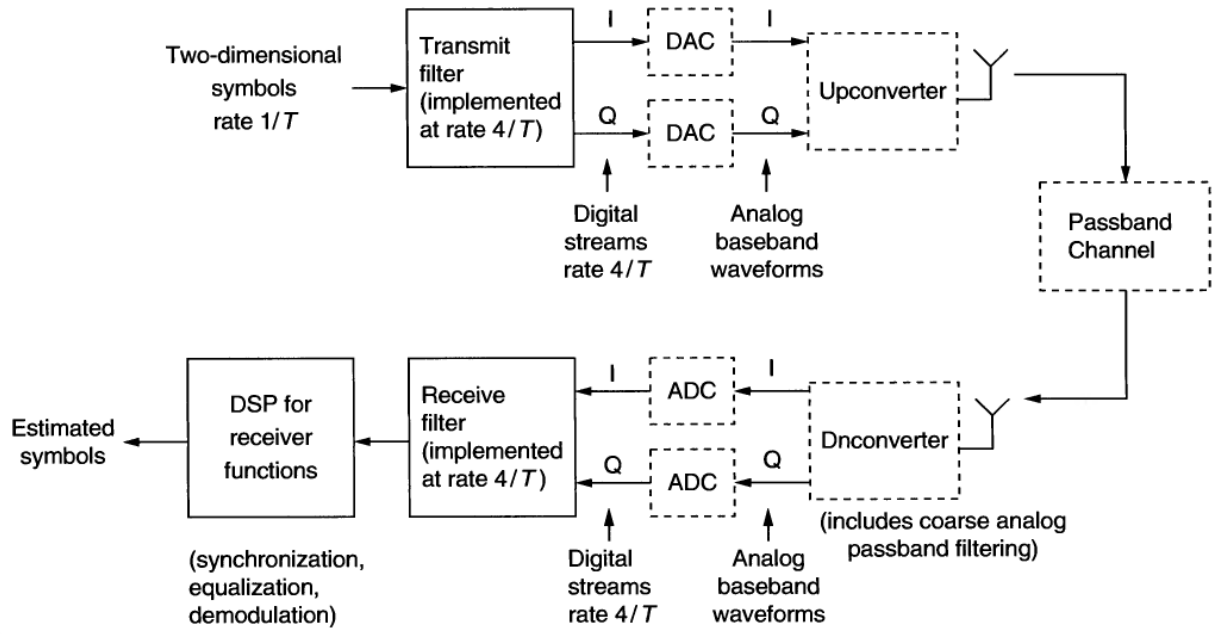
Linear Modulation With Two Dimensional Constellation

ECE 146 Lab 4

December 3, 2019



Student: Ivan Arevalo
Perm Number: 5613567
Email: ifa@ucsb.com

Department of Electrical and Computer Engineering, UCSB

0 Introduction

This lab implements a linearly modulated system (Figure 1) over a noisy channel using the complex-baseband representation with a variety of signal constellations. Our model does not include DAC and ADC effects for simplicity. We are choosing the cascaded transmitter and receiver filters to be a raised cosine (RC) pulse by choosing each to be a square root raised cosine pulse (SRRC) which inherits the Nyquist property of the sinc pulse. The excess bandwidth and smooth frequency shape of a RC pulse results in a time domain attenuation proportional to $\frac{1}{t^3}$, mitigating intersymbol interference (ISI) in presence of slight sampling perturbations at the receiver. We will simulate sending 12000 random bits under 5 different signal constellations, adding white Gaussian noise to the receiver input signal, and comparing the probability error from the recovered bits to the ideal probability error.

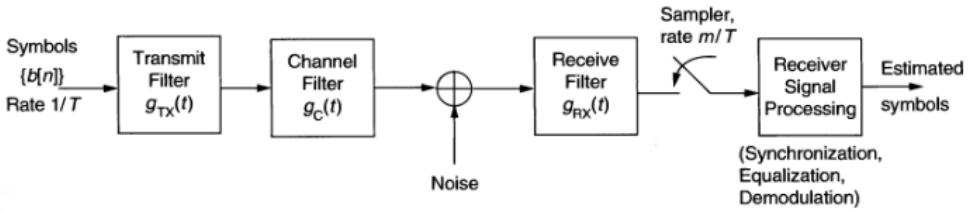


Figure 1: Block diagram of a linearly modulated system, modeled in complex baseband

1 Linear Modulation Modeled with two dimensional constellation

1.1 N/A

1.2 Generating random bits

```
function [output] = randbit(numBits, size)
% randbit generates random bits taking values in {0,1}
output = (sign(rand(numBits,1) -.5)+1)/2;
output = reshape(output, size);
end
```

1.3 Mapping bits to symbols for different signal constellations

```
function [output_vector] = bpskmap(input_binary_vector)
%BPSKMAP Maps binary bits to BPSK symbols
symbol_bits = reshape(input_binary_vector.', 1, [])';
output_vector = (symbol_bits*2 -1);
```

```
function [output_vector] = qpskmap(input_binary_vector)
%QPSKMAP Maps binary bits to QPSK symbols
assert(mod(numel(input_binary_vector), 2) == 0,'qpskmap:invalidInputSize','Sequence
    length is not divisible by 2')
symbol_bits = reshape(input_binary_vector.', 2, [])';
output_real = (1/sqrt(1))*(symbol_bits(:,1)*2 -1);
output_imag = (1/sqrt(1))*(symbol_bits(:,2)*2 -1)*1i;
output_vector = output_real + output_imag;
```

```
function [output_vector] = fourpammap(input_binary_vector)
%4PAMMAP maps binary bits to 4PAM symbols
assert(mod(numel(input_binary_vector), 2) == 0,'fourpammap:invalidInputSize','Sequence
    length is not divisible by 2')
symbol_bits = reshape(input_binary_vector.', 2, [])';
output_vector = (symbol_bits(:,1)*2 -1).*(symbol_bits(:,2)*2 + 1);
```

```
function [output_vector] = sixteenqammap(input_binary_vector)
%SIXTEENQAMMAP maps binary bits to 16QAM symbols
assert(mod(numel(input_binary_vector), 4) ==
    0,'sixteenqammap:invalidInputSize','Sequence length is not divisible by 4')
symbol_bits = reshape(input_binary_vector.', 4, [])';
output_real = (symbol_bits(:,1)*2 -1).*(symbol_bits(:,3)*2 + 1);
output_imag = (symbol_bits(:,2)*2 -1).*(symbol_bits(:,4)*2 + 1)*1i;
output_vector = output_real + output_imag;
```

```
function [output_vector] = eightpskmap(input_binary_vector)
%8PSKMAP Maps binary bits to 8PSK symbols
assert(mod(numel(input_binary_vector), 3) == 0,'eightpskmap:invalidInputSize','Sequence
    length is not divisible by 3');
symbol_bits = reshape(input_binary_vector.', 3, [])';
output_real = (1/sqrt(2))*(symbol_bits(:,2)*2 -1);
output_imag = (1/sqrt(2))*(symbol_bits(:,1)*2 -1)*1i;
output_vector = output_real + output_imag;
output_vector = output_vector.*exp(1i.*(-pi/8+pi/8.*exp(1i*pi.*sum(symbol_bits')')));
```

1.4 Transmitting BPSK symbols through a noiseless system

```
% CREATE SRCC TRANSMITTER AND RECEIVER FILTERS
oversampling_factor = 4;
m = oversampling_factor;
%parameters for sampled raised cosine pulse
a = 0.22;
length_RC = 5;% (truncated outside [-length*T,length*T])
[transmit_filter, transmit_filter_time] = sqrtraised_cosine(a,m,length_RC);
receive_filter = transmit_filter(end:-1:1); % matched filter
receive_filter_time = transmit_filter_time;

ts=1/4; % Normalized sampling interval T = 1

signal_timedomain = transmit_filter; %sinusoidal pulse in our example
fs_desired = 1/64; %desired frequency granularity
Nmin = ceil(1/(fs_desired*ts)); %minimum length DFT for desired frequency granularity
%for efficient computation, choose FFT size to be power of 2
Nfft = 2^(nextpow2(Nmin)); %FFT size = the next power of 2 at least as big as Nmin
%Alternatively, one could also use DFT size equal to the minimum length
%Nfft=Nmin;
%note: fft function in Matlab is just the DFT when Nfft is not a power of 2
%freq domain signal computed using DFT
%fft function of size Nfft automatically zeropads as needed
signal_freqdomain = ts*fft(signal_timedomain,Nfft);
%fftshift function shifts DC to center of spectrum
signal_freqdomain_centered = fftshift(signal_freqdomain);
fs=1/(Nfft*ts); %actual frequency resolution attained
%set of frequencies for which Fourier transform has been computed using DFT
freqs = ((1:Nfft)-1-Nfft/2)*fs;


%% GENERATE 12,000 RANDOM BITS, MAP TO BPSK, AND SEND THROUGH SYSTEM

%NUMBER OF SYMBOLS
nsymbols = 12000; % Binary 0 mapped to -1, Binary 1 mapped to +1
bit_sequence = randbit(12000, [12000 1]);
%BPSK SYMBOL GENERATION
bpsk_symbols = bpskmap(bit_sequence);

%UPSAMPLE BY m
nsymbols_upsampled = 1+(nsymbols-1)*m;%length of upsampled symbol sequence
symbols_upsampled = zeros(nsymbols_upsampled,1);%initialize
symbols_upsampled(1:m:nsymbols_upsampled)=bpsk_symbols;%insert symbols with spacing m
t_s = receive_filter_time(end)-receive_filter_time(end-1);
symbols_upsampled_time = (0:nsymbols_upsampled)*t_s;

%NOISELESS MODULATED SIGNAL
[tx_output, tx_output_time] = contconv2(symbols_upsampled,transmit_filter,
    symbols_upsampled_time(1),transmit_filter_time(1), ts);
tx_output = m*tx_output; % Scale by m
[rx_output, rx_output_time] = contconv2(tx_output, receive_filter, tx_output_time(1),
    receive_filter_time(1), ts);
```

1.5 Adding white Gaussian Noise to receiver input

We will be adding independent and identically distributed (i.i.d.) complex Gaussian noise at the receiver input. We will determine variance for the Gaussian noise corresponding to a signal to noise ratio (SNR) given by the ideal bit error probability 0.01.

