ECE 3793
Signals and Systems

MATLAB Project #1 Report

Gallogly College of Engineering University of Oklahoma
November 17, 2017
By Abel Ararso and Thomas Jung

I. Introduction

The Purpose of this Computer project is to learn and get better at taking digital communication signals. We mostly focused on Frequency shift keyed signals(FSK) and Phase shift keyed signals(PSK). A Frequency shift keyed signal transmitter sends a slightly different frequency in each symbol period while in Phase shifted keyed signal, the transmitter sends a slightly different phase bit sequence.

II. Methods

Part 1

We start the project by writing scripts to read the four-different data file which is given on MATLAB and we setup a parameter for each of them. Since a sampling rates were provided, we could compute the number of samples per symbols by using equation (1). Then, we use the reshape command to organize the data into symbols.

$$n = F_s \cdot T \qquad (1)$$

$$L = \frac{length\ (x)}{n}$$

After that, we sample the correct rate using equation (2). To perform equation 2, we first took out unnecessarily given equation out such as Arect() from (2) since they were equivalent to 1 and use dot product of the result form the reshape command (2). Following that, we store the result in to a scalar array. We then use a real command to the frequency of the selected stored scalar array to separate complex part from the real part. To perform this, the max command will take the highest frequency corresponding to the largest real output of the dot product and store it in to index. We start the index of the array from 1 and subtract 1 from the result. Finally, we cover the detected frequencies in binary number by using *dec2bin* command in MATLAB.

$$x(t) = \exp(j2\pi \Delta F m_k t) \qquad (2)$$

Where M_k is the frequency index of the k_{th} symbol

For Phase shift keyed signals, we follow exactly similar steps however we were more focused on transmitter slightly different phase bit sequence. We do the dot product on the result form reshape command in to result from equation (3) united of equation (2). Then, we store the result in to a scalar array by using max command from the highest phase corresponding to the largest real output of the dot product. Just like frequency shift keyed signals, we start the index of the array from 1 and subtract 1 from the result. Also, instead of using *dc2bin* command we use *dc2grycod* to obtain a gray code.

$$x(t) = \exp(jm_k 2\pi/M) \qquad (3)$$

Part 2

We compute the energy per bit signal by taking the integral of one period. Since we know the integral of a power is always 1, by using equation (4) we determined energy per symbol is equal to 1 micro second. To find energy per bit we divide energy per symbol by k, k is equal to log2 (M).

To define the ratios of interest, range from -10 dB to 15 dB to find SNR, we use *db2mag* command to convert the dB to decimal. By using equation (5) and (6) we compute the values of N_o and *Pn.* We then use the computed value P_n and used it in equation (7) to add noise to the given data file. Then, we use a for loop to add different SNR values to the noise.

To count the number of differences (errors) from the result of part 1, we subtract the bit sequence difference from part 2. After that we divide the error by range to obtain a BER. Using MATLAB's *semilogy* command we Plot the BER on the vertical axis and SNR on the horizontal axis.

$$E_s = \int_0^T 1\,dt \qquad (4)$$

$$n_0 = \frac{E_b}{SNR} \qquad (5)$$

$$p_1 = n_0 \cdot B \qquad (6)$$

$$noise = sqrt(P_n/2) * (randn(N,1) + 1j * randn(N,1); \qquad (7)$$

Since our computer takes a long time to run part 1 and part 2, we should combine the two-part in to one big loop which is going to assist in reducing the run time by around 10 minutes for each data file. It was observed that the values stored within the binary array were 49 or 48 instead of the expected 1 or 0, and it was realized that this 49 or 48 represents asci vector value that is equivalent to 1 or 0 in decimal. Thus, we had to force it to be 1 or 0 in our code.

