Educate Elevate Enlighten

JSS Mahavidyapeetha

JSS Science And Technology University

(Established Under JSS Science and Technology University Act No. 43 of 2013)

(Formerly Known as SJCE)

JSS SCIENCE AND TECHNOLOGY UNIVERSITY MYSURU

"Lab Experiment Report"

Submitted for the fulfillment of the CIE for the course

DSP Lab

(EC57L)

Submitted by

NAME	USN
Mohamed Farhan Fazal	01JST18EC055

Under the guidance of

SUJATHAKUMARI B A
Associate Professor

&

SUPREETHA M
Assistant Professor

DEPARTMENT OF ELECTRONICS AND COMMUNICATION

SJCE MYSURU- 570006

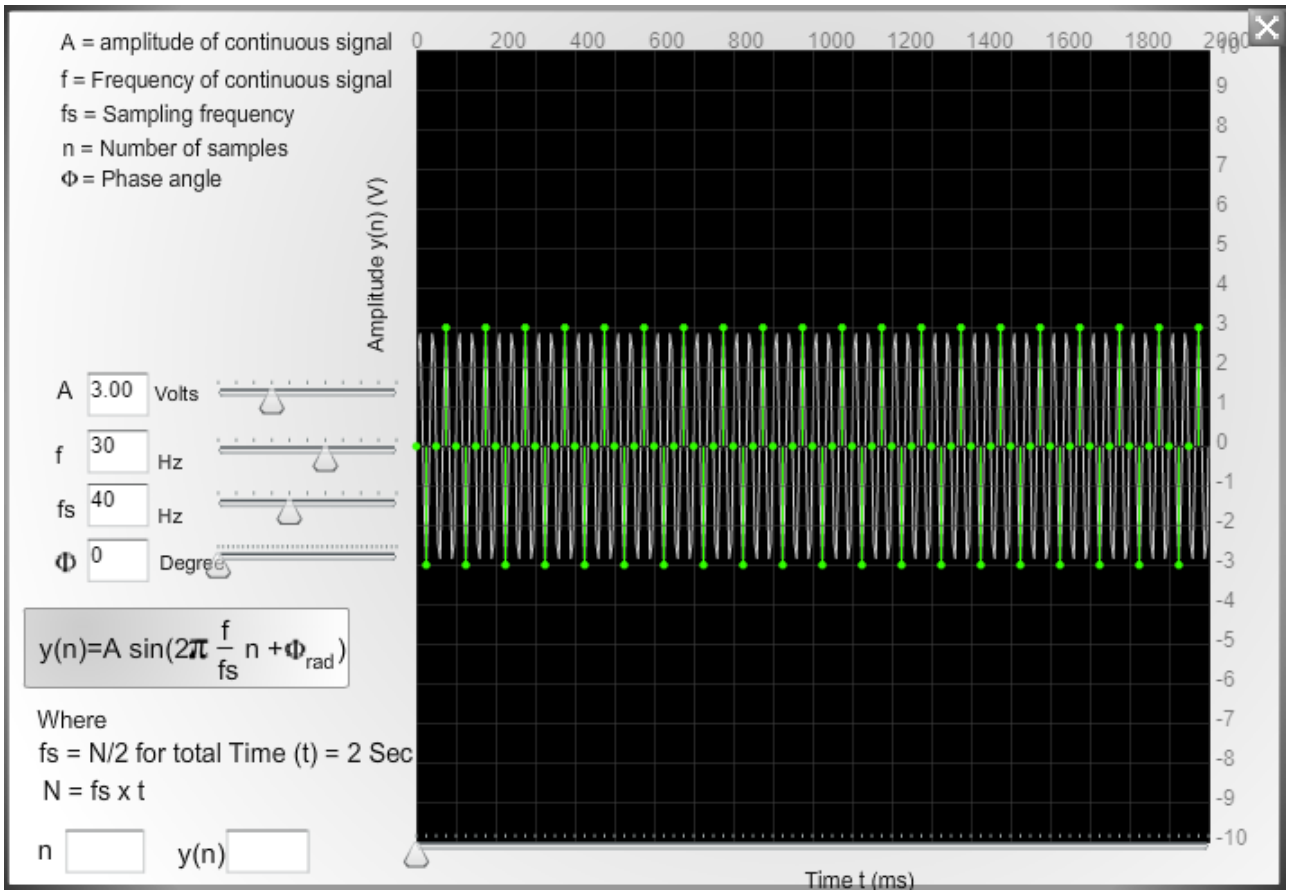
EC57L DSP - LAB

Lab Experiments' Report

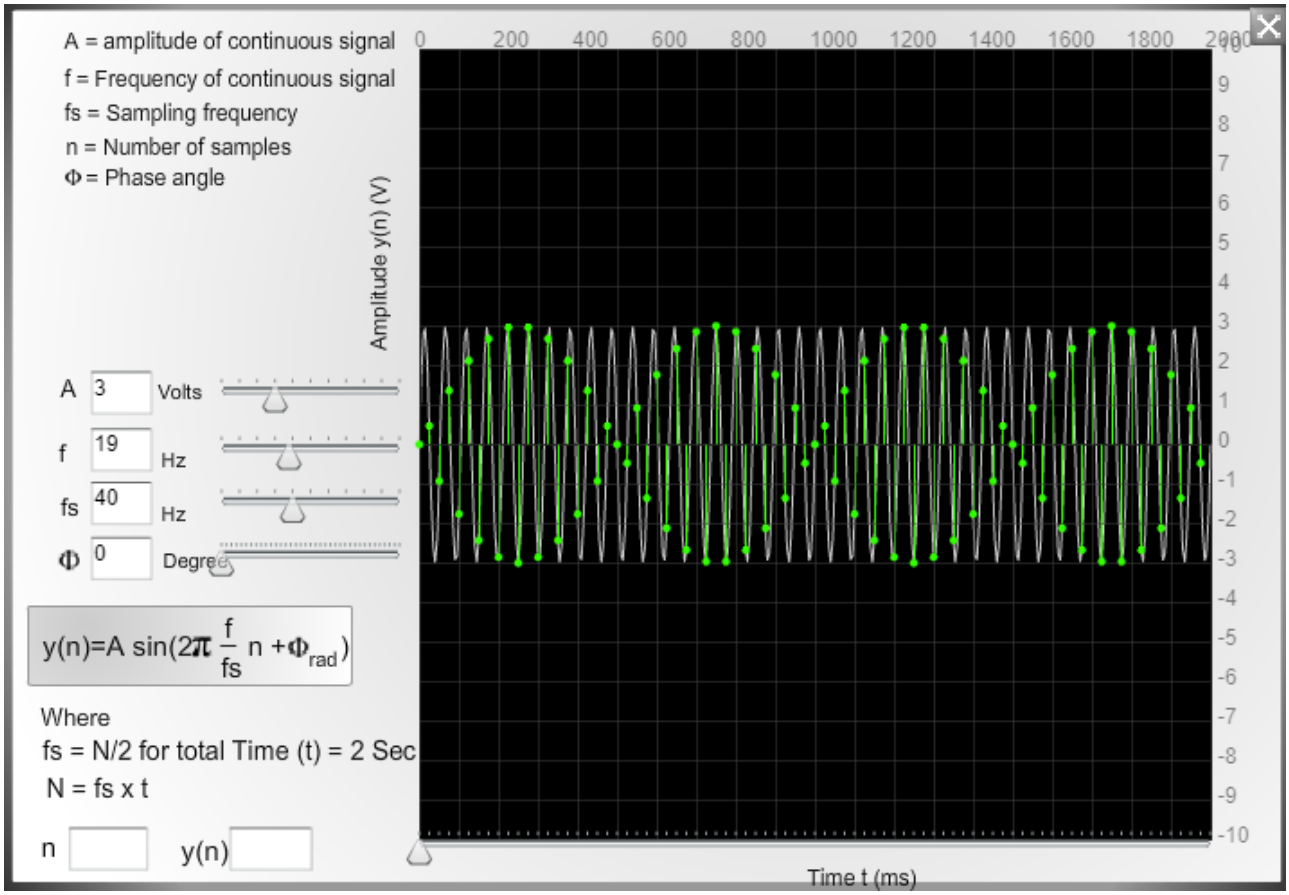
Experiment 1: Sampling Theorem

Sampling in Virtual Lab IIT-K

- Effect of Under Sampling



- Effect of Perfect Sampling



Sampling in MATLAB

Code:

