8

Communication System Models and Simulation in **MATLAB®**

The growing complexity of wireless communication systems imposes the use of computer simulations to evaluate system performance that leads to efficient system design and reduces the development time. MATLAB® has been one of the most effective and powerful software tools for simulation of wireless system performance. The efficient way that MATLAB® handles array and matrix manipulations provides system designers with a fast tool for the simulation of a complex communication systems. In addition, MATLAB® has a huge number of ready-to-use functions in its toolboxes that eases the implementation of complex system models.

In simulation of communication systems, MATLAB® has the capability of random signal generation that simulates wireless communication signals through its embedded random number generators and the functions that generate modulation formats. Through its RF and Communications toolboxes, MATLAB® represents an efficient tool for the design, modeling and simulation of nonlinear systems. On the other hand, MATLAB® optimization toolbox offers ready to use tools for nonlinear model optimization and model embedding in system simulations. Together, these tools make MATLAB® an optimum tool for the efficient simulation of nonlinearity in wireless communication systems. In addition, MATLAB® offers the capability of spectrum estimation and system metric simulation including BER simulations.

In this chapter, the basic MATLAB® tools used for the simulation of wireless communication systems are presented. These include deterministic and random signal generation, simulation of a wide variety of modulation formats, simulation of wireless communication standards, and the simulation of performance of a wireless system. In addition, Simulation of the performance of an overall communication system in Simulink® is presented. Simulink® tools are used as a verification platform for the concepts presented in this book.

8.1 Simulation of Communication Systems

Simulation of communication systems is usually done with the objective of estimating system performance by computing the response of a system to random process samples using a computer program. In general, a computer simulation is used to simulate signals that flow through a real system and to estimate its response. Therefore, a simulation process must include the generation of random process samples, and then using computer calculations to estimate the output of the system based on a given system model (Jeruchem *et al.*, 2000). Thus, a computer simulation involves three main processes that are random signal generation, a mathematical or empirical system model and the calculations of the system response using a computer algorithm. These processes are usually referred to as Monte Carlo (MC) simulations that are basically based on the simulation of a random phenomena without fully repeating the underlying physical experiment.

8.1.1 Random Signal Generation

The generation of samples of a random process is done using random number generation algorithms based on various models of random processes, see Appendix B. The efficiency of these algorithms in producing the statistical properties of a certain random process controls the accuracy of the overall simulation process. Since a random process can be viewed as a collection of random variables with certain probability distribution and correlation properties, the generation of a random process with a given probability distribution is based on random number generators. In many computer software platforms, the generation of random numbers with a given probability distribution is based on transformation of uniformly distributed random variables by a memoryless transformation. Independent uniformly distributed random numbers can be generated using simple recursive algorithms. To establish the correlation properties of the generated random process, a linear transformation with memory is used to produce the required correlation and spectral properties of a random process.

In simulation of random processes, the important property of *Ergodicity* is used to estimate the statistical properties of the generated random process. Ergodicity implies that the statistical properties of a random process such as expected value, variance, moments and correlation functions can be estimated from the time averages of the simulated process, which means that a single realization of a process is enough to fully characterize the process. Ergodicity does not have a rigorous mathematical basis but is rather based on practical observation of real communication signals. In computer simulations, the assumption that a process is ergodic greatly simplifies the models for random processes and hence, it represents the basis of computer simulation of communication systems.

8.1.2 System Models

System models in a simulation environment can either be generated mathematically (exact mathematical operation) or empirically (using block models derived from measured data). As discussed in Chapter 3, the choice of the model for a given phenomena depends on two main factors: firstly complexity that in turn affects the quality of the simulations and secondly accuracy that affects the capability of the simulations to faithfully reproduce system response. Unfortunately, these are conflicting requirements where increasing the level of details in a model results in increased computer calculations. However, increasing the level of detail of a given model enables finer design space parameters to be explored than is usually achieved by measurements or formula-based approaches to the evaluation of system performance. The expense of increased computer calculations is overcome by the advances in computer processing software and hardware technologies that improve day after day.

8.1.3 Baseband versus Passband Simulations

One important issue in the simulation of communication systems is the problem of simulating bandpass systems and signals, that is, systems that involve modulation of a baseband signal. Bandpass signals have a frequency spectrum that is concentrated around a carrier frequency. Similarly, a bandpass system is a system whose frequency response is limited to a frequency band that is centered around a carrier frequency. The very high carrier frequency of wireless communication systems implies that a very high sampling rate is needed when simulating such signals or systems since the sampling rate is commensurate with the carrier frequency by Nyquist theorem. Nyquist's theorem states that the sampling rate of a bandlimited signal or system must be equal to at least twice the bandwidth in order not to introduce aliasing distortion. Therefore, if a linear system to be simulated has a spectrum that spreads between $\pm B/2$, that is, has a bandwidth of B, then the sampling rate must be greater than or equal to B. This means that a large number of samples are needed to fully represent the signal in a simulation environment, which means a large number of computations.

In nonlinear systems, the situation is different because nonlinearity usually results in widening the spectrum of the input signal. For example, if a nonlinear system is represented by a polynomial model, the system results in an increase in the bandwidth of the signal by a factor N. Hence, in order to avoid aliasing, the sampling rate must be at least NB. A simple illustration is a square-law device $y(t) = x^2(t)$ that multiplies the signal by itself in the time domain. By Fourier transform properties, multiplication in the time domain is equivalent to convolution in the frequency domain, hence the output spectrum of the square law device is $X(f) * X(f)$ where $*$ indicates convolution. In this case, it can be shown that if $x(t)$ has a bandwidth B, then $Y(f) = X(f) * X(f)$ has a bandwidth of $2B$. In general, the contribution of higher-order nonlinearities to the output of the system become minor for large values of N. Therefore, a lower sampling rate than NB can be used if a certain aliasing error can be tolerated. The optimum sampling rate in this case can be obtained experimentally by running simulation of the spectrum at the output of nonlinearity at different sampling rates and then estimating the sampling rate that keeps the aliasing error at an acceptably low level (Jeruchem *et al.*, 2000).

The solution to the problem of high sampling rate requirements of the simulation is to use complex envelope simulations. As shown in Chapter 4, the complex envelope of a signal or the baseband equivalent of a system contains all the useful information, and hence a system can be simulated using only its baseband equivalent and the complex envelope of the input and output signals. In this case, the sampling rate is proportional to the bandwidth of the complex envelope and not to the carrier frequency, which means much lower computations than the case of simulating the signal at the passant.

8.2 Choosing the Sampling Rate in MATLAB® Simulations

The sampling rate is the rate at which the message signal is sampled during the simulation. Therefore, each signal simulated in MATLAB® must be associated with a sampling rate `Fd` that associates entries of the signal value with time increments of `1/Fd`. For example, for a signal taken from an alphabet of M values, the signal consists of integers in the range `[0, M-1]`. If the sampling rate is specified to be `Fd`, then the signal values represent samples taken at time increments `1/Fd`. In general, the sampling rate of the signal can be found from the knowledge of the time vector associated with the signal:

```
Fd = size(x,1) / (max(t)-min(t));
```

for a signal `x` sampled at times `t`.

Baseband digital modulation functions in MATLAB® do not have a built-in notion of time, hence, there is no need to include a sampling rate in the argument of functions that perform modulation. However, when the signal is to be plotted versus time, a time vector based on the sampling rate must be specified to associate signal values with time increments. For example, in M-ary modulation, the mapping process increases the sampling rate of the signal from `Fd` to `Fs`, whereas the demapping process decreases the sampling rate from `Fs` to `Fd`. If the baseband signal is a binary signal with sampling rate, say `Fd=8kHz`, then the output of an 8-PSK modulator would be a signal with sampling rate `Fs=1kHz`. That is, the modulated symbols are integers drawn from an alphabet `[0,1,...,7]` with time increments of 1 ms. The input and output of the modulator can be plotted vs. time using a time vector generated as

```
tx=[0:length(x)-1]*1/Fd
ty=[0:length(x)-1]*1/Fs
```

On the other hand, when designing a pulse-shaping filter, the sampling rates at the input and output of the filter must be specified since these filters perform up-sampling of data and hence the ratio of the sampling rate at the input and output of the filter `Fs/Fd` (which must be an integer) must be specified. One exception where the sampling rate needs to be specified in the argument of the modulation function, is nonlinear modulation functions (FSK and its variants) where individual values of `Fs` need to be specified.

8.3 Random Signal Generation in MATLAB®

Random signal generation in MATLAB® Communications toolbox is performed using a number of functions that are used to generate random sequences with given statistical properties. Random signal generation in MATLAB® is used to simulate either noise signal or signal sources, compute the error rates or generate scatter plots or eye diagrams of digital modulation formats.

8.3.1 White Gaussian Noise Generator

Perhaps the most common random signal is the White Gaussian Noise (WGN) that is used to model noise in communication systems. The *wgn* function generates either real or complex white Gaussian noise process with a specified power in dB, dBm or linear units. For example, the command below generates a complex white Gaussian noise column vector of length N whose power is K dBm in a load impedance of R Ohms.

```
x = wgn(N,1,K,R,'complex','powertype');
```

Another function is the *awgn* which can be used to add additive WGN (AWGN) to an input signal.

8.3.2 Random Matrices

The *randsrc* function generates random matrices whose entries are chosen independently from a specified alphabet A where each element of the alphabet is equally probable. For example, the command

```
x = randsrc(N,M,A)
```

generates an $N \times M$ matrix whose entries are chosen from an alphabet A. For example, to generate an Independent Identically Distributed (i.i.d.) sequence of bipolar data, the alphabet A is chosen as $A = [1 - 1]$.

If the elements of the alphabet are to be chosen with nonequal probabilities, then the vector that contains the alphabet is concatenated with the probabilities of the elements:

```
x = randsrc(N,M,[A; P])
```

For example, the following command produces an 3×4 random matrix with elements chosen from an alphabet [123] with probabilities [0.1 0.3 0.6]

```
x = randsrc(3,4,[1,3,5; .1,.3,.6])
```

8.3.3 Random Integer Matrices

The *randint* function generates a matrix of random integers chosen from a range of values. For example, the command

```
x = randint(N,M,[a,b])
```

generates an $N \times M$ matrix whose entries are chosen from the values between a and b. A special case is the generation of random binary matrices when $a = 0$ and $b = 1$.

The `randerr` function generates binary matrices with a specified number of zeros and ones and is meant for testing error-control coding. The command

```
x = randerr(N,M,[0,1,2,...; p0,p1,p2,...])
```

generates an $N \times M$ binary matrix having the property that each row contains zero ‘1’s with probability p_0 or one ‘1’ with probability p_1, etc. For example, the command

```
x = randerr(5,4,[1,2; 0.4,0.6])
```

produces a 5-by-4 matrix where each row has one ‘1’ with probability 0.4 or two ‘1’s with probability 0.6.

8.4 Pulse-Shaping Filters

Pulse-shaping filters can be constructed using `fdesign.pulseshaping` specification objects from the Signal Processing Toolbox (The Signal Processing Toolbox, 2009) where the properties of the filter such as the shape, order, stop-band attenuation, roll-off factor and frequency response can be specified. The command

```
d = fdesign.pulseshaping(sps,shape,...spec,value1,value2,...,fs,
                              magunits)                            (8.1)
```

constructs the filter `d` which has specifications as shown in Table 8.1.

The command

```
H = design(d);
```

produces the frequency response/impulse response of the filter. The frequency response can be plotted using the command

```
fvtool(H)
```

and the impulse response can be plotted using the command:

```
fvtool(H, 'impulse').
```

8.4.1 Raised Cosine Filters

Pulse-shaping filters can be constructed using `fdesign.pulseshaping` specification objects by setting the "shape" property to 'Raised Cosine'. The values (`value1, value2,...`) in Equation (8.1) define the values of the string `spec` and are defined as follows:

- `Ast`: stopband attenuation (in dB).
- `Beta`: roll-off factor expressed as a real-valued scalar ranging from 0 to 1.
- `Nsym`: filter order in symbols.
- `N`: filter order (must be even).
- The length of the impulse response is $N + 1$.

The string `magunits` specifies the units for magnitude specification of the input arguments that take one of the following entries:

- linear: specify the magnitude in linear units;
- dB: specify the magnitude in dB (decibels);
- squared: specify the magnitude in power units.

Table 8.1 Pulse shaping

String	Description
sps	oversampling
shape	'Raised Cosine', 'Square Root Raised Cosine', 'Gaussian'
spec	'Ast,Beta' 'Nsym,Beta', 'Nsym,BT', 'N,Beta'
fs	sampling frequency of the signal to be filtered (Hz)
magunits	units for magnitude specification of the input arguments

When the `magunits` argument is omitted, all magnitudes are assumed to be in decibels.
For example, the command:

```
d = fdesign.pulseshaping(8,'Raised Cosine,'Ast,Beta',30,0.3)
H = design(d);
fvtool(H)
```

produces a pulse shaping filter with over sampling of 8, stop-band attenuation of 30 dB and a roll-off factor of 0.3. Figure 8.1 shows the impulse response and the frequency response of the resulting pulse-shaping filter.

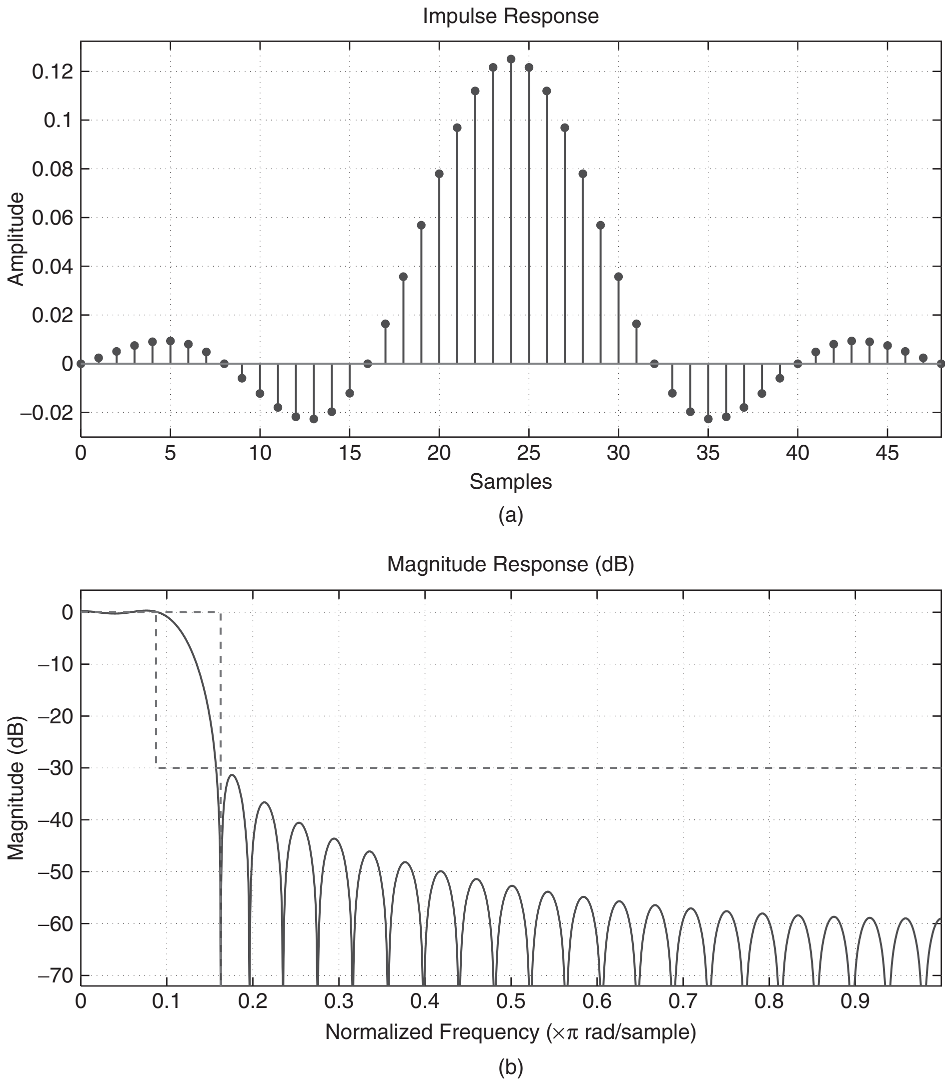


Figure 8.1 Raised cosine filter; (a) impulse response and (b) frequency response.

