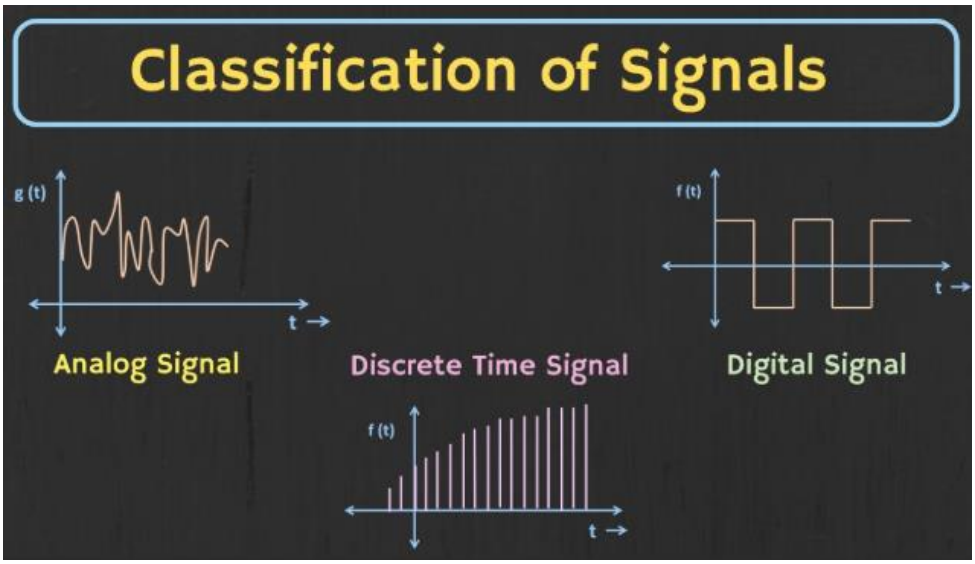
ECE 252

Fundamental of Communication Systems



Name	ID	DEP
Hana Walaa Abdelhakim	2200653	ECE
Alaa Abdelhakeem Mahmoud	2200498	CSE
Nour Ali Abdelrahman	2200292	CSE
Neama Esam Mohamed Saied	2200371	ECE
Miran Mohamed Sayed	2200604	CSE
Mariam Amr Mohammad	2200704	ECE

Analog:

- ## Project Description:

  This project involves comprehensive analysis and implementation of various analog communication techniques using Octave.

  The work focuses on signal processing, Fourier analysis, filtering, and modulation/demodulation schemes for two distinct signals.

- ## Project Components:

  1. Signal Analysis

  Plot `x(t)` and derive its Fourier transform analytically and numerically in Octave.

  Estimate bandwidth (BW) where power drops to 5% of its peak.

  2. Filtering Effects

  Apply ideal LPFs (BW = 1Hz and 0.3Hz) to `x(t)` and compare input/output signals.

  3. Second Signal `m(t)`

  Repeat signal analysis (time/frequency domain) for a piecewise cosine signal.

  4. FDM Modulation

  Modulate `x(t)` (DSB-SC at 20Hz) and `m(t)` (SSB) with a 2Hz guard band.

  Choose USB/LSB, define carrier `$c_2(t)$`, and plot composite signal `s(t)` and spectrum `S(f)`.

  5. Demodulation

  Implement coherent demodulators for both channels and recover original signals.

  6.Tools:

  Octave for simulation and analysis.

Project :

- **Analog:**

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

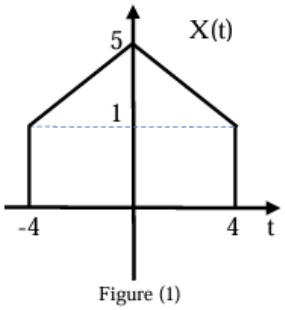
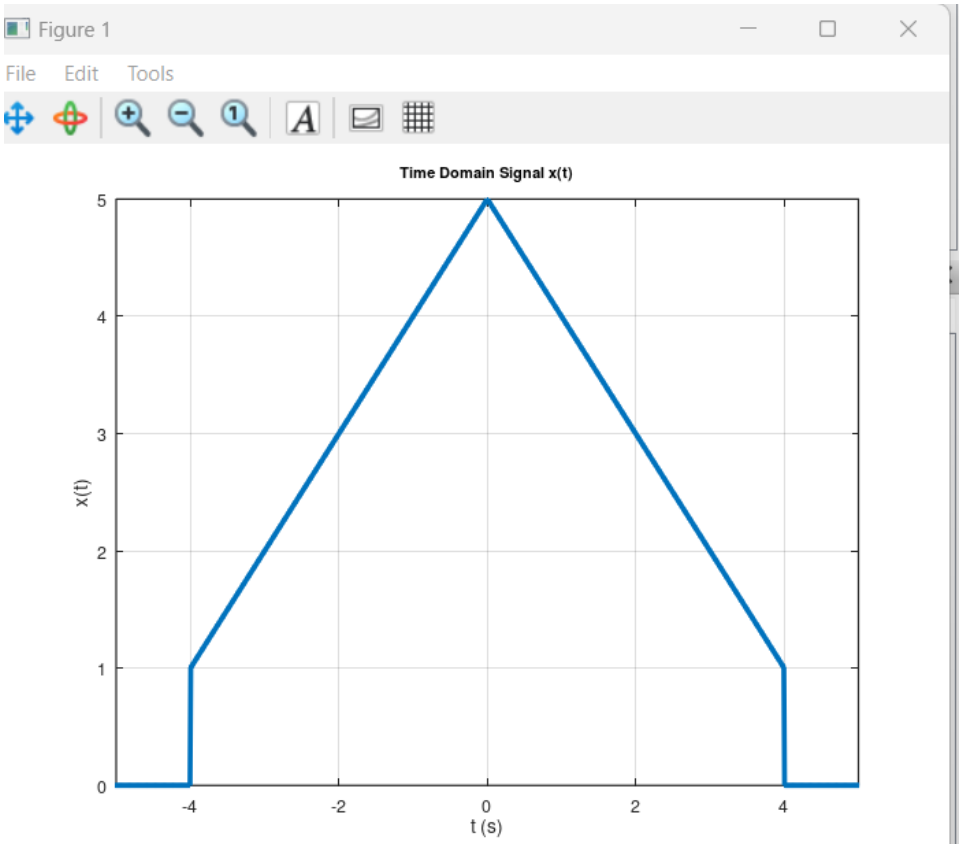


Figure (1)

- Code:

```
>> clear
>> close all
>> %% 1. plot the signal x(t)
>> fs = 100;
>> ts = 1/fs;
>> df = 0.01;
>> T_m = 1/df;
>> N = ceil(T_m/ts);
>> t = -((N-1)*ts)/2:ts:((N-1)*ts)/2;
>> x = zeros(size(t));
>> leftside = (t > -4) & (t < 0);
>> x(leftside) = t(leftside) + 5;
>> rightside = (t >= 0) & (t < 4);
>> x(rightside) = 5 - t(rightside);
>> figure;
>> plot(t, x, 'LineWidth',1.5);
>> xlabel('t (s)');
>> xlim([-5 5]);
>> ylabel('x(t)');
>> title('Time Domain Signal x(t)');
>> grid on;
```

- Output:



2. Derive an analytical expression for its Fourier transform.

- Code:

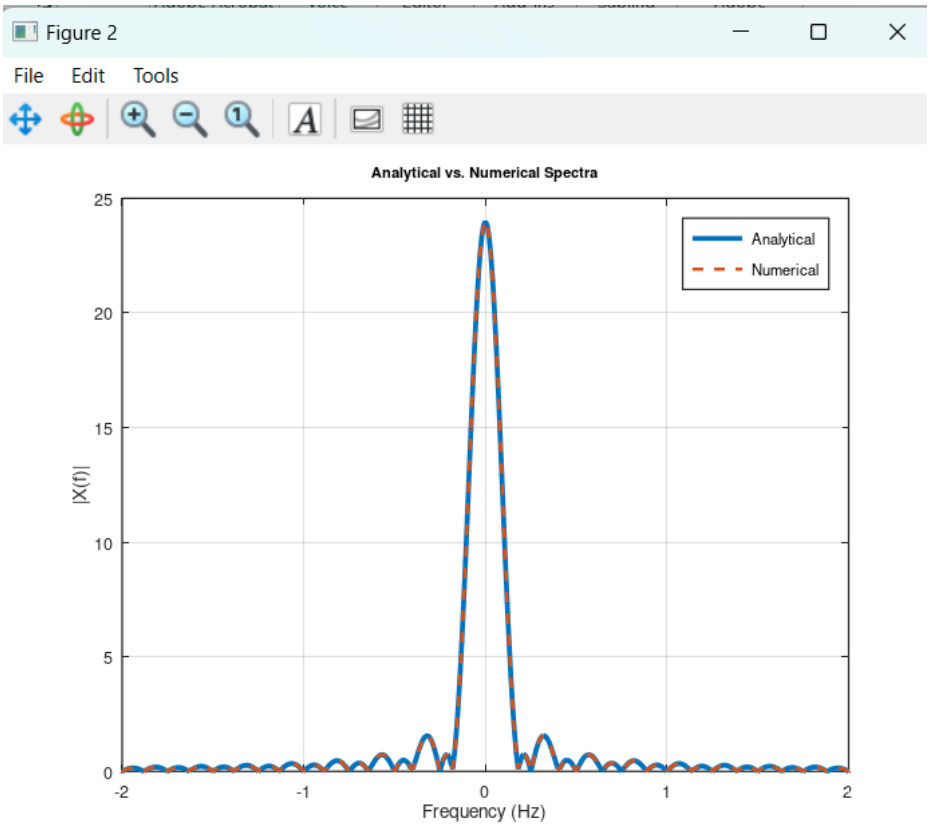
```
>> %% 2. analytical expression fourier transform X(f) to plot later

>> if(rem(N,2)==0)
  f = - (0.5*fs) : df : (0.5*fs-df) ;
else %% Odd
  f = - (0.5*fs-0.5*df) : df : (0.5*fs-0.5*df) ;
end
>> X_analytic = 16*((sinc(4*f)).^2) + 8*sinc(8*f);
>> X_analytic(f == 0) = 24;
```

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.

- Code:

```
>> %% 3. calculate fourier transform
>> X = fftshift(fft(x))*ts;
>> figure;
>> plot(f, abs(X_analytic), 'LineWidth',1.5);
>> hold on;
>> plot(f, abs(X), '--','LineWidth',1);
>> xlabel('Frequency (Hz)');
>> ylabel('|X(f)|');
>> legend('Analytical','Numerical');
>> title('Analytical vs. Numerical Spectra');
>> xlim([-2, 2]);
>> grid on;
```

- Output:



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

- Code:

```
>> %% 4. estimate the BW
>> Power = abs(X_analytic).^2;
>> Pmax = max(Power);
>> bandi = find(Power >= 0.05 * Pmax);
>> index = find(Power == Pmax);
>> BW_est = max(abs(f(bandi)));
>> disp(['Estimated BW of x(t): ', num2str(BW_est), ' Hz']);
Estimated BW of x(t): 0.13 Hz
```

- Output:

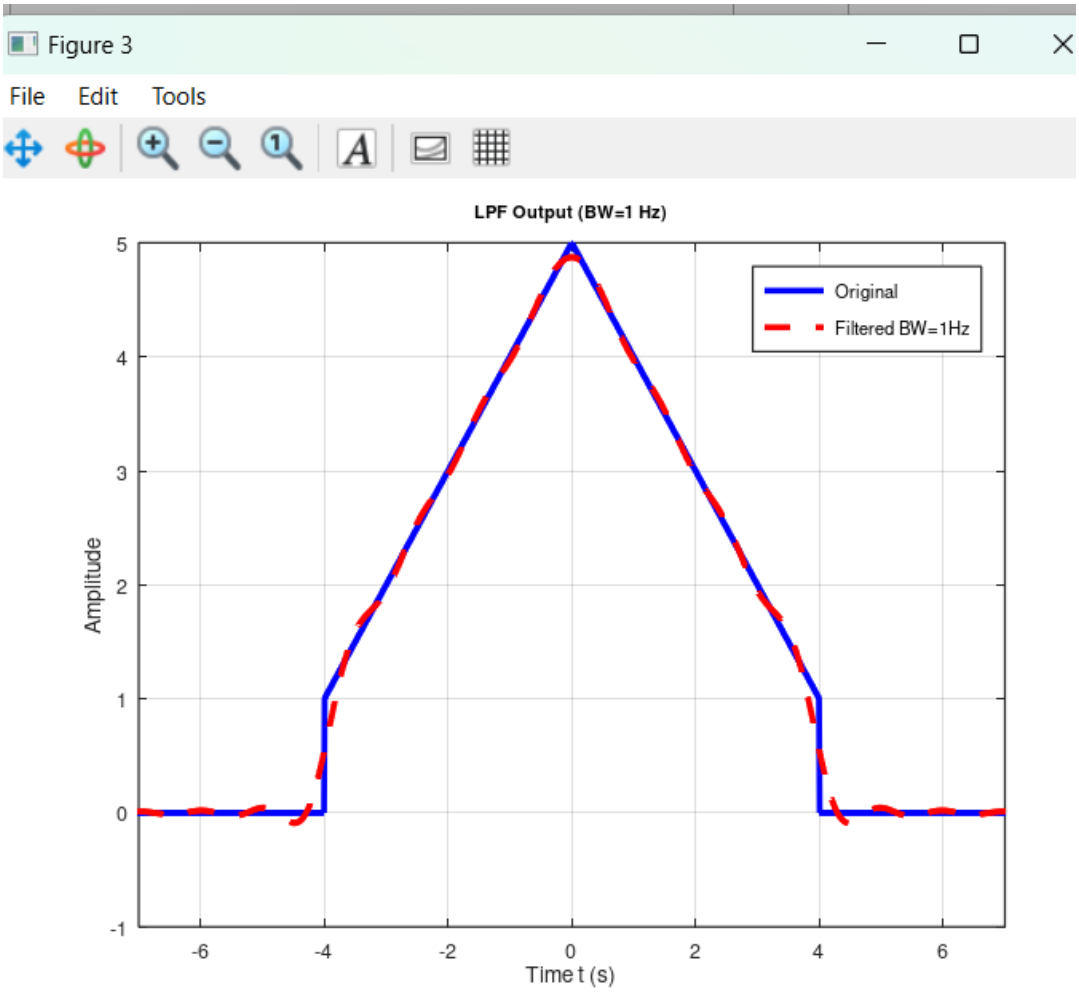
```
>> disp(['Estimated BW of x(t): ', num2str(BW_est), ' Hz']);
Estimated BW of x(t): 0.13 Hz
```

5. If this signal is to pass through a perfect LPF with BW= 1Hz. Plot the output of the filter in the time domain along with the input signal.

- Code:

```
>> %% 5. ideal LPF with BW = 1 Hz
>> H1 = abs(f) <= 1;
>> Y1 = X .* H1;
>> y1 = real(ifft(ifftshift(Y1) /ts));
>> figure;
>> plot(t, x, 'b','LineWidth',1.5);
>> hold on;
>> plot(t, y1, 'r--','LineWidth',1.5);
>> xlabel('Time t (s)');
>> xlim([-7 7]);
>> ylabel('Amplitude');
>> legend('Original','Filtered BW=1Hz');
>> title('LPF Output (BW=1 Hz)');
>> grid on;
```

- Output:

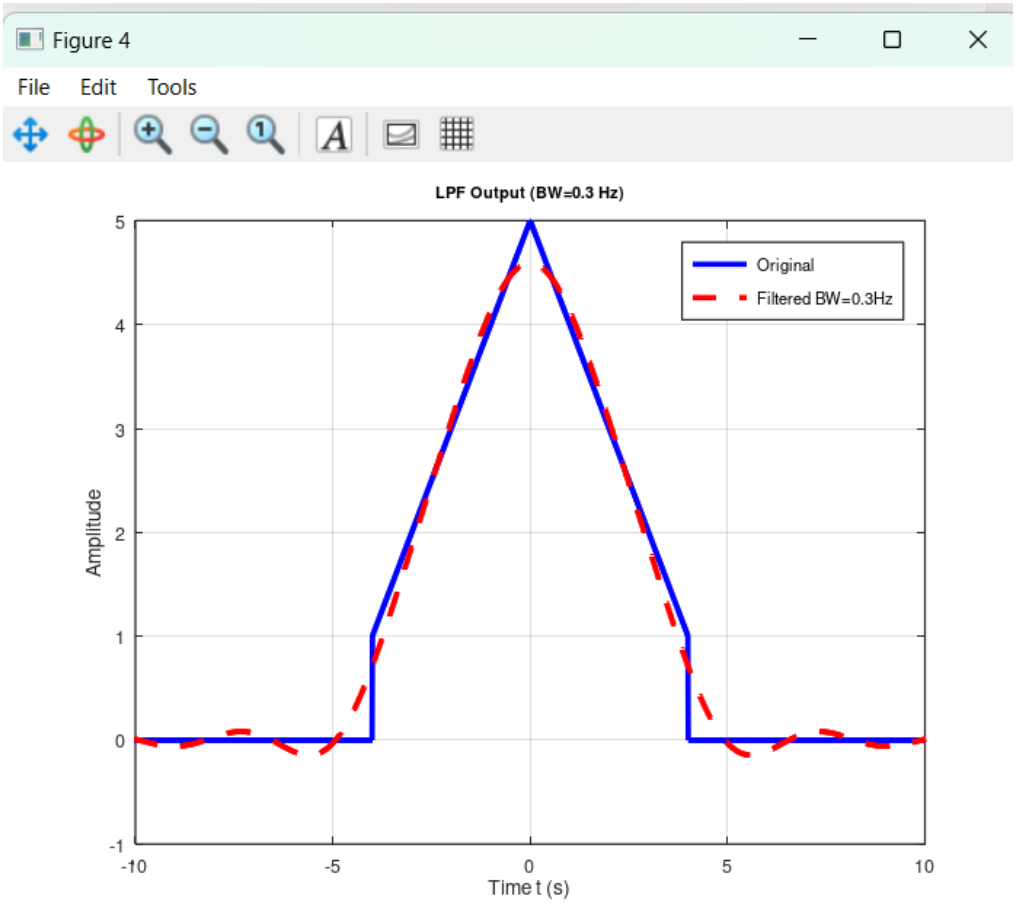


6. Repeat (5) if the LPF BW is reduced to be = 0.3 Hz.

- Code:

```
%% 6. ideal LPF with BW = 0.3 Hz
H2 = abs(f) <= 0.3;
Y2 = X .* H2;
y2 = real(ifft(ifftshift(Y2) /ts));
figure;
plot(t, x, 'b','LineWidth',1.5);
hold on;
plot(t, real(y2(1:length(t))), 'r--','LineWidth',1.5);
xlabel('Time t (s)');
xlim([-10 10]);
ylabel('Amplitude');
legend('Original','Filtered BW=0.3Hz');
title('LPF Output (BW=0.3 Hz)');
grid on;
```

- Output:



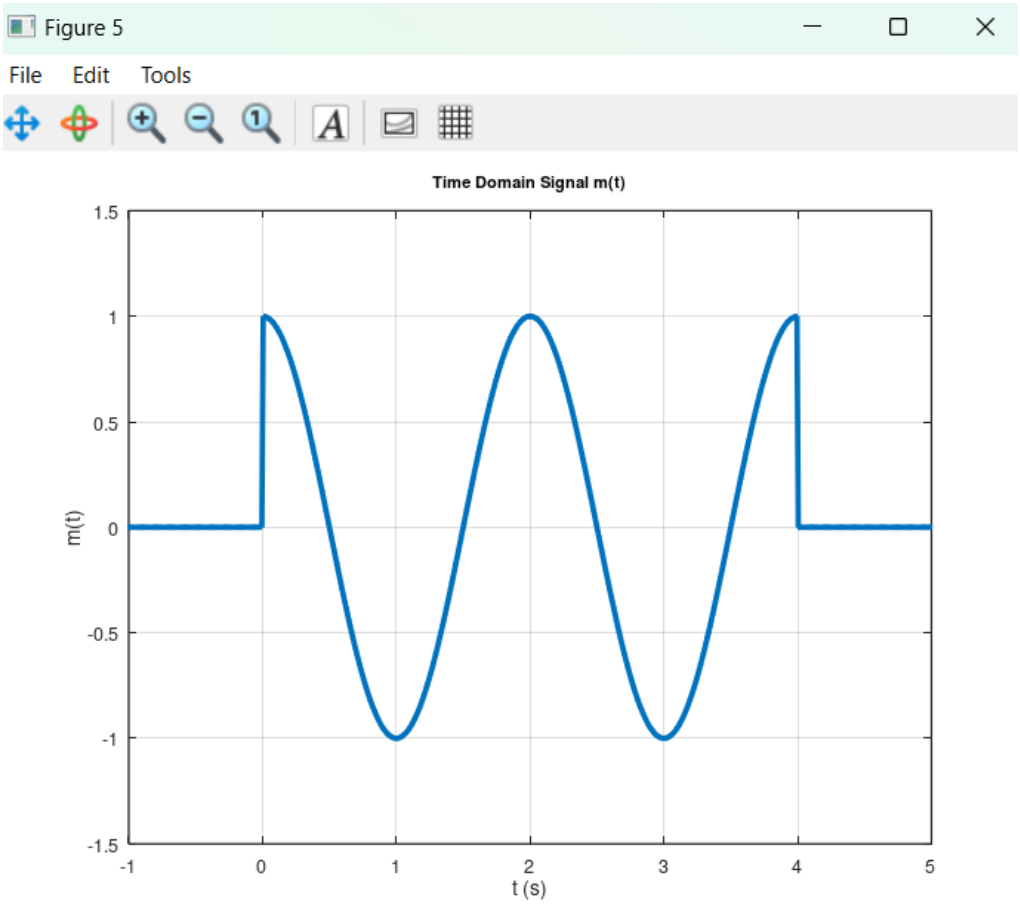
7. For m(t) defined as below, Repeat steps 1-4.

$$m(t) = \begin{cases} \cos(2\pi * 0.5 * t) & 0 < t < 4 \\ 0 & otherwise \end{cases}$$

- Code:(STEP ONE)

```
Command Window
>> %% 7.1) define the signal m(t)
>> m = zeros(size(t));
>> fm = 0.5;
>> m(rightside) = cos(2*pi*fm*t(rightside));
>> figure;
>> plot(t, m, 'LineWidth',1.5);
>> xlabel('t (s)');
>> xlim([-1 5]);
>> ylabel('m(t)');
>> ylim([-1.5 1.5]);
>> title('Time Domain Signal m(t)'); grid on;
```

- Output:



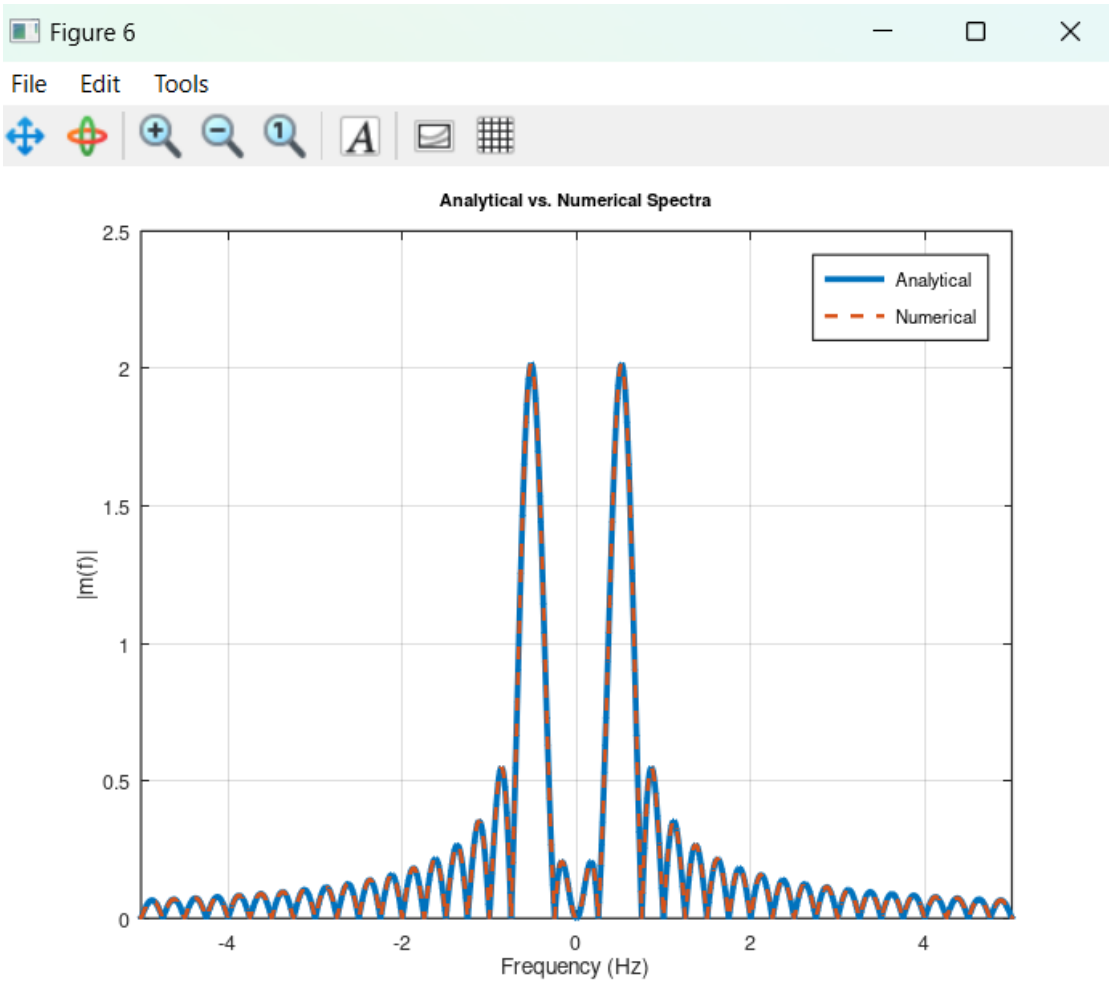
- Code:(STEP TWO)

```
>> %% 7.2) analytical expression fourier transform M(f) to plot later
>> T_m = 4;
>> M_analytic = 2 * ( sinc((f - fm) * T_m) .* exp(-1j * pi * (f - fm) * T_m) + ...
    sinc((f + fm) * T_m) .* exp(-1j * pi * (f + fm) * T_m) );
```

