EEE 431 – Telecommunications I

Project 2 Report

Oğuz Altan

21600966

Section: 01

Part 1

In this part, we implement a binary pulse amplitude modulation (PAM). First, we generate a large number of 0 and 1 bits randomly and accordingly determine transmitted signal in MATLAB. For sampling period, we pick $T/20$. After, we simulate a white Gaussian noise process and obtain the corresponding signal.

For the bits, we use triangular signals. The triangular signal for bit 0 is the inverse of the one corresponding to bit 1. The fundamental frequencies for triangles are chosen to be 1 Hz and sampling rate is chosen to be 20.

For our case, the matching filter is same as the triangular signal corresponding to bit 1. Received signal is the summation of the transmitted signal and the corresponding AWGN, which can be represented as:

$$r = s + n$$

where r represents received signal, s represents transmitted signal and n represents additive white Gaussian noise in the channel. For each bit, corresponding nosie is a additive White Gaussian Noise (AWGN) with variance $\frac{N_o}{2}$.

The AWGN is calculated using signal-to-noise ratio (SNR):

$$SNR = \frac{E_s}{N_o} = \frac{1}{N_o} \Rightarrow N_o = \frac{1}{SNR}$$

$$SNR = 10^{\frac{SNR_{dB}}{10}}$$

$$N_o = \frac{1}{10^{\frac{SNR_{dB}}{10}}}$$

For our case, SNR_{db} is chosen to be $5\ dB$.

After the normalization process, so that the energy of the signals are normalized to 1, the triangles corresponding to bits are shown:

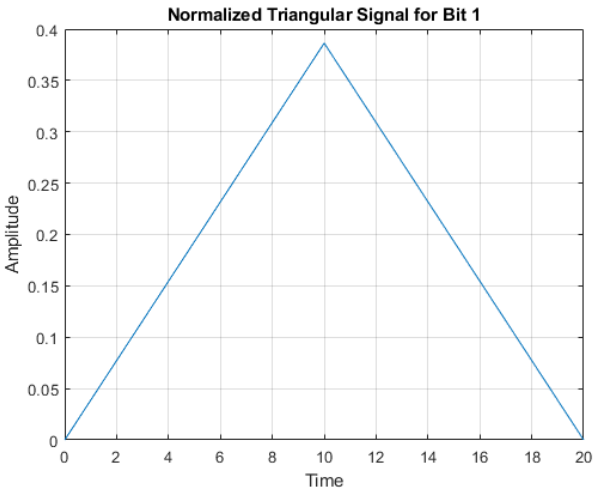

Figure 1: Triangular Signal for Bit 1

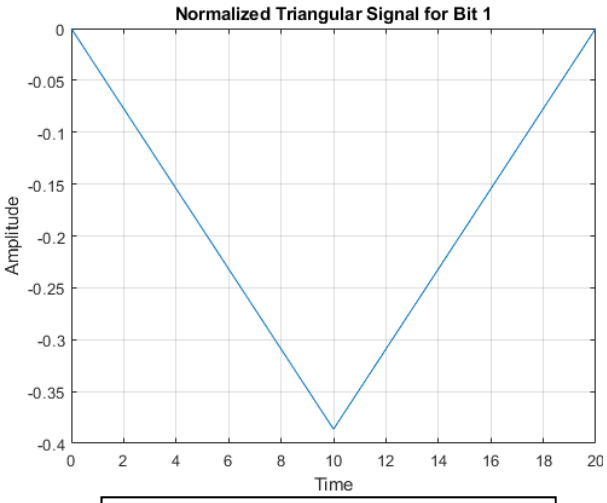

Figure 2: Triangular Signal for Bit 0

For the sake of example, for randomly generated 10 bits, the transmitted, received and filter signals are given below:

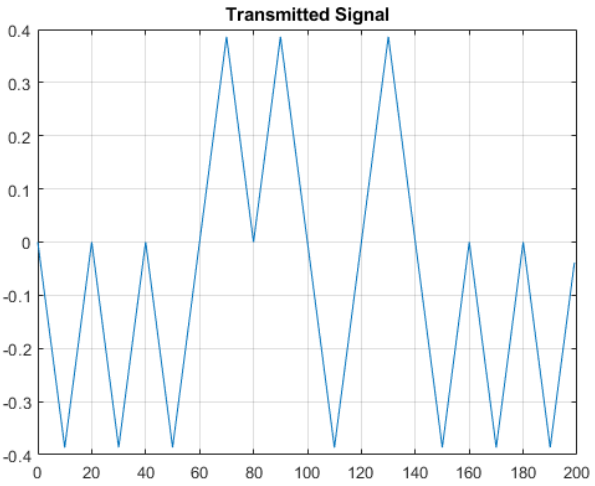

Figure 3: Transmitted Signal

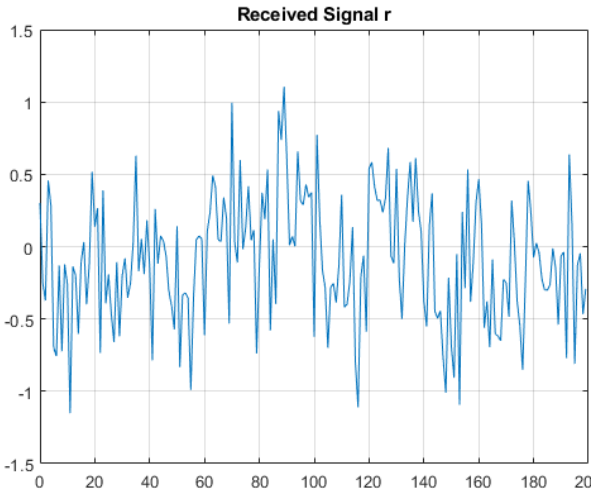

Figure 4: Received Signal

As it can be seen, the received signal at the receiver is corrupted by the AWGN. We implement a matched filter type receiver, by convolving received signal with the matched filter and estimate transmitted bits.

For 10^5 bits, we run simulations and estimate the error rate. The estimated error rate is calculated by MATLAB as 0.6515%

Comparing this result with the theoretical one, where it is shown mathematically that, using Q-function, the probability of error is:

$$Q\left(\sqrt{\frac{2}{N_o}}\right) = 0.6\%$$

It can be seen that both results are nearly same. If number of bits increase, the estimated error rate can be approximated to the thereotical results more succesfully, neverthless, our results can be accepted as successful.

Part 2

In this part, we increase the sampling rate. Increasing the sampling rate causes signal vector to get bigger by dimension and this causes the noise to increase. Therefore, we expect our erorr rate to increase. To compensate it, we adjust the variance of the noise and compute the error rate as 0.6048% which is very close to theoretical result.

Part 3

In this part, instead of using the matched filter, we use another receiver filter with a rectangular impulse response (extending from 0 to T). This new receiver filter is given below:

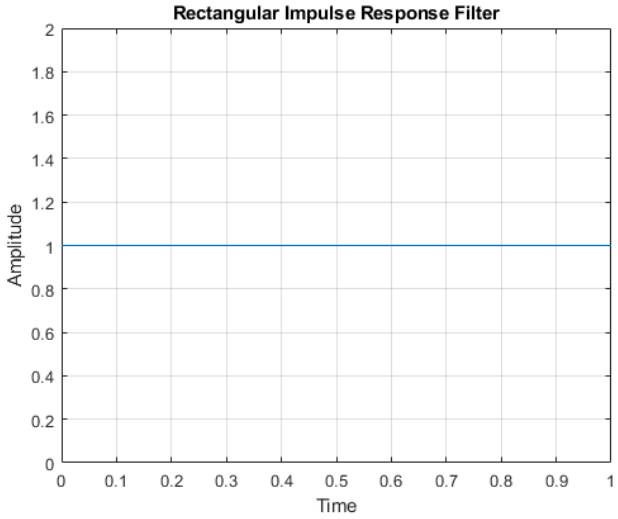

Figure 5: New Filter

Now,we convolve our received signal with the new filter and compute the error rate. The result is $1.4892\ \%$, which is bigger than what we have computed in first part. The reason is this new fillter is not ideal filter, unlike the first one, therefore error increases and estimation success decreases.

