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^2_e, 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
```

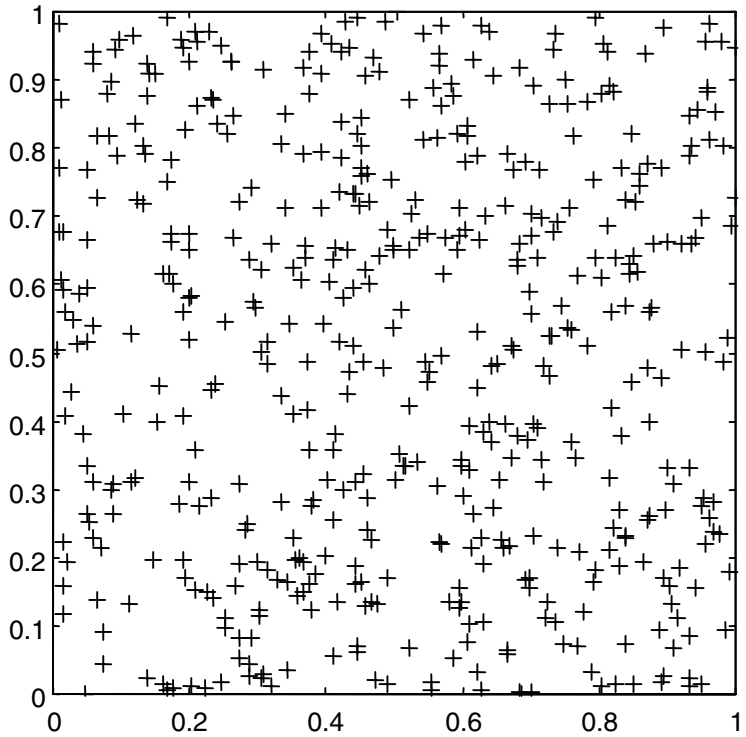


Figure 9.1 Uniformly distributed random points.

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.

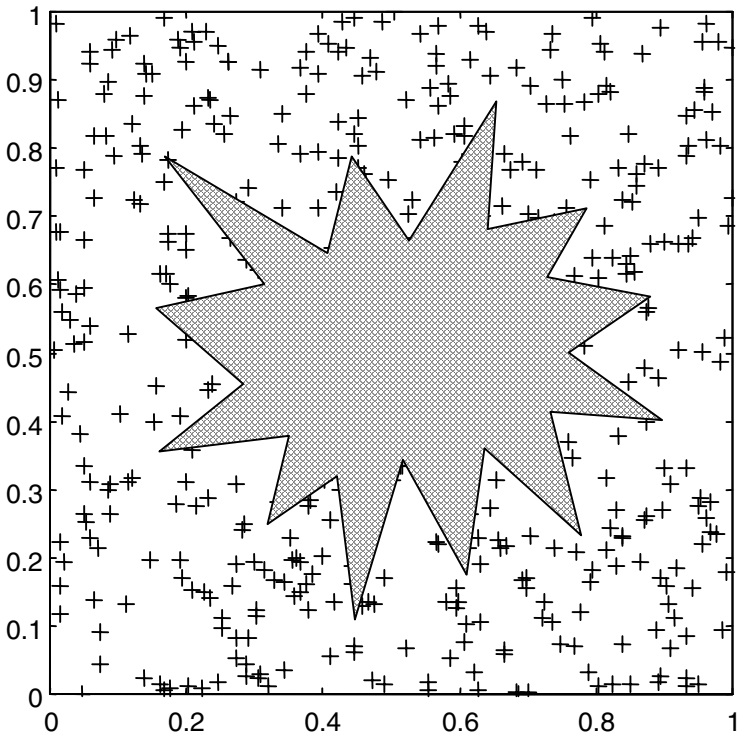


Figure 9.2 Monte Carlo estimation of an area.

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].

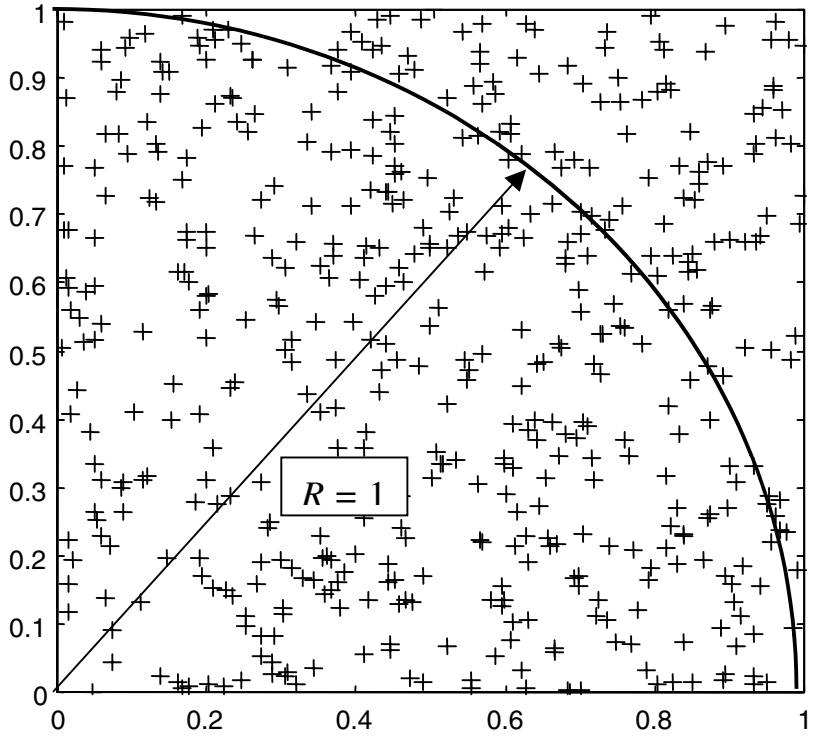


Figure 9.3 Estimation of π.

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
```

