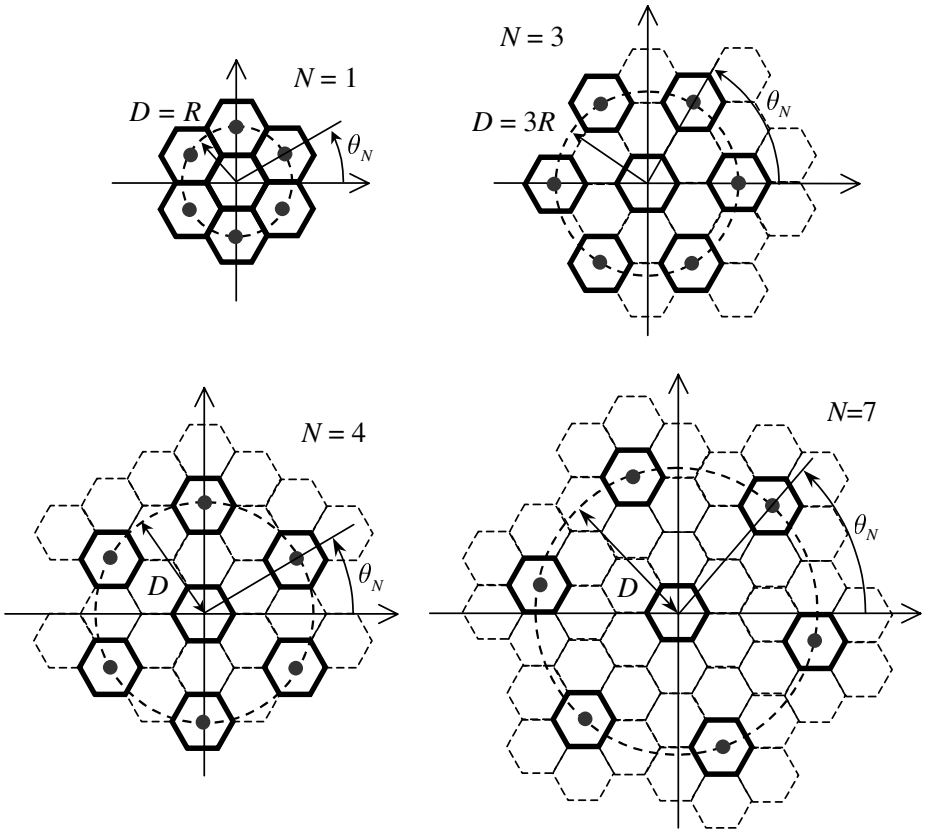


Figure 17.9 Location of co-channel cells or $N = 1$, $N = 3$, $N = 4$, and $N = 7$.

Table 17.1 Relationship between N, R, and θ_N in Figure 17.9

cluster size N	D	θ_N
1	R	$\pi/6$
3	$3R$	0
4	$2\sqrt{3}R$	$\pi/6$
7	$\sqrt{21}R$	$\arctan\left(2/\sqrt{3}\right)$

```
% Location of base stations (center cell is located at x = 0, y = 0)
% Location (angular) of the center cell of each cluster in the
% first tier.
theta_N = [pi/6 0 pi/6 asin(1/(2*sqrt(7)))];
% Angular distance between the center cells of all 6 clusters in
% first tier.
theta = pi/3*[0:5]';
aux_1 = [1 0 2 3 0 0 4];
ind = aux_1(cluster_size);
% Location [x,y] of the center cells of all clusters in the
% first tier.
bs_position = [sqrt(3*cluster_size)*r_cell*cos(theta + ...
    theta_N(ind)) sqrt(3*cluster_size)*r_cell*sin(theta + ...
        theta_N(ind))];
```

Note that, in this MATLAB program segment, `bs_position(i,1)` and `bs_position (i,2)` correspond to the components in the directions $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$, respectively, of the vector representing the location of the i^{th} base station.

Generation of Snapshots of Mobiles' Locations and Computation of *SIR*

In this part of the simulation, the actual Monte Carlo evaluation is performed. Snapshots of the locations of the mobiles sharing the same channel are generated, and for each snapshot, the statistics of *SIR* are computed. This part of the simulation is performed M times.

Step 1: A mobile is placed within each cell. The mobiles are assumed to be uniformly distributed over the cell area. As mentioned before, the sectors of a given cell are allocated different sets of channels. Therefore, co-channel interference occurs only among sectors allocated the same set of channels. In our simulation, the mobiles at the center cell and at the co-channel cells are assumed to be located within the same sector in their cells. Using a polar coordinate system, the location of the i^{th} mobile within its cell can be described by the distance r_i between the mobile and its serving base station, and the angle β_i between a reference and the direction of propagation between the mobile and its base station (see Figure 17.10). Note that β_i and r_i are defined with respect to the coordinate system centered at the base station of the i^{th} cell. Since the distribution of the mobile location is uniform over the cell area, β_i is uniformly distributed over the range $[0, 2\pi]$, and the distance r follows the pdf

$$f_{R_i}(r_i) = 2r_i/R^2, \qquad 0 \le r_i \le R. \tag{17.31}$$

Note that, for simplicity, the cell is assumed to be circular in the simulation.

The sector used in a particular snapshot is chosen randomly, with equal probabilities of choosing any sector:

$$\begin{aligned} 120^\circ \text{ sectoring} &\Rightarrow \Pr\{\text{sector } i \text{ is chosen}\} = 1/3, \qquad i = 1, 2, 3. \\ 60^\circ \text{ sectoring} &\Rightarrow \Pr\{\text{sector } i \text{ is chosen}\} = 1/6, \qquad i = 1, \cdots, 6. \end{aligned}$$

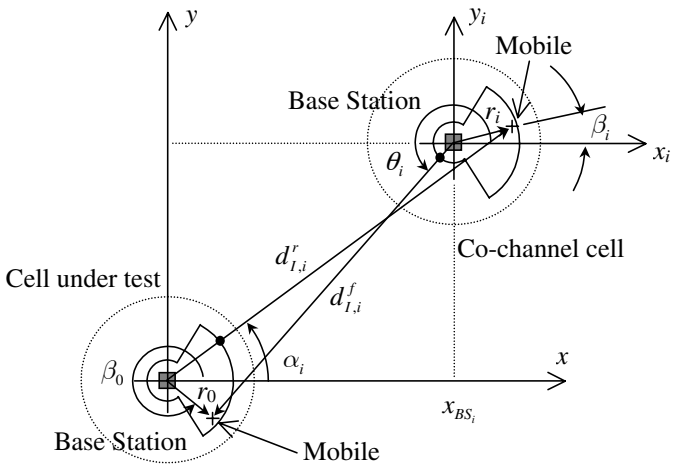


Figure 17.10 Representation of mobile location.

Once the sector is chosen, the angles β_i can be determined, noting that β_i is uniformly distributed over the chosen sector:

$$120^\circ \text{ sectoring} \Rightarrow \ (2s-3)\pi/3 < \beta_i \leq (2s-1)\pi/3$$
$$60^\circ \text{ sectoring} \Rightarrow \ (s-1)\pi/3 < \beta_i \leq s\pi/3,$$

where s is the sector selected ($s = \{1, 2, 3\}$, for 120° sectoring and $s = \{1, \cdots, 6\}$, for 60° sectoring). Figure 17.11 shows an example of a snapshot for 120° sectoring, where the mobiles are located in Sector 1. This part of the simulation can be implemented as shown in the following MATLAB code:

```
% Determination of the sector to simulated in this snapshot
%
% --- Select (randomly) a sector ---
sector = unidrnd(num_sectors(sec));
%
% --- Place the desired mobile within the select sector ---
des_user_beta = rand(1)*phi_BW(sec) + phi_center(sector,sec);
des_user_r = sqrt(rand(1).*(r_cell^2));
%
% --- Place co-channel mobiles within the selected sector of
% co-channel cells---
co_ch_user_beta = rand(6,1)*phi_BW(sec) + phi_center(sector,sec);
co_ch_user_r = sqrt(rand(6,1))*(r_cell);
```

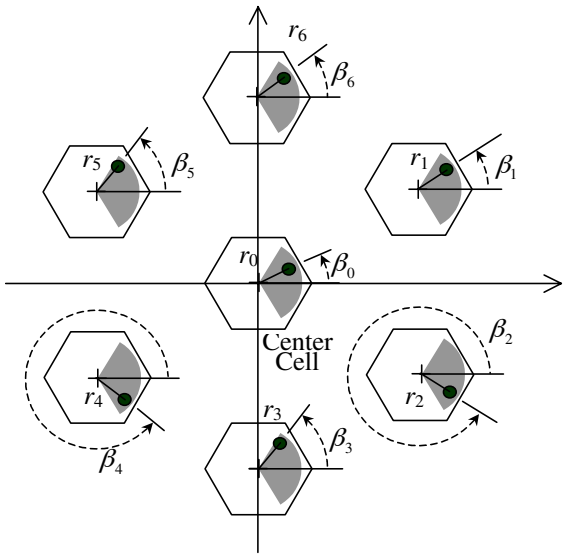


Figure 17.11 Snapshot for 120° sectoring with mobiles assumed to be in Sector 1.

The `(x,y)` location of the the desired and co-channel mobiles are computed according to

```
des_user_position = des_user_r*[cos(des_user_beta) ...
    sin(des_user_beta)];
co_ch_user_position = [co_ch_user_r.*cos(co_ch_user_beta) ...
co_ch_user_r.*sin(co_ch_user_beta)] + bs_position;
```

The next step is to determine the distances between the co-channel mobiles to the base station at the center cell (to be used to compute the mean values of the interference signals on the reverse link) and the distances between the mobile at the center cell and the co-channel base stations (to be used to determine the mean values of the interference signals on the forward link).

Step 2: Determination of the distances between mobiles and base stations. Since we are interested in computing the co-channel interference at the base station and mobile in the center cell, we need to determine the following:

- *The location of the co-channel mobile stations with respect to the base station at the center cell:* Using vector notation, the location of the i^{th} mobile station with respect to the base station at the center cell is (see Figure 17.10)

$$d^r_{I,i}\,(\cos\alpha_i\,\hat{\mathbf{x}} + \sin\alpha_i\,\hat{\mathbf{y}}) = r_i\,(\cos\beta_i\,\hat{\mathbf{x}} + \sin\beta_i\,\hat{\mathbf{y}}) + (x_{BS_i}\,\hat{\mathbf{x}} + y_{BS_i}\,\hat{\mathbf{y}}). \tag{17.32}$$

The term $d^r_{I,i}$ is the T-R separation between the base station at the center cell and the i^{th} mobile, and α_i is the angle of arrival of the signal from the i^{th} mobile, impinging on the base station antenna. Therefore, $d^r_{I,i}$ and α_i are the length and direction of the vector given by the right hand side of (17.32).

- *The location of the mobile station at the center cell with respect to the i^{th} co-channel base station:* Again using vector notation, we have (see Figure 17.10)

$$d^f_{I,i}\,(\cos\theta_i\,\hat{\mathbf{x}} + \sin\theta_i\,\hat{\mathbf{y}}) = r_0\,(\cos\beta_0\,\hat{\mathbf{x}} + \sin\beta_0\,\hat{\mathbf{y}}) - (x_{BS_i}\,\hat{\mathbf{x}} + y_{BS_i}\,\hat{\mathbf{y}}), \tag{17.33}$$

where $d^f_{I,i}$ is the T-R separation between the i^{th} base station and the mobile at the center cell, and θ_i is the angle of departure of the signal transmitted by the i^{th} base station toward the mobile at the center cell.

Step 3: Determination of the statistics of *SIR* on both links. In this step, three different approaches for computing the statistics of *SIR* will be presented. In all three approaches, the lognormal variation of the received signals will be taken into account. This requires the mean value of the desired and interference signals.

- *Mean value and standard deviation in dB of each signal*

  1. *Desired signals:* Using (17.9), the mean values in decibel units of the desired signals on the forward and reverse link are

$$m^f_S = P_{T,BS} - 10\gamma\log_{10} r_0 \quad \text{dBW} \tag{17.34}$$

  and

$$m^r_S = P_{T,MS} - 10\gamma\log_{10} r_0 \quad \text{dBW} \tag{17.35}$$

  respectively. Note that the base station antenna gains on both links in these expression are set to 0 dB, since the mobile is located within the sector of the base station. Also, the mobile antenna gains on both links are set to 0 dB, since we are assuming omnidirectional antennas. The standard deviations of the desired signals are equal to the shadowing standard deviation. Thus

$$\sigma^f_S = \sigma^r_S = \sigma$$

  2. *Interference signals:* The determination of the mean values of the total interference signals on both links is more involved. As we mentioned before, the total co-channel interference on both links is modeled as the sum of the individual co-channel interference signals. This gives

$$I^f = \sum_i I^f_i \tag{17.36}$$

and

$$I^r = \sum_i I_i^r \tag{17.37}$$

for the forward and reverse links, respectively. Figure 17.12 shows an example snapshot for 120° sectoring. Six co-channel cells are simulated but, for clarity, only signals from co-channel Cells 1 and 4 are shown. Since shadowing effects and path loss are taken into account in this model, the total interference signals I^f and I^r are the sum of the lognormally distributed signals I_i^f and I_i^r, respectively. As discussed before, we assume that both I^f and I^r are lognormally distributed. The mean and standard deviation, in dB, of I^f and I^r are functions of the means and standard deviations of the individual interference signals and can be computed using Wilkinson's or Schwartz and Yeh's methods, as discussed in Appendix C. The mean values, in dBW, of I_i^f and I_i^r can be determined using (17.9). This gives

$$m_{I,i}^f = P_{T,BS} + G_{T,i}(\phi_{T,i}) - 10\gamma \log_{10} d_{I,i}^f \quad \text{dBW} \tag{17.38}$$

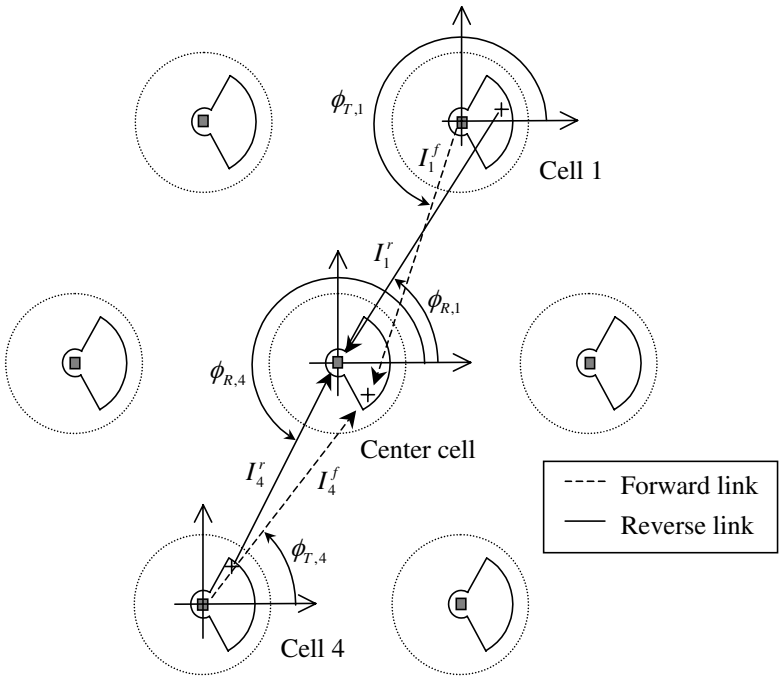


Figure 17.12 Example of a snapshot for 120° sectoring.

and

$$m_{I,i}^r = P_{T,MS} + G_{R,0}(\phi_{R,i}) - 10\gamma \log_{10} d_{I,i}^r \quad \text{dBW} \tag{17.39}$$

The antennas gains $G_{T,i}(\phi_{T,i})$ and $G_{R,0}(\phi_{R,i})$ depend on the relative positions of the mobiles and follow the description given in (17.28) and (17.29). Note that we are assuming omnidirectional antennas at the mobiles on both links. The standard deviations of the interference signals are equal to the shadowing standard deviation σ

$$\sigma_{I,i}^f = \sigma_{I,i}^r = \sigma \text{ (all cells)} \tag{17.40}$$

Once the means and standard deviations of all co-channel signals have been determined, we apply Wilkinson's or Schwartz and Yeh's method (see Appendix C) to compute the means m_I^f and m_I^r, and standard deviations σ_I^f and σ_I^r, in dB, of the total co-channel interference on both links. The computation of the moments of the signals on the forward link is carried out using the following MATLAB code:

```
% --- DESIRED USER ---
m_S_fwd = P_BS - 10*K*n_path*log10(des_user_r);
%
% --- CO-CHANNEL USERS ---
% --- Location of desired mobile with respect to
%     co-channel cells ---
aux_01 = ((des_user_position(1) - bs_position(:,1))+ ...
    sqrt(-1)*(des_user_position(2) - bs_position(:,2)));
beta_fwd = angle(aux_01);
d_I_fwd = abs(aux_01);
% --- Computation of antenna gain at co-channel cells
clear gain_fwd
for k = 1:n_co_ch_users
    if (beta_fwd(k) >= ...
        sector_min(sector,sec)) & (beta_fwd(k) < ...
            sector_max(sector,sec))
        gain_fwd(k) = in_beam;
    else
        gain_fwd(k) = out_beam;
    end
end
% --- Computation of mean value and standard deviation ---
m_I_fwd = P_BS - 10*K*n_path*log10(d_I_fwd) + gain_fwd.';
sigma_I_fwd = sigma_int*ones(length(m_I_fwd),1);
[m_I_total_fwd, sigma_I_total_fwd] = ...
    wilkinson(m_I_fwd,sigma_I_fwd,corr_fwd);
```

In a similar manner, the computation of the moments of the signals on the reverse link is carried out using the following MATLAB code:

```
% --- DESIRED USER ---
m_S_rev = P_MS - 10*K*n_path*log10(des_user_r);
%
% --- CO-CHANNEL USERS ---
% --- Location of co-channel users ---
aux_02 = (co_ch_user_position(:,1) + ...
    sqrt(-1)*co_ch_user_position(:,2));
beta_rev = angle(aux_02);
d_I_rev = abs(aux_02);
%
% --- Computation of antenna gain at center cell
clear gain_rev
for k = 1:n_co_ch_users
    if (beta_rev(k) >= ...
        sector_min(sector,sec)) & (beta_rev(k) < ...
            sector_max(sector,sec))
        gain_rev(k) = in_beam;
    else
        gain_rev(k) = out_beam;
    end
end

% --- Computation of mean value and standard deviation ---
m_I_rev = P_MS - 10*K*n_path*log10(d_I_rev) + gain_rev.';
sigma_I_rev = sigma_int*ones(length(m_I_rev),1);
[m_I_total_rev, sigma_I_total_rev] = ...
    wilkinson(m_I_rev,sigma_I_rev,corr_rev);
```

Note that in this example, we use Wilkinson's method for computing the mean and standard deviation of the total interference [10]. The MATLAB code for implementing Wilkinson's method is given in Appendix D.

- *Steps for computing the statistics of* SIR

  1. *Step 1: Computation of mean and standard deviation of* SIR*:* In this approach, we compute the means m_{SIR}^f and m_{SIR}^r, and standard deviations σ_{SIR}^f and σ_{SIR}^r of the *SIR,* expressed in dB, on both the forward and reverse links. These moments will be used to compute the outage and reliability probabilities, as described later. Since the desired signal and the total co-channel interference are normal random variables, when expressed in dB, the signal to interference ratio *SIR*, in dB, is also a normal variable. For the forward link, SIR^f is given by

$$SIR^f = S_{dB}^f - I_{dB}^f \tag{17.41}$$

  with mean and standard deviation, in dB, given by

$$m_{SIR}^f = m_S^f - m_I^f \tag{17.42}$$

and

$$\sigma_{SIR}^{f} = \sqrt{(\sigma_S^f)^2 + (\sigma_I^f)^2} \tag{17.43}$$

Equations (17.41), (17.42), and (17.43) are also used for the reverse link. The results of this step are m_{SIR}^{f}, σ_{SIR}^{f}, m_{SIR}^{r}, and σ_{SIR}^{r}. The MATLAB code for this portion of the simulation follows:

```
m_SIR_fwd(i) = m_S_fwd - m_I_total_fwd;
sigma_SIR_fwd(i) = sqrt(sigma_S^2 + sigma_I_total_fwd^2 -...
    2*corr_fwd*sigma_S*sigma_I_total_fwd);
m_SIR_rev(i) = m_S_rev - m_I_total_rev;
sigma_SIR_rev(i) = sqrt(sigma_S^2 + sigma_I_total_rev^2 -...
    2*corr_rev*sigma_S*sigma_I_total_rev);
```

Note that, at this point in the simulation we start collecting the simulation results. In Step 1 we save the means and standard deviations in arrays controlled by index `i`.