Part 4

In this part, we investigate the effects of timing error on the system performance. We have the optimal filter but sampling occurs at not perfect time instants, and it is off by some time difference ΔT. Time difference ΔT is assumed to be $\frac{T}{20}$. Assuming that transmission takes place in isolation and chosen off times are $-10\Delta T, -5\Delta T, -3\Delta T, 3\Delta T, 5\Delta T \; and \; 10\,\Delta T$, we compute corresponding errors:

Sampling Time Differences	Error Rate
$-10\Delta T$	37.1107%
$-5\Delta T$	6.1334%
$-3\Delta T$	2.1277%
$3\Delta T$	0.8763%
$5\Delta T$	2.1196%
$10\Delta T$	22.5822%

Table 1: Sampling Time Differences and Corresponding Error Rates for Isolated Transmission

Observing the table, it can be seen that as sampling time gets further from T, the error increases as T is the perfect point for sampling time. Considering our triangular signal shape, getting fruther from T means to move away from the peak point of triangle, thus the amplitude of that point decreases. Therefore, error increases.

Part 5

In this part, unlike the previous part, we assume that there are preceding and succeeding transmissions of other bits, which causes reduction in the signal part and also we observe effects of the other consecutive symbol transmissions. Therefore, we expect the error rates to increase. Computing error rates, we get:

Sampling Time Differences	Error Rate
$-10\Delta T$	39.34%
$-5\Delta T$	6.8%
$-3\Delta T$	2.21%
$3\Delta T$	0.83%
$5\Delta T$	2.19%
$10\Delta T$	25.5%

Table 2: Sampling Time Differences and Corresponding Error Rates for Not-Isolated Transmission

As it can be seen, the error rates are increased, compared to the ones in previous part.

Part 6

In this part, we simulate binary orthogonal frequency shift keying (FSK) with coherent detection. For this purpose, we use two orthogonal basis functions, which are:

$$\varphi_1 = \left(\sqrt{\frac{2}{T_s}}\right)\cos(2\pi f_1 t + \phi_1)$$

$$\varphi_2 = \left(\sqrt{\frac{2}{T_s}}\right)\cos(2\pi f_2 t + \phi_2)$$

Our signal are not triangle anymore, they are shown as:

$$s = \left(\sqrt{\frac{2}{T_s}}\right)\cos(2\pi f_i t) \qquad i = 1,2$$

For bit 1, we use s with $i = 1$ and for bit 0, we use s with $i = 0$.

Our received signal is:

$$r = s + n$$

In the receiver, we convolve our receiver filter with basis functions one by one. Our decision rule is if the convolution of received signal and φ_1 is bigger than the on with φ_2, we decide on bit 1 and vice-versa for bit 0. An example sinusoidal signal for bit 1 with Ts = 20 and corresponding example received signal are given below:

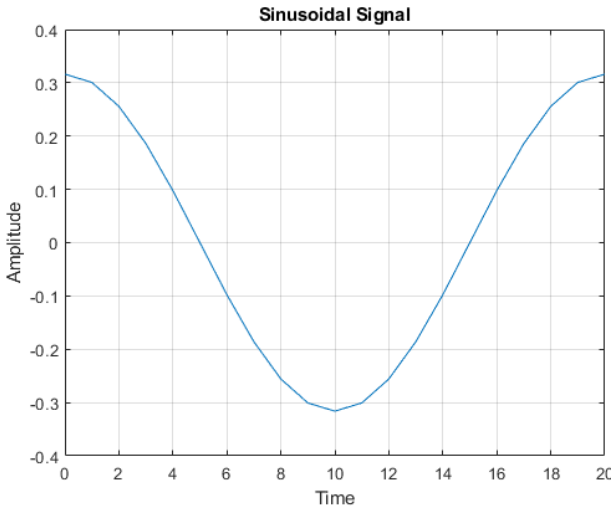

Figure 6: Sinusoidal Signal for Bit 1

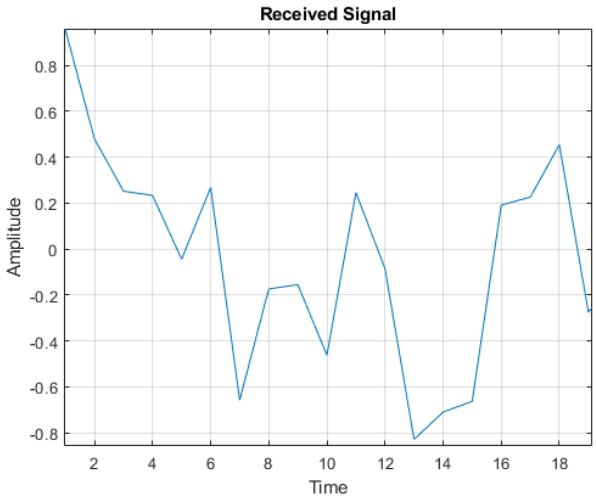

Figure 7: Received Signal for Bit 1

The error rate is computed as 3.83%. Comparing this result with the theoretical one, where it is shown mathematically that, using Q-function, the probability of error is:

$$Q\left(\sqrt{\frac{1}{N_o}}\right) = 3.77\%$$

It can be seen that both results are nearly same. If number of bits increase, the estimated error rate can be approximated to the thereotical results more succesfully, neverthless, our results can be accepted as successful.

Part 7

For the non-coherent detection case, for binary FSK, we need two other orthonormal basis functions. The full list of orthonormal basis functions are as following:

$$\varphi_{11} = \left(\sqrt{\frac{2}{T_s}}\right)\cos(2\pi f_1 t)$$

$$\varphi_{12} = \left(\sqrt{\frac{2}{T_s}}\right)\sin(2\pi f_1 t)$$

$$\varphi_{21} = \left(\sqrt{\frac{2}{T_s}}\right)\cos(2\pi f_2 t)$$

$$\varphi_{22} = \left(\sqrt{\frac{2}{T_s}}\right)\sin(2\pi f_2 t)$$

In this case, we compute the convolution of received signal and basis functions one by one. We compare the sum of squares of these convolution operations.

$$c_{11} = conv(r, \varphi_{11}) \qquad c_{12} = conv(r, \varphi_{12})$$

$$c_{21} = conv(r, \varphi_{21}) \qquad c_{12} = conv(r, \varphi_{22})$$

If $c_{11}^2 + c_{12}^2 > c_{21}^2 + c_{22}^2$ then we decide on signal 1, and signal 0 for opposite case.

The error rate is computed as 10.28%. Comparing this result with the theoretical one, where it is shown mathematically that the probability of error is:

$$\frac{1}{2}e^{-\frac{1}{2N_o}} = 10.21$$

It can be seen that both results are nearly same. Therefore, this process can be accepted as successful.