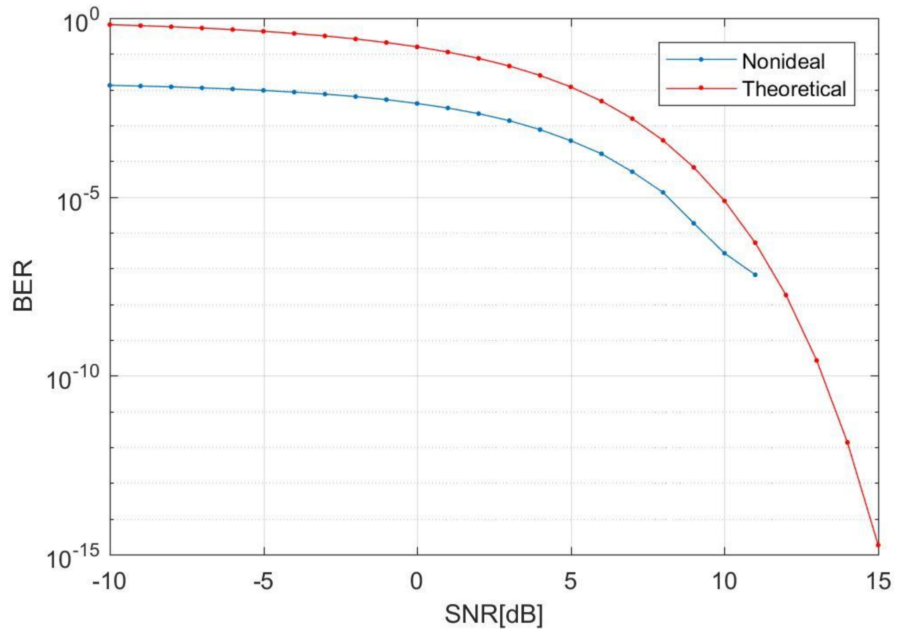

Figure 1. *Four Frequency shift keyed signals(FSK).*

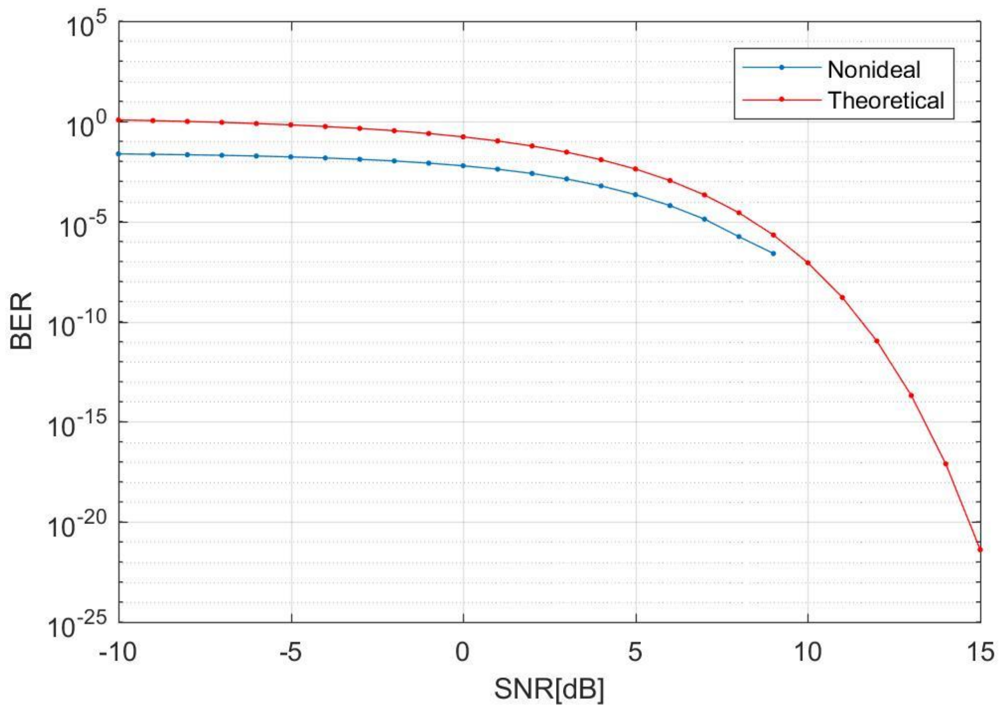

Figure 2. *Eight Frequency shift keyed signals(FSK).*

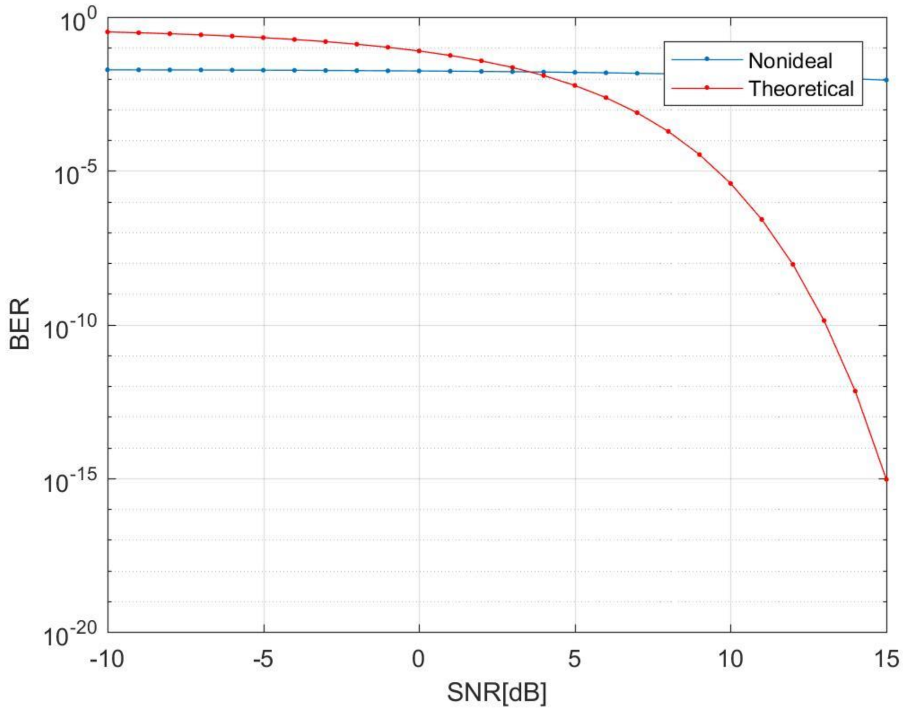

Figure 3. *Four Phase shift keyed signal*

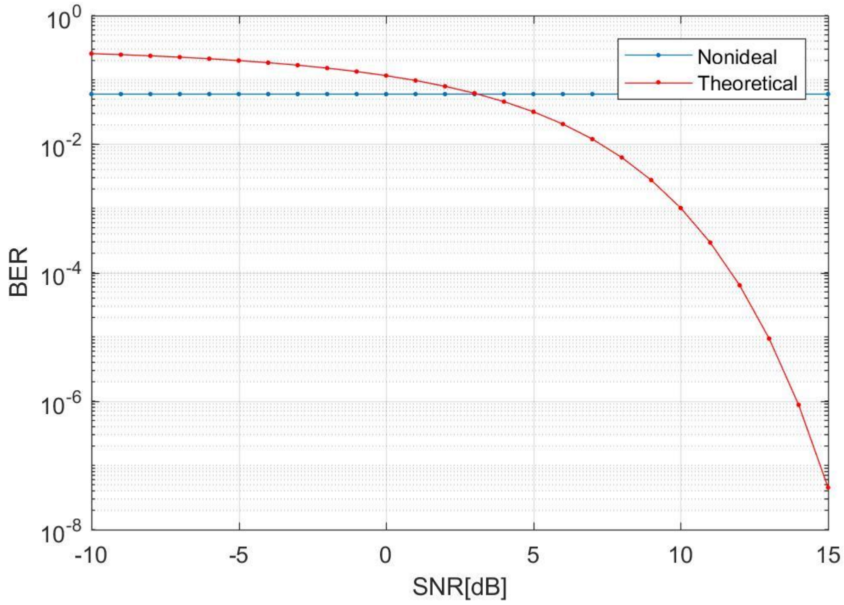

Figure 4. *Eight* Phase shift keyed signal

Part 3

In this part, we used the FFT command to calculate the frequency spectrum of the different communication signals. Even if the data is long, having lots of symbols is important to obtain accurate result. Therefore, we computed the frequency spectrum with 200 symbols size, which meant 10000 samples. Equation (8) was used as well as power in dB conversion was also used (10*log(10)(X)) in order to plot the result.

