Faculty of Engineering - Ain Shams University

جامعة عين شمس

1839

كلية الهندسة - جامعة عين شمس

Fundamental of Communication Systems

ECE 252

Analog & Digital Communication Project

Submitted to:

Dr. Mazen Erfan

Dr. Alaa Fathy

Eng. Ahmed Al-Sayed

Eng. Ahmed Khaled

Date of Submission:

June 5th

Name	ID
Amr Ahmed Wahidi	2200429
Ammar Ahmed Wahidi	2200360
Abdelrahman Essam Fahmy	2200251
Mohamed Yehia Saeed	2200918
Sobhey Mohamed Osman	2200477

Contents

Project Overview:

Our project is divided into two main parts (analog part & digital part). In both, we try to apply the concepts that we have learned from our professors to deal with signals either with analog tools or digital tools. Some of the concepts that we have used is the modulation and demodulation processes and different kinds of modulation, as the modulation process helps us to avoid noise & distortion of the signals, and it reduces the length of the antenna used in devices. Now, we will dig into the tasks we performed.

Analog Communication

Overview:

The Analog Part of this project focuses on the analysis, modulation, and transmission of analog signals using fundamental communication techniques. The tasks involve signal generation, Fourier analysis, bandwidth estimation, filtering, and modulation schemes such as DSB-SC (Double-Sideband Suppressed Carrier) and SSB (Single Sideband). Additionally, we explore Frequency Division Multiplexing (FDM) to transmit multiple signals simultaneously over a shared channel.

Key Objectives:

1. Signal Representation & Analysis
    - Time-domain plotting of piecewise and sinusoidal signals.
    - Derivation of analytical Fourier transforms.
    - Numerical computation of Fourier transforms using FFT and comparison with analytical results.
    - Bandwidth estimation based on power spectral density.
2. Filtering & Reconstruction
    - Application of ideal Low-Pass Filters (LPF) with varying bandwidths (1 Hz and 0.3 Hz).
    - Comparison of filtered outputs with original signals in the time domain.

3. Modulation Techniques
   - DSB-SC Modulation: Used for transmitting x(t) with a carrier frequency of 20 Hz.
   - SSB Modulation (USB): Applied to m(t) to conserve bandwidth, with a carrier frequency of 23 Hz.
   - FDM Implementation: Combines DSB-SC and SSB signals with a 2 Hz guard band.
4. Demodulation & Recovery
   - Coherent detection for both DSB-SC and SSB signals.
   - Time-domain comparison of original and demodulated signals to validate fidelity.

Define Terms

```
df= 0.01;

fs = 100;

ts = 1/fs;

T = 1/df;

N = round(T*fs);

t = -T/2: ts: T/2-ts;

f = (-fs/2):df:(fs/2-df);
```

1. Plot the function x(t) shown on Octave

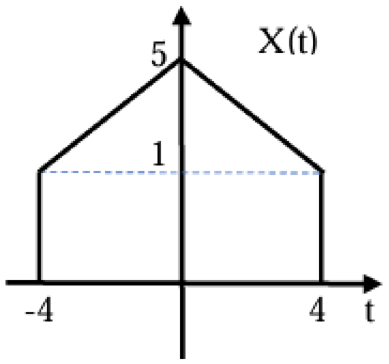
To plot this function, we need to define it first, as shown

In Figure 1.

X(t) = t+5

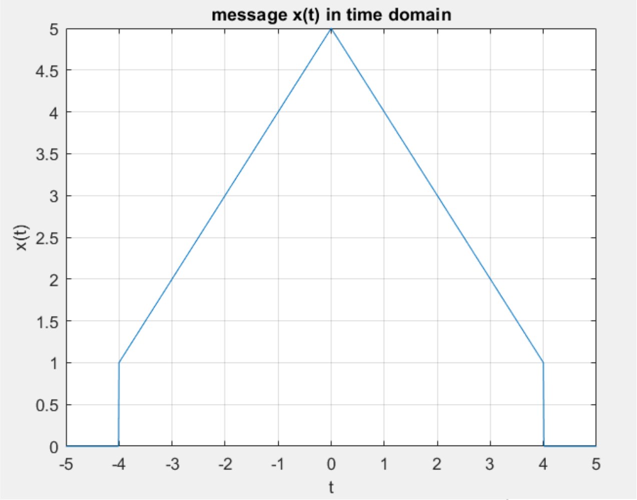
for -4<=t<0 & X(t) = -t+5

for 0<=t<=4



X(t)

```
x = zeros(size(t));

x(t>=-4 & t<0) = t(t>=-4 & t<0) + 5;

x(t>=0 & t<=4) = -t(t>=0 & t<=4) + 5;

figure(1);

plot (t,x);

xlabel('t');
ylabel('x(t)');

title('message x(t) in time domain');
xlim([-5 5]);

grid on;
```



2. Derive an analytical expression for its Fourier transform

Recall the differentiation property By applying it, we can find that

$$x(t) \gg\gg\gg\gg\gg\gg\gg\gg\gg > x(f)$$

$$\frac{dx(t)}{dt} \gg\gg\gg\gg\gg\gg\gg\gg\gg > j(2\pi f) * x(f)$$

$$j(2\pi f) * x(f) \ =$$

$$e^{j(2\pi f)(4)} \ - \ e^{-j(2\pi f)(4)} + 4 * sinc(4f) * e^{j(2\pi f)(2)} \ - \ 4 * sinc(4f) * \ e^{-j(2\pi f)(2)}$$

Where $sin(\theta) \ = \frac{e^{j(\theta)} - e^{-j(\theta)}}{2j}$

$$j(2\pi f) * x(f) \ = \ 2 * j * sin(8\pi f) + 4 * sinc(4f) * 2j * sin(4\pi f)$$

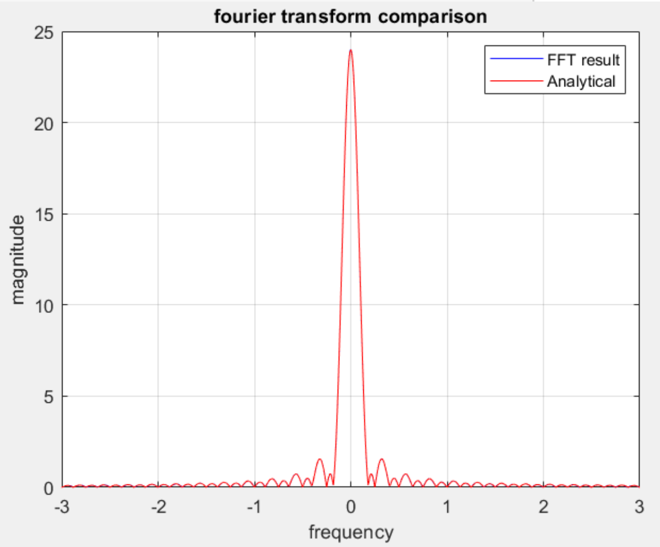
by dividing both sides $j(2\pi f)$ *and making some substituions.*

$$x(f) \ = \ 8sinc(8f) + 16sinc^2(4f)$$

```
G = 8*sinc(8*f)+16*(sinc(4*f).^2); %analytical
```

3. FFT of the signal

```
XF = fftshift(fft(x))*ts;
XF_mag = abs(XF);          % function is non-periodic so multiply with ts
figure;
plot(f,XF_mag, 'b');
hold on;
plot (f,abs(G), 'r');
xlabel('frequency');


ylabel('magnitude')
title('fourier transform comparison');

legend('FFT result','Analytical');

xlim([-3 3]);
grid on;
```



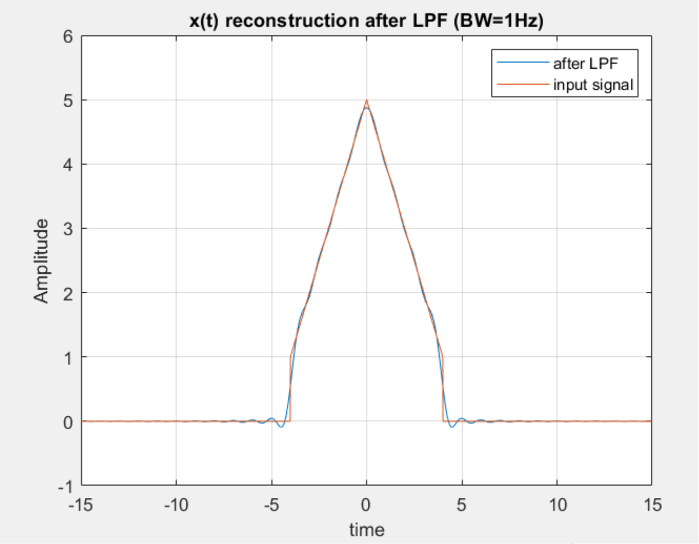
4. Estimate the BW defined as the frequency band after which the power spectrum of the signal drops to 5% of its maximum value.

```
power_m = abs(XF.^2);
P_mag_max = max(power_m);
threshold = 0.05 * P_mag_max;
index1 = find(power_m >= threshold);
BW1_estimate = max(abs(f(index1)))
```

```
BW1_estimate =

    0.1300
```