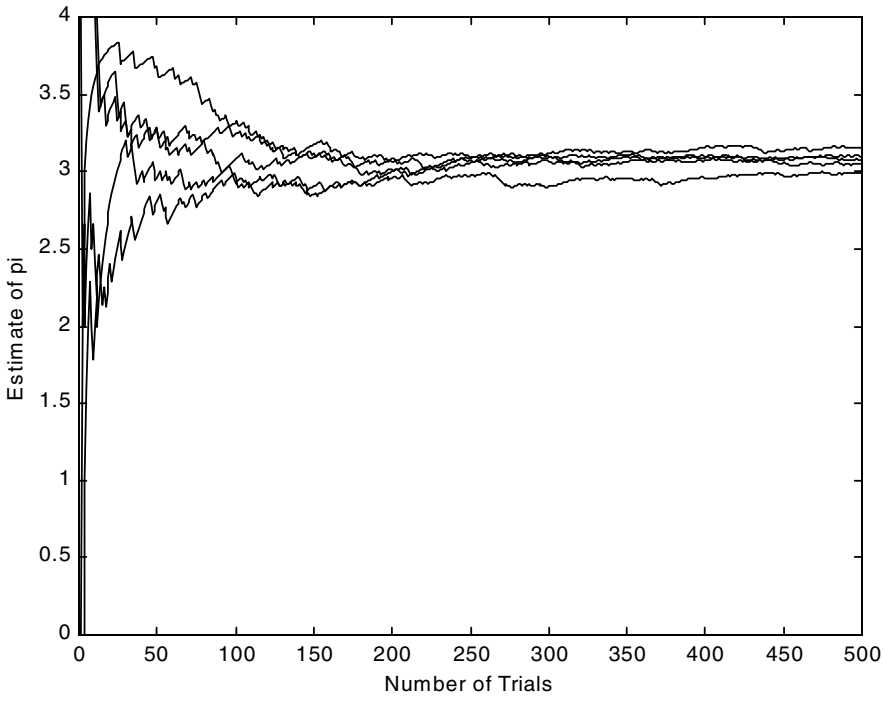


Figure 9.4 Monte Carlo estimate of π.

```
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

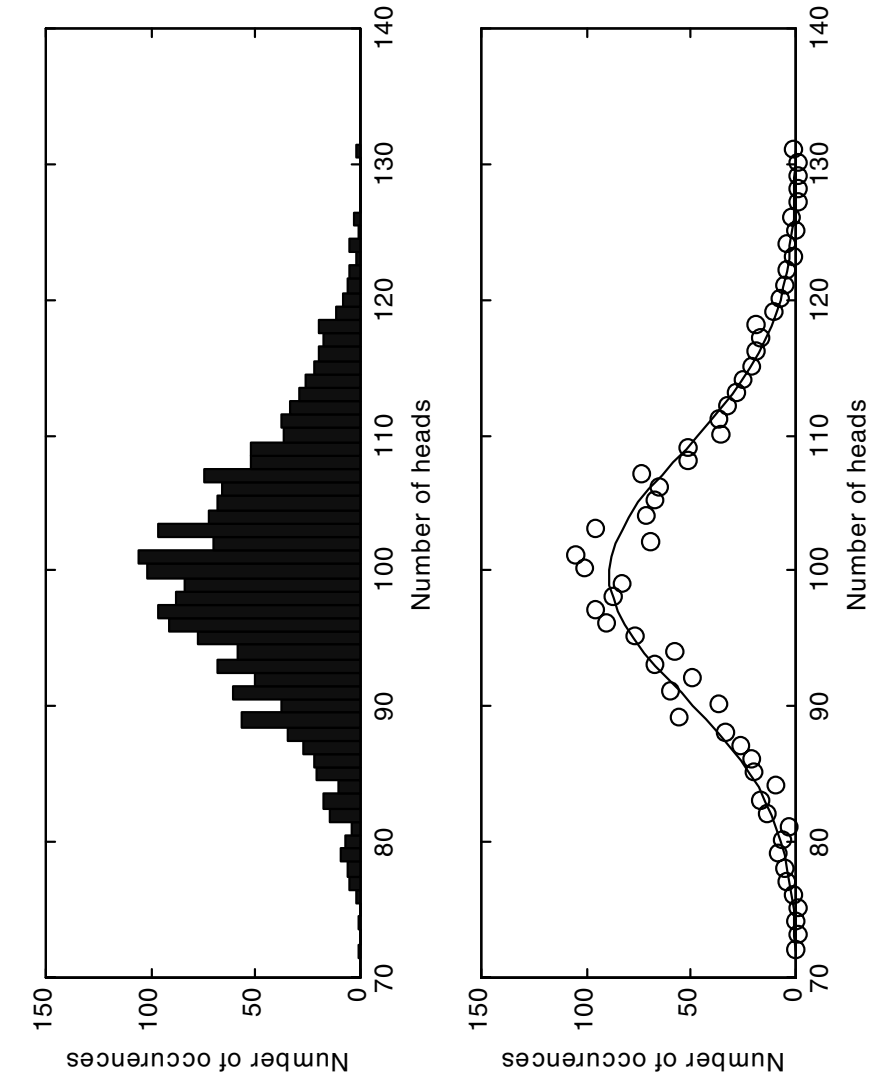


Figure 9.5 Result of coin-tossing experiment.

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,

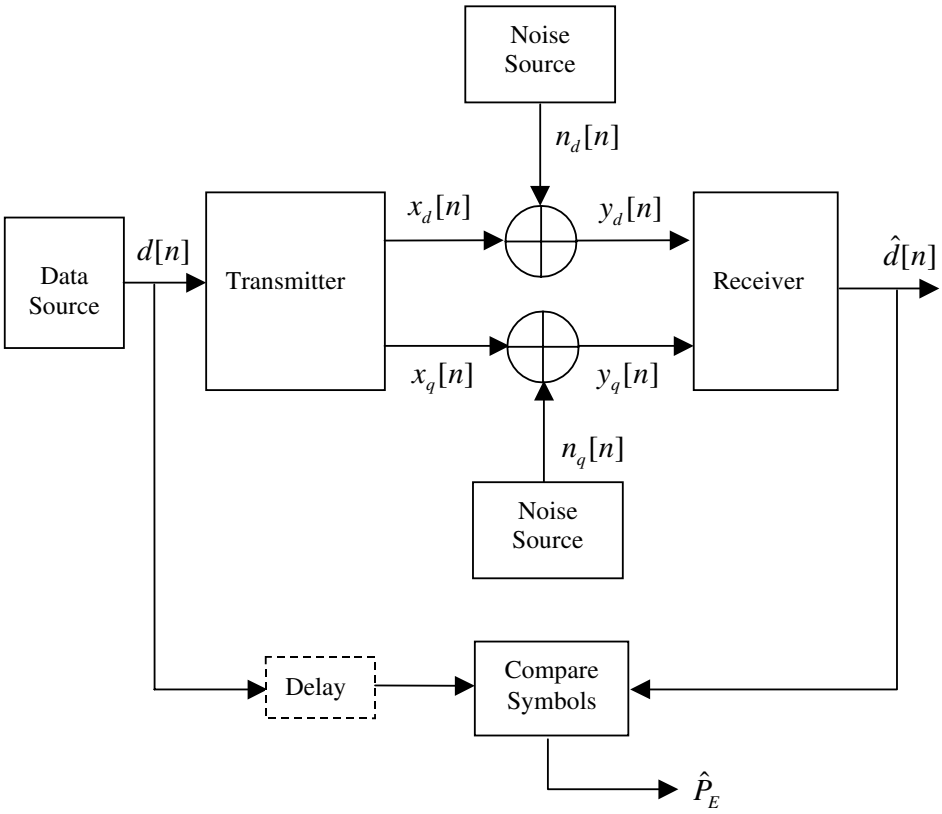


Figure 9.6 Simulation model for simple communication system.

$x_d(t)$ 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

$$x_d[n] = \cos(\pi d[n]) = \begin{cases} 1, & d[n] = 0 \\ -1, & d[n] = 1 \end{cases} \tag{9.30}$$

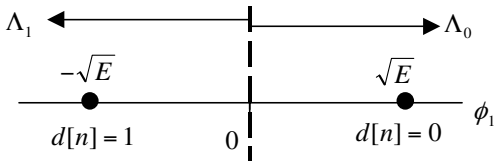


Figure 9.7 Signal space representation of binary PSK.

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. ■

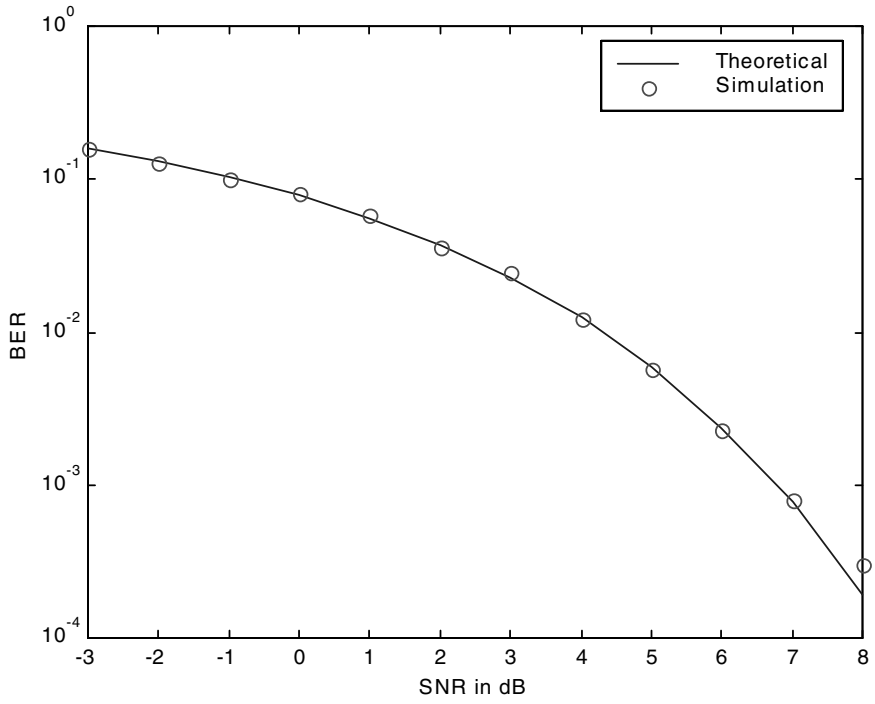


Figure 9.8 Binary phase-shift keying.

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

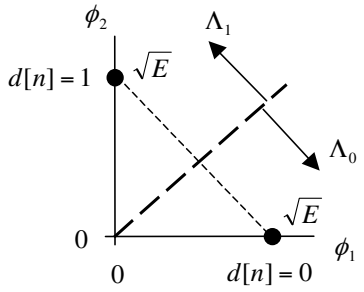


Figure 9.9 Signal-space representation for binary FSK.

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

$$\Pr\left(E|d[n]=1\right) = \int_{k_T}^{\infty} \frac{1}{\sqrt{2\pi}\sigma_n} \exp\left[-\frac{1}{2\sigma_n^2}\left(x-\nu_1(T)\right)^2\right] dx \qquad (9.36)$$

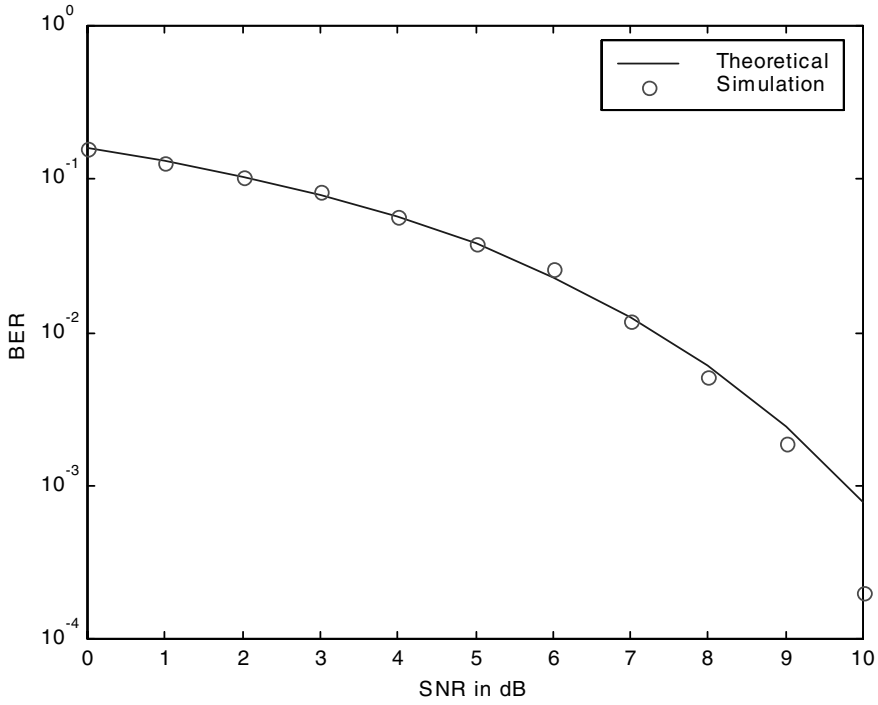


Figure 9.10 Binary frequency-shift keying.

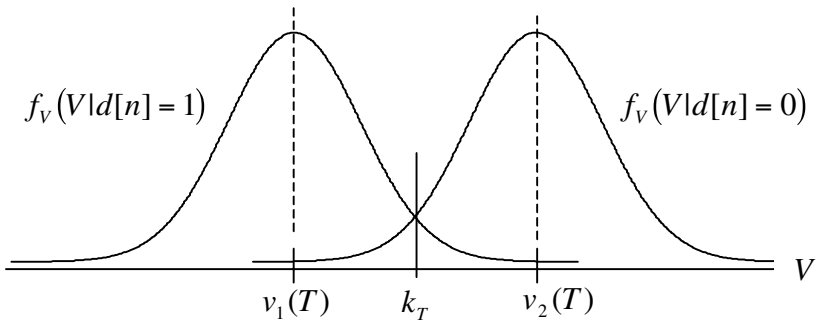


Figure 9.11 Conditional pdfs for binary signaling in Gaussian noise.

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.
```