- Code: (STEP 3)

```
>> %% 7.3) calculate fourier transform of m(t)
>> M = fftshift(fft(m))*ts;
>> figure;
>> plot(f, abs(M_analytic), 'LineWidth',1.5);
>> hold on;
>> plot(f, abs(M), '--','LineWidth',1);
>> xlabel('Frequency (Hz)');
>> ylabel('|m(f)|');
>> legend('Analytical','Numerical');
>> title('Analytical vs. Numerical Spectra');
>> xlim([-5,5]);
>> grid on;
```

- Output:



- Code: (STEP 4)

```
>> %% 7.4) estimate the BW of m(t)
>> Power_m = abs(M_analytic).^2;
>> Pm = max(Power_m);
>> bandm = abs(f(Power_m>=0.05*Pm));
>> BW_m = max(bandm);
>> fprintf('Estimated BW of m(t): %.2f Hz\n',BW_m);
Estimated BW of m(t): 0.91 Hz
```

- Output:

```
>> fprintf('Estimated BW of m(t): %.2f Hz\n',BW_m);
Estimated BW of m(t): 0.91 Hz
```

8-FDM Modulation Scheme: It is required to transmit x(t) and m(t) on different channels. Where x(t) is modulated in DSB-SC, and m(t) is modulated by SSB. Each channel bandwidth is 2 Hz. - - The modulated signal is s1(t): x(t) is to modulate a carrier signal c1(t) = cos (ωct) with carrier frequency fc = 20Hz, Use x(t) from step 5. The modulated signal is s2(t): m(t) is to modulate a carrier signal c2(t), such that there is only 2 Hz guard (empty) band between the two channels.

9. State whether you will use USB or LSB. → USB

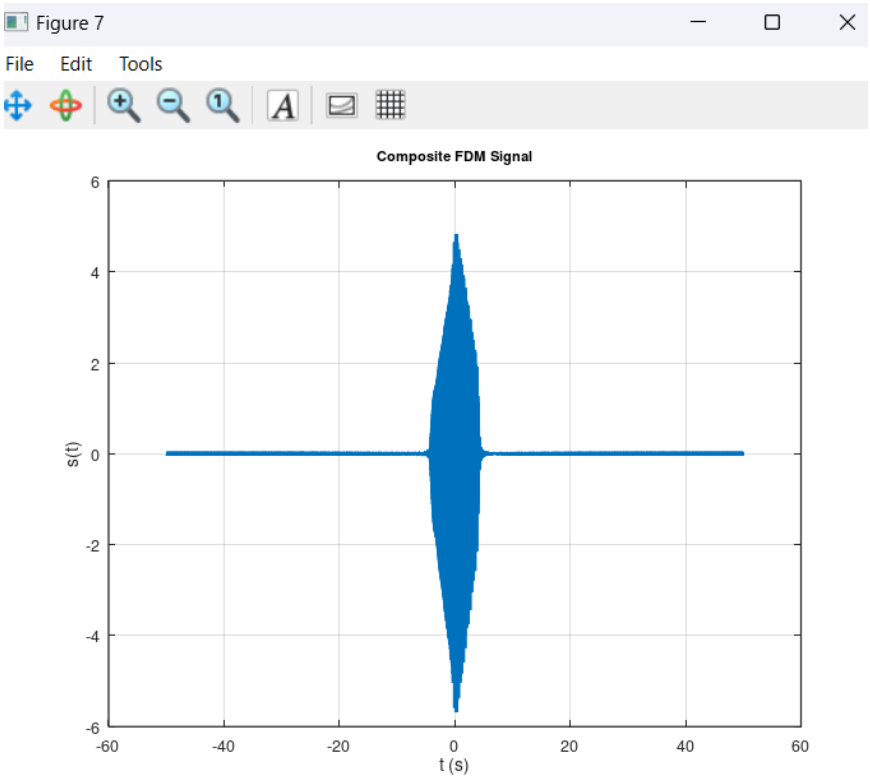
10. Write an appropriate value for c2(t) → C2= 2*Cos(2*PI*fc2*t)

11. Plot s(t) which is s1(t) + s2(t) then Plot S(f).

- Code:

```
>> %% 8. FDM modulation (x(t) in DSB-SC, m(t) in SSB)
>> fc1 = 20;
>> c1 = cos(2*pi*fc1*t);
>> s1 = y1 .* c1;
>> fc2 = 23; %% fc1(20) + BW(1) + guard band(2)
>> c2 = 2 * cos(2*pi*fc2*t);
>> H = (f>fc2 & f<(fc2+2)) | (f<-fc2 & f>-(fc2+2)); %% USB
>> s2 = m .* c2;
>> S2 = fftshift(fft (s2)) *ts;
>> S2 = S2.*H;
>> s2 = real(ifft(ifftshift(S2) / ts));
>> s = s1 + s2;
>> figure;
>> plot(t, s, 'LineWidth',1.2);
>> xlabel('t (s)');
>> ylabel('s(t)');
>> title('Composite FDM Signal');
>> grid on;
```

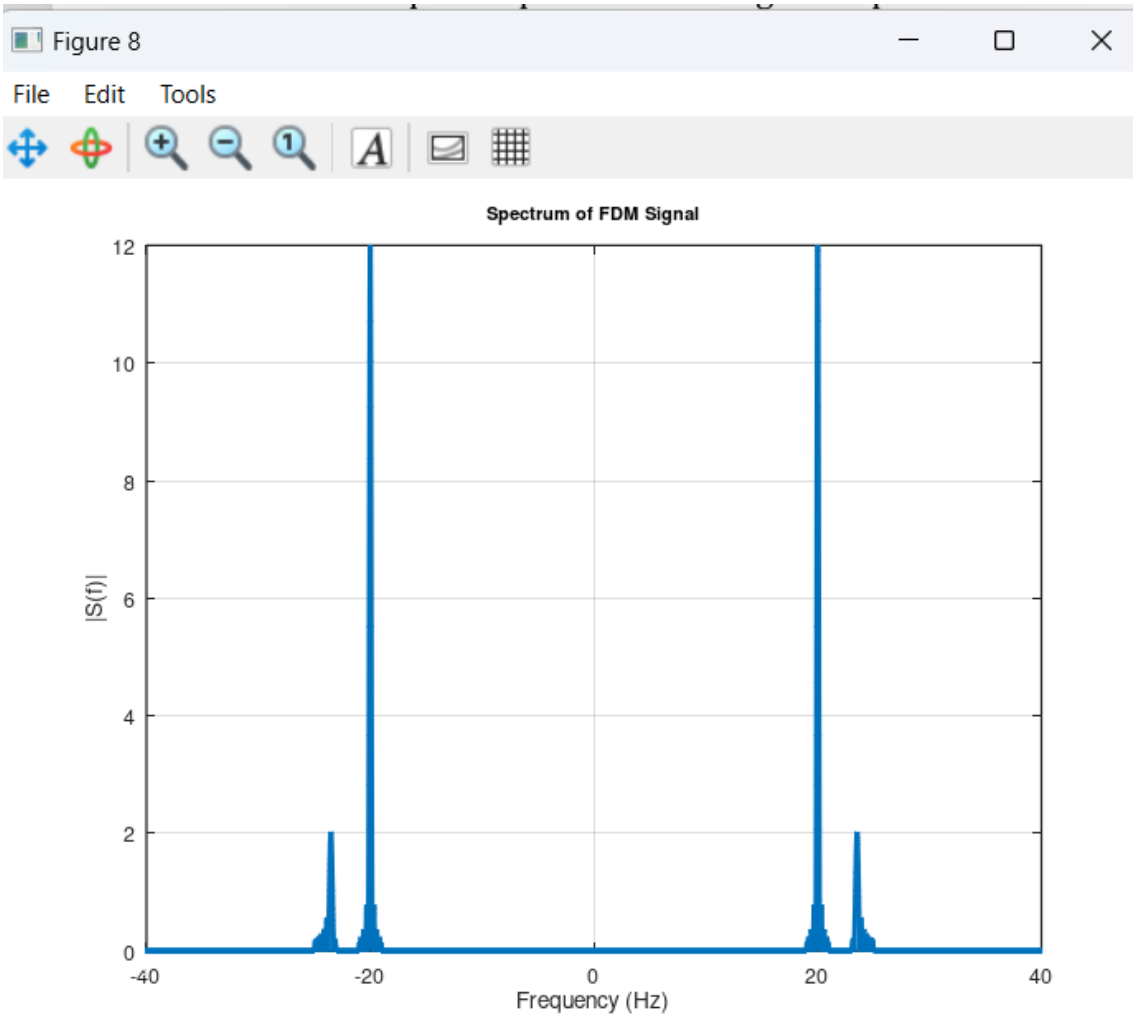
- Output:



- Code:

```
>> S = fftshift(fft (s)) *ts;
>> figure;
>> plot(f, abs(S), 'LineWidth',1.2);
>> xlabel('Frequency (Hz)');
>> ylabel('|S(f)|');
>> title('Spectrum of FDM Signal');
>> xlim([-40,40]);
>> grid on;
```

- Output:

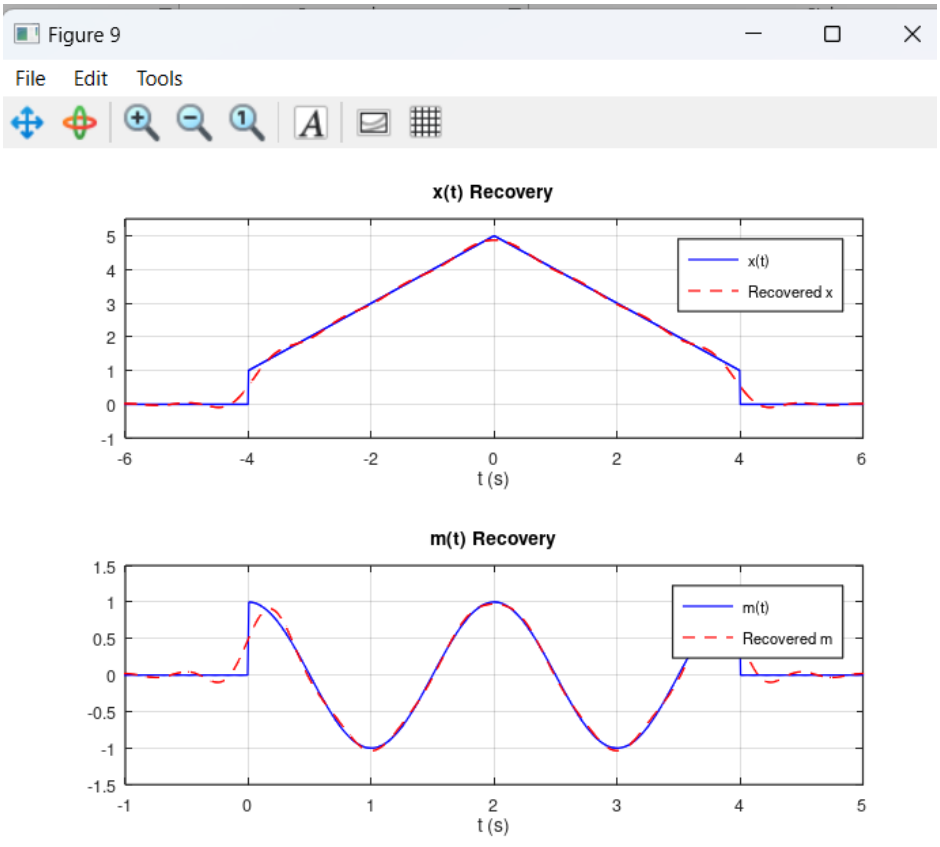


12. Create a coherent demodulator for each channel and plot the received messages and the input messages on the same figure.

- Code:

```
>> %% 12. coherent demodulation
>> xr_unfiltered = 2 * s .* cos(2*pi*fc1*t);
>> Xr_unfiltered = fftshift(fft (xr_unfiltered)) *ts;
>> H_x = double(abs(f) <= (2));
>> Xr = H_x .* Xr_unfiltered;
>> x_rec = real(ifft(ifftshift(Xr) /ts));
>> mr_unfiltered = 2 * s .* cos(2*pi*fc2*t);
>> Mr_unfiltered = fftshift(fft (mr_unfiltered)) *ts;
>> H_m = double(abs(f) <= (2));
>> Mr = H_m .* Mr_unfiltered;
>> m_rec_full = real(ifft(ifftshift(Mr) /ts));
>> m_rec = m_rec_full(1:length(t));
>> figure;
>> subplot(2,1,1);
>> plot(t, x, 'b', t, x_rec, 'r--');
>> legend('x(t)','Recovered x');
>> xlabel('t (s)');
>> xlim([-6 6]);
>> ylim([-1 5.5]);
>> title('x(t) Recovery');
>> grid on;
>> subplot(2,1,2);
>> plot(t, m, 'b', t, m_rec, 'r--');
>> legend('m(t)','Recovered m');
>> xlabel('t (s)');
>> xlim([-1 5]);
>> ylim([-1.5 1.5]);
>> title('m(t) Recovery'); grid on;
```

- Output:



Digital

- ## Project Description:

  This project involves implementing and analyzing digital communication techniques, including line coding and digital modulation schemes (ASK, FSK, PSK) using Octave. The goal is to compare different encoding methods, simulate transmitter-receiver systems, and evaluate performance under varying conditions.

- ## Project Components:

Part I: Line Coding Comparison

1. Generate Random Bit Stream

Create a random 64-bit sequence (0s and 1s).

2. Implement Two Line Codes

Choose one from (AMI, CMI, Manchester) and one from (Unipolar NRZ, Polar NRZ).

3. Time & Frequency Domain Analysis

Plot waveforms of both encoding schemes.
Compute and compare their power spectral densities (PSD).

4. Advantages & Disadvantages

Discuss bandwidth efficiency, DC component, synchronization, and noise immunity.

Part II: Digital Modulation & Demodulation

1. Modulation Scheme (ASK/FSK/PSK)

Modulate the random bit stream using a carrier frequency > bit rate.
Plot time-domain and frequency-domain signals.

2. Coherent Receiver with Phase Errors

Demodulate the signal with oscillator phase offsets:30°, 60°, 90°.
Analyze the impact on signal recovery (BER, distortion).

3. Performance Comparison

Compare output waveforms at different phases.
Discuss robustness against phase misalignment.

Project:

- ## Digital:

  ### Part I

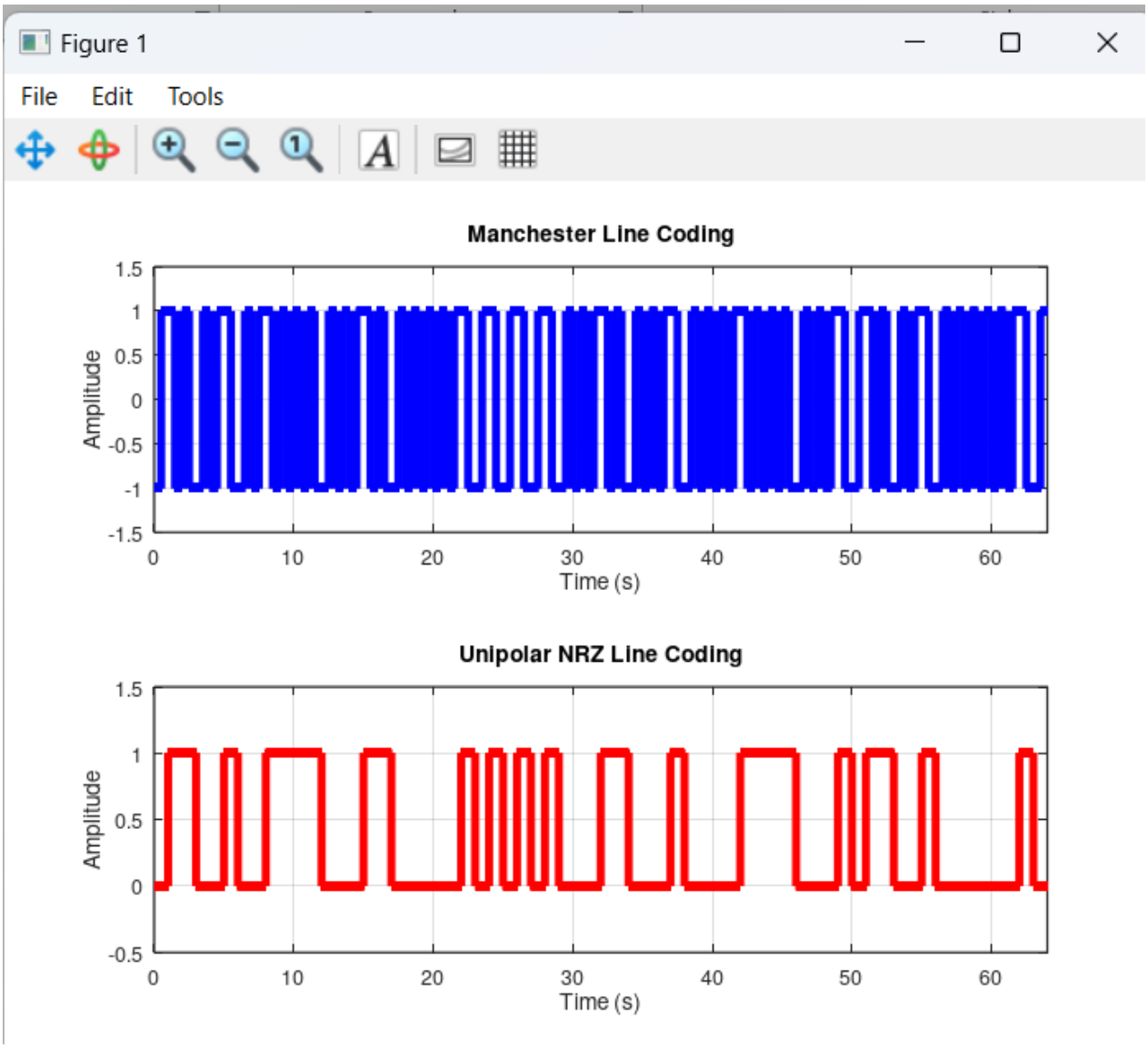
  Using Octave simulator, you have to develop a code for line coding. Each group will choose two types of line codes and make a comparison between them. Each group should select one of these line codes: AMI, CMI, and Manchester to be compared with one of these codes: unipolar non return to zero, and polar non-return to zero. Each group must plot the time and spectral domains of the pulses. The students should select the number of bits to be at least 64. This bit stream should be selected to be random, which means that the type of each bit is randomly selected by the program code to be either '1' or '0'. The report must have the program code. The report must include explanations of the temporal and spectral plots of the two-line codes and the advantages of one on the other.

  - Code:

```
Command Window
>> n = 64;
>> fs = 1000;
>> ts = 1/fs;
>> bitrate = 1;
>> Tb = 1/bitrate;
>> N = ceil((Tb*n)/ts);
>> samples_per_bit = fs / bitrate;
>> t = 0 : ts : (N-1)*ts;
>> bits = randi([0 1], 1, n);
>> disp('Random bitstream:');
Random bitstream:
>> disp(bits);
 Columns 1 through 23:

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

 Columns 24 through 46:

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

 Columns 47 through 64:

   0   0   0   1   0   1   1   0   0   1   0   0   0   0   0   0   1   0
>> manchester = zeros(size(t));
>> unipolar = zeros(size(t));
>> for i = 1:n
    idx = (i-1)*samples_per_bit + 1 : i*samples_per_bit;
    half = floor(samples_per_bit / 2);
    % manchester
    if bits(i) == 1
        manchester(idx(1:half)) = 1;
        manchester(idx(half+1:end)) = -1;
    else
        manchester(idx(1:half)) = -1;
        manchester(idx(half+1:end)) = 1;
    end
    % unipolar NRZ
    if bits(i) == 1
        unipolar(idx) = 1;
    else
        unipolar(idx) = 0;
    end
end
```

```
>> %% plot manchester
>> figure;
>> subplot(2,1,1);
>> plot(t, manchester, 'b', 'LineWidth', 2);
>> title('Manchester Line Coding');
>> xlabel('Time (s)'); ylabel('Amplitude');
>> xlim([0 64]);
>> ylim([-1.5 1.5]);
>> grid on;
>> %% plot unipolar NRZ
>> subplot(2,1,2);
>> plot(t, unipolar, 'r', 'LineWidth', 2);
>> title('Unipolar NRZ Line Coding');
>> xlabel('Time (s)'); ylabel('Amplitude');
>> xlim([0 64]);
>> ylim([-0.5 1.5]);
>> grid on;
```

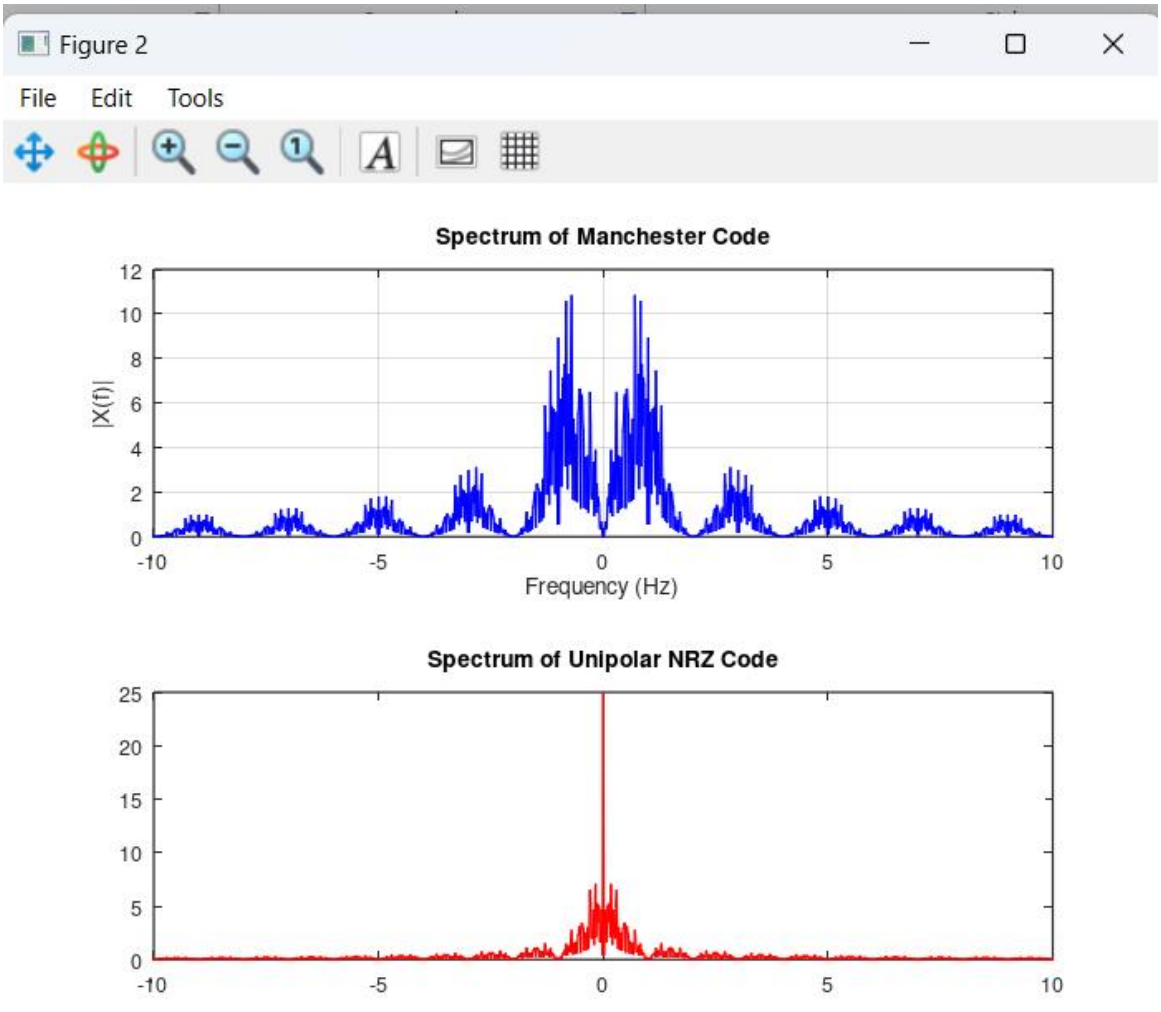
- Output:



- Code:

```
>> %% frequency domain
>> df = bitrate/n;
>> if(rem(N,2)==0) %% Even
  f = - (0.5*fs) : df : (0.5*fs-df) ;
else %% Odd
  f = - (0.5*fs-0.5*df) : df : (0.5*fs-0.5*df) ;
end
>> M = abs(fftshift(fft(manchester))* ts);
>> U = abs(fftshift(fft(unipolar))* ts);
>> figure;
>> subplot(2,1,1);
>> plot(f, M, 'b');
>> xlim([-10 10]);
>> title('Spectrum of Manchester Code');
>> xlabel('Frequency (Hz)'); ylabel('|X(f)|'); grid on;
>> subplot(2,1,2);
>> plot(f, U, 'r');
>> xlim([-10 10]);
>> title('Spectrum of Unipolar NRZ Code');
>> xlabel('Frequency (Hz)'); ylabel('|X(f)|'); grid on;
```

- Output:



Time Domain Analysis

Unipolar NRZ:

- Represents '1' as a high voltage level and '0' as zero voltage
- No transitions between bits of the same value
- DC component present due to the unipolar nature
- Long sequences of 1s or 0s can cause baseline wandering

Manchester:

- Each bit is represented by a transition in the middle of the bit period
- '1' is represented by a high-to-low transition
- '0' is represented by a low-to-high transition
- Always has a transition in the middle of each bit period, which aids in clock recovery
- No DC component as the signal is balanced

Frequency Domain Analysis

Unipolar NRZ:

- The spectrum has a main lobe that extends from DC to the bit rate
- Significant DC component (visible as energy at 0 Hz)
- Nulls occur at integer multiples of the bit rate
- More energy concentrated at lower frequencies

Manchester:

- The spectrum has a main lobe that extends from DC to twice the bit rate
- No DC component (no energy at 0 Hz)
- Nulls occur at the bit rate
- Energy is more spread out in frequency compared to Unipolar NRZ

Advantages Comparison

Manchester Advantages:

1. Self-clocking: The guaranteed transitions in each bit period make clock recovery easier
2. No DC component: Can be used in systems that cannot tolerate DC components
3. Better synchronization: More reliable for long sequences of identical bits
4. Error detection: Violations of the encoding rules can be detected

Unipolar NRZ Advantages:

1. Simplicity: Easier to implement and understand
2. Bandwidth efficiency: Requires less bandwidth than Manchester (half the bandwidth)
3. Lower power: Only transitions between 0 and 1, not between +1 and -1

Conclusion

Manchester coding provides better synchronization and no DC component at the cost of higher bandwidth requirements. Unipolar NRZ is simpler and more bandwidth-efficient but suffers from DC component and synchronization issues. The choice between them depends on the specific application requirements regarding synchronization needs, bandwidth constraints, and DC component tolerance.

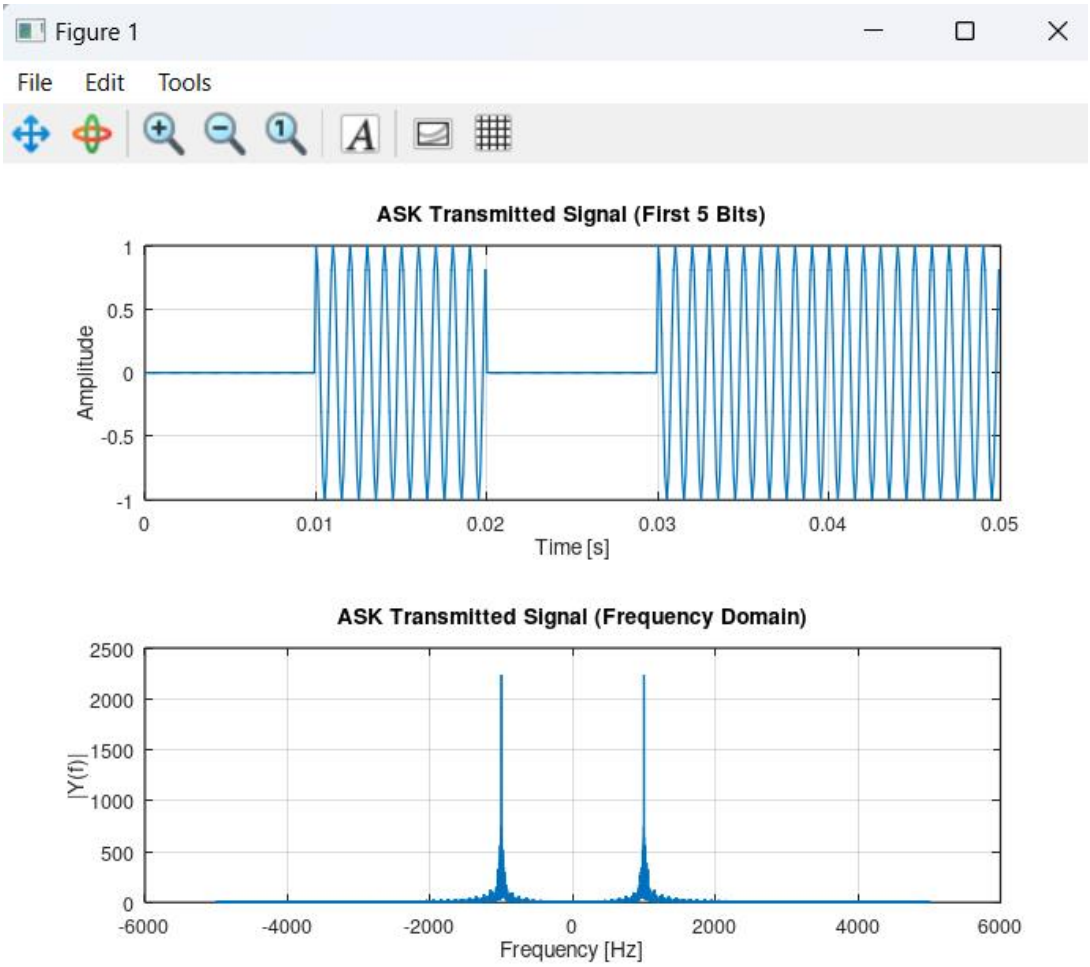
Part II

Each group develops a program code for the transmitter and coherent receiver of either of ASK, FSK and PSK systems. The baseband data can be the random data explained above. The carrier frequency should be selected to be higher than the bit rate. The student must plot the temporal and spectrum of the signal at outputs of the transmitter and the receiver. In the receiver, each group should select the oscillator phase to be 30o, 60o, and 90o, and comment of the results each group will get.

- Code:

```
Command Window
>> N = 100;
>> Rb = 100;
>> Tb = 1/Rb;
>> fs = 10000;
>> fc = 1000;
>> samples_per_bit = fs * Tb;
>> %% Generate Random Binary Data
>> data = randi([0 1], 1, N);
>> data_upsampled = repelem(data, samples_per_bit);
>> t = (0:length(data_upsampled)-1) / fs;
>> %% ASK Modulation
>> carrier = cos(2*pi*fc*t);
>> ask_signal = data_upsampled .* carrier;
>> %% Plot Transmitted Signal (Zoomed Time + Frequency)
>> num_bits_to_plot = 5;
>> samples_to_plot = samples_per_bit * num_bits_to_plot;
>> figure;
>> subplot(2,1,1);
>> plot(t(1:samples_to_plot), ask_signal(1:samples_to_plot));
>> title(['ASK Transmitted Signal (First ', num2str(num_bits_to_plot), ' Bits)']);
>> xlabel('Time [s]'); ylabel('Amplitude'); grid on;
>> % Frequency Domain
>> Y = fftshift(fft(ask_signal, 2^nextpow2(length(ask_signal))));
>> f = linspace(-fs/2, fs/2, length(Y));
>> subplot(2,1,2);
>> plot(f, abs(Y));
>> title('ASK Transmitted Signal (Frequency Domain)');
>> xlabel('Frequency [Hz]'); ylabel('|Y(f)|'); grid on;
```

