**Experiment 1 (PLOTTING)**

% Create a vector 'x' ranging from 0 to 10 with a step size of 0.1

x = 0:0.1:10;

% Calculate the sine values of each element in vector 'x' and store it in 'y'

y = sin(x);

% Calculate the cosine values of each element in vector 'x' and store it in 'z'

z = cos(x);

% Create a 3-row, 1-column subplot layout and select the first subplot

subplot(3,1,1);

% Plot the sine function using the values in vectors 'x' and 'y'

plot(x, y);

% Add a grid to the current subplot

grid on;

% Select the second subplot for subsequent plotting

subplot(3,1,2);

% Plot the cosine function using the values in vectors 'x' and 'z'

plot(x, z);

% Add a grid to the current subplot

grid on;

% Keep the current plot active, allowing for additional plots

hold on;

% Select the third subplot for subsequent plotting

subplot(3,1,3);

% Create a stem plot of the cosine function in the third subplot

stem(x, z);

% Add a grid to the current subplot

grid on;

% Keep the current plot active

hold on;

% Add a red stem plot of the sine function to the third subplot

stem(x, y, 'r');

**Experiment 2 (Generating a Signal)**

% Generation of sinusoidal signals: x(t) = 2sin(2πt - π/2)

% Define the time vector 't' from -5 to 5 with a step size of 0.01

t = -5:0.01:5;

% Generate the sinusoidal sequence x(t) = 2sin(2πt - π/2)

x = 2 \* sin(2 \* pi \* t - pi/2);

% Plot the sinusoidal sequence

plot(t, x);

% Add a grid to the plot

grid on;

% Set the axis limits for better visualization

axis([-6 6 -3 3]);

% Label the y-axis as 'x(t)'

ylabel('x(t)');

% Label the x-axis as 'Time (sec)'

xlabel('Time (sec)');

% Add a title to the plot

title('Figure 2.1');

**Experiment 3 (Generation of discrete time signals)**

% Generation of discrete time signals

% Define the discrete time vector 'n' from -5 to 5

n = -5:5;

% Define the discrete signal sequence 'x'

x = [0 0 1 1 -1 0 2 -2 3 0 -1];

% Plot the discrete signal using stems

stem(n, x);

% Set the axis limits for better visualization

axis([-6 6 -3 3]);

% Label the x-axis as 'n'

xlabel('n');

% Label the y-axis as 'x[n]'

ylabel('x[n]');

% Add a title to the plot

title('Figure 2.2');

**Experiment 4 (Generation of random sequences)**

% Generation of random sequences

% Define the time vector 'n' from 0 to 10

n = 0:10;

% Generate a random sequence 'x' with values between 0 and 1

x = rand(1, length(n));

% Generate a random sequence 'y' with values from a standard normal distribution

y = randn(1, length(n));

% Plot the sequence 'x'

plot(n, x);

grid on;

% Keep the current plot active to add another plot

hold on;

% Plot the sequence 'y' in red ('r')

plot(n, y, 'r');

% Label the y-axis as 'x & y'

ylabel('x & y');

% Label the x-axis as 'n'

xlabel('n');

% Add a title to the plot

title('Figure 2.3');

**Experiment 5 (Generating a discrete periodic signal Signal)**

% Given sequence

n = 0:4;

x = [1 1 2 -1 0];

% Plot the original sequence x(n) in the first subplot

subplot(2,1,1);

stem(n, x);

grid on;

axis([0 4 -1 2]); % Set axis limits

xlabel('n');

ylabel('x(n)');

title('Figure 2.4(a)');

% Create a periodic extension of x(n) in the second subplot

xtilde = [x, x, x]; % Create a periodic extension

length\_xtilde = length(xtilde);

n\_new = 0:length\_xtilde-1;

% Plot the periodic sequence xtilde(n) in the second subplot in red

subplot(2,1,2);

stem(n\_new, xtilde, 'r');

grid on;

xlabel('n');

ylabel('periodic x(n)');

title('Figure 2.4(b)');

**Experiment 6 (Generating Square wave) using loop**

% Clear command window and workspace

clear;

clc;

% Input the value of odd 'n' from the user

n = input('Insert the value of odd n: ');

% Create a time vector 't' from 0 to 1 with a step size of 0.001

t = 0:0.001:1;

% Initialize the variable 'sum' to store the cumulative sum

sum = 0;