1.6 Determine SNR corresponding to ideal bit error

We will plot bit probability error in logarithmic scale as a function of signal to noise ratio given by Eb/No in dB, and find the corresponding signal to noise ratio for an ideal probability of 0.01.

```
snrdb = 0:0.01:10; %vector of SNRs (in dB) for which to evaluate error probability
snr = 10.^(snrdb/10); %vector of raw SNRs
pe = qfunction(sqrt(2*snr)); %vector of error probabilities
semilogy(snrdb,pe);
ylabel('Error Probability');
xlabel('SNR (dB)') ;
[closest_error, snr_db_index] = min(abs(pe-0.01));
ideal_snr_db = snrdb(snr_db_index);
fprintf("The SNR corresponding to 0.01 probability error is " + ideal_snr_db + "dB\n");
```

```
The SNR corresponding to 0.01 probability error is 4.32dB
```

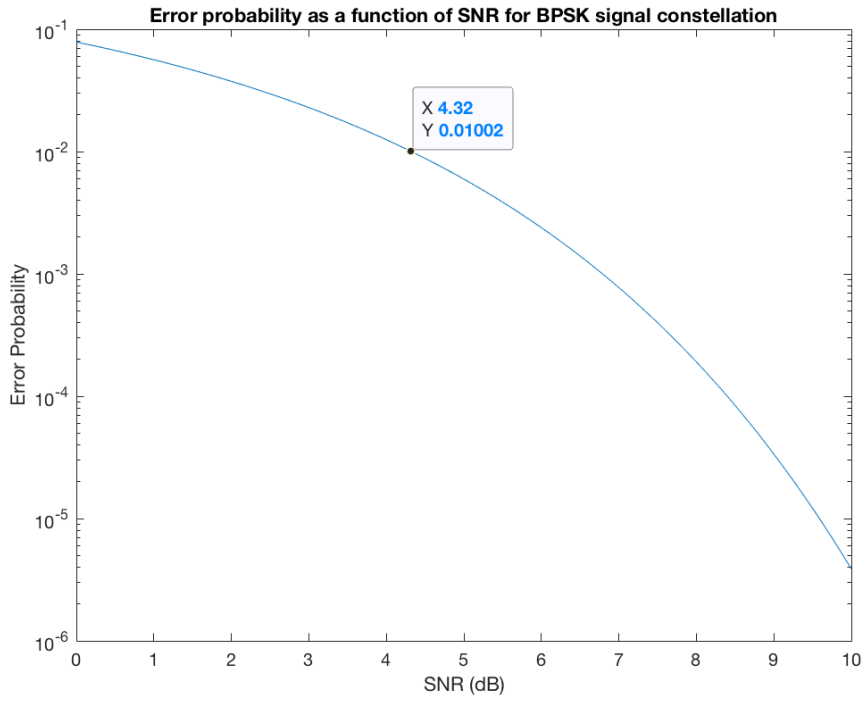


Figure 2: Bit error probability as a function of SNR

1.7 Find decision statistics for BPSK at receiver input and output

We first need to determine the variance to generate our Gaussian noise based on the measured SNR from Figure 2. We can then pass our signal through the transmitter, add noise and then pass the noisy signal through the receiver. Finally we can sample the received signal and plot the decision statistics.

```
ideal_snr_raw = 10.^(ideal_snr_db/10); % Raw SNR from SNR in dB.
bpsk_variance = m/(2*ideal_snr_raw);

mean = 0;
size = [nsymbols_upsampled + 40,1]; % 40 extra samples from transmitter convolution
noise_real = normrnd(mean, sqrt(bpsk_variance), size);
noise_imag = normrnd(mean, sqrt(bpsk_variance), size)*1i;
noise = noise_real + noise_imag;

% ADD NOISE AT THE TRANSMITTER OUTPUT
tx_output_with_noise = tx_output + noise;

% DECISION STATISTICS AT RECEIVER INPUT

% Calculate delay and sampling space
start_index_of_sequence_tx = find(tx_output_time==0);
end_index_of_sequence_tx = find(tx_output_time==nsymbols-1);

select_samples_tx =
    tx_output_with_noise(start_index_of_sequence_tx:m:end_index_of_sequence_tx);
figure('Name', 'Section 1.7 Receiver Input');
plot(select_samples_tx,'.');
title('Transmitter Output'); ylabel('Imaginary'); xlabel('Real');
grid on; daspect([1 1 1]);

% DECISION STATISTICS AT RECEIVER OUTPUT

% Calculate delay and sampling space
[rx_output_noise, rx_output_noise_time] = contconv2(noise, receive_filter,
    symbols_upsampled_time(1), receive_filter_time(1), ts);
% Pass noise through receiver and add to noiseless symbols at
% receiver output.
rx_output_total = rx_output + rx_output_noise;

start_index_of_sequence_rx = find(rx_output_time==0);
end_index_of_sequence_rx = find(rx_output_time==nsymbols-1);

select_samples_rx =
    rx_output_total(start_index_of_sequence_rx:m:end_index_of_sequence_rx);
figure('Name', 'Section 1.7 Receiver Output');
plot(select_samples_rx,'.');
title('Transmitter Output'); ylabel('Imaginary'); xlabel('Real');
grid on; daspect([1 1 1]);
```

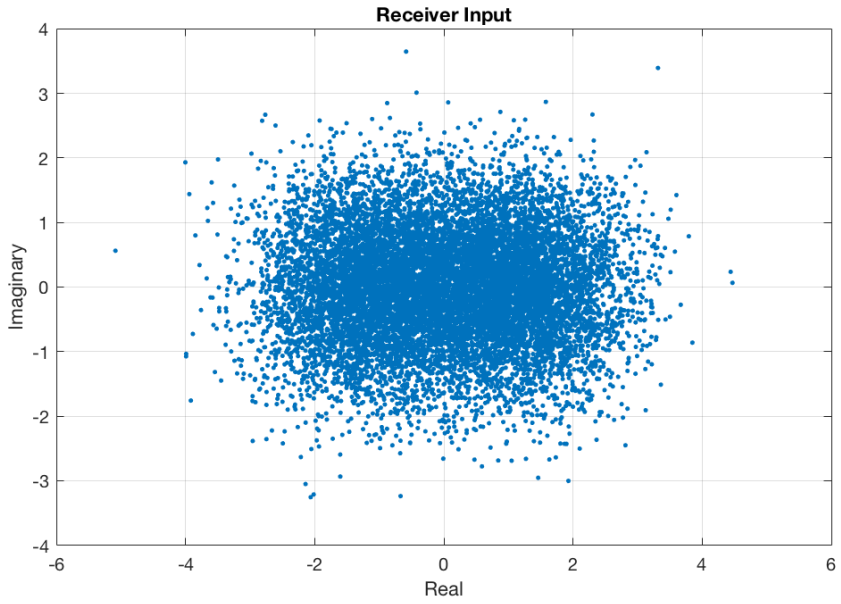


Figure 3: Receiver input decision statistics

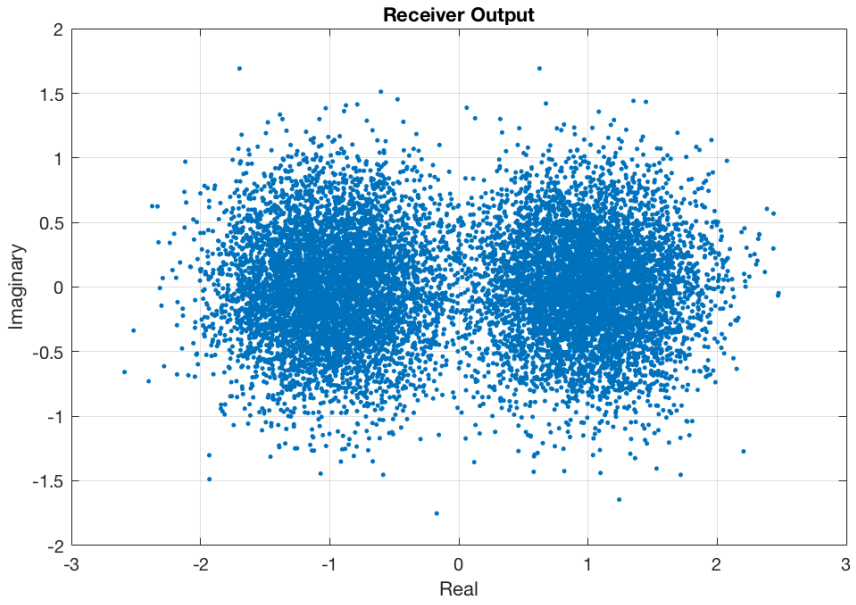


Figure 4: Receiver output decision statistics

Observing Figure 3 and 5, we notice that the constellation is noisier at the receiver input than at the output. We are able to clearly distinguish decision statistics at the receiver output.