■

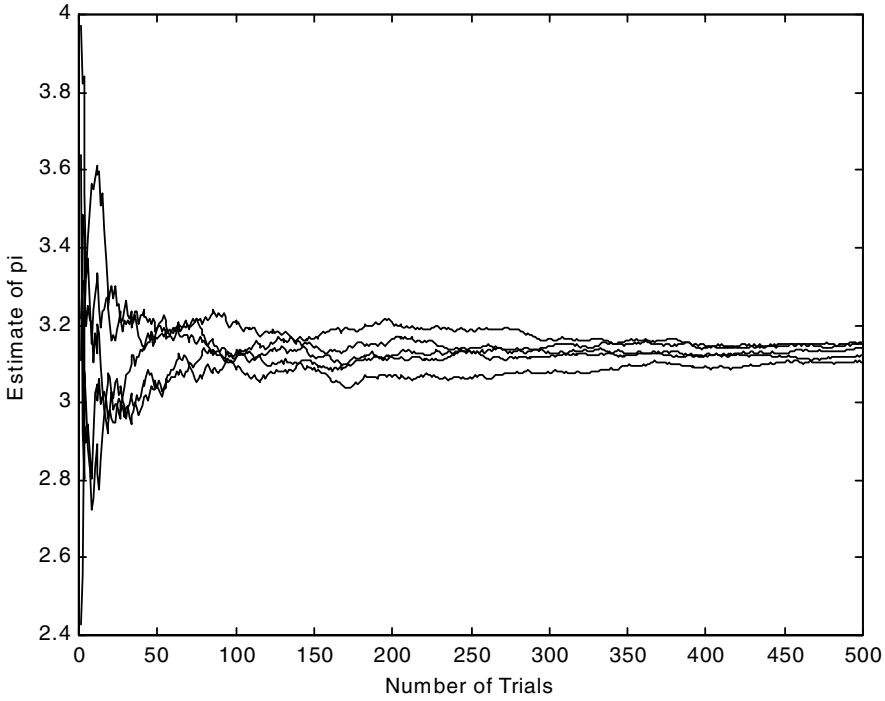


Figure 9.12 Estimation of π using Monte Carlo integration.