% Generate the signal by summing sine waves with odd harmonics

for f = 1:2:n

w = sin(2 \* pi \* f \* t);

sum = sum + w;

end

% Plot the generated signal in a single subplot

subplot(1,1,1);

% Plot the cumulative sum of sine waves

plot(t, sum);

% Add a grid to the plot

grid on;

% Provide a title for the plot

title('Sum of Sine Waves with Odd Harmonics');

% Label the x-axis as 'Time'

xlabel('Time');

% Label the y-axis as 'Amplitude'

ylabel('Amplitude');

**Experiment 7 (Generating Unit Step Discrete Time Signal)**

% Clear the command window, current variables, and close all open figures

clc;

clear all;

close all;

% Prompt the user to enter the range 'N'

N = input('Enter the range: ');

% Define the time vector 'n' from -N to N

n = -N:1:N;

% Create a unit step function 'y' (0 for n < 0, 1 for n >= 0)

y = [zeros(1, N), 1, ones(1, N)];

% Plot the unit step function using stems

stem(n, y);

% Set the axis limits for better visualization

axis([-(N+1) N+1 -0.5 1.5]); % [-x x -y y]

% Label the x-axis as 'Time'

xlabel('Time');

% Label the y-axis as 'Amplitude of Y'

ylabel('Amplitude of Y');

% Add a title to the plot

title('Generating Unit Step Function');

***Experiment 8 :*** Unit impulse signal

%Code of implementation of impulse signal in Matlab:

clc;

clear all;

m1=input('enter the value of x-axis in negative side:');

m2=input('enter the value of x-axis in positive side:');

n=m1:m2;

x=(n==0);%it works as if statement like n=-5:5( 0 0 0 0 0 1 0 0 0 0 0 0)

stem(n,x);

xlabel('n');

ylabel('amplitude');

title('Unit impulse signal');

or,  
  
clc;

clear all;

close all;

m1=input('enter the value of x-axis in negative side:');

m2=input('enter the value of x-axis in positive side:');

n=-m1:m2;

d=[zeros(1,m1) 1 zeros(1,m2)];

stem(n,d);

xlabel('n');

ylabel('amplitude');

title('unit impulse signal');

**Experiment 9** Generating and plotting ramp discrete time signal.

% Close all open figures and clear the command window

close all;

clc;

% Input the lower limit 'n1' from the user

n1 = input('Enter lower limit: ');

% Input the upper limit 'n2' from the user

n2 = input('Enter upper limit: ');

% Create a vector 'n' from 'n1' to 'n2' with a step of 1

n = n1:1:n2;

% Generate a ramp function 'x' (n for n >= 0, 0 for n < 0)

x = n .\* (n >= 0);

% Plot the ramp function using stems in blue ('b')

stem(n, x, 'b');

% Set the axis limits for better visualization

axis([n1-1 n2+1 -1 n2+1]); % [-x, x, -y, y]

% Label the x-axis as 'time'

xlabel('time');

% Label the y-axis as 'Amplitude of Y'

ylabel('Amplitude of Y');

% Add a title to the plot

title('Ramp Function');

**Experiment 10 (Time reversal using a discrete sinusoidal function [use of fliplr( ) and values of x-axis(angle) in radian)**

% Close all open figures and clear the command window

close all;

clc;

% Define the time vector t1 from 0 to 2\*pi with a step of 0.2

t1 = 0:0.2:2\*pi;

% Generate the original signal x1 using sine function

x1 = sin(t1);

% Generate the time-reversed signal x2 by flipping x1 and negating the time values

x2 = fliplr(x1);

t2 = -fliplr(t1);

% Plot the original signal in the first subplot

subplot(2,1,1);

stem(t1, x1, 'LineWidth', 2);

xlim([-10 10]); % Set x-axis limits

title('\bf\fontsize{25}Original Signal');

xlabel('\bf\fontsize{20}Samples');

ylabel('\bf\fontsize{20}Amplitude');

grid on;

ax = gca;

ax.XAxis.FontSize = 15;

ax.XAxis.FontWeight = 'bold';

ax.YAxis.FontSize = 15;

ax.YAxis.FontWeight = 'bold';

% Plot the time-reversed signal in the second subplot

subplot(2,1,2);

stem(t2, x2, 'LineWidth', 2);

xlim([-10 10]); % Set x-axis limits

title('\bf\fontsize{25}Time Reversed Signal');

xlabel('\bf\fontsize{20}Samples');

ylabel('\bf\fontsize{20}Amplitude');

grid on;

ax = gca;

ax.XAxis.FontSize = 15;

ax.XAxis.FontWeight = 'bold';

ax.YAxis.FontSize = 15;

ax.YAxis.FontWeight = 'bold';

