Lab 3 - Digital Communications

Colt Thomas

August 16, 2021

Introduction

This lab consists of three parts which will cover the techniques of implementing digital communications systems in MATLAB. The first section will discuss the principles of sampling and digitizing signals. Section two discusses the importance of eye diagrams and illustrates various pulse shapes. The final portion will apply the discussed principles by implementing a Binary Phase Shift Keying (BPSK) communication system across an Additive White Gaussian Noise (AWGN) channel, and will discuss techniques used to get an adequate Bit Error Rate (BER).

1 - Sampling and Digitization

For this section, the exercise on page 346 or the Lathi book was followed to produce the plots below. Code can be found in the appendix of this lab document as "Exsample.m", "sampandquant.m", and "uniquan.m".

1.1 Test Signal

a) Plot time versus test signal (xsig) (figure 1).

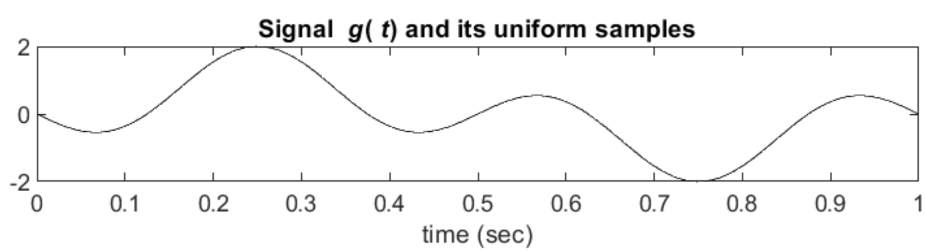

Figure 1: Plot of our original signal in time which we will use to sample and digitize.

b) Plot freq versus test signal spectrum (Xsig) (figure 2).

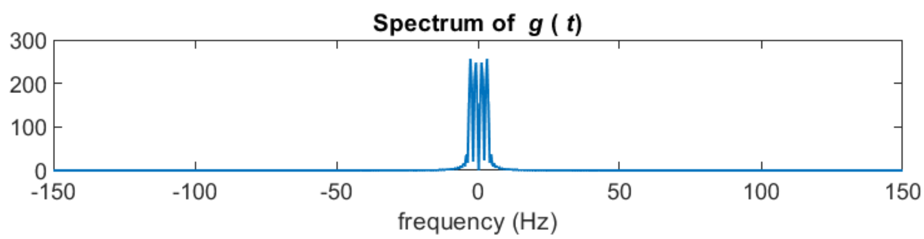

Figure 2: Plot of our original signal spectrum. Note the amplitude of 250 and our signal components at 1 and 3Hz

1.2 Sixteen Quantization Levels

a) Plot time versus sampled signal (s_out) (3).

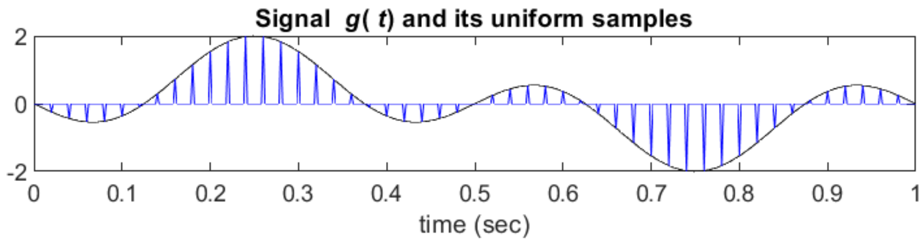

Figure 3: Plot of our sampled signal in time.

b) Plot freq versus sampled signal spectrum (S_out) (4).

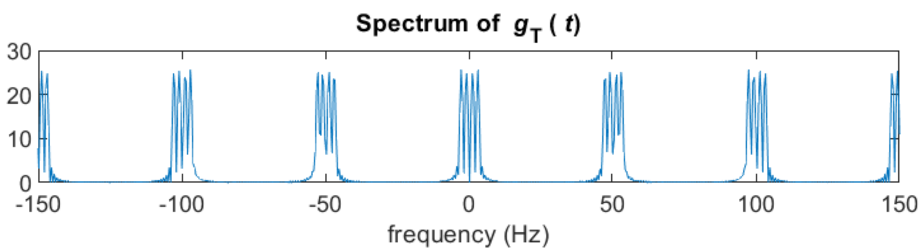

Figure 4: Plot of our sampled signal spectrum.

c) Plot time versus the sample and hold (flat top) reconstruction (sqh_out) (5).

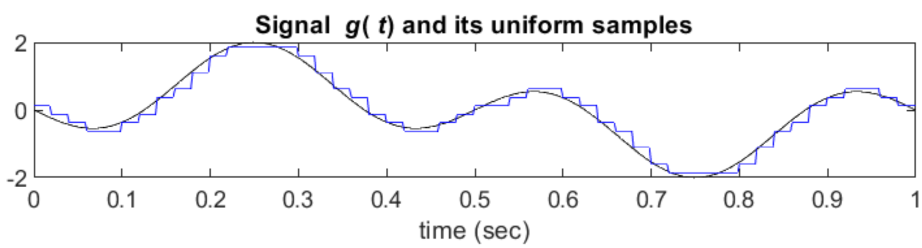

Figure 5: Plot of our sampled signal spectrum versus the sample sample and hold reconstruction.

d) What is the signal-to-quantization noise ratio (SQNR)?

This can be found by using the "snr()" command in MATLAB, which yields us $\boxed{0.0614}$

1.3 Aliasing

a) At what sampling rate will 'xsig' begin to alias?

The highest frequency component is at 3Hz (the sine component of xsig), so we need to at least sample twice that $\boxed{6hz}$. It will begin to alias below that value. We can set ts=0.166 to see the aliasing.

b) Plot time versus xsig at this sampling rate.