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

$$\Pr\left\{-\beta\sigma_{\widehat{I}} \le \widehat{I} - I \le \beta\sigma_{\widehat{I}}\right\} = 1 - \alpha \tag{9.53}$$

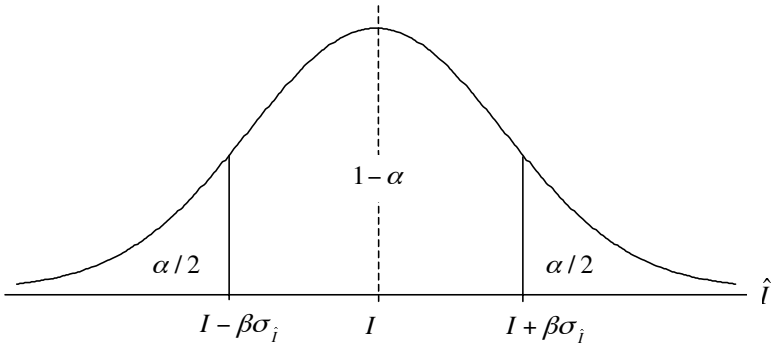


Figure 9.13 Confidence interval.

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. ■

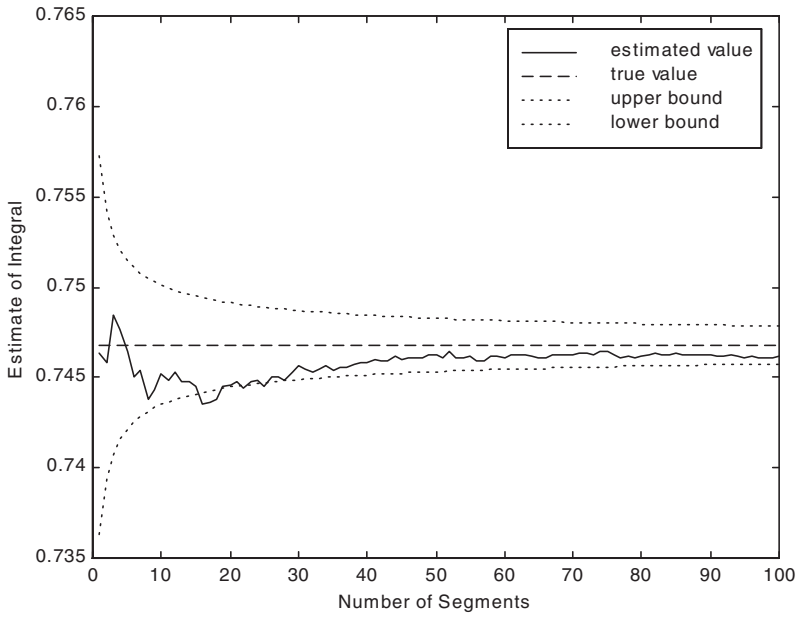


Figure 9.14 Monte Carlo estimate and 90% confidence interval.

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

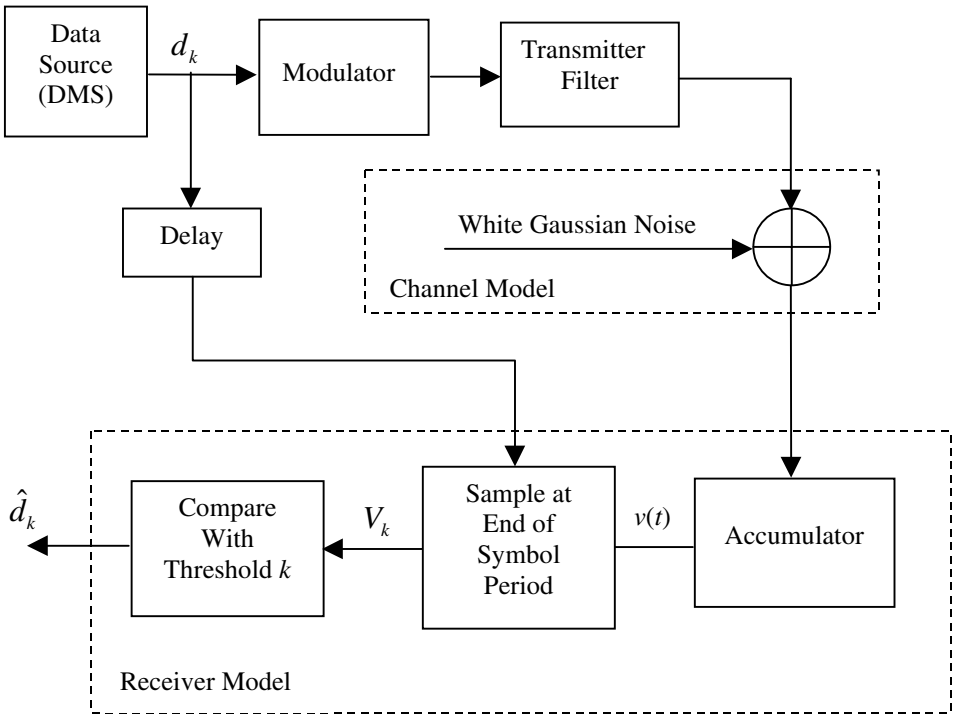


Figure 10.1 Basic communications system.

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

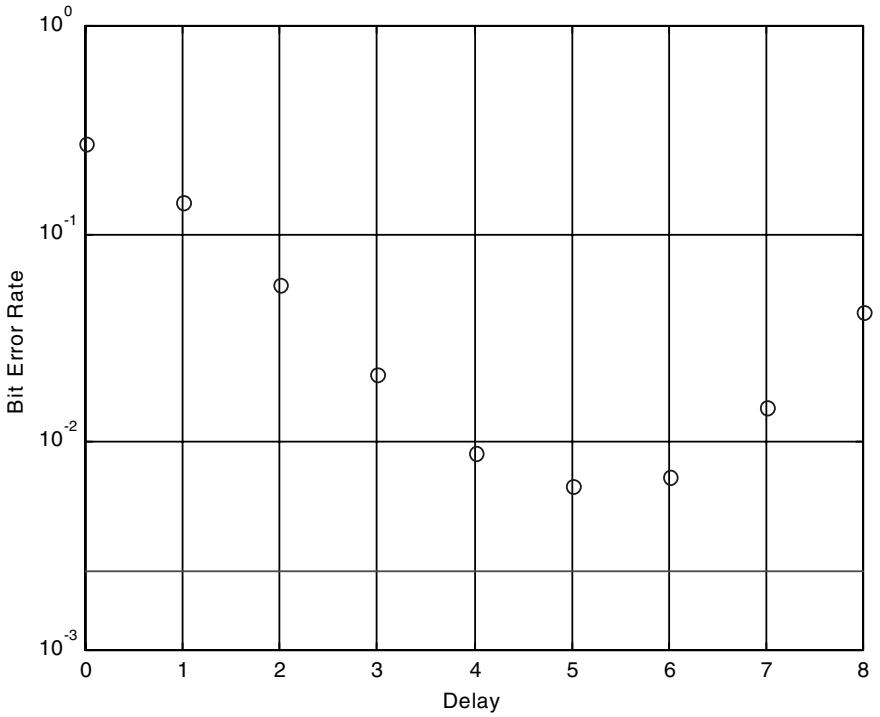


Figure 10.2 Preliminary simulation used to determine delay.