$$x = fftshift\big(fft(x, Lfft)\big); \qquad (8)$$

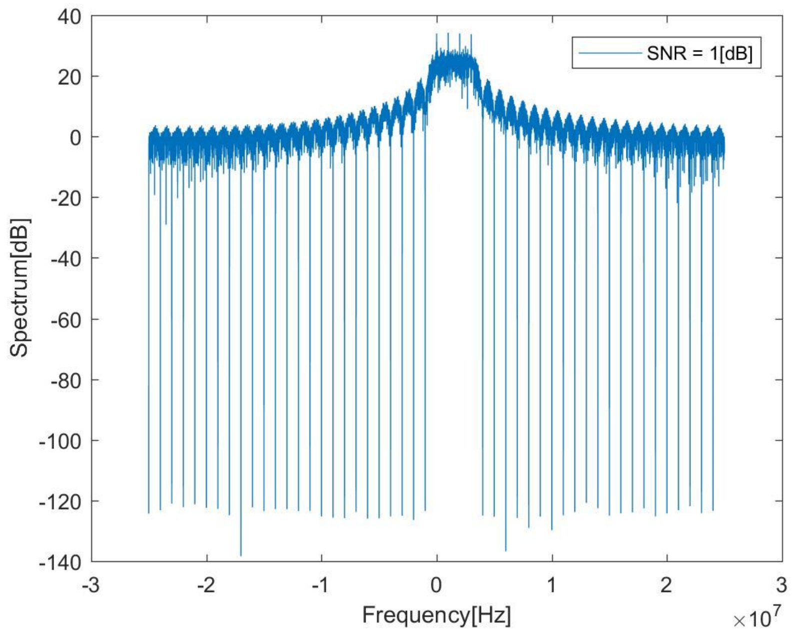

Figure 1. *Four Frequency shift keyed signals(FSK).*

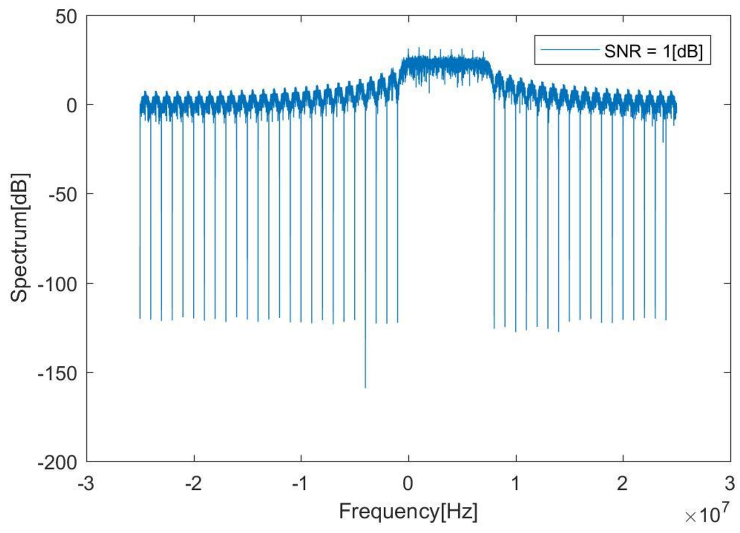

Figure 2. *Eight Frequency shift keyed signals(FSK).*

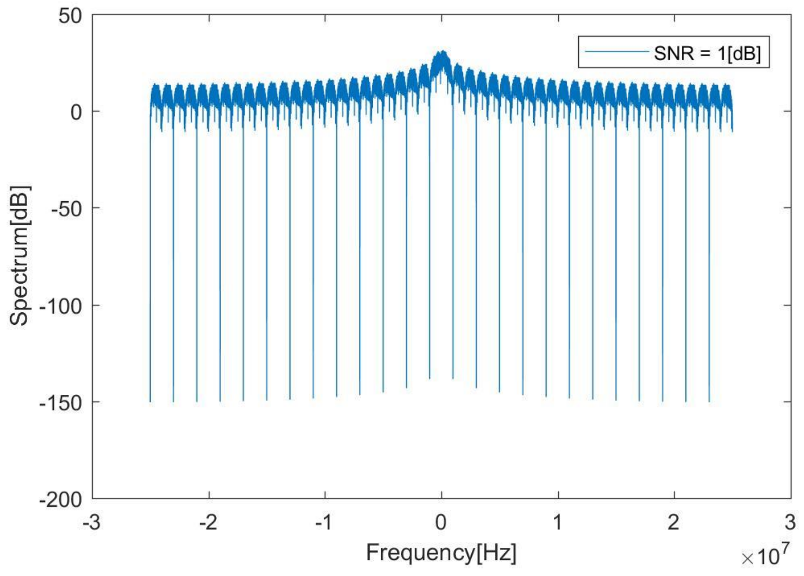

Figure 3. *Four Phase shift keyed signal*

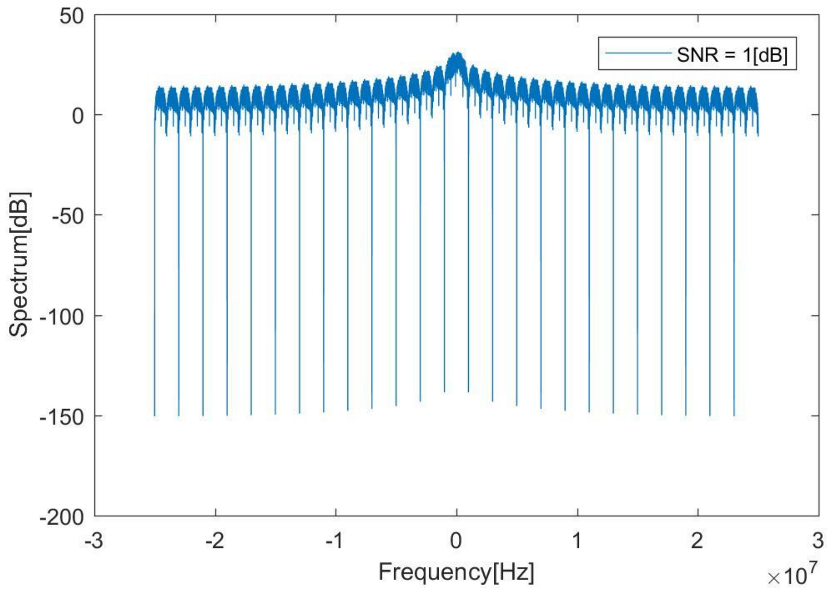

Figure 4. *Eight Phase shift keyed signal*

III. Result.

As we can see from the Figure 1 through 4, it was supposed to be converging toward .5 since this is using binary and it means there is 50% chance of 1 and 0. 4/8pdk in Fig. 3 and Fig. 4 also seem to be wrong as non-ideal frequency is showing a straight line instead of similar conversion to the theoretical values. However, Fig. 1 and 2 shows similar curve with theoretical signal to noise ratio(SNR) of bit rate error(BER), which means it's only off by a some type of shift possibly… also graphs are showing that as SNR is increasing BET is decreasing, which makes sense.

In Figure 5~8, it shows frequency vs spectrum plot that can be observed after plotting. We can clearly see in the mean area of the probability distribution that in Fig.5 shows 4 clear lines pop up toward north and Fig. 6 shows 8 clear lines pop up toward north again. These are the frequency that should be observed. 4 for 4fsk and 8 for 8fsk. Advantages of using 4fsk is that it saves bandwidth space of 4fsk if 4fsk was used rather than 8fsk. The disadvantage of using 4fsk would be being able to only send 2 bits of information. Advantage of using 8fsk is being able to send 3 bits of information while the disadvantage is that it takes 8 MHz to use it and that is 4fsk more of a bandwidth than 4fsk that could be used by someone else or for something else. Another problem is the Fig.7 and 8 with the 4/8pdk not showing any clear peaks like those of Fig.5 and Fig.6… this may be due to those presenting phasor rather than frequency.

IV. Conclusion.

In conclusion, the project helped relate the concepts of frequency shit keyed signal and phase shift keyed signal. Before this project, due to having little knowledge and experience handling matrices, so much effort and time had to be used in trouble shooting and fixing matrix related issues. In addition, understanding and remembering the variables for each key concept of this project was a difficult task. It took a long reservation of determination and desperation to get it going often times while facing so many unknowns. But now, we have a better understanding of how to analyze data based on what we have been provided.