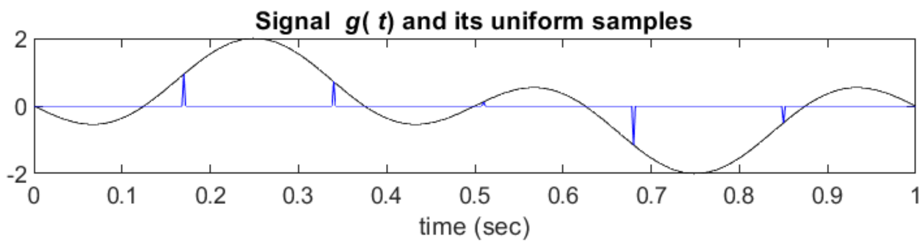

Figure 6: Plot of our sampled signal spectrum at the sampling frequency of 6hz.

c) Plot freq versus the spectrum of xsig at this sampling rate.

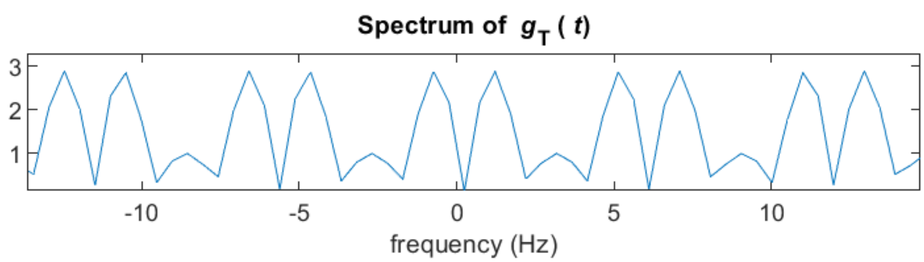

Figure 7: Plot of our sampled signal spectrum at the threshold sample rate.

1.4 Four Quantization Levels

a) Change the number of quantization levels, L, to 4. Plot time versus the new sample and hold output (sqh_out).

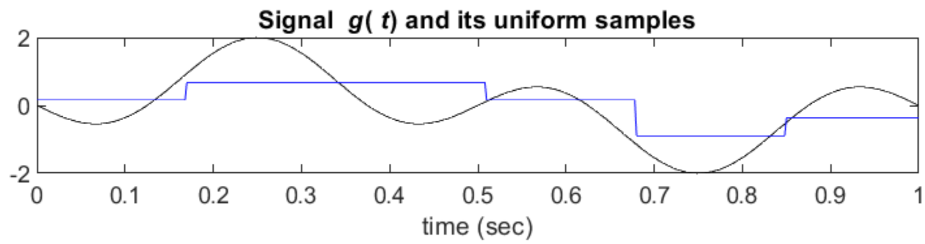

Figure 8: Plot of our sampled signal in time with a lower quantization level of 4 (from 16). Notice how distorted the signal looks in comparison with the original.

b) Compute the FFT of this new sqh_out and plot freq vs. spectrum.

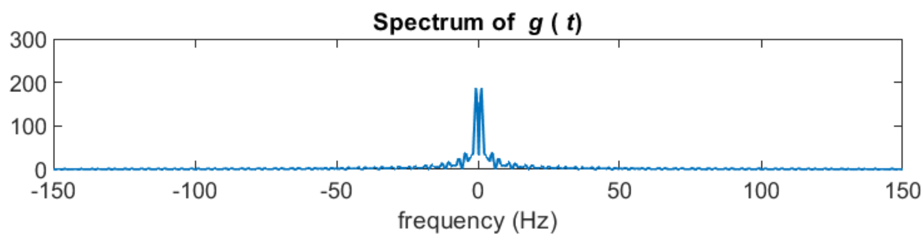

Figure 9: Plot of our sampled signal spectrum with a lower quantization level of 4. Notice that the signal is attenuated.

c) What is the signal-to-quantization noise ratio now?

It is rather substantial. Using the same "snr()" function we get $\boxed{4.9473}$ for our signal to noise ratio.

d) What is the voltage delta between bits now?

The voltage delta is lower than the amplitude of the actual signal... which seems to be $\boxed{\approxeq 0.5}$

2 - Eye Diagram Analysis

This section covers pulse shapes and compares them using an eye diagrams. These are useful for finding optimal sample times for various pulses and can be used to spot inter-symbol interference (ISI).

2.1 Data Pulse Trains

a) Plot the data pulse train for the ideal return-to-zero case (yrz).

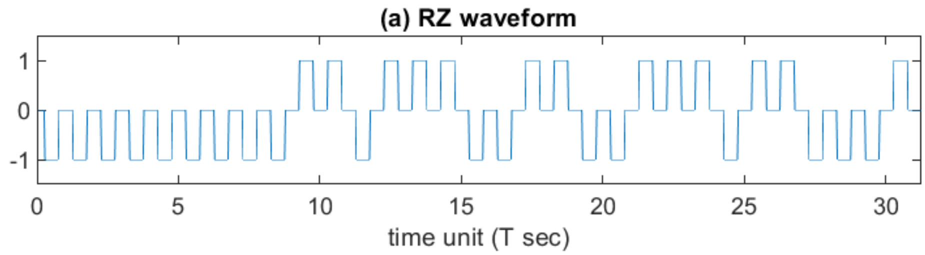

Figure 10: Data pulse train for ideal return-to-zero.

b) Plot the data pulse train for the ideal non-return-to-zero case (ynrz).

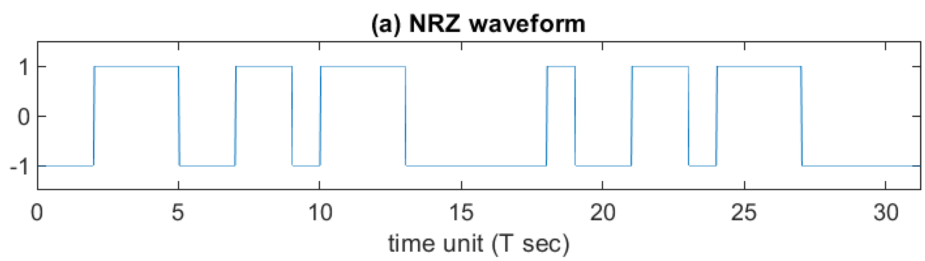

Figure 11: Data pulse train for ideal non return-to-zero.

c) Plot the data pulse train for the half-sine pulse (ysine).
d) Plot the data pulse train for the raised-cosine pulse (yrcos).