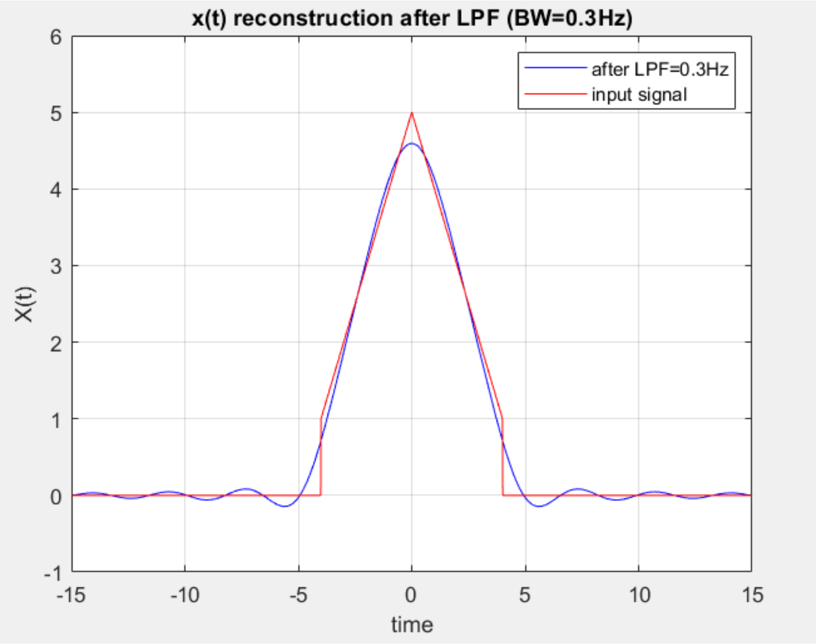
5. LPF Filtering with BW=1Hz

```
BW = 1;

H_lpf1 = abs(f) < BW; % Renamed H to avoid conflict
x_rec = real(ifft(ifftshift(H_lpf1.*XF)/ts));
figure;
plot(t,x_rec);

hold on;
plot(t,x);

legend('after LPF','input signal');

title('x(t) reconstruction after LPF (BW=1Hz)');


xlabel('time');

ylabel('Amplitude');

xlim([-15 15]);

ylim([-1 6]);

grid on;
```



6. LPF Filtering with BW=0.3Hz

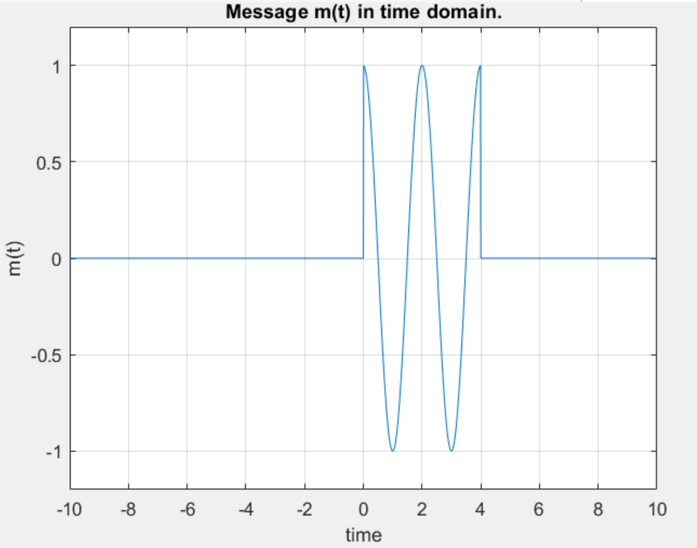
```
BW_03 = 0.3; % Renamed BW to avoid conflict
H_lpf2 = abs(f) < BW_03; % Renamed H to avoid conflict
x_rec2 = real(ifft(ifftshift(H_lpf2.*XF)/ts));
figure;
plot(t,x_rec2,'b');
hold on;
plot(t,x,'r');
legend('after LPF=0.3Hz', 'input signal');
title('x(t) reconstruction after LPF (BW=0.3Hz)');
xlabel('time');
ylabel('X(t)');
xlim([-15 15]);
ylim([-1 6]);
grid on;
```



7. Analysis of m(t) = cos(2π·0.5·t) for 0 < t < 4

Step 1

```
m = zeros(size(t));

m(t>0 & t<4) = cos(2*pi*0.5*t(t>0 & t<4));

figure;

plot(t,m);

title('Message m(t) in time domain.');

xlabel('time');

ylabel('m(t)');

xlim([-10 10]);

ylim([-1.2 1.2]);

grid on;
```



Step 2

Recall that: $\cos(\theta) = \frac{e^{J\theta} + e^{-J\theta}}{2}$, We see that our function is considered as a cosine function multiplied with a rectangle with width = 4

So we can rewrite it as :   m(t) = cos(2π*0.5*t) * rectangle with width = 4

Remember: Multiplication in any domain is converted to convolution in the another domain.

cos(2π*0.5*t) = $\frac{e^{J(2\pi*0.5*t)}+e^{-J(2\pi*0.5*t)}}{2}$ >>   0.5*(δ(f-0.5) + δ(f+0.5))

rectangle with width = 4   >>   4*$e^{-J(2\pi f*2)}$*sinc(4f)

Remember: Convolution with an impulse means shifting the signal's center to the impulse.

By applying the previous concepts, we can deduce that:

M(f) = 2*[$e^{-J(2\pi(f-0.5)*2)}$*sinc(4(f-0.5)) +$e^{-J(2\pi(f+0.5)*2)}$*sinc(4(f+0.5))]

Remark: the exponential term in the previous equation doesn't affect the amplitude of the spectrum, it just represents a phase shift (due to the time shift of the pulse). For this reason, we have ignored this term, so the equation can be written as :

M(f) = 2*[sinc(4(f-0.5)) +sinc(4(f+0.5))]

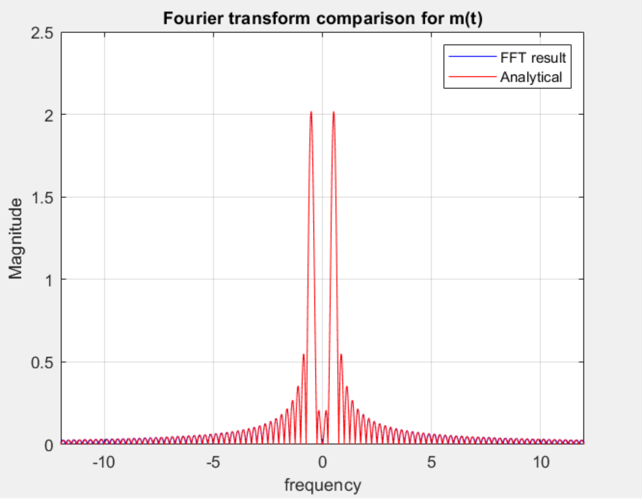
```
GF= 2*sinc(4*(f-0.5))+ 2*sinc(4*(f + 0.5));
```

Step 3

Use Octave to calculate the Fourier transform of the signal with sampling frequency fs =100 Hz and resolution equal to 0.01 Hz, and then plot it together with the analytical expression on one graph

```
M = fftshift(fft(m))*ts;
figure;
plot(f,abs(M),'b');
hold on;
```

```
plot(f,abs(GF),'r');
title('Fourier transform comparison for m(t)');
legend('FFT result','Analytical');
xlabel('frequency');
ylabel('Magnitude');
xlim([-12 12]);
grid on;
```



Step 4

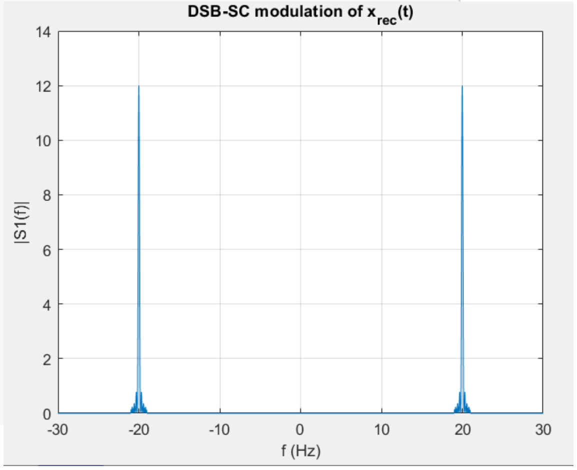
```
power_m = abs(M.^2);
P_mag_max = max(power_m);
threshold = 0.05 * P_mag_max;
index2 = find(power_m >= threshold);
BW2_estimate = max(abs(f(index2)))
```

```
BW2_estimate =

    0.9100
```

8. FDM Modulation Scheme

Modulation of message x(t) using DSB-SC modulation

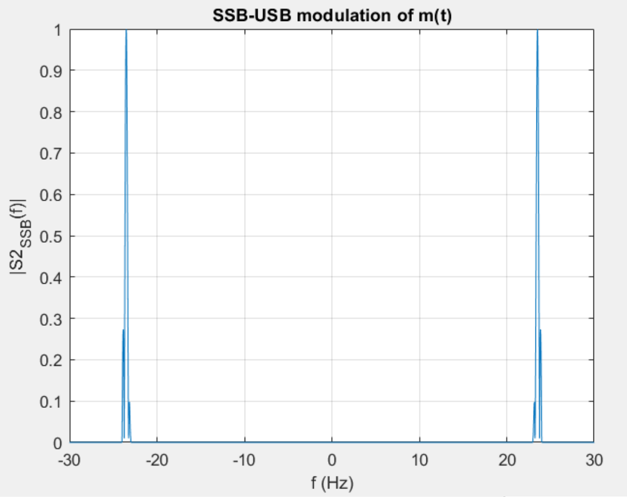
```
fc1 = 20;

carrier1 = cos(2*pi*fc1*t);

s1= x_rec.*carrier1;
S1 = fftshift(fft(s1))*ts;
figure;

plot(f,abs(S1));

xlabel('f (Hz)');

ylabel('|S1(f)|');
```



```
title('DSB-SC modulation of x_{rec}(t)');
xlim([-30 30]);


grid on;
```

Modulation of message m(t) using SSB modulation

```
fc2 = 23;               %USB is used
carrier2 = cos(2*pi*fc2*t);
s2 = m.*carrier2;
S2 = fftshift(fft(s2))*ts;
H_ssb = zeros(size(f)); % Renamed H to avoid conflict

% For USB, passband is [fc2, fc2 + BW_of_m] and [-fc2 - BW_of_m, -fc2]
% Assuming BW_of_m is around 1 Hz for m(t) = cos(pi*t) for 0<t<4

BW_m_approx = 1; % Approximate bandwidth of m(t)
H_ssb(f>=fc2 & f<=(fc2+BW_m_approx))=1; % Positive frequencies for USB
H_ssb(f<=-fc2 & f>=(-fc2-BW_m_approx))=1; % Negative frequencies for USB

S2SSB = S2 .*H_ssb;
s2ssb = real(ifft(ifftshift(S2SSB)/ts));

figure;
plot(f,abs(S2SSB));


title('SSB-USB modulation of m(t)');

xlabel('f (Hz)');

ylabel('|S2_{SSB}(f)|');

xlim([-30 30]);


grid on;
```



9. State whether you will use USB or LSB.

As studied in lectures and tutorials, we know that both USB & LSB have the same information and contain the same data, so there is no difference between sending any one of them.