**Experiment 11** Time reversal using a discrete sinusoidal function [use of fliplr()and values of x-axis(angle) in degree]

% Clear all existing figures and the command window

close all

clc

% Generate time values from 0 to 360 in steps of 10 degrees

t1 = 0:10:360;

% Calculate the sine values for the original signal

x1 = sind(t1);

% Flip the original signal horizontally to create a reversed version

x2 = fliplr(x1);

% Flip and negate the time values for the reversed signal

t2 = -fliplr(t1);

% Plot the original signal in the top subplot

subplot(211)

stem(t1, x1, 'LineWidth', 2)

xlim([-400 400])

ylim([-1.5 1.5])

title('\bf\fontsize{25}Original Signal')

xlabel('\bf\fontsize{20}Samples')

ylabel('\bf\fontsize{20}Amplitude')

grid on;

ax = gca;

ax.XAxis.FontSize = 15;

ax.XAxis.FontWeight = 'bold';

ax.YAxis.FontSize = 15;

ax.YAxis.FontWeight = 'bold';

% Plot the time-reversed signal in the bottom subplot

subplot(212)

stem(t2, x2, 'LineWidth', 2)

xlim([-400 400])

ylim([-1.5 1.5])

title('\bf\fontsize{25}Time Reversed Signal')

xlabel('\bf\fontsize{20}Samples')

ylabel('\bf\fontsize{20}Amplitude')

grid on;

ax = gca;

ax.XAxis.FontSize = 15;

ax.XAxis.FontWeight = 'bold';

ax.YAxis.FontSize = 15;

ax.YAxis.FontWeight = 'bold';

***Experiment 12 (Signal Addition)***

**addition.m ->**

clear all;

clc;

% Define the first set of values for x and y for the first signal

x1 = [-5 -4 -3 -2 -1 0];

y1 = [2 5 4 6 3 5];

% Define the second set of values for x and y for the second signal

x2 = [-2 -1 0 1 2];

y2 = [8 9 2 5 6];

% Plot the first signal in the first subplot

subplot(3,1,1);

stem(x1, y1);

grid on;

grid minor;

axis([-10 10 -8 8]);

% Plot the second signal in the second subplot

subplot(3,1,2);

stem(x2, y2);

grid on;

grid minor;

axis([-10 10 -8 8]);

% Find the range of x values that cover both signals

n = min(min(x1), min(x2)):1:max(max(x1), max(x2));

% Call the add\_function to add the two signals

[y] = add\_function(n, x1, x2, y1, y2);

% Plot the added signal in the third subplot

subplot(3,1,3);

stem(n, y);

grid on;

grid minor;

axis([-10 10 -8 8]);

**add\_function.m ->**

function [y] = add\_function(n, x1, x2, y1, y2)

% Initialize arrays to store the values of the two signals for addition

m1 = zeros(1, length(n));

m2 = zeros(1, length(n));

% Temporary variable to keep track of the position in y1 and y2 arrays

temp = 1;

% Loop to fill m1 with values from y1 based on the common x range

for i = 1:length(n)

if (n(i) >= min(x1) && n(i) <= max(x1))

m1(i) = y1(temp);

temp = temp + 1;

else

m1(i) = 0;

end

end

% Reset the temporary variable

temp = 1;

% Loop to fill m2 with values from y2 based on the common x range

for i = 1:length(n)

if (n(i) >= min(x2) && n(i) <= max(x2))

m2(i) = y2(temp);

temp = temp + 1;

else

m2(i) = 0;

end

end

% Add the two signals to get the final result

y = m1 + m2;

end

**Experiment 13 (Signal Multiplication)**

Multiplicaton.m ->

clc;

clear all;

close all;

% Define the first set of values for x and y for the first signal

x1 = [0:0.1:10];

y1 = sin(x1);

% Define the second set of values for x and y for the second signal

x2 = [-5:0.1:7];

y2 = 4 \* sin(x2);

% Plot the first signal in the first subplot

subplot(3,1,1);

stem(x1, y1);

grid on;

grid minor;

axis([-5 10 -5 5]);

% Plot the second signal in the second subplot

subplot(3,1,2);

stem(x2, y2);

grid on;

grid minor;

axis([-5 10 -5 5]);

% Find the new range of the signal

n = min(min(x1), min(x2)):0.1:max(max(x1), max(x2));