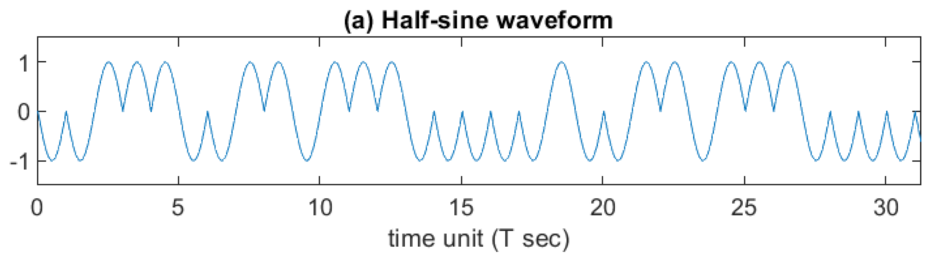

Figure 12: Data pulse train for ideal half-sine pulse.

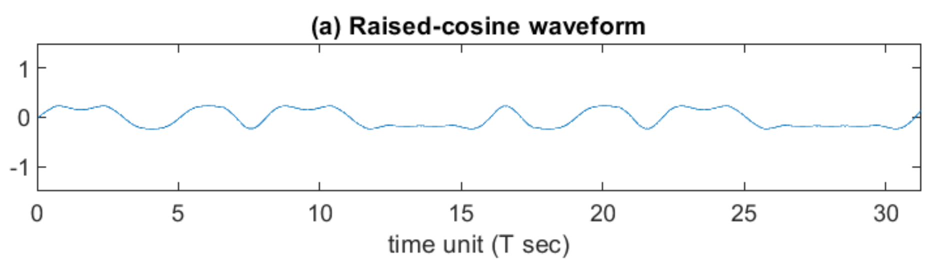

Figure 13: Data pulse train for ideal raised-cosine pulse.

2.2 Eye Diagrams

a) Plot the eye diagram for the ideal return-to-zero pulse (yrz).

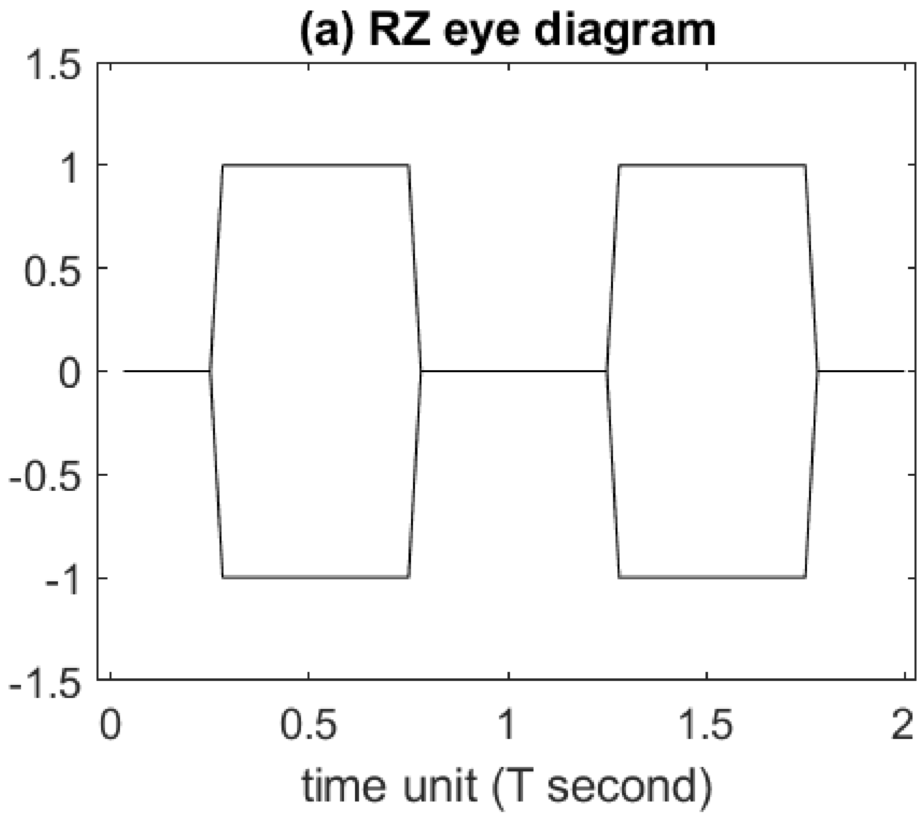

Figure 14: Eye diagram for return-to-zero pulse. Notice that our sampling windows are centered at 0.5 and 1.5 seconds, with a window width of 0.5 seconds.

b) Plot the eye diagram for the ideal non-return-to-zero pulse (ynrz).
c) Plot the eye diagram for the half-sine pulse (ysine).
d) Plot the eye diagram for the raised-cosine pulse (yrcos).