2. *Step 2, Method A: Sampling the* SIR *using the mean and standard deviation of* SIR*:* Since we know the means m_{SIR}^{f} and m_{SIR}^{r}, and standard deviations σ_{SIR}^{f} and σ_{SIR}^{r} of *SIR* on both links from Step 1, we can sample the normal random processes SIR^f and SIR^r. Therefore, each snapshot will be associated with samples of *SIR* on each link, denoted by SIR_{2A}^{f} and SIR_{2A}^{r}. These samples are the result of sampling a normal random process with mean m_{SIR} and standard deviation σ_{SIR}. The results of Step 2, Method A, are values for SIR_{2A}^{f} and SIR_{2A}^{r} as defined by the following MATLAB code.

```
SIR_fwd_2(i) = normrnd(m_SIR_fwd(i),sigma_SIR_fwd(i));
SIR_rev_2(i) = normrnd(m_SIR_rev(i),sigma_SIR_rev(i));
```

3. *Step 2, Method B: Sampling* SIR *using the mean and standard deviation of the desired and individual interference signals:* To this point, we have assumed that the total co-channel interference, modeled as the sum of individual co-channel interference signals, is lognormally distributed, or normally distributed when expressed in dB. In Step 2, Method B, *we do not make any assumption regarding the distribution of the total interference.* A sample of each signal received at the mobile (forward link) and base station (reverse link) at the center cell is determined, assuming that the signals, expressed in dB, are normally distributed having moments S^f, S^r, I_i^f, and I_i^r. The total interference signals on each link is computed as

$$I^f = 10\log_{10}\left(\sum_{i=1}^{6} 10^{I_i^f/10}\right) \tag{17.44}$$

and

$$I^r = 10\log_{10}\left(\sum_{i=1}^{6} 10^{I_i^r/10}\right) \tag{17.45}$$

The samples of SIR on both links, denoted by SIR_{2B}^f and SIR_{2B}^r, are given by

$$SIR_{2B}^f = S^f - I^f \tag{17.46}$$

and

$$SIR_{2B}^r = S^r - I^r \tag{17.47}$$

These are computed using the following MATLAB code:

```
des_sig_spl_fwd = normrnd(m_S_fwd,sigma_S);
int_sig_spl_fwd = normrnd(m_I_fwd,sigma_I_fwd);
tot_int_sig_spl_fwd = 10*log10(sum(10.^...
    (int_sig_spl_fwd/10)));
SIR_spl_fwd_2B(i) = des_sig_spl_fwd - tot_int_sig_spl_fwd;
des_sig_spl_rev = normrnd(m_S_rev,sigma_S);
int_sig_spl_rev = normrnd(m_I_rev,sigma_I_rev);
tot_int_sig_spl_rev = 10*log10(sum(10.^...
    (int_sig_spl_rev/10)));
SIR_spl_rev_2B(i) = des_sig_spl_rev - tot_int_sig_spl_rev;
```

At this point, we have completed the processing of one snapshot. The other $M-1$ snapshots needed to complete the overall simulation are processed generating collections of length M of the following:

- **From Step 1:** mean values m_{SIR}^f and m_{SIR}^r, and standard deviations σ_{SIR}^f and σ_{SIR}^r of SIR on each link
- **From Step 2, Method A:** samples of SIR on both links, SIR_{2A}^f and SIR_{2A}^r
- **From Step 2, Method B:** samples of SIR on both links, SIR_{2B}^f and SIR_{2B}^r

17.3.2 Processing the Simulation Results

By processing the results of the simulation, we can estimate the performance of the overall cellular system, in terms of outage probability and other performance indicators. In this section, we will present examples of simulation results comparing the performance of cellular systems operating under the following six different configurations:

- Cluster size $N = 4$ and omnidirectional base station antennas
- Cluster size $N = 4$ and 120° sectoring

- Cluster size $N = 4$ and 60° sectoring
- Cluster size $N = 7$ and omnidirectional base station antennas
- Cluster size $N = 7$ and 120° sectoring
- Cluster size $N = 7$ and 60° sectoring

The sectorized antennas at the base stations have a front-to-back ratio of 30 dB. The shadowing standard deviation is set to 8 dB, and the path loss exponent is assumed to be $\gamma = 4$. In order to obtain statistically valid results, 1,000 snapshots are simulated.

Outage Probability

The mean and standard deviation of *SIR*, expressed in dB, were computed for each snapshot in the simulation using Step 1. These moments correspond to the *SIR* measured at the mobile and base station at the center cell for a specific snapshot of locations of the co-channel mobiles. Therefore, we can compute the outage probability of the cellular system, at the mobile and base station, for a given specific situation. In a previous section outage probability was defined. It is repeated here for convenience:

$$\begin{aligned} P_{outage}(SIR_0) &= \Pr[SIR < SIR_0] \\ &= \int_0^{SIR_0} \frac{1}{\sqrt{2\pi}\,\sigma_{SIR}} \exp\left[\frac{-(x - m_{SIR})^2}{2\sigma_{SIR}^2}\right] dx \\ &= 1 - Q\left(\frac{SIR_0 - m_{SIR}}{\sigma_{SIR}}\right) \end{aligned} \tag{17.48}$$

where $Q(\cdot)$ is the Gaussian Q-function. Therefore, by using the mean and standard deviation computed at each snapshot, from Step 1 in the simulation, we obtain a *sample* of outage probability. The average outage probability of the cellular system simulated, denoted as $\overline{P}_{outage}(SIR_0)$, is then computed by averaging the samples of outage probabilities $P_{outage}(SIR_0)$, computed for each snapshot. The average outage probability is often referred to as *area-averaged outage probability*, since each element in the averaging process corresponds to a location in the cell area. The average outage probabilities at different thresholds, computed using the results from Step 1 for cluster size $N = 7$ and 120° sectoring, are shown in Figure 17.13. The average outage probabilities for cluster size $N = 7$ and 120° sectoring, computed from the results of Step 2, Methods A and B, are also shown in Figure 17.13.

The average outage probability can also be computed using the results from Step 2, Methods A or B. This provides a collection of samples of *SIR* on both links. Based on these collections, we can estimate the probability density functions of the *area-averaged SIR*, computing first the histograms of the collections. For example, Figure 17.14 shows the histograms of the samples of area-averaged *SIR* obtained from Step 2, Method A. These histrograms approximate the actual pdf

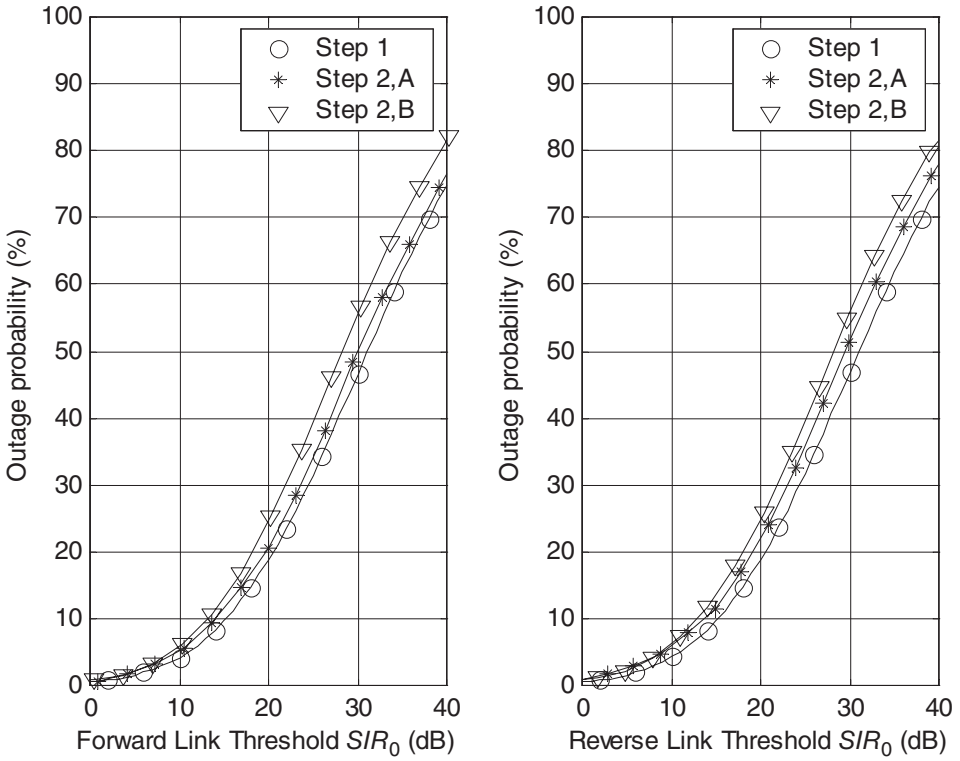


Figure 17.13 Average outage probability on both forward and reverse links using the methods described (cluster size $N = 7$, 120° sectoring, $\sigma = 8dB$, and $\gamma = 4$).

of the area-averaged *SIR*. Using the histogram, we can then estimate the average outage probabilities on both links.

We can now compare the performance of all six configurations simulated. Figure 17.15 shows the average outage probability of each configuration, using Step 1 on the forward link. As expected, sectoring improves the performance of a cellular system, that is, reduces the probability that the *SIR* drops below a given threshold. For example, a system using cluster size $N = 7$ and three sectors per cell performs better, in terms of link quality, than a system using cluster size $N = 7$, but with omnidirectional antennas. The outage probability at $SIR_0 = 18$ dB (which is the threshold usually used in AMPS) in the former system is 15%, versus an outage probability of 35% in the later system.

However, sectoring degrades the capacity of the system, in terms of the maximum carried traffic. As an illustration, Table 17.2 shows the carried traffic per cell for each configuration used, together with the corresponding outage probability at $SIR_0 = 18$ dB, obtained from Figure 17.15. The carried traffic was computed assuming that there are 395 channels available for the whole system and that a

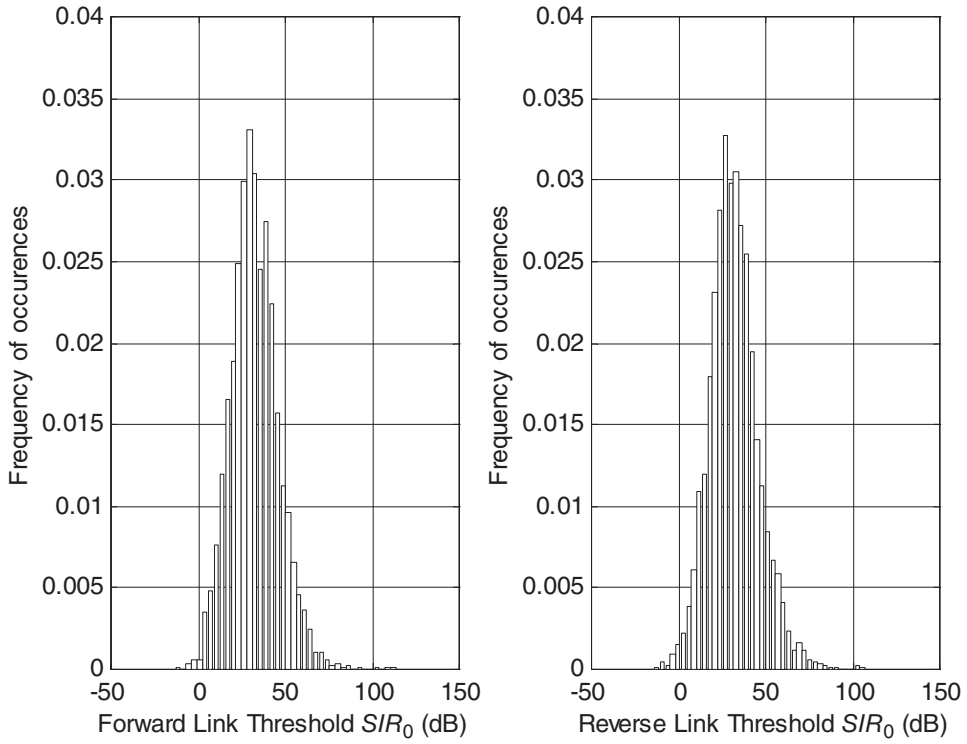


Figure 17.14 Histogram of *SIR* samples obtained using Approach 2 for both forward and reverse links using the Step 2, Method A (cluster size $N = 7$, 120° sectoring, $\sigma = 8dB$, and $\gamma = 4$).

blocking probability of 0.02 is acceptable. We clearly see the tradeoff between capacity and link quality. As the capacity improves, the link quality degrades.

System Performance over the Cell Area

The average outage probability, as discussed and computed in the preceding section, tells us about the performance of the cellular system averaged over the cell area. As a consequence of the averaging operation, the existence of a high outage probability (undesirable) at a given location may be compensated by a low outage probability at another location. Sometimes, in performance analysis, it is desirable to have a measure of the percentage of the cell area where the performance of the system (outage probability) is acceptable or is not acceptable. This measure can be computed from the simulation results.

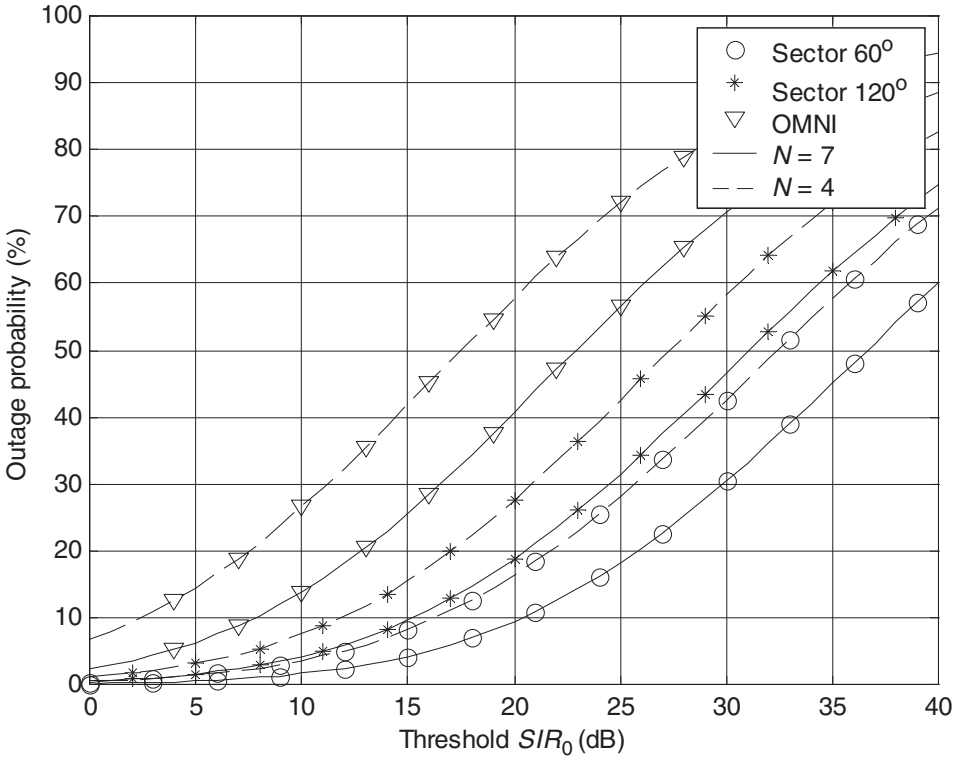


Figure 17.15 Average outage probability on forward link for all configurations.

Table 17.2 Carried Traffic per Cell and Outage Probability

Antenna	**Cluster Size** $N = 4$	**Cluster Size** $N = 7$
Omni	86.0 Erlangs @ $P_{outage} = 0.52$	45.9 Erlangs @ $P_{outage} = 0.35$
3-sector	71.2 Erlangs @ $P_{outage} = 0.22$	34.5 Erlangs @ $P_{outage} = 0.15$
6-sector	59.0 Erlangs @ $P_{outage} = 0.13$	26.1 Erlangs @ $P_{outage} = 0.07$

The *reliability probability* $P_{relia}(SIR_0)$is defined as the probability that *SIR* is *larger* than a given threshold SIR_0. Therefore, at a given location, we have

$$\begin{aligned} P_{relia}(SIR_0) &= \Pr[SIR > SIR_0] \\ &= Q\left(\frac{SIR_0 - m_{SIR}}{\sigma_{SIR}}\right) = 1 - P_{outage}(SIR_0) \end{aligned} \tag{17.49}$$

Let us now assume that the performance of the system is considered acceptable if the reliability probability $P_{relia}(SIR_0)$ at a given threshold SIR_0 is *larger* than a threshold P_{min}. Then, using the computed values of $P_{relia}(SIR_0)$, we can estimate the percentage of the cell area where the system performance is acceptable (i.e., where $P_{relia}(SIR_0) > P_{min}$ holds) by computing

$$P_{area}(SIR_0) = \frac{\text{number of locations where } P_{relia}(SIR_0) > P_{min}}{\text{total number of locations}} \tag{17.50}$$

Note that the computation of $P_{area}(SIR_0)$ is only possible using the results of Step 1 in the simulation, since the mean and standard deviation of *SIR* are required. Figure 17.16 shows the percentage of the cell area where $P_{relia}(SIR_0) > 0.75$ holds on the forward link for all configurations that were simulated.

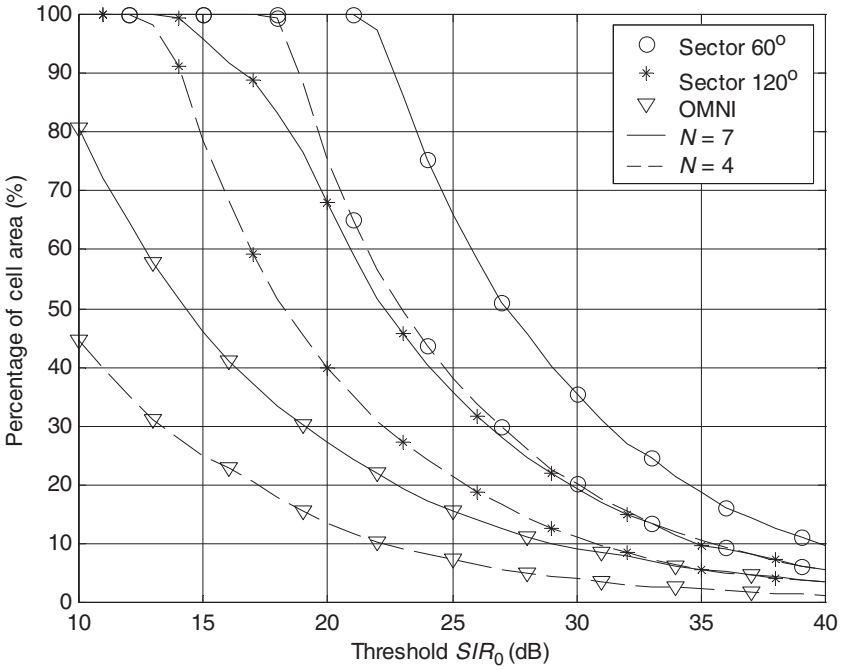


Figure 17.16 Percentage of the cell area where $P_{relia}\,(SIR_0) > 0.75$ holds on the forward link for all configurations simulated ($\sigma = 8$, $\gamma = 4$).