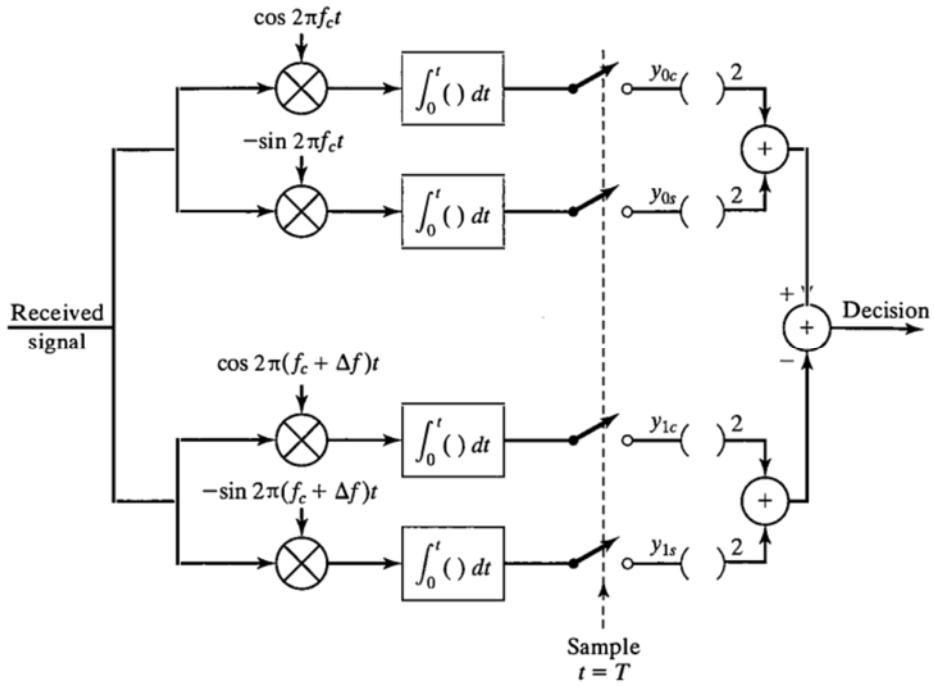

Figure 8: Non-Coherent Detector for Binary FSK [1]

References

[1] John G. Proakis and Masoud Salehi. Fundamentals of Communications Systems. Pearson, 2014.