In our code, we have used the USB.

10. An appropriate value for $c_2(t)$.

Given that BW of the channel = 2 Hz & guard band = 2 Hz & USB is used.

For DSB-SC, the carrier frequency is at the center of the channel.

For SSB, the carrier frequency is at the edge of the channel.

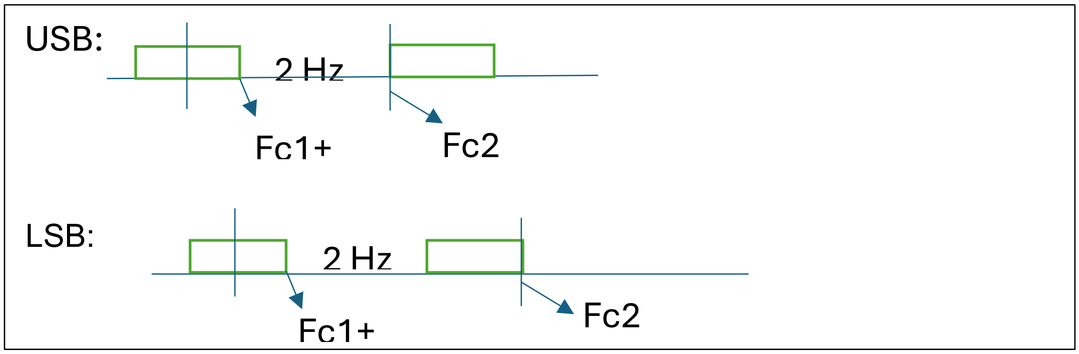
To get an appropriate value of Fc_2, we need to ensure a 2 Hz guard between Fc_2 and the highest frequency of channel one (Fc_1+1).

$Fc_2 = Fc_1+1+2$ where $Fc_1 =20$ >>>>>>>> $Fc_2=23$ Hz

Additionally(in case the LSB is used):

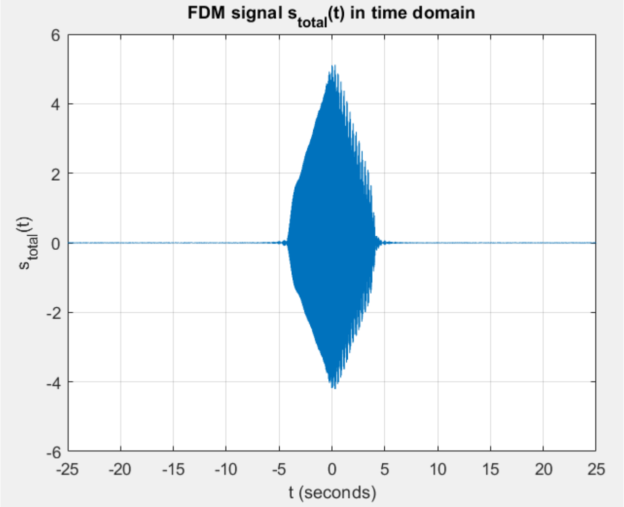
To get an appropriate value of Fc_2, we need to ensure a 2 Hz guard and a 2 Hz BW of the channel between Fc_2 and the highest frequency of channel one (Fc_1+1).

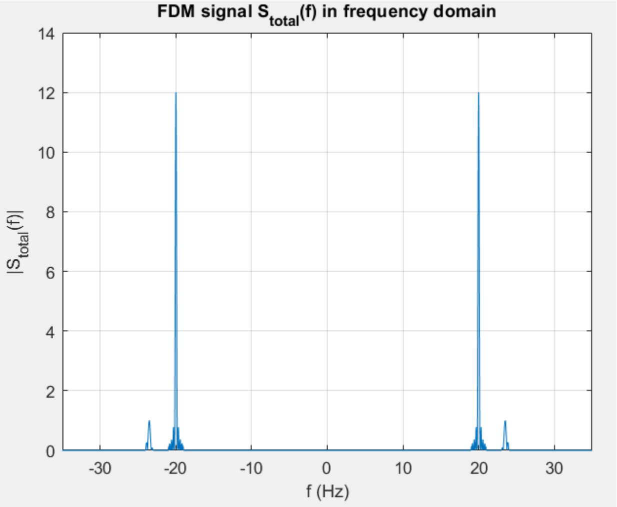
$Fc_2 = Fc_1+1+2+2$ where $Fc_1 =20$ >>>>>>>> $\boxed{Fc_2=25 \text{ Hz}}$



11. FDM Signal Visualization

```
stotal = s1 + s2ssb; % FDM: s1 (DSB-SC) + s2ssb (SSB)
figure;
plot(t,stotal);
xlabel('t (seconds)');
ylabel('s_{total}(t)');
title('FDM signal s_{total}(t) in time domain');
xlim([-25 25]);
grid on;


Stotal = fftshift(fft(stotal))*ts;


figure;


plot(f,abs(Stotal));    %two spectrums on the same graph
xlabel('f (Hz)');
ylabel('|S_{total}(f)|');
title('FDM signal S_{total}(f) in frequency domain');
xlim([-35 35]);
grid on;
```

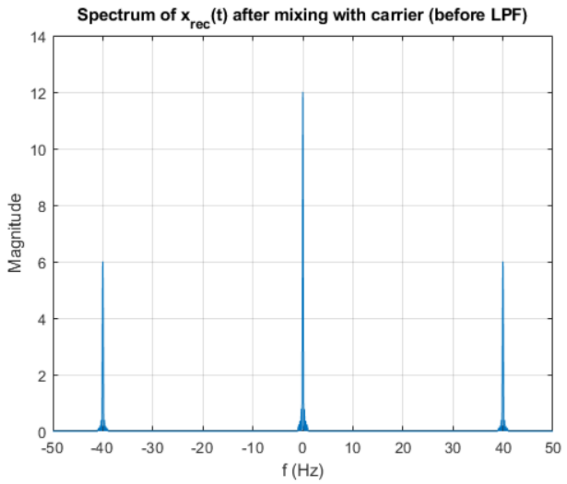


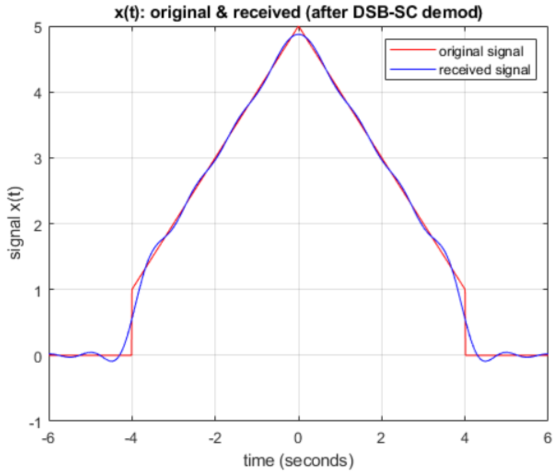


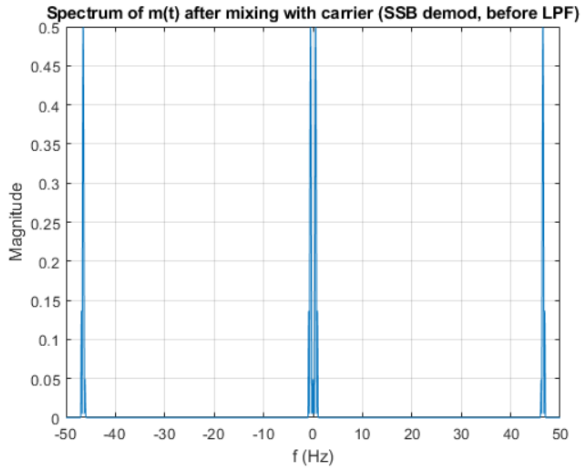
12. Coherent Detector

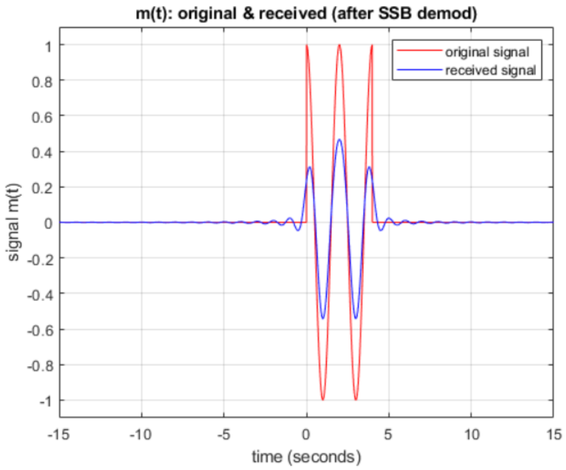
```
x_before_LPF = s1 .*carrier1; % s1 already is x_rec . carrier1
X_before_LPF = fftshift(fft(x_before_LPF))*ts;
figure;
plot(f,abs(X_before_LPF));
title('Spectrum of x_{rec}(t) after mixing with carrier (before LPF)');
xlabel('f (Hz)');
ylabel('Magnitude');
grid on;

H_demod_x = abs(f)<BW; % Using BW=1 for LPF for x(t) (consistent with x_rec)
x_received = 2*real(ifft(ifftshift(H_demod_x.*X_before_LPF)/ts)); % Multiply by 2 for
DSB-SC demod
figure;
plot(t,x,'r'); % Plot original x for comparison
hold on;
plot(t,x_received,'b');
xlabel('time (seconds)');
ylabel('signal x(t)');
title('x(t): original & received (after DSB-SC demod)');
legend('original signal','received signal');
xlim([-6 6]);
grid on;

% for message m(t)
% For SSB, coherent detection involves multiplying by carrier and LPF
m_before_LPF = s2ssb .*carrier2;
M_before_LPF = fftshift(fft(m_before_LPF))*ts;
figure;
plot(f,abs(M_before_LPF));
title('Spectrum of m(t) after mixing with carrier (SSB demod, before LPF)');
xlabel('f (Hz)');
ylabel('Magnitude');
grid on;

% Bandwidth for LPF for m(t) should be related to BW_m_approx
H_demod_m = abs(f)<BW_m_approx;
m_received = 2*real(ifft(ifftshift(H_demod_m.*M_before_LPF)/ts)); % Multiply by 2 for
SSB demod with cos
figure;
plot(t,m,'r');
```

```
hold on;
plot(t,m_received,'b');
xlabel('time (seconds)');
ylabel('signal m(t)');
title('m(t): original & received (after SSB demod)');
legend('original signal','received signal');
xlim([-15 15]);
ylim([-1.1 1.1]);
grid on;
```









