```
% Joe Gibson
% EGR 280 - Lab 7
% Digital Communications

%Clear the command window
clc;

%Seed the random number generator
SEED = sum(100*clock);
rand('seed', SEED);

%Define the positive pulse
pulse =    [0
   0.007617389414865
   0.031455342184539
   0.070092094352314
   0.120328669495781
   0.177475320616964
   0.235843683614484
   0.289387944478742
   0.332414859116208
   0.360269954756471
   0.369907985284340
   0.360269954756471
   0.332414859116208
   0.289387944478742
   0.235843683614484
   0.177475320616964
   0.120328669495781
   0.070092094352314
   0.031455342184539
   0.007617389414865];

% Part 1 %
%%%%%%%%%%
%Generate a Binary Pulse vector and its corresponding Binary Vector
[signal, binary] = genBinaryPulse(5000, pulse);

%Define the SNR in terms of dB
SNRmin = -8;
SNRmax = 10;
SNR = SNRmin:1:SNRmax;

%Convert to Linear SNR
SNRL = 10 .^ (SNR./10);

% Part 2 %
%%%%%%%%%%
%Compute the theoretical BER
BER = qfunc(sqrt(SNRL));
```

```
%Determine the Ep for the pulse signal
Ep = trapz(pulse .* pulse);

%Determine the length of the signal
L = length(signal);

%Generate a random signal
r = randn(L, 1);

%Define gamma
gamma = 0;

% Part 3 %
%%%%%%%%%%
%Simulate the matched filter over the SNR range (-8 to 10)
Sn8 = matchfilter(signal, r, Ep, SNRL(1), gamma, pulse);
Sn7 = matchfilter(signal, r, Ep, SNRL(2), gamma, pulse);
Sn6 = matchfilter(signal, r, Ep, SNRL(3), gamma, pulse);
Sn5 = matchfilter(signal, r, Ep, SNRL(4), gamma, pulse);
Sn4 = matchfilter(signal, r, Ep, SNRL(5), gamma, pulse);
Sn3 = matchfilter(signal, r, Ep, SNRL(6), gamma, pulse);
Sn2 = matchfilter(signal, r, Ep, SNRL(7), gamma, pulse);
Sn1 = matchfilter(signal, r, Ep, SNRL(8), gamma, pulse);
S0 = matchfilter(signal, r, Ep, SNRL(9), gamma, pulse);
S1 = matchfilter(signal, r, Ep, SNRL(10), gamma, pulse);
S2 = matchfilter(signal, r, Ep, SNRL(11), gamma, pulse);
S3 = matchfilter(signal, r, Ep, SNRL(12), gamma, pulse);
S4 = matchfilter(signal, r, Ep, SNRL(13), gamma, pulse);
S5 = matchfilter(signal, r, Ep, SNRL(14), gamma, pulse);
S6 = matchfilter(signal, r, Ep, SNRL(15), gamma, pulse);
S7 = matchfilter(signal, r, Ep, SNRL(16), gamma, pulse);
S8 = matchfilter(signal, r, Ep, SNRL(17), gamma, pulse);
S9 = matchfilter(signal, r, Ep, SNRL(18), gamma, pulse);
S10 = matchfilter(signal, r, Ep, SNRL(19), gamma, pulse);

%Determine the bit error rates (BERs) of the different noisy signals