17.4 Summary

This chapter provided a fundamental overview of the key factors involved in simulating the overall performance of a wireless communication system. Several design parameters, such as cellular frequency reuse (cluster size), blocking probability, average signal-to-interference ratio, and antenna beamwidth all impact the overall system performance of a wireless system. Furthermore, these design parameters are all interrelated, and often are used to make tradeoffs in system performance. For example, a smaller cellular cluster size increases the available channel capacity of a system, but increases the interference level of each user. Likewise, sectoring lowers the co-channel interference, yet decreases system capacity. Simulation is needed to explore the impact of various tradeoffs, and the diminishing returns offered by various system design strategies. The level of co-channel interference offered by the undesired users sets the operating noise floor as seen by a particular mobile user at a particular location, and this chapter illustrated how to properly model and simulate the effect of system design parameters across a mobile communication system. The simulation strategy is to create a Monte Carlo analysis of many snapshots of system performance seen by many mobile users operating in random locations, whereby the collection of snapshots are used to build statistics of operating performance over the desired coverage areas. System simulation requires careful modeling of the spatial effects of users, the specified antenna patterns, the assignment of radio channels over space, and the appropriate large-scale interference levels over that space. This chapter provided examples and methodologies to successfully account for these key issues for the design and simulation of wireless communication systems.

17.5 Further Reading

Over the past several years a number of textbooks have been developed that cover the basic concepts of cellular radio systems at varying levels of detail. A sampling of these includes the following:

T. S. Rappaport, *Wireless Communications: Principles and Practice*, 2nd ed., Upper Saddle River, NJ: Prentice Hall, 2002.

G. L. Stuber, *Principles of Mobile Communication*, Boston: Kluwer, 1996.

W. C. Y. Lee, *Mobile Cellular Communication: Analog and Digital Systems*, 2nd ed., New York: McGraw-Hill, 1995.

W. C. Y. Lee, *Mobile Communications Engineering: Theory and Applications*, 2nd ed., New York: McGraw-Hill, 1998.

V. K. Garg and J. E. Wilkes, *Wireless and Personal Communications Systems*, Upper Saddle River, NJ: Prentice Hall, 1996.

In addition to the above listed books, a number of other books are available that treat specific topics within the broader area of cellular and mobile communications. Although this list is growing rapidly, representative examples available at the current time include the following:

H. L. Bertoni, *Radio Propagation for Modern Wireless Systems*, Upper Saddle River, NJ: Prentice Hall, 2000.

V. K. Garg and J. E. Wilkes, *Principles and Applications of GSM*, Upper Saddle River, NJ: Prentice Hall, 1999.

J. C. Liberti, Jr. and T. S. Rappaport, *Smart Antennas for Wireless Communications: IS-95 and Third Generation CDMA Applications*, Upper Saddle River, NJ: Prentice Hall, 1999.

B. Pattan, *Satellite-Based Cellular Communications*, New York: McGraw-Hill, 1998.

B. Pattan, *Robust Modulation Methods & Smart Antennas in Wireless Communications*, Upper Saddle River, NJ: Prentice Hall, 2000.

17.6 References

1. T. S. Rappaport, *Wireless Communications: Principles and Practice*, 2nd ed., Upper Saddle River, NJ: Prentice Hall, 2002.

2. J. C. Liberti Jr. and T. S. Rappaport, *Smart Antennas for Wireless Communications: IS-95 and Third Generation CDMA Applications*, Upper Saddle River, NJ: Prentice Hall, 1999.

3. A. A. M. Saleh and R. A. Valenzuela, "A Statistical Model for Indoor Multipath Propagation," *IEEE Journal on Selected Areas on Communications*, Vol. JSAC-5, No. 2, February 1987, pp. 128–137.

4. T. S. Rappaport et al., "Statistical Channel Impulse Response Models for Factory and Open Plan Building Radio Communication System Design," *IEEE Transactions on Communications*, Vol. COM-39, No. 5, May 1991, pp. 794–806.

5. D. C. Cox, R. Morris, and A. Norris, "800 MHz Attenuation Measured in and Around Suburban Houses," *AT&T Bell Laboratories Technical Journal*, Vol. 673, No. 6, July-August 1984.

6. H. Hashemi, "The Indoor Radio Propagation Channel," *Proceedings of the IEEE*, Vol. 81, No. 7, July 1993, pp. 943–968.

7. A. A. Abu-Dayya and N. C. Beaulieu, "Outage Probabilities in the Presence of Correlated Log-Normal Interferers," *IEEE Transactions on Vehicular Technology*, Vol. 43, No. 1, February 1994, pp. 164–173.

8. S. C. Schwartz and Y. S. Yeh, "On the Distribution Function and Moments of Power Sums with Lognormal Interferers," *Bell System Technical Journal*, Vol. 61, September 1982, pp. 1441–1462.

9. L. F. Fenton, "The Sum of Log-Normal Probability Distributions in Scatter Transmission Systems," *IRE Transactions on Communications Systems*, Vol. CS-8, March 1960, pp. 57–67.

10. P. Cardieri and T. S. Rappaport, "Statistical Analysis of Co-Channel Interference in Wireless Communications Systems," *Wireless Communications and Mobile Computing*, Vol. 1, No. 1, January-March 2001, pp. 111–121.

11. P. Cardieri and T. S. Rappaport, "Application of Narrow-Beam Antennas and Fractional Loading Factor in Cellular Communication Systems," *IEEE Transactions on Vehicular Technology*, Vol. 50, No. 2, March 2001, pp. 1–11.

17.7 Problems

17.1 Determine the smallest allowable physical distance between the centers of co-channel cells if $N = 4$ reuse is used, and each cell has a radius of 2 km.

17.2 What would the proper cellular reuse factor be if the minimum distance in Problem 17.1 was designed for 6 km?

17.3 Using your answer in Problem 17.2, how many channels would be assigned to each cell assuming that the authorized spectrum was 20 MHz in bandwidth, and each duplex channel used 100 kHz forward link and 100 kHz reverse link?

17.4 Assume the signal to interference ratio *SIR* follows a Gaussian distribution with mean 30 dB and standard deviation 10 dB. Compute the outage probability at $SIR_0 = 17$ dB.

17.5 Derive the Erlang B expression given in (17.7).

17.6 Execute the MATLAB script given in Appendix A to validate that it reproduces the Erlang B blocking probability shown in Figure 17.3.

17.7 Assume that the allocated spectrum for a cellular system is 25 MHz in bandwidth, for a single link, and that each channel uses 200 kHz. Also, assume that 50% of the users generate a traffic of 0.25 Erlang/user, while the other 50% generate 0.02 Erlang/user. For cluster sizes $N = 3, 4$, and 7, compute the maximum number of users per cell, at blocking probability 0.02.

17.8 Sectoring offers improvement in interference, but creates capacity loss due to trunking. This is shown in Example 17.4 in the text. Rework Example 17.4 for the following cellular architectures: (a) cluster size $N = 3$, with 3 sectors per cell; (b) cluster size $N = 3$, with 6 sectors per cell.

17.9 Using a Gaussian random number generator to represent independent lognormal shadowing about the distance-dependent mean path loss, estimate the mean and standard deviation of *SIR* at points A, B, and C indicated in Figure 17.17. Also, compute the outage probability at $SIR_0 = 17$ dB at all three

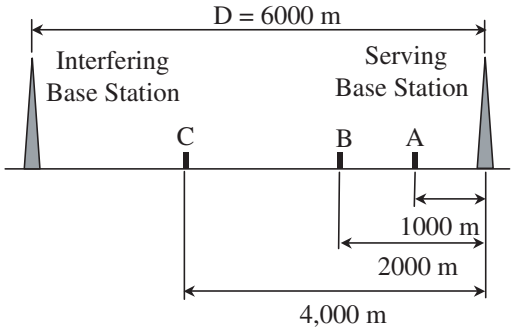


Figure 17.17 Serving and interfering base stations in Problem 17.9.

locations. Assume path loss exponent $\gamma = 3.5$ and shadowing with standard deviation 8 dB. Both base stations transmit at 10 W and are equipped with omnidirectional antennas. Assume that the T-R separation distances are much larger than the base station antenna height. Compare your simulation results with analytical results.

17.10 Using Wilkinson's method compute the mean and standard deviation for the sum of two uncorrelated lognormal signals (see Figure 17.18). Assume the first signal has a mean of −50 dBm, and the second has a mean of −45 dBm. Assume both signals have the same standard deviation of 7 dB.

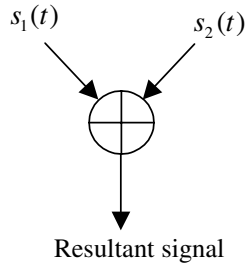


Figure 17.18 Illustration for Problem 17.10.

17.8 Appendix A: Program for Generating the Erlang B Chart

```
% File: c17_erlangb.m
C_1 = [1:10];
A_1 = linspace(0.1,10,50);
C_2 = [12:2:20];
A_2 = linspace(3,20,50);
C_3 = [30:10:100];
A_3 = linspace(13,120,50);
for i = 1:length(C_1)
    for j = 1:length(A_1)
        p_1(j,i) = erlang_b(A_1(j),C_1(i));
    end
end
for i = 1:length(C_2)
    for j = 1:length(A_2)
        p_2(j,i) = erlang_b(A_2(j),C_2(i));
    end
end
for i = 1:length(C_3)
    for j = 1:length(A_3)
        p_3(j,i) = erlang_b(A_3(j),C_3(i));
    end
end
%
% The following code plots the results.
x1 = [.1 .1 .2 .2 .3 .3 .4 .4 .5 .5 .6 .6 .7 .7 .8 .8 .9 .9 ];
y1 = 10.^[-4 1 1 -4 -4 1 1 -4 -4 1 1 -4 -4 1 1 -4 -4 -1];
y2 = [.1 .1 .2 .2 .3 .3 .4 .4 .5 .5 .6 .6 .7 .7 .8 .8 .9 .9];
x2 = 10.^[-1 3 3 -1 -1 3 3 -1 -1 3 3 -1 -1 3 3 -1 -1 3];
loglog(A_1,p_1,‘k-’,A_2,p_2,‘k-’,A_3,p_3,‘k-’,...
 x1,y1,‘k:’,10*x1,y1,‘k:’,...
 100*x1,y1,‘k:’,1000*x1,y1,‘k:’,...
 x2,y2,‘k:’,x2,0.1*y2,‘k:’,x2,0.01*y2,‘k:’);
xlabel(‘Offered traffic (Erlangs)’)
ylabel(‘Blocking probability’)
axis([0.1 120 0.001 0.1])
text(.115, .115,‘C=1’)
text(.6, .115,‘C=2’)
text(1.18, .115,‘3’)
text(2, .115,‘4’)
text(2.8, .115,‘5’)
text(7, .115,‘10’)
text(9, .115,‘12’)
```

```
text(17, .115,'20')
text(27, .115,'30')
text(45, .115,'50')
text(100, .115,'100')
% End of script file.
```

The preceding MATLAB program makes the function call `erlang_b(A,c)`. The code for `erlang_b(A,c)` is listed in Section 17.2.2.

17.9 Appendix B: Initialization Code for Simulation

The code that follows is the initialization code for the cellular simulation. The overall simulation can be developed by adding the code segments given in Section 17.3 to this initialization code.

```
% ===== Pre-defined simulation parameters ==========================
r_cell = 1000;          % cell radius (in meters)
n_co_ch_users = 6;      % number of co-channel users
P_BS = 0;               % BS transmitter power (in dBW)
P_MS = 0;               % MS transmitter power (in dBW)
corr_fwd = 0.0;         % correlation coefficient - forward link
corr_rev = 0.0;         % correlation coefficient - reverse link
K = 1;                  % constant in the link equation
in_beam = 0;            % maximum gain of sectorized antennas (in dB)
%
% --- Limits (angles) of each sector ---
sector_min = zeros(6,3);
sector_max = zeros(6,3);
sector_min(:,1) = (pi/3)*[-3:2]';
sector_min([1:3],2) = pi/3*[-3 -1 1]';
sector_min(1,3) = -pi;
sector_max(:,1) = sector_min(:,1) + pi/3;
sector_max([1:3],2) = sector_min([1:3],2) + 2*pi/3;
sector_max(1,3) = pi;
%
% --- Center of each sector ----
phi_center = zeros(6,3);
phi_center(:,1) = (pi/3)*[-3:2]';
phi_center([1:3],2) = (pi/3)*[-3 -1 1]';
%
% --- Beamwidth of each sector ---
phi_BW = [1 2 6]*pi/3;
%
% --- Number of sectors -----
num_sectors = [6 3 1];
%
% ===== User Inputs ========================================
num_snapshots = input('Number of snapshots = ');
cluster_size = input('Cluster size (3, 4 or 7) = ');
n_path = input('Path loss exponent = ');
sigma_int = input('Shadowing std deviation - interference (dB) = ');
sigma_S = input('Shadowing std deviation - desired signal (dB) = ');
sec = input('Sectorization...
    (1=>60 degree, 2=>120 degree, 3=>omni). enter: ');
ftb = input('Front-to-back ratio of the BS antennas (dB) = ');
```

```
out_beam = in_beam - ftb;
% End of script file.
```

17.10 Appendix C: Modeling Co-Channel Interference

Consider the situation in which N interference signals arrive at a receiver from N co-channel transmitters. Assuming that the effects of small-scale fading are averaged out, the local mean power level I_i of the i^{th} signal undergoes lognormal variation. Using decibel units, the local mean power level can be modeled as [1, 2]

$$X_i = 10\log_{10} I_i = m_{X_i} + \chi_i \qquad \text{dBW} \tag{17.51}$$

where m_{X_i} is the area mean power (or, alternatively, average large-scale propagation path loss) and χ_i is a zero-mean normally distributed RV in dB with standard deviation σ_{X_i}, also in dB, due to the shadowing caused by large obstacles [1, 2]. The area mean m_{X_i} is usually modeled as a function of the T-R separation distance d_i, the path loss exponent n, the transmitted power $P_{T,i}$ in dBm, and the transmitter and receiver antenna gains $G_{T,i}$ and $G_{R,i}$, both in dB. This gives

$$m_{X_i} = P_{T,i} + G_{T,i} + G_{R,i} - 10n\log_{10} d_i \qquad \text{dBW} \tag{17.52}$$

Under the reasonable assumption that the individual signals I_i add incoherently, the total interference signal is modeled as the sum of N lognormally distributed signals

$$I = \sum_{i=1}^{N} I_i \tag{17.53}$$

The distribution of I is an important consideration for modeling the impact of multiple interferers, and thus dictates the resulting total interference at a receiver. Multiple transmitters, displaced throughout a geographic area, may each provide different levels of interference, based on the particular physical distances to the receiver. Thus, the effects of shadowing upon each interfering signal and the sum of the interference at the receiver from all interfering signals must be considered to determine an accurate interference level at any particular location. Note that, depending on the strength of the particular individual interferers, the resulting composite interference level may vary widely, and if each interferer produces a random signal level, the composite signal level will also be random. It is well accepted that the distribution of I can be approximated by another lognormal distribution [7–10], or, equivalently, that

$$X = 10\log_{10} I \tag{17.54}$$

follows a normal distribution. Assuming that the sum I is lognormally distributed, Wilkinson's method [9] and and Schwartz and Yeh's method [8] allow for the computation of the mean m_X and standard deviation σ_X of X.

In the derivation of these two methods, it is convenient to use the natural logarithm instead of the base 10 logarithm to define the normal RV that corresponds to a lognormal RV. Thus, we define the normal RV Y_i as

$$Y_i = \ln I_i \tag{17.55}$$

with mean value m_{Y_i} and standard deviation σ_{Y_i} given, respectively, by

$$m_{Y_i} = \lambda\, m_{X_i} \qquad \text{and} \qquad \sigma_{Y_i} = \lambda\, \sigma_{X_i} \tag{17.56}$$

where $\lambda = \ln(10)/10$. Note that $Y_i = \lambda X_i$.

Using (17.53) and (17.55) and recalling that we are approximating the distribution of I by a lognormal distribution, we have

$$I = e^{Y_1} + e^{Y_2} + \cdots + e^{Y_N} \approx e^Z = 10^{X/10} \tag{17.57}$$

where Z (in logarithmic units) and X (in dB) are normally distributed, and $Z = \lambda X$. Either Wilkinson's method or Schwartz and Yeh's method can then be used to compute the mean value and standard deviation of Z (m_Z and σ_Z) or X (m_X and σ_X) from the mean values and standard deviations of the summands Y_i, as shown subsequently.

For generality, it is useful to assume that the individual signals I_i may be correlated to each other. This correlation may be due to the fact that there is a common physical obstruction that induces shadowing loss for particular propagation paths, such as vegetation or building structures. Therefore, even signals coming from different directions may be attenuated by the same obstacles, leading to correlation among the received signals. To consider the correlated interference signals case, let us define the correlation coefficient r_{Y_i,Y_j} of Y_i and Y_j by

$$r_{Y_i,Y_j} = \frac{E\{(Y_i - m_{Y_i})(Y_j - m_{Y_j})\}}{\sigma_{Y_i}\, \sigma_{Y_j}} \tag{17.58}$$

Since Y_i is simply a scaled version of X_i, it follows that r_{Y_i,Y_j} is the correlation coefficient of X_i and X_j.

17.10.1 Wilkinson's Method

According to Wilkinson's method, the mean value and standard deviation of Z in (17.57) are found by matching the first and second moments of I with those of $I_1 + I_2 + \cdots + I_N$. For the first moment, we have

$$E\{e^Z\} = E\{e^{Y_1} + e^{Y_2} + \cdots + e^{Y_N}\} \tag{17.59}$$

The moments in (17.59) are evaluated by observing that, for a normal RV u with mean value m_u and variance σ_u^2, and any integer l, it turns out that [14]

$$E\{e^{lu}\} = \exp(lm_u + \frac{1}{2}l^2\sigma_u^2) \tag{17.60}$$

where l denotes the moment order for the normal random variable u. Therefore, to evaluate the first moment of $\exp(Z)$ where the random variable Z is assumed to be Gaussian, we see that

$$E\{e^Z\} = \exp(m_Z + \sigma_Z^2/2) \tag{17.61}$$

and

$$E\{e^{Y_1} + e^{Y_2} + \cdots + e^{Y_N}\} = \sum_{i=1}^{N} \exp(m_{Y_i} + \sigma_{Y_i}^2/2) \tag{17.62}$$

Using (17.61) and (17.62) in (17.59), we have

$$u_1 = \exp(m_Z + \sigma_Z^2/2) = \sum_{i=1}^{N} \exp(m_{Y_i} + \sigma_{Y_i}^2/2) \tag{17.63}$$

The summation in (17.63) is a function of the mean values m_{Y_i} and standard deviations σ_{Y_i} of the summands Y_i, which are assumed to be known either through measurement or through the use of a propagation model.

Matching the second moments of I and $I_1 + I_2 + \cdots + I_N$, we have

$$E\{e^{2Z}\} = E\{(e^{Y_1} + e^{Y_2} + \cdots + e^{Y_N})^2\} \tag{17.64}$$

Using again the property (17.60) in both sides of (17.64), we obtain

$$u_2 = \exp(2m_Z + 2\sigma_Z^2) \tag{17.65}$$

which is

$$u_2 = \sum_{i=1}^{N} \exp(2m_{Y_i} + 2\sigma_{Y_i}^2) + 2\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} \exp(m_{Y_i} + m_{Y_j}) \cdot \exp\left[\frac{1}{2}(\sigma_{Y_i}^2 + \sigma_{Y_j}^2 + 2r_{Y_i,Y_j}\sigma_{Y_i}\sigma_{Y_j})\right] \tag{17.66}$$

Equation (17.66) can be evaluated using the mean values m_{Y_i}, standard deviations σ_{Y_i} and correlation coefficients r_{Y_i,Y_j}.

Equations (17.63) and (17.66) form a system of equations with unknowns m_Z and σ_Z. By solving this system of equations, and using $Z = \lambda X$, we finally obtain

$$m_X = (1/\lambda)(2\ln u_1 - \frac{1}{2}\ln u_2) \tag{17.67}$$

$$\sigma_X = (1/\lambda)\sqrt{\ln u_2 - 2\ln u_1} \tag{17.68}$$

Therefore, Wilkinson's method consists of computing the terms u_1 and u_2 using (17.63) and (17.66), applying the means, standard deviations, and correlation coefficients of the summands, and solving the system of equations defined by (17.67) and (17.68). An important feature of Wilkinson's method is that the assumption of $\sum_i I_i$ being lognormally distributed is actually used in the computation of m_X and σ_X. The MATLAB program for Wilkinson's method is given in Appendix D.