(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

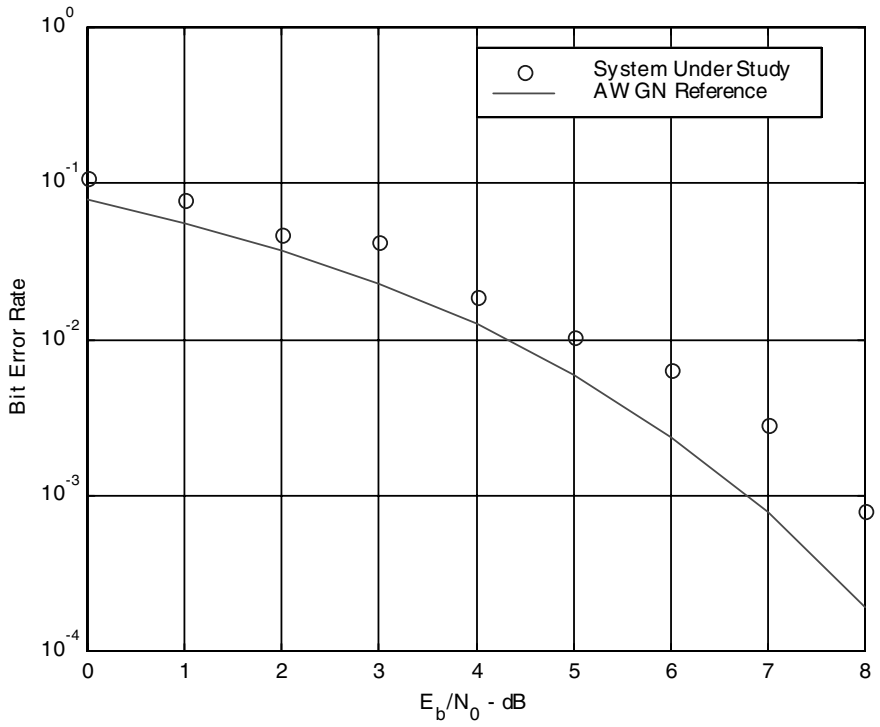


Figure 10.3 Binary PSK simulation for system with ISI.

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));
```

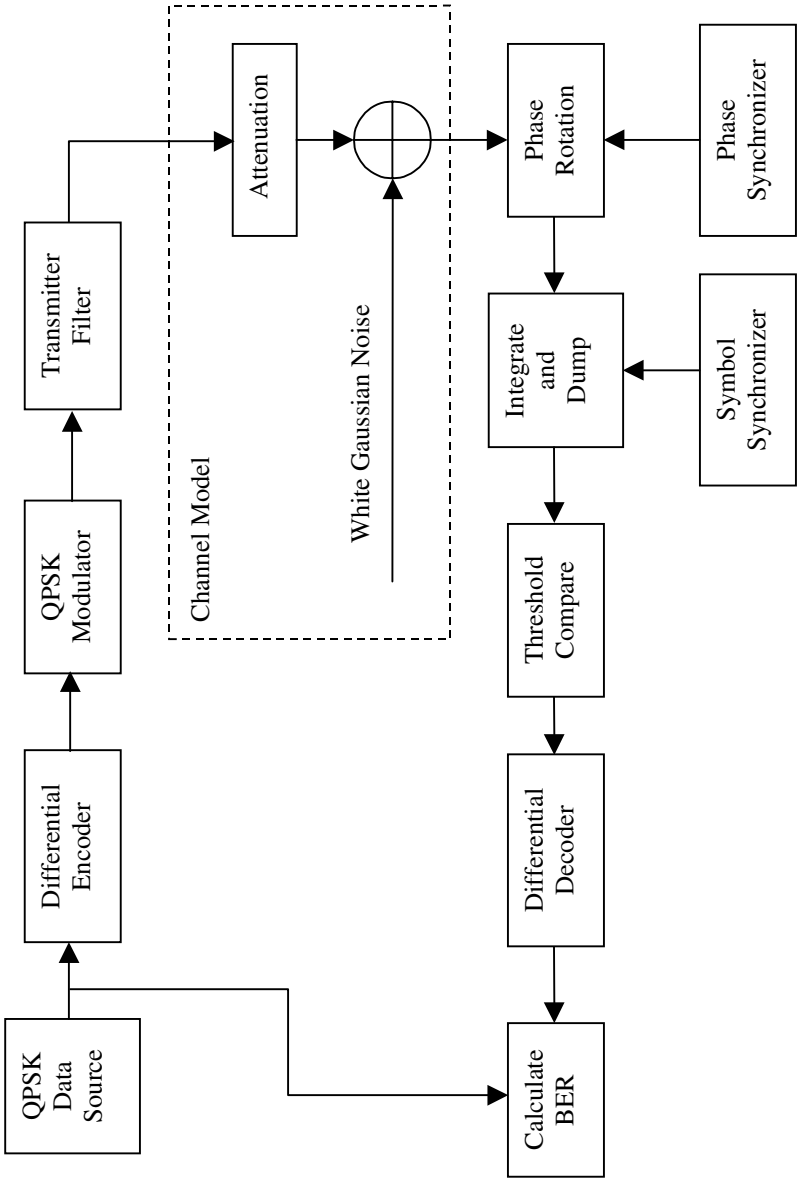


Figure 10.4 QPSK communications system.

```
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:

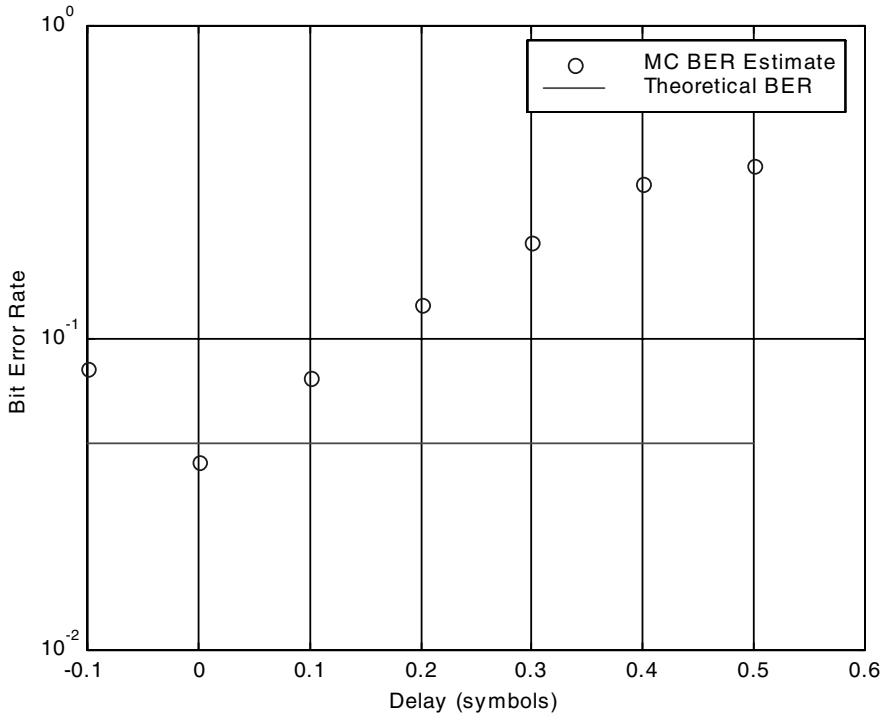


Figure 10.5 Preliminary simulation used to determine delay.

```
% 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.
```