BERn8 = bitError(binary, Sn8);
BERn7 = bitError(binary, Sn7);
BERn6 = bitError(binary, Sn6);
BERn5 = bitError(binary, Sn5);
BERn4 = bitError(binary, Sn4);
BERn3 = bitError(binary, Sn3);
BERn2 = bitError(binary, Sn2);
BERn1 = bitError(binary, Sn1);
BER0 = bitError(binary, S0);
BER1 = bitError(binary, S1);
BER2 = bitError(binary, S2);
BER3 = bitError(binary, S3);
BER4 = bitError(binary, S4);
```

```
BER5 = bitError(binary, S5);
BER6 = bitError(binary, S6);
BER7 = bitError(binary, S7);
BER8 = bitError(binary, S8);
BER9 = bitError(binary, S9);
BER10 = bitError(binary, S10);

%Create an expirimental BER vector using the calculated values
expBER = [BERn8
    BERn7
    BERn6
    BERn5
    BERn4
    BERn3
    BERn2
    BERn1
    BER0
    BER1
    BER2
    BER3
    BER4
    BER5
    BER6
    BER7
    BER8
    BER9
    BER10];

%Plot a clean signal
[signalPlot, bin] = genBinaryPulse(5, pulse);
figure(1)
plot(signalPlot);
grid on;
xlabel('t');
ylabel('V(t)');
title('Clean Binary Signal');

%Plot a noisy signal
rn = randn(100, 1);
[noisySignal1] = noisySignal(signalPlot, rn, Ep, SNRL(19));
figure(2);
plot(noisySignal1);
grid on;
xlabel('t');
ylabel('V(t)');
title('Noisy Binary Signal: SNR = 10dB');

%Plot the theoretical BER and the expiremental BER on the same axis
%using logarithmic scaling for the Y axis
figure(3);
```

```
semilogy(SNRL, BER, SNRL, expBER);
%axes('YScale', 'log');
grid on;
xlabel('Linear Signal to Noise Ratio');
ylabel('Bit Error Rate');
title('Theoretical and Experimental Bit Error Rates: Gamma = 1');

% Part 4 %
%%%%%%%%%%
%Repeat above with Gamma of 0.8 and 1.2
gammaLow = 0.8;
gammaHigh = 1.2;

%Simulate the matched filter over the SNR range (-8 to 10)
Sn8 = matchfilter(signal, r, Ep, SNRL(1), gammaLow, pulse);
Sn7 = matchfilter(signal, r, Ep, SNRL(2), gammaLow, pulse);
Sn6 = matchfilter(signal, r, Ep, SNRL(3), gammaLow, pulse);
Sn5 = matchfilter(signal, r, Ep, SNRL(4), gammaLow, pulse);
Sn4 = matchfilter(signal, r, Ep, SNRL(5), gammaLow, pulse);
Sn3 = matchfilter(signal, r, Ep, SNRL(6), gammaLow, pulse);
Sn2 = matchfilter(signal, r, Ep, SNRL(7), gammaLow, pulse);
Sn1 = matchfilter(signal, r, Ep, SNRL(8), gammaLow, pulse);
S0 = matchfilter(signal, r, Ep, SNRL(9), gammaLow, pulse);
S1 = matchfilter(signal, r, Ep, SNRL(10), gammaLow, pulse);
S2 = matchfilter(signal, r, Ep, SNRL(11), gammaLow, pulse);
S3 = matchfilter(signal, r, Ep, SNRL(12), gammaLow, pulse);
