



Lab Manual

Digital Signal Processing Lab

(CSE-310 / EEE-308)

Noor Md Shahriar

BSc in EEE, [RUET](#)

Senior Lecturer

Co-chairman, Dept. of EEE

University of Global Village (UGV)

874/322, C&B Road, Barishal, Bangladesh.

 Contact: +8801743500587

 [Facebook](#) |  [LinkedIn](#) |  [Twitter](#)



Reference Book:

1. Vinay K. Ingle, John G. Proakis - Digital Signal Processing Using MATLAB, 3rd Edition
Cengage Learning (2011)
2. Michael Weeks- Digital Signal Processing DSP Using MATLAB and Wavelets

Contents

| | |
|--------------------------------------|----|
| Course Rationale | 2 |
| Course Objectives: | 2 |
| Course Learning Outcomes (CLOs)..... | 2 |
| Course Outline | 2 |
| Course Schedule..... | 3 |
| Assessment Pattern | 5 |
| Experiment No.: 01 | 7 |
| Experiment No.: 02 | 9 |
| Experiment No.: 03 | 19 |
| Experiment No.: 04 | 25 |
| Experiment No.: 05 | 33 |
| Experiment No.: 06 | 37 |
| Experiment No.: 07 | 39 |
| Experiment No.: 08 | 42 |

Course Rationale

The rationale for this Sessional is to provide students with practical experience in understanding, analyzing, and applying digital signal processing (DSP) concepts. This lab complements theoretical coursework by enabling students to interact directly with DSP algorithms and systems using computational tools like MATLAB. By working on real-world signal processing problems, students develop the skills necessary to design and evaluate digital systems for applications in communication, control, and multimedia, bridging the gap between theory and practical implementation.

Course Objectives:

1. Provide hands-on experience with MATLAB to simulate and analyze digital signals and systems.
2. Enable students to understand and implement key DSP operations such as convolution, correlation, and z-transforms.
3. Develop proficiency in designing and evaluating FIR filters for practical applications.
4. Enhance problem-solving skills by working on real-world DSP challenges in the frequency and time domains.
5. Strengthen the understanding of analog-to-digital conversion and digital representation of signals.

Course Learning Outcomes (CLOs)

| | |
|--------------|--|
| CLO 1 | Understand DSP Fundamentals: Demonstrate an understanding of DSP concepts, signal representation, and MATLAB basics. |
| CLO 2 | Analyze Signals: Using mathematical and computational tools to analyze and interpret discrete-time signals in time and frequency domains. |
| CLO 3 | Apply DSP Techniques: Implement and manipulate signal processing techniques such as convolution, correlation, and z-transform in MATLAB. |
| CLO 4 | Design Digital Filters: Design and simulate FIR filters for signal processing applications and evaluate their performance. |

Course Outline

| Sl. No. | Topic & Details | Class Hours |
|---------|---|-------------|
| 1 | Introduction to MATLAB: Basics of MATLAB environment and initial programming setup | 2 |
| 2 | Continuous-Time and Discrete-Time Representation of Signals: Visualization and basic operations | 2 |

| | | |
|--------------|---|-----------------|
| 3 | Analog to Digital Conversion: Sampling, quantization, and coding | 2 |
| 4 | Manipulation of Discrete-Time (DT) Signals: Shifting, scaling, and inversion | 2 |
| 5 | Convolution and Correlation of Discrete-Time Sequences: Understanding DSP fundamentals | 3 |
| 6 | z-Transform in MATLAB: Analysis and computation of z-domain representation | 3 |
| 7 | Frequency Domain Analysis of DT Signals: Fourier analysis and spectral representation | 3 |
| 8 | Design of Finite Impulse Response (FIR) Filter: Techniques and implementation in MATLAB | 4 |
| Total | | 21 Hours |

Course Schedule

| Week | Topic & Details | Teaching-Learning Strategy | Assessment Strategy | CLO Mapping |
|------|---|--|--|--------------|
| 1 | Introduction to MATLAB: Basics, interface navigation, and scripting | Lecture, hands-on MATLAB exercises | Lab performance, viva | CLO 1 |
| 2 | Continuous Time and Discrete-Time Representation of Signals: Visualizing signals in MATLAB | Interactive demonstration, MATLAB implementation | Practical evaluation, report submission | CLO 1, CLO 2 |
| 3 | Analog to Digital Conversion: Sampling, quantization, and aliasing concepts | Lecture, guided simulations | Lab performance, viva | CLO 1, CLO 3 |
| 4 | Manipulation of Discrete-Time (DT) Signals: Scaling, shifting, and reversing signals | MATLAB coding tasks, collaborative learning | Assessment of code accuracy, oral discussion | CLO 2, CLO 3 |
| 5 | Convolution and Correlation of Discrete-Time Sequences: Mathematical and graphical interpretation | Lecture, MATLAB examples | Lab performance, troubleshooting challenges | CLO 2 |
| 6 | z-Transform in MATLAB: Properties and applications in signal processing | Concept-focused explanation, computational tasks | Report evaluation, programming assignment | CLO 2, CLO 3 |