17.10.2 Schwartz and Yeh's Method

Schwartz and Yeh proposed a method based on the *exact* computation of the mean value m_X and standard deviation σ_X of the sum of $N = 2$ lognormal RVs. For $N > 2$, a recursive approach is used, approximating the sum of two lognormal RVs by another lognormal RV, and computing the mean and standard deviation of the sum. The details of this method are not included here. The interested student is referred to the literature [8, 10].

17.11 Appendix D: MATLAB Code for Wilkinson's Method

```
% File: wilkinson.m
function [m_out,std_out] = wilkinson(m_x,std_x,r)
% this function computes the mean and standard deviation
% of the sum of two lognormal RV's
% Input and ouput values are in dB.
lambda = 0.1*log(10);
m_x_cmp = m_x;
v_x_cmp = std_x.^2;
m_y = lambda*m_x_cmp;
v_y = (lambda^2)*v_x_cmp;
u_1 = 0;
for i = 1:length(m_y)
    u_1 = u_1 + exp(m_y(i) + v_y(i)/2);
end
a = 0;
for i = 1:length(m_y)
    a = a + exp(2*m_y(i) + 2*v_y(i));
end
b = 0;
for i = 1:length(m_y)-1
    for j = i+1:length(m_y)
        b = b + exp( m_y(i) + m_y(j) )*...
        exp(0.5*(v_y(i) + v_y(j) + ...
        2*r*sqrt(v_y(i))*sqrt(v_y(j))));
    end
end
u_2 = a + 2*b;
% mean and variance or the variable Z, which is normal
% in natural units
m_z = 2*log(u_1) - 0.5*log(u_2);
std_z = sqrt(log(u_2) - 2*log(u_1));
% mean and variance of the variable X, which is normal in dB.
g = 10*log10(exp(1));
m_out = g*m_z;
std_out = g*std_z;
% End of function file.
```

Chapter 18

TWO EXAMPLE SIMULATIONS

We conclude our study of simulation techniques and methodology by presenting two very different examples. The first example is a simulation of a CDMA system operating in a multipath/fading environment. Thermal noise and interference are also included in the simulation model. A number of performance results can be generated by the simulation model. These include the bit error rate (BER) as a function of the thermal noise level (P_E versus E_b/N_0), BER as a function of the spreading factor (spread-spectrum processing gain), and BER as a function of the number of interfering signals. Even though the system presented here is more complex than those presented in previous chapters, a number of simplifications are made so that the MATLAB code can be executed with a reasonable runtime. If desired, the simplifications can be removed and the resulting simulations can be coded in a compiled language for faster execution. Thus, this example can serve as a template for more advanced investigations. Monte Carlo simulation is used for this first example.

The second example is a multichannel (FDM) satellite communications system. A nonlinear power amplifier is assumed, and the purpose of the simulation is to evaluate the effect of the nonlinearity on the performance of the system. This second simulation makes use of the semianalytic simulation technique. Since semianalytic

simulations typically execute much faster than Monte Carlo simulations, simplifying the simulation model is not as important as in the CDMA example.

18.1 A Code-Division Multiple Access System

The code-division multiple access (CDMA) simulator emulates a simple CDMA communications link. CDMA systems are widely deployed, and also serve as building blocks for more advanced systems. They are described in detail in the literature [1, 2]. A block diagram of the simulation model is illustrated in Figure 18.1. The simulation includes the effects of Additive White Gaussian Noise (AWGN), specular multipath, and Multiple Access Interference (MAI). Using this simulation, one can estimate the BER as a function of the bit energy to noise spectral density ratio (E_b/N_0), the number of interferers (NoI), and the spreading factor (SF). One may apply any specular multipath channel provided that the channel has no more than five multipath components.

18.1.1 The System

Unlike FDMA (frequency division multiple access) or TDMA (time division multiple access), which separate wireless users by assigning different frequency bands or time slots, respectively, CDMA separates users through the assignment of different signature (code) sequences. Accordingly, a particular CDMA system will have at its disposal a signature sequence set consisting of a collection of pseudo-random sequences. This collection of sequences must have low cross-correlation properties; that is, the inner product of any two sequences in this collection must be small. This is necessary so that the receiver can accept the signal from a desired wireless user while rejecting the signals from the other (undesired) co-channel wireless users. In addition, each sequence in this collection must have correlation properties that allow the multipath effects commonly found in wireless channels to be exploited through the use of RAKE receivers or through space-time reception techniques.

The CDMA system assigns each user a specific signature sequence from the signature sequence set. This signature sequence is used to generate a spreading waveform, whose symbol rate (hereafter called the chip rate) is much higher than the symbol rate of the information-bearing signal (hereafter called the symbol rate). The ratio of the chip rate to the symbol rate is called the spreading factor or processing gain. Spreading factors typically range from values as low as 8 to values as high as 512.

CDMA systems typically use direct sequence spread spectrum techniques. The information-bearing waveform is modulated (multiplied) by the user's spreading waveform to produce a spread spectrum signal, which is then radiated into the channel. At the channel output, the receiver (we assume a simple correlation receiver) correlates the incoming signal with the user's spreading (signature) waveform. This accomplishes two tasks. It allows the user's information-bearing signal (the desired signal) to pass through the correlation receiver, and it rejects the information-bearing signals (interference) of all other users.

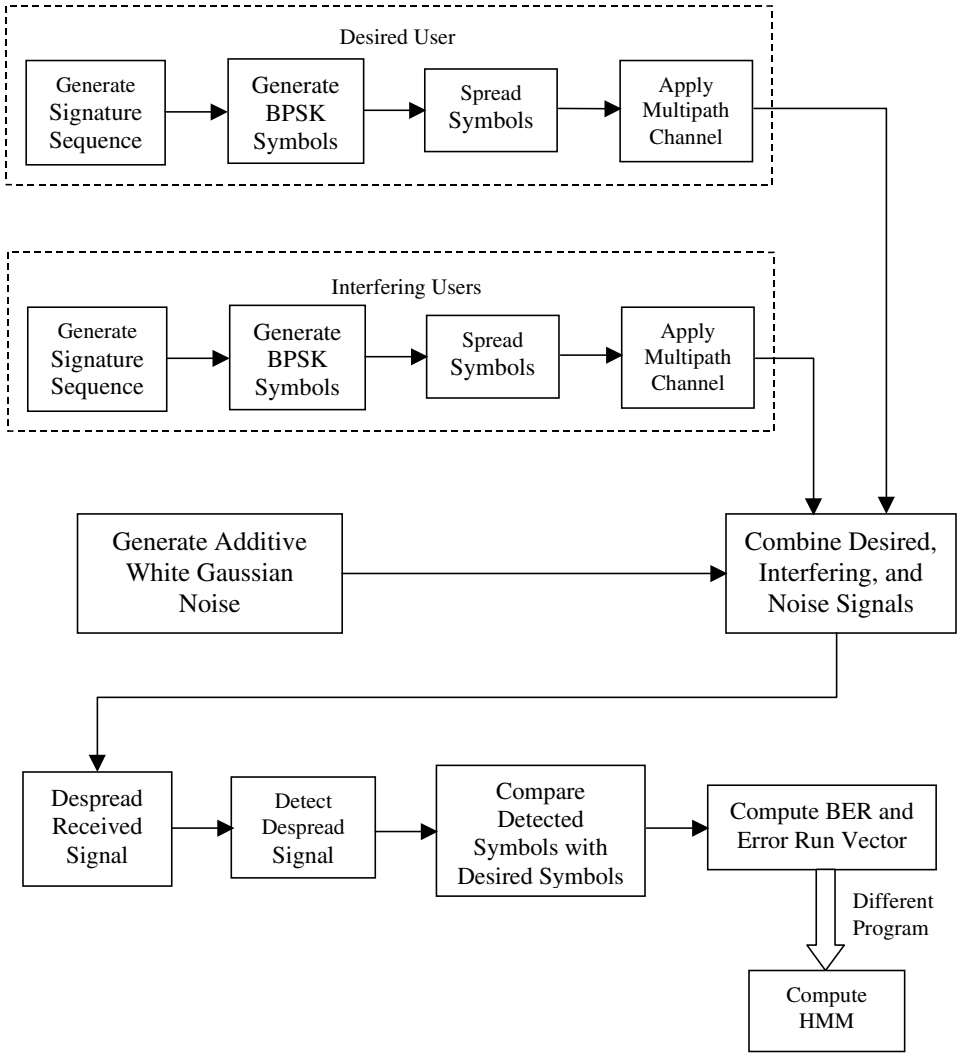


Figure 18.1 Methodology for CDMA example.

In order to create a computationally efficient simulation of a CDMA communications link, the following simplifications/assumptions are incorporated:

1. Perfect power control is assumed, and each signal is assumed to arrive at the receiver with equal average power.

2. The signature/spreading sequence for the desired signal is a PN-sequence, as discussed in Chapter 7. Accordingly, the allowable spreading factors must be equal to $2^R - 1$, where R is a positive integer.

3. The signature/spreading sequence used for each MAI signal is the same PN-sequence. However, the PN-sequence is shifted in a circular fashion so as to achieve the desired correlation properties. It is important to note that, even though these desired correlation properties are guaranteed in an AWGN channel, it is possible to lose these correlation properties in a specular multipath channel, as modeled here, since the desired and interfering signals are separated in time by an integer number of chips. There is a small probability that these delays may equal the multipath delays, resulting in pessimistic results when simultaneously simulating interference and multipath.

4. Binary phase shift keying modulation is assumed and pulse shaping is neglected.

5. The multipath channel exhibits Rayleigh fading on each multipath component with the exception of the line-of-sight component, which is not faded. The interfering signals do not exhibit fading.

6. The delays of each multipath component are limited to integer multiples of the chip duration.

7. The desired signal and all MAI signals are chip synchronized at the receiver.

8. The receiver is a simple correlation receiver. No rake receiver is used.

Neglecting the detailed modeling of the pulse shape and the corresponding matched filters at the receiver has a dramatic effect on the speed at which the simulation executes, since the underlying waveforms in the system can be executed at one sample per chip. Reasonable results will be obtained if zero-ISI pulse shapes (e.g., root raised cosine pulse shape) are used in the system.[1]

In a practical application, channel measurements would be used to determine a power-delay profile, and the simulation would be executed using a suitable approximation to the measured power-delay profile. This ensures that the simulation model accurately characterizes the environment in which the wireless system is deployed. In this simulation, however, it is important to note that a delay profile is explicitly specified and the power profile is computed so that a given Ricean K-factor is satisfied. The motivation for doing this lies in the fact that students of communication theory have some understanding of the relationship between system performance and the Ricean K-factor. For example, for K large, system performance is essentially equivalent to performance in an additive Gaussian noise environment. For $K \cong 0$, the environment is Rayleigh fading and can be flat fading or frequency selective fading depending on the number of multipath components and the delays associated with those components. Analytical solutions for the BER exist for both of these limiting situations [3], and these solutions serve as sanity checks for the simulation results.

[1] At the time of this writing, a detailed simulation of the wideband CDMA (WCDMA) uplink and downlink standards, which does not contain these resrtictions, can be downloaded from the Virginia Tech MPRG web site. The URL is http://www.mprg.org/research/wcdma/wcdmasim.php.

For this simulation, the multipath delay profile is simply a vector of integers, where each integer represents the delay of each multipath component in terms of the chip periods, which is the symbol period divided by the spreading factor. The simulation assumes that the first element in this vector corresponds to the line-of-sight (LOS) component and that the remaining elements correspond to the scattered (multipath) components. The length of this vector is determined by the total number of received components (the LOS component plus the scattered components).

After the delay profile and the Ricean K-factor is specified, a power profile is calculated so that the specified K-factor is realized. The resulting power profile, along with the delay profile, can be displayed by the postprocessor if desired. One may also modify the simulation code to support the input of both a delay and power profile. Such a modification is a straightforward endeavor.

We now examine the manner in which the power profile is determined so that the specified Ricean K-factor is satisfied. The average amplitude of each scattered component is produced via a random number generator, having outputs uniformly distributed between 0 and 1. Let

$$A_{scat} = \begin{bmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_{M-1} \end{bmatrix} \tag{18.1}$$

denote the average amplitude of each scattered component. Note that M is the total number of multipath components ($M-1$ scattered signals plus the line-of-sight component). One can then compute the average scattered energy, which is given by

$$E_{scat} = \sum_{i=1}^{M-1} \alpha_i^2 \tag{18.2}$$

The average energy in the LOS component is computed as

$$E_{LOS} = KE_{scat} \tag{18.3}$$

where K is the Ricean K-factor. The average amplitude of the LOS component is then

$$\alpha_{LOS} = \sqrt{E_{LOS}} \tag{18.4}$$

The average amplitude profile is obtained by adding α_{LOS} to the vector for the scattered components A_{scat}. The resulting vector is then normalized so that the strongest multipath component has a value of 1.0. Thus:

$$\begin{aligned} A &= \frac{\begin{bmatrix} \alpha_{LOS} & \alpha_1 & \alpha_2 & \cdots & \alpha_{M-1} \end{bmatrix}}{\max\left\{\begin{bmatrix} \alpha_{LOS} & \alpha_1 & \alpha_2 & \cdots & \alpha_{M-1} \end{bmatrix}\right\}} \\ &= \begin{bmatrix} a_{LOS} & a_1 & a_2 & \cdots & a_{M-1} \end{bmatrix} \end{aligned} \tag{18.5}$$

The power profile is then obtained by squaring the elements in A. Therefore:

$$P = \begin{bmatrix} a_{LOS}^2 & a_1^2 & a_2^2 & \cdots & a_{M-1}^2 \end{bmatrix} \tag{18.6}$$

Clearly, for cases where $K > 1$, the LOS component will be the strongest received component, and $a_{LOS} = 1.0$. However, for the case in which $K \leq 1$, the situation is not as clear. In such a case, the average scattered energy exceeds the average energy in the LOS component. However, the average scattered energy, E_{scat}, may be spread across several multipath components. In such situations, the LOS component may, or may not, be the strongest multipath component. Displaying the power profile will, of course, allow the user to identify the strongest received component.

Once the delay profile and the amplitude (power) profile are specified, the channel model is determined. The channel model is illustrated in Figure 18.2. The Rayleigh random process generator is illustrated in Figure 18.3 and is implemented as discussed in Chapter 7. Two independent Gaussian random variables, $x_d[n]$ and $x_q[n]$, are generated and filtered to produce $y_d[n]$ and $y_q[n]$. The filter type in the Rayleigh process generator is arbitrary and was chosen to be a fourth-order Chebyshev filter with 0.5 dB passband ripple and a bandwidth equal to the doppler frequency. The magnitude, $\sqrt{y_d^2[n] + y_q^2[n]}$, is a Rayleigh random variable.

The simulation also requires the specification of the E_b/N_0 parameter. This parameter determines the ratio of the average bit energy in the strongest multipath component (the LOS component for $K > 1$) to the thermal noise energy, that is, E_b/N_0. Note that the average power of the strongest multipath component will be normalized to unity. Accordingly, the simulation scales the energy of the white Gaussian (thermal) noise so that average E_b/N_0 in the strongest multipath component equals the E_b/N_0 parameter

18.1.2 The Simulation Program

The example CDMA simulation has a block serial structure in which blocks of 1,000 symbols are serially processed. As discussed in Chapter 10, the block serial structure is used for computational efficiency. The simulation is completely contained in the function `c18_cdmasim`. It is invoked by the MATLAB call

$$[\texttt{BER}, \texttt{Errors}] = \texttt{c18\_cdmasim}(\texttt{N}, \texttt{SF}, \texttt{EbNo}, \texttt{NoI}, \texttt{MPathDelay}, \texttt{KfactordB}) \quad (18.7)$$

The MATLAB code for the function `c18_cdmasim` is given in Appendix A. The preceding line of MATLAB code fully defines the main simulation program (the simulation engine). To this is added a preprocessor and a postprocessor. For our application, these are combined into a single program. An example preprocessor/postprocessor is also given in Appendix A. These should be viewed only as examples, and the student is encouraged to experiment with these and customize them as desired.

The input parameters are defined in Table 18.1. It is important to observe the restrictions. Note that these restrictions apply to each call to the simulator. One may certainly develop a preprocessor in which a given parameter is iterated over a range of values.

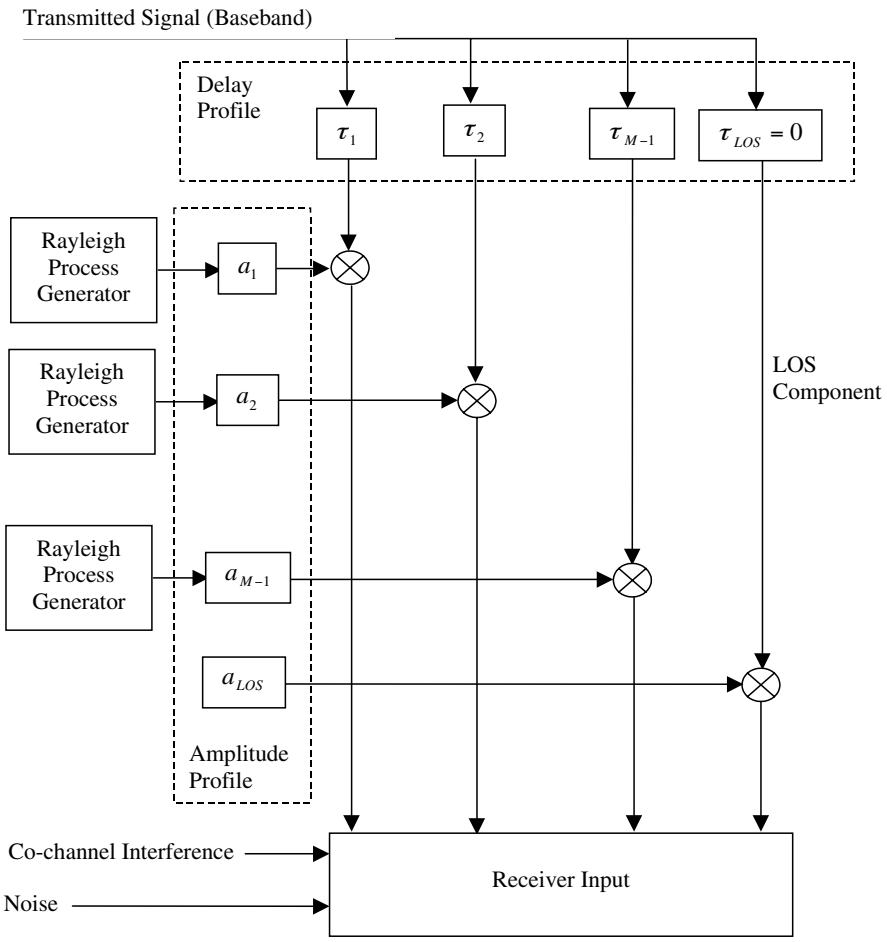


Figure 18.2 Channel model for CDMA example.

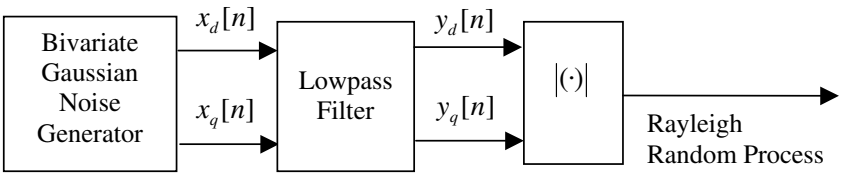


Figure 18.3 Rayleigh random process generator.