S4 = matchfilter(signal, r, Ep, SNRL(13), gammaLow, pulse);
S5 = matchfilter(signal, r, Ep, SNRL(14), gammaLow, pulse);
S6 = matchfilter(signal, r, Ep, SNRL(15), gammaLow, pulse);
S7 = matchfilter(signal, r, Ep, SNRL(16), gammaLow, pulse);
S8 = matchfilter(signal, r, Ep, SNRL(17), gammaLow, pulse);
S9 = matchfilter(signal, r, Ep, SNRL(18), gammaLow, pulse);
S10 = matchfilter(signal, r, Ep, SNRL(19), gammaLow, pulse);

%Determine the bit error rates (BERs) of the different noisy signals
BERn8 = bitError(binary, Sn8);
BERn7 = bitError(binary, Sn7);
BERn6 = bitError(binary, Sn6);
BERn5 = bitError(binary, Sn5);
BERn4 = bitError(binary, Sn4);
BERn3 = bitError(binary, Sn3);
BERn2 = bitError(binary, Sn2);
BERn1 = bitError(binary, Sn1);
BER0 = bitError(binary, S0);
BER1 = bitError(binary, S1);
BER2 = bitError(binary, S2);
BER3 = bitError(binary, S3);
BER4 = bitError(binary, S4);
BER5 = bitError(binary, S5);
BER6 = bitError(binary, S6);
```

```
BER7 = bitError(binary, S7);
BER8 = bitError(binary, S8);
BER9 = bitError(binary, S9);
BER10 = bitError(binary, S10);

%Create an expirimental BER vector using the calculated values
expBER = [BERn8
    BERn7
    BERn6
    BERn5
    BERn4
    BERn3
    BERn2
    BERn1
    BER0
    BER1
    BER2
    BER3
    BER4
    BER5
    BER6
    BER7
    BER8
    BER9
    BER10];

%Plot the theoretical BER and the expiremental BER on the same axis
%using logarithmic scaling for the Y axis for GAMMA = 0.8 (Low)
figure(4);
semilogy(SNRL, BER, SNRL, expBER);
grid on;
xlabel('Linear Signal to Noise Ratio');
ylabel('Bit Error Rate');
title('Theoretical and Experimental Bit Error Rates: Gamma = 0.8');

%Simulate the matched filter over the SNR range (-8 to 10)
Sn8 = matchfilter(signal, r, Ep, SNRL(1), gammaHigh, pulse);
Sn7 = matchfilter(signal, r, Ep, SNRL(2), gammaHigh, pulse);
Sn6 = matchfilter(signal, r, Ep, SNRL(3), gammaHigh, pulse);
Sn5 = matchfilter(signal, r, Ep, SNRL(4), gammaHigh, pulse);
Sn4 = matchfilter(signal, r, Ep, SNRL(5), gammaHigh, pulse);
Sn3 = matchfilter(signal, r, Ep, SNRL(6), gammaHigh, pulse);
Sn2 = matchfilter(signal, r, Ep, SNRL(7), gammaHigh, pulse);
Sn1 = matchfilter(signal, r, Ep, SNRL(8), gammaHigh, pulse);
S0 = matchfilter(signal, r, Ep, SNRL(9), gammaHigh, pulse);
S1 = matchfilter(signal, r, Ep, SNRL(10), gammaHigh, pulse);
S2 = matchfilter(signal, r, Ep, SNRL(11), gammaHigh, pulse);