1.8 Measuring error probability

```
% RECOVER NOISY SAMPLES AT RECEIVER INPUT
recoverd_samples_tx = sign(real(select_samples_tx));
recovered_bit_sequence_tx = (recoverd_samples_tx + 1)./2;
error_tx = 1*(1/nsymbols)*sum(abs(bit_sequence-recovered_bit_sequence_tx));
fprintf("The probability error measured at the receiver input was " + error_tx + "\n");
fprintf("This compares to the ideal error probability by " + (error_tx-0.01) + "\n");


% RECOVER NOISY SAMPLES AT RECEIVER INPUT
recoverd_samples_rx = sign(real(select_samples_rx));
recovered_bit_sequence_rx = (recoverd_samples_rx + 1)./2;
error_rx = 1*(1/nsymbols)*sum(abs(bit_sequence-recovered_bit_sequence_rx));
fprintf("The probability error measured at the receiver output was " + error_rx + "\n");
fprintf("This compares to the ideal error probability by " + (error_rx-0.01) + "\n");
fprintf("This probability error is " + (error_tx-0.01)/error_rx + " times smaller than
    that found at the receiver input\n");
```

```
The probability error measured at the receiver input was 0.11242
This compares to the ideal error probability by 0.10242
The probability error measured at the receiver output was 0.010667
This compares to the ideal error probability by 0.00066667
This probability error is 9.6016 times smaller than that found at the receiver input
```

1.9 Ideal SNR in 4PAM constellation

```
% GENERATE 12,000 RANDOM BITS AS 2 PARALLEL VECTORS OF 6000 BITS AND MAP TO 4PAM

%NUMBER OF SYMBOLS
nsymbols = 6000; % Binary 0 mapped to -1, Binary 1 mapped to +1
bit_sequence = randbit(12000, [6000 2]);
%4PAM SYMBOL GENERATION
fourpam_symbols = fourpammap(bit_sequence);

%PLOT ERROR PROBABILITY IN LOG SCALE VS. SNR IN DB.
snrdb = 0:0.01:10; %vector of SNRs (in dB) for which to evaluate error probability
snr = 10.^(snrdb/10); %vector of raw SNRs
%vector of error probabilities
pe = qfunction(sqrt((4/5)*snr)); %Book with sqrt inside as suggested by TA
% pe = (3/4)*qfunction(sqrt((4)*snr)); %TA derived BER
% pe = qfunction((4/5)*snr); %Unaltered book equation

semilogy(snrdb,pe);
ylabel('Error Probability');
xlabel('SNR (dB)') ;
[closest_error, snr_db_index] = min(abs(pe-0.01));
```

```
ideal_snr_db = snrdb(snr_db_index);

%
fprintf("The SNR corresponding to 0.01 probability error is " + ideal_snr_db + "dB\n");
```

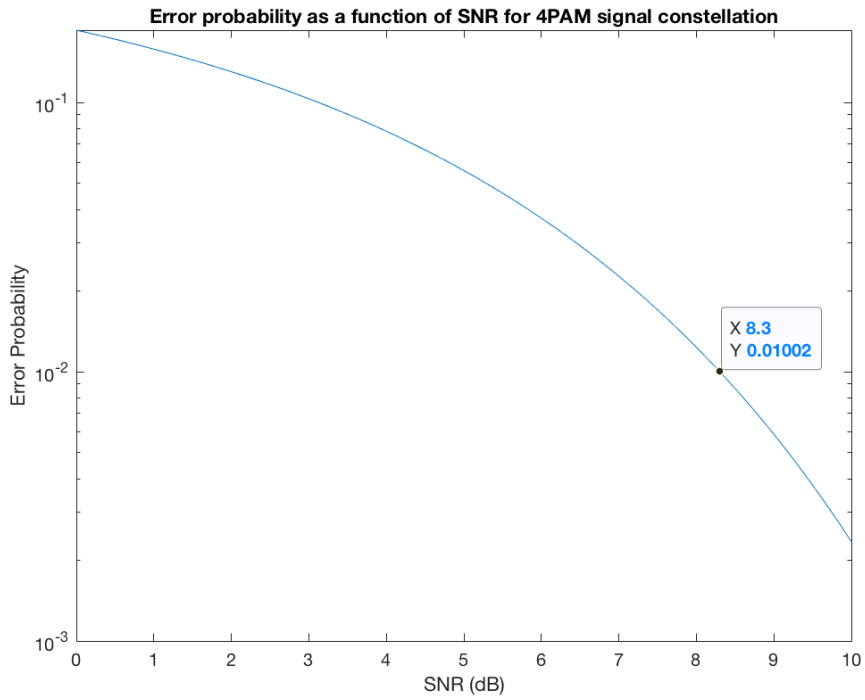


Figure 5: Receiver output decision statistics

```
The SNR corresponding to 0.01 probability error is 8.3dB
```

1.10 Decision statistics and error probability of a 4PAM constellation

```
FIND VARIANCE FROM IDEAL SNR FOUND PREVIOUSLY
ideal_snr_raw = 10.^(ideal_snr_db/10); % Raw SNR from SNR in dB.
fourpam_variance = (5*m)/(4*ideal_snr_raw);

%%%%% Ta sigma function
% fourpam_variance = ((sqrt(m))/(sqrt(2)*erfcinv(0.01*(4/3))))^2;
%%%%%

%UPSAMPLE MAPPED SEQUENCE BY m
nsymbols_upsampled = 1+(nsymbols-1)*m;%length of upsampled symbol sequence
symbols_upsampled = zeros(nsymbols_upsampled,1);%initialize
symbols_upsampled(1:m:nsymbols_upsampled)=fourpam_symbols;%insert symbols with spacing m
t_s = receive_filter_time(end)-receive_filter_time(end-1);
symbols_upsampled_time = (0:nsymbols_upsampled)*t_s;
```

```
% GENERATE NOISE VECTOR
mean = 0;
size = [nsymbols_upsampled + 40,1]; % 40 extra samples from transmitter convolution
noise_real = normrnd(mean, sqrt(fourpam_variance), size);
noise_imag = normrnd(mean, sqrt(fourpam_variance), size)*1i;
noise = noise_real + noise_imag;

%NOISELESS MODULATED SIGNAL THROUGH TRANSMITTER
[tx_output, tx_output_time] = contconv2(symbols_upsampled,transmit_filter,
    symbols_upsampled_time(1),transmit_filter_time(1), ts);
tx_output = m*tx_output; % Scale by m

% ADD NOISE AT THE TRANSMITTER OUTPUT
tx_output_with_noise = tx_output + noise;

% PASS NOISY SIGNAL THROUGH TRANSMITTER
[rx_output_with_noise, rx_output_with_noise_time] = contconv2(tx_output_with_noise,
    receive_filter, tx_output_time(1), receive_filter_time(1), ts);

% RECOVER NOISY SYMBOLS AT RECEIVER OUTPUT AND PLOT DECISION STATISTICS
start_index_of_sequence_rx = find(rx_output_with_noise_time==0);
end_index_of_sequence_rx = find(rx_output_with_noise_time==nsymbols-1);

select_samples_rx =
    rx_output_with_noise(start_index_of_sequence_rx:m:end_index_of_sequence_rx);
figure('Name', 'Section 1.10 Receiver Output variance from book without sqrt inside');
plot(select_samples_rx,'.');
title('Decision Statistics at Receiver Output'); ylabel('Imaginary'); xlabel('Real');
grid on; daspect([1 1 1]);

% ESTIMATE THE TWO PARALLEL BIT STREAMS WITH AN APPROPRIATE DECISION RULE
% AND COMPARE PROBABILITY ERROR AGAINST IDEAL
recovered_bit_stream = [(sign(real(select_samples_rx))+1)/2
    (sign((abs(real(select_samples_rx))/2)-1)+1)/2];
error_rx =
    1*(1/numel(recovered_bit_stream))*sum(sum(abs(bit_sequence-recovered_bit_stream)));
fprintf("The probability error measured at the receiver output was " + error_rx + "\n");
fprintf("This compares to the ideal error probability by " + (error_rx-0.01) + "\n");
```