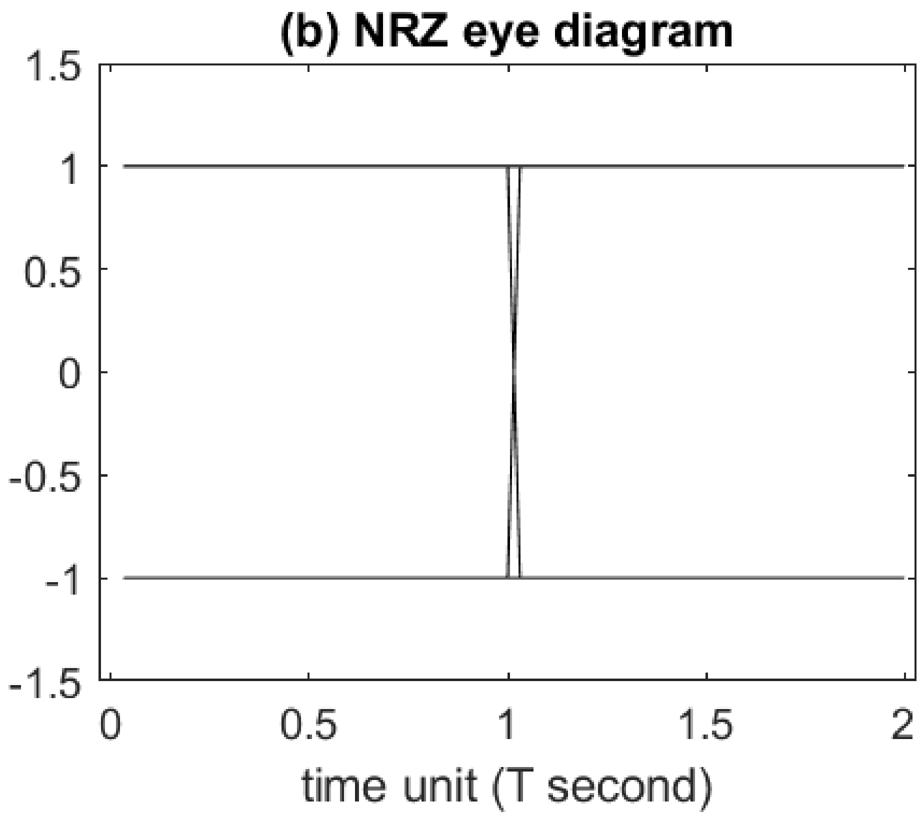

Figure 15: Eye diagram for non return-to-zero pulse. This is a very forgiving pulse shape with a wide area for sampling.

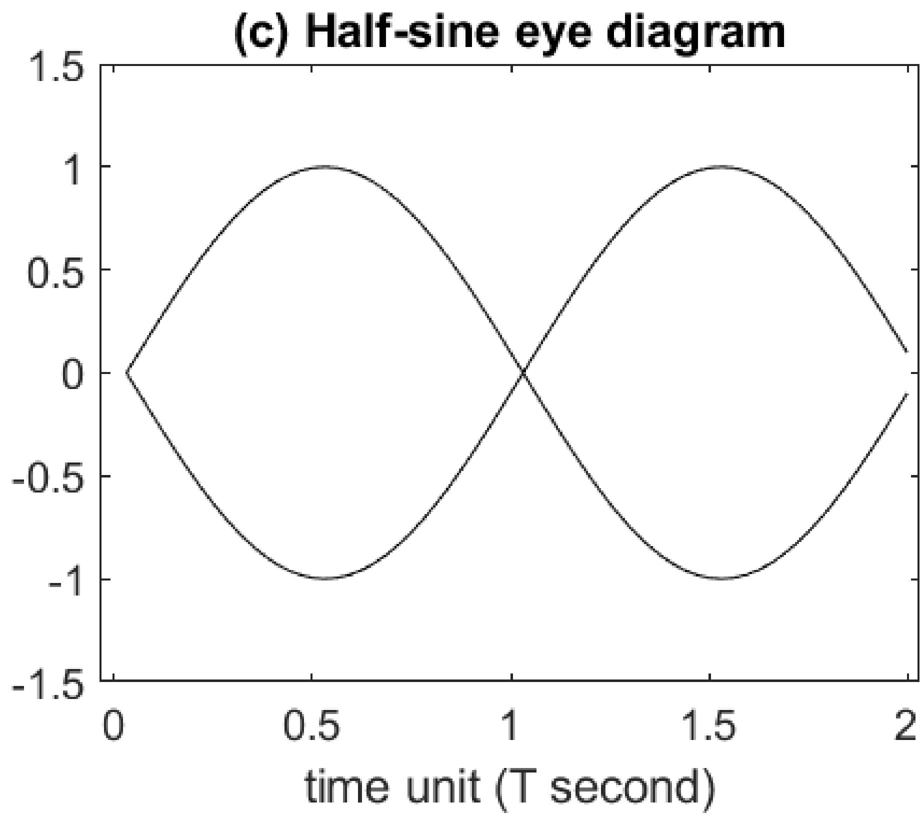

Figure 16: Eye diagram for the half-sine pulse. This has optimal peaks right at 0.5 and 1.5 seconds, but we can still easily detect the symbol otherwise assuming low noise.