S3 = matchfilter(signal, r, Ep, SNRL(12), gammaHigh, pulse);
S4 = matchfilter(signal, r, Ep, SNRL(13), gammaHigh, pulse);
S5 = matchfilter(signal, r, Ep, SNRL(14), gammaHigh, pulse);
```

```
S6 = matchfilter(signal, r, Ep, SNRL(15), gammaHigh, pulse);
S7 = matchfilter(signal, r, Ep, SNRL(16), gammaHigh, pulse);
S8 = matchfilter(signal, r, Ep, SNRL(17), gammaHigh, pulse);
S9 = matchfilter(signal, r, Ep, SNRL(18), gammaHigh, pulse);
S10 = matchfilter(signal, r, Ep, SNRL(19), gammaHigh, pulse);

%Determine the bit error rates (BERs) of the different noisy signals
BERn8 = bitError(binary, Sn8);
BERn7 = bitError(binary, Sn7);
BERn6 = bitError(binary, Sn6);
BERn5 = bitError(binary, Sn5);
BERn4 = bitError(binary, Sn4);
BERn3 = bitError(binary, Sn3);
BERn2 = bitError(binary, Sn2);
BERn1 = bitError(binary, Sn1);
BER0 = bitError(binary, S0);
BER1 = bitError(binary, S1);
BER2 = bitError(binary, S2);
BER3 = bitError(binary, S3);
BER4 = bitError(binary, S4);
BER5 = bitError(binary, S5);
BER6 = bitError(binary, S6);
BER7 = bitError(binary, S7);
BER8 = bitError(binary, S8);
BER9 = bitError(binary, S9);
BER10 = bitError(binary, S10);

%Create an expirimental BER vector using the calculated values
expBER = [BERn8
    BERn7
    BERn6
    BERn5
    BERn4
    BERn3
    BERn2
    BERn1
    BER0
    BER1
    BER2
    BER3
    BER4
    BER5
    BER6
    BER7
    BER8
    BER9
    BER10];

%Plot the theoretical BER and the expiremental BER on the same axis
%using logarithmic scaling for the Y axis for GAMMA = 1.2 (High)
```

```
figure(5);
semilogy(SNRL, BER, SNRL, expBER);
grid on;
xlabel('Linear Signal to Noise Ratio');
ylabel('Bit Error Rate');
title('Theoretical and Experimental Bit Error Rates: Gamma = 1.2');

% Part 5 %
%%%%%%%%%%
%Create a new pulse template based on a square wave
pulse =    [0
   0.369907985284340
   0.369907985284340
   0.369907985284340
   0.369907985284340
   0.369907985284340
   0.369907985284340
   0.369907985284340
   0.369907985284340
   0.369907985284340
   0.369907985284340
   0.369907985284340
   0.369907985284340
   0.369907985284340
   0.369907985284340
   0.369907985284340
   0.369907985284340
   0.369907985284340
   0.369907985284340
   0.369907985284340];

%Calculate the Ep with the new pulse
Ep = trapz(pulse .* pulse);

%Repeat part 3 with the new pulse and Ep
%Simulate the matched filter over the SNR range (-8 to 10)
Sn8 = matchfilter(signal, r, Ep, SNRL(1), gamma, pulse);