Table 18.1 CDMA System Input Parameters

Parameter	Type	Description	Restrictions
N	Scalar	Number of symbols simulated	None
SF	Scalar	Spreading factor	$SF = 2^n$, $n \leq 12$
E_b/N_0	Scalar	Ratio of bit energy to noise PSD	None
NoI	Scalar	Number of Interferers	$0 \leq NoI \leq SF-1$
MPathDelay	Vector	Multipath delay profile	Vector of monotonically increasing non-negative integers
KfactordB	Scalar	Ricean K-factor in dB	None

18.1.3 Example Simulations

Baseline Validation

In order to ensure that the simulation is properly calibrated, the CDMA simulator was first executed with a Ricean K-factor of 100 dB. This K-factor is sufficiently large to ensure that, even though a very small level of multipath is present, the scattered power is sufficiently attenuated to make the approximation that all of the received power is in the LOS component. Thus, the only degrading effect is Gaussian noise and, as a result, the error probability for the system is

$$P_E = Q\left(\sqrt{\frac{2E_b}{N_0}}\right) \tag{18.8}$$

The simulation was performed with $N = 200{,}000$ symbols processed for each value of E_b/N_0. The value of E_b/N_0 was varied from 0 to 8 dB in 1-dB steps. The MATLAB dialog for the simulation run is

```
>> c18_cdmacal
Enter number of symbols to be processed > 200000
Enter Eb/No vector > 0:8
```

The results are given in Table 18.2 and in Figure 18.4. The number of errors occurring for each value of E_b/N_0 are given in Table 18.2. This data is particularly important in a calibration run to ensure that a sufficient number of errors

Table 18.2 Error Counts for Calibration Run

E_b/N_0 (dB)	0	1	2	3	4	5	6	7	8
Errors	15773	11347	7583	4516	2416	1169	475	159	53

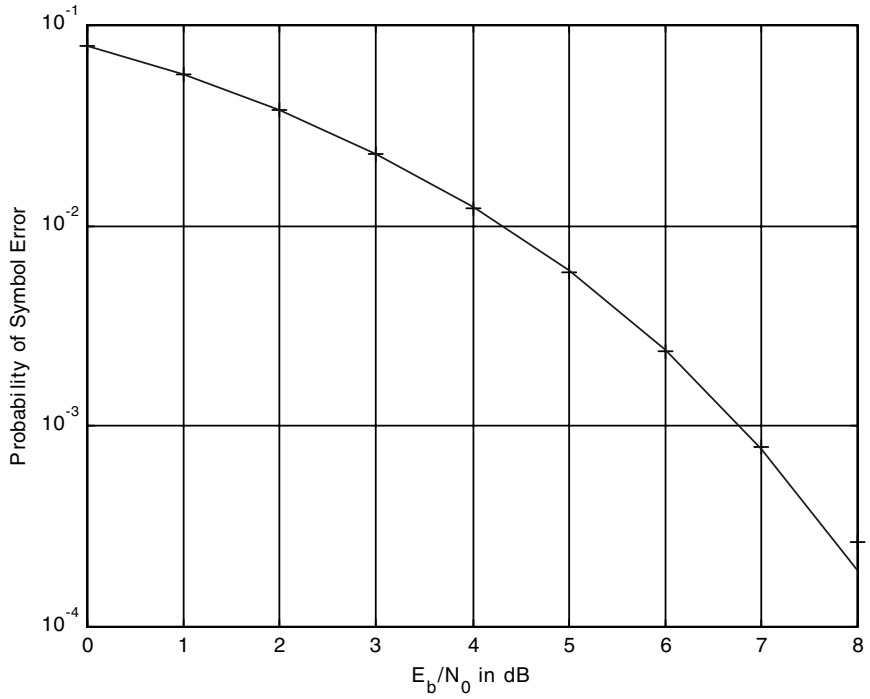


Figure 18.4 Result of calibration run.

are obtained at each value of E_b/N_0 to provide sufficient statistical reliability. The probability of error P_E as a function of E_b/N_0 is given in Figure 18.4. The solid line gives the theoretical result described by (18.8) and the simulation results are indicated by the plus signs at integer values of E_b/N_0. It can be seen that accurate results are achieved except, perhaps, for $E_b/N_0 = 8$ dB. Table 18.2 indicates that the error probability for $E_b/N_0 = 8$ dB is based on 53 errors, which is small compared to the number of errors occurring at other values of E_b/N_0. The MATLAB preprocessor code is given in Appendix B (`c18_cdmacal.m`).

Performance as a Function of E_b/N_0 and the Ricean K-factor

This example generates the BER as a function of both E_b/N_0 and the Ricean K-factor. Thus, two of the variables given in Table 18.1 must be iterated. Specifically, E_b/N_0 is stepped from 0 dB to 10 dB in 1-dB steps. Three values of the Ricean K-factor are used; -20 dB, 0 dB, and 20 dB. The processor/postprocessor (`c18_cdmaex1.m`) used to generate these results is given in Appendix B. The following dialog is used:

```
>> c18_cdmaex1
Enter number of symbols to be processed  > 100000
Enter Eb/No vector > 0:10
Enter KfatordB vector > [-20 0 20]
```

Executing the program yields the results illustrated in Figure 18.5. The top curve in Figure 18.5 corresponds to a K-factor of -20 dB ($K = 0.01$), which produces a result closely approximating Rayleigh fading. The bottom curve in Figure 18.5 corresponds to a K-factor of 20 dB ($K = 100$), which produces a result closely approximating performance in a Gaussian noise environment. Results for both large K and $K \cong 0$ can be derived analytically [3] and used to verify the simulation. The middle curve corresponds to a K-factor of 0 dB ($K = 1$) and is usually derived using simulation. Note that the middle curve ($K = 1$) is less smooth than the other curves. This is due to the time-varying nature of the channel, which is most pronounced for intermediate values of K, where the power in the LOS component and the total scattered power are approximately equal. Very long simulations are required to smooth out these temporal variations.

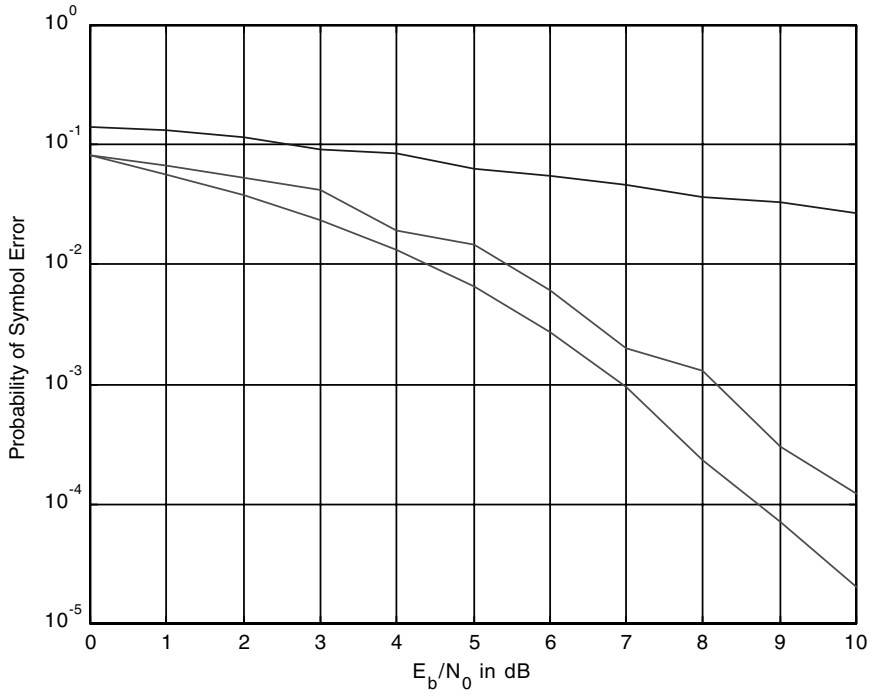


Figure 18.5 Error probability as a function of E_b/N_0 with the Ricean K-factor as a parameter. ($K = 100$ (bottom curve), $K = 1$ (middle curve, and $K = 0.01$ (top curve))

There are many other possibilities for making interesting investigations. The interested student should develop a preprocessor that allows one to study the effect of the spreading factor for a given delay profile and at a given E_b/N_0. This should be done at several different values of E_b/N_0 and for several different delay profiles. A similar study can be conducted by varying the number of interferers.

It is clear from the preceding paragraph that a number of meaningful simulations can be generated using this simplified CDMA simulation. As the simplifications are removed, more parameters are introduced into the simulation model. Running parametric studies on systems having a large number of parameters generates a huge amount of data, all of which must be analyzed before a "best" design can be selected. As a result, the simulation study must be organized carefully so that the data created by the various simulations can be "mined" in a way that supports the design by identifying the most critical parameters and acceptable ranges for these parameters. The process of organizing multiple simulations and mining the data created by these simulations is not covered in this book. This is, however, an important topic and should receive attention prior to beginning a detailed simulation study in which many different simulations are conducted.

18.1.4 Development of Markov Models

Figure 18.1 shows the generation of a hidden Markov model (HMM) for the CDMA system. This is easily accomplished using the tools discussed in Chapter 15. In order to determine and evaluate the HMM, three different MATLAB programs are developed. (The semi-Markov model is used in order to reduce the computational burden.) The first of these programs executes the CDMA simulation and determines the semi-Markov model. The second program generates an error vector based on the state transition matrix of the semi-Markov model. In addition, the bit error rates based on the CDMA system, the bit error rate predicted by the semi-Markov model, and the bit error rate based on an error sequence generated by the semi-Markov model are compared. The third program compares the error vectors created by the original simulation and by the semi-Markov model. Both the statistical quantity $\Pr\{0^m|1\}$ and a histogram of the error-free run lengths are used in this comparison. These three programs are now examined.

Program 1: c18_cdmahmm1.m

In this program the parameters of the CDMA simulation are defined and the simulation is executed. As shown in (18.7), two outputs are generated by the simulation. These are the bit error rate `BER` and the run-length error vector required for determining the Markov model, which is denoted `ErrorRun`. Here we generate the semi-Markov model for the channel operated at $K = 1$ (0 dB) and an E_b/N_0 of 5 dB. The MATLAB program follows:

```
% File: c18_cdmahmm1.m
N = 100000;
EoNdB = 5; EbNo = 10^(EoNdB/10);    % specify Eb/No and Eb/No in dB
KdB = 0; SF = 7; NoI = 0;           % specify parameters
```

```
MPathDelay = [0 3 4];                % specify multipath delay
%
% Run CDMA simulation.
%
[BER,ErrorRun] = c18_cdmasim(N,SF,EbNo,NoI,MPathDelay,KdB);
%
% Develop runlength vector in required form.
%
lenER = length(ErrorRun);
row2 = zeros(1,lenER);
row2(2:2:lenER)=1;
runcode1(1,:) = ErrorRun; runcode1(2,:) = row2;
%
% Generate semi-Markov model.
%
[A_matrix, pi_est] = c15_semiMarkov(runcode1,50,[2 1]);
save cdmadata1 N BER ErrorRun A_matrix runcode1
% End of script file.
```

Executing the first program, `c18_cdmahmm1`, provides the semi-Markov model based on 50 iterations. This result is

$$\widehat{A} = \begin{bmatrix} 0.9494 & 0 & 0.0506 \\ 0 & 0.9714 & 0.0286 \\ 0.2251 & 0.7419 & 0.0330 \end{bmatrix} \tag{18.9}$$

Note that a number of variables are saved to disk. These variables are required in the other two programs and are saved so that the other two programs can be executed at a later date.

Program 2: c18_cdmahmm2.m

In the second program, an error vector is generated by processing a sequence of symbols through the channel represented by the semi-Markov model generated by the preceding program. In addition, the probability of error is generated using three techniques. The first bit error probability is generated by the original CDMA simulation. The second bit error probability is the BER predicted by the semi-Markov model (raising $\widehat{A}$ to a high power as described in Chapter 15). The third bit error probability is determined by counting the errors resulting from passing a large number of symbols (25,000 in this case) through the semi-Markov channel model. The MATLAB program follows:

```
% File: c18_cdmahmm2.m
load cdmadata1                       % load data from c18_cdmahmm1
NN = 25000                           % number of points to be used
[out] = c18_errvector(A_matrix,NN);  % generate error vector
%
```

```
% Compute and display three error probabilities.
%
pe2 = A_matrix^100;
pe2 = pe2(1,3);
pe3 = sum(out/NN);
a = [' The predicted error probabilities for the CDMA system:'];
b = [' From the original simulation PE = ',num2str(BER),'.'];
c = [' Predicted from the semi-Markov model PE = ',num2str(pe2),'.'];
d = [' From the reconstructed error vector ',num2str(pe3),'.'];
%
disp(a)
disp(b) % display PE from simulation
disp(c) % display PE predicted from semi-Markov model
disp(d) % display PE from reconstructed error vecor
save cdmadata2 out
% End of script file.
```

The error vector based on the HMM is generated by the function `c18_errvector`, which is essentially identical to `c15_errvector`, which was originally discussed in Chapter 15. The program `c18_errvector` is given in Appendix C. The only differences between `c18_errvector` and `c15_errvector` is that `c18_errvector` is a function rather than a script and that it generates the error vector for a specific state transition matrix. Executing the second program, `c18_cdmahmm2`, provides the following results:

```
>> c18_cdmahmm2
NN =
    25000
The predicted error probabilities for the CDMA system:
    From the original simulation PE = 0.03185.
    Predicted from the semi-Markov model PE = 0.032054.
    From the reconstructed error vector 0.03204.
```

We see that the three error probabilities agree closely. It should also be noted that the error probabilities agree with the point given on Figure 18.5 for $K = 1$ (middle curve) and $E_b/N_0 = 5$ dB.

Program 3: c18_cdmahmm3

The third program allows comparison of the original error sequence generated by the CDMA simulation and the error sequence generated by the semi-Markov model. We use two comparisons. The first of these is to plot $\Pr\{0^m|1\}$ for both error sequences, as was done in Chapter 15. The second method of comparison is the histogram of the error-free runs for both sequences. The MATLAB code for accomplishing this follows:

```
% File: c18_cdmahmm3.m
load cdmadata1                          % load data from c18_cdmahmm1
load cdmadata2                          % load data from c18_cdmahmm2
runcode2 = c15_seglength(out);
c15_intervals2(runcode1,runcode2)       % display intervals
%
% Build histograms.
%
aa1 = runcode1(1,:);
efd1 = aa1(1:2:length(aa1));
aa2 = runcode2(1,:);
efd2 = aa2(1:2:length(aa2));
figure
subplot(2,1,1)
[N,x] = hist(efd1,20);
%hist(efd1,x)
bar(x,N,1)
xlabel('Histogram Bin')
ylabel('Number of Samples')
subplot(2,1,2)
hist(efd2,x);
xlabel('Histogram Bin')
ylabel('Number of Samples')
% End of script file.
```

Executing the program yields the interval display illustrated in Figure 18.6, and the histograms of the error-free run lengths illustrated in Figure 18.7. In both of these figures, the results from the CDMA system simulation are shown in the top frame, and the results from the semi-Markov model are shown in the bottom frame. Both figures show reasonably good agreement. In Figure 18.7 note the difference in the scaling for the bar heights. This difference in scaling results because the CDMA simulation was performed for 100,000 symbols, and in testing the semi-Markov model, 25,000 symbols were used. Thus, the scale difference is four.

There are several reasons for the differences in the top and bottom frames in Figures 18.6 and 18.7. First, the HMM must be an accurate representation of the original channel. This requires that the data sequence upon which the model is derived be sufficiently long, and that a sufficient number of iterations in deriving the model be performed, in order to ensure that convergence is reached. Also, an appropriate number of states must be used. In addition, once the model is derived it must be tested by processing a sufficient number of symbols in order to obtain statistically reliable results. These concerns are typically addressed by performing experiments such as those discussed here.

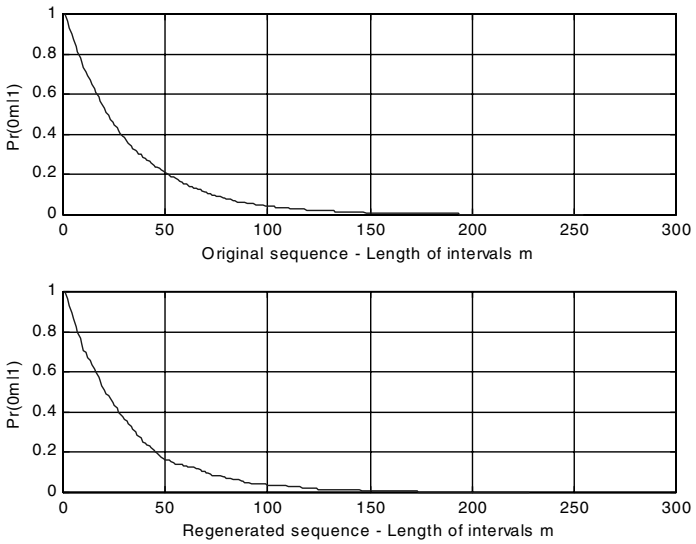


Figure 18.6 $\Pr\{0^m|1\}$ for original error vector (top frame) and $\Pr\{0^m|1\}$ for the reconstructed error vector (bottom frame).

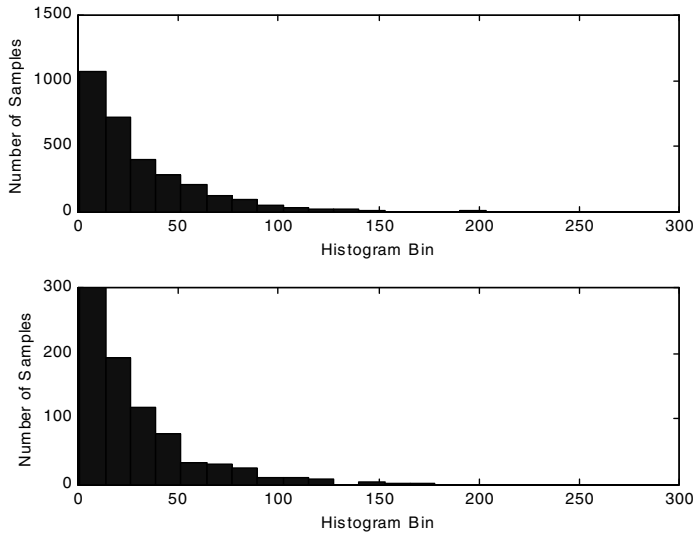


Figure 18.7 Histogram of error-free run for original error vector (top frame) and for the reconstructed error vector (bottom frame).

18.2 FDM System with a Nonlinear Satellite Transponder

The primary objective of this example is to examine a simulation study that illustrates several important modeling and simulation techniques. Important concepts illustrated in this example include the lowpass representation of FDM signals, AM/AM and AM/PM nonlinear models for high-power amplifiers, and semianalytic BER estimation.

18.2.1 System Description and Simulation Objectives

The system being simulated in this example is a communication link in a satellite data network consisting of 48 ground stations sending high-speed data to eight regional data centers. Each data center processes six channels of data, referred to as an FDM group. The modulation format used for each channel is QPSK at a symbol rate of 8 Msymbols/second. The signal is filtered by a square root raised cosine (SQRC) filter[2] with a roll-off factor of 20%, amplified by a linear power amplifier and transmitted in the uplink to the satellite. The 48 ground stations in the network access the satellite using an FDMA scheme in which each ground station is assigned a different carrier frequency by the network controller. Carriers are separated by a guard band of 400 KHz, and the carrier spacing is 10 MHz. The frequency plan for the 48-channel transponder is illustrated in Figure 18.8.

All 48 uplink signals are received by a single uplink antenna at the satellite. The received signal at the satellite can be expressed as

$$x(t) = \sum_{i=1}^{48} a_i(t - \tau_i) \cos\left[2\pi f_i(t - \tau_i) + \phi_i(t - \tau_i)\right] \tag{18.10}$$

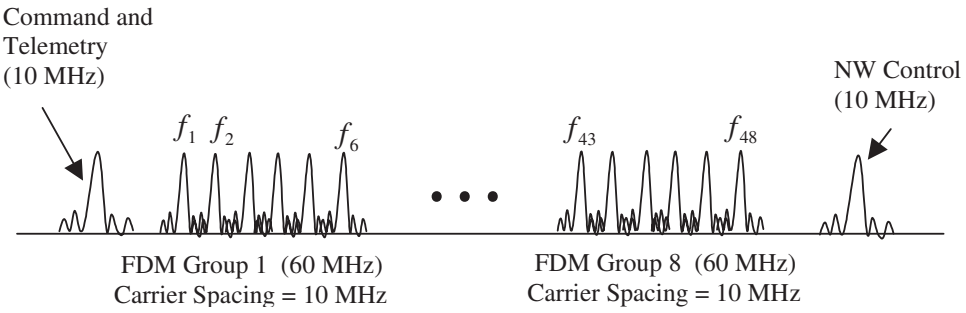


Figure 18.8 Frequency plan for a 48-channel transponder system.