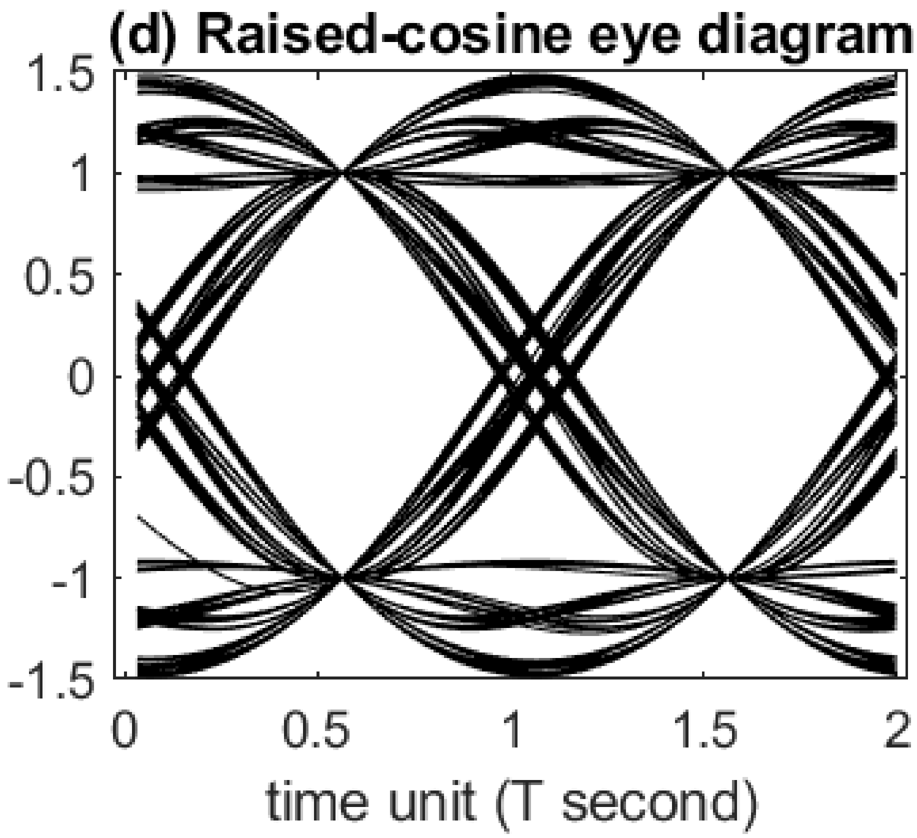

Figure 17: Eye diagram for a raised cosine pulse. Note that this pulse requires higher timing precision to sample correctly and is more prone to ISI.

2.3 Questions

a) As the eye diagram becomes more “open”, what happens to “inter-symbol interference”, that is the chance of interference between bits?

The eye just lines up all the possible overlap within a symbol frame. When sampling at the open part of the eye, the chance of inter-symbol interference is at its lowest. For the return-to-zero (and nrz) there is no ambiguity in the opening of the eye. Note that with return-to-zero we have extreme ambiguity if our timing is off and we sample outsize of the eye where the amplitude is at zero. Non return-to-zero has a significantly lower likelihood of experiencing inter-symbol interference.

b) What is the cost of having less chance of inter-symbol interference?

The pulse shapes that have less ISI tend to require more power and bandwidth.

3 - BER Curves

Plot the BER vs Eb/No curve for a BPSK system operating over an AWGN channel. This should be designed so that you are generating random bits through a modulator, the signals affected by the channel and then the demodulator. You should have functions to handle whatever statics gathering you need (e.g., tracking the number of incorrect bits received). Show the performance down to a BER of 1E-3 and justify how many bits you use to achieve proper statistics. Plot your simulated performance curve as well as a theoretical performance curve for BPSK. There are some MATLAB exercises in the book (e.g., Exercise 9.1 Binary Polar Signaling with Different Pulses and others) that can provide inspiration.

This was a rather challenging portion of the lab that required some strategy to achieve the necessary Bit Error Rate (BER). The code is below in the appendix for reference. To achieve this value a few things were needed, with parameters set:

1. Number of bits: 4000
2. Symbol Period: 4 seconds
3. Carrier Frequency: 10 KHz
4. Sample Offset
5. Hamming (7,4) Encoding
6. Low Pass Filter

The first four items/parameters have the largest effect on the BER. I tried to send more bits and see if I could reduce the BER, but somehow the average number of bit errors would increase as I added bits. Sample offset was crucial since we needed to collect samples towards the middle of each pulse. The half-sine pulse was used in this application, which means a sample period should be centered at half of the symbol period. (see line 97 of the code). Fixing this reduced my initial error from a 0.5 probability (missing every sample in that case) to about 0.2 alone with noise. Even without noise, having an incorrect offset can cause a lot of data to be missed. The carrier frequency and symbol period were selected to avoid aliasing/quantization noise. I didn't have substantial time to investigate the relation, but this would be an area to investigate that could enable more bits to be sent since it directly impacts bandwidth and bit rate.

I used a lowpass filter and the Hamming Encoding technique to reduce noise sufficiently. This significantly increased the complexity of the code and runtime, but it was able to meet the required BER. See figure 18 for the performance in comparison to 19. Between the two techniques, the lowpass filter with a corner frequency of about 10KHz was able to significantly reduce the BER by an order of magnitude. The Hamming code wasn't able to reduce the BER that substantially, but it did make a slight difference. The lowpass filter alone would have been sufficient in this case. If the Hamming code were made larger we could potentially see an improvement at the cost of run time complexity. It would be more productive to experiment with the symbol period size, carrier frequency, or possibly look at other techniques. These techniques include, but are not limited to: differential encoding, PN codes, timing and phase correction, and more. There could also be some algorithmic errors that may not be accounted for in this implementation as well.

BER calculations were done using examples from the end of chapter 9 in the Lathi book.

