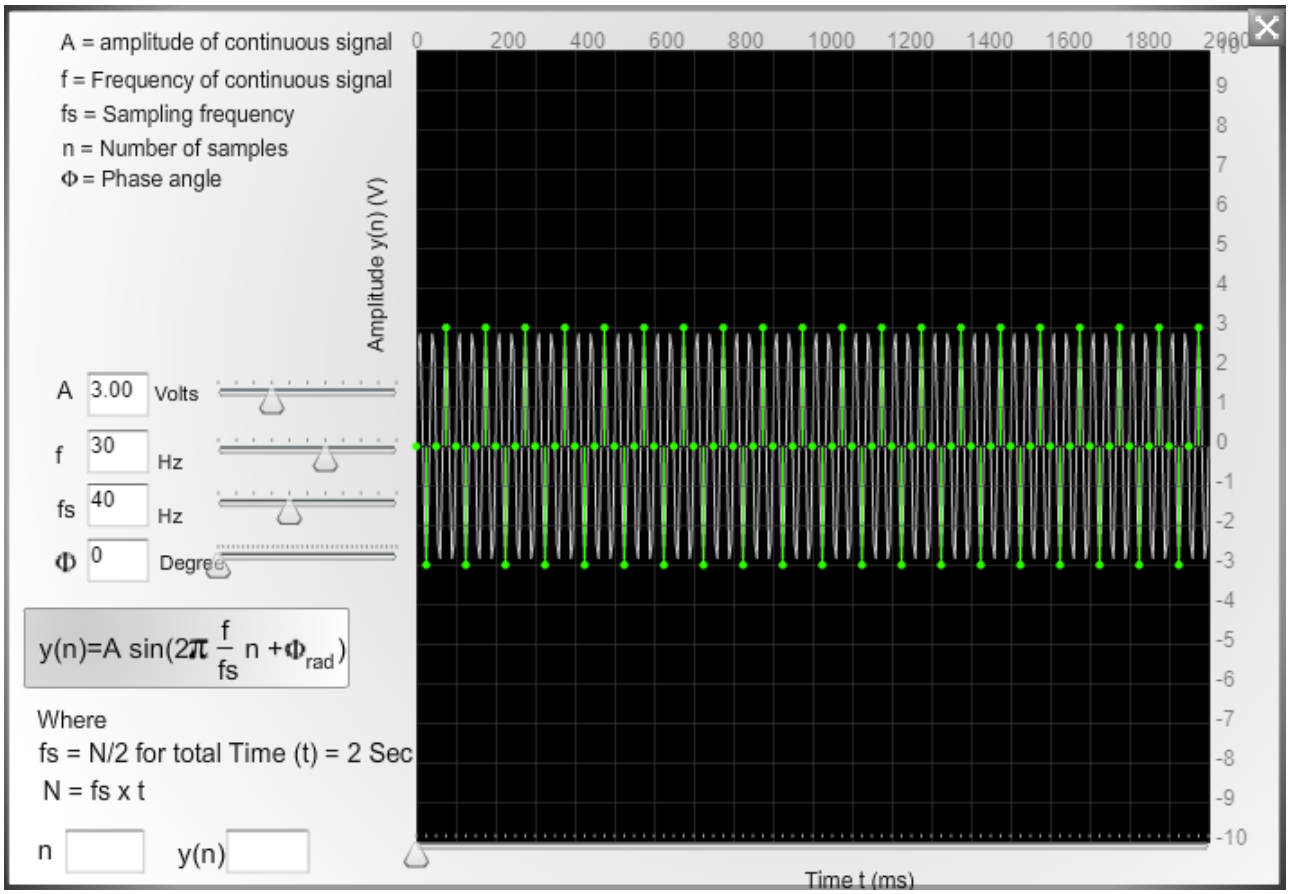
EC57L DSP - LAB

Lab Experiments' Report

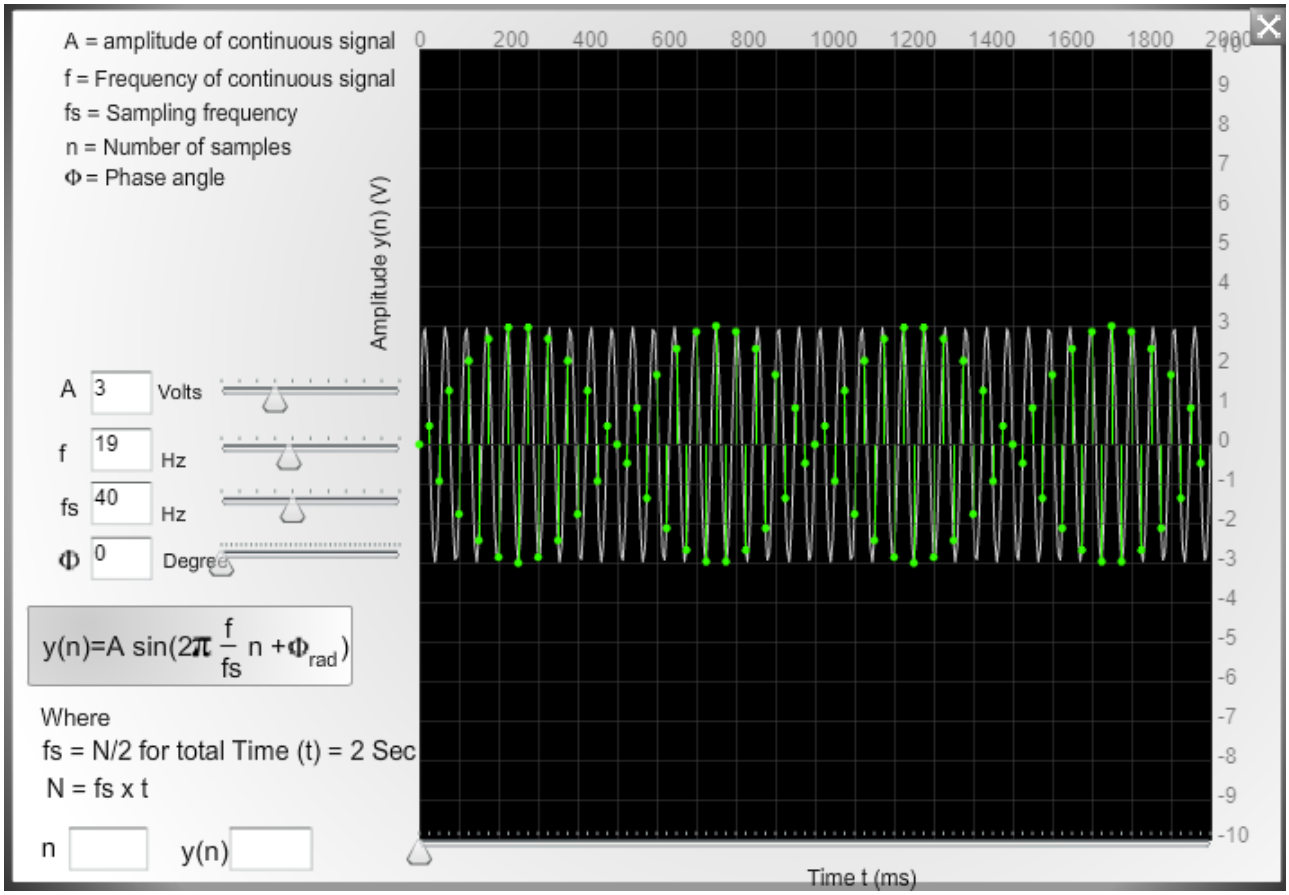
Experiment 1: Sampling Theorem

Sampling in Virtual Lab IIT-K

- Effect of Under Sampling



- Effect of Perfect Sampling



Sampling in MATLAB

Code:

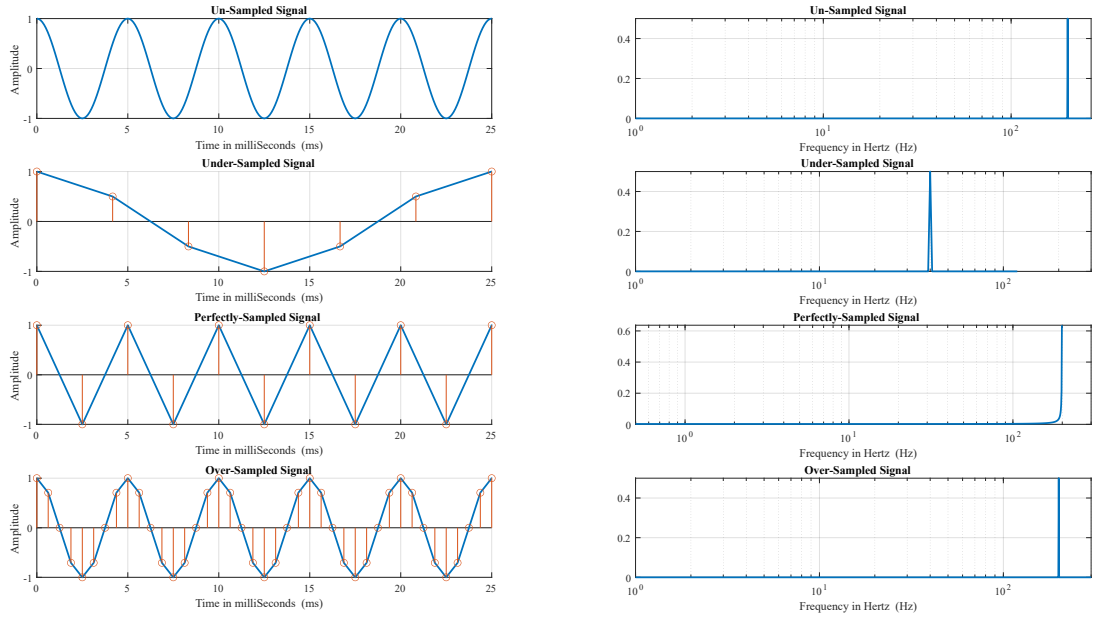
```
% THIS IS THE VERISON 2 OF THE EXPERIMENT 1 - VERIFICATION OF THE NYQUIST'S
THEOREM FOR SAMPLING
% AUTHOR: MOHAMED FARHAN FAZAL

clear all;
close all;
clc;
% Clearing the Workspace and Command Window

% Initiallizing the Variables

widthOfTheLine = 1.5;
numberOfWaves = 5; % Number of waveforms to be shown
messageSignalFrequency = input("Enter the Message Signal Frequency: "); % Taking
the user's input for Message Signal Frequency
initialSamplingFreq = 50*messageSignalFrequency; % Sampling Frequency for
Unsampled Signal
timePerSample = 1/initialSamplingFreq; % Time needed for a single sample
stopTime = 1; % Samples to be generated up to.
timeAxis = 0:timePerSample:stopTime-timePerSample; % Generating the time axis
totalNumberOfSamples = size(timeAxis,2); % Calculating the number of samples
samplingFrequencyInterval = initialSamplingFreq/totalNumberOfSamples; %
```

```
Calculating the frequency interval to generate the frequency axis;
frequencyAxis = -
initialSamplingFreq/2:samplingFrequencyInterval:initialSamplingFreq/2-
samplingFrequencyInterval; % Generating the Frequency Axis
phiDegrees = 90; % in degrees

phi = phiDegrees * pi / 180;



% Plotting the Unsampled Signal
% In time Domain
subplot(421);
xData = sin(2*pi*messageSignalFrequency*timeAxis + phi);
plot(1000*timeAxis, xData, "lineWidth", widthOfTheLine);
set(get(gca, 'XLabel'), 'String', 'Time in milliSeconds  (ms)');
set(get(gca, 'YLabel'), 'String', 'Amplitude');
set(get(gca, 'Title'), 'String', 'Un-Sampled Signal');
grid on;
axis([0 1000*numberOfWaves/messageSignalFrequency -1 1]);

% In Frequency Domain
subplot(422);
xDataFFT = fftshift(fft(xData));
semilogx(frequencyAxis, abs(xDataFFT)/totalNumberOfSamples, "lineWidth",
widthOfTheLine);
set(get(gca, 'XLabel'), 'String', 'Frequency in Hertz  (Hz)');
set(get(gca, 'Title'), 'String', 'Un-Sampled Signal');
grid on;
axis([0 messageSignalFrequency+messageSignalFrequency/3 0 inf]);


% Plotting Under Sampled Signal

underSamplingFrequency = 1.2*messageSignalFrequency; % Deciding the Frequency for
demonstraring undersampling
under_timePerSample = 1/underSamplingFrequency; % Time needed for a single sample
under_timeAxis = 0:under_timePerSample:stopTime-under_timePerSample; % Generating
the time axis
under_totalNumberOfSamples = size(under_timeAxis,2); % Calculating the number of
samples
under_samplingFrequencyInterval =
underSamplingFrequency/under_totalNumberOfSamples; % Calculating the frequency
interval to generate the frequency axis;
under_frequencyAxis = -
underSamplingFrequency/2:under_samplingFrequencyInterval:underSamplingFrequency/2-
under_samplingFrequencyInterval; % Generating the Frequency Axis

% In time Domain
subplot(423);
under_xData = sin(2*pi*messageSignalFrequency*under_timeAxis + phi);
hold on;
plot(1000*under_timeAxis, under_xData, "lineWidth", widthOfTheLine);
```

```
stem(1000*under_timeAxis, under_xData);
set(get(gca, 'XLabel'), 'String', 'Time in milliSeconds  (ms)');
set(get(gca, 'YLabel'), 'String', 'Amplitude');
set(get(gca, 'Title'), 'String', 'Under-Sampled Signal');
hold off;
grid on;
axis([0 1000*numberOfWaves/messageSignalFrequency -1 1]);

% In Frequency Domain
subplot(424);
under_xDataFFT = fftshift(fft(under_xData));
semilogx(under_frequencyAxis, abs(under_xDataFFT)/under_totalNumberOfSamples,
"lineWidth", widthOfTheLine);
set(get(gca, 'XLabel'), 'String', 'Frequency in Hertz  (Hz)');
set(get(gca, 'Title'), 'String', 'Under-Sampled Signal');
grid on;
axis([0 messageSignalFrequency+messageSignalFrequency/2 0 inf]);



% Plotting Perfect Sampled Signal

perfectSamplingFrequency = 2*messageSignalFrequency; % Deciding the Frequency for
demonstraring perfectsampling
perfect_timePerSample = 1/perfectSamplingFrequency; % Time needed for a single
sample
perfect_timeAxis = 0:perfect_timePerSample:stopTime; % Generating the time axis
%removed "-perfect_timePerSample"
perfect_totalNumberOfSamples = size(perfect_timeAxis,2); % Calculating the number
of samples
perfect_samplingFrequencyInterval =
perfectSamplingFrequency/perfect_totalNumberOfSamples; % Calculating the frequency
interval to generate the frequency axis;
perfect_frequencyAxis = -
perfectSamplingFrequency/2:perfect_samplingFrequencyInterval:perfectSamplingFreque
ncy/2-perfect_samplingFrequencyInterval; % Generating the Frequency Axis

% In time Domain
subplot(425);
perfect_xData = sin(2*pi*messageSignalFrequency*perfect_timeAxis + phi);
hold on;
plot(1000*perfect_timeAxis, perfect_xData, "lineWidth", widthOfTheLine);
stem(1000*perfect_timeAxis, perfect_xData);
set(get(gca, 'XLabel'), 'String', 'Time in milliSeconds  (ms)');
set(get(gca, 'YLabel'), 'String', 'Amplitude');
set(get(gca, 'Title'), 'String', 'Perfectly-Sampled Signal');
hold off;
grid on;
axis([0 1000*numberOfWaves/messageSignalFrequency -1 1]);

% In Frequency Domain
subplot(426);
perfect_xDataFFT = fftshift(fft(perfect_xData));
```

```
semilogx(perfect_frequencyAxis,
abs(perfect_xDataFFT)/perfect_totalNumberOfSamples, "lineWidth", widthOfTheLine);
set(get(gca, 'XLabel'), 'String', 'Frequency in Hertz  (Hz)');
set(get(gca, 'Title'), 'String', 'Perfectly-Sampled Signal');
grid on;
axis([0 messageSignalFrequency+messageSignalFrequency/2 0 inf]);


% Plotting Over Sampled Signal

overSamplingFrequency = 8*messageSignalFrequency; % Deciding the Frequency for
demonstraring oversampling
over_timePerSample = 1/overSamplingFrequency; % Time needed for a single sample
over_timeAxis = 0:over_timePerSample:stopTime-over_timePerSample; % Generating the
time axis
over_totalNumberOfSamples = size(over_timeAxis,2); % Calculating the number of
samples
over_samplingFrequencyInterval = overSamplingFrequency/over_totalNumberOfSamples;
% Calculating the frequency interval to generate the frequency axis;
over_frequencyAxis = -
overSamplingFrequency/2:over_samplingFrequencyInterval:overSamplingFrequency/2-
over_samplingFrequencyInterval; % Generating the Frequency Axis

% In time Domain
subplot(427);
over_xData = sin(2*pi*messageSignalFrequency*over_timeAxis + phi);
hold on;
plot(1000*over_timeAxis, over_xData, "lineWidth", widthOfTheLine);
stem(1000*over_timeAxis, over_xData);
set(get(gca, 'XLabel'), 'String', 'Time in milliSeconds  (ms)');
set(get(gca, 'YLabel'), 'String', 'Amplitude');
set(get(gca, 'Title'), 'String', 'Over-Sampled Signal');
hold off;
grid on;
axis([0 1000*numberOfWaves/messageSignalFrequency -1 1]);

% In Frequency Domain
subplot(428);
over_xDataFFT = fftshift(fft(over_xData));
semilogx(over_frequencyAxis, abs(over_xDataFFT)/over_totalNumberOfSamples,
"lineWidth", widthOfTheLine);
set(get(gca, 'XLabel'), 'String', 'Frequency in Hertz  (Hz)');
set(get(gca, 'Title'), 'String', 'Over-Sampled Signal');
grid on;
axis([0 messageSignalFrequency+messageSignalFrequency/2 0 inf]);

% Clearing the Workspace and the Command Window
clear all;
clc;
```

Results



Experiment 2: DFT, IDFT, DTFT

Code:

1. dft.m

```
function DFT = dft(sequence)

sequenceLength = length(sequence);
twiddleFactorMatrix = twiddleFactor(sequenceLength);
DFT = twiddleFactorMatrix*sequence.';

end
```

2. dtft.m

```
function [DTFT dtftPointLength] = dtft(sequence)
    sequenceLength = length(sequence);
    dtftPointLength = 1000;
    DTFT = dft([sequence zeros(1,dtftPointLength-sequenceLength)]);
end
```

3. idft.m

```
function IDFT = idft(sequence)
    IDFT = conj(dft(conj(sequence.')))/length(sequence);
end
```

4. setAxisLimits.m

```
function setAxisLimits(axisData)
    padding = 0.1; % Relative to the overall output
    axisLength = axisData(2) - axisData(1);
    axisHeight = axisData(4) - axisData(3);
    axis([axisData(1) - padding * axisLength axisData(2) + padding * axisLength
axisData(3) - padding * axisHeight axisData(4) + padding * axisHeight]);
end
```

5. setPlotAttributes.m

```
function setPlotAttributes(xAxisLabel, yAxisLabel, plotTitle)
    set(get(gca, 'XLabel'), 'String', xAxisLabel);
    set(get(gca, 'YLabel'), 'String', yAxisLabel);
    set(get(gca, 'Title'), 'String', plotTitle);
end
```

6. twiddleFactor.m

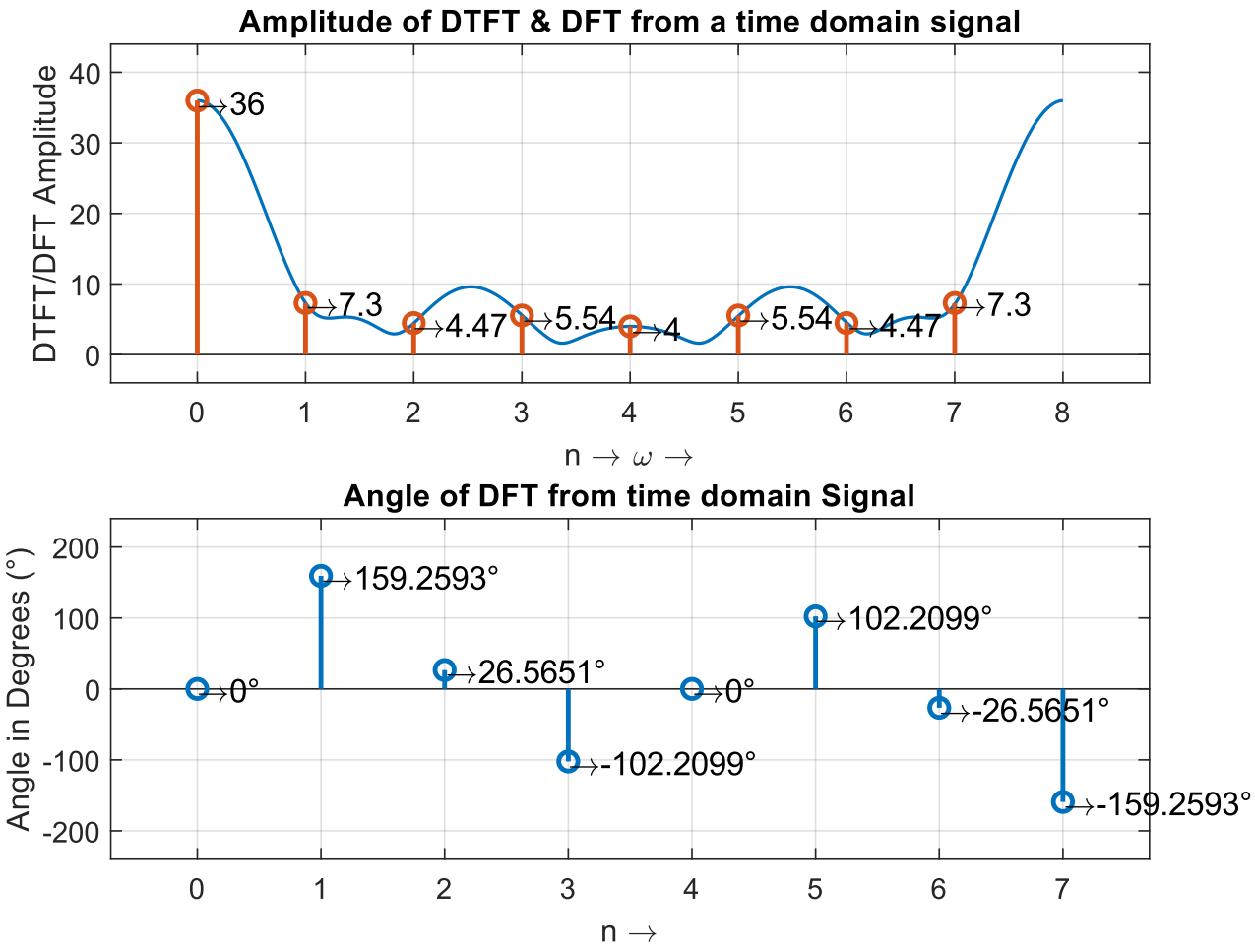
```
function twiddleMatrix = twiddleFactor(sequenceLenght, needMatrix)
    %arguments
    %sequenceLenght
    %needMatrix = 1;
    %end
    needMatrix = 1;
    twiddleMatrix = complex(ones([sequenceLenght sequenceLenght]));
    theta = 2 * pi / sequenceLenght;

    for index1 = 2:sequenceLenght
        for index2 = 2:sequenceLenght
            twiddleMatrix(index1, index2) = cos(theta * (index2 - 1) * (index1 -
1)) - i * sin(theta * (index2 - 1) * (index1 - 1));
        end
    end

    if ~needMatrix
        twiddleMatrix = twiddleMatrix(:, 2);
    end
```