Sn7 = matchfilter(signal, r, Ep, SNRL(2), gamma, pulse);
Sn6 = matchfilter(signal, r, Ep, SNRL(3), gamma, pulse);
Sn5 = matchfilter(signal, r, Ep, SNRL(4), gamma, pulse);
Sn4 = matchfilter(signal, r, Ep, SNRL(5), gamma, pulse);
Sn3 = matchfilter(signal, r, Ep, SNRL(6), gamma, pulse);
Sn2 = matchfilter(signal, r, Ep, SNRL(7), gamma, pulse);
Sn1 = matchfilter(signal, r, Ep, SNRL(8), gamma, pulse);
S0 = matchfilter(signal, r, Ep, SNRL(9), gamma, pulse);
S1 = matchfilter(signal, r, Ep, SNRL(10), gamma, pulse);
S2 = matchfilter(signal, r, Ep, SNRL(11), gamma, pulse);
S3 = matchfilter(signal, r, Ep, SNRL(12), gamma, pulse);
S4 = matchfilter(signal, r, Ep, SNRL(13), gamma, pulse);
S5 = matchfilter(signal, r, Ep, SNRL(14), gamma, pulse);
```

```
S6 = matchfilter(signal, r, Ep, SNRL(15), gamma, pulse);
S7 = matchfilter(signal, r, Ep, SNRL(16), gamma, pulse);
S8 = matchfilter(signal, r, Ep, SNRL(17), gamma, pulse);
S9 = matchfilter(signal, r, Ep, SNRL(18), gamma, pulse);
S10 = matchfilter(signal, r, Ep, SNRL(19), gamma, pulse);

%Determine the bit error rates (BERs) of the different noisy signals
BERn8 = bitError(binary, Sn8);
BERn7 = bitError(binary, Sn7);
BERn6 = bitError(binary, Sn6);
BERn5 = bitError(binary, Sn5);
BERn4 = bitError(binary, Sn4);
BERn3 = bitError(binary, Sn3);
BERn2 = bitError(binary, Sn2);
BERn1 = bitError(binary, Sn1);
BER0 = bitError(binary, S0);
BER1 = bitError(binary, S1);
BER2 = bitError(binary, S2);
BER3 = bitError(binary, S3);
BER4 = bitError(binary, S4);
BER5 = bitError(binary, S5);
BER6 = bitError(binary, S6);
BER7 = bitError(binary, S7);
BER8 = bitError(binary, S8);
BER9 = bitError(binary, S9);
BER10 = bitError(binary, S10);

%Create an expirimental BER vector using the calculated values
expBER = [BERn8
    BERn7
    BERn6
    BERn5
    BERn4
    BERn3
    BERn2
    BERn1
    BER0
    BER1
    BER2
    BER3
    BER4
    BER5
    BER6
    BER7
    BER8
    BER9
    BER10];

%Plot the theoretical BER and the expiremental BER on the same axis
%using logarithmic scaling for the Y axis
```

```
figure(6);
semilogy(SNRL, BER, SNRL, expBER);
grid on;
xlabel('Linear Signal to Noise Ratio');
ylabel('Bit Error Rate');
title('Theoretical and Experimental Bit Error Rates: Gamma = 1');
```

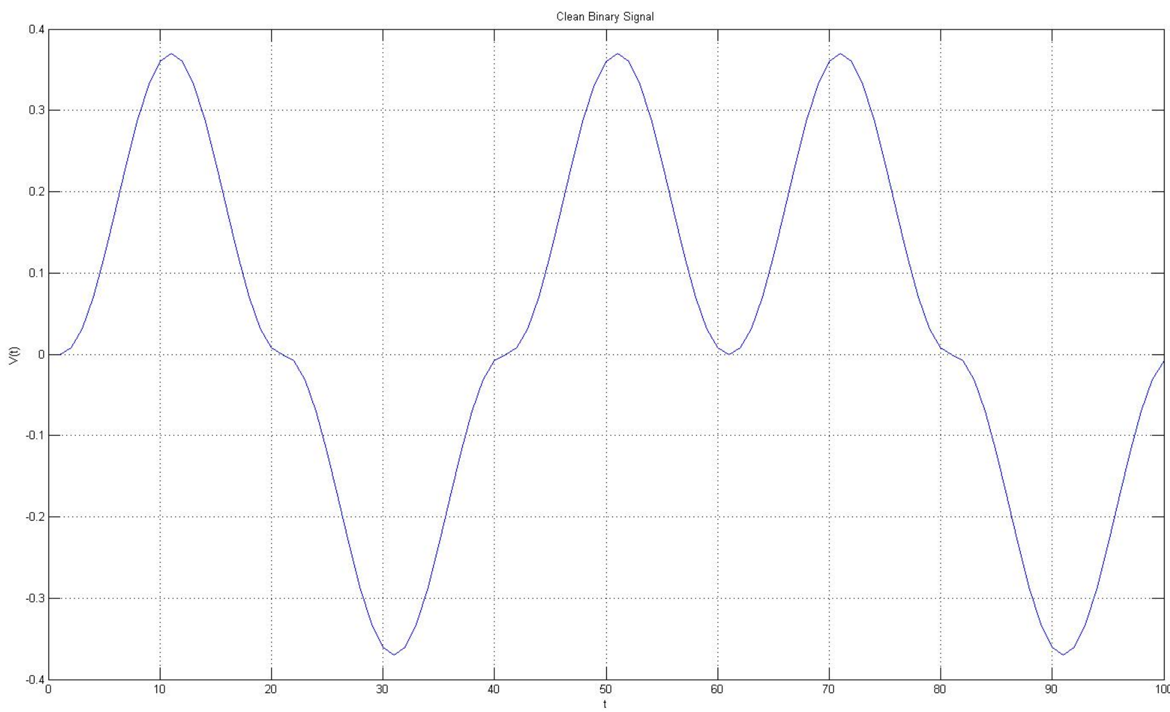
```
function [signal, binary] = genBinaryPulse(N, pulse);
%Usage: [signal, binary] = genBinaryPulse(N, pulse)