Digital data can also be filtered by raised cosine pulse-shaping filters using the function `rcosflt` that applies a raised cosine filter with specified characteristic to input data. The command

```
y=rcosflt(x,Fd,Fs,type,rolloff,delay);
```

filters the digital data x whose sampling rate is `Fd` by a raised cosine filter and produces filtered data y whose sampling rate is `Fs`. The command up samples the input data by a factor `Fs/Fd` before filtering where the ratio must be an integer. The function up samples the input data by inserting `Fs/Fd-1` zeros between input data samples. The up sampled data consists of `Fs/Fd` samples per symbol and has sampling rate `Fs`.

The string `type` specifies the type of the RC filter. The available types of the raised cosine filter are 'fir', 'fir/sqrt' that produces FIR square-root RC filter, 'iir' that produces an IIR raised cosine filter and 'iir/sqrt' that produces an IIR square-root RC filter.

The value `rolloff` specifies the roll-off factor of the filter that determines the bandwidth of the filter. For example, a roll-off factor of r means that the bandwidth of the filter is $1 + r$ times the input sampling frequency, `Fd`. This also means that the transition band of the filter extends from r`Fd` to $(1 + r)$`Fd`.

The value `delay` specifies the group delay taps of the filter that defines the time between the filter's initial response and its peak response. The group delay influences the size of the output, as well as the order of the filter.

Another way of filtering the data by an RC filter is to first specify the filter transfer function using the `rcosine` function and then applying `rcosflt` to the input data. For example

```
[num,den] = rcosine(Fd,Fs,'iir',rolloff,delay);
y = rcosflt(x,Fd,Fs,'iir/filter',num,den,delay);
```

where `[num,den]` contain the filter transfer function coefficients in ascending order of powers of z^{-1}.

8.4.2 Gaussian Filters

A Gaussian filter can be constructed if the shape property in Equation (8.1) is specified as 'Gaussian'. In this case, the entries for *spec* are chosen as `'Nsym,BT'` where the string entries are defined as follows:

- `Nsym`: filter order in symbols.
- `BT`: the 3 dB bandwidth–symbol time product.

For example the following command:

```
d = fdesign.pulseshaping(8,'Gaussian','Nsym,BT',4,0.2)
H = design(d);
fvtool(H)
```

produces a Gaussian filter with oversampling = 8, filter order of 4 symbols and $BT = 0.2$. Figure 8.2 shows the frequency response and the impulse response of the designed Gaussian pulse-shaping filter.

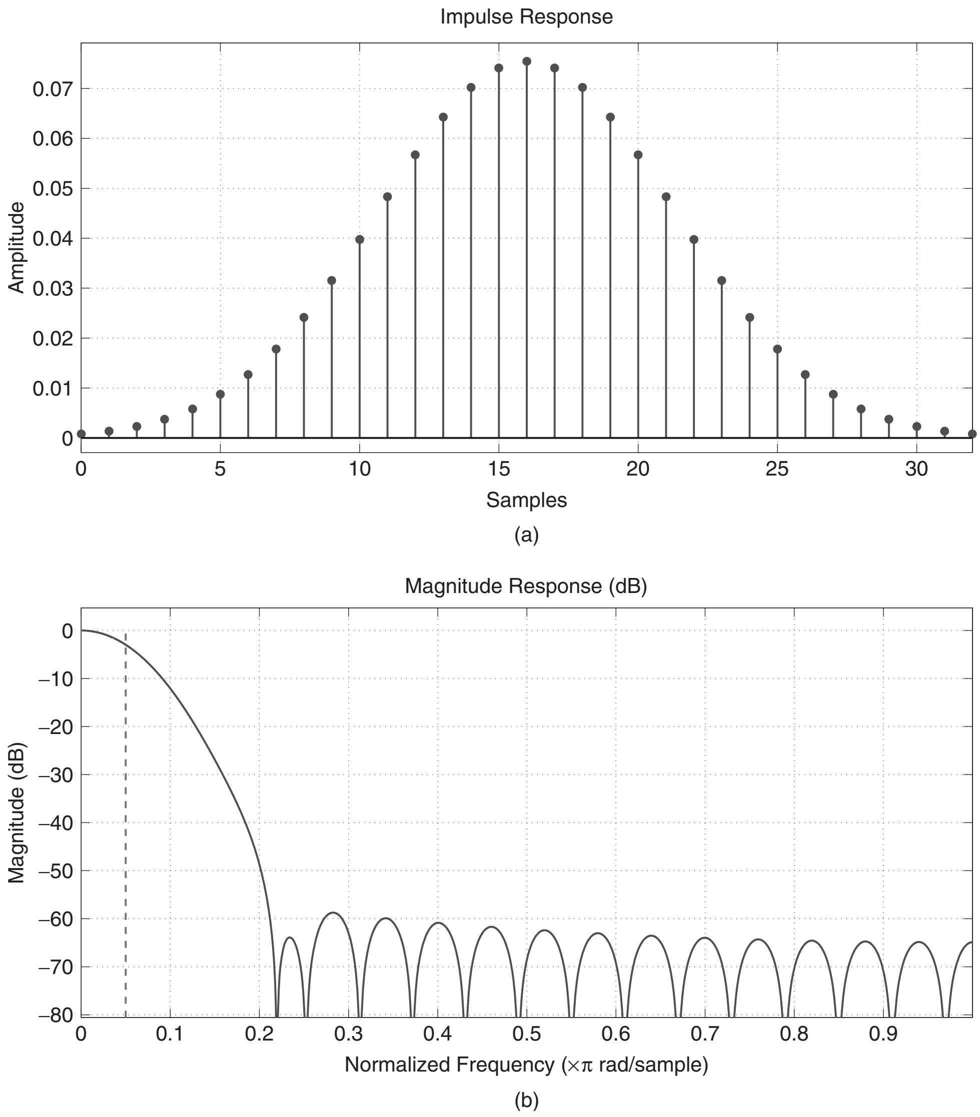


Figure 8.2 Gaussian pulse-shaping filter; (a) impulse response and (b) frequency response.

8.5 Error Detection and Correction

Error detection and correction coding is used to combat errors that result from channel impairments. MATLAB® has a number of functions and objects for generation of coding techniques to perform encoding and decoding of data. Table 8.2 lists the main functions used in implementing Block Codes, Convolutional Codes and Cyclic Redundancy Check codes. Since coding techniques are out of the scope of this book, the reader is referred to MATLAB® documentation (The Communication Toolbox, 2009) for more information.

Table 8.2 Block coding techniques: toolbox functions and objects

Code Type	Functions
Linear	block, encode, decode, gen2par, syndtable
Cyclic	encode, decode, cyclpoly, cyclgen, gen2par, syndtable
BCH	bchenc, bchdec, bchgenpoly
LDPC	fec.ldpcenc, fec.ldpcdec
Hamming	encode, decode, hammgen, gen2par,syndtable
Reed–Solomon	rsenc, rsdec, rsgenpoly, rsencof,rsdecof
Convolutional Codes	convenc vitdec, poly2trellis

8.6 Digital Modulation in MATLAB®

Baseband simulation of digital modulation formats in MATLAB® is based on producing the complex envelope of the output of a modulator given the complex envelope of a message signal. The complex envelope of the modulated signal can be related to the passband output of the modulator using Equation (4.8). M-ary modulation of a digital signal is implemented in two steps: firstly; the input binary bit stream is mapped into a symbol stream where each symbol represents an n-tuple of bits and takes the values [0, 1, .., $M - 1$], secondly; the resulting symbol stream is then modulated using one of the modulation schemes such as Amplitude Shift Keying (ASK), Phase Shift Keying (PSK) or Frequency Shift Keying (FSK) and their variants.

8.6.1 Linear Modulation

Baseband simulation of linear digital modulation formats such as PSK, DPSK, QPSK, OQPSK, PAM and QAM is done using the *modem* objects. A `modem` object is a MATLAB® variable that contains information about the modulation class, M-ary number, and the constellation mapping (The Communication Toolbox, 2009). The `modem` object takes a symbol stream as input and produces the modulation object. The modulated signal can be generated using the `modulate` function that operates on the `modem` variable. Table 8.3 shows the modem object of different modulation formats.

Table 8.3 The modem objects used to generate different modulation formats (The Communication Toolbox, 2009)

Modulation Type	Modem Object
DPSK	modem.dpskmod and modem.dpskdemod
General QAM	modem.genqammod and modem.genqamdemod
MSK	modem.mskmod and modem.mskdemod
OQPSK	modem.oqpskmod and modem.oqpskdemod
PAM	modem.pammod and modem.pamdemod
PSK	modem.pskmod and modem.pskdemod
QAM	modem.qammod and modem.qamdemod

Table 8.4 Argument description of the modem object (The Communication Toolbox, 2009)

Argument	Description
M	M-ary value.
PhaseOffset	Phase offset of ideal signal constellation in radians.
Constellation	Ideal signal constellation.
SymbolOrder	Type of mapping employed for mapping symbols to ideal constellation points. The choices are: -‘binary’: for binary mapping -‘gray’: for gray mapping -‘user-defined’: for custom mapping
SymbolMapping	Symbol mapping is a list of integer values from 0 to $M-1$ that correspond to ideal constellation points.
InputType	Type of input to be processed by QAM modulator object. The choices are: -‘bit’: for bit/binary input -‘integer’: for integer/symbol input

The modem object of any of the modulation formats in Table 8.3 can be constructed as follows:

```
H = modem.xxxmod('M', value, 'PHASEOFFSET', value,...
 'SYMBOLORDER', value,'INPUTTYPE', value)
```

where `xxx` takes one of the values shown in Table 8.3 and defines the modulation type. The variables in the argument of the modem object are defined as shown in Table 8.4.

The command

```
 y = modulate(H, x)
```

where `H` is the handle to a modulator object and `x` is a message signal, produces the complex envelope of the modulated signal.

As an example, the following script produces the complex envelope of a 16QAM modulated signal. The script starts by generating a random bit stream, maps the bit stream to 4-bit symbols using the `bi2de` function and then modulates the resulting symbol stream using 16 QAM with rectangular pulse shaping.

```
%File Name: QAM.m
%% Setup
% Define parameters.
M = 16; % Size of signal constellation
k = log2(M); % Number of bits per symbol
n = 10000; % Number of bits to process
nsamp = 4; % Oversampling rate
hMod = modem.qammod(M); % Create a 32-QAM modulator
%% Signal Source
% Create a binary data stream as a column vector.
```

```
x = randint(n,1); % Random binary data stream
% Plot first 40 bits in a stem plot.
stem(x(1:40),'filled');
title('Random Bits');
xlabel('Bit Index'); ylabel('Binary Value');
%% Bit-to-Symbol Mapping
% Convert the bits in x into k-bit symbols.
xsym = bi2de(reshape(x,k,length(x)/k).','left-msb');
%% Stem Plot of Symbols
% Plot first 10 symbols in a stem plot.
figure; % Create new figure window.
stem(xsym(1:10));
title('Random Symbols');
xlabel('Symbol Index'); ylabel('Integer Value');
%% Modulation
hMod=modem.qammod(M);
y = modulate(hMod,xsym); % Modulate using 32-QAM.
%% Follow with rectangular pulse shaping.
ypulse = rectpulse(y,nsamp);
```

Figure 8.3 shows the bit/symbol mapping and the resulting 16QAM modulated waveform.

8.6.2 Nonlinear Modulation

For FSK and MSK modulation formats, the function `FSKMOD` can be used. The command

```
    y = FSKMOD(x,M,FREQ_SEP,NSAMP,Fs,PHASE_CONT,SYMBOL_ORDER);
```

produces the complex envelope of an M-ary FSK modulated signal given a message signal `x` where

- `M`: is the alphabet size and must be an integer power of two.
- `x`: is the message signal that must consist of integers between 0 and $M-1$.
- `FREQ_SEP`: is the desired separation between successive frequencies, in Hz.
- `NSAMP`: denotes the number of samples per symbol and must be an integer greater than 1.
- `Fs`: specifies the sampling frequency (Hz). The default sampling frequency is 1.
- `PHASE_CONT`: specifies the phase continuity across FSK symbols. `PHASE_CONT` can be either `'cont'` for continuous phase, or `'discont'` for discontinuous phase. The default is `'cont'`.
- `SYMBOL_ORDER`: specifies how the function assigns binary words to corresponding integers. If `SYMBOL_ORDER` is set to `'bin'` (default), then the function uses a natural binary-coded ordering. If `SYMBOL_ORDER` is set to `'gray'`, then the function uses a gray-coded ordering.