Results

DFT and DTFT

- Input Sequence: 4 3 2 6 8 4 6 3

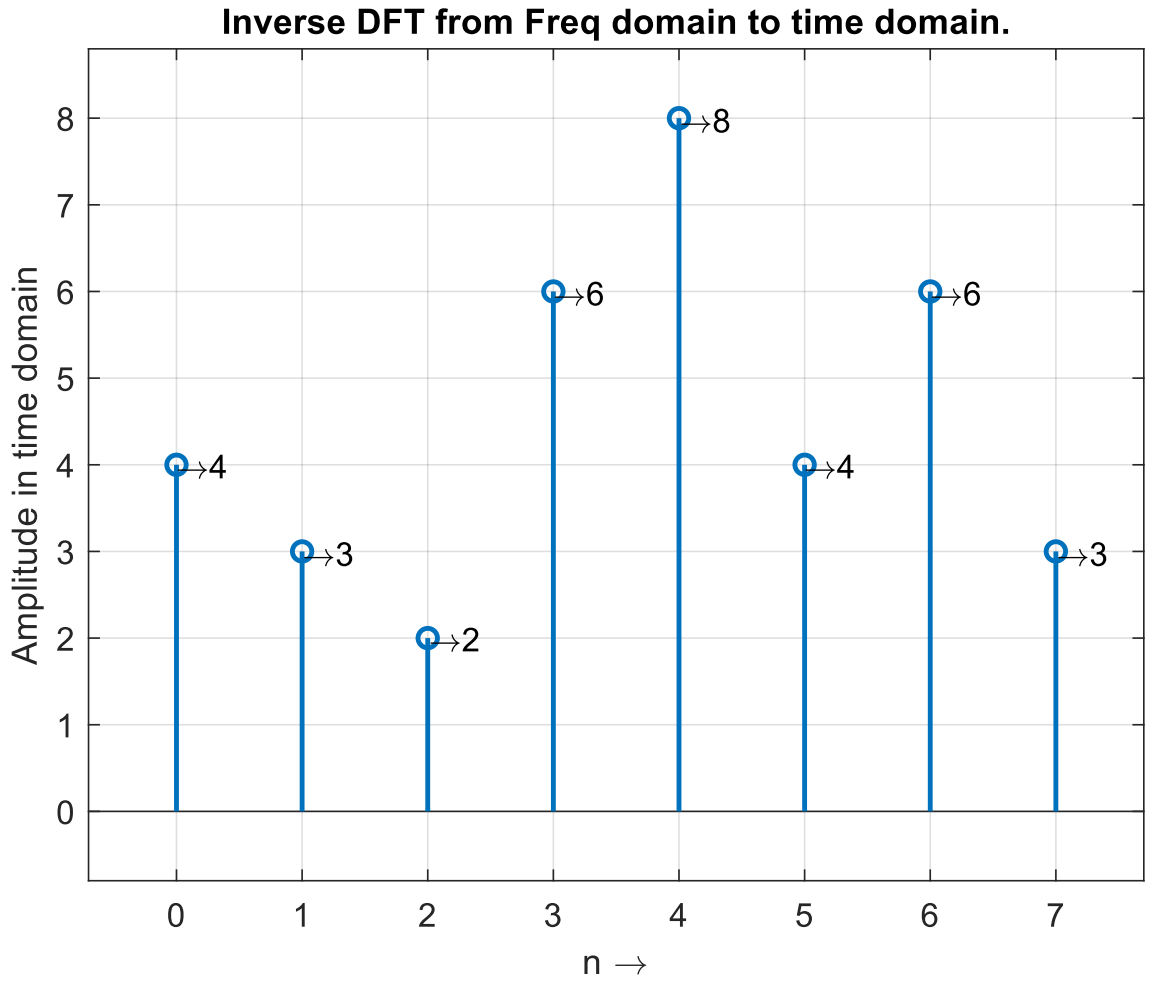
```
Enter the Sequence in time domain: [4 3 2 6 8 4 6 3]

ans =

    "Calculated using manually coded function"


ans =

  36.0000 + 0.0000i
  -6.8280 + 2.5860i
   4.0000 + 2.0000i
  -1.1720 - 5.4140i
   4.0000 + 0.0000i
  -1.1720 + 5.4140i
   4.0000 - 2.0000i
  -6.8280 - 2.5860i


ans =

    "Calculated using builtin coded function"


ans =

  36.0000 + 0.0000i
  -6.8280 + 2.5860i
   4.0000 + 2.0000i
  -1.1720 - 5.4140i
   4.0000 + 0.0000i
  -1.1720 + 5.4140i
   4.0000 - 2.0000i
  -6.8280 - 2.5860i
```



IDFT

- Input Sequence:

```
36.0000 + 0.0000i
-6.8280 + 2.5860i
 4.0000 + 2.0000i
-1.1720 - 5.4140i
 4.0000 + 0.0000i
-1.1720 + 5.4140i
 4.0000 - 2.0000i
-6.8280 - 2.5860i
```



Experiment 3: Properties of DFT 1

1. Linearity

Code:

```
clear all;
close all;
clc;

widthOfLine = 1;

sequence1 = input("Enter sequence 1: \n");
sequence2 = input("Enter sequence 2: \n");

% sequence1 = [1 2 3 4];
% sequence2 = [5 6 7 8];

figure("Name", "Using manually coded functions !");

DFT1 = dft(sequence1);
DFT2 = dft(sequence2);
combinedDFT = dft(sequence1 + sequence2);
```

```
disp("Using manually coded functions !");

disp(strcat(['DFT of sequence 1: x1(n)']));
disp(round(DFT1, 3));

disp(strcat(['DFT of sequence 2: x2(n)']));
disp(round(DFT2, 3));

disp(strcat(['DFT of sum of x1(n) and x2(n): ']));
disp(round(combinedDFT, 3));

disp(strcat(['Sum of DFTs x1(n) and x2(n): ']));
disp(round(DFT1 + DFT2, 3));

xAxis = linspace(0, length(sequence2) - 1, length(sequence2));

subplot(4, 2, 1);
localY = abs(DFT1);
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("n \rightarrow", "Amplitude", "DFT of sequence 1 $$ x_1(n) $$");
text(xAxis, localY, strcat("\leftarrow", num2str(round(localY, 1))));


subplot(4, 2, 2);
localY = angle(round(DFT1,5))*180/pi;
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("n \rightarrow", "Phase (°)", "DFT of sequence 1 $$ x_1(n) $$");
text(xAxis, localY, strcat("\leftarrow", num2str(round(localY, 1))));

subplot(4, 2, 3);
localY = abs(DFT2);
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("n \rightarrow", "Amplitude", "DFT of sequence 2 $$ x_2(n) $$");
text(xAxis, localY, strcat("\leftarrow", num2str(round(localY, 1))));


subplot(4, 2, 4);
localY = angle(round(DFT2,5))*180/pi;
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("n \rightarrow", "Phase (°)", "DFT of sequence 2 $$ x_2(n) $$");
text(xAxis, localY, strcat("\leftarrow", num2str(round(localY, 1))));

subplot(4, 2, 5);
localY = abs(combinedDFT);
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("n \rightarrow", "Amplitude", "DFT of sum of $$ x_1(n) $$ and $$
x_2(n) $$");
```

```
text(xAxis, localY, strcat("\leftarrow", num2str(round(localY, 1))));


subplot(4, 2, 6);
localY = angle(round(combinedDFT,5))*180/pi;
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("n \rightarrow", "Phase (°)", "DFT of sum of $$ x_1(n) $$ and $$
x_2(n) $$");
text(xAxis, localY, strcat("\leftarrow", num2str(round(localY, 1))));

subplot(4, 2, 7);
localY = abs(DFT1+DFT2);
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("n \rightarrow", "Amplitude", "Sum of DFTs of $$ x_1(n) $$ and
$$ x_2(n) $$");
text(xAxis, localY, strcat("\leftarrow", num2str(round(localY, 1))));


subplot(4, 2, 8);
localY = angle(round(DFT1+DFT2,5))*180/pi;
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("n \rightarrow", "Phase (°)", "Sum of DFTs of $$ x_1(n) $$ and
$$ x_2(n) $$");
text(xAxis, localY, strcat("\leftarrow", num2str(round(localY, 1))));



figure("Name", "Using builtin functions !")
disp("Using builtin functions !");

DFT1 = fft(sequence1);
DFT2 = fft(sequence2);
combinedDFT = fft(sequence1 + sequence2);

disp(strcat(['DFT of sequence 1: x1(n)']));
disp(round(DFT1.', 3));

disp(strcat(['DFT of sequence 2: x2(n)']));
disp(round(DFT2.', 3));

disp(strcat(['DFT of sum of x1(n) and x2(n): ']));
disp(round(combinedDFT.', 3));

disp(strcat(['Sum of DFTs x1(n) and x2(n): ']));
disp(round((DFT1 + DFT2).', 3));

xAxis = linspace(0, length(sequence2) - 1, length(sequence2));

subplot(4, 2, 1);
localY = abs(DFT1.');
```

```
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("n \rightarrow", "Amplitude", "DFT of sequence 1 $$ x_1(n) $$");
text(xAxis, localY, strcat("\leftarrow", num2str(round(localY, 1))));


subplot(4, 2, 2);
localY = angle(round(DFT1.',5))*180/pi;
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("n \rightarrow", "Phase (°)", "DFT of sequence 1 $$ x_1(n) $$");
text(xAxis, localY, strcat("\leftarrow", num2str(round(localY, 1))));

subplot(4, 2, 3);
localY = abs(DFT2.');
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("n \rightarrow", "Amplitude", "DFT of sequence 2 $$ x_2(n) $$");
text(xAxis, localY, strcat("\leftarrow", num2str(round(localY, 1))));


subplot(4, 2, 4);
localY = angle(round(DFT2.',5))*180/pi;
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("n \rightarrow", "Phase (°)", "DFT of sequence 2 $$ x_2(n) $$");
text(xAxis, localY, strcat("\leftarrow", num2str(round(localY, 1))));

subplot(4, 2, 5);
localY = abs(combinedDFT.');
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("n \rightarrow", "Amplitude", "DFT of sum of $$ x_1(n) $$ and $$
x_2(n) $$");
text(xAxis, localY, strcat("\leftarrow", num2str(round(localY, 1))));


subplot(4, 2, 6);
localY = angle(round(combinedDFT.',5))*180/pi;
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("n \rightarrow", "Phase (°)", "DFT of sum of $$ x_1(n) $$ and $$
x_2(n) $$");
text(xAxis, localY, strcat("\leftarrow", num2str(round(localY, 1))));

subplot(4, 2, 7);
localY = abs((DFT1+DFT2).');
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("n \rightarrow", "Amplitude", "Sum of DFTs of $$ x_1(n) $$ and
$$ x_2(n) $$");
text(xAxis, localY, strcat("\leftarrow", num2str(round(localY, 1))));
```

```
subplot(4, 2, 8);
localY = angle(round((DFT1+DFT2).',5))*180/pi;
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("n \rightarrow", "Phase (°)", "Sum of DFTs of $$ x_1(n) $$ and
$$ x_2(n) $$");
text(xAxis, localY, strcat("\leftarrow", num2str(round(localY, 1))));
```