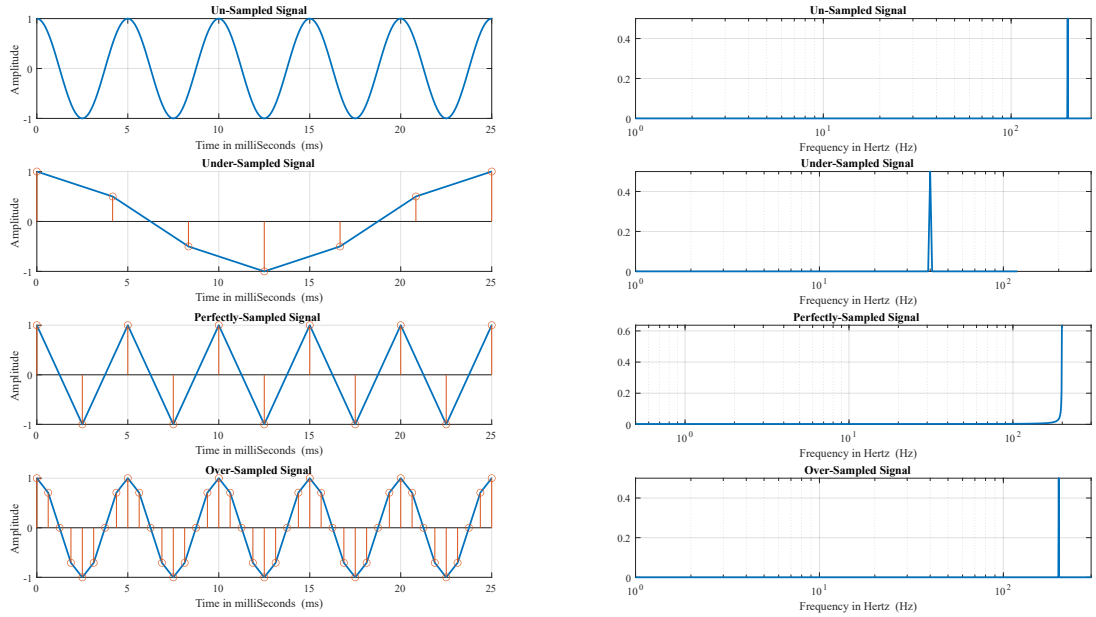
```
% THIS IS THE VERISON 2 OF THE EXPERIMENT 1 - VERIFICATION OF THE NYQUIST'S
THEOREM FOR SAMPLING
% AUTHOR: MOHAMED FARHAN FAZAL

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

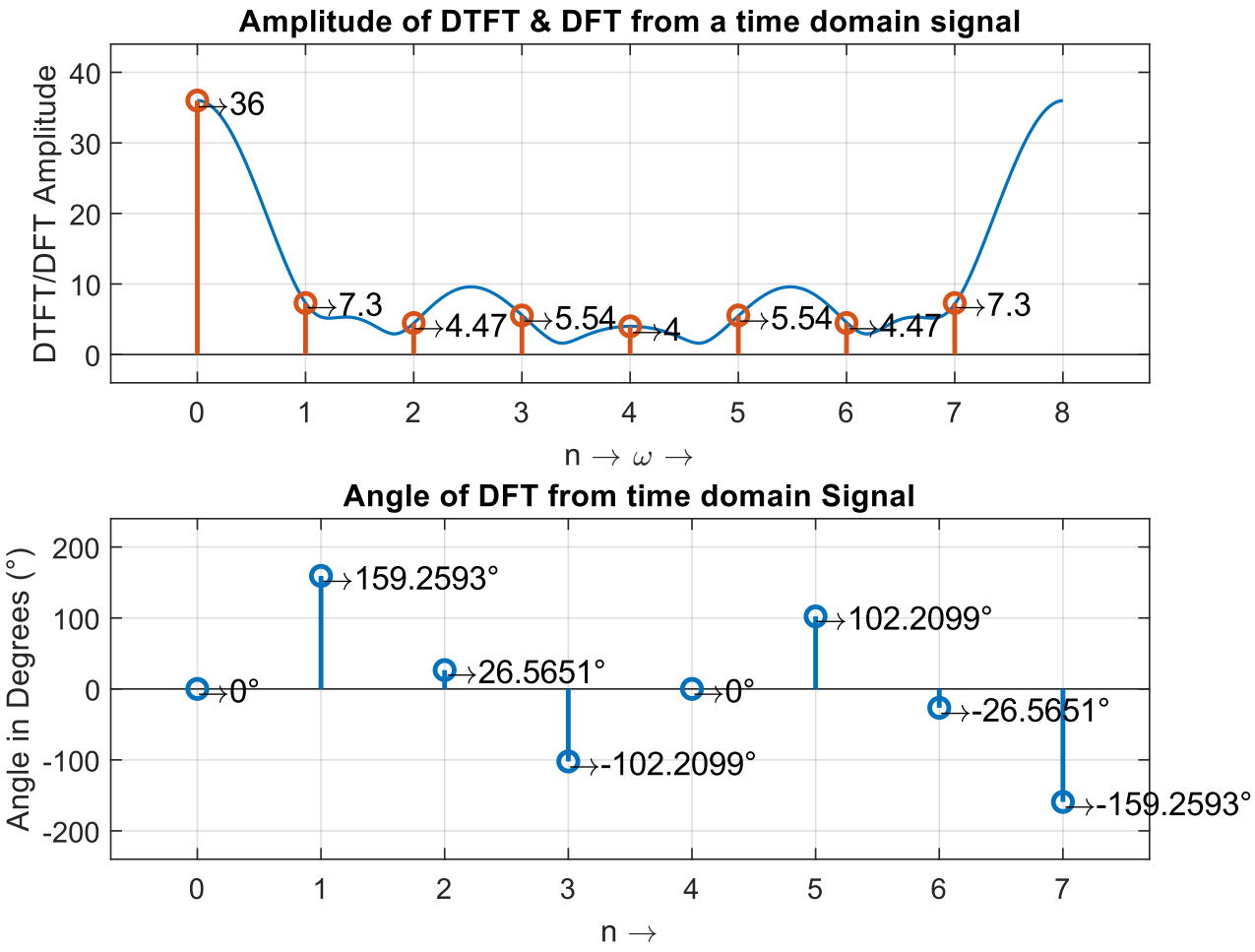
```
function twiddleMatrix = twiddleFactor(sequenceLenght, needMatrix)
    %arguments
    %sequenceLenght
    %needMatrix = 1;
    %end
    needMatrix = 1;
    twiddleMatrix = complex(ones([sequenceLenght sequenceLenght]));
    theta = 2 * pi / sequenceLenght;

    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

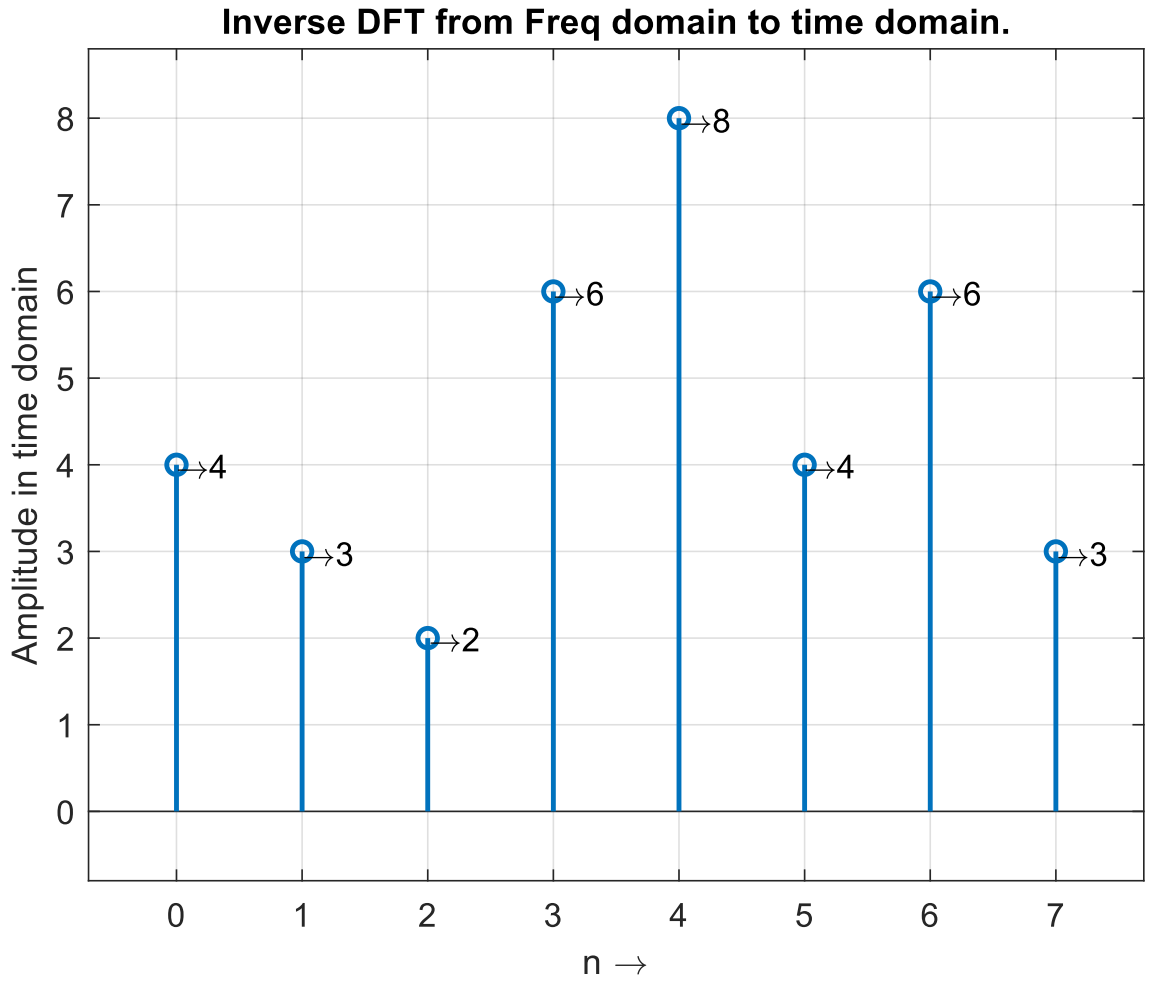
```
Enter the Sequence in time domain: [4 3 2 6 8 4 6 3]

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
```

Experiment 2
DFT | IDFT | DTFT

DFT of $x(n)$ is given by

$$X(K) = \sum_{n=0}^{N-1} x(n) W \qquad N = 8 \qquad x(n) = \{4, 3, 2, 6, 8, 4, 6, 3\}$$

$$X(K) = \sum_{n=0}^{7} x(n) W_8^{nK}$$

Using direct calculation method

$$W_8^0 = 1 \quad W_8^1 = 0.707-0.707j \quad W_8^2 = -j \quad W_8^3 = -0.707-0.707j$$
$$W_8^4 = -1 \quad W_8^5 = -0.707+0.707j \quad W_8^6 = j \quad W_8^7 = 0.707+0.707j$$

$$X(0) = 4+3+2+6+8+4+6+3 = 36$$
$$X(1) = 4+3W_8^1+2W_8^2+6W_8^3+8W_8^4+4W_8^5+6W_8^6+3W_8^7 = -6.82+2.58j$$
$$X(2) = 4+3W_8^2+2W_8^4+6W_8^6+8W_8^8+4W_8^{10}+6W_8^{12}+3W_8^{14} = 4+2j$$
$$X(3) = 4+3W_8^3+2W_8^6+6W_8^9+8W_8^{12}+4W_8^{15}+6W_8^{18}+3W_8^{21} = -1.17-5.41j$$
$$X(4) = 4+3W_8^4+2W_8^8+6W_8^{12}+8W_8^{16}+4W_8^{20}+6W_8^{24}+3W_8^{28} = 4$$
$$X(5) = -1.17+5.41j = X^*(3) \quad \text{(Conjugate Symmetry Property)}$$
$$X(6) = 4-2j = X^*(2)$$
$$X(7) = -6.82-2.58j = X^*(1)$$

$$X(K) = \{36, -6.82+2.58j, 4+2j, -1.17-5.41j, 4, -1.17+5.41j, 4-2j, -6.82-2.58j\}$$

Same in matlab

IDFT

$$X(K) = \sum_{n=0}^{N-1} x(n) W_N^{nK} \times \frac{1}{N} \qquad N = 8$$

IDFT using DFT

$$x(n) = \left\{ \sum_{K=0}^{N-1} X^*(K)\, W_N^{Kn} \right\}^*$$

$X(K) = \{36, -6.82+2.58j, 4+2j, -1.17-5.41j, 4,$
$-1.17+5.41j, 4-2j, -6.82-2.58j\}$