[2]See Example 5.9. Note that the input symbols are represented by a sequence of impulses.

which can be written

$$x(t) = \mathrm{Re}\left\{ \sum_{i=1}^{48} a_i(t-\tau_i) \exp\left[j\phi(t-\tau_i)\right] \right.$$
$$\left. \cdot \exp\left(-j2\pi f_i \tau_i\right) \exp\left[j2\pi(f_i - f_0)t\right] \exp\left(j2\pi f_0 t\right) \right\} \tag{18.11}$$

where f_0 is the reference frequency used to define the overall complex envelope. The lowpass complex envelope from the i^{th} ground station is the complex baseband (QPSK) signal $a_i(t)\exp\left[j\phi_i(t)\right]$. It is transmitted at carrier frequency f_i. The uplink transmitter for the i^{th} ground station is illustrated in Figure 18.9. The time delay, τ_i, represents the propagation delays of the i^{th} channel uplink. The lowpass complex envelope of $x(t)$ is, from (18.11),

$$\widetilde{x}(t) = \sum_{i=1}^{48} a_i(t-\tau_i) \exp\left[j\phi(t-\tau_i)\right] \exp\left(j2\pi f_i \tau_i\right) \exp\left[j2\pi(f_i - f_0)t\right] \tag{18.12}$$

The uplink signals are filtered by the input filters on the satellite, as shown in Figure 18.10, to form eight FDM groups, as shown in the frequency plan (Figure 18.8). Each group is amplified by a high-power traveling wave tube amplifier (TWTA). The amplified signals are filtered again to remove out-of-band spectral regrowth and intermodulation (IM) products. Each carrier group is then transmitted from the satellite to the regional data centers via eight downlink spot beams, with each beam directed toward a regional center.

In order to provide maximum power in the downlink, the TWT amplifiers are operated close to saturation. However, since six FDM carriers are amplified by each TWTA, intermodulation (IM) distortion will have a significant impact on the BER performance in the downlink. Adjacent channel interference (ACI) is another factor that affects link performance. The goal of this study is to simulate one of the downlinks (consisting of one group of 6 FDM carriers) from the satellite to the regional data center and assess the impact of thermal noise, IM distortion, intersymbol interference (ISI) introduced by the filters, and ACI on the downlink performance.

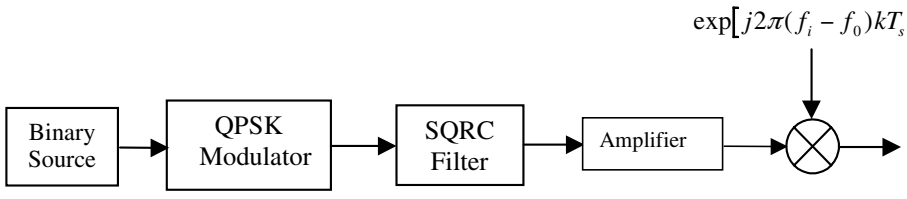


Figure 18.9 The i^{th} transmitter (uplink).

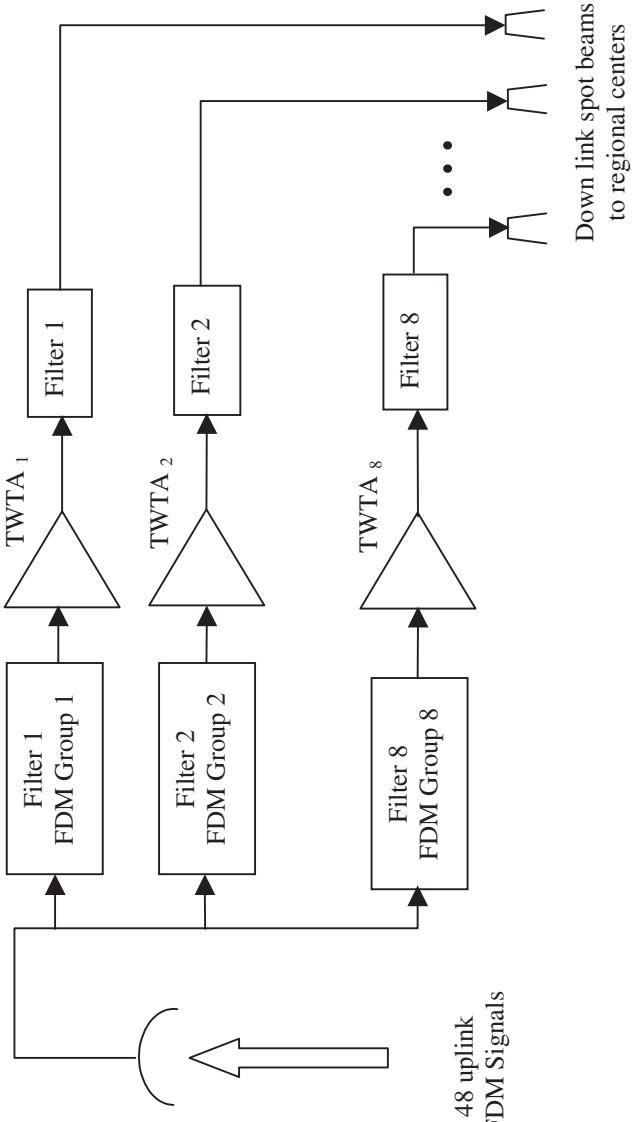


Figure 18.10 Details of satellite transponder.

18.2.2 The Overall Simulation Model

The simulation model is illustrated in Figure 18.11. The FDM signal is generated by mapping six random binary bit streams into QPSK symbols and modulating them at appropriate carrier frequencies. The initial steps in creating the simulation model for this example are the following:

1. Time and frequency scaling: The symbol rate is scaled by $8(10^6)$, that is, the symbol rate is normalized to 1 symbol/second so that all rates and frequencies

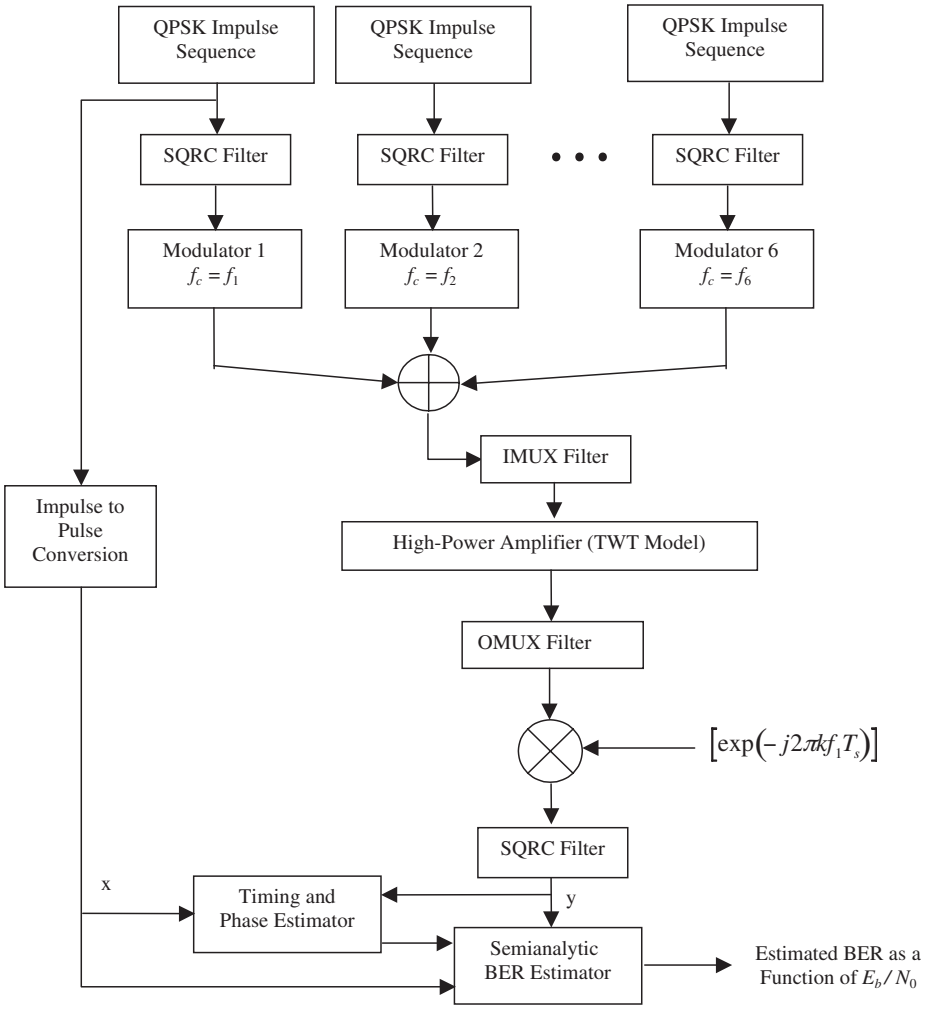


Figure 18.11 Simulation model for satellite communications system.

can be specified in units of Hz rather than MHz, and time units are in seconds rather than microseconds.

2. Reference frequency for lowpass equivalent representation: Lowpass equivalent representations of the signals and system elements are used in this simulation. The six FDM carriers are spaced by $\Delta f = 1.25$ Hz. For simulation purposes we will use a reference frequency of zero corresponding to the center frequency of the six carrier groups so that the carrier frequencies (offsets) in the lowpass equivalent representation will be

$$\begin{array}{ll} f_1 = \Delta f/2, & f_4 = -f_1 \\ f_2 = \Delta f + (\Delta f/2), & f_5 = -f_2 \\ f_3 = 2\Delta f + (\Delta f/2), & f_6 = -f_3 \end{array} \tag{18.13}$$

   This is illustrated in Figure 18.12.

3. Sampling rate: The sampling rate is chosen on the basis of the overall bandwidth of the lowpass equivalent representation shown in Figure 18.12, which is 3.75 Hz. Hence, a sampling rate of $16 \times 3.75 = 60$ will be adequate. Since the symbol rate is 1 symbol/second for each QPSK source, a sampling rate of 64 samples per symbol is chosen for all the simulations.

4. BER estimation method: Since the noise in the downlink is additive and Gaussian, and the receiver is linear, we can use a semianalytic estimator for this example. The semianalytic QPSK BER estimator discussed in Chapter 10 is used.

5. Simulation length: With semianalytic error rate estimation, the simulation length is determined by the number of symbols needed to simulate ISI and adjacent channel interference. Since the filters are SQRC, ISI is mainly due to the input and output multiplexing filters which are fairly wideband with near ideal frequency response. Hence it is not necessary to simulate a very long input sequence. We choose a simulation length of 512 symbols, with the first 256 symbols used for calibration purposes, and the last 256 symbols used for semianalytic BER estimation.

With these overall simulation parameter values established, we now turn our attention to the simulation models for the transmitter, the filters, the nonlinear amplifier, the demodulator, and the semianalytic estimator.

18.2.3 Uplink FDM Signal Generation

The group of six FDM carriers arriving in the uplink from six different ground stations can be modeled as coming from a single ground station, with arbitrary delays and phases associated with each carrier. The signals representing the traffic are generated by mapping the data symbols from each ground station to a QPSK constellation, filtering it through an SQRC filter, and modulating the filter output with the appropriate carrier frequency. The SQRC filters are implemented in the

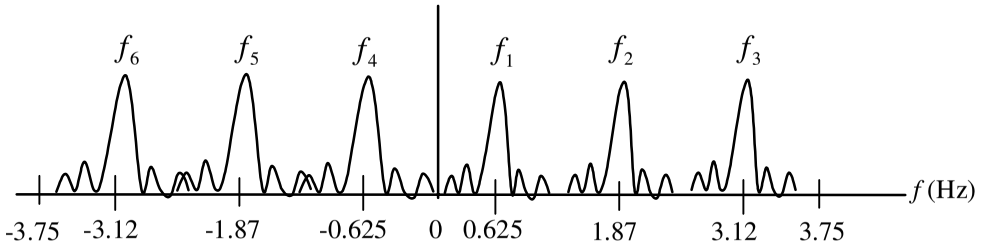


Figure 18.12 Lowpass equivalent representation of the six FDM carriers.

time domain as FIR filters using a sampled and truncated version of the impulse response given by

$$h_R(t) = (8\beta)\,\frac{\cos\left[(R+2\beta)\,\pi t\right] + (8\beta t)^{-1}\sin\left[(R+2\beta)\,\pi t\right]}{(\pi\sqrt{R})[1-(8\beta t)^2]} \tag{18.14}$$

with $R = 1$ and $\beta = 0.2$. The impulse response is truncated to a duration of eight symbols. Since the filters are implemented using the impulse response, the input to the filters are impulse sequences rather than pulse sequences.

The six filtered signals that form an FDM group are modulated and added to create the uplink FDM signal. A phase offset and time delay is introduced for each of the six carriers to represent the random propagation delays from different ground stations to the satellite. These operations can be summarized as:

1. The k^{th} QPSK signal is defined as

$$s_k[n] = \sum_{m=1}^{N} (A_{km} + jB_{km})\,\delta\left(\frac{n}{f_s} - kT_s - \tau_k\right) \tag{18.15}$$

   where A_{km} and B_{km} are the direct and quadrature components of the k^{th} input, $1 \leq k \leq 6$, T_s is the symbol time, τ_k is a time delay, N is the total number of symbols in the sequence, and f_s is the sampling frequency.

2. The k^{th} filtered signal is defined as

$$x_k[n] = s_k[n] \circledast p[n] \tag{18.16}$$

   where $p[n]$ is the impulse response of the root raised cosine filter, as defined by (18.14), and $\circledast$ denotes convolution.

3. The k^{th} modulated signal is defined by

$$y_k[n] = a_k \circledast x_k[n] \circledast \exp\left[j\left(\frac{2\pi n f_k}{f_s} + \theta_k\right)\right] \tag{18.17}$$

   where a_k and f_k are the amplitude and frequency of the k^{th} carrier, respectively, and θ_k represents a random phase offset.

4. The FDM signal is defined by

$$z[n] = \sum_{k=1}^{6} y_k[n] \tag{18.18}$$

18.2.4 Satellite Transponder Model

The satellite transponder consists of an input multiplex (IMUX) filter that isolates each of the eight carrier groups, a TWT amplifier for each six-carrier FDM group, and an output multiplex (OMUX) filter. There is also a frequency translation

operation that takes place in the satellite since the uplink and downlink frequencies are usually different. We will assume this frequency translation to be ideal. For this simulation, the IMUX and OMUX filters are implemented using fourth-order Butterworth filters having a 3-dB bandwidth of 4 Hz. The ISI introduced by these filters is minimal. The TWT amplifier is modeled as an AM-to-AM and AM-to-PM memoryless nonlinearity with unit gain as discussed in Chapter 12. The AM-to-AM and AM-to-PM characteristics are assumed to be given in table form. The "backoff" is the only parameter of the model.

18.2.5 Receiver Model and Semianalytic BER Estimator

The six carriers in each FDM group are demodulated individually at the receiver in the ground station at each of the regional centers. We are interested in estimating the BER for one of the FDM carriers, and the demodulation of any of the six carriers can be accomplished by a frequency translation operation followed by SQRC filtering as shown in Figure 18.13. For demodulation and detection purposes, it is necessary to have timing and phase reference. While these functions are normally performed by the synchronization subsystems in the receiver, we are not explicitly simulating these functional blocks in this example. Synchronization is assumed to be perfect.

The impact of ISI introduced by the filters and the effects of the nonlinear amplifier, including IM distortion and downlink noise, can be estimated using a semianalytic error rate estimator, since the receiver is linear and the downlink noise is additive and Gaussian at the receiver input. A number of calibration operations need to be completed prior to estimating the error rate as a function of the downlink E_b/N_0. These operations include:

1. Estimating E_b at the receiver input for each carrier

2. Cross-correlating the input and output waveforms and estimating timing and phase offsets from the cross-correlation function

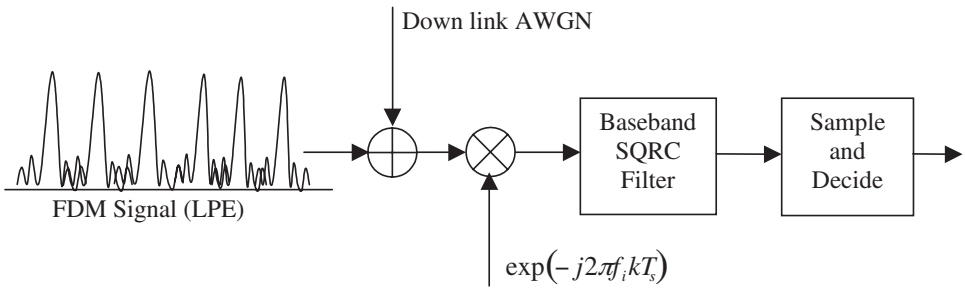


Figure 18.13 Receiver model at the downlink ground station.

3. Estimating the noise bandwidth of the receiver by injecting an impulse at the receiver input, and squaring and adding the values of the impulse response to obtain the noise bandwidth designed by (10.12)

4. Delaying the input waveform appropriately prior to error rate estimation

18.2.6 Simulation Results

In this section we present simulation results illustrating the effects of nonlinear distortion and additive Gaussian noise on the BER performance of the FDM system. The simulation program is given in Appendix D. The purpose of the simulation is to illustrate the combined effect of the nonlinear TWT model, ACI, and additive noise. The simulation products include plots of the power spectral density (PSD) at the input and output of the TWT model, the signal constellation illustrating the effects of the nonlinearity, and the BER due to nonlinear distortion and noise. (Since semianalytic BER estimation is used, the point scattering observed in the signal constellation is due to the nonlinearity, ACI, and filtering. Noise effects are not included in the signal constellation.) Only two simulation results are examined here in detail. By changing the carrier amplitudes, spacing, and the TWT input backoff, the interested student can easily examine a number of interesting effects.

Baseline Validation

It is obviously important to "sanity check" the baseline simulation so that the individual functional blocks in the simulation model and the overall methodology are validated. In all subsequent simulations, the models and the methodology remain the same and only the parameter values are changed. Thus, the baseline validation simulation establishes the credibility for all subsequent simulations. This can easily be accomplished by executing the simulation for a single channel and ensuring that the TWT model is operated in the linear region. By setting

$$a_k = \begin{cases} 1, & k = 1 \\ 0, & k = 2, 3, 4, 5, 6 \end{cases} \tag{18.19}$$

we ensure that ACI does not occur. By operating with an input backoff of -20 dB we ensure that nonlinear distortion within the bandwidth of the Channel 1 signal is negligible. Thus, the only significant degrading effect is additive Gaussian noise, and the result for this case is well-known. (Note that the small amount of ISI induced by the IMUX and OMUX filters is not considered significant.)

The PSD at the input and output of the TWT model, operating at -20 dB backoff, is illustrated in Figure 18.14. The fact that only one channel is active is obvious. Also, spectral spreading due to intermodulation is negligible, which illustrates that the TWT is operating in a region that is very nearly linear.

The signal constellation, without the effects of noise, is illustrated in the left-hand frame in Figure 18.15. Note that there is some slight scattering of the signal points, illustrating that there is a very small residual nonlinear effect and/or ISI present. The signal points in the signal constellation are used to generate the BER

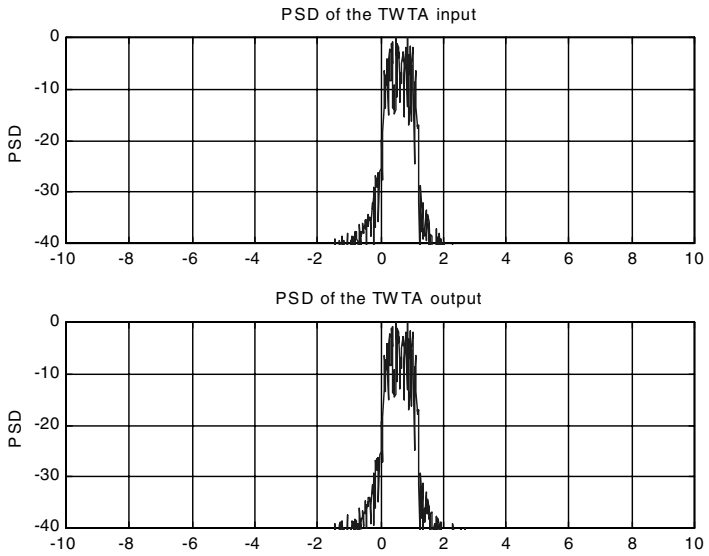


Figure 18.14 PSD at input and output of the TWT model (input backoff is −20 dB.