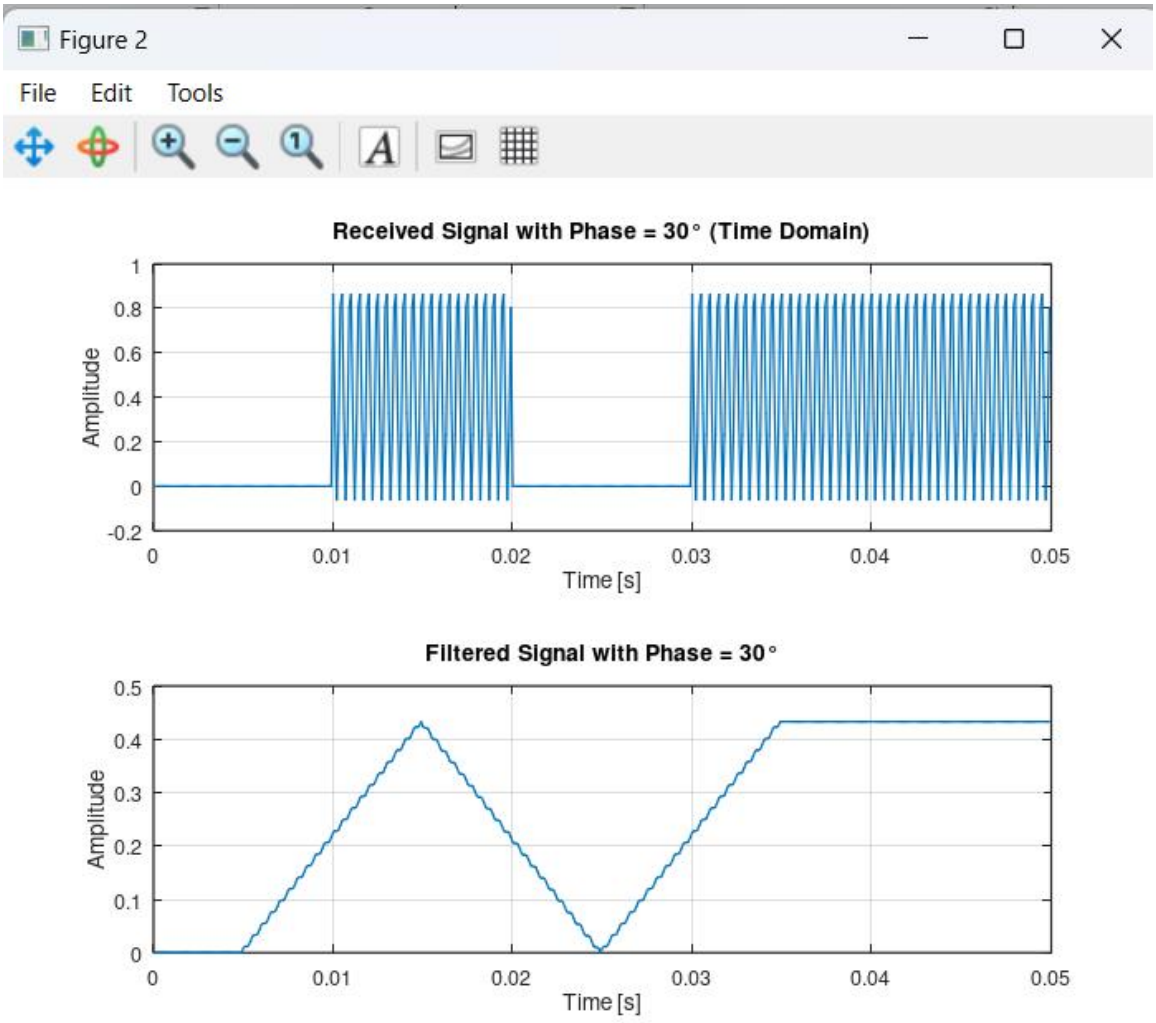
- Output:



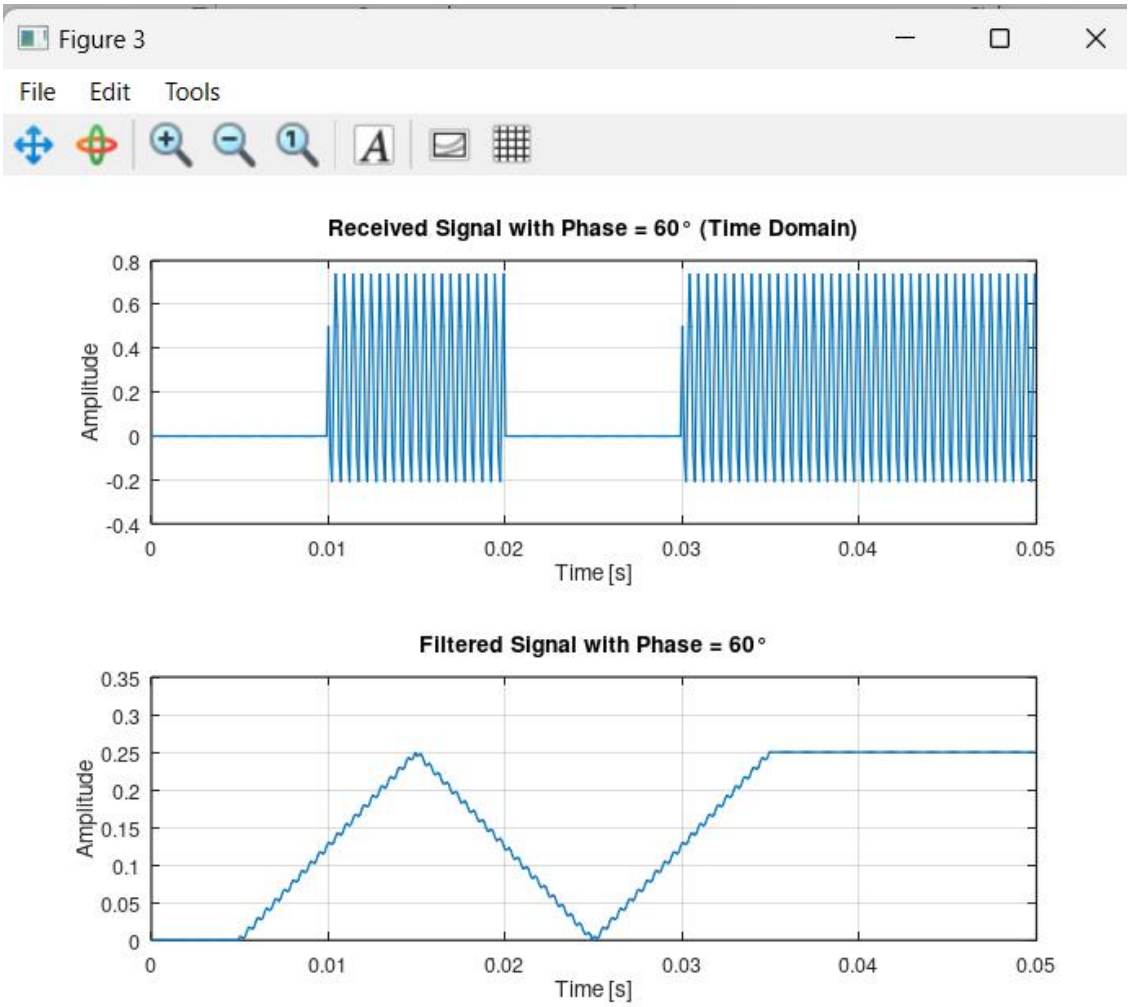
- Code:

```
>> %% Coherent Demodulation with Phase Offsets
>> phases_deg = [30, 60, 90];
>> for i = 1:length(phases_deg)
    phase_rad = deg2rad(phases_deg(i));
    osc = cos(2*pi*fc*t + phase_rad);
    received = ask_signal .* osc;
    % Low-pass filter using moving average
    lpf = ones(1, round(samples_per_bit)) / samples_per_bit;
    filtered = conv(received, lpf, 'same');
    % Downsample and make decisions
    sampled = filtered(round(samples_per_bit/2):samples_per_bit:end);
    recovered_bits = sampled > 0.25;
    % Bit error analysis
    num_errors = sum(recovered_bits ~= data);
    disp(['Phase = ', num2str(phases_deg(i)), '°: Bit Errors = ', num2str(num_errors), ' / ', num2str(N)]);
    % Plot Received and Filtered Signal (Zoomed)
    figure;
    subplot(2,1,1);
    plot(t(1:samples_to_plot), received(1:samples_to_plot));
    title(['Received Signal with Phase = ', num2str(phases_deg(i)), '° (Time Domain)']);
    xlabel('Time [s]'); ylabel('Amplitude'); grid on;
    subplot(2,1,2);
    plot(t(1:samples_to_plot), filtered(1:samples_to_plot));
    title(['Filtered Signal with Phase = ', num2str(phases_deg(i)), '°']);
    xlabel('Time [s]'); ylabel('Amplitude'); grid on;
end
Phase = 30°: Bit Errors = 0 / 100
Phase = 60°: Bit Errors = 16 / 100
Phase = 90°: Bit Errors = 50 / 100
```

- Output:(30 DEGREE)



- Output: (60 DEGREE)



- Output: (90 DEGREE)