2. Periodicity

Code:

```
clear all;
close all;
clc;

sequence = input("Enter the input sequence: ");


N = length(sequence);
disp(strcat("Period N: ", num2str(N)))
n = 1:N;
k = n;
xAxis = n-1;
widthOfLine = 1.5;


DFT(k) = dft(sequence(n));
shiftedDFT(k) = dft(sequence(mod(N+n,N+1)+1));

disp("DFT of x(n): ");
disp(DFT.');

disp("DFT of x(n+N): ");
disp(shiftedDFT.');

figure('Name','Using manually coded functions !')

subplot(2,2,1);
localY = abs(DFT.');
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("n \rightarrow", "Amplitude", "DFT of $$ x(n) $$");
text(xAxis, localY, strcat("\leftarrow", num2str(round(localY, 1))));
grid on;

subplot(2, 2, 2);
localY = angle(round(DFT.',5))*180/pi;
stem(xAxis, localY, "lineWidth", widthOfLine);
```

```
setAxisLimits(axis);
setPlotAttributes("n \rightarrow", "Phase (°)", "DFT of $$ x(n) $$");
text(xAxis, localY, strcat("\leftarrow", num2str(round(localY, 1))));
grid on;

subplot(2,2,3);
localY = abs(shiftedDFT.');
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("n \rightarrow", "Amplitude", "DFT of $$ x(n+N) $$");
text(xAxis, localY, strcat("\leftarrow", num2str(round(localY, 1))));
grid on;

subplot(2, 2, 4);
localY = angle(round(shiftedDFT.',5))*180/pi;
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("n \rightarrow", "Phase (°)", "DFT of $$ x(n+N) $$");
text(xAxis, localY, strcat("\leftarrow", num2str(round(localY, 1))));
grid on;


% Using builtin functions

DFT(k) = fft(sequence(n));
shiftedDFT(k) = fft(sequence(mod(N+n,N+1)+1));

disp("DFT of x(n): ");
disp(DFT.');

disp("DFT of x(n+N): ");
disp(shiftedDFT.');

figure('Name','Using builtin matlab functions !')

subplot(2,2,1);
localY = abs(DFT.');
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("n \rightarrow", "Amplitude", "DFT of $$ x(n) $$");
text(xAxis, localY, strcat("\leftarrow", num2str(round(localY, 1))));
grid on;

subplot(2, 2, 2);
localY = angle(round(DFT.',5))*180/pi;
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("n \rightarrow", "Phase (°)", "DFT of $$ x(n) $$");
text(xAxis, localY, strcat("\leftarrow", num2str(round(localY, 1))));
grid on;

subplot(2,2,3);
localY = abs(shiftedDFT.');
```

```
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("n \rightarrow", "Amplitude", "DFT of $$ x(n+N) $$");
text(xAxis, localY, strcat("\leftarrow", num2str(round(localY, 1))));
grid on;

subplot(2, 2, 4);
localY = angle(round(shiftedDFT.',5))*180/pi;
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("n \rightarrow", "Phase (°)", "DFT of $$ x(n+N) $$");
text(xAxis, localY, strcat("\leftarrow", num2str(round(localY, 1))));
grid on;
```

3. Circular time shift property of a sequence

Code:

```
clear all;
close all;
clc

sequence = input("Enter the sequence: ");
m = input("Enter the number of shifts: ");

% sequence = [4 -4 5 -5];
% m = 2;

N = length(sequence);

n = 1:N;
k = n;
xAxis = n - 1;
widthOfLine = 1;

x(n) = sequence;

DFT(k) = dft(x(n));
disp("Input Sequence:");
disp(sequence);
disp("DFT of the input sequence is:");
disp(DFT(k));

shiftedDFT(k) = dft(x(circshift(n, m)));
disp("Circularly Shifted Signal:");
disp(x(circshift(n, m)));
disp("DFT of circularly shifted signal:");
disp(shiftedDFT);

shiftedUsingProperty(k) = DFT * (exp(-i * 2 * pi * (k - 1) * m / N) .* eye(N));
```

```
disp("DFT of circularly shifted signal using property:");
disp(shiftedUsingProperty);

% DFT of signal

subplot(3, 2, 1);
localY = abs(DFT.');
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("n \rightarrow", "Amplitude", "DFT of $$ x(n) $$");
text(xAxis, localY, strcat("\leftarrow", num2str(round(localY, 1))));
grid on;

subplot(3, 2, 2);
localY = angle(round(DFT.', 5)) * 180 / pi;
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("n \rightarrow", "Phase (°)", "DFT of $$ x(n) $$");
text(xAxis, localY, strcat("\leftarrow", num2str(round(localY, 1))));
grid on;

% DFT of shifted signal

subplot(3, 2, 3);
localY = abs(shiftedDFT.');
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("n \rightarrow", "Amplitude", "DFT of $$ x(n+m) $$");
text(xAxis, localY, strcat("\leftarrow", num2str(round(localY, 1))));
grid on;

subplot(3, 2, 4);
localY = angle(round(shiftedDFT.', 5)) * 180 / pi;
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("n \rightarrow", "Phase (°)", "DFT of $$ x(n+m) $$");
text(xAxis, localY, strcat("\leftarrow", num2str(round(localY, 1))));
grid on;

% DFT of shifted signal using property

subplot(3, 2, 5);
localY = abs(shiftedUsingProperty.');
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("n \rightarrow", "Amplitude", "DFT of $$ x(n+m) $$ using
property");
text(xAxis, localY, strcat("\leftarrow", num2str(round(localY, 1))));
grid on;

subplot(3, 2, 6);
localY = angle(round(shiftedUsingProperty.', 5)) * 180 / pi;
stem(xAxis, localY, "lineWidth", widthOfLine);
```

```
setAxisLimits(axis);
setPlotAttributes("n \rightarrow", "Phase (°)", "DFT of $$ x(n+m) $$ using
property");
text(xAxis, localY, strcat("\leftarrow", num2str(round(localY, 1))));
grid on;
```

Experiment 4: Properties of DFT 2

Code:

1. reverse.m

```
function REVERSED = reverse(sequence)
    REVERSED = circshift(fliplr(sequence), 1);
end
```

2. circularConvolution.m

```
function convolutionResult = circularConv(sequence1, sequence2)
    N1 = length(sequence1);
    N2 = length(sequence2);

    % CHECKING IF LENGTH OF 2 SEQUENCES IS EQUAL
    if N1 ~= N2
        % FIND OUT THE MAX LENGTH OF 2 SEQUENCES
        N = max(N1, N2);

        % APPENDING ZEROS
        sequence1 = [sequence1 zeros(1,N-N1)];
        sequence2 = [sequence2 zeros(1,N-N2)];
    end
    circularMatrix = toeplitz(sequence1, reverse(sequence1));
    convolutionResult = circularMatrix * sequence2.';
end
```

Circular Convolution

```
clear all;
close all;
clc

widthOfLine = 1;

sequence1 = input("Enter the Sequence 1: ");
sequence2 = input("Enter the Sequence 2: ");
```

```
DFT1 = dft(sequence1);
DFT2 = dft(sequence2);

% CIRCULAR CONVOLUTION IN TIME DOMAIN
timeDomain = circularConvolution(sequence1, sequence2);

% CIRCULAR CONVOLUTION IN FREQUENCY DOMAIN
N1 = length(sequence1);
N2 = length(sequence2);

% FIND OUT THE MAX LENGTH OF 2 SEQUENCES
N = max(N1, N2);

% CHECKING IF LENGTH OF 2 SEQUENCES IS EQUAL
if N1 ~= N2
    % APPENDING ZEROS
    sequence1 = [sequence1 zeros(1,N-N1)];
    sequence2 = [sequence2 zeros(1,N-N2)];
end

% % METHOD 1
% freqDomain = zeros(1,N);

% for i = 0:N-1
%     for j = 0:N-1
%         freqDomain(mod(i+j,N)+1) = freqDomain(mod(i+j,N)+1) +
sequence1(i+1)*sequence2(j+1);
%     end
% end

% % METHOD 2
freqDomain = idft(dft(sequence1).*dft(sequence2));

disp("Circular Convolution in time domain: ");
disp(timeDomain.');

disp("Circular Convolution in frequency domain: ");
disp(abs(freqDomain).');

disp("DFT of sequence 1: ");
disp(DFT1);

disp('DFT of sequence 2: ');
disp(DFT2);

xAxis = 0:N1-1;
subplot(2, 2, 1);
localY = abs(DFT1);
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("k \rightarrow", "Amplitude", "DFT of sequence1 $$ \rightarrow
X(k) $$");
for index = xAxis
```

```
    localY1 = localY(index + 1);
    text(index, localY1, strcat("\leftarrow", num2str(round(localY1, 1))));
end

subplot(2, 2, 2);
localY = angle(round(DFT1,5))*180/pi;
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("k \rightarrow", "Phase (°)", "DFT of sequence1 $$ \rightarrow
X(k) $$");
for index = xAxis
    localY1 = localY(index + 1);
    text(index, localY1, strcat("\leftarrow", num2str(round(localY1, 1))));
end

xAxis = 0:N2-1;
subplot(2, 2, 3);
localY = abs(DFT2);
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("k \rightarrow", "Amplitude", "DFT of sequence2 $$ \rightarrow
X(k) $$");
for index = xAxis
    localY1 = localY(index + 1);
    text(index, localY1, strcat("\leftarrow", num2str(round(localY1, 1))));
end

subplot(2, 2, 4);
localY = angle(round(DFT2,5))*180/pi;
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("k \rightarrow", "Phase (°)", "DFT of sequence2 $$ \rightarrow
X(k) $$");
for index = xAxis
    localY1 = localY(index + 1);
    text(index, localY1, strcat("\leftarrow", num2str(round(localY1, 1))));
end


figure(2);

xAxis = 0:N-1;
subplot(2, 1, 1);
localY = timeDomain;
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("n \rightarrow", "Amplitude", "Convolution in Time Domain");
for index = xAxis
    localY1 = localY(index + 1);
    text(index, localY1, strcat("\leftarrow", num2str(round(localY1, 1))));
end

subplot(2, 1, 2);
```

```
localY = abs(freqDomain);
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("n \rightarrow", "Amplitude", "Convolution in Frequency
Domain");
for index = xAxis
    localY1 = localY(index + 1);
    text(index, localY1, strcat("\leftarrow", num2str(round(localY1, 1))));
end
```

Circular Frequency Shift

```
clear all;
close all;
clc;

widthOfLine = 1;

sequence = input("Enter the sequence x(n): ");
m = input("Enter the value (m) to be shifted: ");


% sequence = [4 5 6 7];
% m = 1;

N = length(sequence);

n = 1:N;
k = n;

x(n) = sequence;

X(k) = fft(x);

x1(n) = x.*exp(i*2*pi*m*(n-1)/N);
X1(k) = fft(x1(n));

X2(k) = circshift(X, m);

disp("DFT of x(n) is: ");
disp(X);

disp("Circularly Shifted X(l-k)N: ");
disp(X2);

disp("DFT of x(n)*e^(j*2*pi*m*n/N) is: ");
disp(X1);

xAxis = 0:N-1;
```

```
subplot(3, 2, 1);
localY = abs(X);
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("k \rightarrow", "Amplitude", "DFT of $$ x(n) \rightarrow X(k)
$$");
for index = xAxis
    localY1 = localY(index + 1);
    text(index, localY1, strcat("\leftarrow", num2str(round(localY1, 1))));
end

subplot(3, 2, 2);
localY = angle(round(X,5))*180/pi;
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("k \rightarrow", "Phase (°)", "DFT of $$ x(n) \rightarrow X(k)
$$");
for index = xAxis
    localY1 = localY(index + 1);
    text(index, localY1, strcat("\leftarrow", num2str(round(localY1, 1))));
end


subplot(3, 2, 3);
localY = abs(X1);
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("k \rightarrow", "Amplitude", "Circularly Shifted DFT signal by
$$ m \rightarrow X(m-k)_N $$");
for index = xAxis
    localY1 = localY(index + 1);
    text(index, localY1, strcat("\leftarrow", num2str(round(localY1, 1))));
end

subplot(3, 2, 4);
localY = angle(round(X1,5))*180/pi;
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("k \rightarrow", "Phase (°)", "Circularly Shifted DFT signal by
$$ m \rightarrow X(m-k)_N $$");
for index = xAxis
    localY1 = localY(index + 1);
    text(index, localY1, strcat("\leftarrow", num2str(round(localY1, 1))));
end

subplot(3, 2, 5);
localY = abs(X2);
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("k \rightarrow", "Amplitude", "DFT of $$ x(n)*e^(j*2*pi*m*n/N)
\rightarrow X(m-k)_N $$");
for index = xAxis
```

```
    localY1 = localY(index + 1);
    text(index, localY1, strcat("\leftarrow", num2str(round(localY1, 1))));
end

subplot(3, 2, 6);
localY = angle(round(X2,5))*180/pi;
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("k \rightarrow", "Phase (°)", "DFT of $$ x(n)*e^{(j*2*pi*m*n/N)}
\rightarrow X(m-k)_N $$");
for index = xAxis
    localY1 = localY(index + 1);
    text(index, localY1, strcat("\leftarrow", num2str(round(localY1, 1))));
end
```