% Call the mul\_function to multiply the two signals

[m] = mul\_function(n, x1, y1, x2, y2);

% Plot the multiplied signal in the third subplot

subplot(3,1,3);

stem(n, m, 'r');

grid on;

grid minor;

axis([-5 10 -5 5]);

mul\_function.m ->

function [m] = mul\_function(n, x1, y1, x2, y2)

% Initialize arrays to store the values of the two signals for multiplication

m1 = zeros(1, length(n));

m2 = m1;

% Temporary variable to keep track of the position in y1 and y2 arrays

temp = 1;

% Loop to fill m1 with values from y1 based on the common x range

for i = 1:length(n)

if (n(i) >= min(x1) && n(i) <= max(x1))

m1(i) = y1(temp);

temp = temp + 1;

else

m1(i) = 0;

end

end

% Reset the temporary variable

temp = 1;

% Loop to fill m2 with values from y2 based on the common x range

for i = 1:length(n)

if (n(i) >= min(x2) && n(i) <= max(x2))

m2(i) = y2(temp);

temp = temp + 1;

else

m2(i) = 0;

end

end

% Multiply the two signals element-wise to get the final result

m = m1 .\* m2;

end

**Experiment 14 (Time Scaling)** y[n]=x[n/2]

% Clear all existing figures, variables, and the command window

close all;

clc;

% Prompt the user to enter the start and end values for the original signal

start\_value = input('Enter the start value: '); % e.g., -6

end\_value = input('Enter the end value: '); % e.g., 6

% Generate a time vector for the original signal

n1 = start\_value:end\_value;

% Define the original signal with proper separation between elements

y = [1, 0.5, 1, 0.5, 1, 0.5, 1, 0.5, 1, 0.5, 1, 0.5, 1];

% Initialize an index variable

index = 1;

% Create a compressed time vector (doubled in length)

n2 = (2 \* start\_value):(2 \* end\_value);

% Preallocate arrays for improved performance

x1 = zeros(size(n2));

y1 = zeros(size(n2));

% Loop to create the compressed signal

for i = 1:length(n2)

x1(i) = n2(i);

if (rem(n2(i), 2) == 0) % Check if the time index is even

y1(i) = y(index); % Assign the corresponding value from the original signal

index = index + 1; % Move to the next value in the original signal

else

y1(i) = 0; % Set odd-indexed values to zero in the compressed signal

end

end

% Plot the original signal in the first subplot

subplot(2, 1, 1);

stem(n1, y, 'r');

xlabel("Time");

ylabel("Amplitude");

grid on;

grid minor;

axis([(start\_value - 1) (end\_value + 1) -2 2]);

title("Original signal Y[n] = X[n]");

% Plot the compressed signal in the second subplot

subplot(2, 1, 2);

stem(x1, y1, 'b');

xlabel("Time");

ylabel("Amplitude");

grid on;

grid minor;

axis([(2 \* start\_value - 1) (2 \* end\_value + 1) -2 2]);

title("Compression signal Y[n] = X[n/2]");

**Experiment 14(ii) (Time Scaling)** y[n]=x[2n]

% Clear all existing figures and the command window

close all;

clc;

% Define the time domain range for the original signal

n1 = -4;

n2 = 9;

% Define the original signal

y = [0 0 0 1 1 1 1 1 0.5 0 0 0 0 0];

% Generate the time vector for the original signal

n = n1:n2;

% Specify the compression factor

value = 2;

% Initialize temporary variables

temp = 1;

% Create compressed signal vectors

for i = 1:length(n)

if (rem(n(i), value) == 0)

x1(temp) = n(i) / value;

y1(temp) = y(i);

temp = temp + 1;

end

end

% Plot the original signal in the first subplot

subplot(2, 1, 1);

stem(n, y, 'r');

xlabel("Time domain");

ylabel("Amplitude");

grid on;

axis([-10 10 -2 2]);

title("Original signal");

% Plot the compressed signal in the second subplot

subplot(2, 1, 2);

stem(x1, y1, 'g');

xlabel("Time domain");

ylabel("Amplitude");

grid on;

grid minor;

axis([-10 10 -2 2]);

title("Compressed signal Y[n] = X[2n]");

**Experiment 15:** A discrete time signal x(n) is shown in figure. Sketch the signal x[n], y[n]=x[n-4] and x[n+4], derived from x[n].