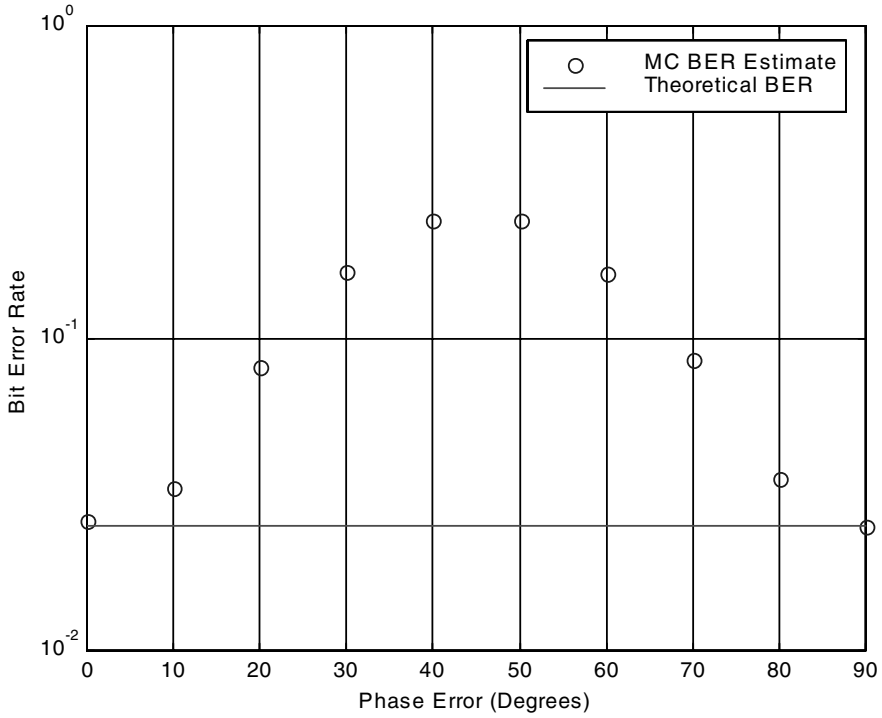


Figure 10.6 Sensitivity of BER to static phase errors.

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.

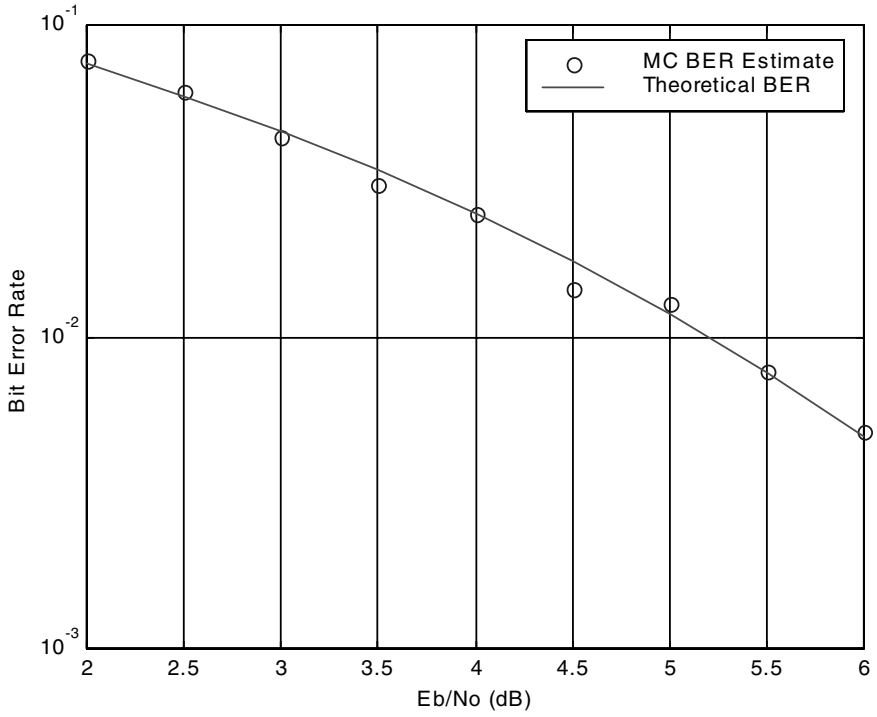


Figure 10.7 Simulation and theoretical results for a QPSK system operating in a AWGN environment.

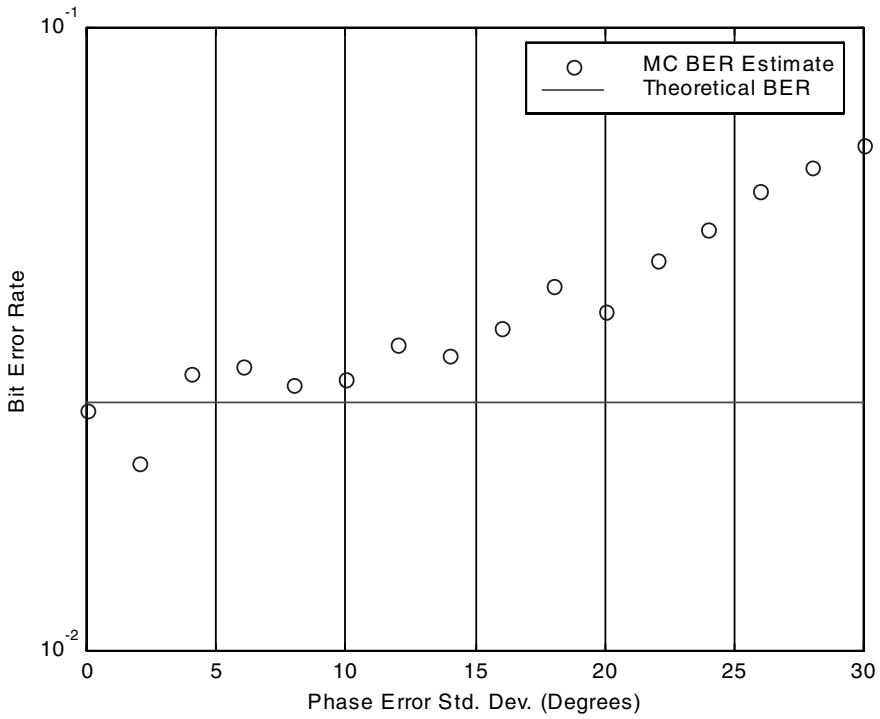


Figure 10.8 Simulation illustrating the sensitivity of QPSK to phase jitter.