$X^*(K) = \{36, -6.82-2.58j, 4-2j, -1.17+5.41j, 4,$
$-1.17-5.41j, 4+2j, -6.82+2.58j\}$

$x(0) = 36 - 6.82 - 2.58j + 4 - 2j - 1.17 + 5.41j + 4 - 1.17 - 5.41j + 4 + 2j - 6.82 + 2.58j = 4$

$x(1) = 36 + (-6.82-2.58j)W_8^1 + (4-2j)W_8^2 + (-1.17+5.41j)W_8^3 + (4)W_8^4 + (-1.17-5.41j)W_8^5 + (4+2j)W_8^6 + (-6.82+2.58j)W_8^7 = 3$

$x(2) = 36 + (-6.82-2.58j)W_8^2 + (4-2j)W_8^4 + (-1.17+5.41j)W_8^6 + (4)W_8^8 + (-1.17-5.41j)W_8^{10} + (4+2j)W_8^{12} + (-6.82+2.58j)W_8^{14} = 2$

$x(3) = 36 + (-6.82-2.58j)W_8^3 + (4-2j)W_8^6 + (-1.17+5.41j)W_8^9 + (4)W_8^{12} + (-1.17-5.41j)W_8^{15} + (4+2j)W_8^{18} + (-6.82+2.58j)W_8^{21} = 6$

$x(4) = 36 + (-6.82-2.58j)W_8^4 + (4-2j)W_8^8 + (-1.17+5.41j)W_8^{12} + (4)W_8^{16} + (-1.17-5.41j)W_8^{20} + (4+2j)W_8^{24} + (-6.82+2.58j)W_8^{28} = 8$

$x(5) = 36 + (-6.82-2.58j)W_8^5 + (4-2j)W_8^{10} + (-1.17+5.41j)W_8^{15} + (4)W_8^{20} + (-1.17-5.41j)W_8^{25} + (4+2j)W_8^{30} + (-6.82+2.58j)W_8^{35} = 4$

Same

$$x(6) = 36 + (-6.82-2.58j)W_8^{6} + (4-2j)W_8^{12} + (-1.17+5.41j)W_8^{18} + (4)W_8^{24} + (-1.17-5.41j)W_8^{30} + (4+2j)W_8^{36} + (-6.82+2.58j)W_8^{42} = 6$$

$$x(7) = 36 + (-6.82-2.58j)W_8^{7} + (4-2j)W_8^{14} + (-1.17+5.41j)W_8^{21} + (4)W_8^{28} + (-1.17-5.41j)W_8^{35} + (4+2j)W_8^{42} + (-6.82+2.58j)W_8^{49} = 3$$

$$x(n) = \{4,\ 3,\ 2,\ 6,\ 8,\ 4,\ 6,\ 2\}$$
$$\uparrow$$

DTFT

$$X(e^{j\Omega}) = \sum_{n=-\infty}^{\infty} x(n)e^{-j\Omega n}$$

$$= \sum_{n=0}^{7} x(n)e^{-j\Omega n}$$

$$= x(0) + x(1)e^{-j\Omega} + x(2)e^{-2j\Omega} + x(3)e^{-3j\Omega} + x(4)e^{-4j\Omega} + x(5)e^{-5j\Omega} + x(6)e^{-6j\Omega} + x(7)e^{-7j\Omega}.$$

Experiment 3
Properties of DFT - 1

① Periodic Similarity Property: $DFT\{x_1(n) + x_2(n)\} = X_1(K) + X_2(K)$

Sequence 1: $x_1(n) = \{\underset{\uparrow}{2}, 3, 4, 5\}$.

Sequence 2: $x_2(n) = \{\underset{\uparrow}{1}, 3, 5, 7\}$

$$X_1(K) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ 4 \\ 5 \end{bmatrix} = \begin{bmatrix} 14 \\ -2+2j \\ -2 \\ -2-2j \end{bmatrix}$$

$$X_2(K) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 5 \\ 7 \end{bmatrix} = \begin{bmatrix} 16 \\ -4+4j \\ -4 \\ -4-4j \end{bmatrix}$$

$$X_1(K) + X_2(K) = \begin{bmatrix} 30 \\ -6+6j \\ -6 \\ -6-6j \end{bmatrix}$$

$x_3(n) = x_1(n) + x_2(n)$.

$= 2 \quad 3 \quad 4 \quad 5$

$+ 1 \quad 3 \quad 5 \quad 7$

$x_3(n) = \{3, 6, 9, 12\}$.

$$DFT\{x_3(n)\} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 3 \\ 6 \\ 9 \\ 12 \end{bmatrix} = \begin{bmatrix} 36 \\ -6+6j \\ -6 \\ -6-6j \end{bmatrix}$$

$\therefore DFT\{x_3(n)\} = X_1(K) + X_2(K)$.

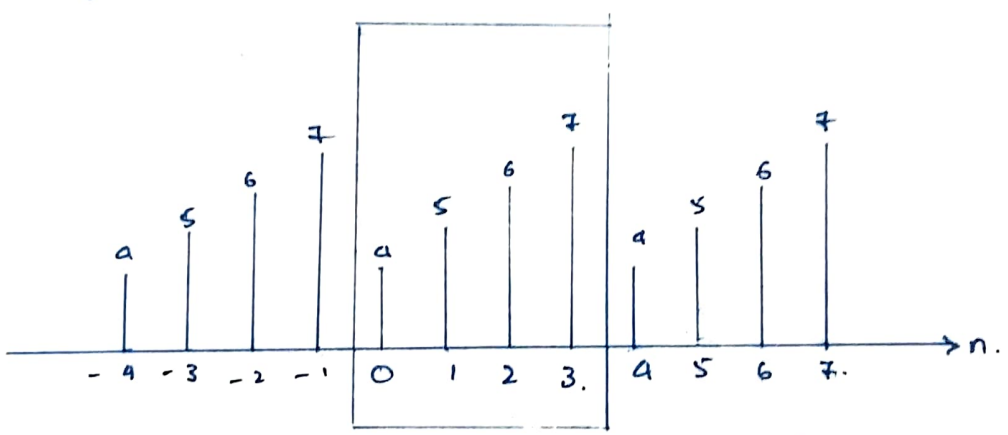
(2) **Periodicity Property**: If $X(k) = DFT\{x(n)\}$ & $x(n+N) = x(n)$

$$X(k) = X(k+n) \quad \text{for all } k.$$

$x(n) = \{4 \;\; 5 \;\; 6 \;\; 7\}$ (↑ at 4) $\quad N \rightarrow$ Period $= 4$.

$$X(k) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 4 \\ 5 \\ 6 \\ 7 \end{bmatrix} = \begin{bmatrix} 22 \\ -2+2j \\ -2 \\ -2-2j \end{bmatrix}$$

$x(N+n) = x(4+n)$.



$x(4+n) = x(n)$ $\quad N = 4$.

$\therefore X(4+k) = X(k) = [22, -2+2j, -2, -2-2j]$.
↑

(3) Circular Shifting Property :-

$x(n) = \{1, 3, 5, 7\}$ $\quad m = 2$

$$X(K) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 5 \\ 7 \end{bmatrix} = \begin{bmatrix} 16 \\ -4+4j \\ -4 \\ -4-4j \end{bmatrix}$$

$x(n-m)_N = x(n-2)_4 = \{5, 7, 1, 3\}$

$$X(K-2) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 5 \\ 7 \\ 1 \\ 3 \end{bmatrix} = \begin{bmatrix} 16 \\ 4-4j \\ -4 \\ 4+4j \end{bmatrix}$$

But $DFT\{x((n-m))_N\} = W_N^{mK} X(K) = W_4^{2K} X(K)$.

$X_1(K) = W_4^{2K} X(K)$.

$X_1(0) = X(0) = 16.$

$X_1(1) = W_4^2 X(1) = -1(-4+4j) = 4-4j.$

$X_1(2) = W_4^4 X(2) = 1(-4) = -4.$

$X_1(3) = W_4^6 X(3) = -1(-4-4j) = 4+4j.$

$X_1(K) = \{16, 4-4j, -4, 4+4j\}$

$X_1(K) = DFT\{x((n-2))_4\} = X(K-2).$

Experiment 4
Properties of DFT - 2

DSP - Lab E-4.

(1) Circular Frequency shift:

Let $x(n)$ be $\{4 \;\; 5 \;\; 6 \;\; 7\}$
$\uparrow$

$x(n) = \{4, 5, 6, 7\}$
$\uparrow$

$$X(K) = \sum_{n=0}^{3} x(n) W_4^{nK} \qquad N = 4.$$

Using matrix method.

$$X(K) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 4 \\ 5 \\ 6 \\ 7 \end{bmatrix} = \begin{bmatrix} 22 \\ -2+2j \\ -2 \\ -2-2j \end{bmatrix}$$

Now if we circularly shift $X(K)$ by d i.e $X(d-K)_N$ should be same as $DFT[x(n) \times e^{(j \times 2\pi \times n \times d/N)}]$

$X(d-K)_N = \{-2-2j, 22, -2+2j, -2\}$ Suppose $d = 1$.
$\uparrow$

Now. $x_1(n) = x(n) \times e^{(j2\pi n \times d/N)}$

$x_1(0) = 4 \quad x_1(1) = 5 \times j = 0 + 5j.$

$x_1(2) = 6(-1) = -6. \quad x_1(3) = 7(-j) = 0 - 7j.$

$x_1(n) = \{4, 5j, -6, -7j\}$
$\uparrow$

$$X_1(K) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 4 \\ 5j \\ -6 \\ -7j \end{bmatrix} = \begin{bmatrix} -2-2j \\ 22 \\ -2+2j \\ -2 \end{bmatrix}$$

Hence Verified

(2)

Circular convolution of $x(n) = \{1, 2, 3, 4\}$ and $h(n) = \{5, 6, 7\}$

1 → Make lengths equal.

$x(n) = \{1, 2, 3, 4\}$ $\quad h(n) = \{5, 6, 7, 0\}$

Time domain method:

~~$h(n-m)$~~ $h(m-n)$ when $m=0$

$h(-n) = \{5, 0, 7, 6\}$

$y(n) = h(n) \circledast x(n)$.

$= x(n) . h(m-n)$.

$$\begin{bmatrix} 5 & 0 & 7 & 6 \\ 6 & 5 & 0 & 7 \\ 7 & 6 & 5 & 0 \\ 0 & 7 & 6 & 5 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 50 \\ 44 \\ 34 \\ 52 \end{bmatrix}$$

Freq Domain.

$X(K) = \{1 + 2W_4^{1K} + 3W_4^{2K} + 4W_4^{3K}\}$

$H(K) = \{5 + 6W_4^{K} + 7W_4^{2K}\}$.