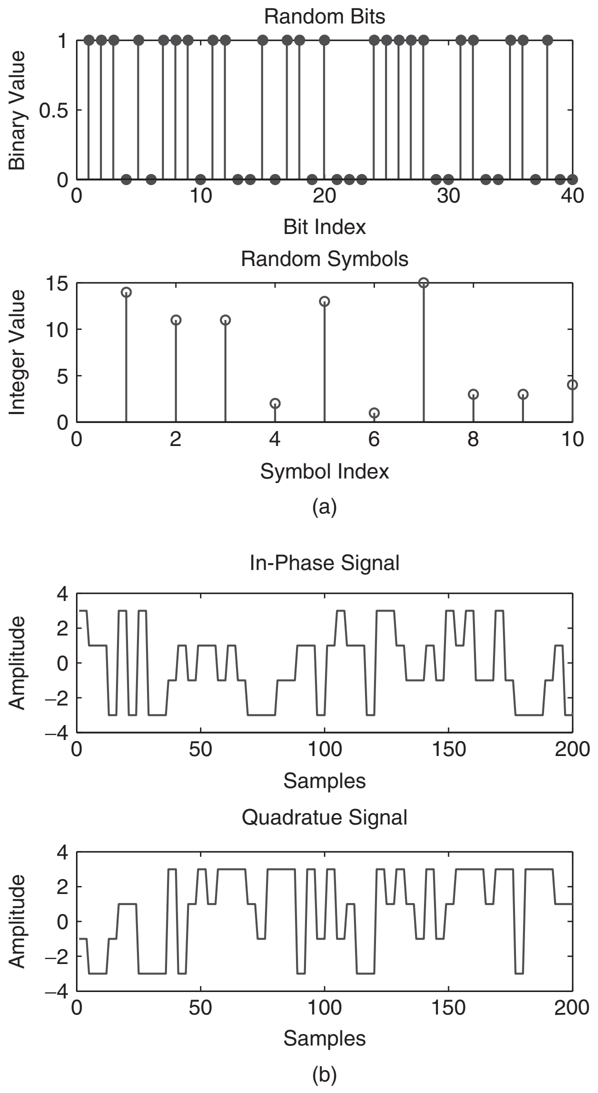


Figure 8.3 16-QAM modulated signal; (a) bit and symbol mapping and (b) modulated waveform.

For example, the following code generates a 16-ary FSK signal for an input binary signal `x`:

```
File Name: FSK.m
%% Setup
% Define parameters.
M = 16; % Size of signal constellation
```

```
k = log2(M); % Number of bits per symbol
n = 1000; % Number of bits to process
nsamp = 4; % Oversampling rate
freq_sep=.2;
Fs=1; %Sampling rate
%% Signal Source
% Create a binary data stream as a column vector.
x = randint(n,1); % Random binary data stream
%% Bit-to-Symbol Mapping
% Convert the bits in x into k-bit symbols.
xsym = bi2de(reshape(x,k,length(x)/k).','left-msb');
% Generation of a continuous phase 16-ary FSK with gray coding
y = FSKMOD(xsym,M,freq_sep,nsamp,Fs,'cont','gray')
```

Figure 8.4 shows the resulting 16-FSK modulated waveform.

For MSK modulation, the modem object `modem.mskmod` can be used as shown in the previous section (Table 8.3).

8.7 Channel Models in MATLAB®

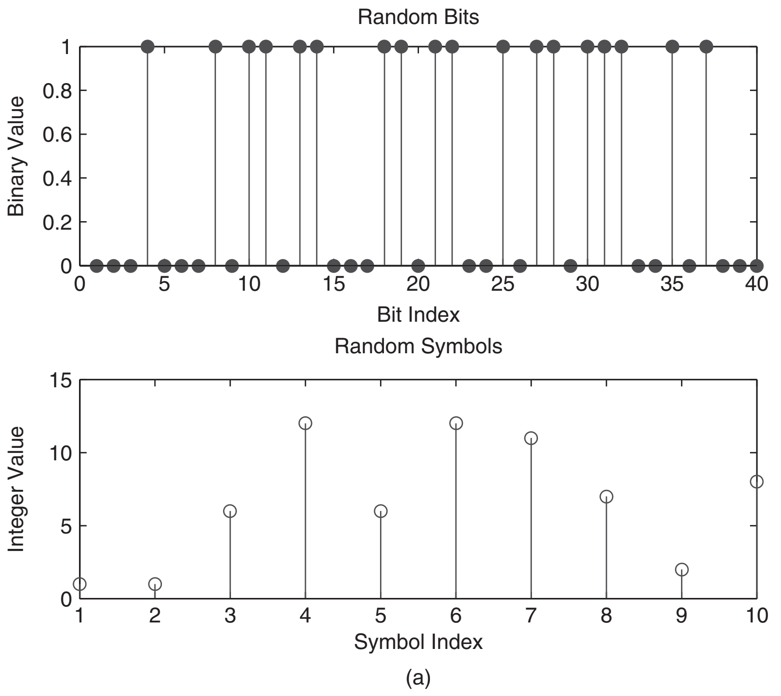
Communication channels introduce impairments to the transmitted signal that cause data errors at the receiver. Different channel models can be assumed when simulating system performance in MATLAB®. These include additive white Gaussian noise channels, fading channels including many models such as Raleigh and Rician fading channel models, MIMO channels and binary symmetrical channels. Each of these models is based on mathematical description of the channel, which is based on the properties of the channel and is dealt with differently in MATLAB®."

Table 8.5 lists the main functions used in implementing different channel types. Since, the main objective of this book is to study the system performance under nonlinear channels and not the above channel models, these models will not be discussed in details. MATLAB® models used to model the degradation of system performance due to nonlinear amplification are discussed in details in the next chapter.

8.8 Simulation of System Performance in MATLAB®

Simulation of communication system performance is usually based on calculating the error rates which result from corruption of a random sequence of data processed by a channel model, by additive noise. Other performance metrics that can be simulated include the eye diagrams, scatter plots, EVM and ACPR, which provide different views for system performance and signal quality at the receiver.

This section provides the necessary tools to simulate these metrics in MATLAB® using different functions such as `biterr`, `symerr eyediagram` and `scatterplot`. These functions are designed to estimate system performance of different wireless standards under various channel models.



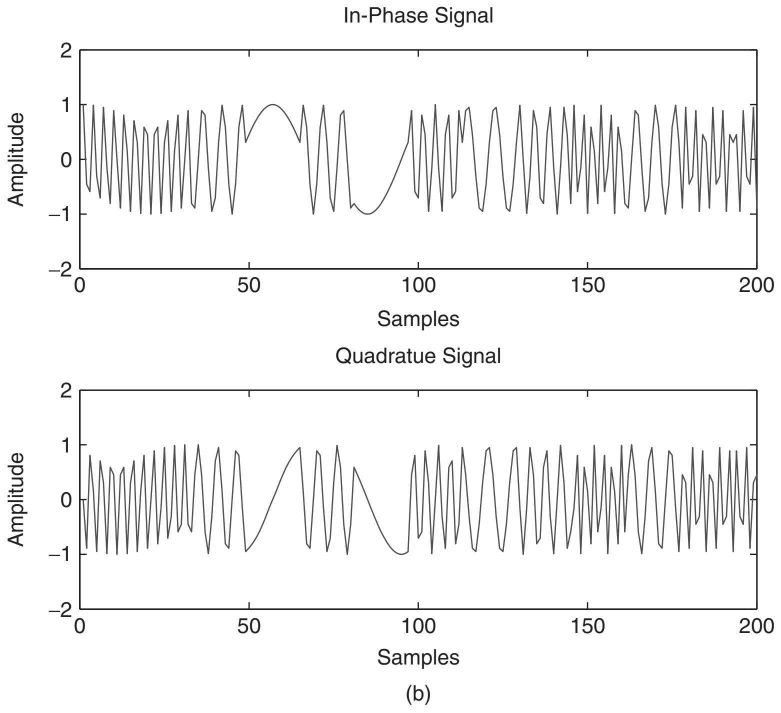


Figure 8.4 16-FSK modulated signal; (a) bit and symbol mapping and (b) modulated waveform.

Table 8.5 Main functions used in implementing different channel types

Channel Type	Function
Additive White Gaussian Noise (AWGN) channel	awgn
Multiple-Input Multiple-Output (MIMO) channel	mimochan
Fading channel	rayleighchan, ricianchan, doppler
Binary symmetric channel	bsc

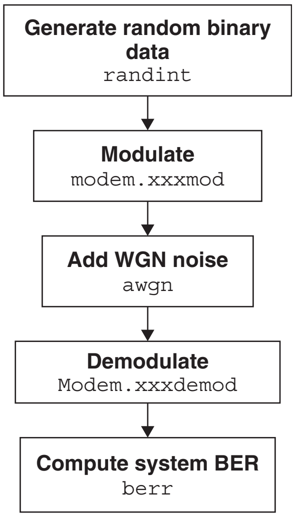


Figure 8.5 Simulation of BER in MATLAB® (The Communication Toolbox, 2009; The Communication Blockset, 2009).

8.8.1 BER

Simulation of BER of a communication system using MC simulations in MATLAB® is based on simulating the transmission of random data through a channel model and then comparing the resulting data with the data before transmission. The channel model is usually assumed to be an AWGN channel where white Gaussian noise is artificially added to the transmitted signal while the other parts of the communication system (transmitter and receiver) are simulated using different system specifications as shown in the previous sections. Figure 8.5 shows a flowchart for the simulation of BER in MATLAB® (The Communication Blockset, 2009).

The functions `biterr` and `symerr` compute the BER and SER by comparing the data entering the transmitter (the original data (bit or symbol levels)) to the data at the output of the receiver. An error is counted if there is a discrepancy between the corresponding bits in the two sequences of bits. The bit or symbol error rate is the number of errors divided by the total number (of bits or symbols) transmitted.

The following script simulates BER of the 16PSK modulation with rectangular pulse shaping. Figure 8.6 shows the result of the simulations.

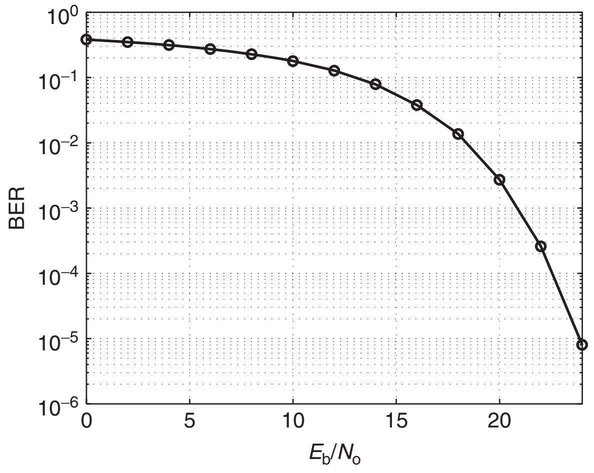
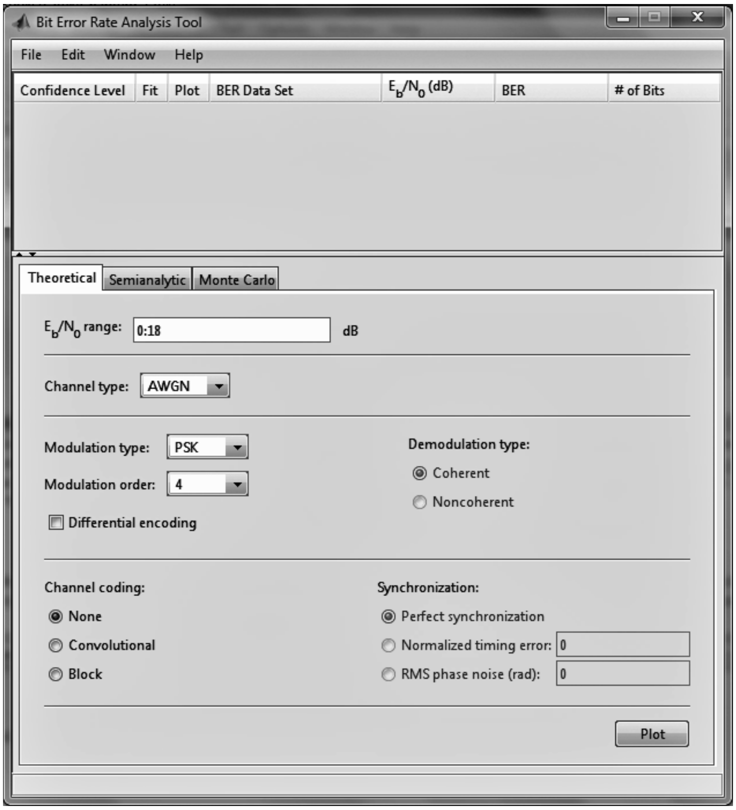


Figure 8.6 Simulated BER of a 16 PSK signal with rectangular pulse shaping.

```
File Name: BER_Example
clear all
%% Setup
% Define parameters.
M = 16; % Size of signal constellation
k = log2(M); % Number of bits per symbol
n = 1e6; % Number of bits to process
nsamp = 4; % Oversampling rate
hMod = modem.pskmod(M); % Create a 16PSK modulator
EbNo=0:2:24; % Range of EbNo
for p=1:length(EbNo)
%% Signal Source
% Create a binary data stream as a column vector.
txbits = randint(n,1); % Random binary data stream
% Plot first 40 bits in a stem plot.
% stem(x(1:40),'filled');
% title('Random Bits');
% xlabel('Bit Index'); ylabel('Binary Value');
%% Bit-to-Symbol Mapping
% Convert the txbits into k-bit symbols.
txsym = bi2de(reshape(txbits,k,length(txbits)/k).','left-msb');
%% Modulation
y = modulate(hMod,txsym); % Modulate using 16PSK.
% Apply rectangular pulse shaping.
ypulse = rectpulse(y,nsamp);
% Transmitted Signal
ytx = ypulse;
%% AWGN Channel
% Apply AWGN channel.
SNR = EbNo(p) + 10*log10(k) - 10*log10(nsamp);
```

```
ynoisy = awgn(ytx,SNR,'measured');
% Received Signal
yrx=downsample(ynoisy,nsamp);
%% Demodulation
% Demodulate signal using 16PSK.
rxsym = demodulate(modem.pskdemod(M),yrx);
% Symbol-to-Bit Mapping
% Undo bit-to-symbol mapping
rxbits = de2bi(rxsym,'left-msb'); % Convert integers to bits.
% Convert rxbits to a vector
rxbits = reshape(rxbits.',numel(rxbits),1);
% BER Computation
% Compare txbits and rxbits to obtain the number of errors and
% the bit error rate.
[NumErrors,BER(p)] = biterr(txbits,rxbits)
% the symbol error rate.
[NumErrors,SER(p)] = symerr(txsym,rxsym);
end
semilogy(EbNo,BER,'o-')
```


Bit Error Rate Analysis Tool
File Edit Window Help
Confidence Level | Fit | Plot | BER Data Set | E_b/N_0 (dB) | BER | # of Bits
Theoretical | Semianalytic | Monte Carlo
Plot

E_b/N_0 range: 0:18 dB

Channel type: AWGN

Modulation type: PSK

Modulation order: 4

☐ Differential encoding

Demodulation type: [✓ Coherent] Noncoherent

Channel coding: [✓ None] Convolutional Block

Synchronization: [✓ Perfect synchronization] Normalized timing error: 0 RMS phase noise (rad): 0

Figure 8.7 BERtool GUI (The Communication Toolbox, 2009).

BER can also be simulated using the Bit Error Rate Analysis Tool (BERTool) in the Communications Toolbox. BERTool is a Graphical User Interface (GUI) that enables BER performance of a communication system to be analyzed using theoretical, semi analytic, or Monte Carlo simulation-based approach. The command `bertool` launches the GUI of the BERTool in MATLAB®. Figure 8.7 shows the BERTool GUI where the type of BER simulations can be chosen from the tabs "Theoretical", "Semi-analytic", or "Monte Carlo".