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. ■

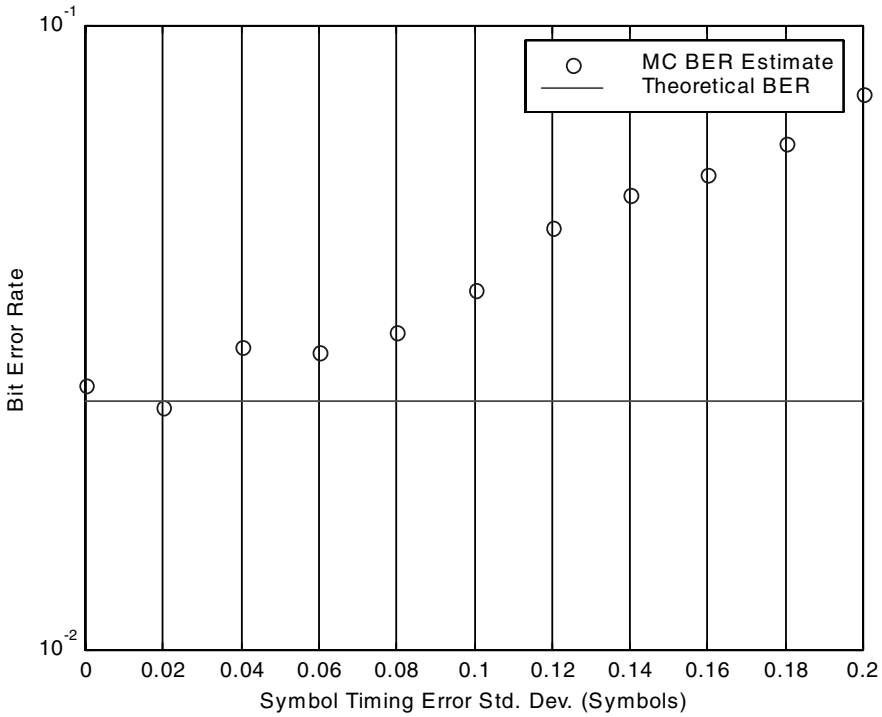


Figure 10.9 Simulation result illustrating the impact of symbol jitter.

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

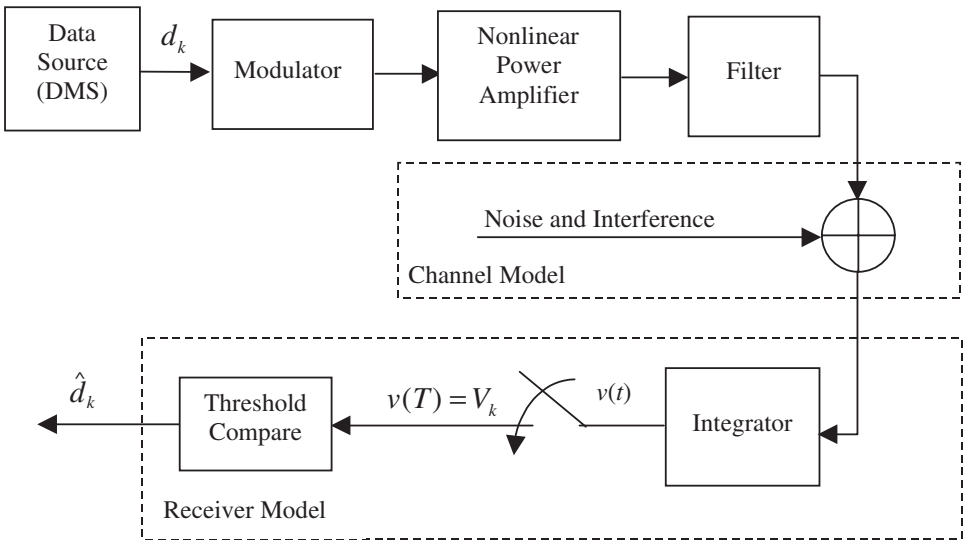


Figure 10.10 Example system to illustrate the SA simulation technique.

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

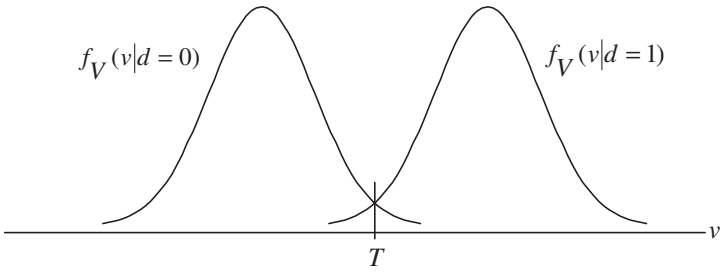
$$Q(x) = \frac{1}{\sqrt{2\pi}}\int_{x}^{\infty} \exp\left(-\frac{t^2}{2}\right) dt \tag{10.10}$$

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.

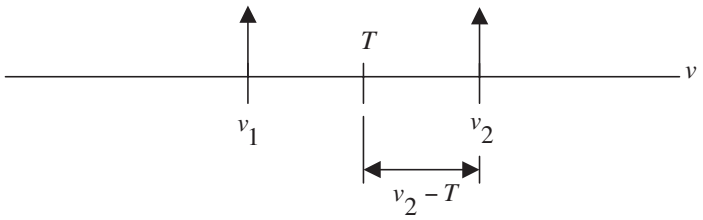
The value of v_1 can be determined by executing a noiseless simulation. If the channel noise is removed, the two conditional pfds shown in Figure 10.11(a) collapse to impulse functions ($\sigma_n = 0$) as shown in Figure 10.11(b). Each of these impulse functions has unity area, and the locations of the 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

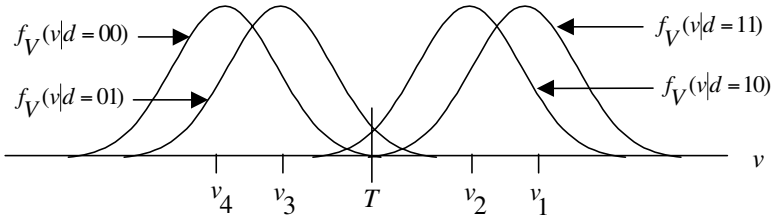
$$B_N = \int_0^\infty |H(f)|^2 \, df \tag{10.12}$$

and is the equivalent noise bandwidth of the receiver. It follows that the probability of error is given by

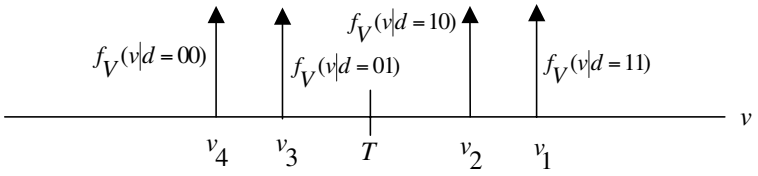
$$P_E = Q\left(\frac{v_1}{\sqrt{N_0 B_N}}\right) \tag{10.13}$$

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

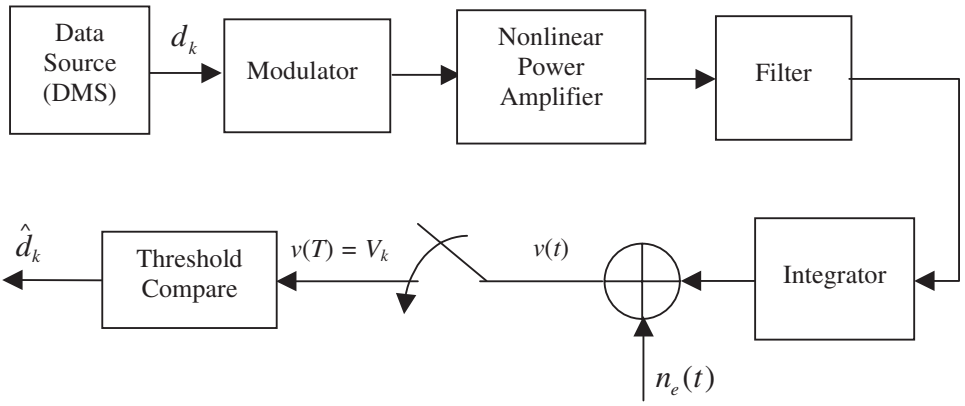


Figure 10.13 Equivalent noise source for semianalytic simulation.