$5 + 6W_4^{K} + 7W_4^{2K} + 10W_4^{K} + 12W_4^{2K} + 14W_4^{3K} + 15W_4^{2K} + 12W_4^{3K}$

$+ 21W_4^{0} + 20W_4^{3K} + 24W_4^{0} + 28W_4^{K}$

$50W_4^{0} + 44W_4^{K} + 34W_4^{2K} + 52W_4^{3K}$

$y(n) = \{50, 44, 34, 52\}$.

(3) Conjugate Symmetry:

(a) Real sequence. $x(n) = \{1, 2, 3, 4\}$

$$DFT[x(n)] = X(K) = \sum_{n=0}^{3} x(n) W_4^{nK}$$

$$X(K) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 10 \\ -2+2j \\ -2 \\ -2-2j \end{bmatrix}$$

According to conjugate symmetric property $X(K) = X^*((-K))_N$ for a real sequence.

$$X^*((-K))_N = \{DFT[x(-n)]\}^*. \qquad x(-n) = \{1, 4, 3, 2\} = x_1(n).$$

$$DFT[x(-n)] = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 1 \\ 4 \\ 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 10 \\ -2-2j \\ -2 \\ -2+2j \end{bmatrix} = X_1(K).$$

$$X_1^*(K) = \begin{bmatrix} 10 \\ -2+2j \\ -2 \\ -2-2j \end{bmatrix} = X(K).$$ Theorem verified.

(b) Complex sequence.

$$x(n) = \{1j, 2, 3, 4\}$$

$$X(k) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 1j \\ 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 9 + 1j \\ -3 + 3j \\ -3 + j \\ -3 - j \end{bmatrix}$$

$$x_1(n) = x(-n)_N = \{1j \quad 4 \quad 3 \quad 2\}$$

$$X_1(k) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 1j \\ 4 \\ 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 9 + 1j \\ -3 - 1j \\ -3 + 1j \\ -3 + 3j \end{bmatrix}$$

$$X_1^*(k) = \begin{bmatrix} 9 - 1j \\ -3 + 1j \\ -3 - 1j \\ -3 - 3j \end{bmatrix} \neq X(k).$$

Hence sequence is not conjugate symmetric since $x(n)$ is not real.

Experiment 5
Properties of DFT - 3

① Time Reversal Property:

Let $x(n) = \{2, 3, 5, 4, 1\}$.

$$X(K) = \sum_{n=0}^{N-1} x(n)\, W_N^{nK} \qquad N = 5.$$

$$\therefore X(K) = \sum_{n=0}^{4} x(n)\, W_5^{Kn}$$

$W_5^0 = 1$ $\quad W_5^1 = 0.309 - 0.951j$ $\quad W_5^2 = -0.809 - 0.587j$

$W_5^3 = -0.8090 + 0.587j$ $\quad W_5^4 = 0.3090 + 0.9511j$.

$X(0) = 2 + 3 + 5 + 4 + 1 = 15.$

$X(1) = 2 + 3W_5^1 + 5W_5^2 + 4W_5^3 + W_5^4 = -4.04 - 2.49j.$

$X(2) = 2 + 3W_5^2 + 5W_5^4 + 4W_5^1 + W_5^3 = 1.54 - 0.22j.$

$X(3) = 2 + 3W_5^3 + 5W_5^1 + 4W_5^{448} + W_5^2 = 1.54 + 0.22j$

$X(4) = 2 + 3W_5^4 + 5W_5^3 + 4W_5^2 + W_5^1 = -4.04 + 2.49j.$

$X(K) = \{15, -4.04 + 2.49j, 1.54 + 0.22j, 1.54 - 0.22j,$
$-4.04 - 2.49j\}.$

$x(-n) = \{2 \;\; 1 \;\; 4 \;\; 5 \;\; 3\} = x(n).$

$$X_1(K) = \sum_{n=0}^{4} x_1(n)\, W_5^{nK}$$

$X_1(0) = 2 + 1 + 4 + 5 + 3 = 15.$

$X_1(1) = 2 + W_5^1 + 4W_5^2 + 5W_5^3 + 3W_5^4 = -4.04 + 2.49j$

$X_1(2) = 2 + W_5^2 + 4W_5^4 + 5W_5^6 + 3W_5^8 = 1.54 + 0.22j$

$$X_1(3) = 2 + W_5^3 + 4W_5^6 + 5W_5^9 + 3W_5^{12} = 1.54 - 0.22j.$$

$$X_1(4) = 2 + W_5^4 + 4W_5^8 + 5W_5^{12} + 3W_5^{16} = -4.04 - 2.49j.$$

Since $X_1(K) = X^*(K) = DFT\{x(-n)\}$.

Time Reversal Property is verified.

Parseval's Theorem: $\sum_{n=0}^{N-1} |x(n)|^2 = \frac{1}{N} \sum_{K=0}^{N-1} |X(K)|^2$

$x(n) = \{2 \underset{\uparrow}{} \quad 3 \quad 5 \quad 4 \quad 1\}$

$P_T = 2^2 + 3^2 + 5^2 + 4^2 + 1^2 = 4 + 9 + 25 + 16 + 1 = 55.$

$X(K) = \sum_{n=0}^{N-1} x(n) W_N^{Kn}$

$X(K) = \{15, \; -4.04 - 2.49j, \; 1.54 - 0.22j, \; 1.54 + 0.22j, \; -4.04 + 2.49j\}$

$$\sum_{K=0}^{4} X(K) X^*(K) = 15^2 + (-4.04 - 2.49j)(-4.04 + 2.49j)$$
$$+ (1.54 - 0.22j)(1.54 + 0.22j)$$
$$+ (1.54 + 0.22j)(1.54 - 0.22j)$$
$$+ (-4.04 + 2.49j)(-4.04 - 2.49j)$$

$= 225 + 22.56 + 2.43 + 2.43 + 22.86 = 275.$

$$P_F = \frac{1}{5N} \sum_{K=0}^{4} X(K) \times X^*(K) = \frac{1}{5} \times 275 = 55.$$

Circular convolution using circular.

Linear convolution $= x(n) * h(n) = \sum x(k)\, h(n-k)$

Circular convolution $= x(n) \textcircled{N} h(n) = \sum x(k)_N\, h((n-k))_N$

$x(n) = \{1, 2, 3, 4, 5\}$

$h(n) = \{6, 7, 8\}$

Length of linear convolution $= 5+3-1 = 7$.

$\textcircled{7} \times 3 \times 3 \times \textcircled{1} = 7 \times 1$

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 2 & 1 \\ 4 & 3 & 2 \\ 5 & 4 & 3 \\ 0 & 5 & 4 \\ 0 & 0 & 5 \end{bmatrix} \begin{bmatrix} 6 \\ 7 \\ 8 \end{bmatrix} = \begin{bmatrix} 6 \\ 19 \\ 40 \\ 61 \\ 82 \\ 67 \\ 40 \end{bmatrix}$$

Circular convolution $=$ 6 19 40 61 82

$+$ 67 40

73 59 40 61 82.

$x(n) \textcircled{N} h(n) = \{73, 59, 40, 61, 82.\}$

Linear Convolution using Circular.

Circular conv $y(n) = h(n) \circledast x(n)$.

$= x((k))_N \; h((n-k))_N$

Linear conv $y(n) = h(n) * x(n)$

$x(k) \cdot h(n-k)$.

$x(n) = \{1, 2, 3, 4, 5\}$ $\quad h(n) = \{6, 7, 8\}$

For linear using circular, first we make the lengths equal to

$x(n) = \{1 \; 2 \; 3 \; 4 \; 5 \; 0 \; 0\}$ length of linear conv.

$h(n) = \{6 \; 7 \; 8 \; 0 \; 0 \; 0 \; 0\}$ $\quad 5+3-1 = 7.$

$$\begin{bmatrix} 1 & 0 & 0 & 5 & 4 & 3 & 2 \\ 2 & 1 & 0 & 0 & 5 & 4 & 3 \\ 3 & 2 & 1 & 0 & 0 & 5 & 4 \\ 4 & 3 & 2 & 1 & 0 & 0 & 5 \\ 5 & 4 & 3 & 2 & 1 & 0 & 0 \\ 0 & 5 & 4 & 3 & 2 & 1 & 0 \\ 0 & 0 & 5 & 4 & 3 & 2 & 1 \end{bmatrix} \begin{bmatrix} 6 \\ 7 \\ 8 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 6 \\ 19 \\ 40 \\ 61 \\ 82 \\ 67 \\ 40. \end{bmatrix}$$

Experiment 6
Radix

Radix-2 8-point DIT FFT.

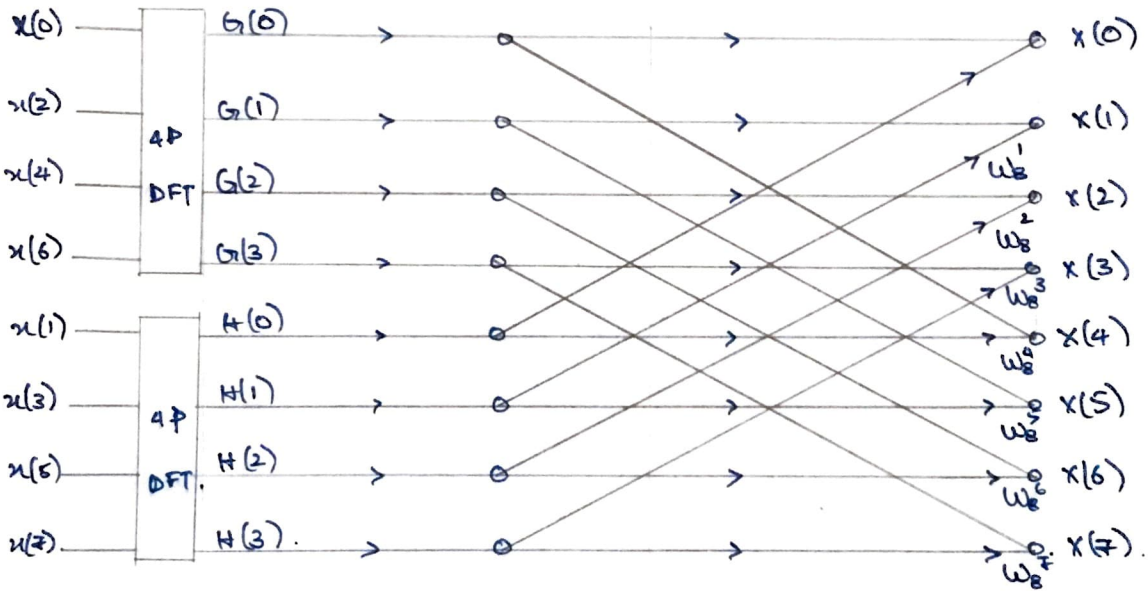
$N = 8 = 2 \times 2 \times 2$.