Digital Communication

Part 1: Line Coding

Overview:

In this task, we were required to simulate and analyze two different line coding techniques using Octave. The main goal was to understand the differences between these coding schemes both in the time domain and the frequency domain. Each group was instructed to select one coding scheme from the set {AMI, CMI, Manchester} and compare it with another from the set {Unipolar NRZ, Polar NRZ.

To fulfill this task, a random bit stream of at least 64 bits was generated using Octave. A custom script was developed to apply the selected line coding techniques to the bit stream. The coded signals were then plotted in the time domain to visualize pulse shapes, and the Power Spectral Density (PSD) was calculated using the Fast Fourier Transform (FFT) to analyze the frequency components.

In our implementation, we selected Manchester coding and Unipolar NRZ for comparison. A comprehensive analysis was conducted to highlight the differences in bandwidth, signal synchronization, and DC component behavior between the two methods.

Initialization and Time Vector Generation

To begin the simulation, we initialized key parameters related to the digital transmission system. These include the bit rate, sampling rate, bit duration, and time vector needed to generate and represent the digital signal accurately in the time domain. Below is a detailed breakdown of the initialization process:

```
stream=randn(1,64);

bit_rate=1e4;

tb=1/bit_rate;

samples_per_rate=1000;

num_bit=length(stream);
```

```
total_samples=num_bit * samples_per_rate;

T = num_bit / bit_rate;

Fs = samples_per_rate * bit_rate;

ts = 1 / Fs;

df=Fs/total_samples;

N = ceil(T / ts);

t = linspace(0, num_bit / bit_rate, total_samples);

if mod(N,2)==0
    f=-(Fs/2):df:(Fs/2)-df;
else
    f=-(0.5*Fs -0.5*df):df:(Fs/2)-0.5*df;
end
```

stream: Generates a random array of 64 numbers (normally distributed). This stream will be thresholder later into binary bits (0 or 1).

bit_rate: Defines how many bits are transmitted per second, which is 10 kbps in this case.

tb: Time duration of one bit, calculated as the inverse of the bit rate.

samples_per_rate: Determines the number of time samples used to represent one bit. A high number ensures smooth plotting and accurate frequency analysis.

Fs (Sampling Frequency): Defines how frequently the signal is sampled in time. Since we use 1000 samples per bit, Fs = 1000 * Rb = 10 MHz

ts (Sampling Period): Time between each sample = $\frac{1}{F_S}$.

N: Number of samples, rounded up to the nearest integer

t: The time vector is used to represent the signal in the time domain with proper sampling

f: Frequency axis vector for plotting the FFT/PSD later. It is centered around 0 to cover both positive and negative frequencies.

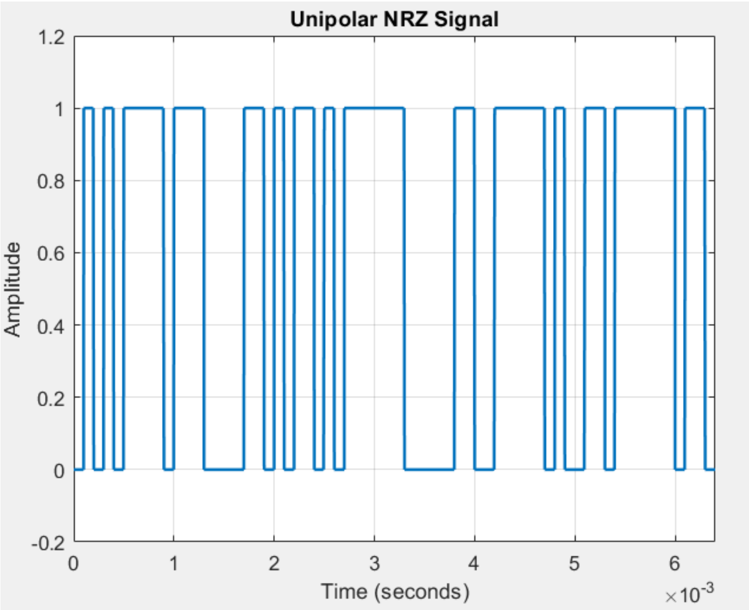
Unipolar Not Return to Zero line coding

In this part, we implement Unipolar NRZ (Non-Return to Zero) line coding. Each bit is mapped to a constant voltage level over its duration:

- Logical '1' is represented by a high level (1).
- Logical '0' is represented by zero (0).

The implementation uses a loop to assign each bit's amplitude over a range of samples corresponding to its time duration.

```
unipolar_nrz=zeros(1,total_samples);


for i =1:num_bit
    start_index=(i-1)*samples_per_rate +1;
    end_index=i*samples_per_rate;
    unipolar_nrz(start_index:end_index) = (stream(i)>0);
end

figure;
plot(t,unipolar_nrz, 'LineWidth', 1.5);
xlabel('Time (seconds)');
ylabel('Amplitude');
title('Unipolar NRZ Signal');
xlim([0 T+5*ts]);
ylim([-0.2 1.2]);
grid on;
```



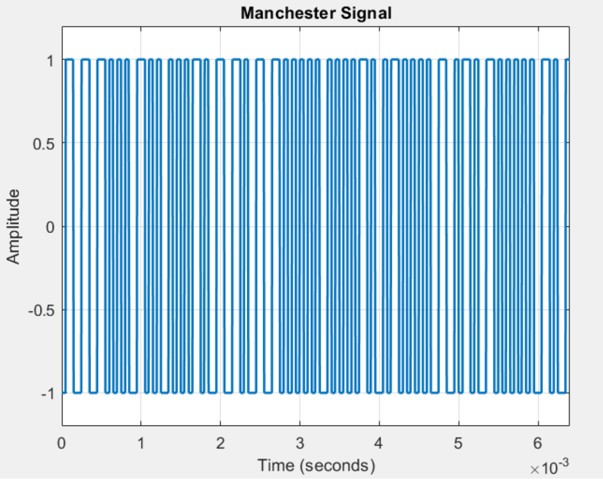
Manchester line coding

In Manchester encoding, each bit is represented by two voltage levels within the same bit period:

- Logical ‘1’ is encoded as high-to-low transition.
- Logical ‘0’ is encoded as low-to-high transition.

This provides both data and clock information, which makes it self-synchronizing.
The implementation divides each bit duration into two halves and assigns the appropriate levels based on the bit value.

```
Manchester=zeros(1,total_samples);

for i =1:num_bit
    start_index=(i-1)*samples_per_rate +1;
    end_index=i*samples_per_rate;
    mid_index = start_index + floor(samples_per_rate/2) - 1;
    if(stream(i)>0)
        Manchester(start_index:mid_index)=1;
        Manchester(mid_index+1:end_index)=-1;
    else
        Manchester(start_index:mid_index)=-1;
        Manchester(mid_index+1:end_index)=1;
    end
end
figure;
plot(t,Manchester, 'LineWidth', 1.5);
xlabel('Time (seconds)');
ylabel('Amplitude');
title('Manchester Signal');
xlim([0 T+5*ts]);
ylim([-1.2 1.2]);
grid on;
```



FFT and PSD for Unipolar NRZ

To analyze the frequency content of the Unipolar NRZ signal, we compute the Fast Fourier Transform (FFT) and Power Spectral Density (PSD):

```
M_unipolar = fftshift(fft(unipolar_nrz))/N;

unipolar_psd = abs(M_unipolar).^2;
```

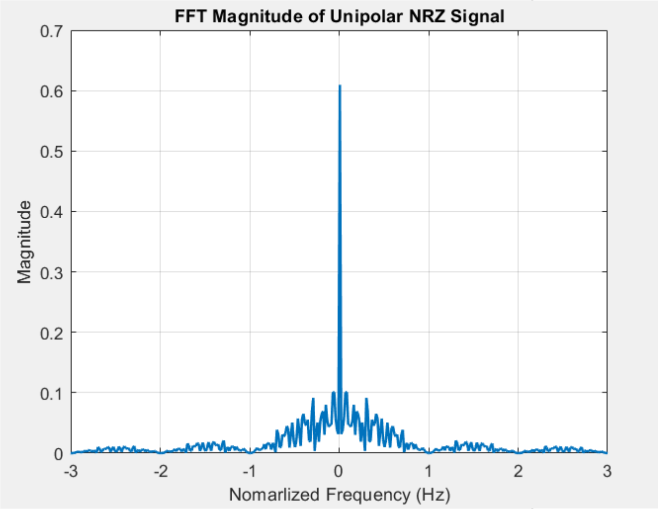
- The signal is transformed to the frequency domain using **fft**, then centered around zero frequency using **fftshift**.