Conjugate Symmetry

```
clear all;
close all;
clc

sequence = input("Enter a 'Real' sequence: ");
N = length(sequence);

n = 1:N;

x(n) = sequence;    % INPUT SEQUENCE
X = dft(x);         % DFT OF INPUT SEQUENCE

x1 = reverse(x);    % REVERSED INPUT SEQUENCE x(-n)
X1 = conj(dft(x1));;  % DFT OF CONJUGATE OF INPUT SEQUENCE


disp("x(n) is: ");
disp(x);

disp("DFT of x(n) ==> X(k)")
disp(X);

disp("X(-n) is: ");
disp(x1);

disp("Conjugate of DFT of x(-n) ==> X*((-k))N");
disp(X1);

if round(X,2) == round(X1,2)
    disp("The Sequence is Conjugate Symmetric");
else
    disp("The Sequence is not Conjugate Symmetric");
end
```

Experiment 5: Properties of DFT 3

Code:

1. toeplitzDSP.m

```
function A = toeplitzDSP(xn, hn)

length_of_xn = length(xn);
length_of_hn = length(hn);
dimensions = length_of_xn + length_of_hn - 1;

number_of_zeros = dimensions - length_of_hn;

z = [hn(1) zeros(1,number_of_zeros)];
A = toeplitz([hn zeros(1,number_of_zeros)],z);
end
```

2. linearConvolution.m

```
function yn = linearConvolution(xn, hn)
    yn = toeplitzDSP(xn, hn)*xn';
end
```

Linear Convolution & Linear Convolution using Circular

```
clear all;
close all;
clc;


xn = input("Enter the first sequence: ");
hn = input("Enter the second sequence: ");

length_of_xn = length(xn);
length_of_hn = length(hn);

linearConvLength = length_of_xn + length_of_hn - 1;

xn = [xn zeros(1, linearConvLength - length_of_xn)];
hn = [hn zeros(1, linearConvLength - length_of_hn)];

linearConv = circularConvolution(xn, hn).';

disp("x(n) and h(n) after making their length equal to length of Linear Conv.
sequence");
```

```
disp("x(n):");
disp(xn);
disp("h(n)");
disp(hn);

disp("Linear Convolution calculated using Circular Convolution is: ");
disp(linearConv);

%% PLOTS

widthOfLine = 1;

xAxis = 0:length(linearConv)-1;
% subplot(1, 2, 1);
localY = abs(linearConv);
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("k \rightarrow", "Amplitude", "Linear Convolution of $$ x(n) $$
with $$ h(n) $$ using Circular Convolution");
for index = xAxis
    localY1 = localY(index + 1);
    text(index, localY1, strcat("\leftarrow", num2str(round(localY1, 1))));
end
```

Circular Convolution & Circular Convolution using Linear

```
clear all;
close all;
clc;


xn = input("Enter the sequence 1: ");
hn = input("Enter the sequence 2: ");

LinearConv = linearConvolution(xn, hn).';
lengthOfCircConv = max(length(xn), length(hn));

circularConv = zeros(1,lengthOfCircConv);

for index = 0:length(LinearConv)-1
    index1 = mod(index, lengthOfCircConv) + 1;
    circularConv(index1) = circularConv(index1) + LinearConv(index + 1);
end

disp("Linear Convolution of x(n) with h(n) is: ");
disp(LinearConv);

disp("Circular Convolution using Linear Convolution is: ");
disp(circularConv);
```

```
%% PLOTS

widthOfLine = 1;

xAxis = 0:length(LinearConv)-1;
subplot(1, 2, 1);
localY = abs(LinearConv);
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("k \rightarrow", "Amplitude", "Linear Convolution of $$ x(n) $$
with $$ h(n) $$");
for index = xAxis
    localY1 = localY(index + 1);
    text(index, localY1, strcat("\leftarrow", num2str(round(localY1, 1))));
end

xAxis = 0:length(circularConv)-1;
subplot(1, 2, 2);
localY = abs(circularConv);
stem(xAxis, localY, "r", "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("k \rightarrow", "Amplitude", "Circular Convolution of $$ x(n)
$$ with $$ h(n) $$");
for index = xAxis
    localY1 = localY(index + 1);
    text(index, localY1, strcat("\leftarrow", num2str(round(localY1, 1))));
end
```

Time Reversal Property

```
% Time reversal property
% DFT[x(N-n)] is given by X(N-k)

clear all;
close all
clc;

% TAKING THE INPUT FROM THE USER
sequence = input("Enter the sequence: ");

N = length(sequence);
n = 1:N;
k = n;

widthOfLine = 1;

% MAIN SEQUENCE AND IT'S DFT
x(n) = sequence;
X(k) = dft(x(n));
```

```
% TIME REVERSED SEQUENCE AND IT'S DFT
x1(n) = reverse(sequence);
X1(k) = dft(x1(n));

% CONJUGATE OF THE MAIN SEQUENCE'S DFT
X2(k) = conj(X(k));

disp("DFT of x(n) is: ");
disp(round(X(k), 3));

disp('The time reversed sequence x(-n) is: ');
disp(round(x1(n), 3));

disp("DFT of time reversed sequence x(-n) is: ");
disp(round(X1(k), 3));

disp("Conjugate of X(k) is: ");
disp(round(X2(k), 3));

disp("The DFT of time reversed sequence is equal to the conjugate of x(n)");
disp("Thus the time reversal property is verified.")


% PLOTTING THE DFT OF MAIN SEQUENCE
xAxis = 0:N-1;
subplot(2, 3, 1);
localY = abs(X);
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("k \rightarrow", "Amplitude", "DFT of $$ x(n) \rightarrow X(k)
$$");
for index = xAxis
    localY1 = localY(index + 1);
    text(index, localY1, strcat("\leftarrow", num2str(round(localY1, 1))));
end

subplot(2, 3, 4);
localY = angle(round(X,5))*180/pi;
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("k \rightarrow", "Phase (°)", "DFT of $$ x(n) \rightarrow X(k)
$$");
for index = xAxis
    localY1 = localY(index + 1);
    text(index, localY1, strcat("\leftarrow", num2str(round(localY1, 1))));
end

% PLOTTING THE DFT OF TIME REVERSED SEQUENCE
xAxis = 0:N-1;
subplot(2, 3, 2);
localY = abs(X1);
stem(xAxis, localY, "lineWidth", widthOfLine);
```

```
setAxisLimits(axis);
setPlotAttributes("k \rightarrow", "Amplitude", "DFT of $$ x(-n) \rightarrow $$ $$
DFT[x(-n)] $$");
for index = xAxis
    localY1 = localY(index + 1);
    text(index, localY1, strcat("\leftarrow", num2str(round(localY1, 1))));
end

subplot(2, 3, 5);
localY = angle(round(X1,5))*180/pi;
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("k \rightarrow", "Phase (°)", "DFT of $$ x(-n) \rightarrow $$ $$
DFT[x(-n)] $$");
for index = xAxis
    localY1 = localY(index + 1);
    text(index, localY1, strcat("\leftarrow", num2str(round(localY1, 1))));
end

% PLOTTING THE CONJUGATE OF THE DFT OF MAIN SEQUENCE
xAxis = 0:N-1;
subplot(2, 3, 3);
localY = abs(X1);
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("k \rightarrow", "Amplitude", "Conjugate of DFT of $$ x(n)
\rightarrow X(N-k) $$");
for index = xAxis
    localY1 = localY(index + 1);
    text(index, localY1, strcat("\leftarrow", num2str(round(localY1, 1))));
end

subplot(2, 3, 6);
localY = angle(round(X1,5))*180/pi;
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("k \rightarrow", "Phase (°)", "Conjugate of DFT of $$ x(n)
\rightarrow X(N-k) $$");
for index = xAxis
    localY1 = localY(index + 1);
    text(index, localY1, strcat("\leftarrow", num2str(round(localY1, 1))));
end
```

Parseval's Theorem

```
% Parseval's Theorem defines energy in time domain
% is equivalent represented in frequency

clear all;
close all
```

```
clc;


sequence = input("Enter the sequence: ");
DFT = dft(sequence);
N = length(sequence);


energyInTime = abs(sum(sequence.*conj(sequence)));
energyInFreq = abs(sum(DFT.*conj(DFT))/N);

disp("Energy in time domain is: ");
disp(energyInTime);

disp("Energy in freq domain is: ");
disp(energyInFreq);

if round(energyInTime, 5) == round(energyInFreq, 5)
    disp("Thus the Parseval's theorem is verified...!");
end
```

Experiment 6: Radix

Code:

1. dit_fft.m

```
% 1. Radix 2 | DIT FFT Algorhithm


function [y, outputStages] = dit_fft(x)

    p = nextpow2(length(x));
    x = [x zeros(1, (2^p) - length(x))];
    N = length(x);
    totalStages = log2(N);
    Half = 1;
    x = bitrevorder(x);

    outputStages = zeros(N, 1);

    for stage = 1:totalStages;

        for index = 0:(2^stage):(N - 1);

            for n = 0:(Half - 1); % creating "butterfly" and saving the results
                pos = n + index + 1;
                pow = (2^(totalStages - stage)) * n;
                w = exp((-1i) * (2 * pi) * pow / N);
                a = x(pos) + x(pos + Half) .* w;
```

```
                b = x(pos) - x(pos + Half) .* w;
                x(pos) = a;
                x(pos + Half) = b;
            end

        end

        outputStages = [outputStages transpose(x)];
        Half = 2 * Half;
    end

    outputStages = outputStages(:, 2:end);

    y = transpose(x); % returning the result from function
end
```

2. dif_fft.m

```
function [y outputStages] = dif_fft(x)
    p = nextpow2(length(x));
    x = [x zeros(1, (2^p) - length(x))];
    N = length(x);
    S = log2(N);
    Half = N / 2;

    outputStages = zeros(N, 1);

    for stage = 1:S;

        for index = 0:(N / (2^(stage - 1))):(N - 1);

            for n = 0:(Half - 1);
                pos = n + index + 1;
                pow = (2^(stage - 1)) * n;
                w = exp((-1i) * (2 * pi) * pow / N);
                a = x(pos) + x(pos + Half);
                b = (x(pos) - x(pos + Half)) .* w;
                x(pos) = a;
                x(pos + Half) = b;
            end

        end

        outputStages = [outputStages transpose(x)];
        Half = Half / 2;
    end

    outputStages = outputStages(:, 2:end);
    y = transpose(bitrevorder(x));
end
```

Radix-2 DFT (DIT & DIF)

```
clear all;
close all;
clc;

sequence = input("Enter a sequence: ");

[DITFFT, ditOutputStages] = dit_fft(sequence);
[DIFFFT, difOutputStages] = dif_fft(sequence);

N = length(DIFFFT);

disp("Intermediate stages of DIT FFT");
disp(ditOutputStages);
disp("Final DFT using DIT algorhithm: ");
disp(DITFFT);


disp("Intermediate stages of DIF FFT");
disp(difOutputStages);
disp('Final DFT using DIF algorhithm: ');
disp(DIFFFT);

% PLOTS
widthOfLine = 1;

xAxis = 0:N-1;
subplot(2, 2, 1);
localY = abs(DIFFFT);
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("k \rightarrow", "Amplitude", "DFT of $$ x(n) \rightarrow X(k)
$$ using DIF FFT algorhithm");
for index = xAxis
    localY1 = localY(index + 1);
    text(index, localY1, strcat("\leftarrow", num2str(round(localY1, 1))));
end

subplot(2, 2, 2);
localY = angle(round(DIFFFT,5))*180/pi;
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("k \rightarrow", "Phase", "DFT of $$ x(n) \rightarrow X(k) $$
using DIF FFT algorhithm");
for index = xAxis
    localY1 = localY(index + 1);
    text(index, localY1, strcat("\leftarrow", num2str(round(localY1, 1))));
end
```

```
subplot(2, 2, 3);
localY = abs(fft(sequence));
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("k \rightarrow", "Amplitude", "DFT of $$ x(n) \rightarrow X(k)
$$ using built in MATLAB function");
for index = xAxis
    localY1 = localY(index + 1);
    text(index, localY1, strcat("\leftarrow", num2str(round(localY1, 1))));
end

subplot(2, 2, 4);
localY = angle(round(fft(sequence),5))*180/pi;
stem(xAxis, localY, "lineWidth", widthOfLine);
setAxisLimits(axis);
setPlotAttributes("k \rightarrow", "Phase", "DFT of $$ x(n) \rightarrow X(k) $$
using built in MATLAB function");
for index = xAxis
    localY1 = localY(index + 1);
    text(index, localY1, strcat("\leftarrow", num2str(round(localY1, 1))));
end
```