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk} = \sum_{n=0}^{7} x(n) W_8^{nk} = \sum_{n \to \text{even}} x(n) W_8^{nk} + \sum_{n \to \text{odd}} x(n) W_8^{nk}$$

$$X(k) = \sum_{n=0}^{3} x(2n) W_8^{2nk} + \sum_{n=0}^{3} x(2n+1) W_8^{(2n+1)k}$$

$$= \sum_{n=0}^{3} x(2n) \cdot W_4^{nk} + \sum_{n=0}^{3} x(2n+1) W_4^{nk} \cdot W_8^{k}$$

$$= G_1(k) + H_1(k) \cdot W_8^{k}$$



$X(0) = G_1(0) + H(0)$.

$X(1) = G_1(1) + W_8^1 H(1)$.

$X(2) = G_1(2) + W_8^2 H(2)$.

$X(3) = G_1(3) + W_8^3 H(3)$.

$X(4) = G_1(0) + W_8^4 H(4)$

$X(5) = G_1(1) + W_8^5 H(1)$

$X(6) = G_1(2) + W_8^6 H(2)$.

$X(7) = G_1(3) + W_8^7 H(3)$.

$$G(k) = \sum_{n=0}^{3} x(2n)\, W_4^{nk} \qquad H(k) = \sum_{n=0}^{3} x(2n+1)\, W_4^{kn}$$

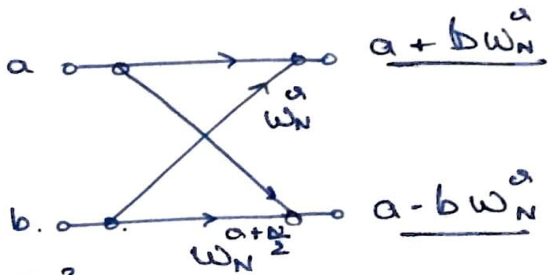
Σ odd + Σ even.

$$\sum_{n=0}^{1} x(2(2n))\, W_4^{2nk} + \sum_{n=0}^{1} x(2(2n+1))\, W_4^{nk(2n+1)}$$

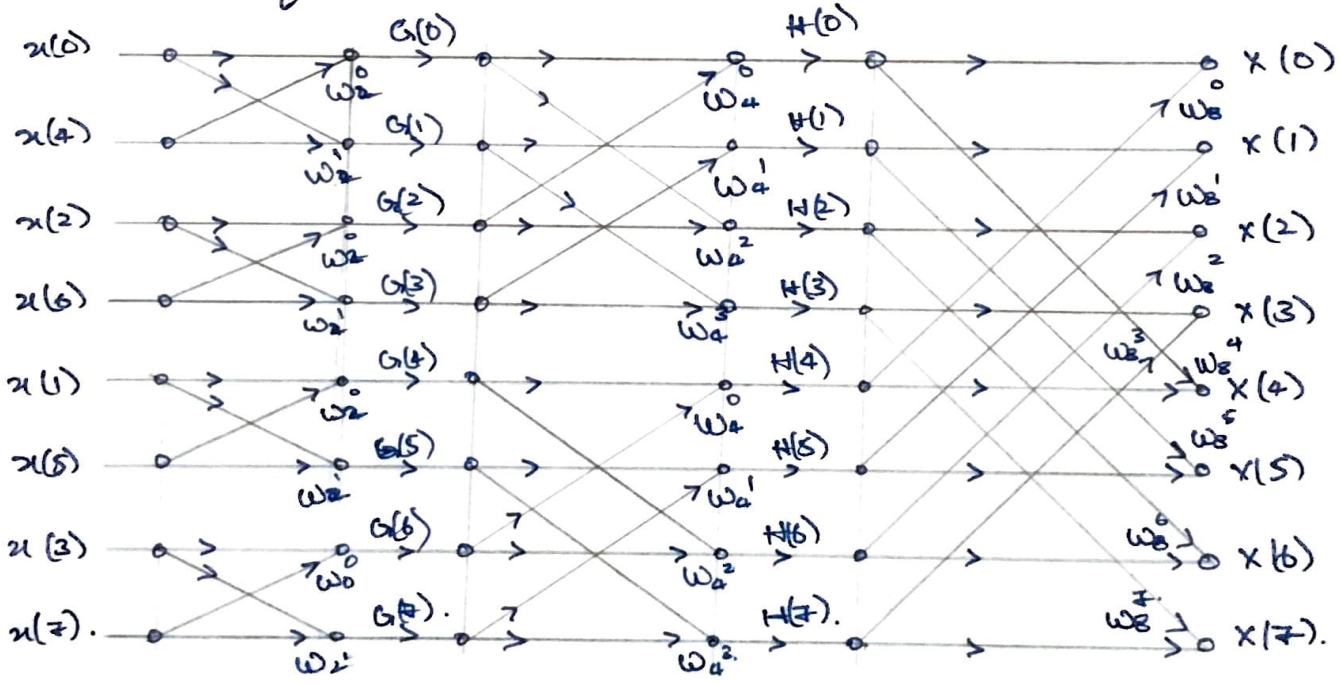
$$\sum_{n=0}^{1} x(4n)\, W_2^{nk} + \sum_{n=0}^{1} \{x(4n+2)\, W_2^{nk} \cdot W_4^{k}$$

$$P(k) + Q(k)\, W_4^{k}$$

Silly $R(k) + S(k)\, W_4^{k}$



DFT of $x(n) = \{1, 2, 3, 4, 5, 6, 7, 8\}$.
0 1 2 3 4 5 6 7



$G(0) = x(0) + x(4) = 6.$

$G(1) = x(0) - x(4) = 1 - 5 = -4.$

$G(2) = x(2) + x(6) = 3 + 7 = 10$

$G(3) = x(2) - x(6) = 3 - 7 = -4.$

$G(4) = x(1) + x(5) = 2 + 6 = 8.$

$G(5) = x(1) - x(5) = 2 - 6 = -4.$

$G(6) = x(3) + x(7) = 4 + 8 = 12.$

$G(7) = x(3) - x(7) = 4 - 8 = -4.$

$\boxed{W_4' = -j.}$

$H(0) = G(0) + G(2) = 6 + 10 = 16.$

$H(1) = G(1) + W_4' G(3) = -4 - j(-4) = -4 + 4j.$

$H(2) = G(0) - G(2) = 6 - 10 = -4.$

$H(3) = G(1) - W_4' G(3) = -4 + j(-4) = -4 - 4j.$

$H(4) = G(4) + G(6) = 8 + 12 = 20$

$H(5) = G(5) + W_4' G(7) = -4 - j(-4) = -4 + 4j.$

$H(6) = G(4) - G(6) = 8 - 12 = -4.$

$H(7) = G(5) - W_4' G(7) = -4 + j(-4) = -4 - 4j.$

$x(0) = H(0) + H(4) = 16 + 20 = 36.$

$x(1) = H(1) + W_8' H(5) = -4 + 4j + \left(\frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}}j\right)(-4 + 4j) = -4 + 9.65j.$

$x(2) = H(2) + W_8^2 H(6) = -4 + (-j)(-4) = -4 + 4j.$

$x(3) = H(3) + W_8^3 H(7) = -4 - 4j + \left(-\frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}}j\right)(-4 - 4j) = -4 + 1.65j$

$x(4) = H(0) - H(4) = -4$

$x(5) = H(1) - W_8' H(5) = -4 - 1.657j$

$x(6) = H(2) - W_8^2 H(6) = -4 - 4j$

$x(7) = H(3) - W_8^3 H(7) = -4 - 9.654j.$

Radix 2 - DIF FFT

$$X(K) = \sum_{n=0}^{N-1} x(n)\, W_N^{Kn}. \qquad N = 8.$$

$$x(K) = \sum_{n=0}^{7} x(n)\, W_N^{Kn} = \sum_{n=0}^{3} x(n)\, W_8^{Kn} + \sum_{4}^{7} x(n)\, W_8^{Kn}.$$

$$= \sum_{n=0}^{3} x(n)\, W_8^{Kn} + \sum_{n=0}^{3} x\left(n+\frac{N}{2}\right) W_N^{K\left(n+\frac{N}{2}\right)}.$$

$$\sum_{n=0}^{3} x(n)\, W_8^{Kn} + \sum_{n=0}^{3} x\left(n+\frac{N}{2}\right) W_8^{Kn} \cdot W_8^{K\frac{N}{2}}.$$

$$X(K) = X_1(K) + X_2(K)\, W_8^{K\frac{N}{2}} = X_1(K) + X_2(K)(-1)^K.$$

For even sequence. $= \sum_{n=0}^{3} \left[x(n) + x\left(n+\frac{N}{2}\right)(-1)^K\right] W_8^{K(n)}.$

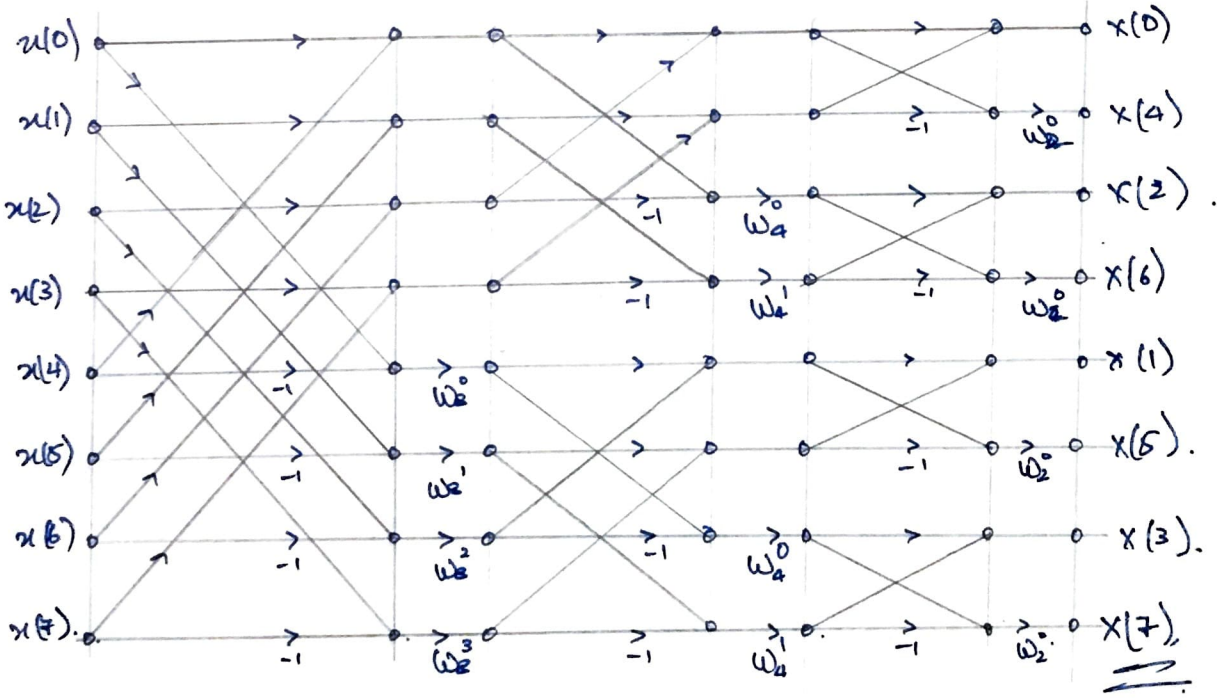
$K = 2R.$

$$X(2K) = X_1(2K) + X_2(2K)(-1)^{2K} = X_1(2K) + X_2(2K).$$

For odd.