- The Power Spectral Density (PSD) is calculated as the squared magnitude of the FFT result.

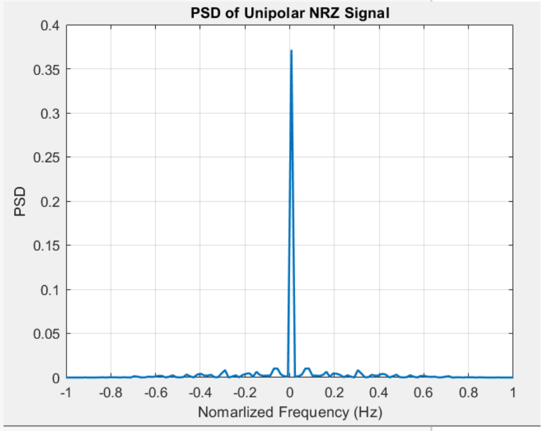
The frequency axis is normalized by the bit rate **tb**, so that the plots are presented in terms of normalized frequency ($f \ * \ t_b$). This helps compare spectra across different line codes independently of their absolute bit rate.

Observation: Unipolar NRZ has a high-power concentration at low frequencies, including a significant DC component (non-zero average value), which limits its effectiveness in systems that use AC coupling or transformers, and reduces synchronization robustness compared to balanced line codes like Manchester.

```
M_unipolar = fftshift(fft(unipolar_nrz))/N;

unipolar_psd = abs(M_unipolar).^2;

figure;

plot(f*tb, abs(M_unipolar), 'LineWidth', 1.5);

xlabel('Nomarlized Frequency (Hz)');

ylabel('Magnitude');

title('FFT Magnitude of Unipolar NRZ Signal');

xlim([-3 3]);


grid on;
```



```
figure;

plot(f*tb, unipolar_psd, 'LineWidth', 1.5);


xlabel('Nomarlized Frequency (Hz)');

ylabel('PSD');

title('PSD of Unipolar NRZ Signal');
xlim([-1 1]);


grid on;
```



FFT and PSD for Manchester

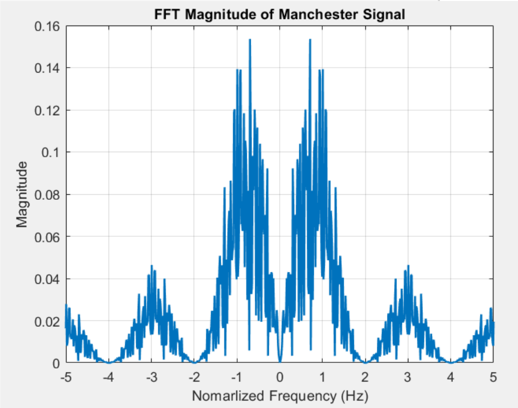
We analyze the frequency characteristics of the Manchester-encoded signal by applying the FFT and computing the Power Spectral Density (PSD):

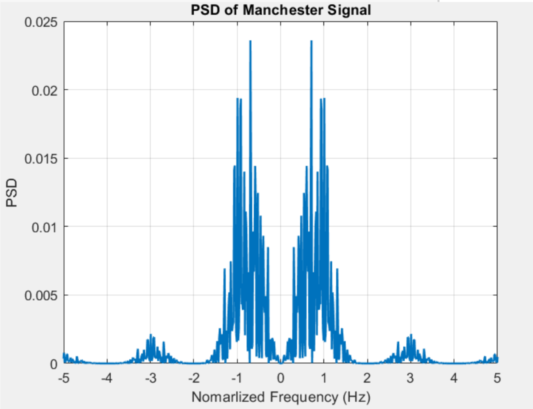
```
M_Manchester = fftshift(fft(Manchester))/N;

Manchester_psd = abs(M_Manchester).^2;
```

The FFT reveals the signal’s spectral content, and the PSD shows how signal power is distributed over frequency.

The frequency axis is normalized using the bit duration tb, allowing for a fair comparison between different coding schemes.

```
M_Manchester = fftshift(fft(Manchester))/N;

Manchester_psd = abs(M_Manchester).^2;

figure;

plot(f*tb, abs(M_Manchester), 'LineWidth', 1.5);


xlabel('Nomarlized Frequency (Hz)');
ylabel('Magnitude');

title('FFT Magnitude of Manchester Signal');
xlim([-5 5])


grid on;


figure;


plot(f*tb, Manchester_psd, 'LineWidth', 1.5);


xlabel('Nomarlized Frequency (Hz)');


ylabel('PSD');


title('PSD of Manchester Signal');


xlim([-5 5])
grid on;
```





Observation:

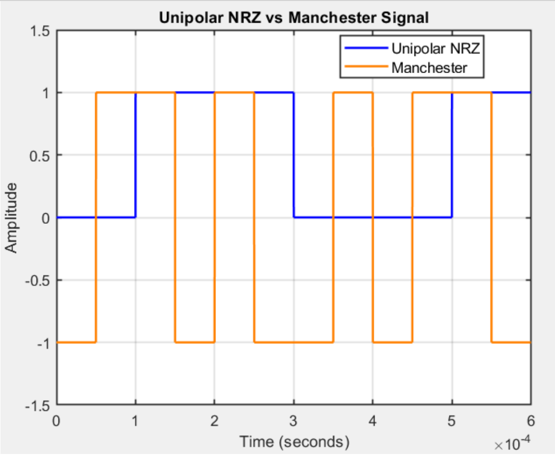
- The Manchester code shows no power at low (including DC) frequencies, which makes it ideal for AC-coupled systems and ensures better clock recovery.
- However, its bandwidth is wider compared to Unipolar NRZ:
  - Manchester bandwidth ≈ 2 × Rb
  - Unipolar NRZ bandwidth ≈ Rb

    This makes Manchester more robust but more demanding in terms of spectral efficiency.

Comprehensive Comparison Between Unipolar NRZ and Manchester Line Coding

To further illustrate the difference in temporal behavior, we display below the random 6 bits of both Unipolar NRZ and Manchester signals in the same time plot for direct comparison:

```
figure;

plot(t, unipolar_nrz, 'b', 'LineWidth', 1.5, 'DisplayName', 'Unipolar NRZ');
hold on;
plot(t, Manchester, 'Color', [1 0.5 0], 'LineWidth', 1.5, 'DisplayName',
'Manchester');
hold off;
xlabel('Time (seconds)', 'FontSize', 12);
ylabel('Amplitude', 'FontSize', 12);
title('Unipolar NRZ vs Manchester Signal', 'FontSize', 14);
xlim([0 T+5*ts]);
ylim([-1.5 1.5]);
grid on;
legend('show', 'Location', 'best', 'FontSize', 10);
set(gca, 'FontSize', 10, 'LineWidth', 1);
```



FEATURE	UNIPOLAR NRZ	MANCHESTER
DC COMPONENT	Non-zero average → High power at DC. This makes it unsuitable for AC-coupling or transformers.	Zero average → DC-balanced, ideal for AC-coupling and transformer-based systems.
POWER DISTRIBUTION	Most power is concentrated in low frequencies, including DC.	Power is shifted away from DC, reducing low-frequency content, which is more efficient.
POWER EFFICIENCY	Wastes power due to constant positive level for 1s.	Efficient use of power; transitions occur for every bit, distributing power more uniformly.
BANDWIDTH (BW)	Narrower: ≈ Rb.	Wider: ≈ 2Rb. More bandwidth required due to transitions every bit.
NOISE IMMUNITY	Less immune to noise: voltage swing is 0 to 1, low margin.	More immune: swing is -1 to +1, giving larger decision margin at the receiver.
TRANSPARENCY	May have issues with long runs of 0s or 1s (e.g., no transitions in "000000").	Fully transparent: all bit patterns (e.g., "00000", "11111", "101010") can be represented.
CLOCK RECOVERY	Difficult: lacks frequent transitions, making timing extraction harder.	Easier: transitions occur every bit, enabling robust clock recovery at the receiver.
ERROR DETECTION	Cannot detect bit errors.	Same: cannot detect bit errors (this is a feature of bipolar codes, not unipolar/Manchester).

- Manchester coding offers superior robustness, clock recovery, and DC balance, making it more suitable for practical communication systems, especially those using AC-coupling or requiring precise synchronization.
- Unipolar NRZ, while having narrower bandwidth, suffers from poor power efficiency and is not suitable for channels that block DC components.

Part 2: Binary Phase-Shift Keying (BPSK)

Overview:

In the simplest form of phase-shift keying known as binary phase-shift keying (BPSK), the pair of signals and used to represent symbols 1 and 0, respectively, are defined by

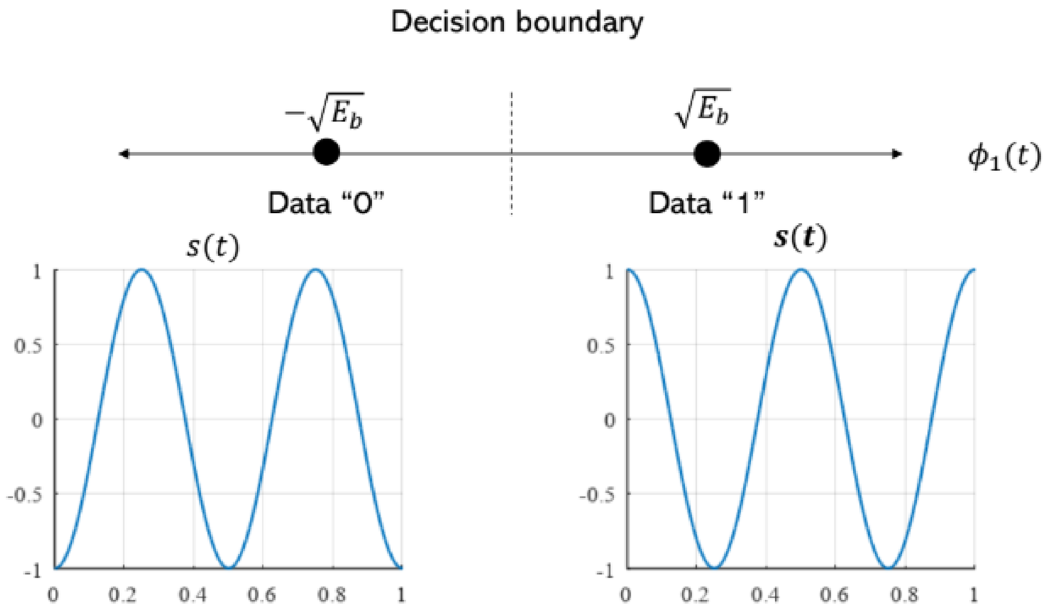
$$s_i(t) = \begin{cases} \sqrt{\dfrac{2E_b}{T_b}}\cos(2\pi f_c t), & \text{for symbol 1 corresponding to } i = 1 \\[2ex] \sqrt{\dfrac{2E_b}{T_b}}\cos(2\pi f_c t + \pi) = -\sqrt{\dfrac{2E_b}{T_b}}\cos(2\pi f_c t), & \text{for symbol 0 corresponding to } i = 2 \end{cases}$$

where each bit is represented by one of two sinusoidal signals that differ in phase by π radians. These signals, known as antipodal signals, maintain a constant envelope, ensuring that the transmitted signal power remains uniform over time. This characteristic makes BPSK a specific case of double-sideband suppressed-carrier (DSB-SC) modulation.