V. Appendix

4 FSK

```
% for making er'thing neat and stuff. >:]

clear all;

close all;

clc

% reading in the datafile from 4FSK.mat and storing the data into a

% variable specifically for 4FSK.

% then clearing up the extra space by removing x

load('4FSK.mat', 'x');

x_fsk4 = x;

clear x;

% M, is the value of our M-FSK; here being 4 for 4-FSK

M = 4;

% amplitude, A is given as 1: really unnecessary to even be here ?

A = 1;

% symbol duration, T = 1 micro-second

T = 1 * 10^(-6);

% delta frequency = 1 MHz

df = 1 * 10^6;

% m = frequency index ranging from 0 to (M-1)

m = 0:(M-1);

% sampling rate = 50 MHz

Fs = 50*10^6;

% conversion of sampling rate to seconds called here as Ts

Ts = 1/Fs;

% Energy per symbol = integral of power from 0 to T, which results in T

% since power is 1 ?
```

```
Es=T;
% just a given
k = log2(M);
% energy per bit
Eb=Es/k;
% signal-to-noise ratio(SNR) in dB from -10 to 15, leads to 26 iterations
SNR_dB =-10:15;
% SNR converted to power
SNR = db2pow(SNR_dB);
% average noise power per unit frequency = Energy per bit over signal-to-noise ratio
No=Eb./SNR;
% bandwidth = Fs, sampling rate (50 MHz)
B = Fs;
% noise power
Pn=No.*Fs;
% number of samples in the data loaded
N = length(x_fsk4);
% symbols, n (50)
n = T*Fs;
% number of symbols
L = length(x_fsk4)/n;
% reshaping our matrix using loaded data as rows being symbols and columns
% being columns
symbol = reshape(x_fsk4,n,L);
% t ranges from 0Ts to 49Ts in intervals of Ts, which ends up with 50
t = 0:Ts:(49*Ts);
% is the varied frequency that will be needed to dot product with the data
% loaded. m leads to 4 or 8 varied frequencies while t leads samples in the
% symbols -> ending up with 4x50.
y = exp(j*2*pi*m.*t'.*df);
% given theoretical ber rate of SNR
SNR_ber=(M/2)*qfunc(sqrt(Es./No));

% preallocation done previously before nested for loops are engaged for
% better efficiency for scalar value of ideal frequency, nonideal freq.,
% bit,bit2,ber
bit = zeros(L, 2);
noise = zeros(length(x_fsk4),1);
nonideal = zeros(length(x_fsk4),1);
nonidealSymbol = zeros(n,L);
nonidealScalar = zeros(M,1);
scalar = zeros(M,1);
bit2 = zeros(L, 2);
ber = zeros(1,length(SNR));

% nested for loop
% z iterating through 26 different values of SNR
% i iterating through 600000 different values of number of symbols
% j iterating through 4/8 different values of number of frequencies
for z=1:length(SNR)
    % given noise equation for random change to ideal frequency
    % leading to nonideal frequency that can be compared to ideal freq.
    % adding the noise to the loaded data(ideal)
    % reshaping the nonideal symbol into proper matrix shape that we want
    % in this case 50x600000
    noise=sqrt(Pn(z)./2).*(randn(N,1)+1i*randn(N,1));
```

```
nonideal=noise+x_fsk4;
nonidealSymbol = reshape(nonideal,[n,L]);

for i = 1:L
    for j = 1:M
        % iterating through ideal and nonideal scalar values after
        % M-dot producting between ideal and nonideal data with varied
        % frequency (M-FSK)
        % dot producting each symbol with various frequencies will
        % result in scalar values of j (which is 1:M). for both
        scalar(j,1) = dot(symbol(:,i), y(:,j));
        nonidealScalar(j,1) = dot(nonidealSymbol(:,i), y(:,j));
    end

    % since scalar will contain real and imaginary values. we will take
    % out the real part and take out max of those real part out of M
    % number of them.
    R = real(scalar);
    [~,idx] = max(R);
    % since index starts from 1 to 4. Need to substract 1 for each
    % index values and convert the index into binary and stack them
    % accordingly -- and this is for ideal
    f_idx = idx-1;
    bit(i,:) = dec2bin(f_idx);

    % if bit somehow gets stored in ascii value. we need to force the
    % ascii value into decimal value.
    % according to internet. ascii value of 49 = 1; and 48 = 0
```

```
if bit(i,1)  == 49
    bit(i,1) = 1;
elseif bit(i,1) == 48
    bit(i,1) = 0;
end

% same thing
if bit(i,2) == 49
    bit(i,2) = 1;
elseif bit(i,2) == 48
    bit(i,2) = 0;
end

% for nonideal values the same step to get binary numbers for those
R2 = real(nonidealScalar);
[~,idx2] = max(R2);
f_idx2 = idx2-1;
bit2(i,:) = dec2bin(f_idx2);

% same old ascii value conversion
if bit2(i,1)  == 49
    bit2(i,1) = 1;
elseif bit2(i,1) == 48
    bit2(i,1) = 0;
end

if bit2(i,2) == 49
    bit2(i,2) = 1;
```

```
        elseif bit2(i,2) == 48
            bit2(i,2) = 0;
        end
    end
    % using nnz function with substracting ideal and nonideal binary to
    % count up the number of error to be devided with number of total
    % sample to get BER, bit error rate.
    % this entire loop handles one iteration of SNR per run for space
    % efficiency since only BER is needed for graphing
    error_cnt=nnz(bit-bit2);
    ber(z)=error_cnt./(N);
end

% figure for part 2 using smilogy with proper labels and grid
figure,
semilogy(SNR_dB,ber,'.-');
grid on;
xlabel('SNR[dB]');
ylabel('BER');
hold on;
semilogy(SNR_dB,SNR_ber,'.-r');
grid on;
xlabel('SNR[dB]');ylabel('BER');
legend('Nonideal','Theoretical');
% graph for part 3
% Lfft should be higher than fft so i made it 1000 more than fft
% took out 200 symbols, ending with 10000
Lfft = 11000;
```

```
partialNumber = 200*50;

partialSym = x_fsk4(1:partialNumber);


%fft = fourier transform

% using given equation and converting that into power in dB

x3 = fftshift(fft(partialSym,Lfft));

x3_dB = 10*log10(abs(x3));

% given faxis equation

faxis = -0.5*Fs + (0:(Lfft-1))*Fs/Lfft;

% plotting the graphs

figure,

plot(faxis,x3_dB(:,1));

hold on;

xlabel('Frequency[Hz]');

ylabel('Spectrum[dB]');

legend('SNR = 1[dB]');
```