$K = 2K+1.$

$$X(2K+1) = X_1(2K+1) + X_2(2K+1)(-1)^{2K+1} = X_1(2K+1) - X_2(2K+1).$$



$x_1(0) = x(0) + x(4) = 1 + 5 = 6$ $\quad x(n) = \{1, 2, 3, 4, 8, 6, 7, 8\}$

$x_1(1) = x(1) + x(5) = 8$

$x_1(2) = x(2) + x(6) = 10$

$x_1(3) = x(3) + x(7) = 12$

$x_1(4) = x(0) - W_8^0 x(4) = -4$

$x_1(5) = x(1) - W_8^1 x(5) = -2.82 + 2.82j$

$x_1(6) = x(2) - W_8^2 x(6) = 4j$

$x_1(7) = x(3) - W_8^3 x(7) = 2.82 + 2.82j$.

$\longrightarrow$

$x_2(0) = x_1(0) + x_1(2) = 16$

$x_2(1) = x_1(1) + x_1(3) = 20$

$x_2(2) = x_1(0) - W_4^0 x_1(2) = -4$

$x_2(3) = x_1(1) - W_4^1 x_1(3) = 4j$

$x_2(4) = x_1(4) + x_1(6) = -4 + 4j$

$x_2(5) = x_1(5) + x_1(7) = 5.65j$

$x_2(6) = x_1(4) - W_4^0 x_1(6) = -4 - 4j$

$x_2(7) = x_1(5) - W_4^1 x_1(7) = 5.65j$.

$\longrightarrow$

$x(0) = x_2(0) + x_2(1) = 36$

$x(4) = x_2(0) - x_2(1) W_2^0 = -4$

$x(2) = x_2(2) + x_2(3) = -4 + 4j$

$x(6) = x_2(2) - x_2(3) = -4 - 4j$

$x(1) = x_2(4) + x_2(5) = -4 + 9.65j$

$x(5) = x_2(4) - x_2(5) = -4 - 1.65j$

$x(3) = x_2(6) + x_2(7) = -4 + 1.65j$

$x(7) = x_2(6) - x_2(7) = -4 - 9.65j$.

$X(k) = \{$ 36 ← 0th place.

$-4 + 9.65j$

$-4 + 4j$

$-4 + 1.65j$

-4

$-4 - 1.65j$

$-4 - 4j$

$-4 - 9.65j$ $\}$.

Cooley Radix $N=6 = 3 \times 2$. DIT FFT.

$x(n) = \{ 7, 8, 9, 8, 7, 6 \}$

2 - Stages.

Radix 2 in 1st stage.

— " — 3 — " — 2nd " .

Sol: $X(K) = \sum_{n=0}^{N-1} x(n) W_N^{nK}$.

$$X(K) = \sum_{n=0}^{1} x(3n) W_N^{3nK} + \sum_{n=0}^{1} x(3n+1) W_N^{(3n+1)K} + \sum_{n=0}^{1} x(3n+2) W_N^{(3n+2)K}$$

N=6.

$$= \sum_{n=0}^{1} x(3n) W_2^{nK} + \sum_{n=0}^{1} x(3n+1) W_2^{nK} \cdot W_6^{K} + \sum_{n=0}^{1} x(3n+2) W_2^{nK} \cdot W_6^{2K}$$

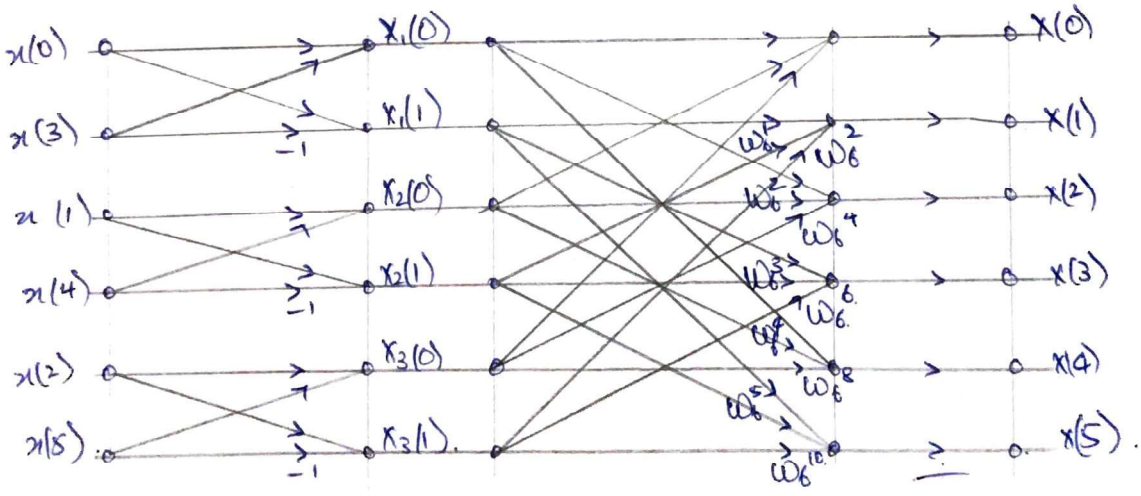
$$= X_1(K) + X_2(K) W_6^{K} + X_3(K) W_6^{2K}$$

$$X_1(K) = \sum_{n=0}^{1} x(3n) W_2^{nK} = x(0) + x(3) W_2^{3K}$$

$$X_2(K) = \sum_{n=0}^{1} x(3n+1) W_2^{nK} = x(1) + x(4) W_2^{K}$$

$$X_3(K) = \sum_{n=0}^{1} x(3n+2) W_2^{nK} = x(2) + x(5) W_2^{K}$$

$$X(K) = X_1(K) + X_2(K) W_6^{K} + X_3(K) W_6^{2K}$$



$x_1(0) = x(0) + x(3) = 15$

$x_1(1) = x(0) - x(3) = -1$

$x_2(0) = x(4) + x(1) = 8 + 7 = 15$

$x_2(1) = x(1) - x(4) = 1$

$x_3(0) = x(2) + x(5) = 9 + 6 = 15$

$x_3(1) = x(2) - x(5) = 9 - 6 = \underline{3}.$

$x(0) = x_1(0) + x_2(0) + x_3(0) = 15 + 15 + 15 = \underline{45}.$

$x(1) = x_1(1) + x_2(1) W_6^1 + x_3(1) W_6^2 = -2 - 3.46j.$

$x(2) = x_1(0) + x_2(0) W_6^2 + x_3(0) W_6^4 = 0$

$x(3) = x_1(1) + x_2(1) W_6^3 + x_3(1) W_6^6 = 1$

$x(4) = x_1(0) + x_2(0) W_6^4 + x_3(0) W_6^8 = 0$

$x(5) = x_1(1) + x_2(1) W_6^5 + x_3(1) W_6^{10} = -2 + 3.46j.$

Direct cal:

$W_6^0 = 1 \qquad W_6^1 = 0.5 - 0.866j. \qquad W_6^2 = -0.5 - 0.866j.$

$W_6^3 = -1 \qquad W_6^4 = -0.5 + 0.866j \qquad W_6^5 = 0.5 + 0.866j.$

$$X(k) = \sum_{n=0}^{5} x(n) W_6^{nk}$$

$x(0) = 7 + 8 + 9 + 8 + 7 + 6 = 45.$

$x(1) = 7 + 8W_6^1 + 9W_6^2 + 8W_6^3 + 7W_6^4 + 6W_6^5 = -2 - 3.46j.$

$x(2) = 7 + 8W_6^2 + 9W_6^4 + 8W_6^6 + 7W_6^8 + 6W_6^{10} = 0$

$x(3) = 7 + 8(-1) + 9 - 8 + 7 - 6 = 1$

$x(4) = 7 + 8W_6^4 + 9W_6^8 + 8W_6^{12} + 7W_6^{16} + 6W_6^{20} = 0.$

$x(5) = 7 + 8W_6^5 + 9W_6^{10} + 8W_6^{15} + 7W_6^{20} + 6W_6^{25} = -2 + 3.46j.$

(b) $N = 6 = 2 \times 3$.