A key distinction between BPSK and other schemes like Binary Amplitude Shift Keying (BASK) lies in this constant envelope property. The modulated BPSK signal preserves its amplitude at a fixed value of √(2Eb/Tb), where *Eb* is the energy per bit and *Tb* is the bit duration. This constancy leads to two main advantages:

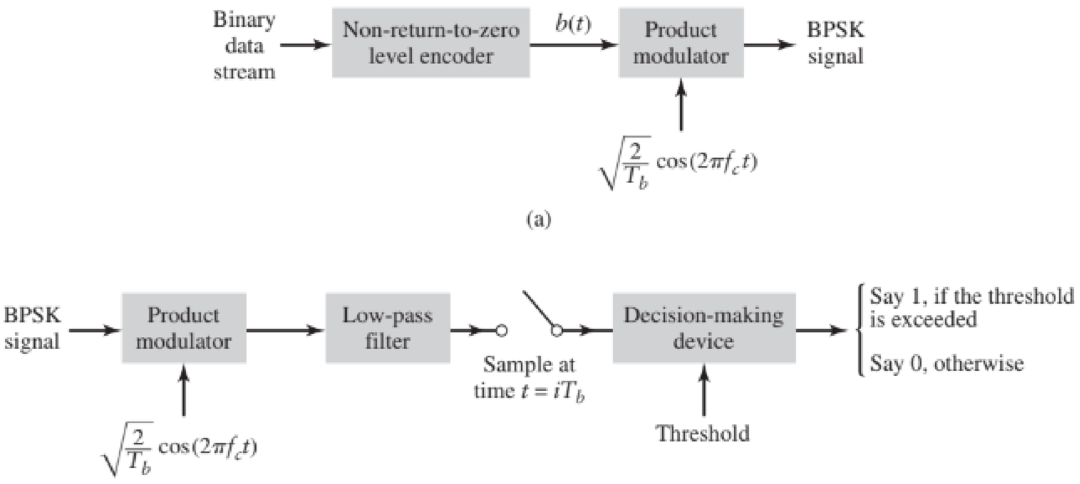
1. the transmitted energy per bit remains constant, implying a stable average power.
2. BPSK requires coherent detection for demodulation, as envelope detection methods are ineffective due to the signal's phase-dependent structure.

Signal Space Diagram



Symbol	Phase	S_i
1	0	$\sqrt{E/2}$
0	π	$-\sqrt{E/2}$

BPSK Modulator and Coherent detector



Binary data stream
Non-return-to-zero level encoder
$b(t)$
Product modulator
BPSK signal
$\sqrt{\frac{2}{T_b}}\cos(2\pi f_c t)$
BPSK signal
Product modulator
Low-pass filter
Sample at time $t = iT_b$
Decision-making device
Threshold
Say 1, if the threshold is exceeded
Say 0, otherwise
$\sqrt{\frac{2}{T_b}}\cos(2\pi f_c t)$

(a)

Octave

Generate a random bitstream

```
stream = randi([0 1], 1, 64);
```

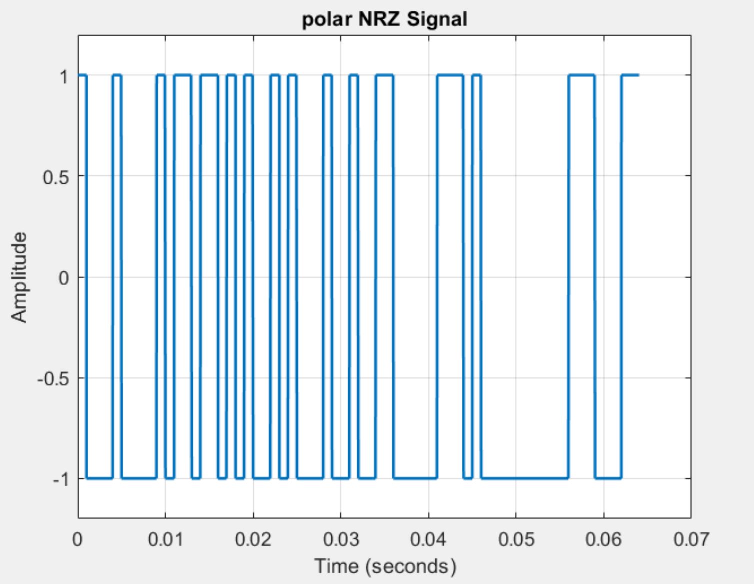
Initialization and Time Vector Generation

```
bit_rate=1e3;

samples_per_rate=1000;  % Each bit will be represented by 100 samples in the time
domain

num_bit=length(stream);
total_samples=num_bit * samples_per_rate;

t = linspace(0, num_bit / bit_rate, total_samples);
```

Polar Not Return to Zero line coding

```
polar_nrz=zeros(1,total_samples);

for i =1:num_bit
    start_index=(i-1)*samples_per_rate +1;
    end_index=i*samples_per_rate;
    polar_nrz(start_index:end_index) = 2 * stream(i) - 1;  % 1 -> +1, 0 -> -1
end

figure;
plot(t,polar_nrz, 'LineWidth', 1.5);
xlabel('Time (seconds)');
ylabel('Amplitude');
title('polar NRZ Signal');
ylim([-1.2 1.2]);
grid on;

disp('Transmitted bits:');
disp(stream);
```

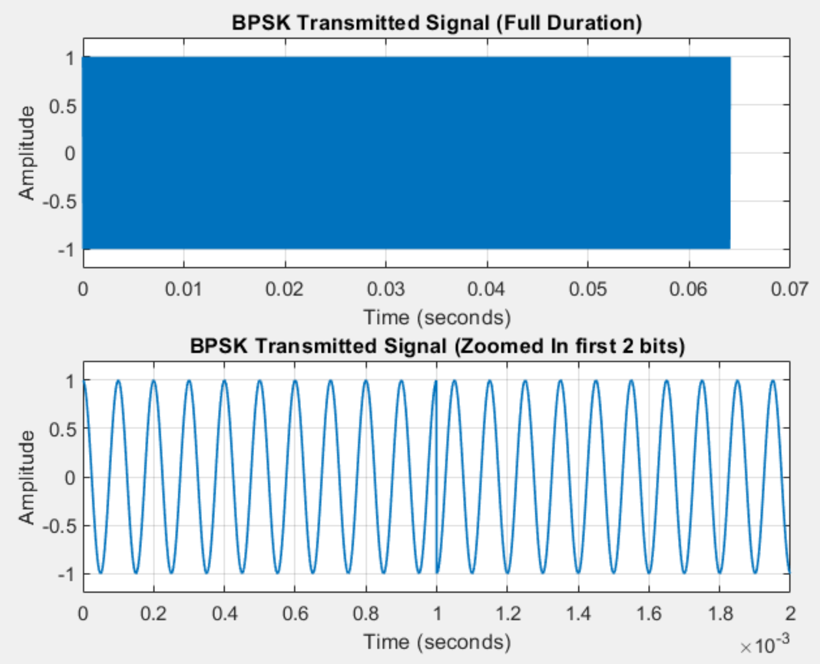
```
Transmitted bits:

  Columns 1 through 13

     1     0     0     0     1     0     0     0     0     1     0     1     1

  Columns 14 through 26

     0     1     1     0     1     0     1     0     0     1     0     1     0

  Columns 27 through 39

     0     0     1     0     0     1     0     0     1     1     0     0     0
  Columns 40 through 52

     0     0     1     1     1     0     1     0     0     0     0     0     0

  Columns 53 through 64

     0     0     0     0     1     1     1     0     0     0     1     1
```