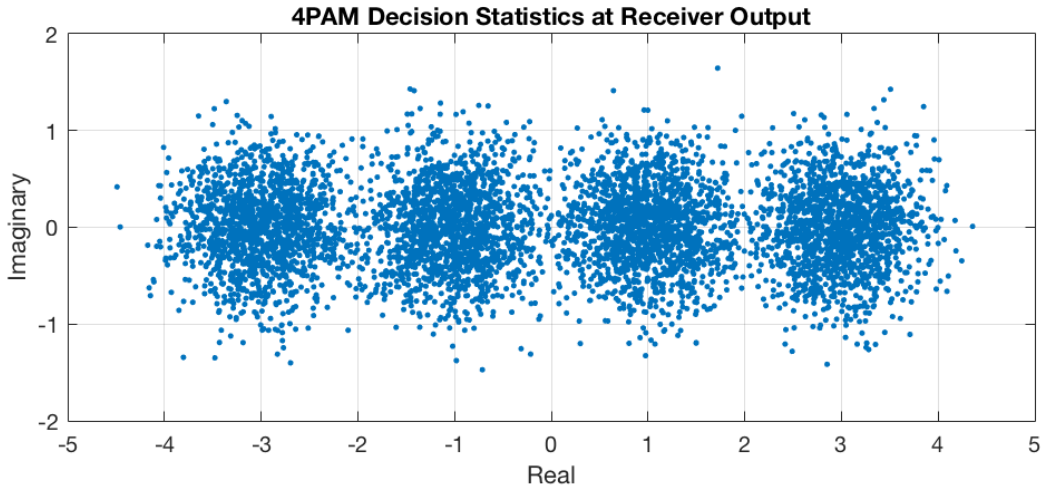


Figure 6: Receiver output decision statistics

```
The probability error measured at the receiver output was 0.0094167
This compares to the ideal error probability by -0.00058333
```

1.11 Decision statistics and error probability of a QPSK constellation

```
% GENERATE 12,000 RANDOM BITS AS 2 PARALLEL VECTORS OF 6000 BITS AND MAP TO QPSK

%NUMBER OF SYMBOLS
nsymbols = 6000; % Binary 0 mapped to -1, Binary 1 mapped to +1
bit_sequence = randbit(12000, [6000 2]);
%4PAM SYMBOL GENERATION
qpsk_symbols = qpskmap(bit_sequence);

%PLOT ERROR PROBABILITY IN LOG SCALE VS. SNR IN DB.
snrdb = 0:0.01:10; %vector of SNRs (in dB) for which to evaluate error probability
snr = 10.^(snrdb/10); %vector of raw SNRs
%vector of error probabilities
pe = qfunction(sqrt(2*snr));
```

```
semilogy(snrdb,pe);
ylabel('Error Probability');
xlabel('SNR (dB)') ;
[closest_error, snr_db_index] = min(abs(pe-0.01));
ideal_snr_db = snrdb(snr_db_index);

%
fprintf("The SNR corresponding to 0.01 probability error is " + ideal_snr_db + "dB\n");

% FIND VARIANCE FROM IDEAL SNR FOUND PREVIOUSLY
ideal_snr_raw = 10.^(ideal_snr_db/10); % Raw SNR from SNR in dB.
qpsk_variance = m/(2*ideal_snr_raw);

%%%%% Ta sigma function
% fourpam_variance = ((sqrt(m))/(sqrt(2)*erfcinv(0.01*)))^2;
%%%%%

%UPSAMPLE MAPPED SEQUENCE BY m
nsymbols_upsampled = 1+(nsymbols-1)*m;%length of upsampled symbol sequence
symbols_upsampled = zeros(nsymbols_upsampled,1);%initialize
symbols_upsampled(1:m:nsymbols_upsampled)=qpsk_symbols;%insert symbols with spacing m
t_s = receive_filter_time(end)-receive_filter_time(end-1);
symbols_upsampled_time = (0:nsymbols_upsampled)*t_s;

% GENERATE NOISE VECTOR
mean = 0;
size = [nsymbols_upsampled + 40,1]; % 40 extra samples from transmitter convolution
noise_real = normrnd(mean, sqrt(qpsk_variance), size);
noise_imag = normrnd(mean, sqrt(qpsk_variance), size)*1i;
noise = noise_real + noise_imag;

%NOISELESS MODULATED SIGNAL THROUGH TRANSMITTER
[tx_output, tx_output_time] = contconv2(symbols_upsampled,transmit_filter,
    symbols_upsampled_time(1),transmit_filter_time(1), ts);
tx_output = m*tx_output; % Scale by m

% ADD NOISE AT THE TRANSMITTER OUTPUT
tx_output_with_noise = tx_output + noise;

% PASS NOISY SIGNAL THROUGH TRANSMITTER
[rx_output_with_noise, rx_output_with_noise_time] = contconv2(tx_output_with_noise,
    receive_filter, tx_output_time(1), receive_filter_time(1), ts);

% RECOVER NOISY SYMBOLS AT RECEIVER OUTPUT AND PLOT DECISION STATISTICS
start_index_of_sequence_rx = find(rx_output_with_noise_time==0);
end_index_of_sequence_rx = find(rx_output_with_noise_time==nsymbols-1);

select_samples_rx =
    rx_output_with_noise(start_index_of_sequence_rx:m:end_index_of_sequence_rx);
figure('Name', 'Section 1.10 Receiver Output');
plot(select_samples_rx,'.');
title('Decision Statistics at Receiver Output'); ylabel('Imaginary'); xlabel('Real');
grid on; daspect([1 1 1]);
```

```
% ESTIMATE THE TWO PARALLEL BIT STREAMS WITH AN APPROPRIATE DECISION RULE
% AND COMPARE PROBABILITY ERROR AGAINST IDEAL
recovered_bit_stream = [(sign(real(select_samples_rx))+1)/2
    (sign(imag(select_samples_rx))+1)/2];
error_rx =
    1*(1/numel(recovered_bit_stream))*sum(sum(abs(bit_sequence-recovered_bit_stream)));
fprintf("The probability error measured at the receiver output was " + error_rx + "\n");
fprintf("This compares to the ideal error probability by " + (error_rx-0.01) + "\n");
```

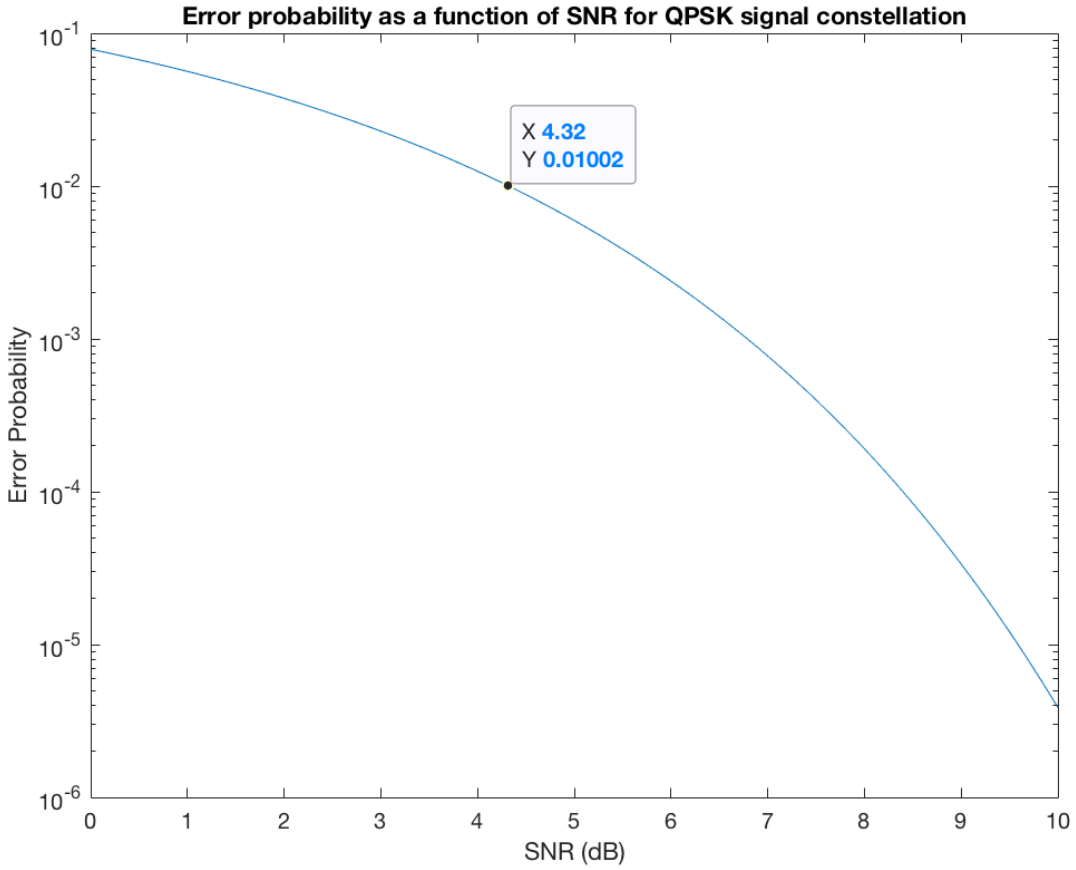


Figure 7: QPSK Bit error probability as a function of SNR

```
The SNR corresponding to 0.01 probability error is 4.32dB
```

