Main Function:

Source Code:

```
clc; %terminal clear
clear all; %all variable clear from work place
close all; %clear all figure from previous open
disp('My Name is Abrar');
% define bit pattern
bits = [1 0 0 1 1 0 1];

bitrate = 2; %bit per second

figure; %open a new figure

% call the function
[t,s]=unipolarnrz(bits,bitrate); % t-> time vector s-> signal amplitude vector

plot(t,s,'linewidth',3);
xlabel('Time in second', 'fontsize',14,'fontWeight','bold'); %x okkho name
ylabel('Amplitude in Volts', 'fontsize',14,'fontWeight','bold'); %y okkho name
title(['Unipolar NRZ: ' num2str(bits)]);
```

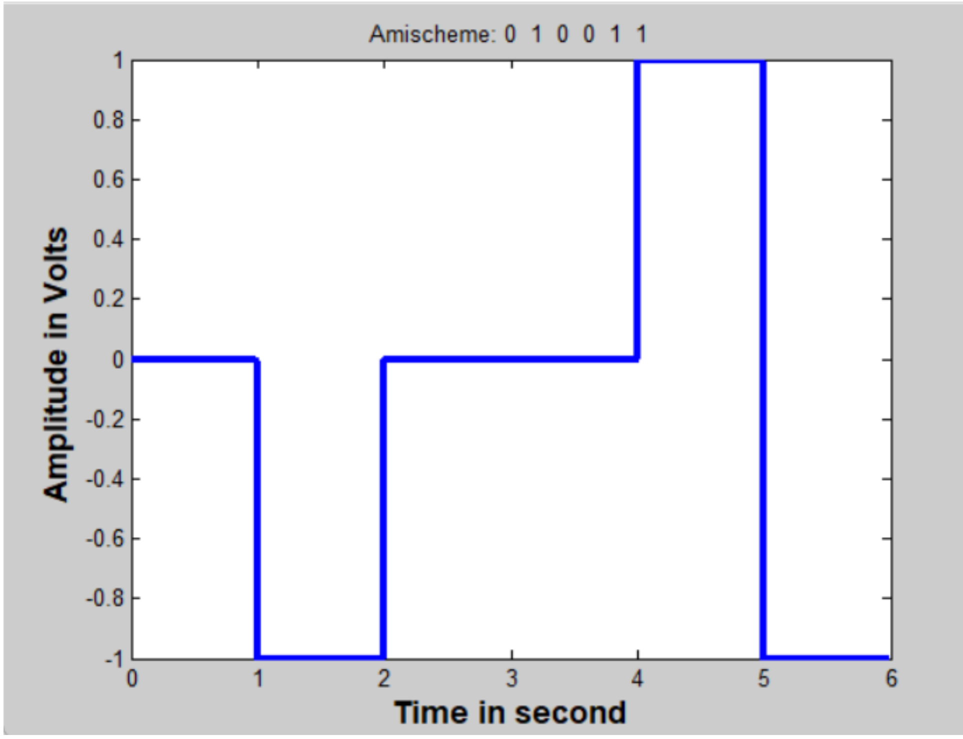
Problem 1: Implementation of AMI Scheme

Source Code:

```
function [t, x] = amischeme(bits, bitrate)
    n = 100; % Samples per bit (increase for smoother waveform)
    T = length(bits) / bitrate; % Total time duration
    N = n * length(bits); % Total number of samples
    dt = T / N; % Time step
    t = 0:dt:(T - dt); % Time vector
    x = zeros(1, length(t)); % Initialize signal
    last = 1;

    for i = 0:length(bits)-1
        bit = bits(i + 1);
        if bit == 1
            last = -last;
            x(i * n + 1:(i + 1) * n) = last;
        else
            x(i * n + 1:(i + 1) * n) = 0;
        end
    end
```

Output:



Problem 2: Implementation of Manchester Scheme

Source Code:

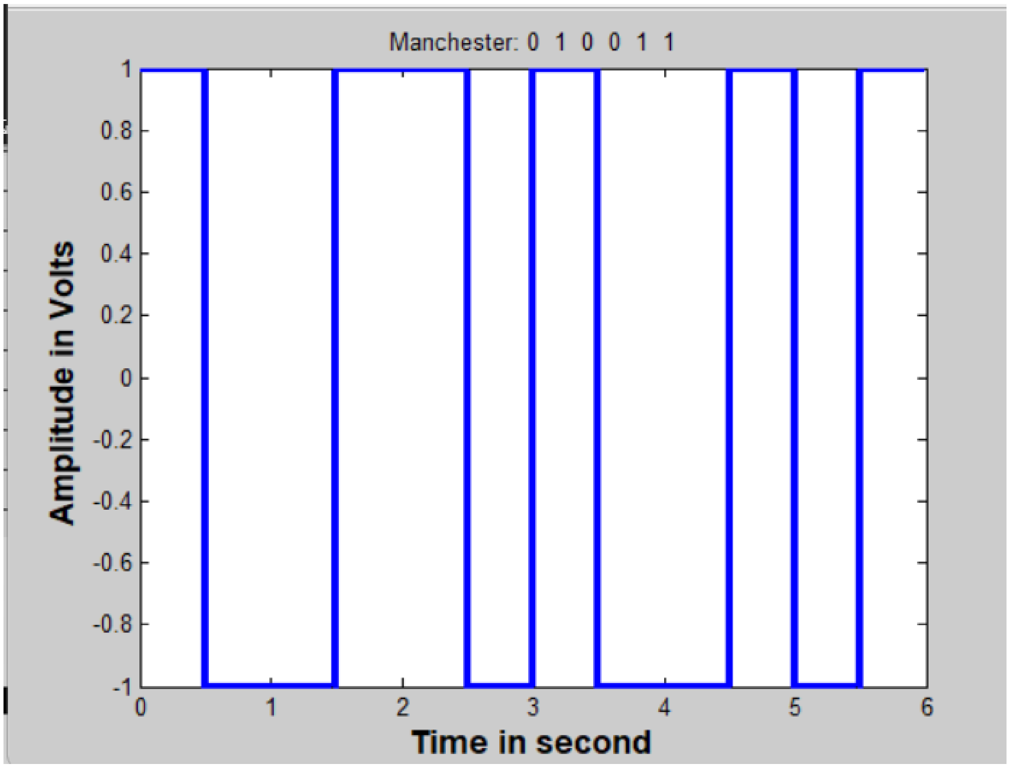
Editor - D:\Academic\2nd year\2nd semester\Data Communication Lab\Lab 2\manchester.m

main.m | diffmanchester.m | manchester.m | polarrz.m | sudoternary.m

```
function [t, x] = manchester(bits, bitrate)
    n = 100; % Samples per bit
    T = length(bits) / bitrate; % Total time duration
    N = n * length(bits); % Total number of samples
    dt = T / N; % Time step
    t = 0:dt:(T - dt); % Time vector
    x = zeros(1, length(t)); % Initialize signal

    for i = 0:length(bits)-1
        bit = bits(i + 1);
        if bit == 1
            x(i * n + 1:i * n + 1 + floor(n/2)-1) = -1;
            x(i * n + 1 + floor(n/2):i * n + 1 + n - 1) = 1;
        else
            x(i * n + 1:i * n + 1 + floor(n/2)-1) = 1;
            x(i * n + 1 + floor(n/2):i * n + 1 + n - 1) = -1;
        end
    end
```

Output:



Problem 3: Implementation of Different Manchester Scheme

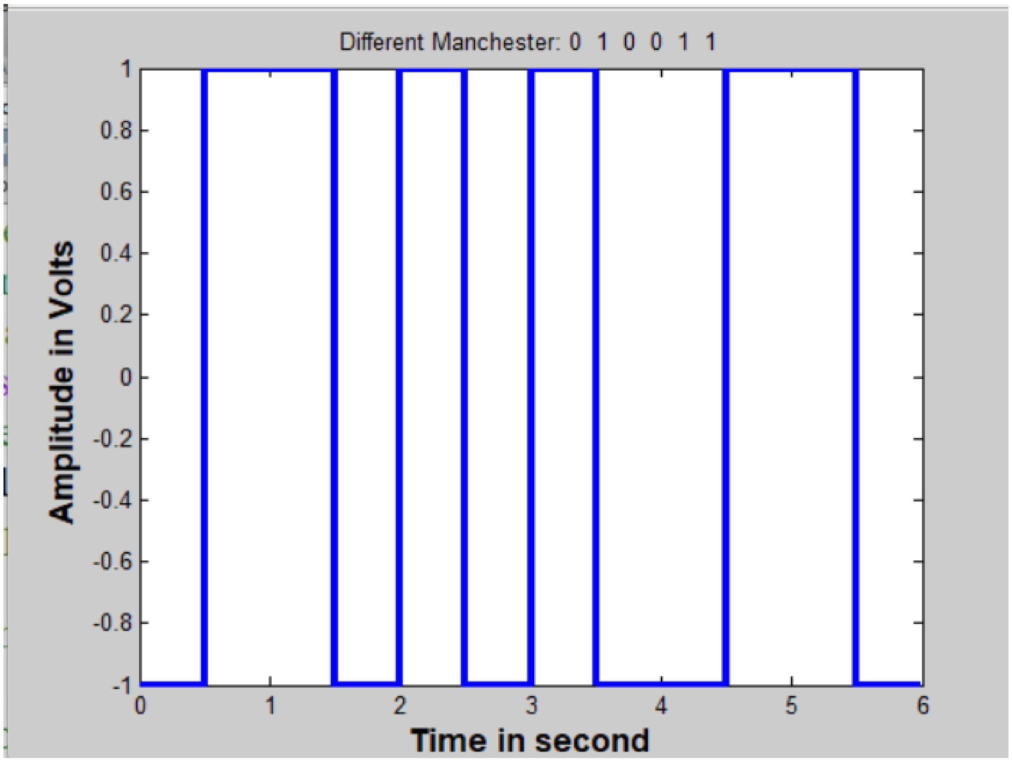
Source Code:

Editor - D:\Academic\2nd year\2nd semester\Data Communication Lab\Lab 2\diffmanchester.m

main.m | diffmanchester.m | polarrz.m | sudoternary.m | +

```
function [t, x] = diffmanchester(bits, bitrate)
    n = 100; % Samples per bit
    T = length(bits) / bitrate; % Total time duration
    N = n * length(bits); % Total number of samples
    dt = T / N; % Time step
    t = 0:dt:(T - dt); % Time vector
    x = zeros(1, length(t)); % Initialize signal
    last = 1;
    for i = 0:length(bits)-1
        bit = bits(i + 1);
        if bit == 0
            last = -last;
            x(i * n + 1:i * n + 1 + floor(n/2)-1) = last;
            x(i * n + 1 + floor(n/2):i * n + 1 + n-1) = -last;
            last = -last;
        else
            x(i * n + 1:i * n + 1 + floor(n/2)-1) = last;
            x(i * n + 1 + floor(n/2):i * n + 1 + n-1) = -last;
            last = -last;
        end
```

Output:



Problem 4: Implementation of Polar Return-to-Zero Scheme

Source Code:

Editor - D:\Academic\2nd year\2nd semester\Data Communication Lab\Lab 2\polarrz.m

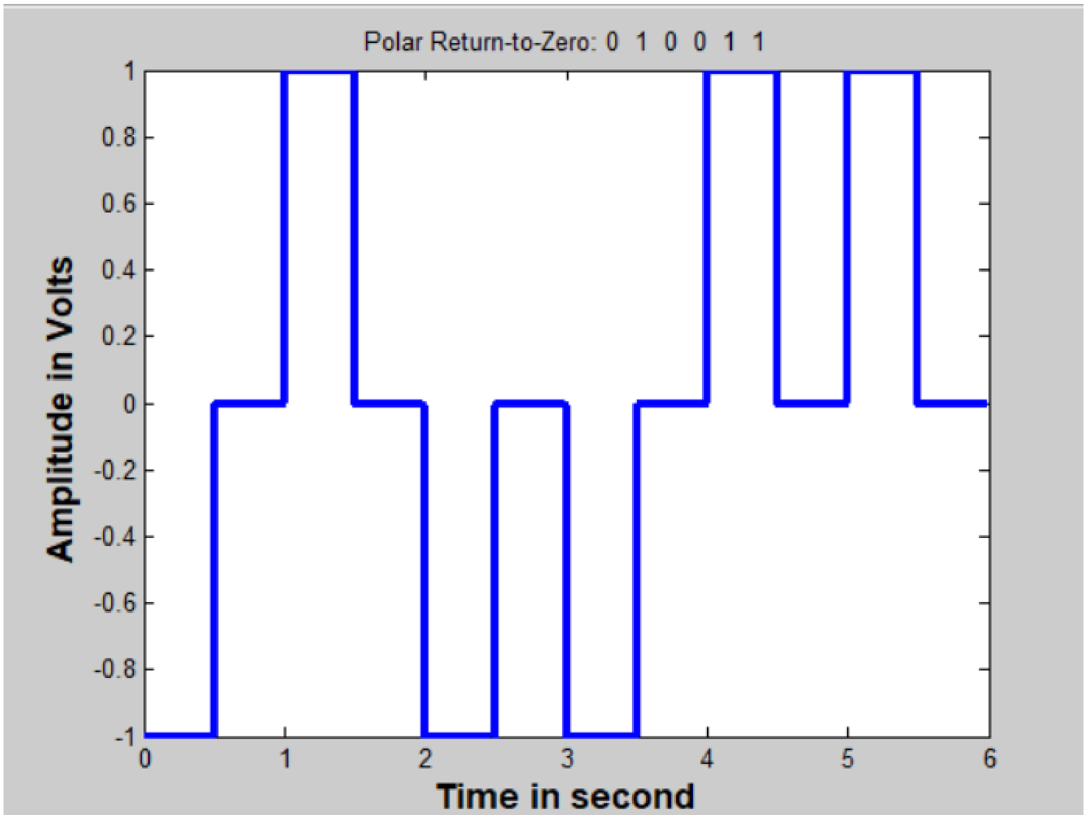
main.m | polarrz.m | sudoternary.m

```
function [t, x] = polarrz(bits, bitrate)
    n = 100; % Samples per bit (increase for smoother waveform)
    T = length(bits) / bitrate; % Total time duration
    N = n * length(bits); % Total number of samples
    dt = T / N; % Time step
    t = 0:dt:(T - dt); % Time vector
    x = zeros(1, length(t)); % Initialize signal

    for i = 0:length(bits)-1
        bit = bits(i + 1);

        if bit == 1
            x(i*n+1:i*n+floor(0.5*n)) =1 ;
        else
            x(i*n+1:i*n+floor(0.5*n)) =-1 ;
        end
    end
end
```