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

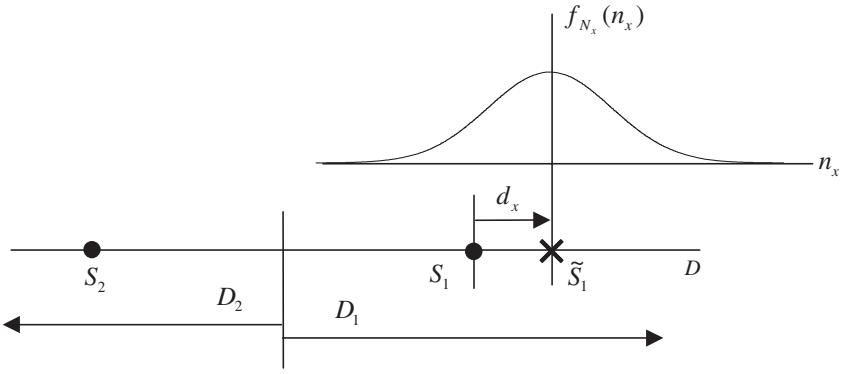


Figure 10.14 Semianalytic BER estimation for PSK.

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. ■

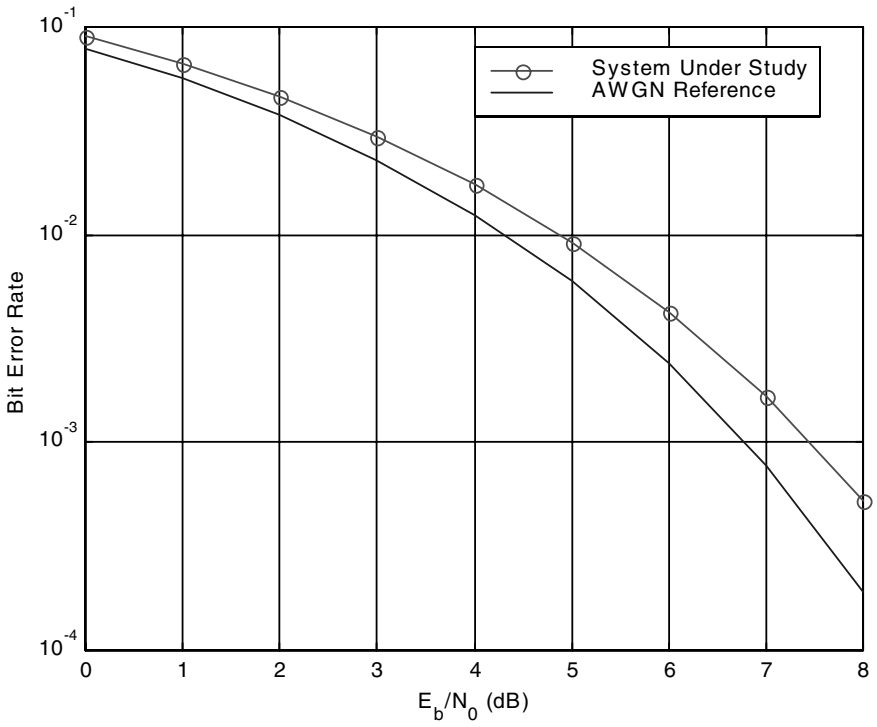


Figure 10.15 Semianalytic BER estimation for binary PSK.

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.

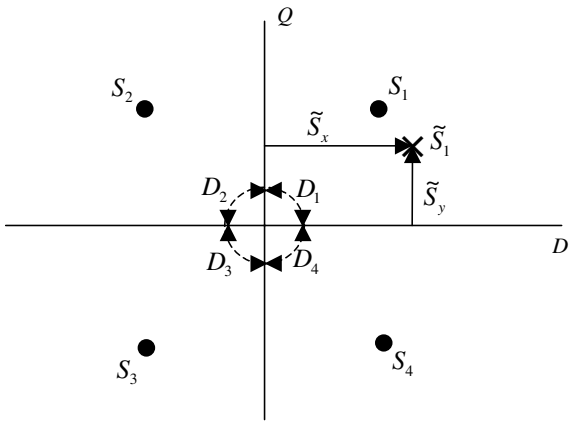


Figure 10.16 Semianalytic BER estimation for QPSK.

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. ■

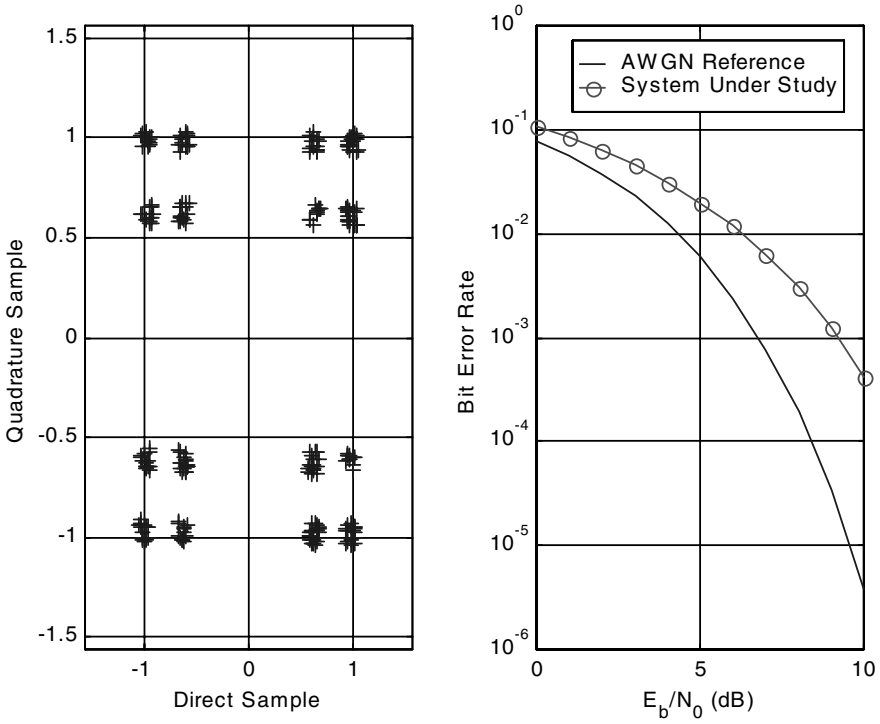


Figure 10.17 QPSK semianalytic simulation results.

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.
```