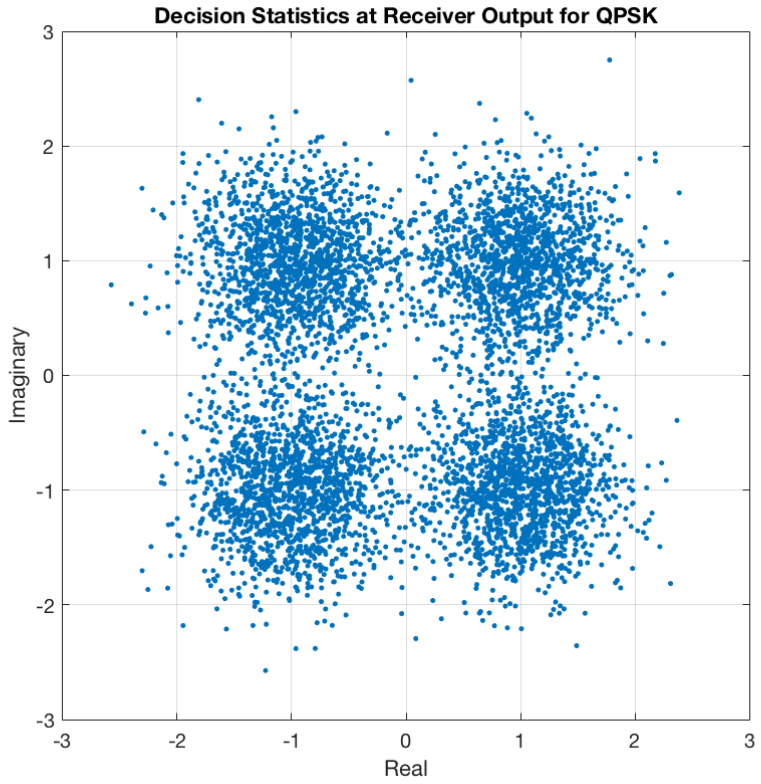


Figure 8: QPSK decision statistics

```
The probability error measured at the receiver output was 0.0091667
This compares to the ideal error probability by -0.00083333
```

1.12 Decision statistics and error probability of a 16QAM constellation

```
% GENERATE 12,000 RANDOM BITS AS 4 PARALLEL VECTORS OF 3000 BITS AND MAP TO 16QAM

%NUMBER OF SYMBOLS
nsymbols = 3000;
bit_sequence = randbit(12000, [3000 4]);
%16QAM SYMBOL GENERATION
sixteenqam_symbols = sixteenqammap(bit_sequence);

%PLOT ERROR PROBABILITY IN LOG SCALE VS. SNR IN DB.
snrdb = 0:0.01:10; %vector of SNRs (in dB) for which to evaluate error probability
snr = 10.^(snrdb/10); %vector of raw SNRs
%vector of error probabilities
pe = qfunction(sqrt((4/5)*snr));

semilogy(snrdb,pe);
ylabel('Error Probability');
xlabel('SNR (dB)') ;
```

```
[closest_error, snr_db_index] = min(abs(pe-0.01));
ideal_snr_db = snrdb(snr_db_index);

%
fprintf("The SNR corresponding to 0.01 probability error is " + ideal_snr_db + "dB\n");

% FIND VARIANCE FROM IDEAL SNR FOUND PREVIOUSLY
ideal_snr_raw = 10.^(ideal_snr_db/10); % Raw SNR from SNR in dB.
sixteenqam_variance = (5*m)/(4*ideal_snr_raw); % Same as 4PAM

%%%%% Ta sigma function
% fourpam_variance = ((sqrt(m))/(sqrt(2)*erfcinv(0.01*)))^2;
%%%%%

%UPSAMPLE MAPPED SEQUENCE BY m
nsymbols_upsampled = 1+(nsymbols-1)*m;%length of upsampled symbol sequence
symbols_upsampled = zeros(nsymbols_upsampled,1);%initialize
symbols_upsampled(1:m:nsymbols_upsampled)=sixteenqam_symbols;%insert symbols with
    spacing m
t_s = receive_filter_time(end)-receive_filter_time(end-1);
symbols_upsampled_time = (0:nsymbols_upsampled)*t_s;

% GENERATE NOISE VECTOR
mean = 0;
size = [nsymbols_upsampled + 40,1]; % 40 extra samples from transmitter convolution
noise_real = normrnd(mean, sqrt(sixteenqam_variance), size);
noise_imag = normrnd(mean, sqrt(sixteenqam_variance), size)*1i;
noise = noise_real + noise_imag;

%NOISELESS MODULATED SIGNAL THROUGH TRANSMITTER
[tx_output, tx_output_time] = contconv2(symbols_upsampled,transmit_filter,
    symbols_upsampled_time(1),transmit_filter_time(1), ts);
tx_output = m*tx_output; % Scale by m

% ADD NOISE AT THE TRANSMITTER OUTPUT
tx_output_with_noise = tx_output + noise;

% PASS NOISY SIGNAL THROUGH TRANSMITTER
[rx_output_with_noise, rx_output_with_noise_time] = contconv2(tx_output_with_noise,
    receive_filter, tx_output_time(1), receive_filter_time(1), ts);

% RECOVER NOISY SYMBOLS AT RECEIVER OUTPUT AND PLOT DECISION STATISTICS
start_index_of_sequence_rx = find(rx_output_with_noise_time==0);
end_index_of_sequence_rx = find(rx_output_with_noise_time==nsymbols-1);

select_samples_rx =
    rx_output_with_noise(start_index_of_sequence_rx:m:end_index_of_sequence_rx);
figure('Name', 'Section 1.10 Receiver Output');
plot(select_samples_rx,'.');
title('Decision Statistics at Receiver Output'); ylabel('Imaginary'); xlabel('Real');
grid on; daspect([1 1 1]);

% ESTIMATE THE TWO PARALLEL BIT STREAMS WITH AN APPROPRIATE DECISION RULE
% AND COMPARE PROBABILITY ERROR AGAINST IDEAL
```

```
% MAP 16QAM SYMBOLS BACK TO BITS
first_bit = (sign(real(select_samples_rx))+1)/2;
second_bit = ((sign(imag(select_samples_rx))+1)/2);
third_bit = (sign((abs(real(select_samples_rx))/2)-1)+1)/2;
fourth_bit = ((sign((abs(imag(select_samples_rx))/2)-1)+1)/2);
recovered_bit_stream = [first_bit second_bit third_bit fourth_bit];

error_rx =
    1*(1/numel(recovered_bit_stream))*sum(sum(abs(bit_sequence-recovered_bit_stream)));
fprintf("The probability error measured at the receiver output was " + error_rx + "\n");
fprintf("This compares to the ideal error probability by " + (error_rx-0.01) + "\n");
```

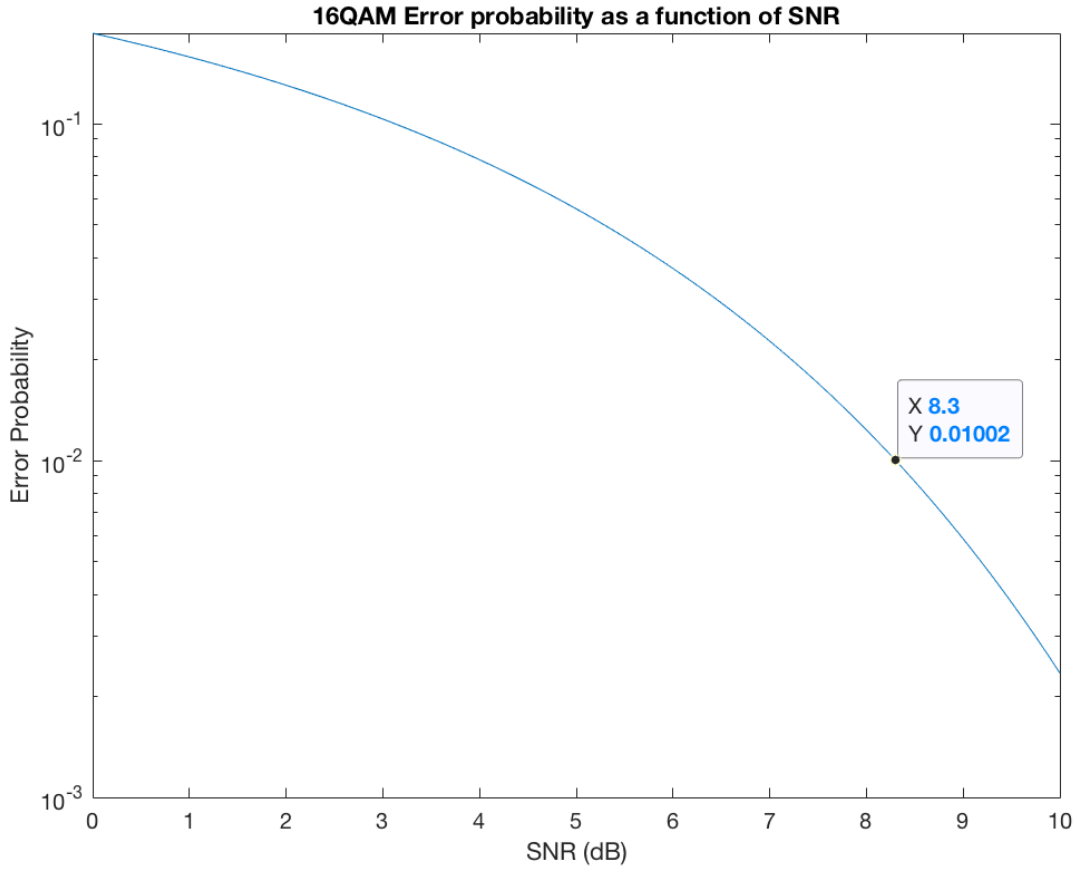


Figure 9: 16QAM Bit error probability as a function of SNR

```
The SNR corresponding to 0.01 probability error is 8.3dB
```