8 FSK

```
% for making er'thing neat and stuff. >:]

clear all;

close all;

clc


% reading in the datafile from 8FSK.mat and storing the data into a

% variable specifically for 8FSK.

load('8FSK.mat', 'x');

x_fsk8 = x;

clear x;
```

```
% M, is the value of our M-FSK; here being 8 for 8-FSK
M = 8;
% amplitude, A is given as 1: really unnecessary to even be here ☺
A = 1;
% symbol duration, T = 1 micro-second
T = 1 * 10^(-6);
% delta frequency = 1 MHz
df = 1 * 10^6;
% m = frequency index ranging from 0 to (M-1)
m = 0:(M-1);
% sampling rate = 50 MHz
Fs = 50*10^6;
% conversion of sampling rate to seconds called here as Ts
Ts = 1/Fs;
% Energy per symbol = integral of power from 0 to T, which results in T
% since power is 1 ☺
Es=T;
% just a given
k = log2(M);
% energy per bit
Eb=Es/k;
% signal-to-noise ratio(SNR) in dB from -10 to 15, leads to 26 iterations
SNR_dB =-10:15;
% SNR converted to power
SNR = db2pow(SNR_dB);
% average noise power per unit frequency = Energy per bit over signal-to-noise ratio
No=Eb./SNR;
```

```
% bandwidth = Fs, sampling rate (50 MHz)
B = Fs;
% noise power
Pn=No.*B;
% number of samples in the data loaded
N = length(x_fsk8);
% symbols, n (50)
n = T*Fs;
% number of symbols (20Million)
L = length(x_fsk8)/n;

% reshaping our matrix using loaded data as rows being symbols and columns
% being columns
symbol = reshape(x_fsk8,n,L);
% t ranges from 0Ts to 49Ts in intervals of Ts, which ends up with 50
t = 0:Ts:(49*Ts);
% is the varied frequency that will be needed to dot product with the data
% loaded. m leads to 4 or 8 varied frequencies while t leads samples in the
% symbols -> ending up with 4x50.
y = exp(j*2*pi*m.*t'.*df);
% given theoretical ber rate of SNR
SNR_ber=(M/2)*qfunc(sqrt(Es./No));

% preallocation done previously before nested for loops are engaged for
% better efficiency for scalar value of ideal frequency, nonideal freq.,
% bit,bit2,ber
noise = zeros(length(x_fsk8),1);
nonideal = zeros(length(x_fsk8),1);
```

```
nonidealSymbol = zeros(n,L);
nonidealScalar = zeros(M,1);
scalar = zeros(M,1);
bit = zeros(L, 3);
bit2 = zeros(L, 3);
ber = zeros(1,length(SNR));

% nested for loop
% z iterating through 26 different values of SNR
% i iterating through 400000 different values of number of symbols
% j iterating through 4/8 different values of number of frequencies
for z=1:length(SNR)
    noise=sqrt(Pn(z)./2).*(randn(N,1)+1i*randn(N,1));
    nonideal=noise+x_fsk8;
    nonidealSymbol = reshape(nonideal,[n,L]);

    % given noise equation for random change to ideal frequency
    % leading to nonideal frequency that can be compared to ideal freq.
    % adding the noise to the loaded data(ideal)
    % reshaping the nonideal symbol into proper matrix shape that we want
    % in this case 50x400000
    for i = 1:L
        for j = 1:M
            % iterating through ideal and nonideal scalar values after
            % M-dot producting between ideal and nonideal data with varied
            % frequency (M-FSK)
            % dot producting each symbol with various frequencies will
            % result in scalar values of j (which is 1:M). for both
            scalar(j,1) = dot(symbol(:,i), y(:,j));
            nonidealScalar(j,1) = dot(nonidealSymbol(:,i), y(:,j));
        end

        % since scalar will contain real and imaginary values. we will take
        % out the real part and take out max of those real part out of M
        % number of them.
        R = real(scalar);
        [~,idx] = max(R);
        % since index starts from 1 to 4. Need to substract 1 for each
        % index values and convert the index into binary and stack them
        % accordingly -- and this is for ideal
        f_idx = idx-1;
        bit(i,:) = dec2bin(f_idx, 3);

        % if bit somehow gets stored in ascii value. we need to force the
        % ascii value into decimal value.
        % according to internet. ascii value of 49 = 1; and 48 = 0
        if bit(i,1)  == 49
            bit(i,1) = 1;
        elseif bit(i,1) == 48
            bit(i,1) = 0;
        end
        % same thing
        if bit(i,2) == 49
            bit(i,2) = 1;
        elseif bit(i,2) == 48
            bit(i,2) = 0;
```

```
end
% same thing
if bit(i,3) == 49
    bit(i,3) = 1;
elseif bit(i,3) == 48
    bit(i,3) = 0;
end

% for nonideal values the same step to get binary numbers for those
R2 = real(nonidealScalar);
[~,idx2] = max(R2);
f_idx2 = idx2-1;
bit2(i,:) = dec2bin(f_idx2, 3);

% same old ascii value conversion
if bit2(i,1)  == 49
   bit2(i,1) = 1;
elseif bit2(i,1) == 48
   bit2(i,1) = 0;
end
% same thing
if bit2(i,2) == 49
   bit2(i,2) = 1;
elseif bit2(i,2) == 48
   bit2(i,2) = 0;
end
% same thing
if bit2(i,3) == 49
```

```
            bit2(i,3) = 1;
        elseif bit2(i,3) == 48
            bit2(i,3) = 0;
        end
    end

    % using nnz function with substracting ideal and nonideal binary to
    % count up the number of error to be devided with number of total
    % sample to get BER, bit error rate.
    % this entire loop handles one iteration of SNR per run for space
    % efficiency since only BER is needed for graphing
    error_cnt=nnz(bit-bit2);
    ber(z)=error_cnt./(N);
end

% figure for part 2 using smilogy with proper labels and grid
figure,
semilogy(SNR_dB,ber,'.-');
grid on;
xlabel('SNR[dB]');
ylabel('BER');
hold on;

semilogy(SNR_dB,SNR_ber,'.-r');
grid on;
xlabel('SNR[dB]');
ylabel('BER');
legend('Nonideal','Theoretical');
```

```
% graph for part 3
% Lfft should be higher than fft so i made it 1000 more than fft
% took out 200 symbols, ending with 10000
Lfft = 11000;
partialNumber = 200*50;
partialSym = x_fsk8(1:partialNumber);

%fft = fourier transform
% using given equation and converting that into power in dB
x3 = fftshift(fft(partialSym,Lfft));
x3_dB = 10*log10(abs(x3));
% given faxis equation
faxis = -0.5*Fs + (0:(Lfft-1))*Fs/Lfft;
% plotting the graphs
figure,
plot(faxis,x3_dB(:,1));
hold on;
xlabel('Frequency[Hz]');
ylabel('Spectrum[dB]');
legend('SNR = 1[dB]');
```