% Clear the command window, removing all existing variables

clc;

% Clear all existing figures

clear all;

% Define the time vector

n = -5:5;

% Define the original signal

x = [0 -1 -0.5 0.5 1 1 1 1 0.5 0 0];

% Plot the original signal in the first subplot

subplot(3, 1, 1);

stem(n, x);

xlabel('Time Sample');

ylabel('Amplitude');

title('Original Signal');

axis([-7 7 min(x)-2 max(x)+2]);

grid on;

grid minor;

% Right shift the signal by adding a constant to the time vector

m = n + 4;

% Plot the right-shifted signal in the second subplot

subplot(3, 1, 2);

stem(m, x);

xlabel('Time Sample');

ylabel('Amplitude');

title('Time Right-Shifted Signal');

axis([-7-2+4 7+2+4 min(x)-2 max(x)+2]);

grid on;

grid minor;

% Left shift the signal by subtracting a constant from the time vector

l = n - 4;

% Plot the left-shifted signal in the third subplot

subplot(3, 1, 3);

stem(l, x);

xlabel('Time Sample');

ylabel('Amplitude');

title('Time Left-Shifted Signal');

axis([-7-2-4 7+2-4 min(x)-2 max(x)+2]);

grid on;

grid minor;

**Experiment 16:** Find the even and odd components of the discrete-time signal x(n), where, x[n] = [5, 6, 3, 4, 1]

% Given time vector and original signal

n = -2:2;

x = [5, 6, 3, 4, 1];

% Creating mirrored versions for negative indices

x\_mirror = fliplr(x); % Reflecting the signal for negative indices

% even and odd components

xe = (x + x\_mirror) / 2; % Even part: (x(n) + x(-n))/2

xo = (x - x\_mirror) / 2; % Odd part: (x(n) - x(-n))/2

% Plotting the original signal

subplot(4, 1, 1);

stem(n, x);

grid on;

axis([-3 3 -1 7]);

xlabel('n');

ylabel('Amplitude');

title('Original Signal');

% Plotting the reversed (mirrored) signal

subplot(4, 1, 2);

stem(n, x\_mirror);

grid on;

axis([-3 3 -1 7]);

xlabel('n');

ylabel('Amplitude');

title('Reversed Signal');

% Plotting the even component

subplot(4, 1, 3);

stem(n, xe, 'b');

grid on;

axis([-3 3 -1 6]);

xlabel('n');

ylabel('Amplitude');

title('Even Signal');

% Plotting the odd component

subplot(4, 1, 4);

stem(n, xo, 'b');

grid on;

axis([-3 3 -3 3]);

xlabel('n');

ylabel('Amplitude');

title('Odd Signal');

**Experiment 17:** Scaling

y[n]=x[2n+3]

% Clear the command window and close all existing figures

clc;

close all;

% Original signal

n = -6:6;

y = [0 0 0 0 -1 -1 0 1 1 0 0 0 0];

% Plotting the original signal

subplot(3, 1, 1);

stem(n, y);

grid on;

axis([-8 8 -2 2]);

xlabel('n');

ylabel('Amplitude');

title('x[n]');

% Shifting the given signal to the right by 3 units

n1 = n - 3;

subplot(3, 1, 2);

stem(n1, y);

grid on;

axis([-8 8 -2 2]);

xlabel('n');

ylabel('Amplitude');

title('x[n+3]');

% Compressing the shifted signal by a factor of 2 and shifting it by 3 units

index = 1;

for i = 1:length(n1)

if (rem(n1(i), 2) == 0)

x1(index) = n1(i) / 2;

y1(index) = y(i);

index = index + 1;

end

end

% Plotting the final signal

subplot(3, 1, 3);

stem(x1, y1);

grid on;

axis([-8 8 -2 2]);

xlabel('n');

ylabel('Amplitude');

title('y[n] = x[2n+3]');

**Experiment 19:** The input x[n] of a LTI system,

% Convolution.m

clc;

clear all;

close all;

% Given signals

x1 = [-1 0 1 2];

y1 = [-1 0.5 1 -0.5];

x2 = [0 1 2 3];

h = [0.5 1 -0.5 0.5];

% Call the convolution function

[n, y] = func\_convoluation(x1, y1, x2, h);

% Plotting the given signal

subplot(3,1,1);

stem(x1, y1);

xlabel('X1');

ylabel('Y1');

title("Given Signal");

% Plotting the impulse response

subplot(3,1,2);

stem(x2, h);

xlabel('x2');

ylabel('h');

title("Impulse Response");