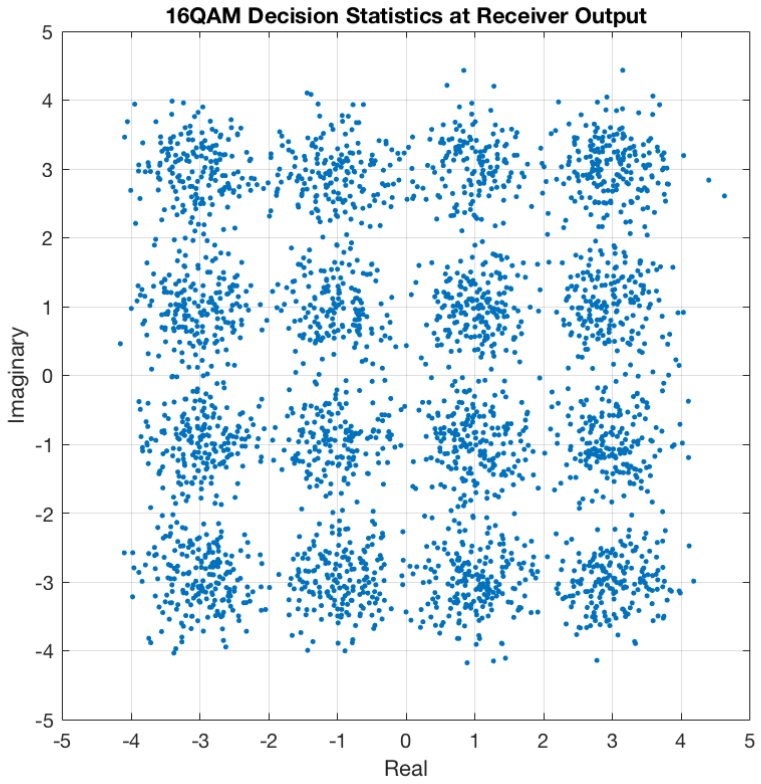


Figure 10: 16QAM decision statistics

```
The probability error measured at the receiver output was 0.00875
This compares to the ideal error probability by -0.00125
>>
```

1.13 Decision statistics and error probability of a 8PSK constellation

```
% GENERATE 12,000 RANDOM BITS AS 3 PARALLEL VECTORS OF 4000 BITS AND MAP TO 8PSK

%NUMBER OF SYMBOLS
nsymbols = 4000;
bit_sequence = randbit(12000, [4000 3]);
%8PSK SYMBOL GENERATION
eightpsk_symbols = eightpskmap(bit_sequence);

%PLOT ERROR PROBABILITY IN LOG SCALE VS. SNR IN DB.
snrdb = 0:0.01:10; %vector of SNRs (in dB) for which to evaluate error probability
snr = 10.^(snrdb/10); %vector of raw SNRs
%vector of error probabilities

pe = qfunction(sqrt((1/2)*(6-3*sqrt(2))*snr));

semilogy(snrdb,pe);
```

```
ylabel('Error Probability');
xlabel('SNR (dB)') ;
[closest_error, snr_db_index] = min(abs(pe-0.01));
ideal_snr_db = snrdb(snr_db_index);

%
fprintf("The SNR corresponding to 0.01 probability error is " + ideal_snr_db + "dB\n");

% FIND VARIANCE FROM IDEAL SNR FOUND PREVIOUSLY
ideal_snr_raw = 10.^(ideal_snr_db/10); % Raw SNR from SNR in dB.
eightpsk_variance = m/(6*ideal_snr_raw);

%%%%% Ta sigma function
% fourpam_variance = ((sqrt(m))/(sqrt(2)*erfcinv(0.01*)))^2;
%%%%%

%UPSAMPLE MAPPED SEQUENCE BY m
nsymbols_upsampled = 1+(nsymbols-1)*m;%length of upsampled symbol sequence
symbols_upsampled = zeros(nsymbols_upsampled,1);%initialize
symbols_upsampled(1:m:nsymbols_upsampled)=eightpsk_symbols;%insert symbols with spacing m
t_s = receive_filter_time(end)-receive_filter_time(end-1);
symbols_upsampled_time = (0:nsymbols_upsampled)*t_s;

% GENERATE NOISE VECTOR
mean = 0;
size = [nsymbols_upsampled + 40,1]; % 40 extra samples from transmitter convolution
noise_real = normrnd(mean, sqrt(eightpsk_variance), size);
noise_imag = normrnd(mean, sqrt(eightpsk_variance), size)*1i;
noise = noise_real + noise_imag;

%NOISELESS MODULATED SIGNAL THROUGH TRANSMITTER
[tx_output, tx_output_time] = contconv2(symbols_upsampled,transmit_filter,
    symbols_upsampled_time(1),transmit_filter_time(1), ts);
tx_output = m*tx_output; % Scale by m

% ADD NOISE AT THE TRANSMITTER OUTPUT
tx_output_with_noise = tx_output + noise;

% PASS NOISY SIGNAL THROUGH TRANSMITTER
[rx_output_with_noise, rx_output_with_noise_time] = contconv2(tx_output_with_noise,
    receive_filter, tx_output_time(1), receive_filter_time(1), ts);

% RECOVER NOISY SYMBOLS AT RECEIVER OUTPUT AND PLOT DECISION STATISTICS
start_index_of_sequence_rx = find(rx_output_with_noise_time==0);
end_index_of_sequence_rx = find(rx_output_with_noise_time==nsymbols-1);

select_samples_rx =
    rx_output_with_noise(start_index_of_sequence_rx:m:end_index_of_sequence_rx);
figure('Name', 'Section 1.10 Receiver Output');
plot(select_samples_rx,'.');
title('Decision Statistics at Receiver Output'); ylabel('Imaginary'); xlabel('Real');
grid on; daspect([1 1 1]);

% ESTIMATE THE TWO PARALLEL BIT STREAMS WITH AN APPROPRIATE DECISION RULE
```

```
% AND COMPARE PROBABILITY ERROR AGAINST IDEAL

% MAP 8PSK SYMBOLS BACK TO BITS
recovered_bit_stream = zeros(nsymbols, 3);
for i=1:nsymbols
    if (angle(select_samples_rx(i)) < pi/8) && (angle(select_samples_rx(i)) >= -pi/8)
        recovered_bit_stream(i, :) = [1 1 1];
    elseif (angle(select_samples_rx(i)) < 3*pi/8) && sign(angle(select_samples_rx(i)) >
        0)
        recovered_bit_stream(i, :) = [1 1 0];
    elseif (angle(select_samples_rx(i)) < 5*pi/8) && sign(angle(select_samples_rx(i)) >
        0)
        recovered_bit_stream(i, :) = [1 0 0];
    elseif (angle(select_samples_rx(i)) < 7*pi/8) && sign(angle(select_samples_rx(i)) >
        0)
        recovered_bit_stream(i, :) = [1 0 1];
    elseif (angle(select_samples_rx(i)) < -7*pi/8) || (angle(select_samples_rx(i)) >=
        7*pi/8)
        recovered_bit_stream(i, :) = [0 0 1];
    elseif (angle(select_samples_rx(i)) < -5*pi/8)
        recovered_bit_stream(i, :) = [0 0 0];
    elseif (angle(select_samples_rx(i)) < -3*pi/8)
        recovered_bit_stream(i, :) = [0 1 0];
    else
        recovered_bit_stream(i, :) = [0 1 1];
    end
end

error_rx =
    1*(1/numel(recovered_bit_stream))*sum(sum(abs(bit_sequence-recovered_bit_stream)));
fprintf("The probability error measured at the receiver output was " + error_rx + "\n");
fprintf("This compares to the ideal error probability by " + (error_rx-0.01) + "\n");
```