The BERtool provides simulation of BER vs. E_b/N_o of various modulation schemes under various channel models, types of demodulation, coding schemes and types of synchronization. The theoretical BER is calculated using a set of theoretical BER data for systems that use an AWGN channel. Monte Carlo simulations are done based on Simulink® models or MATLAB® functions that can be called through the GUI. These models can be modified to include specifications of a given communication system model. The semi analytic technique uses a combination of simulation and analytical models to determine the error rate of a communication system. The semi analytic technique is based on quasi-analytic simulation methods where some a priori knowledge (or assumptions) that allows the construction of an abstraction (that can be dealt with analytically) are implied (Jeruchem *et al.*, 2000).

The following code is used by the *Monte Carlo Simulation* as a simulation function in the BERTool (The Communication Toolbox, 2009). Figure 8.8 shows the results of MC simulations compared to theoretical BER.

```
% File Name: bertool_simfcn.m
function [ber, numBits] = bertool_simfcn(EbNo,...
maxNumErrs, maxNumBits)
% Import Java class for BERTool.
import com.mathworks.toolbox.comm.BERTool;
% Initialize variables related to exit criteria.
totErr = 0;  % Number of errors observed
numBits = 0; % Number of bits processed
% - Set up parameters. - % Set up initial parameters.
% Define parameters.
M = 16; % Size of signal constellation
k = log2(M); % Number of bits per symbol
siglen = 10000; % Number of bits to process
hMod = modem.qammod(M); % Create a 32-QAM modulator
hDemod = modem.qamdemod(hMod); % Create a DPSK
ntrials = 0; % Number of passes through the loop
% Simulate until number of errors exceeds maxNumErrs
% or number of bits processed exceeds maxNumBits.
while((totErr < maxNumErrs)  && (numBits < maxNumBits))
   % Check if the user clicked the Stop button of BERTool.
   if (BERTool.getSimulationStop)
      break;
   end
% - Proceed with simulation.
% Modulate the message signal
% Create a binary data stream as a column vector.
msg = randint(siglen, 1); % Generate message sequence.
% msg =randint(siglen, 1, 2, 9973);
% Bit-to-Symbol Mapping
% Convert the bits in msg into k-bit symbols.
xsym = bi2de(reshape(msg,k,length(msg)/k).','left-msb');
% Modulation-Transmitted Signal
txsig = modulate(hMod,xsym); % Modulate using 32-QAM.
% Channel
% Send signal over an AWGN channel.
snr = EbNo+10*log10(k);
txsignoisy = awgn(txsig,snr,'measured');
% Received Signal
rxsig=txsignoisy;
% Demodulate signal using 16-QAM.
decodmsg = demodulate(hDemod, rxsig); % Demodulate.
% Undo the bit-to-symbol mapping performed earlier.
zsym = de2bi(decodmsg,'left-msb'); % Convert integers to bits.
% Convert z from a matrix to a vector.
zbit = reshape(zsym.',numel(zsym),1);
newerrs = biterr(msg,zbit); % Errors in this trial
ntrials = ntrials + 1; % Update trial index.
% Update the total number of errors.
totErr = totErr + newerrs;
% Update the total number of bits processed.
numBits = ntrials * siglen;
   % - Be sure to update totErr and numBits.
   % - INSERT YOUR CODE HERE.
end % End of loop
% Compute the BER.
ber = totErr/numBits
```

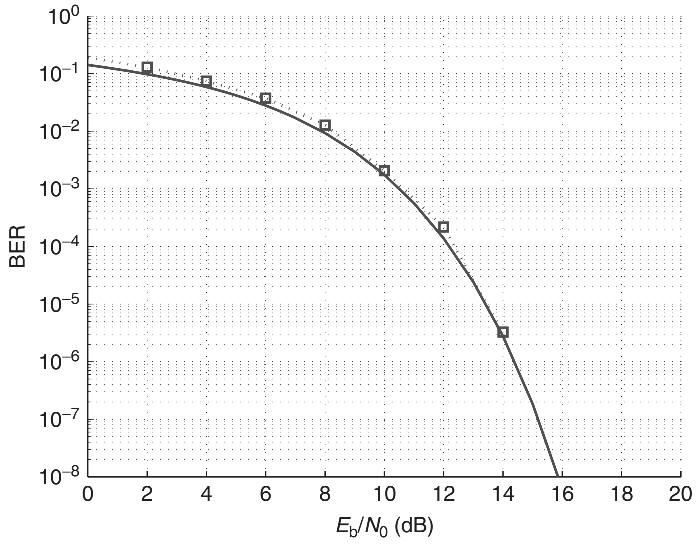


Figure 8.8 BER of 16 QAM system obtained from BERtool; solid: theoretical and □: simulated.

8.8.2 Scatter Plots

The function `scatterplot` plots the signal constellation of the transmitted or the received signals where the in-phase vs. the quadrature components of the modulated signals are plotted. To plot the signal constellation of a modulated signal generated using the `modem.xxxmod` object, the `Constellation` property of the object can be used. For example, the following script:

```
% File Name: ScatPlot.m
% Number of points in constellation
M = 16;
% Modulator object
hMod=modem.pskmod(M);
pt = hMod.Constellation; % Vector of points in constellation
% Plot the constellation.
scatterplot(pt);
```

retrieves the points in a 16 PSK modulated signal constellation and plots the signal constellation using `scatterplot`. Figure 8.9 shows the resulting constellation diagram.

A scatter plot can also be generated using the `commscope` object. For example, the command:

```
hScope = commscope.ScatterPlot
update(h,x)
```

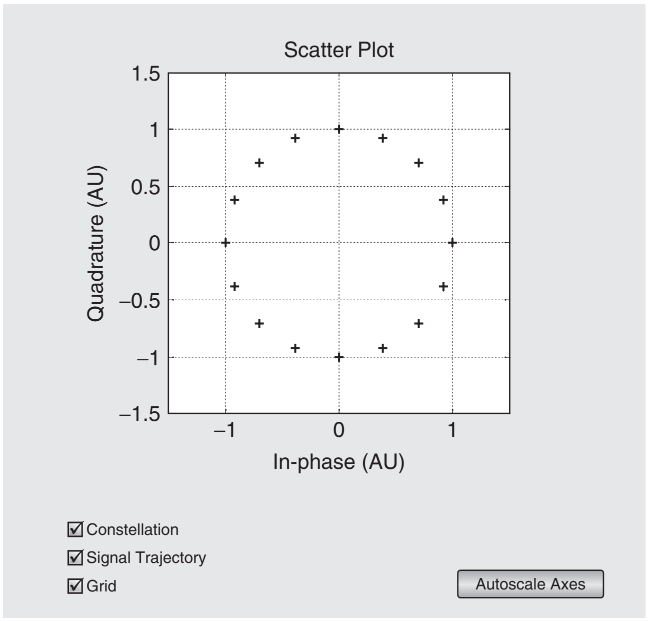


Figure 8.9 Scatter plot of a 16 PSK modulated signal with RRC pulse shaping.

produces a constellation diagram of the transmitted signal `x`. The following script produces the constellation diagram of a 16PSK signal with RC pulse shaping using the `commscope` object.

```
% File Name: ScatPlot_CommScope.m
% Number of points in constellation
M = 16;
% Modulator object
hMod=modem.pskmod(M); % Modulator object
% Design a pulse shaping filter
nsamp = 16; % up sampling rate
hFilDesign = fdesign.pulseshaping(nsamp,'Raised Cosine',...
...'Nsym,Beta',nsamp,0.50);
hFil = design(hFilDesign);
% Create a scatter plot
hScope = commscope.ScatterPlot
% Set the samples per symbol to the upsampling rate
% of the signal
hScope.SamplesPerSymbol = nsamp;
% Generate data symbols
d = randi([0 hMod.M-1], 100, 1);
% Generate modulated symbols
sym = modulate(hMod, d);
% Apply pulse shaping
xmt = filter(hFil, upsample(sym, nsamp));
% Set the constellation value of the scatter plot to
% the expected constellation
hScope.Constellation = hMod.Constellation;
% Set MeasurementDelay to the group delay of the filter
groupDelay = (hFilDesign.NumberOfSymbols/2);
hScope.MeasurementDelay = groupDelay /hScope.SymbolRate;
% Update the scatter plot with transmitted signal
update(hScope, xmt)
% Display the ideal constellation
hScope.PlotSettings.Constellation = 'on';
```

8.8.3 Eye Diagrams

An eye diagram is a plot of the transmitted signal against time on a fixed-interval axis (T) where at the end of the fixed interval, the signal is wrapped around to the beginning of the time axis. The eye diagram is useful for studying signal quality and in particular the effects of ISI in digital communication systems. Several quality measures of the received signal such as the level of noise and distortion and quality of synchronization can be obtained from an eye diagram. In general, an open eye diagram means that signal distortion is minimum, while a closed eye diagram means that the signal suffers from distortion due to ISI.

The command `eyediagram(x,n,period)` creates an eye diagram for the signal x, plotting n samples in each trace with the horizontal axis range between – `period`/2 and `period`/2. For example, the following script produces an eye diagram for a 16QAM signal with a square-root raised cosine pulse shaping.

```
% File Name:
%% Setup
% Define parameters.
M = 16; % Size of signal constellation
k = log2(M); % Number of bits per symbol
n = 10000; % Number of bits to process
% Oversampling rate
nsamp = 4;
% Create a 16-QAM modulator
hMod = modem.qammod(M);
%% Signal Source
% Create a binary data stream as a column vector.
x = randint(n,1); % Random binary data stream
%% Bit-to-Symbol Mapping
% Convert the bits in x into k-bit symbols.
xsym = bi2de(reshape(x,k,length(x)/k).','left-msb');
%% Modulation
hMod=modem.qammod(M);
y = modulate(hMod,xsym); % Modulate using 32-QAM.
% Upsample and apply square root raised cosine filter.
ytx = rcosflt(y,1,nsamp,'fir/sqrt');
% Create eye diagram for part of filtered signal.
eyediagram(ytx(1:1000),nsamp*2);
```

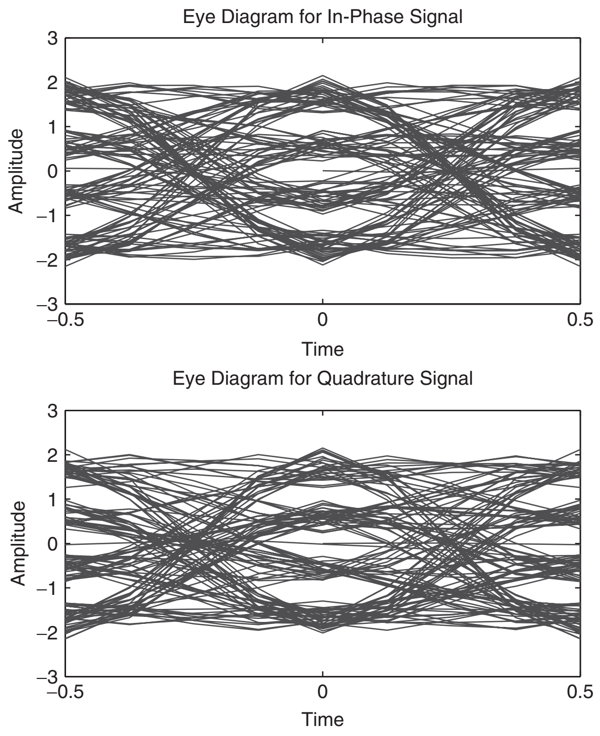


Figure 8.10 Eye diagram of a 16 PSK modulated signal with RRC pulse shaping.

Figure 8.10 shows the resulting eye diagram. Note that the `eyediagram` command creates an eye diagram for part of the filtered signal before the addition of noise where only the effect of the pulse shaping is studied. The figure shows that the signal suffers from significant ISI because the filter is a square root raised cosine filter, not a full raised cosine filter.

The eye diagram can also be generated using the `commscope` object. For example, the command:

```
hScope = commscope.eyediagram
update(h,x)
```

produces an eye diagram of the transmitted signal `x` in a similar manner as the constellation diagram described above.

8.9 Generation of Communications Signals in MATLAB®

8.9.1 Narrowband Gaussian Noise

A narrowband Gaussian noise signal can be generated by passing a white Gaussian noise signal through either a lowpass or a bandpass filter. The following code generates a narrowband Gaussian noise signal at baseband.

```
% File Name: NBGNSpecPlot.m
clear all
% Set noise bandwidth and sampling rate
NoiseBW=4;
Ts=4.8828e-2;
% Calculate filter impulse response
t0=-1e1:Ts:1e1;
FiltCoeff=sinc(NoiseBW*t0);
%% Generate a complex white Gaussian noise signal
% with power=0 dBm
Noise0= wgn(1,132000,0,1,'complex','dBm');
% Apply an ideal  bandpass filter to the white noise signal
Noise=filter(FiltCoeff,1,Noise0);
% Normalize the power to 0 dBm
P=10*log10(mean(abs(Noise).^2)/(50*.001));
Pscale=10^(P/10);
Noise=sqrt(1/Pscale)*(Noise);
%% Calculate the ACF and PSD of the noise signal
W=2^11;
Rn=xcorr(Noise',Noise',W,'biased');
len=length(Rn);
Sn=fftshift(fft(hanning(len).*Rn/len));
% Generate a frequency vector
m=len/2;
delf=2/(Ts*len);
x=-(m):(m-1);
f=delf*x;
% Scale the power of the signal
```

```
Pin=5;
ps=sqrt(10^(Pin/10));
SndB=10*log10(ps^2*Sn/.05);
P=10*log10(sum(abs(ps^2*Sn))/0.05);
%% Plot PSD of NBGN process
plot(f,SndB,'b')
```

Figure 8.11 shows the resulting spectrum of a NBGN signal at baseband.

8.9.2 OFDM Signals

Figure 8.12 shows a flow chart for the procedure used to generate a OFDM signal according to the transmitter model in Figure 2.12 (a). A random data source generates baseband user data that is then entered to a channel encoder and interleaver. Coded data is then mapped using a constellation mapper that generates one of the following modulation formats BPSK, QPSK, or QAM. The serial modulated data symbols are then entered to a Serial-to-Parallel converter (S/P) that produces N data symbol streams with rate $R = R\mathrm{s}/N$. The parallel data streams are then mapped to time domain using the IFFT block. This block generates a vector of N elements, where each complex number element represents one sample of the OFDM symbol.