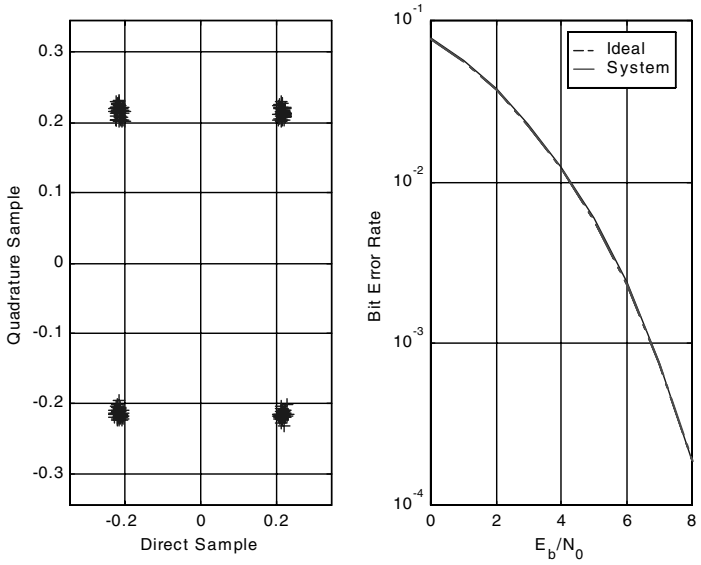


Figure 18.15 Signal constellation due to nonlinear effects, and BER due to noise.

performance, which is illustrated in the right-hand frame of Figure 18.15. Note that the difference between the simulation result and the ideal (theoretical) result is negligible. Thus, the baseline simulation is validated.

Nonlinear and Noise Effects −5 FDM Carriers

In this simulation, the amplitude of Carrier 5 is set equal to zero, with all other carrier amplitudes set equal to one. (Carrier 5 is chosen in order to produce a spectrum with a gap and, also, to ensure that those channels yielding the most significant ACI to Channel 1 are active.) The input backoff of the TWT is set equal to −5 dB in order to dramatically illustrate the nonlinear operation of the TWT model. In other words, we use the parameters

$$a_k = \begin{cases} 0, & k = 5 \\ 1, & k = 1, 2, 3, 4, 6 \end{cases} \tag{18.20}$$

and

$$\texttt{ibo} = -5 \tag{18.21}$$

The computed spectra at the input and output of the TWT are illustrated in Figure 18.16. Note the gap in the input PSD at the spectral position defined by f_5. The PSD at of the TWT output shows that this portion of the spectrum is partially filled in due to the intermodulation distortion caused by the nonlinearity. This "filling in" is called spectral regrowth, and is a major source of error in a FDM system operating with a nonlinearity. The interested student should rerun the simulation with a TWT input backoff of −20 dB, where the TWT operates as a nearly linear amplifier, and show that the spectral regrowth is greatly diminished.

The effects of ISI, ACI, noise, and distortion due to the TWT is illustrated in Figure 18.17. The signal constellation, illustrating the point scattering primarily due to nonlinear effects resulting from the small backoff used in the TWT model, is illustrated in the left-hand frame of Figure 18.17. These signal points are used by the semianalytic BER estimator to determine the BER for the overall system, which is illustrated in the left-hand frame of Figure 18.17. The degrading impact of the nonlinearity on the BER can clearly be seen.

18.2.7 Summary and Conclusions

In this example we presented an end-to-end simulation study of a complex communication system. The example illustrates several fundamental principles in modeling and simulation, including the lowpass equivalent representation of FDM signals, the simulation of nonlinearities, semianalytic BER estimation, verification of the calibration of the simulation, and the overall methodology of simulating a nonlinear multichannel system. We also illustrated a systematic approach to building confidence in the simulation results by starting with a baseline simulation of a near ideal version of the system, and then including transmission impairments one at a time, and comparing the incremental as well as overall performance degradations at each step with the results obtained in the previous step.

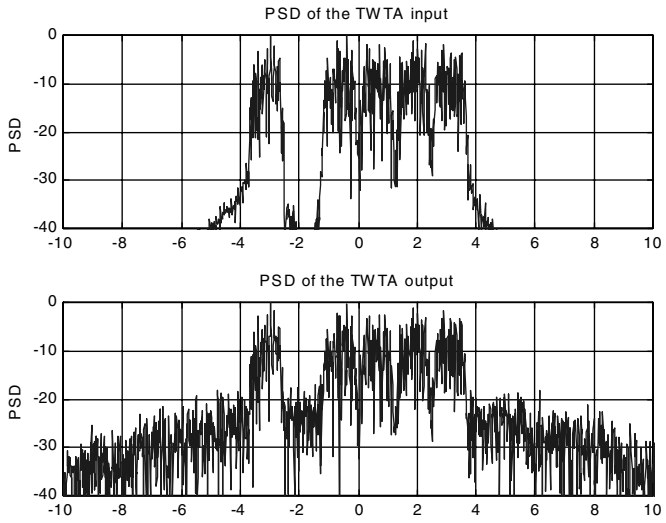


Figure 18.16 PSDs at the input and output of the TWT model (the input backoff is −5 dB).

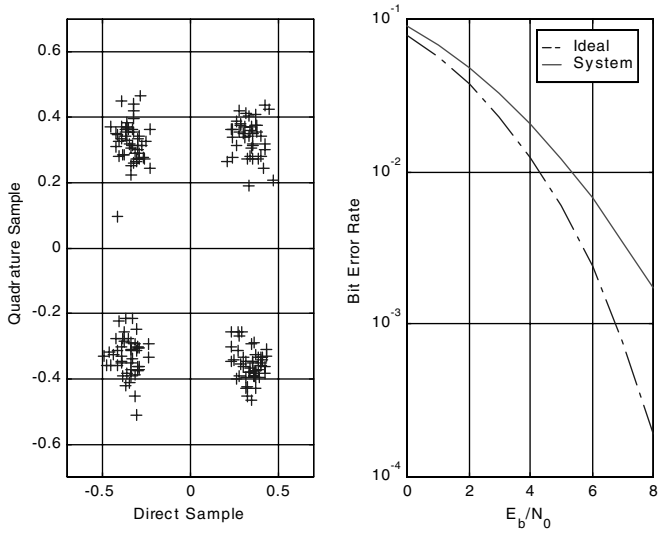


Figure 18.17 BER and signal constellation (input backoff = −5 dB and, for the signal constellation, $E_b/N_0 = 8$ dB).

We hope that this example provided the reader with a better "feel" for how to approach a complex simulation problem.

18.3 References

1. A. J. Viterbi, *CDMA: Principles of Spread Spectrum Communication*, Reading, MA: Addison-Wesley, 1995.

2. R. E. Ziemer and R. L. Peterson, *Introduction to Digital Communication*, 2nd ed., Upper Saddle River, NJ: Prentice Hall, 2001.

3. R. E. Ziemer and W. H. Tranter, *Principles of Communications: Systems, Modulation and Noise*, 5th ed., New York: Wiley, 2002.

18.4 Appendix A: MATLAB Code for CDMA Example

```
% File:  c18_cdmasim.m
function [BER,ErrorRun]=c18_cdmasim(N,SF,EbNo,...
    NumInterferers,MPathDelay,Kfactor_dB)
rand('state',sum(100*clock)); randn('state',sum(100*clock));
NIterate = 1e3;                              % default block size
NumberOfIterations = ceil(N/NIterate);
ErrorState = 0; ErrorRun = []; RunCount = 1; % itialize
Kfactor = 10^(Kfactor_dB/10);                % linear units
EbNolinear = 10^(EbNo/10);                   % linear units
MPathComponents = length(MPathDelay);
%
% Determine amplitude in multipath components and store as
% vector.  Determine the total power in all the scattered components.
% Determine LOS component.
%
MPathAmp(2:MPathComponents) = rand(MPathComponents-1,1);
ScatPower = MPathAmp*MPathAmp.';
MPathAmp(1) = sqrt(ScatPower*Kfactor);       % LOS component.
%
% Determine which component has the largest energy (amplitude).
% Normalize vector so that strongest component has unit amplitude.
%
[fee MaxComponent] = max(MPathAmp); MPathAmp = MPathAmp/fee;
%
% Design IIR filter for fading signal.
%
FilterOrder = 4; Ripple = 0.5; BW = 0.01;     % filter parameters
[b,a] = cheby1(FilterOrder,Ripple,BW);        % 4th order fitler
%
% Error checking.
%
if NumInterferers > (SF-1)
    error(['NumInterferers must not exceed ',int2str(SF-1),'.'])
end
if length(MPathDelay) ~= length(MPathAmp)
    error('MPathDelay and MPathAmp must have the same length.')
end
if min(MPathDelay) < 0
    error('MPathDelay must not have negative components.')
end
fee = diff(MPathDelay);
if min(fee) <= 0
```

```
    error('MPathDelay must be monotonically increasing.')
end
clear fee
%
% End Error Checking.
%
% Generate spreading sequences.  The spreading sequences for
% the interferers are shifted versions of the desired sequence
% with a shift offset.
%
DesiredSequence = MSequence(SF+1); % desired signal
offset = fix(length(DesiredSequence)/(NumInterferers+1));
M = length(DesiredSequence);
for k=1:NumInterferers
    InterfererSequence(k,:)  = [DesiredSequence(M-(k-1)*offset:M) ...
        DesiredSequence(1:M-1-(k-1)*offset)];
end
%
% The simulation loop begins here.
%
zf = zeros(FilterOrder,MPathComponents);
for cnt=1:NumberOfIterations
    %
    % Generate and spread symbols for dsired and interfering users.
    %
    DesiredSymbols = sign(rand(1,NIterate)-0.5);
    InterferingSymbols = sign(rand(NumInterferers,NIterate)-0.5);
    DesiredChips = reshape(DesiredSequence.'*DesiredSymbols,1,...
        M*NIterate);
    for k=1:NumInterferers
        InterferingChips(k,:)  = reshape(InterfererSequence(k,:).'*...
            InterferingSymbols (k,:),1,M*NIterate);
    end
    %
    % Generate noise.
    %
    NoiseAmplitude = sqrt(SF/(2*EbNolinear));
    MaxDelay = max(MPathDelay);
    DesiredNoise = NoiseAmplitude*randn(1,M*NIterate+MaxDelay);
    %
    % Apply multipath.
    %
    MPathLinAmp = MPathAmp;
    MPathComponents = length(MPathDelay);
    DesiredMPathSignal = zeros(1,M*NIterate+MaxDelay);
```

```
if NumInterferers > 0,
    InterferingMPathSignal = ...
        zeros(NumInterferers,M*NIterate+MaxDelay);
    for k=1:MPathComponents
        index = 1+MPathDelay(k):NIterate*M+MPathDelay(k);
        InterferingMPathSignal(:,index) = ...
            InterferingMPathSignal(:,index) + ...
            MPathLinAmp(k)*InterferingChips;
    end
end
for k=1:MPathComponents
    if k==1, fading = ones(1,M*NIterate);
    else
        fading = randn(size(DesiredSymbols))+...
            j*randn(size(DesiredSymbols));
        [fading zf(:,k)] = filter(b,a,fading,zf(:,k));
        fading = interp(fading,SF);
        fading = abs(fading/sqrt(mean(fading.*conj(fading))));
    end
    if k == MaxComponent
        fadesign = sign(fading);
    end
    faa(k,:)  = MPathLinAmp(k)*fading;
    index = 1+MPathDelay(k):NIterate*M+MPathDelay(k);
    DesiredMPathSignal(index) = ...
        DesiredMPathSignal(index)+...
        (MPathLinAmp(k)*fading).*DesiredChips;
end
%
% Add intererence and noise.
%
if NumInterferers > 0
    IncomingSignal = DesiredMPathSignal+...
        sum(InterferingMPathSignal,1)+DesiredNoise;
else
    IncomingSignal = DesiredMPathSignal+DesiredNoise;
end
%
% Receive and detect incoming signal.
%
index = 1+MPathDelay(MaxComponent):M*NIterate+...
    MPathDelay(MaxComponent);
IncomingChips = reshape(fadesign.*...
    IncomingSignal(index),M,NIterate);
DespreadSymbols = DesiredSequence*IncomingChips;
```

```
    DetectedSymbols = sign(DespreadSymbols);
    %
    % Compute Bit Error Rate
    %
    ErrorVector = 0.5*abs(DetectedSymbols-DesiredSymbols);
    ErrorsIterate(cnt) = sum(ErrorVector);
    BERIterate(cnt) = ErrorsIterate(cnt)/NIterate;
    for k=1:NIterate
        if (ErrorVector(k) == 0) & (ErrorState == 0)
            RunCount = RunCount+1;
        elseif (ErrorVector(k) == 0) & (ErrorState == 1)
            ErrorRun = [ErrorRun RunCount];
            RunCount = 1; ErrorState = 0;
        elseif (ErrorVector(k) == 1) & (ErrorState == 0)
            ErrorRun = [ErrorRun RunCount];
            RunCount = 1; ErrorState = 1;
        elseif (ErrorVector(k) == 1) & (ErrorState == 1)
            RunCount = RunCount+1;
        else
            s1 = sprintf('ErrorVector(%d)=%d, ...
                ErrorState=%d!  Unexpected Condition!');
            error(s1);
        end
    end
end
Errors = sum(ErrorsIterate); BER = mean(BERIterate);
ErrorRun = [ErrorRun RunCount];
% End of function file.
```

18.4.1 Supporting Functions

MSquence.m

```
% File:  MSequence.m
function Sequence=MSequence(SF)
SFlog2=log2(SF);
if rem(SFlog2,1) ~= 0, error('SF must be an integer power of 2'); end
switch SFlog2
case 3
    R = 3; instate = zeros(1,3); instate(R) = 1;
    N = 2^R-1; generator = [0 1 3];
case 4
    R = 4; instate = zeros(1,4); instate(R) = 1;
    N = 2^R-1; generator = [0 1 4];
case 5
```

```
    R = 5; instate = zeros(1,5); instate(R) = 1;
    N = 2^R-1; generator = [0 2 5];
case 6
    R = 6; instate = zeros(1,6); instate(R) = 1;
    N = 2^R-1; generator = [0 1 6];
case 7
    R = 7; instate = zeros(1,7); instate(R) = 1;
    N = 2^R-1; generator = [0 3 7];
case 8
    R = 8; instate = zeros(1,8); instate(R) = 1;
    N = 2^R-1; generator = [0 2 3 4 8];
case 9
    R = 9; instate = zeros(1,9); instate(R) = 1;
    N = 2^R-1; generator=[0 4 9];
case 10
    R = 10; instate=zeros(1,10); instate(R) = 1;
    N = 2^R-1; generator = [0 3 10];
case 11
    R = 11; instate = zeros(1,11); instate(R) = 1;
    N = 2^R-1; generator = [0 2 11];
case 12
    R = 12; instate = zeros(1,12); instate(R) = 1;
    N = 2^R-1; generator = [0 1 4 6 12];
otherwise
    error('SF must be a power of 2, >= at 8 and <= 2^12.')
end
[Sequence,Outstate] = ...
    LinearFeedbackShiftRegister(R,generator,instate,N);
Sequence = 1 - 2*Sequence;
% End of function file.
```

LinearFeedbackShiftRegiater,m

```
%File:  LinearFeedbackShiftRegister.m
function [y,outstate]=...
    LinearFeedbackShiftRegister(R,generator,instate,N)
if max(generator) > R
    error(['The degree of the generator polynomial, ',...
        int2str(max(generator)),', cannot exceed R, ',int2str(R),'.']);
end
if length(instate) > R
    error(['The length of the input state vector, ',...
        int2str(length(instate)),', cannot exceed R, ',int2str(R),'.']);
end
```

```
a = sort(generator); P = length(generator); M = length(instate)+1;
for k=1:N
    fee = instate((generator(2)));
    for q=3:P
        fee = bitxor(fee,instate(generator(q)));
    end
    instate = [fee instate]; y(k) = instate(1); instate(M) = [];
end
outstate = instate;
% End of function file.
```

18.5 Appendix B: Preprocessors for CDMA Application

18.5.1 Validation Run

```
% File: c18_cdmacal.m
N = input('Enter number of symbols to be processed > ');
EoN = input('Enter Eb/No vector > ');
SF = 7;
NoI = 0;
MPathDelay = [0 3 4];
KfactordB = 100;
len_EoN = length(EoN);
BER = zeros(1,len_EoN);
h = waitbar(0,'Calibration Run');
for j=1:len_EoN
    EbNo = EoN(j);
    [BER(j),ErrorRun] = c18_cdmasim(N,SF,EbNo,NoI,MPathDelay,...
        KfactordB);
    waitbar(j/len_EoN)
end
close(h)
z = 10.^(EoN/10);
BERT = q(sqrt(2*z));
semilogy(EoN,BER,'+k',EoN,BERT,'-')
xlabel('E_b/N_0 in dB')
ylabel('Probability of Symbol Error')
grid
% End of script file.
```

18.5.2 Study Illustrating the Effect of the Ricean K-Factor

```
% File: c18_cdmaK.m
N = input('Enter number of symbols to be processed > ');
EoN = input('Enter Eb/No vector > ');
KdB = input('Enter KfatordB vector > ');
SF = 7;
NumInterferers = 0;
MPathDelay = [0 3 4];
len_EoN = length(EoN);
len_KdB = length(KdB);
BER = zeros(len_KdB,len_EoN);
for i=1:len_KdB
    KfactordB = KdB(i);
    for j=1:len_EoN
        EbNo = EoN(j);
        [BER(i,j),ErrorRun] = ...
            c18_cdmasim(N,SF,EbNo,NumInterferers,MPathDelay,...
```

```
                KfactordB);
        display = [‘KfactordB = ’,num2str(KfactordB),...
            ‘ Eb/No = ’,num2str(EbNo),‘.’];
        disp(display)
    end
end
semilogy(EoN,BER)
xlabel(‘E_b/N_0 in dB’)
ylabel(‘Probability of Symbol Error’)
grid
% End of script file.
```

18.6 Appendix C: MATLAB Function c18_errvector.m

```
% File: c18_errvector.m
function [out] = c18_errvector(A_matrix,NN)
A = A_matrix;
B = [1 1 0; 0 0 1];
state = 1;                    % initial state
total_states = size(A,1);
out = zeros(1,NN);            % initialize error vector
state_seq = zeros(1,NN);      % initialize state sequence
h = waitbar(0,'Calculating Error Vector');
%
u2 = rand(1);                 % get random number
if u2>B(1,state)              % test for error
    out(1) = 1;               % record error
end
state_seq(1) = state;         % record state
for t=2:NN
    u1 = rand(1);             % get random number
    cum_sum = [0 cumsum(A(state,:))];
    for i=1:total_states      % loop to determine new state
        if u1>=cum_sum(i) & u1<cum_sum(i+1);
            state = i;        % assign new state
        end
    end
    state_seq(t) = state;     % new record state
    u2 = rand(1);             % get random number
    if u2>B(1,state)
        out(t) = 1;           % record error
    end
    waitbar(t/NN)
end
close(h)
% End of function file.
```