%Where signal is the vector containing the pulse waveform for
%each binary value, binary is the binary vector, N is the
%length of the binary vector, and pulse is the pulse waveform for
%a value of '1'

%Generate the random vector
r = rand(N, 1);

%Initialize the binary vector
binary = zeros(N, 1);

%Determine the binVector
for i = 1:N
    if(r(i)>= 0.5)
        binary(i) = 1;
    else
        binary(i) = 0;
    end
end

%Initialize the binPulse
signal = zeros(0, 1);

%Determine the binPulse
for i = 1:N
    if(binary(i) == 1)
        signal = [signal; pulse];
    else
        signal = [signal; -pulse];
    end
end
```

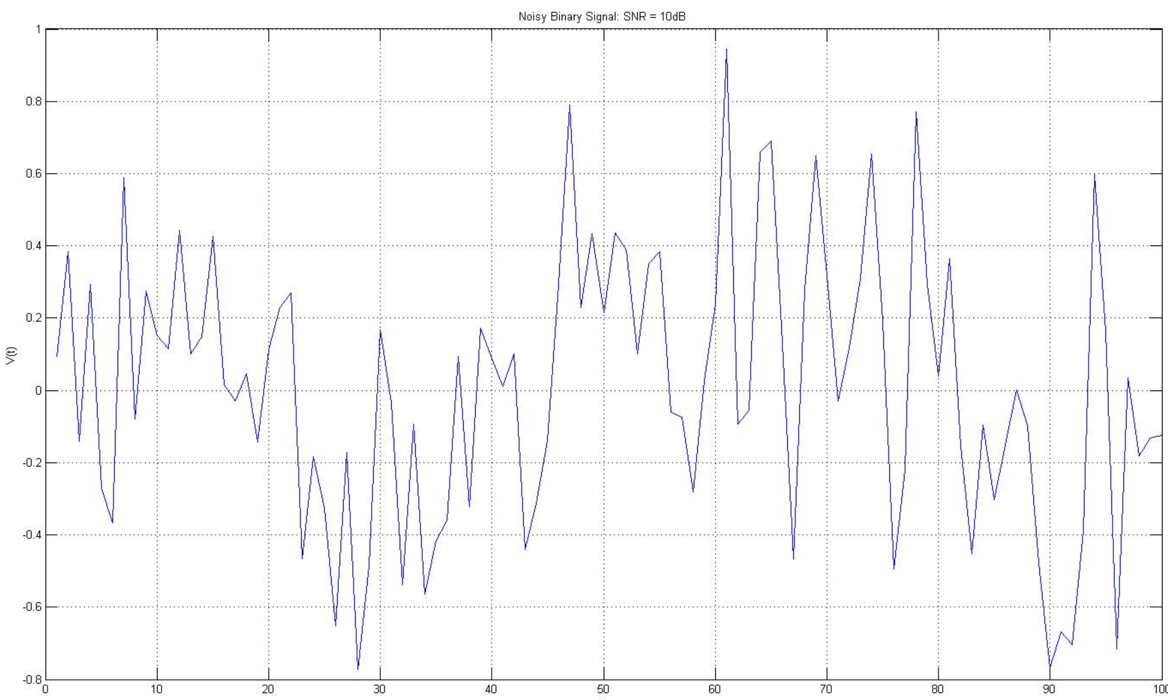
```
function [receivedDigits] = matchfilter(signal, r, Ep, SNRL, gamma,
pulse)
%Usage: [receivedDigits] = matchfilter(signal)
%Where receivedDigits is the vector containing the digits
%received by the filter, signal is the clean input signal, r is the
%unscaled random noise to be applied to the signal,
%Ep is the mean of the signal, SNRL is the linear
%signal to noise ratio, gamma is the optimum threshold, and
%pulse is the original pulse template

%Determine the length of the signal and the random noise
L = length(signal);
Lr = length(r);

%Check for noise mismatch
if(L ~= Lr)
    error('The length of the noise does not match the signal');
end

%Determine sigma
sigma = sqrt(Ep / SNRL);

%Scale the random signal
noise = sigma .* r;

%Add the noise to the original signal
signal = signal + noise;

%Define the binary vector
receivedDigits = zeros(L/20, 1);

%Define the signal sum;
sum = 0;

%Create a pulse vector the length of the binary vector
pulseV = [];

for i = 1:(L/20)
    pulseV = [pulseV; pulse;];
end

%Create a temp vector for integrating
temp = zeros(20, 1);

%Align the signal to the pulse template, integrate the signal, and
%determine a 1 or 0
b = 1;

for i = 1:20:L
```

```
    c = 1;

    for j = i:i+(20 - 1)
        temp(c) = signal(j) * pulseV(j);
        c = c + 1;
    end

    %Integrate the signal
    sum = trapz(temp);

    %Determine 1 or 0
    if(sum > gamma)
        receivedDigits(b) = 1;
    else
        receivedDigits(b) = 0;
    end

    b = b + 1;
end
```

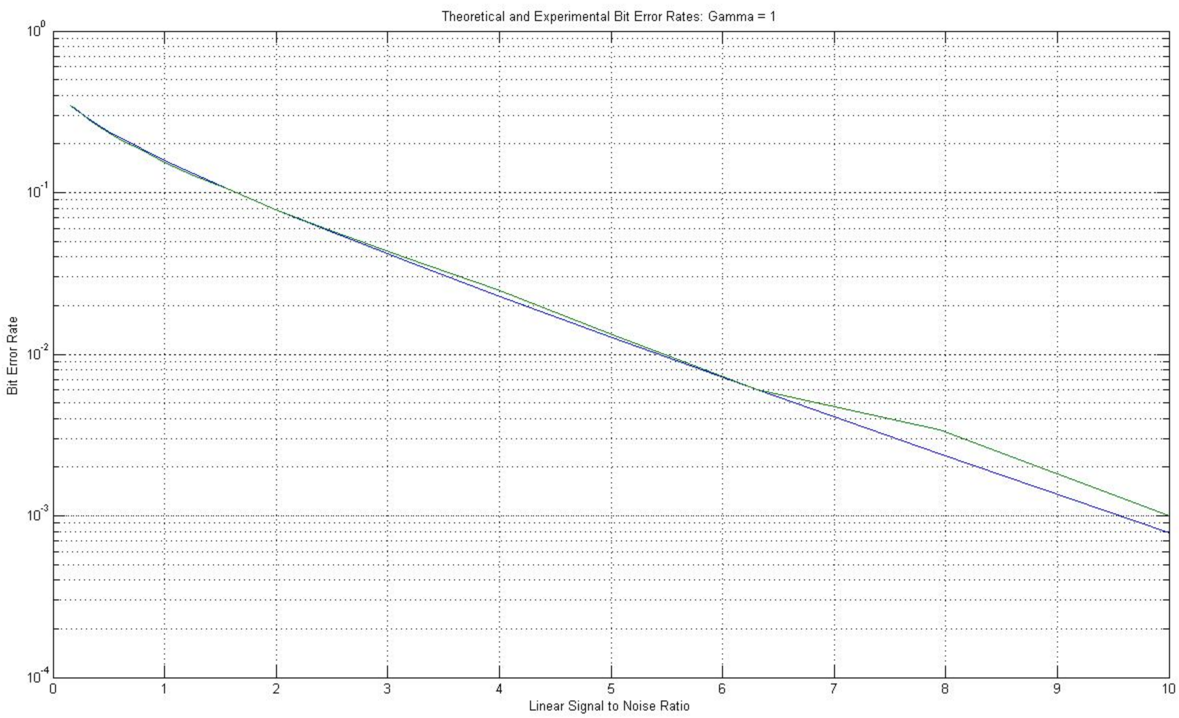
```
function [BER] = bitError(Tx, Rx);
%Usage: [BER] = bitError(Tx, Rx)
%Where BER is the calculated bit error rate, Tx is the transmitted
(clean)
%signal, and Rx is the received (noisy) signal.