The complex baseband OFDM signal can be expressed as:

$$s(n) = \sum_{i=0}^{N-1} d_i e^{\frac{j2\pi i}{N}}, 0 \le n \le N-1 \tag{8.2}$$

This sequence corresponds to samples of the multicarrier signal: that is, the multicarrier signal consists of linearly modulated sub channels, and the right-hand side of

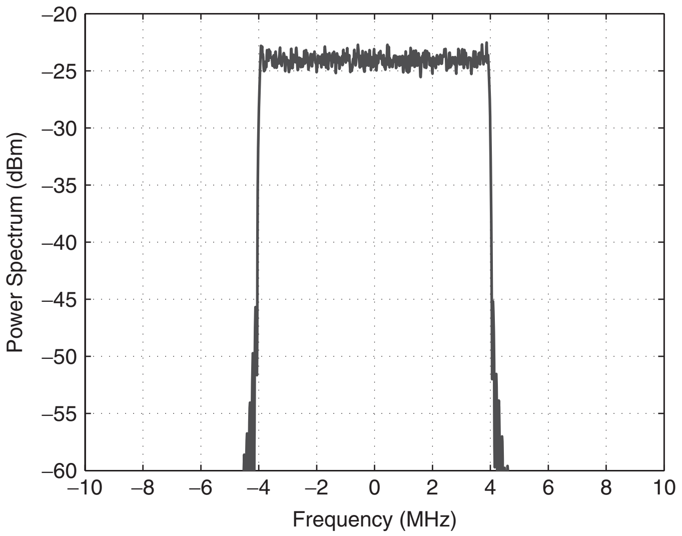


Figure 8.11 PSD of a NBGN signal.

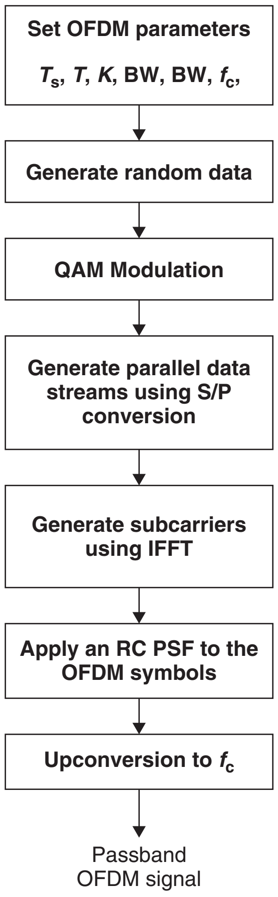

Set OFDM parameters
T_s, T, K, BW, BW, f_c,
Generate random data
QAM Modulation
Generate parallel data streams using S/P conversion
Generate subcarriers using IFFT
Apply an RC PSF to the OFDM symbols
Upconversion to f_c
Passband OFDM signal

Figure 8.12 A flow chart for OFDM signal simulations (Hasan, 2007).

Equation (8.2) corresponds to samples of a sum of QAM symbols d_i each modulated by carrier frequency $e^{\frac{j2\pi i}{N}}$, $i = 0, ..., N-1$. The cyclic prefix is then added to the OFDM symbol, and the resulting time samples $x[n] = x[-\mu], \ldots, .x[N-1] = x[N-\mu], \ldots, x[0], \ldots, x[N-1]$ are ordered by the parallel-to-serial converter and passed through a D/A converter, resulting in the baseband OFDM signal $x(t)$, which is then up converted to frequency f_0. The following script generates an OFDM signals in MATLAB®. Figures 8.13 to 8.15 show the resulting OFDM signal and its PSD.

```
% File Name: OFDMGen.m
clear all
close all
%% OFDM Signal Parameters
K=2000; % number of subcarriers
BWsub=5e3;
BW=K*BWsub; % BW of the modulated signal
```

```
R=2*BW; % Symbol rate of BB QAM data
T=1/R; % baseband elementary period
FS=4096; % IFFT length
Tofdm=FS*T/2; % useful OFDM symbol period
delta=256*T; % guard band duration
Ts=delta+Tofdm; % Total OFDM symbol period
q=10; % carrier period to elementary period ratio
fc=2*q*BW; % carrier frequency
Rs=4*fc; % simulation period
t=0:1/Rs:Tofdm; % Time Variable
% Set up paramters of RC pulse shaping filter
GroupDelay=10;
Alpha=0.4
%% QAM Modulated Data generation
n=K+1;  % Number of QAM symbols
M=16;  % Number of bits per symbol of QAM
% Generate 16 QAM symbols
k = log2(M); % Number of bits per symbol
% Create a 16-QAM modulation object
hMod = modem.qammod(M);
% Create a binary data stream as a column vector.
rand('state',0);
bits = randint(k*n,1); % Random binary data stream
% Convert bits into k-bit symbols.
syms = bi2de(reshape(bits,k,length(bits)/k).','left-msb');
% Modulate using 16-QAM.
bits = modulate(hMod,syms);
figure
plot(real(syms))
%% Generation of parallel data streams of one symbol
A=length(syms);
FFTData=zeros(FS,1);
%Zero padding to match the IFFT length
FFTData(1:(A/2)) = [ bits(1:(A/2)).'];
FFTData((FS-((A/2)-1)):FS) = [ syms(((A/2)+1):A).'];
figure;
plot(real(FFTData));
%% Generation of subcarriers using IFFT
Subcarr=FS.*ifft(FFTData,FS);
t=0:T/2:Tofdm; % Time vector of subcarriers
figure;
plot(t(401:900),real(Subcarr(401:900)));
%% Apply an RC pulse shaping filter to the OFDM symbols
OFDM_BB=rcosflt(Subcarr,1,2*q,'fir',Alpha,GroupDelay);
OFDM_BB=OFDM_BB(1:length(t));
%% Plot the real and imaginary parts of the Baseband signal
figure;
subplot(211);
plot(t(401:900),real(OFDM_BB(401:900)));
subplot(212);
plot(t(401:900),imag(OFDM_BB(401:900)));
```

```
%% Plot the PSD of the baseband signal
Rx=xcorr(OFDM_BB,OFDM_BB,2^11,'biased'); % Autocorrelation function
len=length(Rx);
Sx=fftshift(fft(Rx/len)); % Power spectral density
SxdB_BB=10*log10(Sx/0.05);  % Power spectral density in dBm
len=length(Rx);
m=len/2;
delf=Rs*1e-6/len;
xx=-(m):(m-1);
f=delf*xx;
figure
plot(f,(SxdB_BB))
%% Upconversion and generation passband signal
OFDM=(real(OFDM_BB)').*cos(2*pi*fc*t)-(imag(OFDM_BB)').
     *sin(2*pi*fc*t);
% Plot the modulated signal in time
figure;
plot(t(80:480),OFDM(80:480));
%% Generation and plotting of PSD of passband signal
Rx=xcorr(OFDM,OFDM,2^11,'biased'); % Autocorrelation function
len=length(Rx);
Sx=fftshift(fft(hanning(len)'.*Rx/len)); % Power spectral density
SxdB=10*log10(Sx/0.05);  % Power spectral density in dBm
len=length(Rx);
m=len/2;
delf=Rs*1e-6/len;
xx=-(m):(m-1);
f=delf*xx; % frequency vector
figure
plot(f,(SxdB))
```

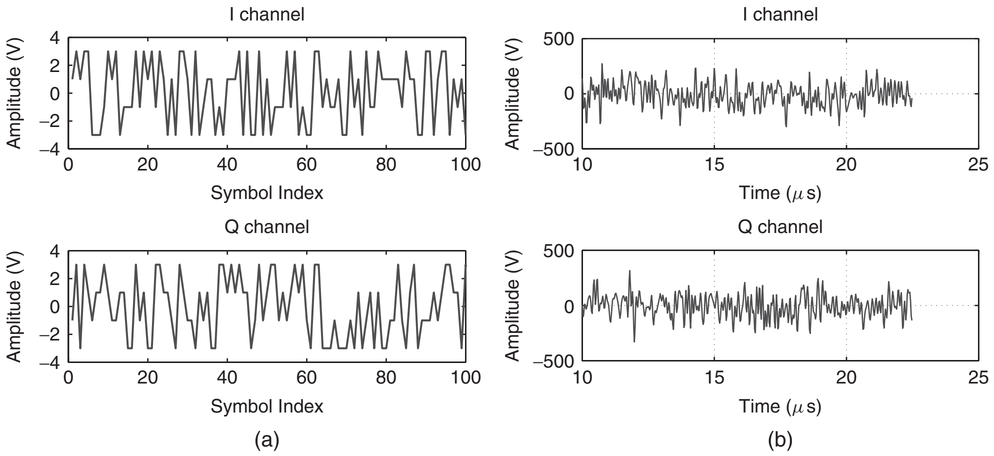
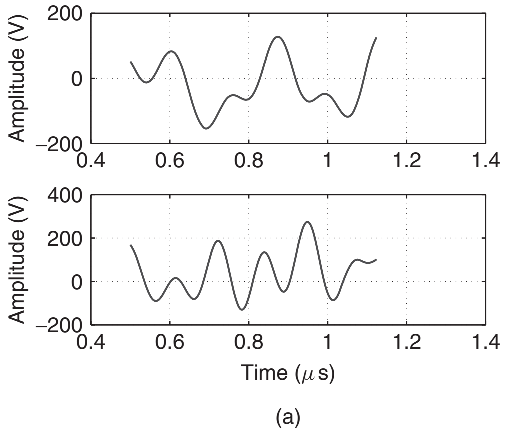


Figure 8.13 OFDM modulation; (a) QAM signal and (b) subcarriers.



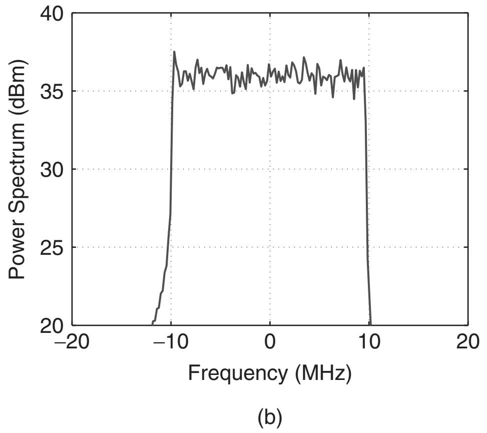
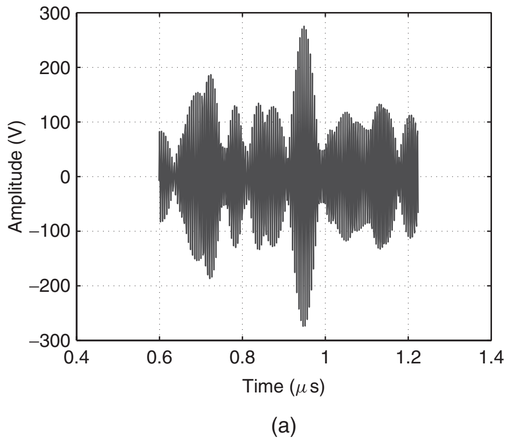


Figure 8.14 OFDM baseband signal; (a) time domain and (b) frequency domain.



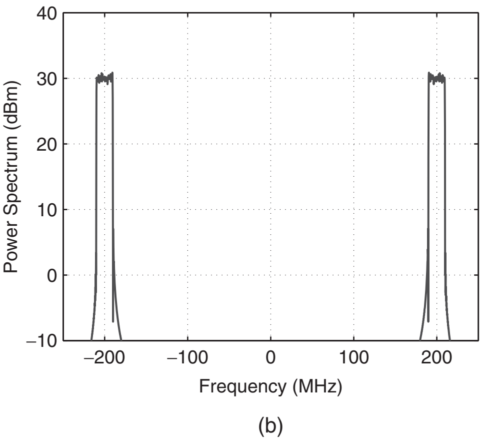


Figure 8.15 OFDM passband signal; (a) time domain and (b) frequency domain.

8.9.3 DS-SS Signals

Various models for DS-SS signal generation in simulation exist. The basic idea in the generation of DS-SS signals is to perform spreading of a randomly generated baseband sequence using a spreading sequence. Spreading sequences can be generated using random sequence generators. In multiuser CDMA systems, orthogonal spreading sequences (such as orthogonal Walsh codes) are used to produce a multiuser coded signal that consists of the sum of spread user data. Figure 8.16 shows a block diagram for a basic CDMA signal generator. $2n$ random streams that represent user data (each user has I and Q data streams) are first generated and then oversampled in order to perform spreading by a code. The oversampling rate is equal to spreading factor since each bit is

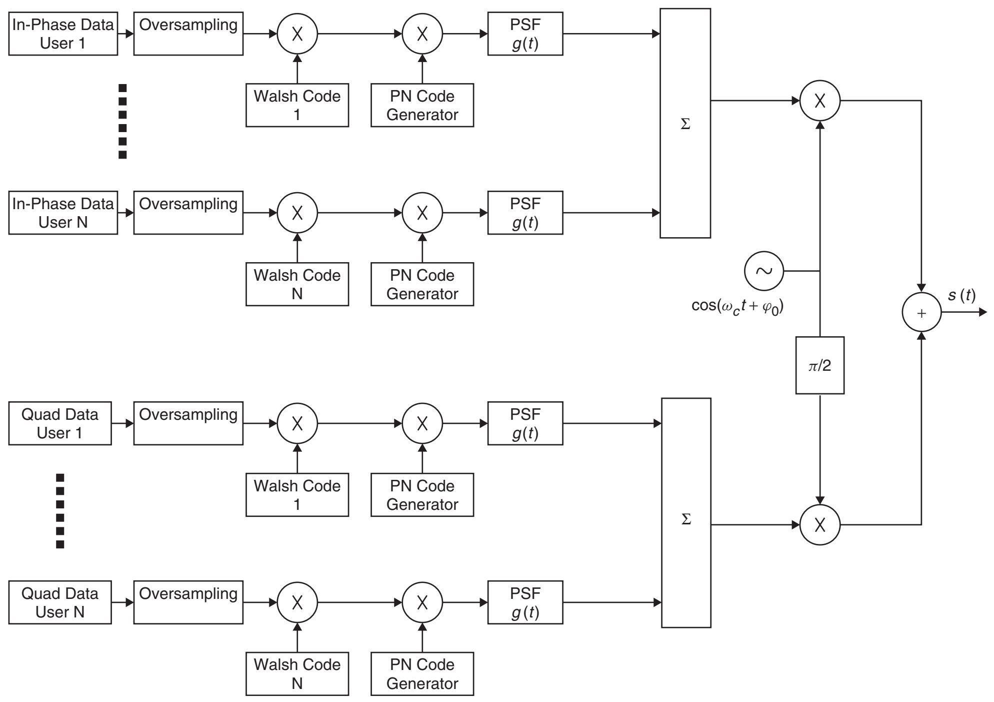


Figure 8.16 A block diagram of CDMA signal generation in MATLAB®.