18.7 Appendix D: MATLAB Code for Satellite FDM Example

```
% File: c18_nlsat.m
%
% Frequently adjusted parameters
%
a1 = 1.0; a2 = 1.0; a3 = 1.0;              % carrier amplitudes
a4 = 1.0; a5 = 0.0; a6 = 1.0;              % carrier amplitudes
ibo = -5;                                  % TWT input backoff in dB
%
% Default parameters
%
R = 1; tsymbol = 1/R;                      % symbol rate/time
df = 1.244;                                % carrier spacing
beta = 0.2;                                % beta
nsamples = 64;                             % samples/symbol
nsymbols = 256;                            % total symbols
ncorr = 64;                                % symbols for correlation
nstart = 64;                               % BER start (symbol #)
ebn0db = 0:1:8;                            % ebno vector in dB
%
tb = tsymbol/2; fs = nsamples*R; ts = 1/fs;
%
% Set up frequency, delays, and phase offsets
% for the six FDM signals.
%
f1 = (df/2); f2 = f1+df; f3 = f2+df; f4=-f1; f5=-f2; f6=-f3;
delay1 = 0; delay2 = 8; delay3 = 16;
delay4 = 32; delay5 = 40; delay6 = 50;
phase1 = 0.0; phase2 = pi/64; phase3 = pi/32;
phase4 = pi/48; phase5 = pi/16; phase6 = pi/40;
omega1 = 2*pi*f1; omega2 = 2*pi*f2; omega3 = 2*pi*f3;
omega4 = -omega1; omega5 = -omega2; omega6 = -omega3;
%
% Begin Simulation
%
% Carrier 1 is demodulated, so set it up first.
% Genereate the first QPSK impulse sequence and convert to NRZ.
% Note that in vector x, the first nsample/2-1 samples are zero.
%
M = 4;                                          % QPSK
[xin1] = mpsk_impulses(M,nsymbols,nsamples); % QPSK impulse is
                                                %     middle sample
b = ones(1,nsamples);
```

```
x = filter(b,1,xin1);                          % NRZ QPSK waveform
                                               %     samples
nduration = 8;                                 % duration of impw in
                                               %    symbols
[imp1,impw] = sqrc_time(R,beta,nsamples,nduration);
txout1 = filter(imp1,1,xin1);                  % SQRC filtered
                                               %     Chanel 1 signal
%
% Generate 5 other QPSK impulse sequences
%
[xin2] = mpsk_impulses(M,nsymbols,nsamples);
[xin3] = mpsk_impulses(M,nsymbols,nsamples);
[xin4] = mpsk_impulses(M,nsymbols,nsamples);
[xin5] = mpsk_impulses(M,nsymbols,nsamples);
[xin6] = mpsk_impulses(M,nsymbols,nsamples);
%
% Stagger inputs to make unsynchronized
% to minimize envelope transitions
%
[xin2] = delayr1(xin2,delay2);
[xin3] = delayr1(xin3,delay3);
[xin4] = delayr1(xin4,delay4);
[xin5] = delayr1(xin5,delay5);
[xin6] = delayr1(xin6,delay6);
%
% SQRC filter the five symbol waveforms
%
txout2 = filter(imp1,1,xin2);
txout3 = filter(imp1,1,xin3);
txout4 = filter(imp1,1,xin4);
txout5 = filter(imp1,1,xin5);
txout6 = filter(imp1,1,xin6);
%
% Modulate all six carriers and generate txout
%
KNN = 1:nsymbols*nsamples;
txout1 = a1*txout1.*exp(i*omega1*KNN*ts)*exp(i*phase1);
txout2 = a2*txout2.*exp(i*omega2*KNN*ts)*exp(i*phase2);
txout3 = a3*txout3.*exp(i*omega3*KNN*ts)*exp(i*phase3);
txout4 = a4*txout4.*exp(i*omega4*KNN*ts)*exp(i*phase4);
txout5 = a5*txout5.*exp(i*omega5*KNN*ts)*exp(i*phase5);
txout6 = a6*txout6.*exp(i*omega6*KNN*ts)*exp(i*phase6);
txout = txout1+txout2+txout3+txout4+txout5+txout6;
%
% Input MUX filter (Normalized BW = 4.5 Hz)
```

```
%
[bmi ami] = butter(4,4.5/(fs/2));
imuxout = filter(bmi,ami,txout);
%
% TWTA Model
%
run('twt_data1'); twtdata = data;                    % Get TWTA Data
ampout = twt_model(imuxout,twtdata,ibo);             % TWT model
%
% Output MUX filter (Normalized BW = 4.5 Hz)
%
[bmo amo] = butter(4,4.5/(fs/2));
omuxout = filter(bmo,amo,ampout);
%
% Plot (if desired) TWTA input and output PSD
%
psdflag = 1;                                  % omit psd calculation?
if psdflag==1
    nfft = (nsymbols/2)*nsamples;nmax=nsymbols*nsamples;
    temp(1:nfft) = imuxout(nfft+1:nmax);
    [logpsd,freq,ptotal,pmax] = log_psd(temp,4096,ts);
    figure;subplot(2,1,1);
    plot(freq,logpsd); axis([-10 10 -40 0]);
    title('PSD of the TWTA input'); ylabel('PSD'); grid;
    temp(1:nfft) = ampout(nfft+1:nmax);
    [logpsd,freq,ptotal,pmax] = log_psd(temp,4096,ts);
    subplot(2,1,2)
    plot(freq,logpsd); axis([-10 10 -40 0]);
    title('PSD of the TWTA output'); ylabel('PSD'); grid;
end
%
% Compute eb at the input to the RX filter for a single carrier
%
[n1 n2] = size(omuxout); nxx=n1*n2;
eb = 0.5*tb*sum(sum(abs(omuxout).^2))/nxx;
%
% Compute eb for one carrier only
%
eb = eb*(a1*a1)/(a1*a1+a2*a2+a3*a3+a4*a4+a5*a5+a6*a6)
%
% Demodulate: Frequency shift the first carrier
%
ydemod = omuxout.*exp(-i*omega1*KNN*ts);
%
% Receive SQRC filter
```

```
%
nduration = 16;
[imp2,impw] = sqrc_freq_nosinc(R,beta,nsamples,nduration);
y = filter(imp2,1,ydemod);
%
% Simulation is complete.
%
% Set up BER estimator.
%
hh = impz(imp2,1); ts=1/fs;
nbw = (fs/2)*sum(abs(hh).^2) % noise BW of the receiver
corlength = ncorr*nsamples;
%
% Find the maximum magnitude of the cross correlation
% and find the lag corresponding to it.
%
[cor lags] = vxcorr(x(1:corlength),y(1:corlength));
cmax = max(max(abs(cor))); nmax = find(abs(cor)==cmax);
timelag = -lags(nmax); corrmag = cmax; theta = -angle(cor(nmax));
%
y = y*exp(-i*theta);                          % derotate
%
% Delay the input, and do BER estimation starting with nstart.
% Make sure the index does not exceed number of input points.
% eb is the true (real) signal power computed at the RX input.
% Decision time is the mid sample in each bit + timing_offset.
%
maxindex = nsymbols*nsamples-(3*nsamples/2)-timelag;
%
% Index is the array of pointers to mid points of each symbol
% Start BER estimation at nstart (skip the first few symbols).
%
index = ((nstart)*nsamples+(nsamples/2):nsamples:maxindex);
%
timelag1 = timelag+(nsamples/2); % middle sample decision statistics
xx = x(index);                   % first symbol in x is first
                                 %     nsample/2
yy = y(index+timelag1+1);
%
% QPSK BER estimation
%
[peideal,pesystem] = qpsk_berest(xx,yy,ebn0db,eb,tb,nbw);
%
% Plot results
%
```

```
figure; subplot(1,2,1)
yscale = 1.5*max(real(yy));
plot(yy,'+')
xlabel('Direct Sample'); ylabel('Quadrature Sample'); grid;
axis([-yscale yscale -yscale yscale])
subplot(1,2,2)
semilogy(ebn0db,peideal,'-.',ebn0db,pesystem); grid;
xlabel('E_b/N_0'); ylabel('Bit Error Rate')
legend('Ideal','System')
% End of script file.
```

18.7.1 Supporting Functions

A number of the supporting functions for this exmple appeared previously and are not given here. These are:

Program `qpsk_berest.m` is defined in Appendix D of Chapter 10.

Program `log_psd.m` is defined in Appendix A of Chapter 7.

Program `vxcoor.m` is defined in Appendix B of Chapter 10.

mpsk_impulses.m

```
function [x]=mpsk_impulses(M,nsymbols,nsamples)
% This function generates a random complex MPSK impulse sequence
% nsymbols in length. Each symbol is sampled at a rate of
% nsamples/bit. All samples except the 'middle' sample within a
% symbol interval are zero.
u = rand(1,nsymbols);
rinteger = round((M*u)+0.5);
phase = pi/M+((rinteger-1)*(2*pi/M));
x = zeros(1,nsymbols*nsamples);
for m=1:nsymbols
    index = (m-1)*nsamples + round(nsamples/2);
    x(1,index) = exp(i*phase(m));
end
% End of function file.
```

sqrc_time.m

```
% File: sqrc_time.m
function [imp,impw] = sqrc_time(R,beta,nsamples,nsymbols)
%
%
% This function provides the impulse response for
% an FIR implementation of the sqrc filter
% ****** This version does not include a 1/sinc ******
```

```
% R is the symbol rate; 0<beta<1; beta is normalized by R/2
% nsamples is the number of samples per symbol
% nsymboles is the duration of the impulse response in symbols
% Impulse response extends from -nsymbols/2 to +nsymbols/2
%
beta = beta*(R/2);
a = R+(2*beta); b = R-(2*beta);
length1 = fix(nsamples*nsymbols/2);
ts = (1/R)/nsamples;
time = [-length1*ts:ts:(length1-1)*ts] + 0.00000013;
n = length(time); w = hanning(n);
term1 = a*pi*time; term2 = b*pi*time;
cos1 = cos(a*pi*time); sin1 = sin(b*pi*time);
denominator = (pi*sqrt(R))*((1-((8*beta*time).^2)));
for k=1:n
    numerator = (8*beta*(cos1(k)))+(sin1(k)/time(k));
    if(denominator(k)==0)
        imp(k) = 1;
    else
        imp(k) = numerator/denominator(k);
    end
    impw(k) = imp(k)*w(k);
end
yy = sum(impw.^2);
impw = impw/(yy^0.5);
impw = impw/sqrt(nsamples);
yy = sum(imp.^2);
imp = imp/(yy^0.5);
imp = imp/sqrt(nsamples);
% End of function file.
```

sqrc_freq_nosinc.m

```
% File: sqrc_freq_nosinc.m
function [imp,impw] = sqrc_freq_nosinc(R,beta,nsamples,nsymbols)
% This function computes the impulse response of an SQRC
% filter using a frequency domain/ifft approach; It includes a
% 1/sinc; R = symbol rate; 0<beta<1
% nsamples = number of samples per symbol
% nsymbols = total duration of the impulse responce in symbols
% Number of points in the impulse response = nsymbols*nsamples
% R = 1; nsymbols = 16; nsamples = 32; beta = 0.25
%
n = nsamples*nsymbols; fs = R*nsamples; df = fs/n;
a = ((1-beta)*0.5*R); b = ((1+beta)*0.5*R);
%
```

```
% Fill up the transfer function array with zeros upto fs/2
% Compute and store the values of sqrc up to R
%
H = (zeros(1,n));
for k=1:(n/2)+1
    f = (k-1)*df;
    if(beta==0)
        beta=0.0000001;
    end
    H(k) = cos((pi/(2*beta*R))*(f-a));
    if f < a
        H(k) = 1;
    end
    if f>=b
        H(k) = 0;
    end
end
%
% Fold the negative frequency components to [fs/2/to fs]
%
H((n/2)+2:1:n) = H((n/2):-1:2);
%
%Take inverse fft
%
[impc,time] = linear_fft(H,n,df);
imp = real(impc);
window = hanning(n)';
impw = imp.*window;
yy = sum(impw.^2);
impw = impw/(yy^0.5);
impw = impw/sqrt(nsamples);
yy = sum(imp.^2);
imp = imp/(yy^0.5);
imp = imp/sqrt(nsamples);
% End of function file.
```

delayr1.m

```
function [xout] = delayr1(xin,ndelay)
n = max(max(size(xin)));
if (ndelay==0)
    xout = xin;
else
    xx = zeros(1,ndelay);
    xout = [xx xin(1,1:n-ndelay)];
end
```

```
% End of function file.
```

twt_model.m

```
% File: twt_model.m
function [y] = twt_model(x,twtdata,ibo)
% ibo is a negative value in db indicating the operating point
% ibo = -3db implies that the input POWER is attenuated by 1/2
% When the input power = pave, output power is maximum (=1)
% Normalized input power = (pinstantaneous/pave)*optn_real
%
pwrin = twtdata(:,1); pwrout = twtdata(:,2); phaseout = twtdata(:,3);
n = length(x); ibo_real = 10^(ibo/10);}
%
% Find the average input power and the instantaneous input power
% Normalize the instantaneous input power by the average power.
%
instpwr = 0.5*(abs(x).^2);
avepwr = mean(instpwr);
instpwr = instpwr*(ibo_real/avepwr);
%
% Compute output power and phase
%
outpwr = interp1(pwrin,pwrout,instpwr);
outmag = sqrt(2*outpwr);
outphase = (pi/180)*interp1(pwrin,phaseout,instpwr);
%
% Compute complex envelope of the output
%
y = outmag.*exp(sqrt(-1)*(outphase+angle(x)));
% End of function file.
```

twtdata1.m

```
% File: twt_data1.m
% x in a vector of input complex envelope values
% y is a vector of output complex envelope values
% First column is input power in real values (real power)
% Second column is real output power;
% Third column is phase offset in degrees.
% Data is assumed to be increasing order of input power
% First entry should be input power = 0.0, output power = 0.0
% Last entry should be well beyond the max value of the input
% signal power.
%
data=[
```

```
0.0000        0.0000        32.8510
0.0110        0.0500        32.8510
0.0120        0.0560        32.7570
0.0140        0.0630        32.6540
0.0150        0.0700        32.4840
0.0170        0.0790        32.3210
0.0190        0.0880        32.1930
0.0220        0.0980        31.9740
0.0250        0.1100        31.5650
0.0280        0.1230        31.2160
0.0310        0.1370        30.8170
0.0350        0.1520        30.4400
0.0390        0.1690        30.0840
0.0440        0.1890        29.6240
0.0500        0.2100        29.1450
0.0560        0.2330        28.6350
0.0630        0.2570        28.0410
0.0700        0.2840        27.3180
0.0790        0.3130        26.3990
0.0890        0.3440        25.6150
0.1000        0.3770        24.8700
0.1120        0.4130        23.9820
0.1250        0.4510        23.0500
0.1410        0.4910        22.0380
0.1580        0.5320        21.0060
0.1770        0.5730        19.9700
0.1990        0.6150        18.6400
0.2230        0.6570        17.3930
0.2510        0.6970        16.0620
0.2820        0.7370        14.8880
0.3160        0.7770        13.5740
0.3540        0.8130        12.2540
0.3980        0.8480        11.0110
0.4460        0.8770         9.7070
0.5010        0.9050         8.3830
0.5620        0.9320         6.9150
0.6300        0.9510         5.4690
0.7070        0.9700         4.1550
0.7950        0.9810         2.8770
0.8910        0.9920         1.4660
1.0000        1.0000         0.0000
1.1220        0.9980        -1.5710
1.2580        0.9980        -3.0480
1.4120        0.9890        -4.7500
1.5840        0.9820        -6.5670
```

```
1.7780        0.9760         -8.3850
1.9950        0.9700        -10.2020
2.2480        0.9630        -12.0190
2.5110        0.9570        -13.8360
2.8120        0.9510        -15.6530
3.1620        0.9450        -17.4710
3.5480        0.9380        -19.2880
3.9810        0.9320        -21.1050
4.4660        0.9260        -22.9220
5.0110        0.9200        -24.7390
5.6230        0.9140        -26.5570];
% End of data file.
```

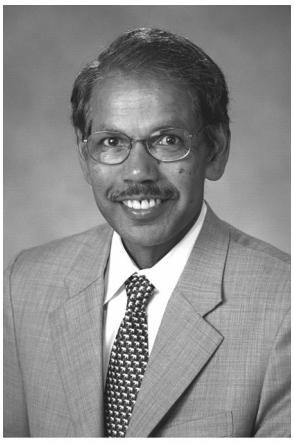
INDEX

ABOUT THE AUTHORS

William H. Tranter

William H. Tranter received the Ph.D. degree from the University of Alabama in 1970. He joined Virginia Tech in July 1997 and serves as the Bradley Professor of Communications and as Director of the Mobile and Portable Radio Research Group. Prior to coming to Virginia Tech, he served for twenty-six years at the

University of Missouri–Rolla (UMR), most recently as the Schlumberger Professor of Electrical Engineering. In 1996, he was a visiting Erskine Fellow at Canterbury University in Christchurch, New Zealand. Dr. Tranter is the co-author of two classic textbooks in communications and signal processing, *Principles of Communications: Systems, Modulation, and Noise*, 5th edition (Wiley, 2001), and *Signals and Systems: Continuous and Discrete*, 4th edition (Prentice Hall 1998). He has received numerous awards for outstanding research and service contributions, including the IEEE Centennial Medal, the IEEE Third Millennium Medal, the IEEE Communications Society Donald W. McLellan Meritorious Service Award, and the IEEE Communications Society Exemplary Publications Award. He also received a number of teaching awards from the University of Missouri–Rolla and from Tau Beta Pi and IEEE. He has served as Editor-in-Chief of the *IEEE Journal on Selected Areas in Communications* and as Director of Journals for the IEEE Communications Society. He currently serves a Vice President–Technical Activities of the IEEE Communications Society. He was elected a Fellow of the IEEE in 1985.



K. Sam Shanmugan

K. Sam Shanmugan received the Ph.D. degree from Oklahoma State University, Stillwater, in Electrical Engineering. He is currently the SBC Professor of Telecommunication in the Electrical Engineering and Computer Science Department at the University of Kansas. Dr. Shanmugan's current research interests are in the areas of wireless communications and computer-aided modeling and analysis of communication systems. Dr. Shanmugan is the author of over 100 publications in the above areas and is the author/co-author of three books, *Digital and Analog Communication Systems* (Wiley, 1979), *Random Signals: Detection Estimation and Data Analysis* (Wiley, 1988), and *Simulation of Communication Systems*, 2nd edition (Kluewer Academic Press, 2000). From 1985 to 1995 Dr. Shanmugan served in a number of leadership positions in the industry: President of STA*R corporation (1985–88), Senior Vice President of Comdisco Systems (1988–93), General

Manager of the Alta Group of Cadence Design Systems (1994–95), and Chief Technical Officer of Systems and Networks (1995–96). During this time he led the development and commercialization of modeling and simulation software for the design of communication systems (BOSS-Block Oriented Systems Simulator) and networks (BONeS-Block Oriented Network Simulator). Dr. Shanmugan was elected a Fellow of the IEEE in 1985 for his contributions to the area of computer-aided design of communication systems. He is also the recipient of many teaching and research awards at the University of Kansas.

Theodore S. Rappaport

Theodore S. Rappaport received the Ph.D. degree from Purdue University in 1987, and was a faculty member at Virginia Tech from 1988–2002. There, he founded the Mobile and Portable Radio Research Group and served as the James S. Tucker Professor of Engineering. In 2002, he joined the ECE faculty at the University of Texas in Austin where he founded the Wireless Networking and Communications Group (WNCG). Dr. Rappaport has 30 patents issued or pending and has authored, co-authored, and co-edited numerous books in the wireless field, including the popular textbooks *Wireless Communications: Principles & Practice* (Prentice-Hall, 1996, 2002), and *Smart Antennas for Wireless Communications: IS-95 and Third Generation CDMA Applications* (Prentice Hall, 1999). He was a recipient of the 1999 Stephen O. Rice Prize Paper Award from the IEEE Communications Society, and the 2002 ASEE Frederick E. Terman outstanding educator award. Rappaport has founded two successful high-tech companies, and is presently Chairman and CEO of Wireless Valley Communications, Inc., a leading software simulation and measurement product company that provides solutions for in-building wireless network deployment and management.

Kurt L. Kosbar

Kurt L. Kosbar is an associate professor of electrical and computer engineering at the University of Missouri–Rolla. He received B.S. degrees in electrical engineering and computer engineering from Oakland University, and M.S. and Ph.D. degrees in electrical engineering from the University of Southern California. While performing his graduate work at USC, Dr. Kosbar was employed by Hughes Aircraft Company, Space and Communications Group, as a staff engineer. Dr. Kosbar was a Fulbright fellow at the Technical University of Gdansk, is a member of the IEEE, and has been inducted into the Tau Beta Pi, Eta Kappa Nu, and Phi Kappa Phi honor societies.