%Determine the length of the two data streams and compare
LTx = length(Tx);
LRx = length(Rx);

if(LTx ~= LRx)
    error('Data streams do not match in length!');
end

%Define the count for error bits
errors = 0;

for i = 1:LRx
    TxBit = Tx(i);
    RxBit = Rx(i);

    if(TxBit ~= RxBit)
        errors = errors + 1;
    end
end

%Determine the bit error rate given the number of errors present
BER = errors / LRx;
```

```
function [noisySignal] = noisySignal(signal, r, Ep, SNRL)
%Usage: [noisySignal] = noise(signal)
%Where noisySignal is the clean signal with noise added,
%signal is the clean signal, r is the random noise, Ep is the signal
mean,
%and SNRL is the linear Signal to Noise Ratio

%Determine sigma
sigma = sqrt(Ep / SNRL);

%Scale the random signal
noise = sigma .* r;

%Add the noise to the original signal
noisySignal = signal + noise;
```

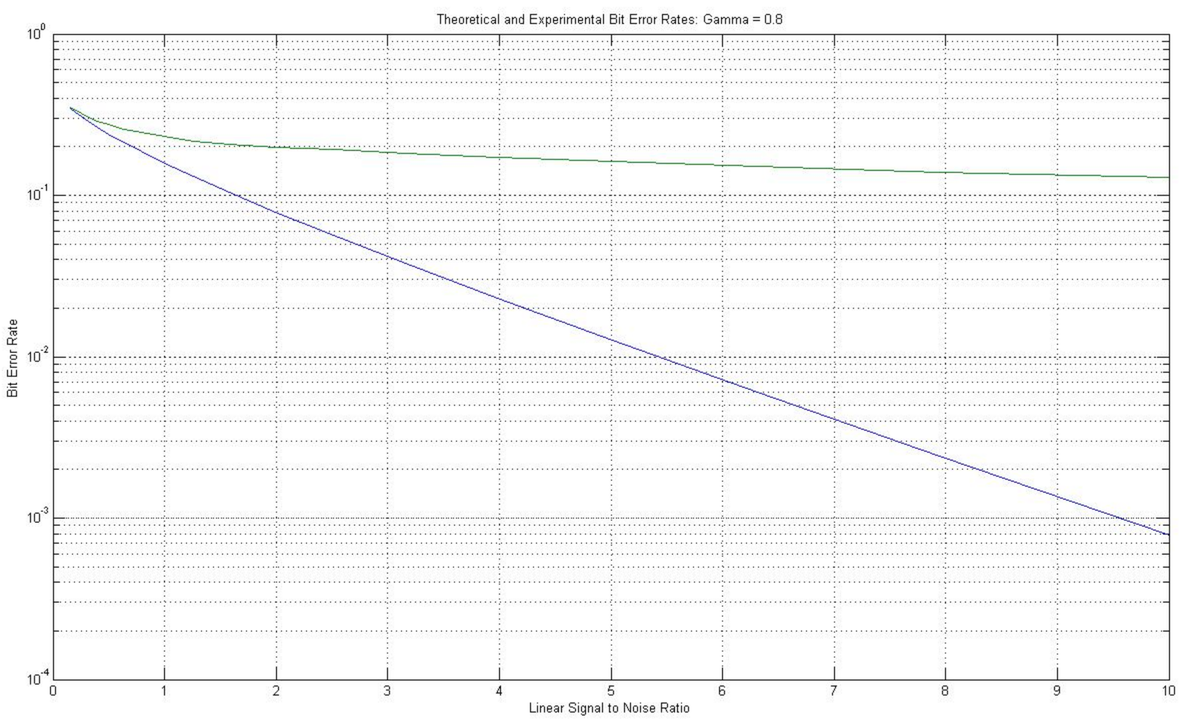

Clean Binary Signal
V(t)
t