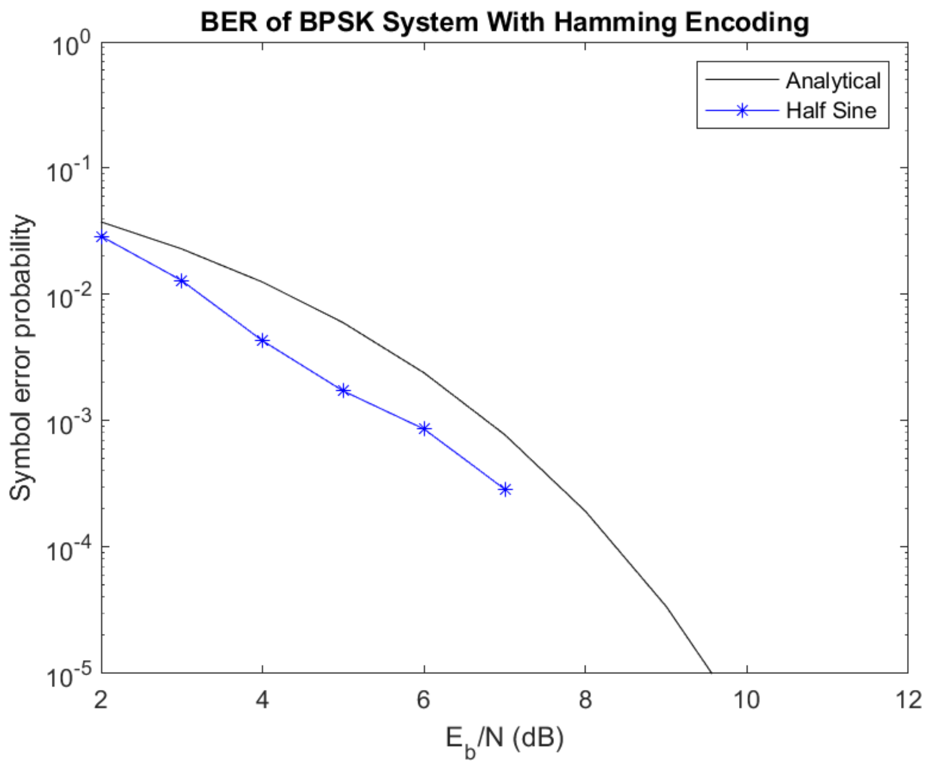

Figure 18: BER Plot of the BPSK Communication System, using a Hamming (7,4) encoding and low pass filter to reduce the errors below the expected performance value.

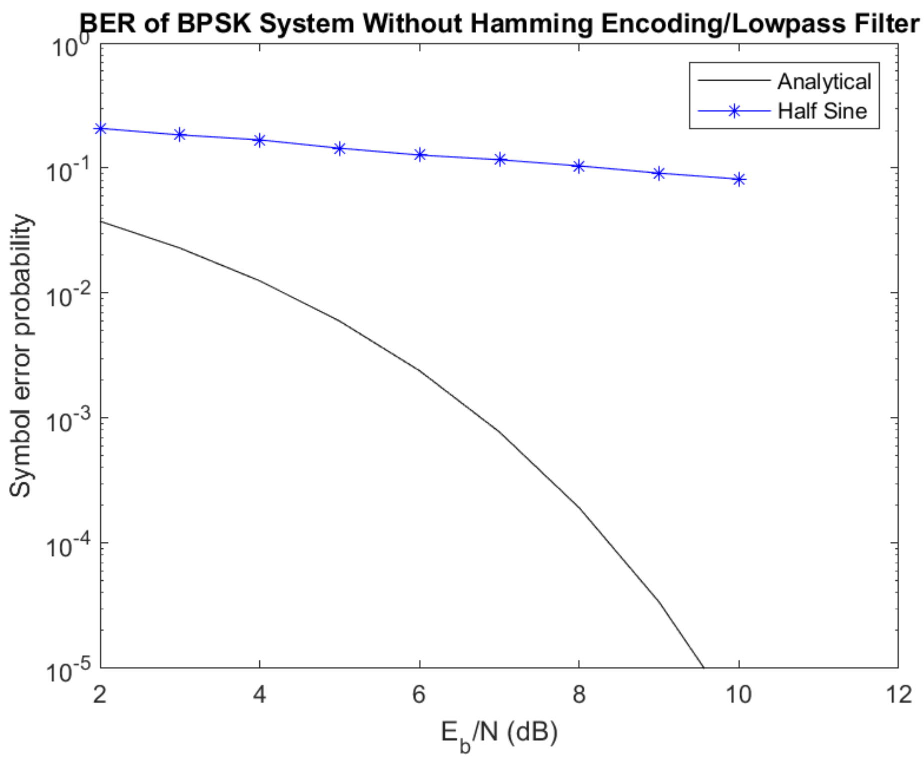

Figure 19: BER Plot of the BPSK Communication System, without the Hamming coding or lowpass filter.

Conclusion

The techniques covered in this lab allowed the successful application of a BPSK communication channel. There could be several upgrades made to improve the performance, with various trade offs to consider. Ideas were covered in the second paragraph of part 3. Other modulation techniques and pulse shapes can be utilized in a similar fashion to create communication channels, using principles covered in this lab.