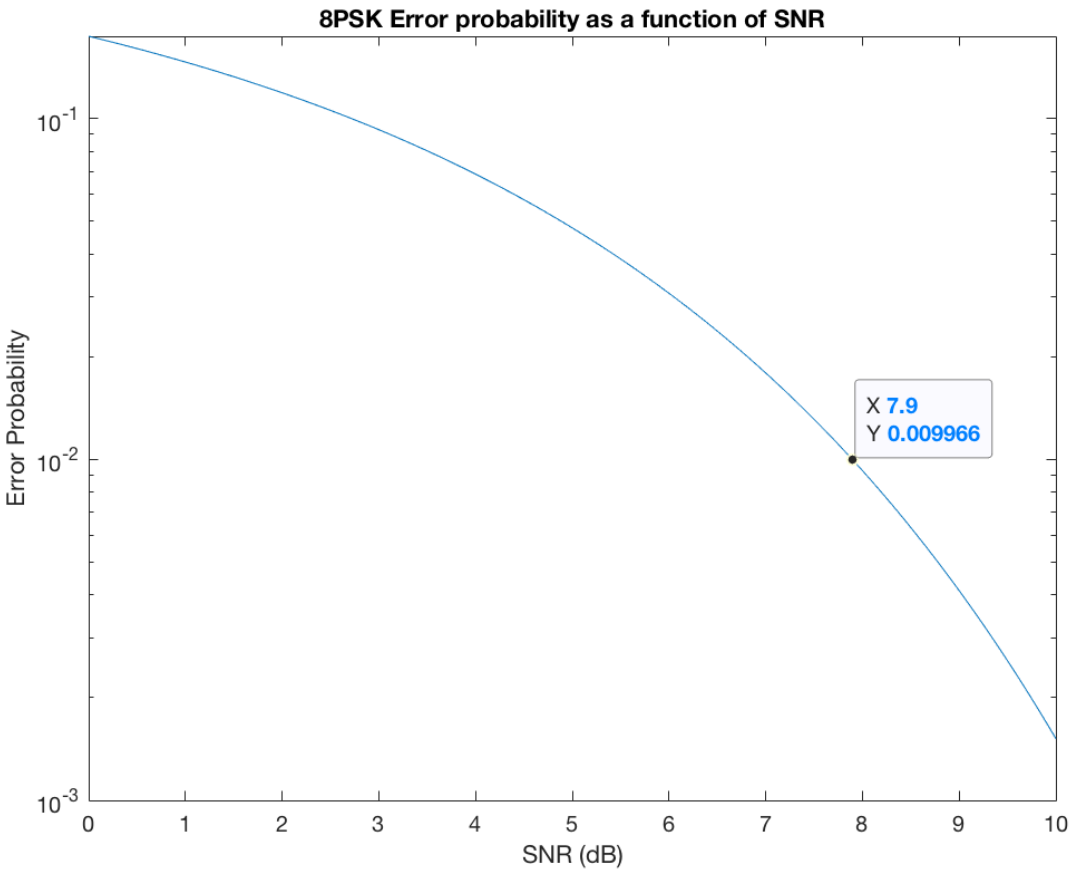


Figure 11: 16QAM Bit error probability as a function of SNR

```
The SNR corresponding to 0.01 probability error is 8.3dB
```

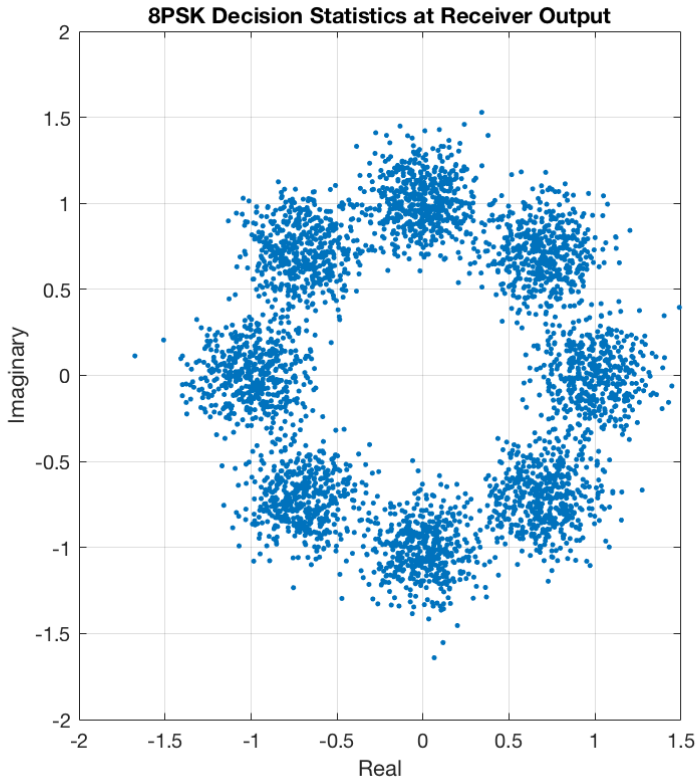


Figure 12: 16QAM decision statistics

```
The probability error measured at the receiver output was 0.0073333
This compares to the ideal error probability by -0.0026667
```

1.14 Decision statistics and error probability of a 16QAM constellation with no ISI

```
% GENERATE 12,000 RANDOM BITS AS 4 PARALLEL VECTORS OF 3000 BITS AND MAP TO 16QAM

%NUMBER OF SYMBOLS
nsymbols = 3000;
bit_sequence = randbit(12000, [3000 4]);
%16QAM SYMBOL GENERATION
sixteenqam_symbols = sixteenqammap(bit_sequence);

%PLOT ERROR PROBABILITY IN LOG SCALE VS. SNR IN DB.
snrdb = 0:0.01:10; %vector of SNRs (in dB) for which to evaluate error probability
snr = 10.^(snrdb/10); %vector of raw SNRs
%vector of error probabilities
pe = (3/4)*qfunction(sqrt((4/5)*snr));

semilogy(snrdb,pe);
```

```
title('16QAM Error probability as a function of SNR');
ylabel('Error Probability');
xlabel('SNR (dB)') ;
[closest_error, snr_db_index] = min(abs(pe-0.01));
ideal_snr_db = snrdb(snr_db_index) + 3; % +3db for part 2

%
fprintf("The SNR corresponding to 0.01 probability error is " + ideal_snr_db + "dB\n");

% FIND VARIANCE FROM IDEAL SNR FOUND PREVIOUSLY
ideal_snr_raw = 10.^(ideal_snr_db/10); % Raw SNR from SNR in dB.
sixteenqam_variance = (5*m)/(4*ideal_snr_raw)/(m^2) % Since going through receiver and
    transmitter, divide by M^2

%%%%% Ta sigma function
% sixteenqam_variance = ((sqrt(m))/(sqrt(2)*erfcinv(2*0.01*4/3)))^2;
%%%%%

%UPSAMPLE MAPPED SEQUENCE BY m
nsymbols_upsampled = 1+(nsymbols-1)*m;%length of upsampled symbol sequence
symbols_upsampled = zeros(nsymbols_upsampled,1);%initialize
symbols_upsampled(1:m:nsymbols_upsampled)=sixteenqam_symbols;%insert symbols with
    spacing m
t_s = receive_filter_time(end)-receive_filter_time(end-1);
symbols_upsampled_time = (0:nsymbols_upsampled)*t_s;

% GENERATE NOISE VECTOR
mean = 0;

% stddev=1/qfuncinv(0.01*4/3)*sqrt(m)*sqrt(m);
size = [nsymbols_upsampled + 0,1]; % 0 extra samples because added at the beggining
noise_real = normrnd(mean, sqrt(sixteenqam_variance), size);
noise_imag = normrnd(mean, sqrt(sixteenqam_variance), size)*1i;
noise = noise_real + noise_imag;

symbols_upsampled = symbols_upsampled + noise;


%NOISELESS MODULATED SIGNAL THROUGH TRANSMITTER AND RECEIVER
[tx_output, tx_output_time] = contconv2(symbols_upsampled,transmit_filter,
    symbols_upsampled_time(1),transmit_filter_time(1), ts);
tx_output = m*tx_output; % Scale by m
[rx_output, rx_output_time] = contconv2(tx_output, receive_filter, tx_output_time(1),
    receive_filter_time(1), ts);

% ADD NOISE AND SIGNAL AT RECEIVER OUTPUT
rx_output_with_noise = rx_output; % B/c copied code from previous


% RECOVER NOISY SYMBOLS AT RECEIVER OUTPUT AND PLOT DECISION STATISTICS
start_index_of_sequence_rx = find(rx_output_time==0);
end_index_of_sequence_rx = find(rx_output_time==nsymbols-1);
```

```
select_samples_rx =
    rx_output_with_noise(start_index_of_sequence_rx:m:end_index_of_sequence_rx);
figure('Name', 'Section 1.10 Receiver Output');
plot(select_samples_rx,'.');
title('Decision Statistics at Receiver Output for 16QAM at 3dB over the SNR
    corresopnding to 0.01 bit error probability');
ylabel('Imaginary'); xlabel('Real');
grid on; daspect([1 1 1]);

% ESTIMATE THE TWO PARALLEL BIT STREAMS WITH AN APPROPRIATE DECISION RULE
% AND COMPARE PROBABILITY ERROR AGAINST IDEAL

% MAP 16QAM SYMBOLS BACK TO BITS
first_bit = (sign(real(select_samples_rx))+1)/2;
second_bit = ((sign(imag(select_samples_rx))+1)/2);
third_bit = (sign((abs(real(select_samples_rx))/2)-1)+1)/2;
fourth_bit = ((sign((abs(imag(select_samples_rx))/2)-1)+1)/2);
recovered_bit_stream = [first_bit second_bit third_bit fourth_bit];

error_rx =
    1*(1/numel(recovered_bit_stream))*sum(sum(abs(bit_sequence-recovered_bit_stream)));
fprintf("The probability error measured at the receiver output was " + error_rx + "\n");
fprintf("This compares to the ideal error probability by " + (error_rx-0.01) + "\n");
```