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
0
10
20
30
40
50
60
70
80
90
100



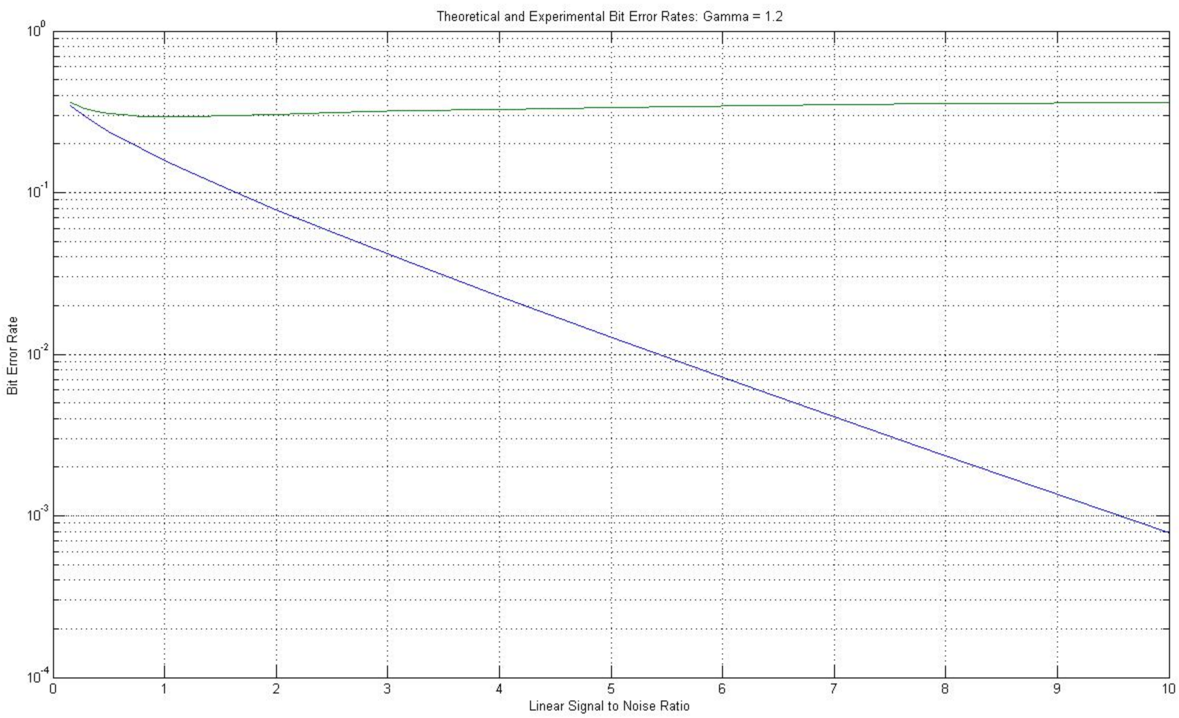
Noisy Binary Signal: SNR = 10dB
V(t)
t



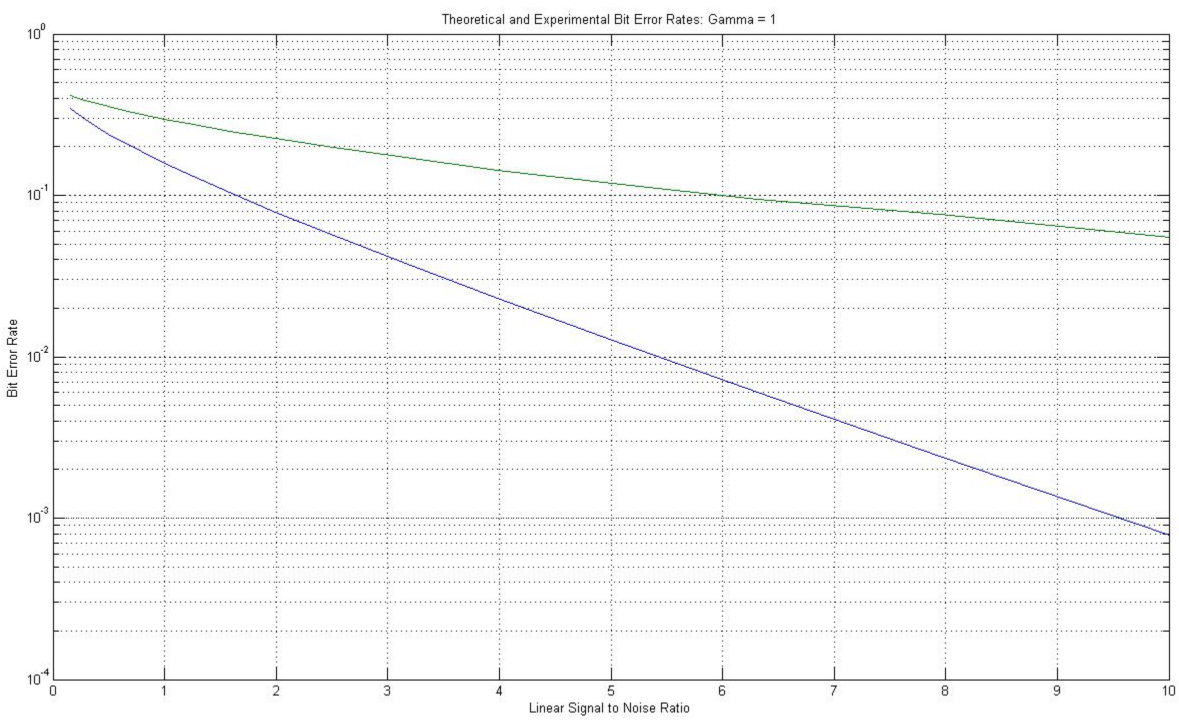
Theoretical and Experimental Bit Error Rates: Gamma = 1
Bit Error Rate
Linear Signal to Noise Ratio
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
0
1
2
3
4
5
6
7
8
9
10



Theoretical and Experimental Bit Error Rates: Gamma = 0.8
Bit Error Rate
Linear Signal to Noise Ratio
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
0
1
2
3
4
5
6
7
8
9
10



Theoretical and Experimental Bit Error Rates: Gamma = 1.2
Bit Error Rate
Linear Signal to Noise Ratio
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
0
1
2
3
4
5
6
7
8
9
10



Theoretical and Experimental Bit Error Rates: Gamma = 1
Bit Error Rate
Linear Signal to Noise Ratio
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
0
1
2
3
4
5
6
7
8
9
10