represented by N chips that represent a Walsh code. Then each user data is multiplied by a unique repeated Walsh sequence to provide orthogonal covering. The resulting user-coded sequence is further multiplied by a PN sequence of the same length. Each coded user data is then pulse shaped using a raised cosine filter and then added together to produce a composite CDMA signal. The script below produces a CDMA signal for a 64-user system where Walsh codes are generated using a Hadamard matrix generator. The script also generates and plots the PSD of the resulting signal.

```
% File Name: DS_CDMA_Gen.m
clear all
% Set the number of user data symbols
n=(2^6);
% Oversampling
nsamp=4;
% Length of Walsh code
N=64;
% Generate Walsh codes of length N
H=hadamard(N);
% Generate repeated Walsh codes of length N*n
WalshCode=repmat(WalshCode,1,n);
% Initialize variables
Ichanc=zeros(1,N*n*nsamp+80);
```

```
Qchanc=Ichanc;
% Generate a PN sequence for the I and Q channels
PNI=randsrc(n*N,1,[1 -1])';
PNQ=randsrc(n*N,1,[1 -1])';
for d=1:N
    bitsI=randsrc(n,1,[1 -1]);
    bitsQ=randsrc(n,1,[1 -1]);
    % Match the sequence length to be multiple of N (1 bit length=N
    % samples)
    I =rectpulse(bitsI,N)';
    Q =rectpulse(bitsQ,N)';
    % Walsh code spreading of the I and Q data
    walshbitsi=WalshCode(d,:).*I;
    walshbitsq=WalshCode(d,:).*Q;
    % Randomize the spreaded I and Q sequences using PN codes
    walshbitsI=PNI.*walshbitsi;
    walshbitsQ=PNQ.*walshbitsq;
    % Filter the I and Q data with an
    % RC pulse shaping FIR filter
    Ichan=rcosflt(walshbitsI,1,nsamp,'fir',0.4,10);
    Qchan=rcosflt(walshbitsQ,1,nsamp,'fir',0.4,10);
    % Generate the composite signal (multiuser signal)
    Ichanc=Ichanc+Ichan';
    Qchanc=Qchanc+Qchan';
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Normalize the power to zero dBm
Pin=10*log10(mean(abs(Ichanc+1i*Qchanc).^2)/(50*.001));
Pscale=10^(Pin/10);
% Complex DS-CDMA signal generation
CDMA=sqrt(1/Pscale)*(Ichanc+1i*Qchanc);
% Compute the autocorrelation estimate
y=transpose(CDMA);
ry=xcorr(y,y,2^11,'biased');
% Calculate the power spectral density of the CDMA signal
len=length(ry);
Sy=fftshift(fft(hanning(len).*ry/len));
% Create a frequency vector
len=length(Sy);
m=len/2;
delf=10/(len);
xx=-(m):(m-1);
f=delf*xx;
% Plot the PSD vs. frequency
SydB=10*log10(Sy/.05);
figure
plot(f,SydB);
```

Figure 8.17(a) shows the I and Q data of user k, Figure 8.17(b) shows the user data after spreading by Walsh code k, Figure 8.17(c) shows the resulting sequence after further

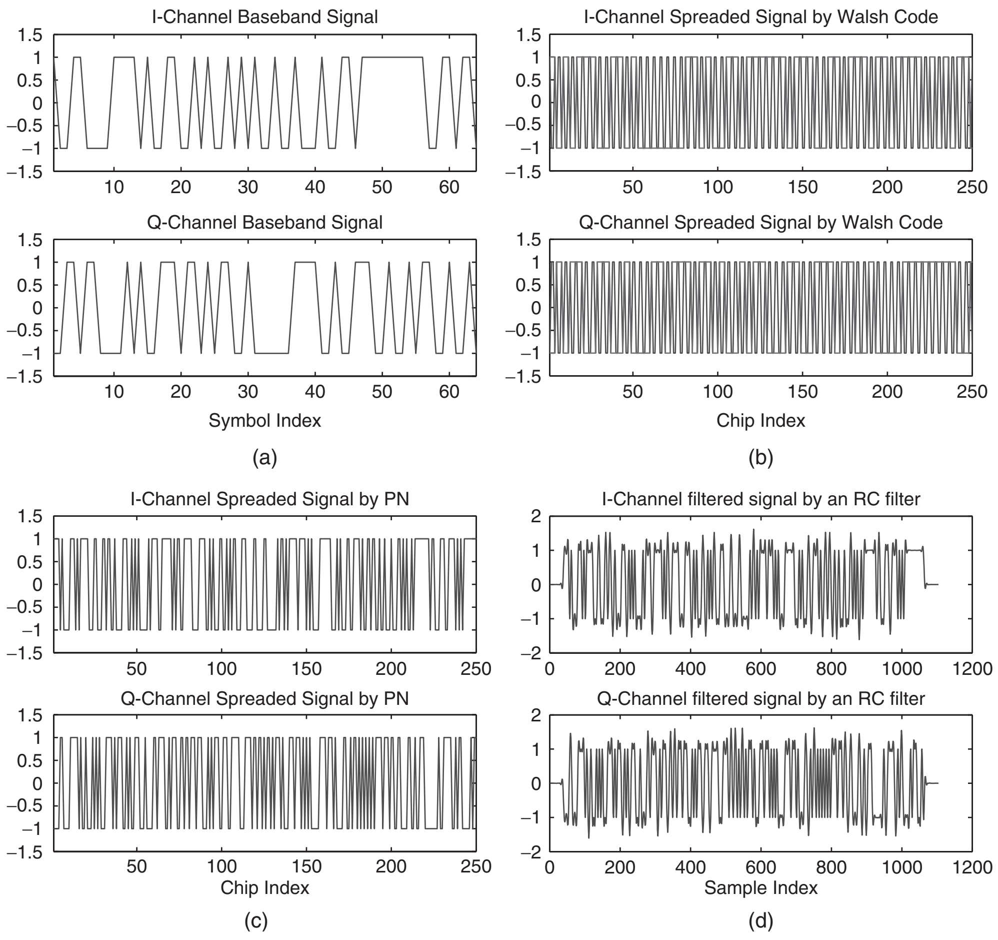


Figure 8.17 DS-CDMA signal generation; (a) I and Q data of user k, (b) after spreading by Walsh code, (c) after PN spreading and (d) after baseband filtering.

spreading by a PN sequence and Figure 8.17(d) shows the resulting signal after baseband filtering (RC pulse shaping). Figure 8.18 shows the PSD of the resulting CDMA signal.

8.9.4 Multisine Signals

A multisine signal $x(t)$ consisting of the sum of an even number of tones with a uniform frequency separation can be represented as (Gharaibeh *et al.*, 2006)

$$x(t) = \sum_{k=-K}^{K} \frac{A}{2} \cos\left(\omega_c t + (k - 1/2)\, \omega_{\mathrm{m}} t + \phi_k\right) \tag{8.3}$$

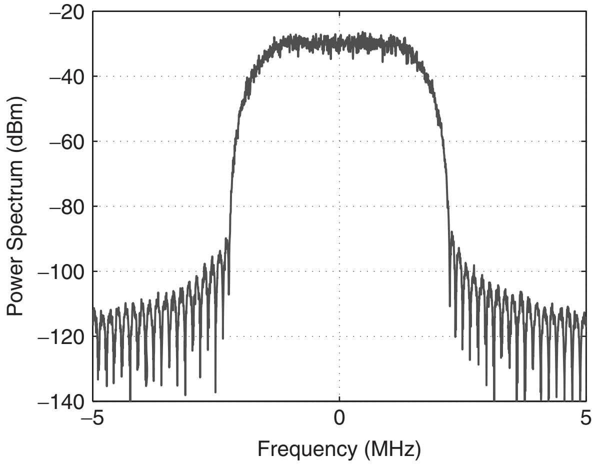


Figure 8.18 PSD of a CDMA signal.

where K is an even number and $K \neq 0$, $\omega_m = \omega_k - \omega_{-k}$ is the frequency separation between the input tones and ϕ_k are the phases of the input tones which are assumed to be independent random phases uniformly distributed in $[0, 2\pi]$.

The multisine signal in Equation (8.3) can be written as a sum of complex conjugate pairs as (Gharaibeh *et al.*, 2006)

$$x(t) = \sum_{k=-K}^{K} \frac{1}{2}\tilde{x}_k(t)e^{j\omega_c t} \tag{8.4}$$

it follows that $\tilde{x}_k = A\cos\left(\left(k - \frac{1}{2}\right)\omega_m t + \theta_{1k}\right)e^{j\theta_{2k}}$ where $\theta_{2k} = (\phi_k + \phi_{-k})/2$ and $\theta_{1k} = (\phi_k - \phi_{-k})/2$ and hence the complex envelope of Equation (8.4) is

$$\tilde{x}(t) = \sum_{k=1}^{K} A\cos\left(\left(k - \frac{1}{2}\right)\omega_m t + \theta_{1k}\right)e^{j\theta_{2k}}. \tag{8.5}$$

For the special case where $\phi_k = -\phi_{-k}$ the above equation can be written as

$$\tilde{x}(t) = \sum_{k=1}^{K} A\cos\left(\left(k - \frac{1}{2}\right)\omega_{\text{m}} t + \phi_k\right). \tag{8.6}$$

This form can be easily simulated under the above assumptions where the only random variable is the phase of each of the tones. Uniformly distributed random phases can be generated using a uniform random number generator in MATLAB®.

The following script generates the complex envelope of a multisine signal that consists of 8 tones with random phases and plots its frequency spectrum. Figure 8.19 shows the time-domain representation of an 8-tone signal and Figure 8.20 shows its PSD.

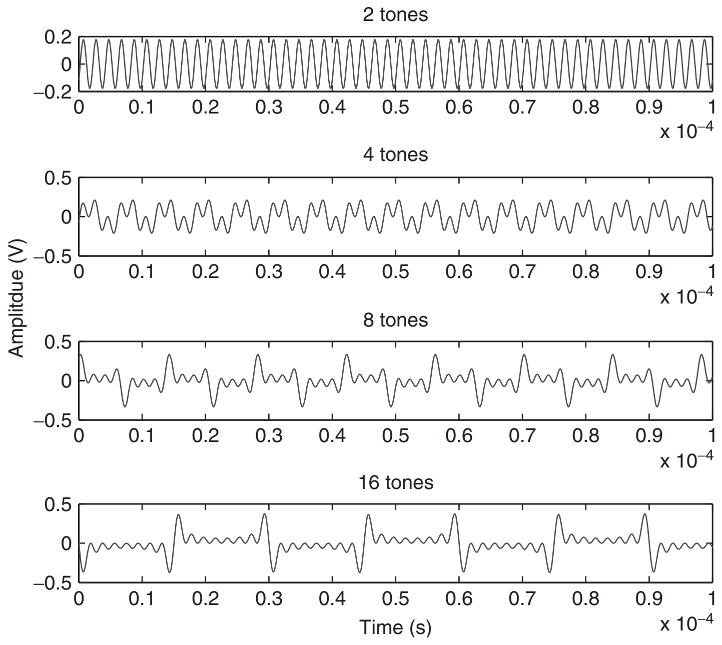


Figure 8.19 A phase-aligned multisine signal for different number of tones.

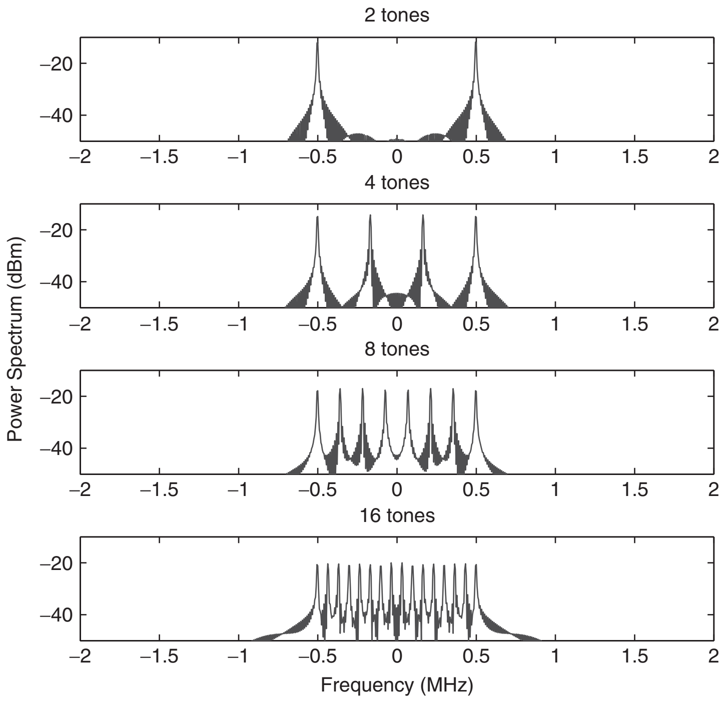


Figure 8.20 Power spectrum of multisine signals for different number of tones.

```
% File Name: MulltiSine_Gen.m
clear all
n=2;   % Number of tones
bw=1e6; % Bandwidth of multisine signal
freq_sep=bw/(n-1); % Frequency separation between tones (Constant)
% Generate the frequency vector
for i=1:n/2
    f(i)=(freq_sep/2)+freq_sep*(i-1);
end
% Set the sampling frequency and generate a time vector
fs=20*f(n/2);  % Sampling frequency
Ts=1/fs; % Sampling time
t=0:Ts:1e3*Ts; % Time vector
% Generate tones with random phases.
Msine=zeros(size(t)); % Initialize multisine signal with zeros
for k=1:(n)/2
    rand('seed',sum(100*clock));
    rand('state',sum(100*clock));
    phi=-pi+(2*pi)*(rand(1)); % Random phases generation for tone k
    Msine=Msine+cos(2*pi*f(k)*t+phi); % Multisine signal
end
% Normalize the power of the multisine signal to zero dB
Pin=10*log10(mean(abs(Msine).^2)/(50*.001));
Pscale=10^(Pin/10);
Msine=sqrt(1/Pscale)*(Msine);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Compute the autocorrelation function and the power spectrum
% of the multisine signal at input power = pin (dBm).
pin=-5; % Input power =-5 dBm
A=sqrt(10^((pin)/10)); % Amplitude scaling
Msine_scaled=A*Msine;
Rx=xcorr(Msine_scaled,Msine_scaled,2^10,'biased'); % Autocorrelation
len=length(Rx);
Sx=fftshift(fft(Rx/len)); % Power spectral density
SxdB=10*log10(Sx/0.05);  % Power spectral density in dBm
% Generate a frequency vector to plot the spectrum
len=length(Rx);
m=len/2;
delf=fs/1e6/len;
xx=-(m):(m-1);
f1=delf*xx;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot the time domain multisine signal
figure
plot(t,Msine_scaled)
axis([0 1e-4 -1.5 1.5])
hold on
```

```
% Plot the power spectrum of multisine signal
figure
plot(f1,SxdB)
axis([-2 2 -50 -10])
```

Note that the above scripts generates a single realization of a multisine signal with a random phase. If a full representation of the random signal is required, then, the script needs to be repeated to generate other realizations. When dealing with simulation of nonlinear distortion of multisines, distortion is estimated by averaging a large number of phase realizations. The number of realizations required depends on the number of tones since as the number of tones increases the probability of having a uniformly distributed phases increases.