Radix-2 IDFT (DIT & DIF)

```
% clear all;
% close all;
% clc;

inputSequence = input("Enter a sequence: ");
sequence = conj(inputSequence);

[DITIFFT, ditOutputStages] = dit_fft(sequence);
[DIFIFFT, difOutputStages] = dif_fft(sequence);

DIFIFFT = DIFIFFT/length(DIFIFFT);
DITIFFT = DITIFFT/length(DITIFFT);

ditOutputStages(:,end) = ditOutputStages(:,end)/length(ditOutputStages(:,end));
difOutputStages(:,end) = difOutputStages(:,end)/length(difOutputStages(:,end));


disp("Intermediate stages of DIT FFT");
disp(ditOutputStages);
disp("Final IDFT using DIT algorhithm: ");
disp(conj(DITIFFT));


disp("Intermediate stages of DIF FFT");
disp(difOutputStages);
```

```
disp('Final IDFT using DIF algorhithm: ');
disp(conj(DIFIFFT));
```

Mix-Radix 2x3

```
clear all;
close all;
clc;

disp("MixRadix 6 point DIT-FFT 2x3");

x = input("Enter a 6 point sequence: ");
% x = [1 -1 2 -2 3 -3];
N = length(x);

X1 = zeros(3,1);
X2 = X1;

for k = 0:2
    for n = 0:2
        X1(k + 1) = X1(k + 1) + x((2 * n) + 1) * twiddleFactor(2*k*n, N);
        X2(k + 1) = X2(k + 1) + x((2 * n + 1) + 1) * twiddleFactor(2*k*n, N);
    end
end

outputStages = [X1; X2];

X = zeros(N, 1);
for k = 0:N-1
    X(k+1) = X1(mod(k,3) + 1) + X2(mod(k,3) + 1) * twiddleFactor(k,6);
end

outputStages = [outputStages X];

disp('Output of stages: ');
disp(outputStages);

disp("Final DFT using Mix Radix DIT FFT 2x3");
disp(X);

disp("DFT calculated using builtin-function: fft():")
disp(fft(x).');
```

Mix-Radix 3x2

```
% MIX RADIX ALGORHITHM FOR N=6
% 6 = 3 x 2 | M1 = 3 | M2 = 6
```

```
clear all;
close all;
clc;

disp("MixRadix 6 point DIT-FFT 3x2");

x = input("Enter a 6 point sequence: ");
N = length(x);

X1(1) = x(1) + x(4);
X1(2) = x(1) - x(4);
X2(1) = x(2) + x(5);
X2(2) = x(2) - x(5);
X3(1) = x(3) + x(6);
X3(2) = x(3) - x(6);

outputStages = [X1.'; X2.'; X3.'];
X = zeros(N, 1);

for k = 0:N - 1
    l = mod(k, 2) + 1;
    X(k + 1) = X1(l) + X2(l) * twiddleFactor(k, N) + X3(l) * twiddleFactor(2 * k,
N);
end

outputStages = [outputStages X];

disp('Output of stages: ');
disp(outputStages);

disp("Final DFT using Mix Radix DIT FFT 3x2");
disp(X);

disp("DFT calculated using builtin-function: fft():")
disp(fft(x).');
```

Experiment 7: CCS

Circular Convolution

Time Domain

```
#include <stdio.h>

int m, n, x[30], h[30], y[30], i, j, k, x2[30], a[30];

void main()
{
    printf("Enter the length of the first sequence: ");
    scanf("%d", &m);
```

```
    printf("Enter the length of the second sequence: ");
    scanf("%d", &n);

    printf("Enter the first sequence: ");
    for (i = 0; i < m; i++)
        scanf("%d", &x[i]);

    printf("Enter the second sequence: ");
    for (j = 0; j < n; j++)
        scanf("%d", &h[j]);

    if (m - n != 0) /*If length of both sequences are not equal*/
    {
        if (m > n) /* Pad the smaller sequence with zero*/
        {
            for (i = n; i < m; i++)
                h[i] = 0;
            n = m;
        }
        for (i = m; i < n; i++)
            x[i] = 0;
        m = n;
    }

    y[0] = 0;
    a[0] = h[0];

    for (j = 1; j < n; j++) /*folding h(n) to h(-n)*/
        a[j] = h[n - j];

    /*Circular convolution*/
    for (i = 0; i < n; i++)
        y[0] += x[i] * a[i];

    for (k = 1; k < n; k++)
    {
        y[k] = 0;
        /*circular shift*/

        for (j = 1; j < n; j++)
            x2[j] = a[j - 1];
        x2[0] = a[n - 1];
        for (i = 0; i < n; i++)
        {
            a[i] = x2[i];
            y[k] += x[i] * x2[i];
        }
    }

    /*displaying the result*/
    printf("Circular convolution of x(n) and h(n) is:\n");
    for (i = 0; i < n; i++)
```

```
        printf("y[%d]=\t%d\n", i, y[i]);
}
```

Frequency Domain

```
#include <stdio.h>
int x[20], h[20], y[7];
int main(void)
{
    int i, j, m, n, N;
    printf("Enter the size of x(n): ");
    scanf("%d", &m);
    printf("Enter the size of h(n): ");
    scanf("%d", &n);
    printf("Enter the values of x(n): ");
    for (i = 0; i < m; i++)
    {
        scanf("%d", &x[i]);
    }
    printf("Enter the values of h(n): ");
    for (i = 0; i < n; i++)
    {
        scanf("%d", &h[i]);
    }
    N = m > n ? m : n;
    // Zero Padding the sequences
    for (i = m; i < N; i++)
    {
        x[i] = 0;
    }
    for (i = n; i < N; i++)
    {
        h[i] = 0;
    }
    // Circ Conv
    for (i = 0; i < N; i++)
    {
        for (j = 0; j <= N; j++)
        {
            y[(i + j) % N] = y[(i + j) % N] + (x[i] * h[j]);
        }
    }
    for (i = 0; i < N; i++)
    {
        printf("y[%d]=\t%d\n",i, y[i]);
    }
    return 0;
}
```

Linear Convolution

```c
#include <stdio.h>

int x[20], h[20], y[20];

int main()
{
    int i, j, m, n;

    printf("Enter length of x(n): ");
    scanf("%d", &m); // Getting Length of x(n)
    printf("\nEnter the length of h(n): ");
    scanf("%d", &n); // Getting Length of h(n)

    printf("\nEnter x(n): ");

    for (i = 0; i < m; i++)
    {
        scanf("%d", &x[i]); // Getting x(n)
    }

    printf("\nEnter h(n): ");
    for (i = 0; i < n; i++)
    {
        scanf("%d", &h[i]); // Getting h(n)
    }


    for (i = 0; i < m + n - 1; i++)
    {
        y[i] = 0;
        for (j = 0; j <= i; j++)
        {
            y[i] = y[i] + (x[j] * h[i - j]); // Calculating Linear Convolution
        }
    }

    //displaying the o/p
    for (i = 0; i < m + n - 1; i++)
    {
        printf("The Value of output y[%d]=%d\n", i, y[i]); // Printing it to
STDOUT
    }
    printf("\n\n");
}
```

DFT

```c
#include<stdio.h>
#include<math.h>
/*
 * main.c
 */

int main(void) {
    int N,k,n,i;
    float pi=3.1416,sumre=0,sumim=0,out_real[8]={0,0},out_imag[8]={0,0};
    int x[32];
    printf("Enter the length of sequence: ");
    scanf("%d", &N);
    printf("Enter the sequence: ");
    for(i=0;i<N;i++){
        scanf("%d",&x[i]);
    }
    for(k=0;k<N;k++){
        sumre=0;
        sumim=0;
        for(n=0;n<N;n++){
            sumre=sumre+x[n]*cos(2*pi*k*n/N);
            sumim=sumim-x[n]*sin(2*pi*k*n/N);
        }
        out_real[k]=sumre;
        out_imag[k]=sumim;
        printf("x([%d])=\t%f\t+\t%fj\n",k,out_real[k],out_imag[k]);
    }

    return 0;
}
```

IDFT

```c
#include<stdio.h>
#include<math.h>
/*
 * main.c
 */

int main(void) {
    int N,k,n,i;
    float pi=3.1416,sumre=0,sumim=0,out_real[8]={0,0},out_imag[8]={0,0};
    int x[32];
    printf("Enter the length of sequence: ");
    scanf("%d", &N);
    printf("Enter the sequence: ");
    for(i=0;i<N;i++){
        scanf("%d",&x[i]);
    }
```

```
    for(k=0;k<N;k++){
        sumre=0;
        sumim=0;
        for(n=0;n<N;n++){
            sumre=sumre+x[n]*cos(2*pi*k*n/N);
            sumim=sumim-x[n]*sin(2*pi*k*n/N);
        }
        out_real[k]=sumre;
        out_imag[k]=sumim;
        printf("x([%d])=\t%f\t+\t%fj\n",k,out_real[k]/8,-out_imag[k]/8);
    }

    return 0;
}
```

Correlation

```
// Code for corelation of two sequences

#include <stdio.h>

int x[20], h[20], y[20], z[20], result;

int main()
{
    int i, j, m, n;

    printf("For Auto-Corelation, enter the same sequence twice.\n");

    printf("Enter length of x(n): ");
    scanf("%d", &m); // Getting Length of x(n)
    printf("Enter the length of h(n): ");
    scanf("%d", &n); // Getting Length of h(n)

    printf("Enter x(n): ");

    for (i = 0; i < m; i++)
    {
        scanf("%d", &x[i]); // Getting x(n)
    }

    printf("Enter h(n): ");
    for (i = n - 1; i >= 0; i--)
    {
        scanf("%d", &h[i]); // Getting h(n)
    }

    for (i = 0; i < m + n - 1; i++)
    {
        y[i] = 0;
```

```
        for (j = 0; j <= i; j++)
        {
            y[i] = y[i] + (x[j] * h[i - j]); // Calculating Co relation
        }
    }

    //displaying the o/p
    for (i = 0; i < m + n - 1; i++)
    {
        result = y[i];
        printf("y[%d]=%d\n", i, y[i]); // Printing it to STDOUT
    }
    printf("\n");
}
```