Output:



Problem 5: Implementation of Pseudo Ternary Scheme

Source Code:

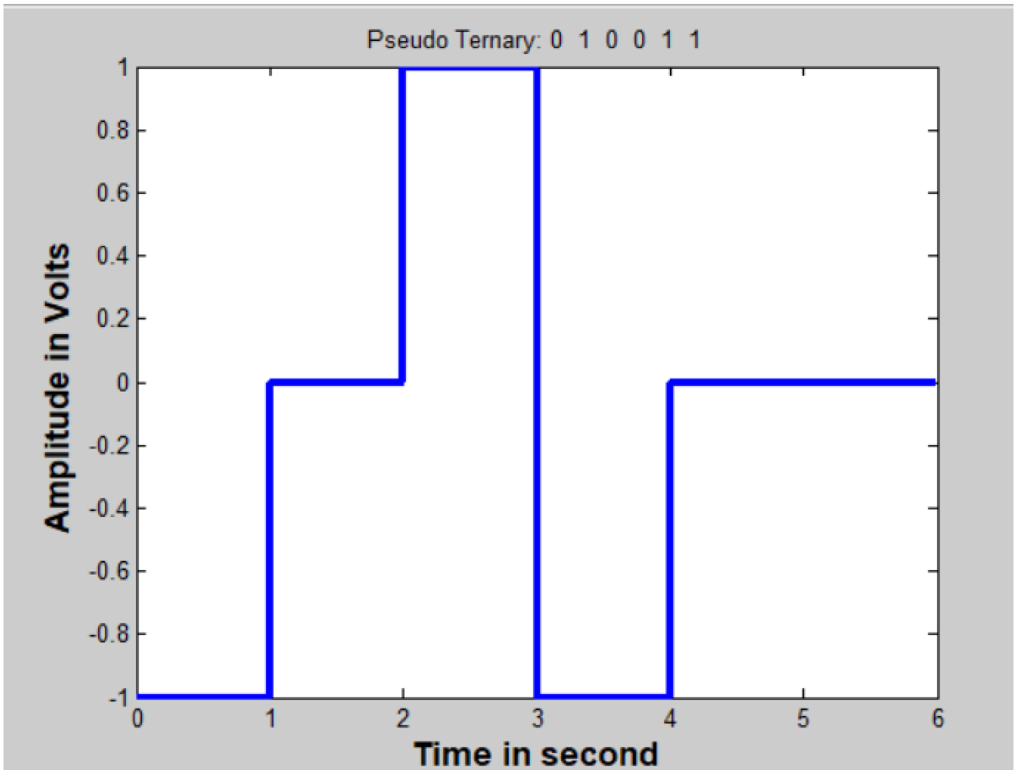
Editor - D:\Academic\2nd year\2nd semester\Data Communication Lab\Lab 2\sudoternary.m*

main.m | sudoternary.m* | +

```
function [t, x] = sudoternary(bits, bitrate)
    n = 100; % Samples per bit (increase for smoother waveform)
    T = length(bits) / bitrate; % Total time duration
    N = n * length(bits); % Total number of samples
    dt = T / N; % Time step
    t = 0:dt:(T - dt); % Time vector
    x = zeros(1, length(t)); % Initialize signal
    last = 1;

    for i = 0:length(bits)-1
        bit = bits(i + 1);
        if bit == 1
            x(i * n + 1:(i + 1) * n) = 0;
        else
            last = -last;
            x(i * n + 1:(i + 1) * n) = last;
        end
    end
end
```

Output:



Pseudo Ternary: 0 1 0 0 1 1
Amplitude in Volts
Time in second
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
1
2
3
4
5
6