8.10 Example

The following script simulates overall communication system operations including data generation, pulse shaping, modulation and demodulation. It also plots an eye diagram and a scatter plot and conducts BER calculations.

```
% File Name: Example_ModDemodBER.m
clear all, close all
%% Setup
% Define parameters.
M = 16; % Size of signal constellation
k = log2(M); % Number of bits per symbol
n = k*1e6; % Number of bits to process
nsamp = 4; % Oversampling rate
hMod = modem.qammod(M); % Create a 32-QAM modulator
%% Data Generation
% Create a binary data stream as a column vector.
txbits = randint(n,1); % Random binary data stream
% Bit-to-Symbol Mapping
% Convert the bits in x into k-bit symbols.
txsym = bi2de(reshape(txbits,k,length(txbits)/k).','left-msb');
%% Modulation
hMod=modem.qammod(M);
QAM = modulate(hMod,txsym); % Modulate using 32-QAM.
%% RC pulse-shaping.
GroupDelay=10;
alpha=.8;
QAM_PS=rcosflt(QAM,1,nsamp,'fir',alpha,GroupDelay);
%% Create a scatter plot
hScope = commscope.ScatterPlot
% Set the samples per symbol to the upsampling rate of the signal
hScope.SamplesPerSymbol = nsamp;
% Set the constellation value of the scatter plot to the
% expected constellation
hScope.Constellation = hMod.Constellation;
% Update the scatter plot with transmitted signal
```

```
update(hScope, QAM)
%% Create an Eye Diagram
eyediagram(QAM_PS(1:1000),nsamp*2);
%% Send signal over an AWGN channel
% Transmitted Signal
txSig = QAM_PS;
% Signal to Noise Ratio
EbNo =15; % In dB
snr = EbNo + 10*log10(k) - 10*log10(nsamp);
txnoisy = awgn(txSig,snr,'measured');
%% Received Signal
rxSig=downsample(txnoisy,nsamp);
%% Demodulation
% Demodulate signal using 16-QAM.
rxsym = demodulate(modem.qamdemod(M),rxSig);
% Remove group delay of pulse shaping filter
rxsym=rxsym(GroupDelay+1:size(rxsym)-GroupDelay);
% Symbol-to-Bit Mapping
% Undo the bit-to-symbol mapping
rxbits = de2bi(rxsym,'left-msb'); % Convert integers to bits.
% Convert rxbits from a matrix to a vector.
rxbits = reshape(rxbits.',numel(rxbits),1);
%% BER Computation
% Compare x and z to obtain the number of errors and
% the bit error rate.
[number_of_errors,bit_error_rate] = biterr(txbits,rxbits)
% the symbol error rate.
[number_of_errors,sym_error_rate] = symerr(txsym,rxsym)
```

8.11 Random Signal Generation in Simulink®

In addition to the random data and noise generators in the main Simulink® sources library (including reading a measured random signal from a.mat file), random data sources and noise generators are available in Comm. Source library in the Communications Blockset library. The Comm. Sources library consists of three main sublibraries: Random Data Sources, Noise Generators and Random Sequence sublibraries (The Communication Blockset, 2009). Figure 8.21 shows the main sublibraries of the communication sources library within the Communications Blockset.

8.11.1 Random Data Sources

A number of random signals can be generated using this sublibrary such as Random Integer Generator, Poisson Integer Generator and Bernoulli Binary Generator. Figure 8.22 shows the Simulink® library of random data sources. The Bernoulli Binary Generator block generates independent random bits (binary integers) where each bit is generated from a Bernoulli random variable. Thus, the generated bit stream can represent binary data sources in real-world communication systems. Random Integer Generator generates vectors of non-negative random integers having uniform distribution, while

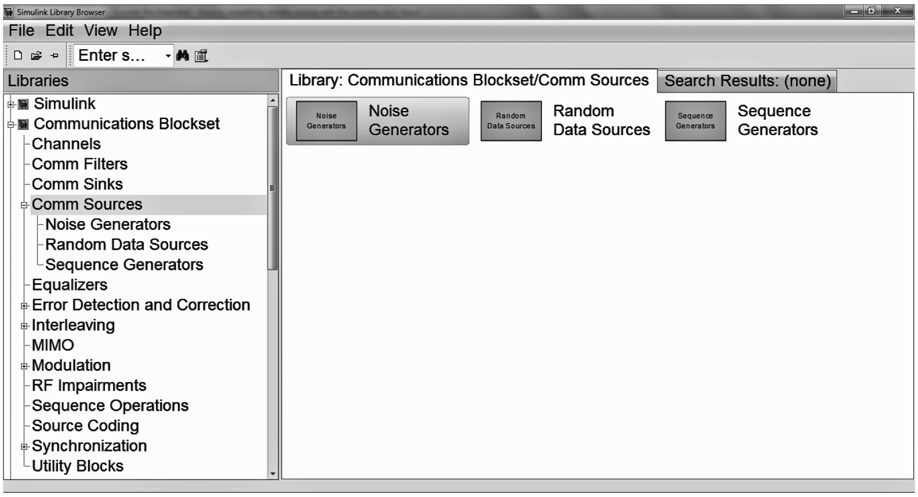


Figure 8.21 Simulink® communication sources (The Communication Blockset, 2009).

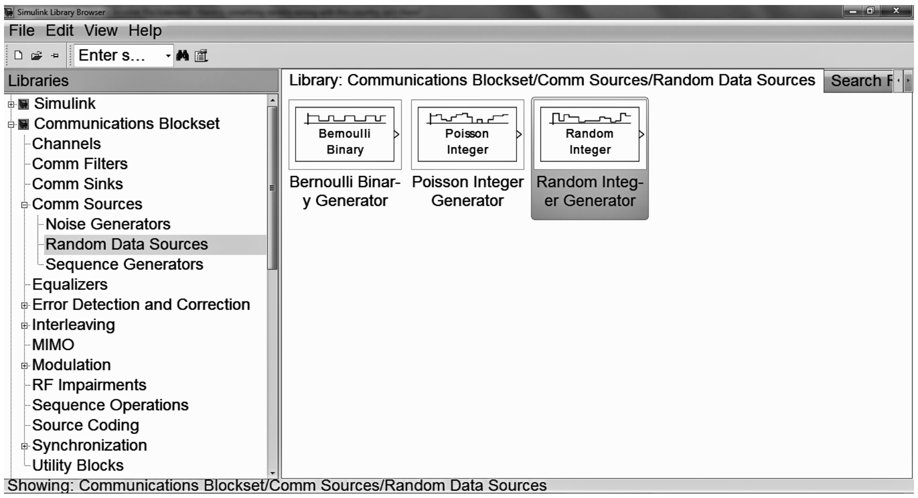
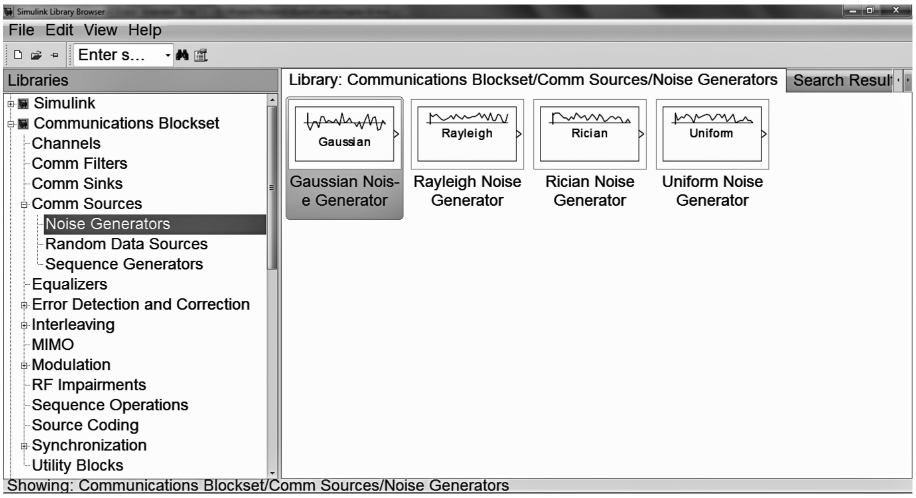


Figure 8.22 Simulink® data sources (The Communication Blockset, 2009).

the Poisson Integer Generator block generates random non-negative integers having a Poisson distribution.

8.11.2 *Random Noise Generators*

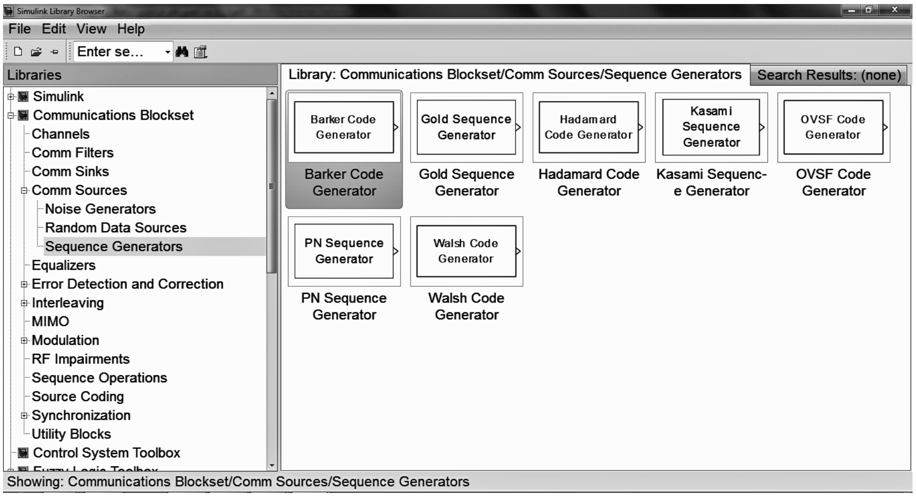
Noise Generators sublibrary contains a number of random real number generators with a given probability distribution that simulate channel noise. The sublibrary includes Gaussian Noise Generator, Rayleigh Noise Generator, Rician Noise Generator and



Library: Communications Blockset/Comm Sources/Noise Generators

Showing: Communications Blockset/Comm Sources/Noise Generators

Figure 8.23 Simulink® noise sources (The Communication Blockset, 2009).



Library: Communications Blockset/Comm Sources/Sequence Generators

Showing: Communications Blockset/Comm Sources/Sequence Generators

Figure 8.24 Simulink® random sequence sources (The Communication Blockset, 2009).

Uniform Noise Generator. Figure 8.23 shows the Simulink® sub-library of random noise sources (The Communication Blockset, 2009).

8.11.3 Sequence Generators

The Sequence Generators sublibrary contains blocks that generate random sequences using shift registers for spreading or synchronization in a communication system. These random sequences include Pseudo-random sequences, Synchronization codes

Table 8.6 Simulink® random sequence generators (The Communication Blockset, 2009)

Type	Random Sequence
Pseudorandom Sequences	Gold sequences, Kasami Sequence, Pseudonoise sequences
Synchronization codes	Barker Code
Orthogonal codes	Hadamard codes, OVSF codes, Walsh codes

and Orthogonal codes. These codes can be used for multiple-access spread spectrum communication systems, synchronization, and data scrambling (The Communication Blockset, 2009). Figure 8.24 shows the Simulink® sublibrary of random sequence sources. Table 8.6 shows the different sequence generator blocks in the sequence generator sublibrary.

8.12 Digital Modulation in Simulink®

The Digital Modulation library in the Communications Blockset consists of blocks that perform baseband simulation of digital modulation/demodulation schemes. The baseband modulation blocks generate the lowpass equivalent of common digital modulation schemes. Figure 8.25 shows the digital modulation sublibraries under the Digital Modulation library. These sublibraries include (The Communication Blockset, 2009):

- amplitude modulation: PAM, ASK, QAM;
- phase modulation PM: MPSK, DQPSK, OQPSK, CPSK;
- frequency modulation: FSK;
- Continuous Phase Modulation (CPM): CPFSK, MSK, GMSK;
- trellis-coded modulation: TCM, MPSK TCM, MQAM TCM.

8.13 Simulation of System Performance in Simulink®

System performance can be simulated in Simulink® using a number of approaches. The “Sinks” library in the “Communications Blockset” contains three types of signal scopes:

- Eye Diagrams: for plotting the eye diagram of a discrete signal.
- Scatter Plots: for plotting scatter plot of a discrete signal.
- Signal Trajectories: for plotting signal trajectory of a discrete signal.

These scopes can be used to facilitate viewing the performance of a digital communication system under channel or RF impairments. Figure 8.26 shows the communications sinks sublibrary.

On the other hand, error rates can be calculated in communication system model using the Error Rate Calculation block that compares input data from a transmitter with input data from a receiver and calculates the error rates or the number of errors at either the bit or symbol levels.

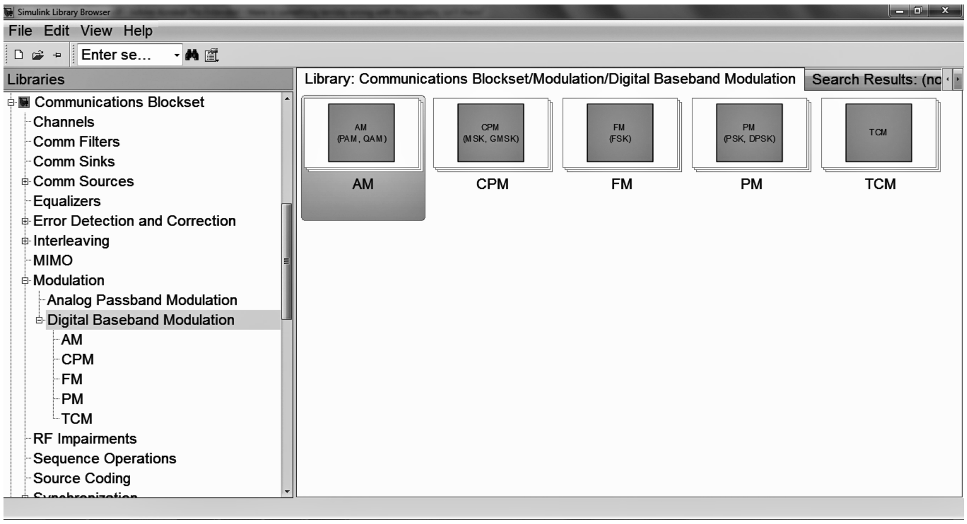


Figure 8.25 Simulink® digital modulation library (The Communication Blockset, 2009).

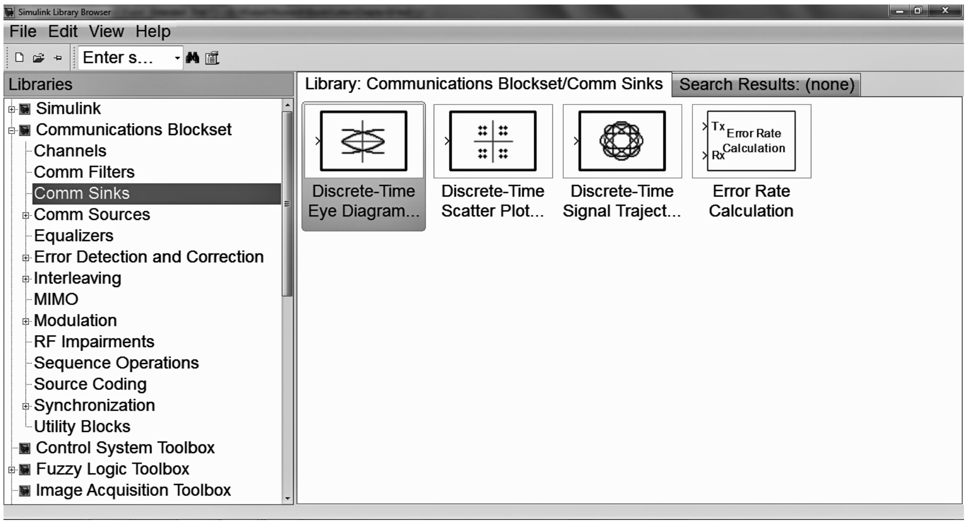


Figure 8.26 Simulink® communication sinks (The Communication Blockset, 2009).