Implement a digital modulation system BPSK transmitter

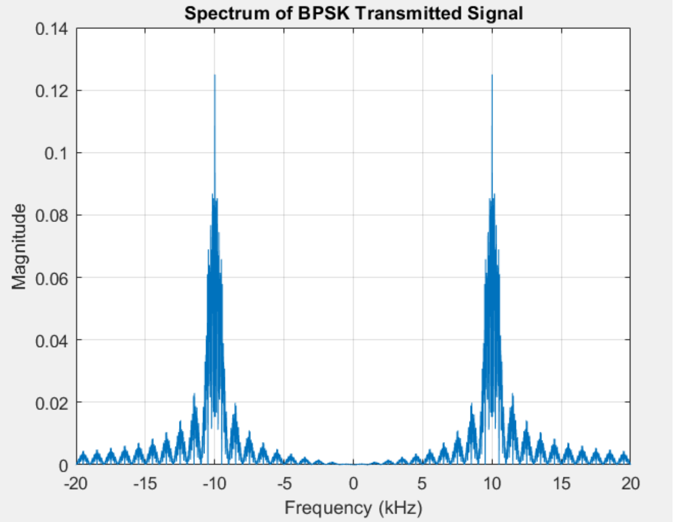
```
%Use bitstream as your baseband input.:(polar_nrz line coding)
energy_bit= 1;
Tb= 1/bit_rate;
%A=sqrt((2*energy_bit)/Tb);
A=1;
fc = 10*bit_rate;
tb = (0:samples_per_rate-1) * (Tb / samples_per_rate);

signal_tx = zeros(1, total_samples);
carrier_wave = A * cos(2 * pi * fc * tb);
for i = 1:num_bit
    start_index=(i-1)*samples_per_rate +1;
    end_index=i*samples_per_rate;
    if polar_nrz(start_index) == 1
        signal_tx(start_index:end_index) = carrier_wave;
    else
        signal_tx(start_index:end_index) = -carrier_wave;
    end
end
figure;


subplot(2,1,1);
plot(t, signal_tx, 'LineWidth', 1);
xlabel('Time (seconds)');
ylabel('Amplitude');
title('BPSK Transmitted Signal (Full Duration)');
ylim([-1.2*A 1.2*A]);
```

```
grid on;

subplot(2,1,2);
plot(t, signal_tx, 'LineWidth', 1);
xlabel('Time (seconds)');
ylabel('Amplitude');
title('BPSK Transmitted Signal (Zoomed In first 2 bits)');
xlim([0 0.002]);
ylim([-1.2*A 1.2*A]);
grid on;
```



Spectrum of BPSK Transmitted Signal

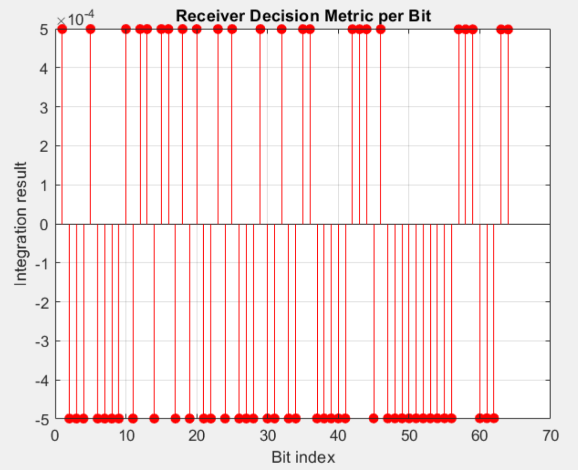
```
T=num_bit /bit_rate ;

fs = samples_per_rate*bit_rate;

ts= 1/fs;

df = fs/total_samples;

N=ceil (T/ts);

if mod(N, 2) == 0
    f=-(fs/2) : df : (fs/2)- df ;
else
    f=-((fs/2)-(df/2)) : df : (fs/2)- (df/2) ;
end
figure;
signal_fft = fftshift(fft(signal_tx))*(1/N);
plot(f/1000, abs(signal_fft));
xlabel('Frequency (kHz)');
ylabel('Magnitude');
title('Spectrum of BPSK Transmitted Signal');
```

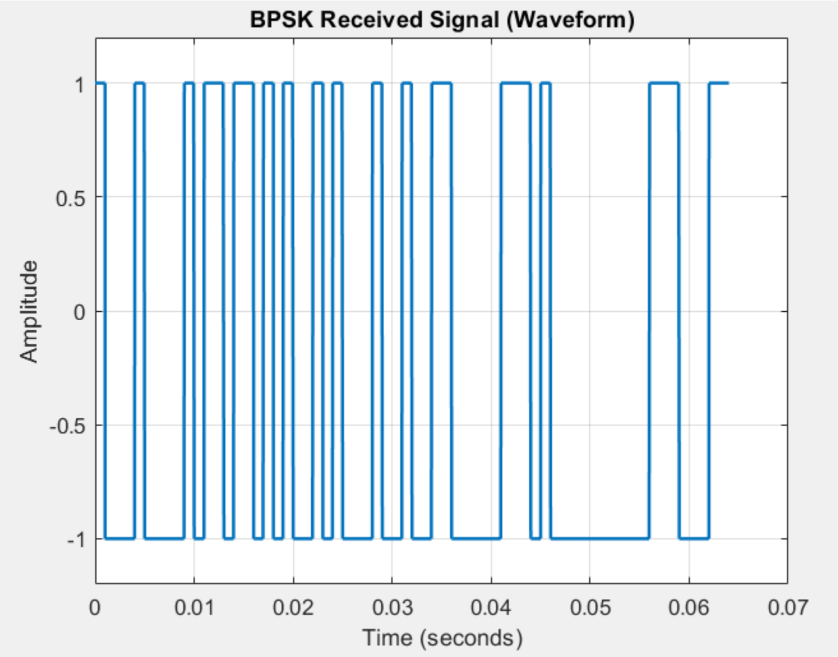


```
grid on;
```

Digital modulation system BPSK receiver

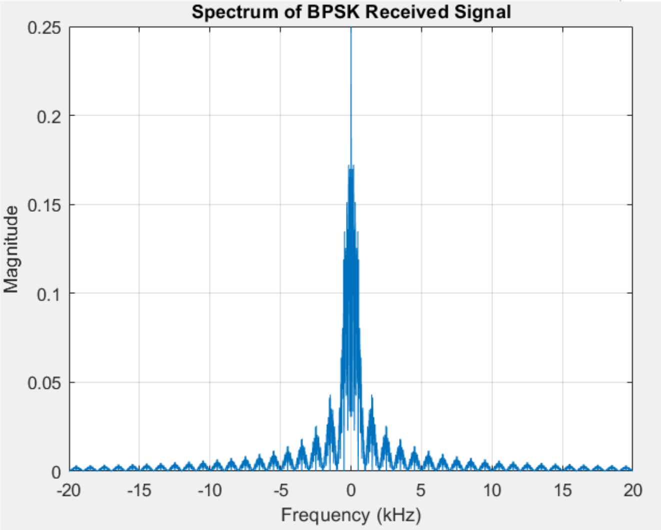
```
% Create matching carrier wave for full duration
carrier_wave_rx = A * cos(2 * pi * fc * t); % Full-length carrier
v = signal_tx .* carrier_wave_rx;           % Coherent demodulation

% Initialize
rx_bits = zeros(1, num_bit);
decision_points = zeros(1, num_bit);

for i = 1:num_bit
    start_index = (i-1)*samples_per_rate + 1;
    end_index = i*samples_per_rate;

    v_segment = v(start_index:end_index);

    t_segment = t(start_index:end_index);  % Time vector for current bit
    integration_result = trapz(t_segment, v_segment);

    decision_points(i) = integration_result;

    % Decision based on sign
    if integration_result > 0
        rx_bits(i) = 1;
    else
        rx_bits(i) = -1;
    end
end

% Compare transmitted vs received bits
disp('Transmitted bits:');
disp(stream);
disp('Received bits:');
disp(rx_bits);

% Plot received signal decision points with red balls
figure;
h = stem(1:num_bit, decision_points, 'filled', 'r');
h.MarkerFaceColor = 'r';
xlabel('Bit index');
ylabel('Integration result');
title('Receiver Decision Metric per Bit');
grid on;

rx_wave = zeros(1, total_samples);
for i = 1:num_bit
    start_index = (i-1)*samples_per_rate + 1;
    end_index = i*samples_per_rate;
    rx_wave(start_index:end_index) = rx_bits(i);
end

figure;
plot(t, rx_wave, 'LineWidth', 1.5);
xlabel('Time (seconds)');
ylabel('Amplitude');
title('BPSK Received Signal (Waveform)');
ylim([-1.2 1.2]);
grid on;
```





Spectrum of BPSK Transmitted Signal

```
T=num_bit /bit_rate ;

fs = samples_per_rate*bit_rate;

ts= 1/fs;

df = fs/total_samples;

N=ceil (T/ts);



if mod(N, 2) == 0
    f=-(fs/2) : df : (fs/2)- df ;
else
    f=-((fs/2)-(df/2)) : df : (fs/2)- (df/2) ;
end

figure;
rx_wave_fft = fftshift(fft(rx_wave))*(1/N);
plot(f/1000, abs(rx_wave_fft));
xlabel('Frequency (kHz)');
ylabel('Magnitude');
title('Spectrum of BPSK Received Signal');
grid on;
xlim([-20 20]);
```