Experiment 8: Digital Filter Design

Butterworth IIT

```
clear all;
close all;
clc;

gainType = input("Select Gain Type:\n1 for Normal\n2 for Decibels: ");

minPassBandGain = input("Enter Passband Gain: ");
maxStopBandGain = input("Enter Stopband Gain: ");

frequencyType = input("Frequency Unit:\n1 for Hz\n2 for Rad/s: ");

passBandFrequency = input('Enter Passband Frequency (Analog Domain): ');
stopBandFrequency = input('Enter Stopband Frequency (Analog Domain): ');

samplingTime = input('Enter the sampling time: ');
disp('Low Pass => 1');
disp('High Pass => 2');
disp('Band Pass => 3');
disp('Band Stop => 4');

filterType = input("Enter filter type: ")

% minPassBandGain = 1.9328;
% maxStopBandGain = 13.9794;
% passBandFrequency = 0.62;
% stopBandFrequency = 1.88;
% samplingTime = 1;
% gainType = 2;
% frequencyType = 2;

if gainType == 1
```

```
    minPassBandGain = -20 * log10(minPassBandGain);
    maxStopBandGain = -20 * log10(maxStopBandGain);
end

if frequencyType == 1
    passBandFrequency = 2 * pi * passBandFrequency;
    stopBandFrequency = 2 * pi * stopBandFrequency;
end

% Finding all the parameters needed in analog domain
selectedOrder = ceil(log10((10^(0.1 * maxStopBandGain) - 1) / (10^(0.1 *
minPassBandGain) - 1)) / (2 * log10(stopBandFrequency / passBandFrequency)));
omegaCp = passBandFrequency / (10^(0.1 * minPassBandGain) - 1)^(1 / (2 *
selectedOrder));
omegaCs = stopBandFrequency / (10^(0.1 * maxStopBandGain) - 1)^(1 / (2 *
selectedOrder));
omegaC = mean([omegaCp omegaCs]);

% Getting transfer function coefficients for analog butterworth filter
[normallisedAnalogNumerator, normallisedAnalogDenominator] = butter(selectedOrder,
1, "low", "s"); % Normallised

% Getting the actual transfer function
normallisedAnalogTransferFunction = tf(normallisedAnalogNumerator,
normallisedAnalogDenominator); % Normallised

switch filterType
    case 1
        % low pass
        [deNormallisedAnalogNumerator, deNormallisedAnalogDenominator] =
butter(selectedOrder, omegaC, "low", "s"); % Denormallised
        [digitalNumerator, digitalDenominator] =
impinvar(deNormallisedAnalogNumerator, deNormallisedAnalogDenominator, 1 /
samplingTime);
    case 2
        % High Pass
        [deNormallisedAnalogNumerator, deNormallisedAnalogDenominator] =
butter(selectedOrder, omegaC, "high", "s"); % Denormallised
        digitalCutoff = 2 * omegaC;
        [digitalNumerator, digitalDenominator] = butter(selectedOrder,
digitalCutoff / pi, 'high');
        % [digitalNumerator, digitalDenominator] =
impinvar(deNormallisedAnalogNumerator, deNormallisedAnalogDenominator, 1 /
samplingTime);
    case 3
        % Band Pass
        [deNormallisedAnalogNumerator, deNormallisedAnalogDenominator] =
butter(selectedOrder, [omegaCp omegaCs], "bandpass", "s"); % Denormallised
        digitalCutoff = 2 * [omegaCp omegaCs];
        [digitalNumerator, digitalDenominator] = butter(selectedOrder,
digitalCutoff / pi, 'bandpass');
    case 4
        % Band stop
```

```
        [deNormallisedAnalogNumerator, deNormallisedAnalogDenominator] =
butter(selectedOrder, [omegaCp omegaCs], "stop", "s"); % Denormallised
        digitalCutoff = 2 * [omegaCp omegaCs];
        [digitalNumerator, digitalDenominator] = butter(selectedOrder,
digitalCutoff / pi, 'stop');
end

deNormallisedAnalogTransferFunction = tf(deNormallisedAnalogNumerator,
deNormallisedAnalogDenominator); % Denormallised
digitalTransferFunction = tf(digitalNumerator, digitalDenominator, 1 /
samplingTime);

% Frequency Response of digital filter
[h, w] = freqz(digitalNumerator, digitalDenominator, 1024);

% Displaying the values in MATLAB command window
disp("OmegaCp:");
disp(omegaCp); % omegaCp
disp("OmegaCs:");
disp(omegaCs); % omegaCs
disp("OmegaC:");
disp(omegaC); % omegaCs

disp('Calculated Order is:');
disp(selectedOrder); % Order N

disp('Normallised H(s) is:');
normallisedAnalogTransferFunction% Show normallised analog filter

disp('Denormallised H(s) is:');
deNormallisedAnalogTransferFunction% Show de-normallised analog filter

disp('H(z) is:');
digitalTransferFunction% Show digital filter transfer function

plot(w / pi, 20 * log10(abs(h))); % plot the frequency response
% w => ranges from 0 to pi so dividing by pi gives x axis ranging from 0 to 1
% 20*log10 => to convert the gain to db
grid on;
% axis([0 22/7 -inf inf]);
setAxisLimits(axis);
setPlotAttributes("Frequency in r/s", "Gain in db", "Frequency response of Digital
Butterworth Filter");
```

Butterworth BLT

```
clear all;
close all;
clc;
```

```
% minPassBandGain = 1.9328;
% maxStopBandGain = 13.9794;
% passBandFrequency = 0.62;
% stopBandFrequency = 1.88;
% samplingTime = 1;
% gainType = 2;
% frequencyType = 2;

gainType = input("Select Gain Type: 1 for Normal, 2 for Decibels: ");

minPassBandGain = input("Enter Passband Gain: ");
maxStopBandGain = input("Enter Stopband Gain: ");

frequencyType = input("Frequency Unit: 1 for Hz, 2 for Rad/s: ");

passBandFrequency = input('Enter Passband Frequency (Analog Domain): ');
stopBandFrequency = input('Enter Stopband Frequency (Analog Domain): ');

samplingTime = input('Enter the sampling time: ');
disp('Low Pass => 1');
disp('High Pass => 2');
disp('Band Pass => 3');
disp('Band Stop => 4');

filterType = input("Enter filter type: ")

if gainType == 1
    minPassBandGain = -20 * log10(minPassBandGain);
    maxStopBandGain = -20 * log10(maxStopBandGain);
end

if frequencyType == 1
    passBandFrequency = 2 * pi * passBandFrequency;
    stopBandFrequency = 2 * pi * stopBandFrequency;
end

% Finding all the parameters needed in analog domain
selectedOrder = ceil(log10((10^(0.1 * maxStopBandGain) - 1) / (10^(0.1 *
minPassBandGain) - 1)) / (2 * log10(stopBandFrequency / passBandFrequency)));
omegaCp = passBandFrequency / (10^(0.1 * minPassBandGain) - 1)^(1 / (2 *
selectedOrder));
omegaCs = stopBandFrequency / (10^(0.1 * maxStopBandGain) - 1)^(1 / (2 *
selectedOrder));
omegaC = mean([omegaCp omegaCs]);

% Getting transfer function coefficients for analog butterworth filter
[normallisedAnalogNumerator, normallisedAnalogDenominator] = butter(selectedOrder,
1, "low", "s"); % Normallised

% Getting the actual transfer function
normallisedAnalogTransferFunction = tf(normallisedAnalogNumerator,
normallisedAnalogDenominator); % Normallised
```

```
switch filterType
    case 1
        % low pass
        [deNormallisedAnalogNumerator, deNormallisedAnalogDenominator] =
butter(selectedOrder, omegaC, "low", "s"); % Denormallised
        [digitalNumerator, digitalDenominator] =
bilinear(deNormallisedAnalogNumerator, deNormallisedAnalogDenominator, 1 /
samplingTime);
    case 2
        % High Pass
        [deNormallisedAnalogNumerator, deNormallisedAnalogDenominator] =
butter(selectedOrder, omegaC, "high", "s"); % Denormallised
        digitalCutoff = 2 * omegaC;
        [digitalNumerator, digitalDenominator] =
bilinear(deNormallisedAnalogNumerator, deNormallisedAnalogDenominator, 1 /
samplingTime);
    case 3
        % Band Pass
        [deNormallisedAnalogNumerator, deNormallisedAnalogDenominator] =
butter(selectedOrder, [omegaCp omegaCs], "bandpass", "s"); % Denormallised
        digitalCutoff = 2 * [omegaCp omegaCs];
        [digitalNumerator, digitalDenominator] = butter(selectedOrder,
digitalCutoff / pi, 'bandpass');
    case 4
        % Band stop
        [deNormallisedAnalogNumerator, deNormallisedAnalogDenominator] =
butter(selectedOrder, [omegaCp omegaCs], "stop", "s"); % Denormallised
        digitalCutoff = 2 * [omegaCp omegaCs];
        [digitalNumerator, digitalDenominator] = butter(selectedOrder,
digitalCutoff / pi, 'stop');
end

deNormallisedAnalogTransferFunction = tf(deNormallisedAnalogNumerator,
deNormallisedAnalogDenominator); % Denormallised
digitalTransferFunction = tf(digitalNumerator, digitalDenominator, 1 /
samplingTime);

% Frequency Response of digital filter
[h, w] = freqz(digitalNumerator, digitalDenominator, 1024);

% Displaying the values in MATLAB command window
disp("OmegaCp:");
disp(omegaCp); % omegaCp
disp("OmegaCs:");
disp(omegaCs); % omegaCs
disp("OmegaC:");
disp(omegaC); % omegaCs

disp('Calculated Order is:');
disp(selectedOrder); % Order N

disp('Normallised H(s) is:');
normallisedAnalogTransferFunction% Show normallised analog filter
```

```
disp('Denormallised H(s) is:');
deNormallisedAnalogTransferFunction% Show de-normallised analog filter

disp('H(z) is:');
digitalTransferFunction% Show digital filter transfer function

plot(w / pi, 20 * log10(abs(h))); % plot the frequency response
% w => ranges from 0 to pi so dividing by pi gives x axis ranging from 0 to 1
% 20*log10 => to convert the gain to db
grid on;
% axis([0 22/7 -inf inf]);
setAxisLimits(axis);
setPlotAttributes("Frequency in r/s", "Gain in db", "Frequency response of Digital
Butterworth Filter");
```

Chebyschev IIT

```
clear all;
close all;
clc;

% minPassBandGain = 0.75;
% maxStopBandGain = 0.23;
% % passBandFrequency = 0.62;
% % stopBandFrequency = 1.88;
% digitalPassBandFrequency = 0.25 * pi;
% digitalStopBandFrequency = 0.63 * pi;
% samplingTime = 2;
% gainType = 1;
% frequencyType = 2;
% % digitalPassBandFrequency = 2 * atan(samplingTime * passBandFrequency / 2);
% % digitalStopBandFrequency = 2 * atan(samplingTime * stopBandFrequency / 2);
% passBandFrequency = (2 * tan(digitalPassBandFrequency / 2)) / samplingTime;
% stopBandFrequency = (2 * tan(digitalStopBandFrequency / 2)) / samplingTime;

disp('Enter the frequencies type');
inputDomain = input("Enter 1 for analog frequencies, 2 for digital: ");

samplingTime = input('Enter the sampling time: ');
gainType = input("Select Gain Type: 1 for Normal, 2 for Decibels: ");

minPassBandGain = input("Enter Passband Gain: ");
maxStopBandGain = input("Enter Stopband Gain: ");

frequencyType = input("Frequency Unit: 1 for Hz, 2 for Rad/s: ");

if inputDomain == 1
    passBandFrequency = input('Enter Passband Frequency (Analog Domain): ');
    stopBandFrequency = input('Enter Stopband Frequency (Analog Domain): ');
```

```
    if frequencyType == 1
        passBandFrequency = 2 * pi * passBandFrequency;
        stopBandFrequency = 2 * pi * stopBandFrequency;
    end

    digitalPassBandFrequency = 2 * atan(samplingTime * passBandFrequency / 2);
    digitalStopBandFrequency = 2 * atan(samplingTime * stopBandFrequency / 2);
else
    digitalPassBandFrequency = input('Enter Passband Frequency (Digital Domain):
');
    digitalStopBandFrequency = input('Enter Stopband Frequency (Digital Domain):
');

    if frequencyType == 1
        digitalPassBandFrequency = 2 * pi * digitalPassBandFrequency;
        digitalStopBandFrequency = 2 * pi * digitalStopBandFrequency;
    end

    passBandFrequency = (2 * tan(digitalPassBandFrequency / 2)) / samplingTime;
    stopBandFrequency = (2 * tan(digitalStopBandFrequency / 2)) / samplingTime;
end

disp('Low Pass => 1');
disp('High Pass => 2');
disp('Band Pass => 3');
disp('Band Stop => 4');

filterType = input("Enter filter type: ");

if gainType == 1
    minPassBandGain = -20 * log10(minPassBandGain);
    maxStopBandGain = -20 * log10(maxStopBandGain);
end

normallisedPassBandFrequency = passBandFrequency / passBandFrequency;
normallisedStopBandFrequency = stopBandFrequency / passBandFrequency;

epsilon = sqrt((10^(0.1 * minPassBandGain)) - 1);

selectedOrder = ceil((maxStopBandGain - 20 * log10(epsilon) + 6) / (6 + 20 *
log10(normallisedStopBandFrequency)));

switch filterType
    case 1
        % low pass
        [normallisedAnalogNumerator, normallisedAnalogDenominator] =
cheby1(selectedOrder, minPassBandGain, normallisedPassBandFrequency, "low", "s");
        [digitalNumerator, digitalDenominator] =
cheby1(normallisedAnalogNumerator, normallisedAnalogDenominator, 1 /
samplingTime);
    case 2
        % High Pass
```

```
        [normallisedAnalogNumerator, normallisedAnalogDenominator] =
cheby1(selectedOrder, minPassBandGain, normallisedPassBandFrequency, "high", "s");
        [digitalNumerator, digitalDenominator] = cheby1(selectedOrder,
minPassBandGain, digitalPassBandFrequency / pi, "high");
    case 3
        % Band Pass
        [normallisedAnalogNumerator, normallisedAnalogDenominator] =
cheby1(selectedOrder, minPassBandGain, [normallisedPassBandFrequency
normallisedStopBandFrequency], "bandpass", "s");
        [digitalNumerator, digitalDenominator] = cheby1(selectedOrder,
minPassBandGain, [digitalPassBandFrequency, digitalStopBandFrequency] / pi,
"bandpass");
    case 4
        % Band stop
        [normallisedAnalogNumerator, normallisedAnalogDenominator] =
cheby1(selectedOrder, minPassBandGain, [normallisedPassBandFrequency
normallisedStopBandFrequency], "stop", "s");
        [digitalNumerator, digitalDenominator] = cheby1(selectedOrder,
minPassBandGain, [digitalPassBandFrequency, digitalStopBandFrequency] / pi,
"stop");
end

normallisedAnalogTransferFunction = tf(normallisedAnalogNumerator,
normallisedAnalogDenominator);
digitalTransferFunction = tf(digitalNumerator, digitalDenominator, 1 /
samplingTime);

[h, w] = freqz(digitalNumerator, digitalDenominator, 1024);

% Display Given
disp("Given: ")
disp('   Min Pass Band Gain - Ap: (in db)');
disp(minPassBandGain);
disp('   Max Stop Band Gain - As: (in db)');
disp(maxStopBandGain);
disp('   Pass Band Frequency (Analog Domain) in rad/s');
disp(passBandFrequency);
disp('   Stop Band Frequency (Analog Domain) in rad/s');
disp(stopBandFrequency);

% Display Normallised Frequencies
disp('Normallised Frequencies')
disp('   Normallised Pass Band Frequency: ');
disp(normallisedPassBandFrequency);
disp('   Normallised Stop Band Frequency: ');
disp(normallisedStopBandFrequency);

% Display Ripple Factor
disp('Epsilon => ε: ');
disp(epsilon);

% DIsplay Order N
disp('Order => N');
```

```
disp(selectedOrder);

% Display Normallised Transferfunction
disp('Normallised Analog Transfer Function H1(s)');
normallisedAnalogTransferFunction

% Display Digital butterworth filter
digitalTransferFunction

plot(w / pi, 20 * log10(abs(h))); % plot the frequency response
% w => ranges from 0 to pi so dividing by pi gives x axis ranging from 0 to 1
% 20*log10 => to convert the gain to db
grid on;
setAxisLimits(axis);
setPlotAttributes("Frequency in r/s", "Gain in db", "Frequency response of Digital
Chebyschev Filter");
```