Signal spectrum calculation blocks can be used to assess system performance by viewing the changes that occur to the input signal spectrum as a result of channel or RF impairments. There are sinks under the Signal Processing Blockset such as the Vector Scope, Spectrum Scope, Matrix Viewer, and Waterfall Scope blocks that can be used to view the signal in the frequency domain. The Spectrum Scope block displays the frequency spectrum of an input signal by computing its FFT.

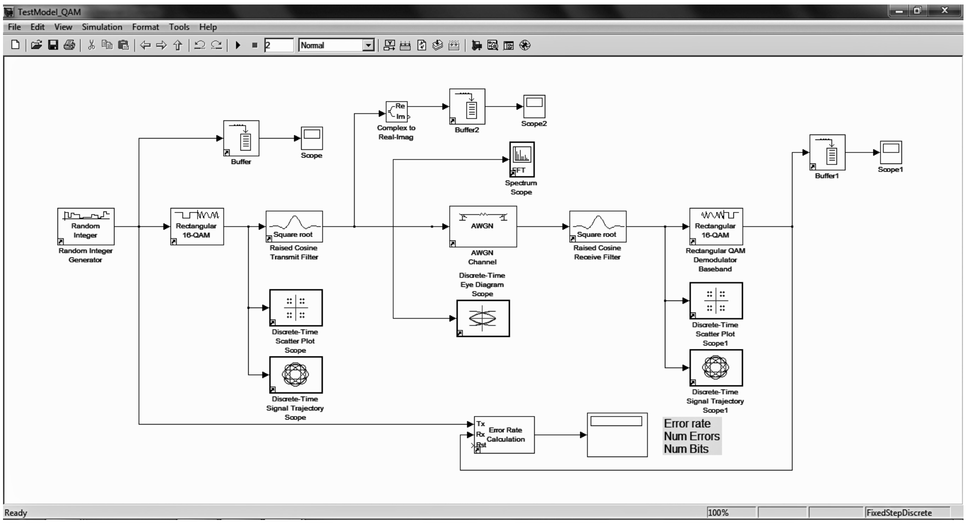

TestModel_QAM
File Edit View Simulation Format Tools Help
Random Integer
Random Integer Generator
Rectangular 16-QAM
Square root
Raised Cosine Transmit Filter
Buffer
Scope
Complex to Real-Imag
Buffer2
Scope2
FFT
Spectrum Scope
AWGN
AWGN Channel
Discrete-Time Eye Diagram Scope
Square root
Raised Cosine Receive Filter
Rectangular 16-QAM
Rectangular QAM Demodulator Baseband
Buffer1
Scope1
Discrete-Time Scatter Plot Scope
Discrete-Time Signal Trajectory Scope
Discrete-Time Scatter Plot Scope1
Discrete-Time Signal Trajectory Scope1
Tx
Rx
Rst
Error Rate Calculation
Error rate
Num Errors
Num Bits
Ready
100%
FixedStepDiscrete

Figure 8.27 Example 1: QAM modulation/demodulation model in Simulink®.

8.13.1 *Example 1: Random Sources and Modulation*

In the simple example shown in Figure 8.27 (File Name: `Example_Mod.mdl`), the various blocks that can be used to assess the performance of communication systems are used. The example consists of a model for simulation of QAM transmitter and receiver system with an AWGN channel. The transmitter model consists of a random integer generator, a QAM modulator and a transmit pulse-shaping filer. The channel is modeled as an AWGN channel model using the AWGN channel block. The receiver model consist of a receive pulse-shaping filter and a QAM demodulator. The parameters of the various blocks are shown in Table 8.7.

The performance of the system in Example 1 is assessed using time scopes, spectrum scopes, signal constellation, signal trajectory and a BER calculator. Figure 8.28 shows the plots generated by various performance blocks where it is shown how noise affects the

Table 8.7 Parameters of various blocks of Example 1

Block	Parameters
Random Integer Generator	M = 16, Sample time = 1e-5
QAM modulator	M = 16, Contellation = Binary
AWGN channel	E_s/N_o = 30 dB
Transmit pulse shaping filer	Group delay = 8, α = 0.3, Upsampling = 8
Receive pulse shaping filter	Group delay = 8, α = 0.3, Downsampling = 8
QAM demodulator	M = 16, Contellation = Binary
Error Rate Calculation	Receive Delay = 6

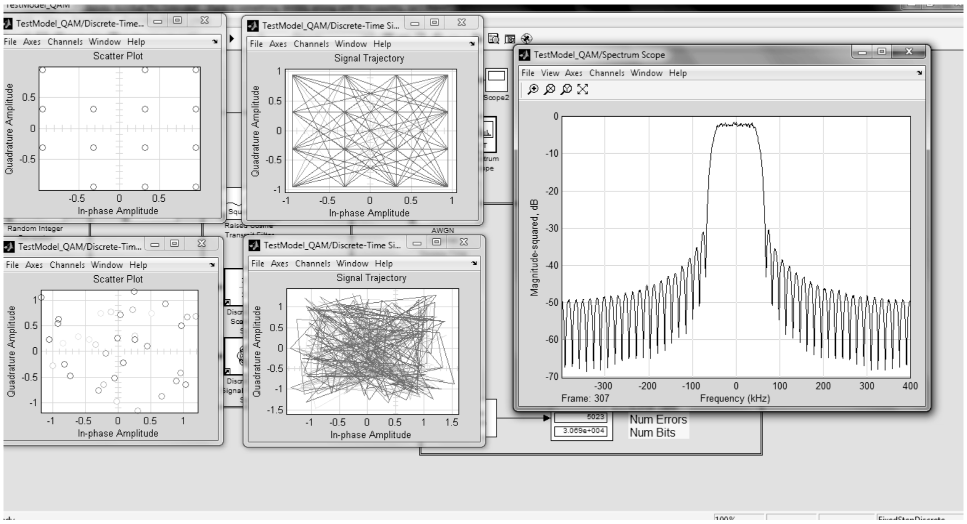


Figure 8.28 Various performance plots for the QAM modulation/demodulation in Example 1.

constellation, signal trajectory and BER of the received signal. Other channel impairments such as fading channels and nonlinear amplification can be added to the model.

The BER calculator compares the transmitted and received symbols and counts the number of errors during the simulation time. It calculates error rate, number of error events or total number of input events at the bit or symbol levels. The SNR can be varied from the AWGN channel block and a plot of BER vs. SNR can be developed.

8.13.2 Example 2: CDMA Transmitter

Example 2, shown in Figure 8.29 (File Name: `Example_CDMA.mdl`), simulates a CDMA transmitter and generates a baseband direct-sequence spread spectrum signal according to IS-95 mobile standard in Simulink®. Binary data at a sampling rate of 1/19200 Hz is generated using the "Bernoulli Random Binary Generator" block and fed into a "QPSK Modulator Block" that outputs complex modulation data. The modulated signal is then multiplied by a Walsh code generated from the "Hadamard Code Generator Block" that generates repeated Walsh code drawn from an orthogonal set of codes with length N (64 in this example). The sample time of the Walsh code (chip time) is 1/(19 200*64) since every bit is multiplied by 64 chips. The spread data at the output of the multiplies is then fed to a SRRC pulse shaping filter with $\alpha = 0.3$, up-sampling of 8 samples and a group delay of 16 samples. The parameters of the various blocks of Example 2 are shown in Table 8.8.

Figure 8.30 shows the spectrum of the signal before spreading and after spreading where it is shown that the bandwidth of the spread signal is 64 times the bandwidth of the signal before spreading; that is, a processing gain of 64.

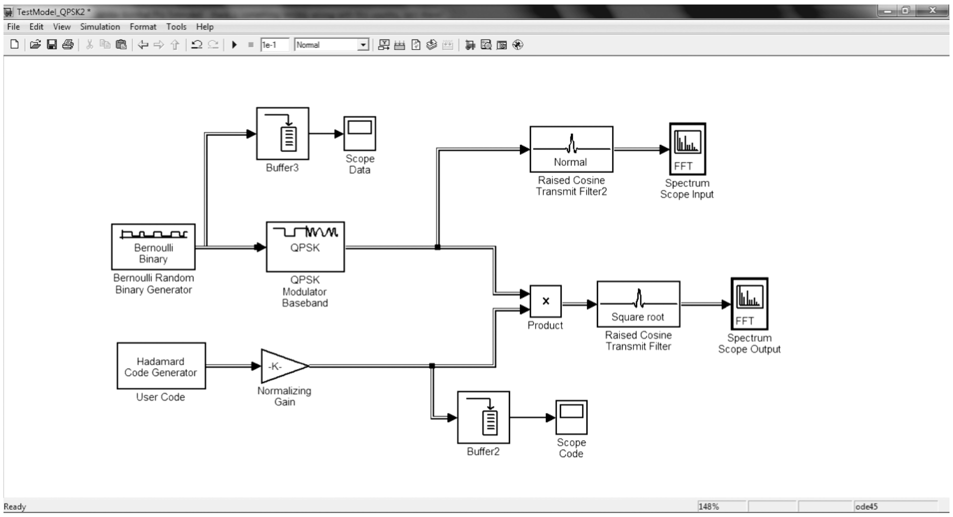


Figure 8.29 Example 2: CDMA transmitter model in Simulink®.

Table 8.8 Parameters of various blocks of Example 2

Block	Parameters
Random Binary Generator	Sample time = 1/19 200 Hz, Samples per frame = 100
QPSK Modulator	Phase offset = $\pi/4$, Contellation = Binary
Hadamard Code Generator	Code length = 64, Sample time = 1/(19 200*64)
RC Transmit Filter	Group delay = 16, α = 0.3, Upsampling = 8

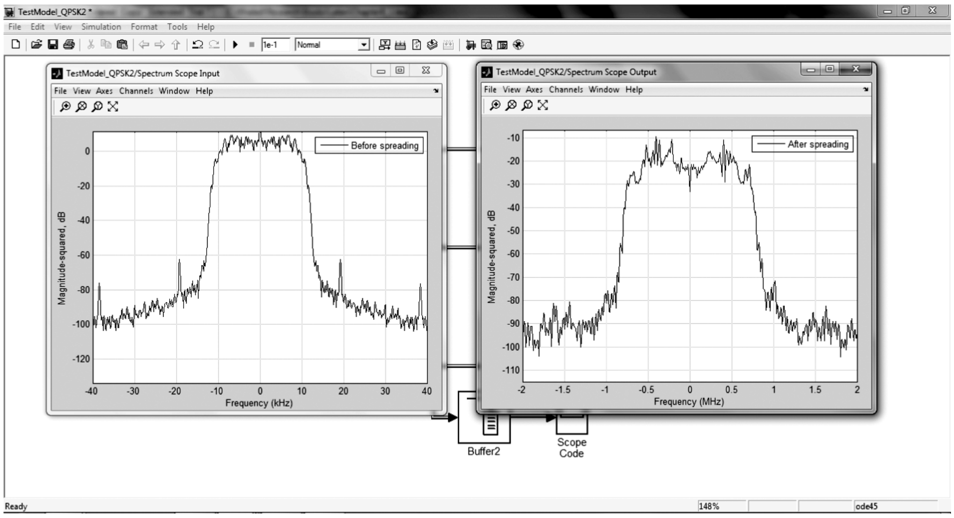


Figure 8.30 Input and output spectrum of CDMA transmitter in Example 2.

Table 8.9 Models of wireless standards in Simulink® (The Communication Blockset, 2009)

Model	Description
256-Channel ADSL	models part of the Asymmetric Digital Subscriber Line (ADSL) technology
Bluetooth® Frequency Hopping	simulates a simple Bluetooth® wireless data link
Bluetooth® Voice Transmission	models part of a Bluetooth® system
Bluetooth® Full Duplex Voice and Data Transmission	simulates the full duplex communication between two Bluetooth® devices
CDMA2000 Physical Layer	simulates part of the downlink physical layer of a wireless communication system according to the cdma2000 specification
Defense Communications: US MIL-STD-188-110B	implements an end-to-end baseband communications system compliant with the U. S. MIL-STD-188-110B military standard
Digital Video Broadcasting – Cable (DVB-C)	models part of the ETSI EN 300 429 standard for cable system transmission of digital television signals
Digital Video Broadcasting – Terrestrial	models part of the ETSI EN 300 744 standard for terrestrial transmission of digital television signals
DVB-S.2 Link, Including LDPC Coding	simulates the state-of-the-art channel coding scheme used in the second generation Digital Video Broadcasting Standard (DVB-S.2)
HIPERLAN/2	models part of HIPERLAN/2 (high-performance radio local area network), European (ETSI) Standard for high-rate wireless LANs
IEEE® 802.11a WLAN Physical Layer	represents an end-to-end baseband model of the physical layer of a Wireless Local Area Network (WLAN) according to the IEEE® 802.11a standard
IEEE® 802.11b WLAN Physical Layer	simulates an implementation of the Direct Sequence Spread Spectrum (DSSS) system.
IEEE® 802.16-2004 OFDM PHY Link, Including Space-Time Block Coding	simulates an end-to-end baseband model of the physical layer of a WMAN according to the IEEE® 802.16-2004 standard
WCDMA Coding and Multiplexing	simulation of the multiplexing and channel decoding structure for the Frequency Division Duplex (FDD) downlink as specified by 3GPP, Release 1999
WCDMA End-to-End Physical Layer	models part of the Frequency Division Duplex (FDD) downlink physical layer of 3G WCDMA.
WCDMA Spreading and Modulation	simulates spreading and modulation for an FDD downlink DPCH channels as specified by the 3GPP, Release 1999.

8.13.3 Simulation of Wireless Standards in Simulink®

The Communications Blockset contains demos of a number of wireless standards that are ready to use. These models simulate a number of common mobile communication and wireless networking standards and can be used to generate real-world communication signals. Table 8.9 lists the available models in Simulink® and their description.

Other wireless standard models such as IS-95 CDMA, GSM, EDGE systems are available at the MATLAB® Central website (MATLAB® Central, 2011).

8.14 Summary

In this chapter, the basics of communication system simulation in MATLAB® and Simulink® have been presented. The discussions on various elements of communication system simulations are supported by examples that can be used for evaluation of system performance for more complex systems. Various simulation issues such as sampling rate of simulations, measurement of system performance from simulated spectra and the choice of the parameters of MATLAB® functions and Simulink® blocks have also been discussed. This chapter serves as an introduction to the next chapter that discusses simulation of nonlinear systems where the objective is to assess the performance of a communication system under nonlinear channel impairments.