Appendix

MATLAB Code

```
clear all; close all;

%random bit generate
num_bits = 1e4;

rand_bit = round(rand(1,num_bits));
signal = [];

for i = 1:num_bits
    if (rand_bit(i) == 1)
        signal = [signal triangleSignal(20)];
    else
        signal = [signal (-1)*triangleSignal(20)];
    end
end

figure;
plot(0:length(signal)-1,signal);
title("Transmitted Signal");
grid on;

% Generate Noise
% Choose an SNR value of 5 dB
SNR_db = 5;
SNR = 10^(SNR_db/10);
No = 1/SNR;
n = sqrt(No/2).* randn(1,length(signal));

r = signal + n;

figure;
plot(0:length(r)-1,r);
title("Received Signal r");
grid on;

filter = triangleSignal(20);

figure;
plot(0:length(filter)-1,filter);
title("Matched Filter");
grid on;

%Part 1
disp("Part 1");

error_vec1  = zeros(1,100);
for i = 1:100
    n = sqrt(No/2).* randn(1,length(signal));
    r = signal + n;
    conv_signal = conv(r,filter);

    for j = 1:length(rand_bit)
        if (conv_signal(20*j)>0)
            check_bits(j) = 1;
        else
            check_bits(j) = 0;
        end
    end

    error = sum(abs(check_bits-rand_bit))/length(rand_bit)*100;
    error_vec(i) = error;
end

error_av = mean(error_vec);
disp(['Average Error Rate for Sampling Rate 20: ', num2str(error_av), '%']);
```

```
%Part 2
disp("Part 2");

signal2 = [];

for i = 1:num_bits
    if (rand_bit(i) == 1)
        signal2 = [signal2 triangleSignal(50)];
    else
        signal2 = [signal2 (-1)*triangleSignal(50)];
    end
end

filter2 = triangleSignal(50);

error_vec2 = zeros(1,100);
for i = 1:100
    n2 = sqrt(No/2).* randn(1,length(signal2));
    r2 = signal2 + n2;
    conv_signal2 = conv(r2,filter2);

    for j = 1:length(rand_bit)
        if (conv_signal2(50*j)>0)
            check_bits2(j) = 1;
        else
            check_bits2(j) = 0;
        end
    end

    error2 = sum(abs(check_bits2-rand_bit))/length(rand_bit)*100;
    error_vec2(i) = error2;
end

error_av2 = mean(error_vec2);
disp(['Average Error Rate for Sampling Rate 50: ', num2str(error_av2), '%']);

%Part 3
disp("Part 3");

t3 = 0:1/20:1-1/20;
filter3 = ones(1,length(t3));
% plot(t3-1,filter3);

error_vec3  = zeros(1,100);

for i = 1:100
    n = sqrt(No/2).* randn(1,length(signal));
    r = signal + n;
    conv_signal3 = conv(r,filter3);

    for j = 1:length(rand_bit)
        if (conv_signal3(20*j)>0)
            check_bits3(j) = 1;
        else
            check_bits3(j) = 0;
        end
    end

    error3 = sum(abs(check_bits3-rand_bit))/length(rand_bit)*100;
    error_vec3(i) = error3;
end

error_av3 = mean(error_vec3);
disp(['Average Error Rate for Not Ideal Filter: ', num2str(error_av3), '%']);

%Part 4
disp("Part 4");
```

```
deltaT_vec = [-10 -5 -3 3 5 10];
error_av = zeros(1,6);

for ts = 1:length(deltaT_vec)

    for i = 1:100
        n = sqrt(No/2).* randn(1,length(signal));
        r = signal + n;
        conv_signal = conv(r,filter);

        for j = 1:length(rand_bit)
            if (conv_signal(20*j + deltaT_vec(ts)) > 0)
                check_bits(j) = 1;
            else
                check_bits(j) = 0;
            end
        end

        error = sum(abs(check_bits-rand_bit))/length(rand_bit)*100;
        error_vec(i) = error;
    end

    error_av(ts) = mean(error_vec);
end

disp(['Error for T-10deltaT ', num2str(error_av(1)), '%'])
disp(['Error for T-5deltaT ', num2str(error_av(2)), '%'])
disp(['Error for T-3deltaT ', num2str(error_av(3)), '%'])
disp(['Error for T+3deltaT ', num2str(error_av(4)), '%'])
disp(['Error for T+5deltaT ', num2str(error_av(5)), '%'])
disp(['Error for T+10deltaT ', num2str(error_av(6)), '%'])

%Part 5
disp("Part 5");

T = 20;
deltaT_vec = [-10 -5 -3 3 5 10];
error_q5 = zeros(1,6);

for i = 1:length(deltaT_vec)
    check_bits_q5 = zeros(1,num_bits);

    %for previous message
    if (deltaT_vec(i) < 0)
        first = T + 1;
        last = 2*T;
        prev_and_current = [zeros(1,T) r];

        for j = 1:num_bits
            conv_prev = conv(prev_and_current(first:last),filter);