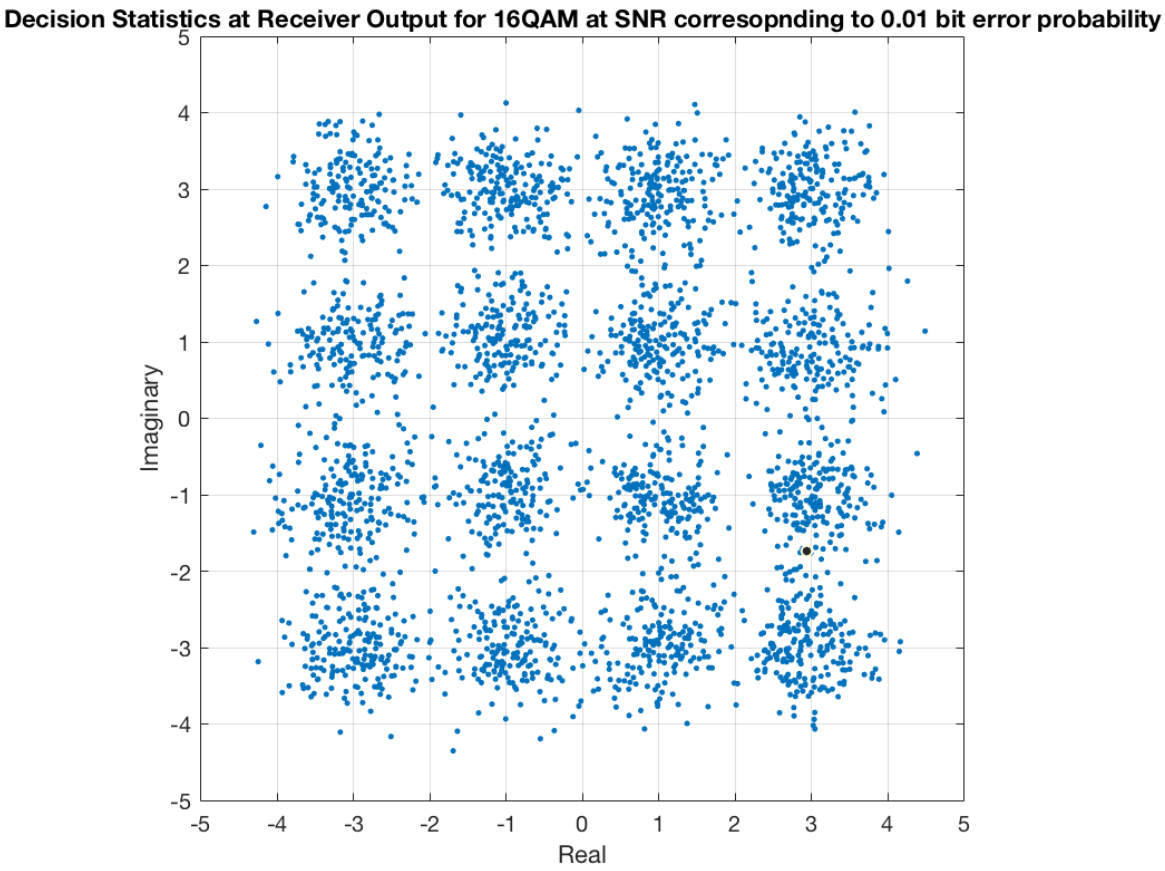


Figure 13: 16QAM Bit error probability as a function of SNR

```
The probability error measured at the receiver output was 0.01
This compares to the ideal error probability by 0
```

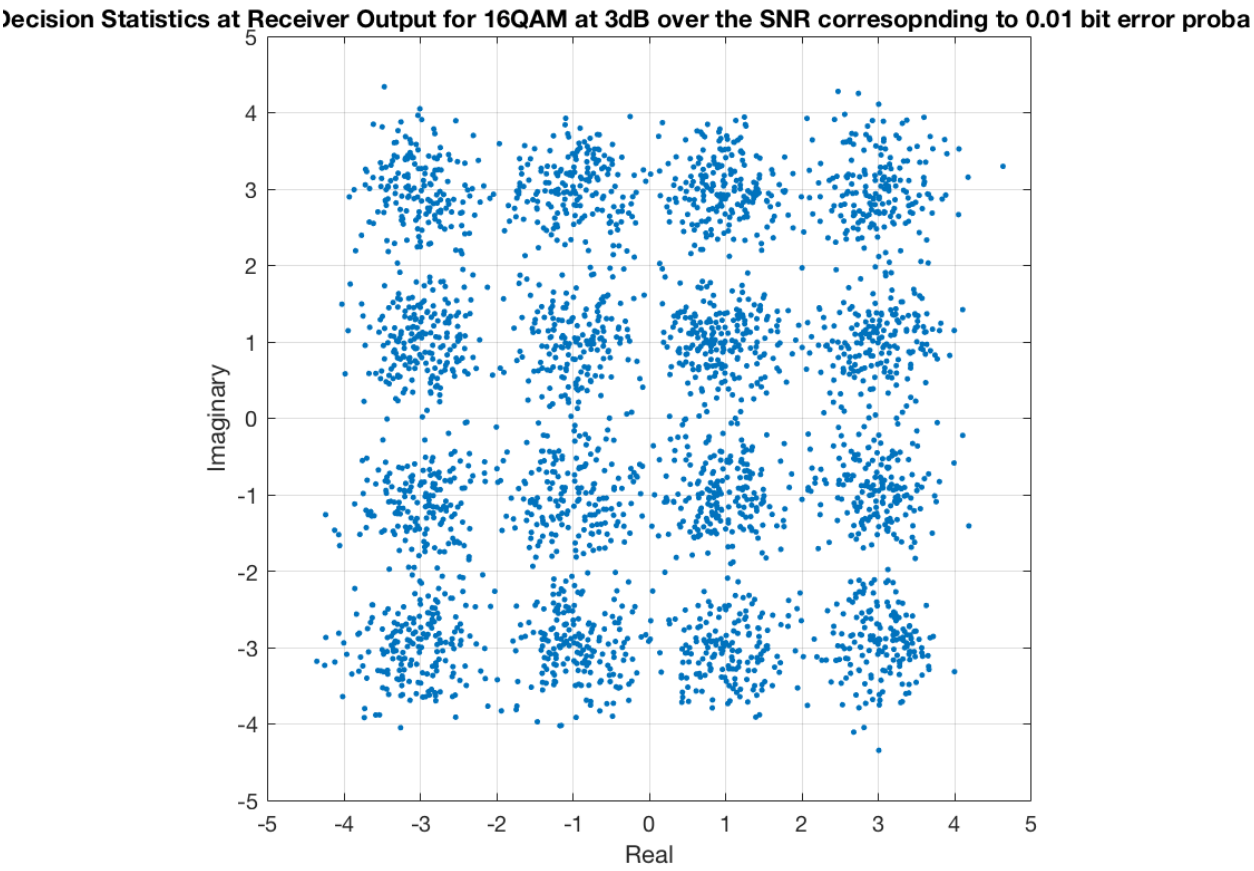


Figure 14: 16QAM Bit error probability as a function of SNR

```
The probability error measured at the receiver output was 0.00025
This compares to the ideal error probability by -0.00975
```

We can see that choosing an SNR to be 3 dB higher than that corresponding to a bit error probability of 0.01, we get a significantly smaller bit error rate since it is inversely proportional to the noise variance.

2 Conclusion

In conclusion, we began this lab by writing functions capable of generating random binary bits and functions that could map those bits into specific signal constellations. We then transmitted BPSK mapped symbols through a noiseless system and ensured we were able to completely recover and reconstruct the signal. Next, we computed the noise variance with the SNR corresponding to the desired bit error of 0.01 for BPSK. We proceeded to add white Gaussian noise at the receiver input with the found variance, sample the receiver output to determine the decision statistics, and compared the resulting error probability against the ideal. We repeated this processes with 4PAM, QPSK, 16QAM, and 8PSK to observe the resulting bit error probabilities. Lastly, we designed a system in which we add the noise before the transmitter filter rather than at the receiver input and measured the error probability achieved in this scenario at the correponsing SNR for 0.01 bit error rate and a SNR that is 3dB higher than this.