4 PSK

```
% for making er'thing neat and stuff. >:]
clear all;
close all;
clc

% reading in the datafile from 4PSK.mat and storing the data into a
% variable specifically for 4PSK.
% then clearing up the extra space by removing x
load('8PSK.mat', 'x');
x_psk4 = x;
clear x;
% the names of these variables are the same as ones stated in the 4/8 fsk
M = 4;
A = 1;
T = 1 * 10^(-6);
df = 1 * 10^6;
Fs = 50*10^6;
m = 0:(M-1);
Fs = 50*10^6;
Ts = 1/Fs;
Es=T;
k = log2(M);
Eb=Es/k;
SNR_dB =-10:15;
SNR = db2pow(SNR_dB);
No=Eb./SNR;
Pn=No.*Fs;
N = length(x_psk4);
n = T*Fs;
L = length(x_psk4)/n;
% reshaping the symbol to 50x600000 (symbol by number of symbols)
symbol = reshape(x_psk4,n,L);
% given frequency equation (A is 1 so ignored and so is rect function for
% the same reason being 1.
```

```
y = exp(j*m.*2*pi/M);

% theoretical SNR bit error rate
SNR_ber=(2/(log2(M)))*qfunc((sqrt((2*Es)./No)*sin(pi/M)));

% preallocation done previously before nested for loops are engaged for
% better efficiency for scalar value of ideal frequency, nonideal freq.,
% bit,bit2,ber
noise = zeros(length(x_psk4),1);
nonideal = zeros(length(x_psk4),1);
nonidealSymbol = zeros(n,L);
nonidealScalar = zeros(M,1);
scalar = zeros(M,1);
bit = zeros(L,2);
bit2 = zeros(L, 2);
ber = zeros(1,length(SNR));

% nested for loop
% z iterating through 26 different values of SNR
% i iterating through 600000 different values of number of symbols
% j iterating through 4/8 different values of number of frequencies
for z=1:length(SNR)
   % given noise equation for random change to ideal frequency
   % leading to nonideal frequency that can be compared to ideal freq.
   % adding the noise to the loaded data(ideal)
   % reshaping the nonideal symbol into proper matrix shape that we want
   % in this case 50x600000
   noise=sqrt(Pn(z)./2).*(randn(N,1)+1i*randn(N,1));
```

```
nonideal=noise+x_psk4;
nonidealSymbol = reshape(nonideal,[n,L]);


for i = 1:L
    for j = 1:M
        % iterating through ideal and nonideal scalar values after
        % M-dot producting between ideal and nonideal data with varied
        % frequency (M-FSK)
        % dot producting each symbol with various frequencies will
        % result in scalar values of j (which is 1:M). for both
        % for 4/8 psk, dot product is abit different than before
        % because I found out that all the values in each of the
        % symbols over the number of symbols had exactly the same
        % values so in order to make more efficient program, I just
        % used the first row since all the values were the same for 50
        % samples per symbol
        scalar(j,1) = dot(symbol(1,i), y(1,j));
        nonidealScalar(j,1) = dot(nonidealSymbol(1,i), y(1,j));
        %scalar(j,1) = dot(symbol(:,i), y(:,j));
        %nonidealScalar(j,1) = dot(nonidealSymbol(:,i), y(:,j));
    end
    % since scalar will contain real and imaginary values. we will take
    % out the real part and take out max of those real part out of M
    % number of them.
    R = real(scalar');
    [~,idx] = max(R);
    % since index starts from 1 to 4. Need to substract 1 for each
    % index values and convert the index into binary and stack them
    % accordingly -- and this is for ideal
    % however due to gray scaling, I had to first convert the index
    % into gray then convert that into binary
    f_idx = idx-1;
    grayIndex = bin2gray(f_idx, 'psk', M);
    bit(i,:) = dec2bin(grayIndex, 2);

    % if bit somehow gets stored in ascii value. we need to force the
    % ascii value into decimal value.
    % according to internet. ascii value of 49 = 1; and 48 = 0
    if bit(i,1)  == 49
        bit(i,1) = 1;
    elseif bit(i,1) == 48
        bit(i,1) = 0;
    end
    if bit(i,2) == 49
        bit(i,2) = 1;
    elseif bit(i,2) == 48
        bit(i,2) = 0;
    end
    % for nonideal values the same step to get binary numbers for those
    R2 = real(nonidealScalar');
    [~,idx2] = max(R2);
    f_idx2 = idx2-1;
    grayIndex2 = bin2gray(f_idx2, 'psk', M);
    bit2(i,:) = dec2bin(grayIndex2);

    if bit2(i,1)  == 49
```

```
            bit2(i,1) = 1;
        elseif bit2(i,1) == 48
            bit2(i,1) = 0;
        end
        if bit2(i,2) == 49
            bit2(i,2) = 1;
        elseif bit2(i,2) == 48
            bit2(i,2) = 0;
        end
    end
     % using nnz function with substracting ideal and nonideal binary to
    % count up the number of error to be devided with number of total
    % sample to get BER, bit error rate.
    % this entire loop handles one iteration of SNR per run for space
    % efficiency since only BER is needed for graphing
    error_cnt=nnz(bit-bit2);
    ber(z)=error_cnt./(N);
end
 % figure for part 2 using smilogy with proper labels and grid
figure,
semilogy(SNR_dB,ber,'.-');
grid on;
xlabel('SNR[dB]');
ylabel('BER');
hold on;

semilogy(SNR_dB,SNR_ber,'.-r');
grid on;
```

```
xlabel('SNR[dB]');
ylabel('BER');
legend('Nonideal','Theoretical');

% graph for part 3
% Lfft should be higher than fft so i made it 1000 more than fft
% took out 200 symbols, ending with 10000
Lfft = 11000;
partialNumber = 200*50;
partialSym = x_psk4(1:partialNumber);

%fft = fourier transform
% using given equation and converting that into power in dB
x3 = fftshift(fft(partialSym,Lfft));
x3_dB = 10*log10(abs(x3));
% given faxis equation
faxis = -0.5*Fs + (0:(Lfft-1))*Fs/Lfft;
% plotting the graphs
figure,
% ?,1) matrix could be wrong for now
plot(faxis,x3_dB(:,1));
hold on;
% fix labels
xlabel('Frequency[Hz]');
ylabel('Spectrum[dB]');
legend('SNR = 1[dB]');
```