            prev_signal = prev_and_current(first-T:last-T);
            temp = conv(prev_signal,filter);
            temp = temp(T:length(temp));
            conv_prev(1:T) = conv_prev(1:T) + temp;

            if conv_prev(deltaT_vec(i) + T) > 0
                check_bits_q5(j) = 1;
            else
                check_bits_q5(j) = 0;
            end
            first = first + T;
            last = last + T;
        end

        %for next message
    else
        first = 1;
        last = T;
        current_and_next = [r zeros(1,T)];
```

```
        for j = 1:num_bits
            conv_next = conv(current_and_next(first:last),filter);
            next_signal = current_and_next(first+T:last+T);
            temp = conv(next_signal,filter);
            temp = temp(1:T);
            conv_next(T:length(conv_next)) = conv_next(T:length(conv_next)) + temp;

            if conv_next(deltaT_vec(i) + T) > 0
                check_bits_q5(j) = 1;
            else
                check_bits_q5(j) = 0;
            end
            first = first + T;
            last = last + T;
        end
    end

    error_q5(i) = sum(abs(check_bits_q5-rand_bit))/length(rand_bit)*100;
end

disp(['Error for T-10deltaT ', num2str(error_q5(1)), '%'])
disp(['Error for T-5deltaT ', num2str(error_q5(2)), '%'])
disp(['Error for T-3deltaT ', num2str(error_q5(3)), '%'])
disp(['Error for T+3deltaT ', num2str(error_q5(4)), '%'])
disp(['Error for T+5deltaT ', num2str(error_q5(5)), '%'])
disp(['Error for T+10deltaT ', num2str(error_q5(6)), '%'])

%Part 6
disp("Part 6");

signal_fsk = [];
check_bits_fsk = zeros(1,length(rand_bit));

Ts = 20;
f1 = 1;
f2 = 4;
t_fsk = 0:1/Ts:1-1/Ts;

for i = 1:num_bits
    if rand_bit(i) == 1
        signal_fsk = [signal_fsk sqrt(2/Ts)*cos(2*pi*f1*t_fsk)];
    else
        signal_fsk = [signal_fsk sqrt(2/Ts)*cos(2*pi*f2*t_fsk)];
    end
end

figure;
one_signal_fsk = [signal_fsk(1:Ts) 0.316227766016838];
plot(0:length(one_signal_fsk)-1,one_signal_fsk);
grid on;
title("Sinusoidal Signal");
xlabel("Time");
ylabel("Amplitude");

n_fsk = sqrt(No/2).* randn(1,length(signal_fsk));
r_fsk = signal_fsk + n_fsk;

figure;
one_signal_fsk_r = r_fsk(1:Ts);
plot(1:length(one_signal_fsk_r),one_signal_fsk_r);
grid on;
title("Received Signal");
xlabel("Time");
ylabel("Amplitude");

basis1_fsk = sqrt(2/Ts)*cos(2*pi*f1*t_fsk);
basis2_fsk = sqrt(2/Ts)*cos(2*pi*f2*t_fsk);

for i = 1:length(rand_bit)
```

```
    c1 = sum((r_fsk((i-1)*Ts+1:i*Ts).*basis1_fsk));
    c2 = sum((r_fsk((i-1)*Ts+1:i*Ts).*basis2_fsk));

    if c1 >= c2
        check_bits_fsk(i) = 1;
    else
        check_bits_fsk(i) = 0;
    end
end

error_fsk = 100*sum(abs(check_bits_fsk-rand_bit))/length(rand_bit);
disp(['Estimated Error Rate for FSK: ', num2str(error_fsk), '%']);

%Part 7
disp("Part 7");

basis11_nc_fsk = sqrt(2/Ts)*cos(2*pi*f1*t_fsk);
basis12_nc_fsk = sqrt(2/Ts)*sin(2*pi*f1*t_fsk);

basis21_nc_fsk = sqrt(2/Ts)*cos(2*pi*f2*t_fsk);
basis22_nc_fsk = sqrt(2/Ts)*sin(2*pi*f2*t_fsk);

check_bits_nc_fsk = zeros(1,length(rand_bit));

for i = 1:length(rand_bit)

    c11 = sum(r_fsk((i-1)*Ts+1:i*Ts).*basis11_nc_fsk);
    c12 = sum(r_fsk((i-1)*Ts+1:i*Ts).*basis12_nc_fsk);
    c21 = sum(r_fsk((i-1)*Ts+1:i*Ts).*basis21_nc_fsk);
    c22 = sum(r_fsk((i-1)*Ts+1:i*Ts).*basis22_nc_fsk);

    if (c11^2 + c12^2) > (c21^2 + c22^2)
        check_bits_nc_fsk(i) = 1;
    else
        check_bits_nc_fsk(i) = 0;
    end
end

error_nc_fsk = 100*sum(abs(check_bits_nc_fsk-rand_bit))/length(rand_bit);
disp(['Estimated Error Rate for Non-Coherent FSK: ', num2str(error_nc_fsk), '%']);

%generate triangular signal
function signal = triangleSignal(sample_rate)

f = 1; %fundamental freq
T = 1/f;
Ts = T/sample_rate; %sampling period
fs = 1/Ts; %sample rate

t = 0:1/fs:T - 1/fs; %calculate t
signal = (1+sawtooth(2*pi*f*t,1/2))/2; %constructing triangle using sawtooth

%normalization
E = sum(signal.*signal); %energy of the signal
signal = signal/sqrt(E);

end
```