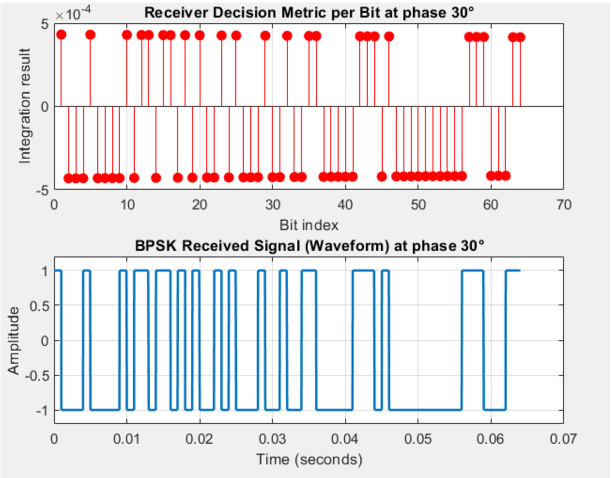


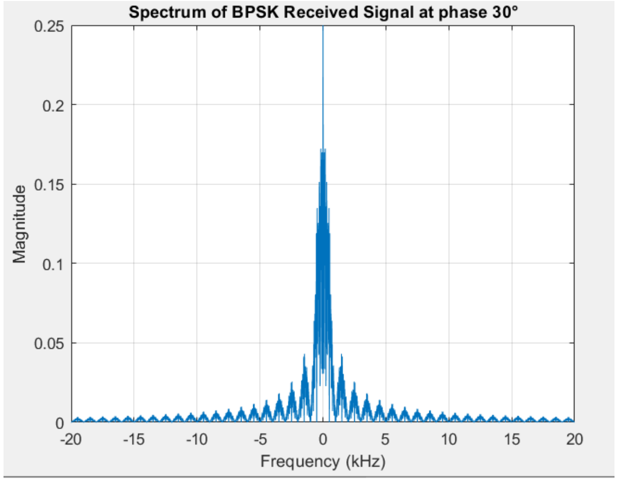
Digital modulation system BPSK receiver with phases

```
phase = [30 60 90] * pi/180;

% Initialize
rx_bits = zeros(1, num_bit);
decision_points = zeros(1, num_bit);
for j = 1:3
    ph = phase(j);
    carrier_wave_rx = A * cos((2 * pi * fc * t)+ph); % Full-length carrier
v = signal_tx .* carrier_wave_rx;           % Coherent demodulation
for i = 1:num_bit
    start_index = (i-1)*samples_per_rate + 1;
    end_index = i*samples_per_rate;

    v_segment = v(start_index:end_index);

    t_segment = t(start_index:end_index);  % Time vector for current bit
    integration_result = trapz(t_segment, v_segment);

    decision_points(i) = integration_result;

    % Decision based on sign
    if integration_result > 0
        rx_bits(i) = 1;
    else
        rx_bits(i) = -1;
    end
end
    % Compare transmitted vs received bits
    disp('Transmitted bits:');
    disp(stream);
    fprintf('Received bits at phase %.0f° \n', ph * 180 / pi);
    disp(rx_bits);

    figure ;
    % Plot received signal decision points
    subplot(2,1,1);
    h = stem(1:num_bit, decision_points, 'filled', 'r');
    h.MarkerFaceColor = 'r';
    xlabel('Bit index');
    ylabel('Integration result');
    title(sprintf('Receiver Decision Metric per Bit at phase %.0f°', ph * 180 / pi));
    grid on;

    rx_wave = zeros(1, total_samples);
    for i = 1:num_bit
        start_index = (i-1)*samples_per_rate + 1;
        end_index = i*samples_per_rate;
        rx_wave(start_index:end_index) = rx_bits(i);
    end

    subplot(2,1,2);
    plot(t, rx_wave, 'LineWidth', 1.5);
    xlabel('Time (seconds)');
    ylabel('Amplitude');
    title(sprintf('BPSK Received Signal (Waveform) at phase %.0f°',  ph*180/pi));
    ylim([-1.2 1.2]);
    grid on;
    T=num_bit /bit_rate ;
    fs = samples_per_rate*bit_rate;
    ts= 1/fs;
    df = fs/total_samples;
    N=ceil (T/ts);

    if mod(N, 2) == 0
        f=-(fs/2) : df : (fs/2)- df ;
    else
        f=-((fs/2)-(df/2)) : df : (fs/2)- (df/2) ;
    end

    figure;
    rx_wave_fft = fftshift(fft(rx_wave))*(1/N);
    plot(f/1000, abs(rx_wave_fft));
    xlabel('Frequency (kHz)');
    ylabel('Magnitude');
    title('Spectrum of BPSK Received Signal');
    title(sprintf('Spectrum of BPSK Received Signal at phase %.0f°',  ph*180/pi));
    grid on;
    xlim([-20 20]);

    % Convert transmitted stream from [1 0 1 ...] to [+1 -1 +1 ...] for comparison
tx_bits = 2*stream - 1;

% Calculate number of bit errors
num_errors = sum(rx_bits ~= tx_bits);

% Bit Error Rate (BER)
ber = num_errors / num_bit;

fprintf('Number of bit errors at phase %.0f°: %d\n', ph * 180 / pi, num_errors);
fprintf('Bit Error Rate (BER) at phase %.0f°: %.4f\n', ph * 180 / pi, ber);

end
```

Phase =30

```
Number of bit errors at phase 30°: 0

Bit Error Rate (BER) at phase 30°: 0.0000
```

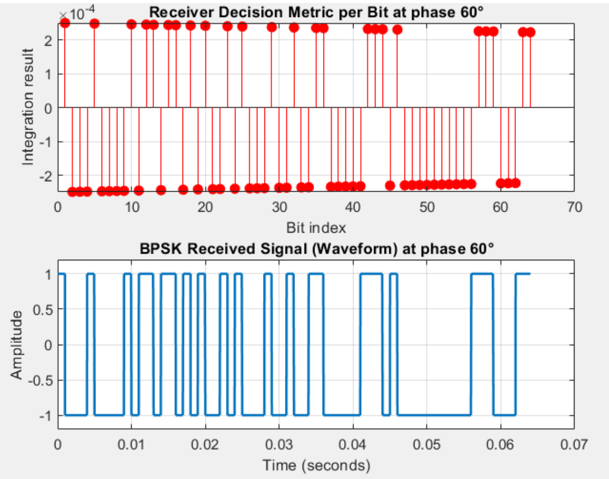


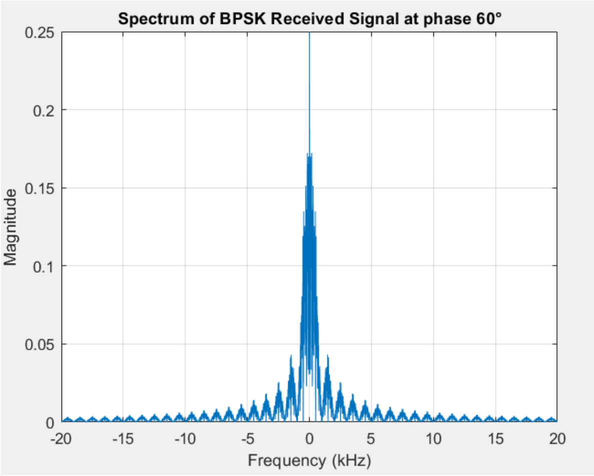


Phase = 60

Number of bit errors at phase 60°: 0

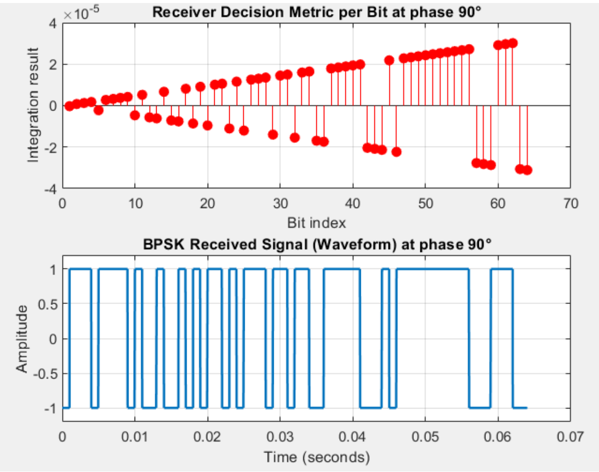
Bit Error Rate (BER) at phase 60°: 0.0000

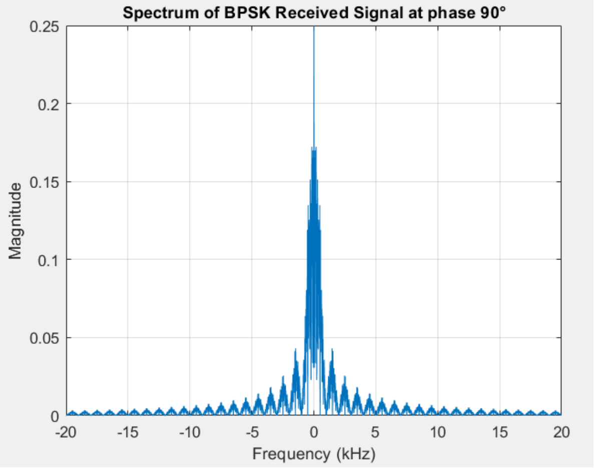




Phase = 90

```
Number of bit errors at phase 90°: 64

Bit Error Rate (BER) at phase 90°: 1.0000
```





Comment

When the oscillator phase is set to 30° and 60°, no mismatch occurs between the transmitted and received bits. This is because the phase shift is still within a range where the demodulator can accurately interpret the signal. Although there is a phase offset, the integrator output (or the output of a low-pass filter, if used) is still sufficiently large to allow the decision device to correctly distinguish between 0 and 1. However, it's important to note that even though 30° and 60° produce correct results now (especially in a noise-free environment), the output amplitude is smaller than in the ideal case of 0° phase shift, where maximum correlation is achieved. So, while the system still works correctly, it is not operating at maximum signal strength.

In contrast, at 90° phase shift, the received signal is completely mismatched with the transmitted one. This is because the transmitter may be using a cosine waveform, while the receiver uses a sine waveform, which are orthogonal to each other. As a result, the integrator or low-pass filter output becomes very small, approaching zero. This weak output then reaches the decision device, which may fail to confidently decide whether the bit is a 0 or a 1. In this specific case, the system interpreted all bits in reverse—the received bits were the exact inversion of the transmitted bits, causing a bit error rate (BER) of 1 .

Conclusion

In Part 2, we implemented and analyzed Binary Phase Shift Keying (BPSK) using Octave. The simulation demonstrated how BPSK modulates digital data by shifting the phase of a carrier signal—typically by 180 degrees—to represent binary symbols. We generated a random bit stream, modulated it using BPSK, and successfully reconstructed the original data using coherent demodulation.

The results confirmed that BPSK is a bandwidth-efficient and power-efficient modulation technique, especially suitable for noisy channels due to its simple structure and robustness against amplitude variations. The clear distinction between phase states allows reliable data recovery at the receiver, making BPSK a fundamental modulation method in digital communication systems.

References

Introduction to Analog and Digital Communications, Simon Haykin, Michael Moher

GitHub Link

https://github.com/Ammar-Wahidi/Analog_Digital-Communication-Project