Matlab Code

Exsample.m

```
% example of sampling, quantization, and zero-order hold
clear; clf;
td=0.002;
t=[0:td:1.];
xsig = sin(2*pi*t)-sin(6*pi*t);
Lsig=length(xsig);

% ts=0.02; %new sampling rate = 50Hz
ts=0.17; % aliasing sample rate = 6hz
Nfactor=ts/td;
% send the signal through a 16-level uniform quantizer
[s_out,sq_out,sqh_out,Delta,SQNR]=sampandquant(xsig,4,td,
    ts);
% receive 3 signals:
% - sampled signal s_out
% - sampled and quantized signal sq_out
% - sampled, quantized and zero-order hold signal sqh_out

% calculate the Fourier Transform
Lfft = 2^ceil(log2(Lsig)+1);
Fmax=1/(2*td);
Faxis=linspace(-Fmax,Fmax,Lfft);
Xsig=fftshift(fft(xsig,Lfft));
S_out=fftshift(fft(s_out,Lfft));
Sqh_out=fftshift(fft(sqh_out,Lfft));
% Examples of sampling and reconstruction using
% - ideal impulse train through LPF
% - flat top pulse reconstruction through LPF

% plot the original signal and the sample signals in time
    and frequency
% domain
figure(1);
subplot(311); sfig1a=plot(t,xsig,'k');
hold on; sfig1b=plot(t,sqh_out(1:Lsig),'b'); hold off;
xlabel('time (sec)');
title('Signal {\it g}({\it t}) and its uniform samples');
subplot(312); sfig1c=plot(Faxis,abs(Sqh_out));
xlabel('frequency (Hz)');
axis([-150 150 0 300])
set(sfig1c, 'Linewidth',1); title('Spectrum of {\it g}
    ({\it t})');
```

```
subplot(313); sfig1d=plot(Faxis,abs(S_out));
xlabel('frequency (Hz)');
axis([-150 150 0 300/Nfactor])
set(sfig1c, 'Linewidth',1); title('Spectrum of {\it g}_T({\it t})');
% calculate the reconstructed signal from ideal sampling and ideal LPF
% Maximum LPF bandwidth equals to BW=floor((Lfft/Nfactor)/2);'
BW=10; % bandwidth is no larger than 10hz
H_lpf = zeros(1,Lfft); H_lpf(Lfft/2-BW:Lfft/2+BW-1)=1; % ideal LPF
S_recv=Nfactor*S_out.*H_lpf;
s_recv=real(ifft(fftshift(S_recv))); % reconstructed f-domain
s_recv=s_recv(1:Lsig); % reconstructed time domain

% plot the ideally reconstructed signal in time and frequency domain
figure(2);
subplot(211); sfig2a=plot(Faxis,abs(S_recv));
xlabel('frequency (Hz)');
axis([-150 150 0 300]);
title('Spectrum of ideal filtering (reconstruction)');
subplot(212); sfig2b=plot(t,xsig,'k-.',t,s_recv(1:Lsig),'b');
legend('original signal', 'reconstructed signal');
xlabel('time (sec)');
title('original signal versus ideally reconstructed signal');
set(sfig2b,'Linewidth',2);
% non-ideal reconstruction
ZOH=ones(1,Nfactor);
s_ni=kron(downsample(s_out,Nfactor),ZOH);
S_ni=fftshift(fft(s_ni,Lfft));
S_recv2=S_ni.*H_lpf; % ideal filtering
s_recv2=real(ifft(fftshift(S_recv2))); % reconstructed f-domain
s_recv2=s_recv2(1:Lsig); % reconstructed t-domain

% plot the ideally reconstructed signal in the time and frequency domain
figure(3)
subplot(211); sfig3a=plot(t,xsig,'b',t,s_ni(1:Lsig),'b');
xlabel('time (sec)');
title('original signal versus flat-top reconstruction');
```

```
subplot(212); sfig3b=plot(t,xsig,'b',t,s_recv2(1:Lsig),'b--');
legend('original signal','LPF reconstruction');
xlabel('time (sec)');
title('original and flat-top reconstruction after LPF');
```

sampandquant.m

```
function [s_out,sq_out,sqh_out,Delta,SQNR]=sampandquant(sig_in,L,td,ts)

if(rem(ts/td,1)==0)
    nfac=round(ts/td);
    p_zoh=ones(1,nfac);
    s_out=downsample(sig_in,nfac);
    [sq_out, Delta, SQNR]=uniquan(s_out,L);
    s_out=upsample(s_out,nfac);
    sqh_out=kron(sq_out,p_zoh);
    sq_out=upsample(sq_out,nfac);
else
        warning('Error! ts/td is not an int');
        s_out=[];sq_out=[];sqh_out=[];Delta=[];SQNR=[];
end
end
```

uniquan.m

```
function [q_out , Delta ,SQNR]=uniquan( sig_in ,L)

sig_pmax=max( sig_in ) ;
sig_nmax=min( sig_in ) ;
Delta=(sig_pmax−sig_nmax ) /L;
q_level=sig_nmax+Delta / 2: Delta : sig_pmax−Delta / 2;
L_sig=length ( sig_in ) ;
sigp=(sig_in −sig_nmax ) / Delta+1/2;
qindex=round ( sigp ) ;
qindex=min( qindex ,L) ;
q_out=q_level ( qindex ) ;
SQNR=20∗log10 (norm( sig_in ) /norm( sig_in −q_out ) ) ;
end
```

PAM2_eye.m

```
% PAM2_eye.m
clear all;clf;
data = sign(randn(1,400)); % generate 400 random bits
Tau = 32; % Sumbol Period
Tped=0.001;
dataup=upsample(data,Tau);
yrz=conv(dataup,prz(Tau));
yrz=yrz(1:end-Tau+1);
ynrz=conv(dataup,pnrz(Tau));
ynrz=ynrz(1:end-Tau+1);
ysine=conv(dataup,psine(Tau));
ysine=ysine(1:end-Tau+1);
Td=4;
yrcos=conv(dataup,prcos(0.5,Td,Tau));
yrcos=yrcos(Td*Tau-Tau/2:end-2*Td*Tau+1);
txis=(1:1000)/Tau;

figure(1)
subplot(311);w1=plot(txis,ynrz(1:1000)); title('(a) NRZ
    waveform');
axis([0 1000/Tau -1.5 1.5]); xlabel('time unit (T sec)');
subplot(312);w2=plot(txis,ysine(1:1000)); title('(a) Half
    -sine waveform');
axis([0 1000/Tau -1.5 1.5]); xlabel('time unit (T sec)');
subplot(313);w3=plot(txis,yrcos(1:1000)); title('(a)
    Raised-cosine waveform');
axis([0 1000/Tau -1.5 1.5]); xlabel('time unit (T sec)');

Nwidth=2;
edged=1/Tau;
figure(2)
subplot(221)
eye1=eyeplot(yrz,Nwidth,Tau,0); title('(a) RZ eye diagram
    ')
axis([-edged Nwidth+edged, -1.5, 1.5]); xlabel('time unit
    (T second)');
subplot(222)
eye2=eyeplot(ynrz,Nwidth,Tau,0); title('(b) NRZ eye
    diagram')
axis([-edged Nwidth+edged, -1.5, 1.5]); xlabel('time unit
    (T second)');
subplot(223)
eye3=eyeplot(ysine,Nwidth,Tau,0); title('(c) Half-sine
    eye diagram')
```

```
axis([-edged Nwidth+edged, -1.5, 1.5]); xlabel('time unit (T second)');
subplot(224)
eye4=eyeplot(yrcos,Nwidth,Tau,0); title('(d) Raised-cosine eye diagram')
axis([-edged Nwidth+edged, -1.5, 1.5]); xlabel('time unit (T second)');
```