$$X(K) = \sum_{n=0}^{5} x(n) W_6^{Kn}$$

$$X(K) = \sum_{n=0}^{2} x(2n) W_6^{2nK} + \sum_{n=0}^{2} x(2n+1) W_6^{(2n+1)K}$$

$$X_1(K) + W_6^{K} X_2(K).$$

$$X_1(K) = \sum_{n=0}^{2} x(2n) W_6^{2nK} = x(0) + x(2) \cdot W_6^{2K} + x(4) W_6^{4K}$$

$$X_1(0) = x(0) + x(2) + x(4). \quad X_1(1) = x(0) + x(2) W_6^{2} + x(4) W_6^{4}$$

$$X_2(0) = x(1) + x(3) + x(5). \quad X_2(1) = x(1) + x(3) W_6^{2} + x(5) W_6^{4}$$

$$X_1(2) = x(0) + x(2) W_6^{4} + x(4) W_6^{8} \quad X_2(2) = x(1) + x(3) W_6^{4} + x(5) W_6^{8}$$

$$X(K) = X_1(K) + W_6^K X_2(K).$$

$$X(0) = X_1(0) + X_2(0).$$

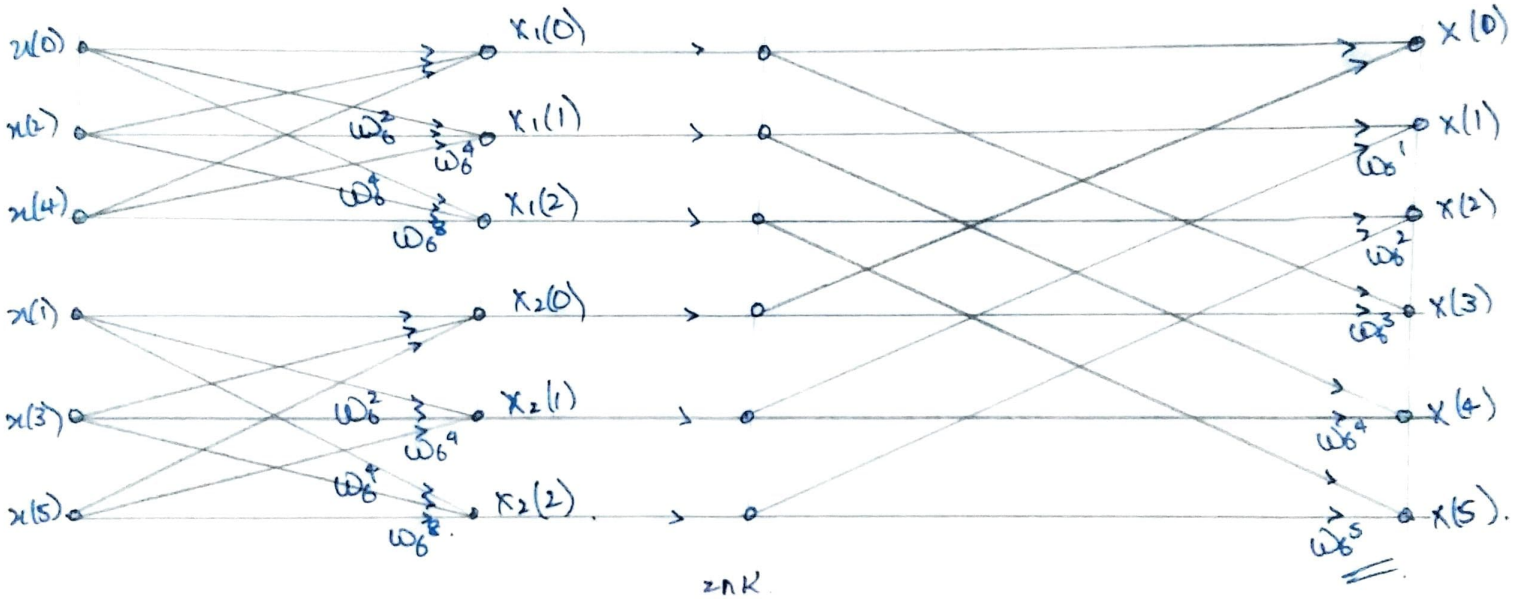
$$X(1) = X_1(1) + W_6^1 X_2(1)$$

$$X(2) = X_1(2) + W_6^2 X_2(2).$$

$$X(3) = X_1(0) + W_6^3 X_2(0).$$

$$X(4) = X_1(1) + W_6^4 X_2(1).$$

$$X(5) = X_1(2) + W_6^5 X_2(2).$$



$$X_1(K) = \sum_{n=0}^{2} x(2n)\, W_6^{2nK}$$

$X_1(0) = 9$

$X_1(1) = -3 + 1.73j$

$X_1(2) = -3 - 1.73j$

$X_2(0) = 12.$

$X_2(1) = -3 + 1.7j$

$X_2(2) = -3 - 1.73j.$

$X(0) = 9 + 12 = 21.$

$X(1) = -3 + 5j + W_6^1(-3 + 1.7j) = -3 + 5.19j.$

$X(2) = -3 - 1.73j + W_6^2(-3 - 1.7j) = -3 + 1.7j.$

$X(3) = 9 - 12 = -3.$

$X(4) = (-3 + 1.73j) + W_6^4(-3 + 1.7j) = -3 - 1.73j$

$X(5) = -3 + 8.5.196j.$

(6) $N = 6 = 2 \times 3$. $X(K) = \sum_{n=0}^{5} x(n) W_6^{nK}$ $x(n) = [1, -1, 2, -2, 3, -3]$.

$$X(K) = \sum_{n=0}^{2} x(2n) W_6^{2nK} + \sum_{n=0}^{2} x(2n+1) W_6^{(2n+1)K}$$

$$X_1(K) + W_6^{K} X_2(K).$$

$$X_1(K) = \sum_{n=0}^{2} x(2n) W_6^{2nK} = x(0) + x(2) \cdot W_6^{2K} + x(4) W_6^{4K}.$$

$X_1(0) = x(0) + x(2) + x(4)$. $X_1(1) = x(0) + x(2) W_6^{2} + x(4) W_6^{4}$

$X_2(0) = x(1) + x(3) + x(5)$. $X_2(1) = x(1) + x(3) W_6^{2} + x(5) W_6^{4}$

$X_1(2) = x(0) + x(2) W_6^{4} + x(4) W_6^{8}$ $X_2(2) = x(1) + x(3) W_6^{4} + x(5) W_6^{8}$

$$X(K) = X_1(K) + \omega_6^K X_2(K).$$

$$X(0) = X_1(0) + X_2(0).$$

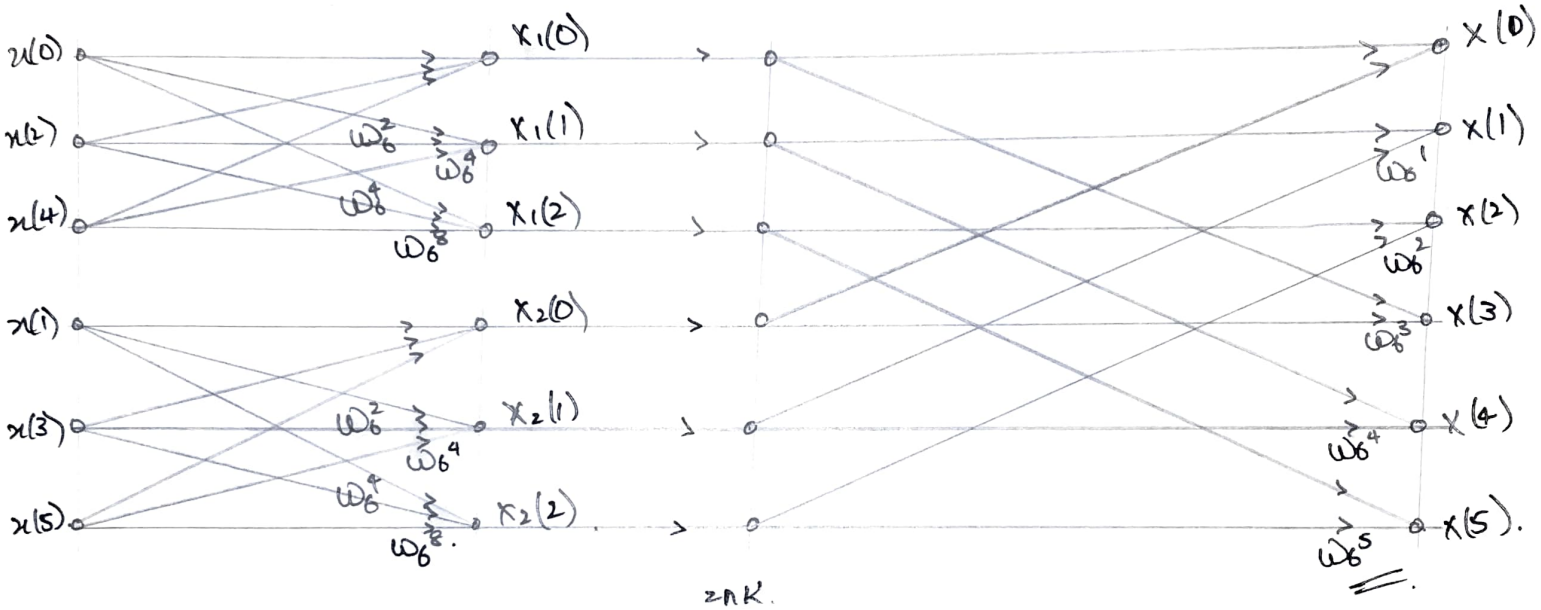
$$X(1) = X_1(1) + \omega_6^1 X_2(1)$$

$$X(2) = X_1(2) + \omega_6^2 X_2(2).$$

$$X(3) = X_1(0) + \omega_6^3 X_2(0).$$

$$X(4) = X_1(1) + \omega_6^4 X_2(1).$$

$$X(5) = X_1(2) + \omega_6^5 X_2(2).$$



$$X_1(K) = \sum_{n=0}^{2} x(2n)\, W_6$$

$X_1(0) = 9$

$X_1(1) = -3 + 1.73j$

$X_1(2) = -3 - 1.73j$

$X_2(0) = 12.$

$X_2(1) = -3 + 1.7j$

$X_2(2) = -3 - 1.73j.$

$X(0) = 9 + 12 = 21.$

$X(1) = -3 + 5j + \omega_6^1(-3 + 1.7j) = -3 + 5.196j.$

$X(2) = -3 - 1.73j + \omega_6^2(-3 - 1.7j) = -3 + 1.7j.$

$X(3) = 9 - 12 = -3.$

$X(4) = (-3 + 1.73j) + \omega_6^4(-3 + 1.7j) = -3 - 1.73j$

$X(5) = -3 + 5.196j.$

Experiment 7
CCS

$x(n) = \{1, 2, 3, 4, 5\}$ $h(n) = \{6, 7, 8\}$

For circular using circular, first we make the lengths equal to

$x(n) = \{1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 0 \quad 0\}$ length of circular convo.

$h(n) = \{6 \quad 7 \quad 8 \quad 0 \quad 0 \quad 0 \quad 0\}$ $5+3-1 = 7$.

$$\begin{bmatrix} 1 & 0 & 0 & 5 & 4 & 3 & 2 \\ 2 & 1 & 0 & 0 & 5 & 4 & 3 \\ 3 & 2 & 1 & 0 & 0 & 5 & 4 \\ 4 & 3 & 2 & 1 & 0 & 0 & 5 \\ 5 & 4 & 3 & 2 & 1 & 0 & 0 \\ 0 & 5 & 4 & 3 & 2 & 1 & 0 \\ 0 & 0 & 5 & 4 & 3 & 2 & 1 \end{bmatrix} \begin{bmatrix} 6 \\ 7 \\ 8 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 6 \\ 19 \\ 40 \\ 61 \\ 82 \\ 67 \\ 40. \end{bmatrix}$$

Circular convolution.