% Plotting the convolution result

subplot(3,1,3);

stem(n, y);

xlabel('n');

ylabel('y');

title("Convolution Sum");

% func\_convoluation.m

function [n, y] = func\_convoluation(x1, y1, x2, h)

% Determine the convolution range

m1 = min(x1) + min(x2);

m2 = max(x1) + max(x2);

% Generate the convolution time vector

n = m1:m2;

% Perform convolution using the built-in function

y = conv(y1, h);

end

Certainly! Let's break down the provided code in simpler terms:

1. \*\*Given Signals:\*\*

```matlab

x1 = [-1 0 1 2];

y1 = [-1 0.5 1 -0.5];

x2 = [0 1 2 3];

h = [0.5 1 -0.5 0.5];

```

These are the input signals `x1` and `x2`, along with their corresponding amplitudes `y1` and `h`.

2. \*\*Convolution Function Call:\*\*

```matlab

[n, y] = func\_convolution(x1, y1, x2, h);

```

This line calls a function `func\_convolution` with the given signals and receives the result in `n` (time vector) and `y` (convolution result).

3. \*\*Plotting the Given Signal:\*\*

```matlab

subplot(3,1,1);

stem(x1, y1);

xlabel('X1');

ylabel('Y1');

title("Given Signal");

```

This creates a subplot and plots the original signal `x1` with amplitudes `y1`.

4. \*\*Plotting the Impulse Response:\*\*

```matlab

subplot(3,1,2);

stem(x2, h);

xlabel('x2');

ylabel('h');

title("Impulse Response");

```

This creates another subplot and plots the impulse response signal `x2` with amplitudes `h`.

5. \*\*Plotting the Convolution Result:\*\*

```matlab

subplot(3,1,3);

stem(n, y);

xlabel('n');

ylabel('y');

title("Convolution Sum");

```

This creates the final subplot and plots the result of the convolution (`n` is the time vector, `y` is the convolution result).

6. \*\*Convolution Function:\*\*

```matlab

function [n, y] = func\_convolution(x1, y1, x2, h)

m1 = min(x1) + min(x2);

m2 = max(x1) + max(x2);

n = m1:m2;

y = conv(y1, h);

end

```

This function determines the convolution time vector (`n`) and performs the convolution using the built-in `conv` function.

In simpler terms, the code is demonstrating convolution between two signals (`x1` and `x2`) by calling a function (`func\_convolution`). It then plots the original signal, impulse response, and the result of the convolution.

**Aliasing**

clc;

clear all;

close all;

% Get user input for signal frequency and oversampling factor

frequency = input('Enter the frequency for the signal:\n');

fprintf("Enter the frequency, it must be greater or less than the Nyquist frequency\n");

oversampling = input('');

% Set up time parameters

Time\_Period = 1 / frequency;

tmin = -0.05;

tmax = 0.05;

% Generate time vector and original signal

time = linspace(tmin, tmax, 400);

amplitude = cos(2 \* pi \* frequency \* time);

% Plot the original signal

subplot(4, 1, 1);

plot(time, amplitude);

grid on; grid minor;

xlabel("Time");

ylabel("Amplitude");

title("Original Signal");

% Nyquist rate Sampling Part.

nyquist\_frequency = 2 \* frequency;

time1 = tmin:(1 / nyquist\_frequency):tmax;

amplitude1 = cos(2 \* pi \* frequency \* time1);

% Plot Nyquist sampling

subplot(4, 1, 2);

plot(time, amplitude);

hold on;

grid on; grid minor;

plot(time1, amplitude1);

title("Nyquist Sampling");

hold off;

% Check if oversampling is greater than Nyquist frequency

if oversampling > nyquist\_frequency

% OverSampling Part.

time1 = tmin:(1 / oversampling):tmax;

amplitude1 = cos(2 \* pi \* frequency \* time1);

subplot(4, 1, 3);

plot(time, amplitude);

hold on;

plot(time1, amplitude1);

grid on; grid minor;

xlabel("Time");

ylabel("Amplitude");

title("OverSampling");

hold off;

else

% UnderSampling Part.

time1 = tmin:(1 / oversampling):tmax;

amplitude1 = cos(2 \* pi \* frequency \* time1);

subplot(4, 1, 3);

plot(time, amplitude);

hold on;

plot(time1, amplitude1);

grid on; grid minor;

xlabel("Time");

ylabel("Amplitude");

title("UnderSampling");

hold off;

end

**Sinc**