Pulse Shape Functions

```
function pout=pnrz(T)
     pout=ones(1,T);
end
```

```
function pout=prz(T)
pout=[zeros(1,T/4) ones(1,T/2) zeros(1,T/4)];
end
```

```
function pout=psine(T)
pout=sin(pi*[0:T-1]/T);
end
```

```
function y=prcos(rollfac,length,T)
y=rcosfir(rollfac,length,T,1, 'normal');
% y=4*rcosdesign(rollfac,length,T,'normal');
end
```

BPSK.m

```
% Part 3 of the lab
clear all;clf;

%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% -- Data Preparation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Generate the random bits
L = 4000;
original_data = sign(randn(1,L)); % generate L random bits
original_data = (original_data+1)/2; % convert to binary

% Perform Hamming (7,4) encoding to reduce errors
G = [1 0 0 0 1 1 1; ...
     0 1 0 0 1 1 0; ...
     0 0 1 0 1 0 1; ...
     0 0 0 1 0 1 1;];
H = [1 1 1 0 1 0 0; ...
     1 1 0 1 0 1 0;...
     1 0 1 1 0 0 1;];

data = [];
tmp = [];

% Perform Encoding
for idx=1:4:L
    tmp = mod(original_data(idx:idx+3)*G,2);
    data = [data,tmp];
end
L = length(data); % the actual length of data is used later
data = 2*data-1; % convert zeros to -1's

%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% -- Pulse Shaping
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% First upsample the data to avoid inter-symbol interference
Tau = 4; % Symbol Period
dataup=upsample(data,Tau);

% Then convolve the data (delta train) with your pulses
```

```
    to represent your
% data
pulseShape = psine(Tau);
pulseShapedData = conv(dataup,pulseShape);

% Create the matched filter by time reversing the pulseshape
matchedFilter=pulseShape(end:-1:1);

% Plot of our data
figure(1)
txis=(1:1000)/Tau;
subplot(411); w1=plot(txis,pulseShapedData(1:1000)); title('(a) Half-sine Data Pulse Train');
axis([0 1000/Tau -1.5 1.5]); xlabel('time unit (T sec)');

%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% -- Modulation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Modulate our pulse shaped data
freq = 2*pi*10000; % frequency in radians
t = linspace(0,L*Tau,length(pulseShapedData));
modulated_x=pulseShapedData.*(sqrt(2)*cos(freq*t));
txis=(1:1000)/Tau;
subplot(412); w2=plot(txis,modulated_x(1:1000)); title('(b) Modulated Data');
axis([0 1000/Tau -1.5 1.5]); xlabel('time unit (T sec)');


%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% -- Noise Channel w/ Demodulation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
signalLength = length(pulseShapedData);
BER=[];
noiseq=randn(signalLength,1);
for i=1:10
    % create AWGN
    Eb2N(i) = i;
    Eb2N_num=10^(Eb2N(i)/10);
    Var_n=1/(2*Eb2N_num);
    signois=sqrt(Var_n);
    awgnois=signois*noiseq;

    % Add noise to the signals at the channel output
    rx_signal = modulated_x+awgnois';
```

```

    % Demodulate by multiplying by the carrier (see fig 6.34 in the book)
    demodulatedSignal = rx_signal(1:length(t)).*(sqrt(2)*cos(freq*t));

    % Apply a Low Pass Filter to cut out the noise *note: this introduces
    % issues
    demodulatedSignal = lowpass(demodulatedSignal, freq, freq*3,'Steepness',0.8);

    % Apply the matched filter, which is the time reversed pulse shape
    output=conv(demodulatedSignal,matchedFilter);

    % Sample the signal. Note we are sampling the middle of the pulse,
    % hence the Tau/2 startpoint
    sampledOutput = demodulatedSignal(Tau/2+1:Tau:end);

    % Use threshold detection to generate received message
    receivedBits = sign(sampledOutput);
    receivedBits = (receivedBits+1)/2;

    % Decode the Hamming (7,4) code to correct errors
    tmp=[];
    decodedBits=[];
    for idx=1:7:length(receivedBits)-1
        tmp = mod(receivedBits(idx:idx+6)*H',2); % grab 7 bits at a time
        if(sum(tmp)==0)
            % Remember that the first two bits are the received info
            decodedBits = [decodedBits,receivedBits(idx:idx+3)]; % save the data bits
        else
            % use the syndrome to get the index where the bit error is
            errIdx = bin2dec(num2str(tmp));
            xor(receivedBits(idx+errIdx),1); % xor to flip the bit
            decodedBits = [decodedBits,receivedBits(idx:idx+3)];
        end
```

```
        end

        % convert received bits back to BPSK form
        receivedBits=2*receivedBits-1;

        % Determine how many bits are correctly received and
            plot the BER
        BER=[BER;sum(abs(data-receivedBits(1:L)))/L]; % for
            all received bits
%       BER=[BER;sum(abs(original_data-decodedBits(1:end)))
    /L]; % for the encoded bits

        Q(i)=0.5*erfc(sqrt((2*Eb2N_num)/2));

end


% Plot the demodulated signal
txis=(1:1000)/Tau;
subplot(413); w3=plot(txis,demodulatedSignal(1:1000));
    title('(c) Demodulated Signal');
axis([0 1000/Tau -1.5 1.5]); xlabel('time unit (T sec)');

txis=(1:1000)/Tau;
subplot(414); w4=plot(txis,receivedBits(1:1000)); title('
    (d) Decoded Signal Data');
axis([0 1000/Tau -1.5 1.5]); xlabel('time unit (T sec)');
figure(2)

grid on;
% subplot(111)
figber=semilogy(Eb2N,Q,'k-',Eb2N,BER,'b-*');
axis([2 12 0.99e-5 1]);
title('BER of BPSK System With Hamming Encoding');
legend('Analytical', 'Half Sine');
fx=xlabel('E_b/N (dB)'); fy=ylabel('Symbol error
    probability');
```