(1) $x(n) = \{1, 2, 3, 4\}$ $h(n) = \{1, -1, 1, -1\}$

Solution:

$x(n) = \{1, 2, 3, 4\}$ $h(-n) = \{1, -1, 1, -1\}$

Time Domain Approach:

$$\begin{bmatrix} 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1-2+3-4 \\ -1+2-3+4 \\ 1-2+3-4 \\ -1+2-3+4 \end{bmatrix} = \begin{bmatrix} -2 \\ 2 \\ -2 \\ 2 \end{bmatrix}$$

Freq Domain Approach.

$$X(K) = 1 + 2W_4^K + 3W_4^{2K} + 4W_4^{3K}.$$

$$H(K) = 1 - W_4^K + W_4^{2K} - W_4^{3K}.$$

$X(K) \times H(K)$

W_4^0	W_4^1	W_4^2	W_4^3
1	-1	1	-1
-2	+2	-2	+2
3	-3	3	-3
-4	4	-4	4
-2	2	-2	2

$\therefore Y(K) = -2W_4^0 + 2W_4^1 - 2W_4^2 + 2W_4^3.$

$\therefore y(n) = \{-2, 2, -2, 2\}$

Co-relation
- Auto-correlation (Same sequence)
- Cross-correlation (Different sequence)

Cross correlation

$$R_{xy} = \sum_{n=-\infty}^{\infty} x(n)\, y(n-k)$$

$$R_{yx} = \sum_{n=-\infty}^{\infty} y(n)\, x(n-k)$$

$R_{xy} \neq R_{yx}$

Auto Correlation

$$R_{xx} = \sum_{n=-\infty}^{\infty} x(n)\, x(n-k)$$

(1) Auto-correlation

$x(n) = \{5, 6, 7, 8, 1, 2, 3, 4\}$

$x(-n) = \{5, 4, 3, 2, 1, 8, 7, 6, \}$

$$\begin{bmatrix}
5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
6 & 5 & 0 & 0 & 0 & 0 & 0 & 0 \\
7 & 6 & 5 & 0 & 0 & 0 & 0 & 0 \\
8 & 7 & 6 & 5 & 0 & 0 & 0 & 0 \\
1 & 8 & 7 & 6 & 5 & 0 & 0 & 0 \\
2 & 1 & 8 & 7 & 6 & 5 & 0 & 0 \\
3 & 2 & 1 & 8 & 7 & 6 & 5 & 0 \\
4 & 3 & 2 & 1 & 8 & 7 & 6 & 5 \\
0 & 4 & 3 & 2 & 1 & 8 & 7 & 6 \\
0 & 0 & 4 & 3 & 2 & 1 & 8 & 7 \\
0 & 0 & 0 & 4 & 3 & 2 & 1 & 8 \\
0 & 0 & 0 & 0 & 4 & 3 & 2 & 1 \\
0 & 0 & 0 & 0 & 0 & 4 & 3 & 2 \\
0 & 0 & 0 & 0 & 0 & 0 & 4 & 3 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 4
\end{bmatrix}
\begin{bmatrix} 5 \\ 4 \\ 3 \\ 2 \\ 1 \\ 8 \\ 7 \\ 6 \end{bmatrix}
=
\begin{bmatrix} 20 \\ 39 \\ 56 \\ 70 \\ 88 \\ 117 \\ 156 \\ 204 \\ 156 \\ 117 \\ 88 \\ 70 \\ 56 \\ 39 \\ 20 \end{bmatrix}$$

Cross correlation

$x(n) = \{1, 2, 3, 4\}$

$h(n) = \{5, 6, 7, 8\}$

Solution

$$R_{xy} = \sum_{n=-\infty}^{\infty} x(n)\, h(n-k)$$

$\therefore h(-n) = \{5, 8, 7, 6\}$

$x(n) = \{1, 2, 3, 4\}$

R_{xy} =

$x(n)$ \ $h(-n)$	8	7	6	5
1	8	7	6	5
2	16	14	12	10
3	24	21	18	15
4	32	28	24	20

$R_{xy} = \{8, 23, 44, 70, 56, 39, 20\}$

$x(-n)$ \ $h(n)$	5	6	7	8
4	20	24	28	32
3	15	18	21	24
2	10	12	14	16
1	5	6	7	8

$R_{yx} = \{20, 39, 56, 70, 44, 23, 8\}$

Experiment 8
Digital Filters

(1) Digital Butterworth low pass Filter. Using IIT at T=1.

$A_p = -1.932$ dB. $\omega_p = 0.2\pi$ rad $= 0.6285$.

$A_s = -13 - 13.9794$ dB. $\omega_s = 0.6\pi$ rad $= 1.8857$.

Solution: For IIT.

$$\Omega_p = \frac{\omega_p}{T} = 0.6285 \text{ rad/s}.$$

$$\Omega_s = \frac{\omega_s}{T} = 1.8857 \text{ rad/s}.$$

Step 1: Order — N.

$$N \geq \frac{\log_{10}\left[\frac{10^{0.1A_s}-1}{10^{0.1A_p}-1}\right]}{2\log_{10}\frac{\Omega_s}{\Omega_p}} = \frac{\log_{10}\left[\frac{10^{0.1\times 13.9794}-1}{10^{0.1\times 1.932}-1}\right]}{2\log_{10}\left[\frac{1.8857}{0.6285}\right]}$$

$N \geq 1.7099$ $\boxed{\therefore N = 2.}$

Step 2:

Calculating Cutoff Ω_c.

$$\Omega_{cp} = \frac{\Omega_p}{(10^{0.1A_p}-1)^{1/2N}} = \frac{0.6285}{(10^{0.1\times 1.932}-1)^{1/4}} = 0.726.$$

$$\Omega_{cs} = \frac{\Omega_s}{(10^{0.1A_s}-1)^{1/2N}} = \frac{1.8857}{(10^{0.1\times 13.9794}-1)^{1/4}} = 0.852.$$

$$\therefore \Omega_c = \frac{\Omega_{cp}+\Omega_{cs}}{2} = 0.7889 \text{ rad/s}.$$

Step 3: Designing Normalised Filter.

$$P_k = \pm e^{j\frac{(2k+1+N)\pi}{2N}}$$

$$P_0 = \pm (0.707 + 0.707j)$$

$$P_1 = \pm (-0.707 - 0.707j)$$

$$H(s) = \frac{1}{(s+0.707-0.707j)(s+0.707+0.707j)}$$

$$\frac{1}{(s+a+jb)(s+a-jb)} = \frac{1}{s^2+2as+a^2+b^2}$$

$$= \frac{1}{(s^2+1.4145+1)}$$

Step 4: De normalising:

$$s \to \frac{s}{\omega_c} = \frac{s}{0.789}$$

$$\therefore H(s) = \frac{1}{s^2+1.414s+1} \quad s \to \frac{s}{0.789}$$

$$H(s) = \frac{0.623}{s^2+1.11s+0.623}$$

Step 5: $s \to z$ transformation.

$$\frac{b}{(s+a)^2+b^2} = \frac{1.78 \times 0.561}{(s+0.555)^2+(0.561)^2} \to \frac{e^{-at}\sin bt\, z^{-1}}{1-2e^{-at}\cos bt\, z^{-1}+e^{-2at}z^{-2}}$$

$$H(z) = \frac{e^{-0.555}\sin(0.561)z^{-1}}{1-2e^{0.555}\cos(0.561)z^{-1}+e^{-2(0.555)}z^{-2}}$$

$$H(z) = \frac{0.305\, z^{-1}}{1 - 0.9721 z^{-1} + 0.3295 z^{-2}}$$

(2) Design a chebyshev filter.

$0.75 \le H(\omega) \le 1 \qquad 0 \le \omega \le 0.25\pi$

$|H(\omega)| \le 0.23 \qquad 0.63\pi \le \omega \le \pi$

BLT: $T = 2$.

Sol:

$A_p = 0.75 \qquad \omega_p = 0.25\pi \qquad \Omega_p = \frac{2}{2}\tan\frac{\omega_p}{2} = 0.414$ rad/s.

$A_s = 0.23 \qquad \omega_s = 0.63\pi \qquad \Omega_s = \frac{2}{2}\tan\frac{\omega_s}{2} = 1.522$ rad/s.

$A_p = -20\log(0.75) = 2.49$ dB. $\qquad \Omega_p = 0.414$ rad/s.

$A_s = -20\log(0.23) = 12.76$ dB. $\qquad \Omega_s = 1.522$ rad/s.

Normalizing:

$\Omega_p' = \frac{\Omega_p}{\Omega_p} = 1 \qquad \Omega_s' = \frac{\Omega_s}{\Omega_p} = 3.676$

Finding ε.

$\varepsilon = \sqrt{10^{0.1A_p} - 1} = 0.8798.$

Order N

$N \ge \frac{A_s - 20\log_{10}(\varepsilon) + 6}{6 + 20\log_{10}(\Omega_s')} \qquad 1.148.$

$\ge 2.657 = 3 = 2.$

$N = 2.$

$$H_1(s) = \frac{K}{(s-s_1)(s-s_2)}.$$

$$s_1 = -0.3583 + j\,0.7927.$$

$$s_2 = -0.3583 - j\,0.7927.$$

$$H_1(s) = \frac{K}{s^2 + 0.7166s + 0.7567}.$$

Finding K.

$$K = \frac{b_0}{\sqrt{1-\varepsilon^2}} \quad \left.\begin{array}{l} \\ \end{array}\right\} \because \text{Nis unity.}$$

$$b_0 = 0.7567.$$

$$\therefore K = 0.56.$$

Scaling transformation -

LP - LP $\qquad s \rightarrow \frac{s}{\omega_{2P}} = \frac{s}{0.414}.$

$$H(s) = \frac{0.56\,(0.414)^2}{s^2 + 0.7166 \times 0.414\,s + 0.7567 \times (0.414)^2}$$

$$H(s) = \frac{0.0959}{s^2 + 0.2966s + 0.129}.$$

Finding $H(z)$.

$$s \to \frac{2}{T}\left[\frac{1-z^{-1}}{1+z^{-1}}\right] \qquad T = 2.$$

$$s \to \left[\frac{1-z^{-1}}{1+z^{-1}}\right]$$

$$H(z) = \frac{0.0959}{\frac{(1-z^{-1})^2}{(1+z^{-1})^2} + 0.3\left[\frac{1-z^{-1}}{1+z^{-1}}\right] + 0.129.}$$

~~0.995~~ $0.0959\left[1+2z^{-1}+z^{-2}\right]$

$1.429 + (-1.742)z^{-1} + 0.829z^{-2}$