8 PSK

```
% for making er'thing neat and stuff. >:]
```

```
clear all;
close all;
clc
% reading in the datafile from 8PSK.mat and storing the data into a
% variable specifically for 8PSK.
% then clearing up the extra space by removing x
load('8PSK.mat', 'x');
x_psk8 = x;
clear x;
% the names of these variables are the same as ones stated in the 4/8 fsk
A = 1;
T = 1 * 10^(-6);
df = 1 * 10^6;
Fs = 50*10^6;
Ts = 1/Fs;
M = 8;
m = 0:(M-1);
Es=T;
k = log2(M);
Eb=Es/k;
SNR_dB =-10:15;
SNR = db2pow(SNR_dB);
No=Eb./SNR;
Pn=No.*Fs;
N = length(x_psk8);
n = T*Fs;
L = length(x_psk8)/n;
% reshaping the symbol to 50x600000 (symbol by number of symbols)
```

```
symbol = reshape(x_psk8,n,L);
% given frequency equation (A is 1 so ignored and so is rect function for
% the same reason being 1.
y = exp(j*m.*2*pi/M);
% theoretical SNR bit error rate
SNR_ber=(2/(log2(M)))*qfunc((sqrt((2*Es)./No)*sin(pi/M)));
% preallocation done previously before nested for loops are engaged for
% better efficiency for scalar value of ideal frequency, nonideal freq.,
% bit,bit2,ber
noise = zeros(length(x_psk8),1);
nonideal = zeros(length(x_psk8),1);
nonidealSymbol = zeros(n,L);
nonidealScalar = zeros(M,1);
scalar = zeros(M,1);
bit = zeros(L, 3);
bit2 = zeros(L, 3);
ber = zeros(1,length(SNR));

% nested for loop
% z iterating through 26 different values of SNR
% i iterating through 600000 different values of number of symbols
% j iterating through 4/8 different values of number of frequencies
for z=1:length(SNR)
    % given noise equation for random change to ideal frequency
    % leading to nonideal frequency that can be compared to ideal freq.
    % adding the noise to the loaded data(ideal)
    % reshaping the nonideal symbol into proper matrix shape that we want
    % in this case 50x600000
```

```
noise=sqrt(Pn(z)./2).*(randn(N,1)+1i*randn(N,1));
nonideal=noise+x_psk8;
nonidealSymbol = reshape(nonideal,[n,L]);
   for i = 1:L
        for j = 1:M
                % iterating through ideal and nonideal scalar values after
      % M-dot producting between ideal and nonideal data with varied
      % frequency (M-FSK)
      % dot producting each symbol with various frequencies will
      % result in scalar values of j (which is 1:M). for both
      % for 4/8 psk, dot product is abit different than before
      % because I found out that all the values in each of the
      % symbols over the number of symbols had exactly the same
      % values so in order to make more efficient program, I just
      % used the first row since all the values were the same for 50
      % samples per symbol
      scalar(j,1) = dot(symbol(1,i), y(1,j));
      nonidealScalar(j,1) = dot(nonidealSymbol(1,i), y(1,j));
   end
   % since scalar will contain real and imaginary values. we will take
   % out the real part and take out max of those real part out of M
   % number of them.
   R = real(scalar');
   [~,idx] = max(R);
   % since index starts from 1 to 4. Need to substract 1 for each
   % index values and convert the index into binary and stack them
   % accordingly -- and this is for ideal
   % however due to gray scaling, I had to first convert the index
   % into gray then convert that into binary
   f_idx = idx-1;
   grayIndex = bin2gray(f_idx, 'psk', M);
   bit(i,:) = dec2bin(grayIndex, 3);
   % if bit somehow gets stored in ascii value. we need to force the
   % ascii value into decimal value.
   % according to internet. ascii value of 49 = 1; and 48 = 0
   if bit(i,1)  == 49
      bit(i,1) = 1;
   elseif bit(i,1) == 48
      bit(i,1) = 0;
   end
   if bit(i,2) == 49
      bit(i,2) = 1;
   elseif bit(i,2) == 48
      bit(i,2) = 0;
   end
   if bit(i,3) == 49
      bit(i,3) = 1;
   elseif bit(i,3) == 48
      bit(i,3) = 0;
   end
   % for nonideal values the same step to get binary numbers for those
   R = real(scalar');
   [~,idx] = max(R);
   f_idx = idx-1;
   grayIndex = bin2gray(f_idx, 'psk', M);
   bit = dec2bin(grayIndex, 3);
```

```
        % the same thing as bit
        if bit2(i,1)  == 49
           bit2(i,1) = 1;
        elseif bit2(i,1) == 48
           bit2(i,1) = 0;
        end
        if bit2(i,2) == 49
           bit2(i,2) = 1;
        elseif bit2(i,2) == 48
           bit2(i,2) = 0;
        end

        if bit2(i,3) == 49
            bit2(i,3) = 1;
        elseif bit2(i,3) == 48
            bit2(i,3) = 0;
        end
    end
    % using nnz function with substracting ideal and nonideal binary to
    % count up the number of error to be devided with number of total
    % sample to get BER, bit error rate.
    % this entire loop handles one iteration of SNR per run for space
    % efficiency since only BER is needed for graphing
    error_cnt=nnz(bit-bit2);
    ber(z)=error_cnt./(N);
end
% figure for part 2 using smilogy with proper labels and grid
figure,
semilogy(SNR_dB,ber,'.-');

grid on;

xlabel('SNR[dB]');

ylabel('BER');

hold on;

semilogy(SNR_dB,SNR_ber,'.-r');

grid on;

xlabel('SNR[dB]');

ylabel('BER');

legend('Nonideal','Theoretical');



% graph for part 3

% Lfft should be higher than fft so i made it 1000 more than fft

% took out 200 symbols, ending with 10000

Lfft = 11000;

partialNumber = 200*50;

partialSym = x_psk8(1:partialNumber);


%fft = fourier transform

% using given equation and converting that into power in dB

x3 = fftshift(fft(partialSym,Lfft));

x3_dB = 10*log10(abs(x3));

% given faxis equation

faxis = -0.5*Fs + (0:(Lfft-1))*Fs/Lfft;

% plotting the graphs

figure,

% ☺,1) matrix could be wrong for now
```

```
plot(faxis,x3_dB(:,1));
hold on;
% fix labels
xlabel('Frequency[Hz]');
ylabel('Spectrum[dB]');
legend('SNR = 1[dB]');
```

Citation

Nathan, Goodman A. “Computer Project #1.”