Chebyschev BLT

```
clear all;
close all;
clc;

% minPassBandGain = 0.75;
% maxStopBandGain = 0.23;
% % passBandFrequency = 0.62;
% % stopBandFrequency = 1.88;
% digitalPassBandFrequency = 0.25 * pi;
% digitalStopBandFrequency = 0.63 * pi;
% samplingTime = 2;
% gainType = 1;
% frequencyType = 2;
% % digitalPassBandFrequency = 2 * atan(samplingTime * passBandFrequency / 2);
% % digitalStopBandFrequency = 2 * atan(samplingTime * stopBandFrequency / 2);
% passBandFrequency = (2 * tan(digitalPassBandFrequency / 2)) / samplingTime;
% stopBandFrequency = (2 * tan(digitalStopBandFrequency / 2)) / samplingTime;

disp('Enter the frequencies type');
inputDomain = input("Enter 1 for analog frequencies, 2 for digital: ");

samplingTime = input('Enter the sampling time: ');
gainType = input("Select Gain Type: 1 for Normal, 2 for Decibels: ");

minPassBandGain = input("Enter Passband Gain: ");
maxStopBandGain = input("Enter Stopband Gain: ");

frequencyType = input("Frequency Unit: 1 for Hz, 2 for Rad/s: ");

if inputDomain == 1
    passBandFrequency = input('Enter Passband Frequency (Analog Domain): ');
```

```
    stopBandFrequency = input('Enter Stopband Frequency (Analog Domain): ');

    if frequencyType == 1
        passBandFrequency = 2 * pi * passBandFrequency;
        stopBandFrequency = 2 * pi * stopBandFrequency;
    end

    digitalPassBandFrequency = 2 * atan(samplingTime * passBandFrequency / 2);
    digitalStopBandFrequency = 2 * atan(samplingTime * stopBandFrequency / 2);
else
    digitalPassBandFrequency = input('Enter Passband Frequency (Digital Domain):
');
    digitalStopBandFrequency = input('Enter Stopband Frequency (Digital Domain):
');

    if frequencyType == 1
        digitalPassBandFrequency = 2 * pi * digitalPassBandFrequency;
        digitalStopBandFrequency = 2 * pi * digitalStopBandFrequency;
    end

    passBandFrequency = (2 * tan(digitalPassBandFrequency / 2)) / samplingTime;
    stopBandFrequency = (2 * tan(digitalStopBandFrequency / 2)) / samplingTime;
end

disp('Low Pass => 1');
disp('High Pass => 2');
disp('Band Pass => 3');
disp('Band Stop => 4');

filterType = input("Enter filter type: ");

if gainType == 1
    minPassBandGain = -20 * log10(minPassBandGain);
    maxStopBandGain = -20 * log10(maxStopBandGain);
end

normallisedPassBandFrequency = passBandFrequency / passBandFrequency;
normallisedStopBandFrequency = stopBandFrequency / passBandFrequency;

epsilon = sqrt((10^(0.1 * minPassBandGain)) - 1);

selectedOrder = ceil((maxStopBandGain - 20 * log10(epsilon) + 6) / (6 + 20 *
log10(normallisedStopBandFrequency)));

switch filterType
    case 1
        % low pass
        [normallisedAnalogNumerator, normallisedAnalogDenominator] =
cheby1(selectedOrder, minPassBandGain, normallisedPassBandFrequency, "low", "s");
        [digitalNumerator, digitalDenominator] = cheby1(selectedOrder,
minPassBandGain, digitalPassBandFrequency / pi, "low");
    case 2
        % High Pass
```

```
        [normallisedAnalogNumerator, normallisedAnalogDenominator] =
cheby1(selectedOrder, minPassBandGain, normallisedPassBandFrequency, "high", "s");
        [digitalNumerator, digitalDenominator] = cheby1(selectedOrder,
minPassBandGain, digitalPassBandFrequency / pi, "high");
    case 3
        % Band Pass
        [normallisedAnalogNumerator, normallisedAnalogDenominator] =
cheby1(selectedOrder, minPassBandGain, [normallisedPassBandFrequency
normallisedStopBandFrequency], "bandpass", "s");
        [digitalNumerator, digitalDenominator] = cheby1(selectedOrder,
minPassBandGain, [digitalPassBandFrequency, digitalStopBandFrequency] / pi,
"bandpass");
    case 4
        % Band stop
        [normallisedAnalogNumerator, normallisedAnalogDenominator] =
cheby1(selectedOrder, minPassBandGain, [normallisedPassBandFrequency
normallisedStopBandFrequency], "bandpass", "s");
        [digitalNumerator, digitalDenominator] = cheby1(selectedOrder,
minPassBandGain, [digitalPassBandFrequency, digitalStopBandFrequency] / pi,
"stop");
end

normallisedAnalogTransferFunction = tf(normallisedAnalogNumerator,
normallisedAnalogDenominator);
digitalTransferFunction = tf(digitalNumerator, digitalDenominator, 1 /
samplingTime);

[h, w] = freqz(digitalNumerator, digitalDenominator, 1024);

% Display Given
disp("Given: ")
disp('   Min Pass Band Gain - Ap: (in db)');
disp(minPassBandGain);
disp('   Max Stop Band Gain - As: (in db)');
disp(maxStopBandGain);
disp('   Pass Band Frequency (Analog Domain) in rad/s');
disp(passBandFrequency);
disp('   Stop Band Frequency (Analog Domain) in rad/s');
disp(stopBandFrequency);

% Display Normallised Frequencies
disp('Normallised Frequencies')
disp('   Normallised Pass Band Frequency: ');
disp(normallisedPassBandFrequency);
disp('   Normallised Stop Band Frequency: ');
disp(normallisedStopBandFrequency);

% Display Ripple Factor
disp('Epsilon => ε: ');
disp(epsilon);

% DIsplay Order N
disp('Order => N');
```

```
disp(selectedOrder);

% Display Normallised Transferfunction
disp('Normallised Analog Transfer Function H1(s)');
normallisedAnalogTransferFunction

% Display Digital butterworth filter
digitalTransferFunction

plot(w / pi, 20 * log10(abs(h))); % plot the frequency response
% w => ranges from 0 to pi so dividing by pi gives x axis ranging from 0 to 1
% 20*log10 => to convert the gain to db
grid on;
setAxisLimits(axis);
setPlotAttributes("Frequency in r/s", "Gain in db", "Frequency response of Digital
Chebyschev Filter");
```

Hanning Filter

```
lengthofsequence = input('Enter the length of Hanning filter: ');
Ftype = input("Select An option for which ftype filter\n1. Low pass filter\n2.
High pass \n3. Bandpass filter\n4. Stopband filter");

switch Ftype
    case 1
        Cuttofffreq = input('Enter the Cuttoff Frequency: ');
        hanningFilterOutput = fir1(lengthofsequence, Cuttofffreq / pi, "low",
hann(lengthofsequence + 1));
        freqz(hanningFilterOutput, 1, 1024);
    case 2
        Cuttofffreq = input('Enter the Cuttoff Frequency: ');
        hanningFilterOutput = fir1(lengthofsequence, Cuttofffreq / pi, "high",
hann(lengthofsequence + 1));
        freqz(hanningFilterOutput, 1, 1024);
    case 3
        FirstCutofffreq = input('Enter the Cutoff freq Omega(L): ');
        SecondCuttofffreq = input('Enter the cutoff freq Omega(U): ');
        Cuttofffreq = [FirstCutofffreq SecondCuttofffreq]
        hanningFilterOutput = fir1(lengthofsequence, Cuttofffreq / pi, "bandpass",
hann(lengthofsequence + 1));
        freqz(hanningFilterOutput, 1, 1024);
    case 4
        typeOfFilter = "stop"
        FirstCutofffreq = input('Enter the Cutoff freq Omega(L): ');
        SecondCuttofffreq = input('Enter the cutoff freq Omega(U): ');
        Cuttofffreq = [FirstCutofffreq SecondCuttofffreq]
        hanningFilterOutput = fir1(lengthofsequence, Cuttofffreq / pi, "stop",
hann(lengthofsequence + 1));
        freqz(hanningFilterOutput, 1, 1024);
end
```

Hamming Filter

```
lengthofsequence = input('Enter the length of Hamming Filter: ');
Ftype = input("Select An option for which ftype filter\n1. Low pass filter\n2.
High pass \n3. Bandpass filter\n4. Stopband filter: ");

switch Ftype
    case 1
        Cuttofffreq = input('Enter the Cuttoff Frequency: ');
        hammingFilterOutput = fir1(lengthofsequence, Cuttofffreq / pi, "low",
hamming(lengthofsequence + 1));
        freqz(hammingFilterOutput, 1, 1024);
    case 2
        Cuttofffreq = input('Enter the Cuttoff Frequency: ');
        hammingFilterOutput = fir1(lengthofsequence, Cuttofffreq / pi, "high",
hamming(lengthofsequence + 1));
        freqz(hammingFilterOutput, 1, 1024);
    case 3
        FirstCutofffreq = input('Enter the Cutoff freq Omega(L): ');
        SecondCuttofffreq = input('Enter the cutoff freq Omega(U): ');
        Cuttofffreq = [FirstCutofffreq SecondCuttofffreq]
        hammingFilterOutput = fir1(lengthofsequence, Cuttofffreq / pi, "bandpass",
hamming(lengthofsequence + 1));
        freqz(hammingFilterOutput, 1, 1024);
    case 4
        typeOfFilter = "stop"
        FirstCutofffreq = input('Enter the Cutoff freq Omega(L): ');
        SecondCuttofffreq = input('Enter the cutoff freq Omega(U): ');
        Cuttofffreq = [FirstCutofffreq SecondCuttofffreq]
        hammingFilterOutput = fir1(lengthofsequence, Cuttofffreq / pi, "stop",
hamming(lengthofsequence + 1));
        freqz(hammingFilterOutput, 1, 1024);
end
```