| | | | | |
|----|---|--|---|----------------------------|
| 7 | Frequency Domain Analysis of DT Signals: Fourier Transform and frequency spectrum analysis | Simulation exercises, collaborative learning | Assessment of understanding via assignments | CLO 2, CLO 3 |
| 8 | Design of FIR Filters: Understanding filter specifications and implementing FIR filters in MATLAB | Step-by-step guide, MATLAB filter design | Filter design accuracy assessment, presentation of findings | CLO 4 |
| 9 | Mid-Term Review and Practical Exam | Hands-on problem-solving, concept revision | Mid-term lab exam evaluation | CLO 1, CLO 2, CLO 3, CLO 4 |
| 10 | Advanced FIR Filter Design: Applying windowing techniques and optimizing filter performance | Guided coding, collaborative group discussion | Filter design accuracy, comparative analysis | CLO 4 |
| 11 | Real-Time Signal Processing Applications Using MATLAB | Case studies, practical simulations | Application-based project assessment | CLO 3, CLO 4 |
| 12 | Practical Implementation of Signal Processing Techniques | Collaborative lab activities | Submission of practical outcomes | CLO 3, CLO 4 |
| 13 | Troubleshooting and Debugging MATLAB Scripts for DSP Applications | Problem-solving tasks, peer learning | Debugging accuracy and efficiency evaluation | CLO 3, CLO 4 |
| 14 | Final Project Work: End-to-end signal processing application development | Independent work with mentor guidance | Final project assessment | CLO 4 |
| 15 | Final Project Review: Peer review, project evaluation, and report preparation | Peer feedback, collaborative improvement suggestions | Evaluation based on peer feedback and improvements | CLO 4 |
| 16 | Final Lab Exam | Hands-on practical exam | Final lab exam performance evaluation | CLO 1, CLO 2, CLO 3, CLO 4 |
| 17 | Final Presentation and Wrap-Up | Student presentations, Q&A sessions | Presentation clarity and understanding of concepts | CLO 4 |

Assessment Pattern

- Continuous Assessment

| Bloom's Category | Tests |
|------------------|-------|
| Imitation | 12 |
| Manipulation | 8 |
| Precision | 6 |
| Articulation | 2 |
| Naturalization | 2 |

- Semester End Examination: (SEE):

| Bloom's Category Marks (out of 30) | Tests (20) | Quiz (10) | External Participation in Curricular/Co-Curricular Activities (20) |
|------------------------------------|------------|-----------|---|
| Imitation | 06 | 06 | Bloom's Affective Domain: (Attitude or will) <ul style="list-style-type: none"> • Attendance: 10 • Viva-Voca: 5 • Report Submission: 5 |
| Manipulation | 04 | 04 | |
| Precision | 06 | | |
| Articulation | 02 | | |
| Naturalization | 02 | | |

References

1. Oppenheim, A. V., Schafer, R. W., & Buck, J. R. **"Discrete-Time Signal Processing"** (3rd Edition). Pearson Education.
2. Proakis, J. G., & Manolakis, D. G. **"Digital Signal Processing: Principles, Algorithms, and Applications"** (4th Edition). Pearson Prentice Hall.
3. Ingle, V. K., & Proakis, J. G. **"Digital Signal Processing Using MATLAB"** (4th Edition). Cengage Learning.
4. Mitra, S. K. **"Digital Signal Processing: A Computer-Based Approach"** (4th Edition). McGraw Hill Education.
5. MATLAB Documentation and Online Resources:
<https://www.mathworks.com/help/matlab/>

| List of Experiments | |
|----------------------------|---|
| 1. | Introduction to MATLAB. |
| 2. | Continuous time and Discrete-time representation of signals |
| 3. | Analog to digital conversion |
| 4. | Manipulation of Discrete-Time (DT) signals |
| 5. | Convolution and correlation of Discrete-Time sequences |
| 6. | z-Transform in Matlab |
| 7. | Frequency domain analysis of DT signals |
| 8. | Design of Finite Impulse Response (FIR) Filter |

Experiment No.: 01

Experiment Name: Introduction to MATLAB.

Objectives:

- To familiarize with MATLAB and some basic commands of MATLAB.
- To know about variable and variable types in MATLAB.
- To know about Matrix manipulation in MATLAB.
- To learn about plotting 2D graphs in MATLAB.
- To become familiar with MATLAB script/editor.
- To become familiar with conditional operators and loops in MATLAB.
- To learn about debugging programs in MATLAB.
- To know how to develop a user-defined function in MATLAB.

Theory:

MATLAB: MATLAB is a powerful tool used by engineers and scientists for numerical computation, data analysis, and algorithm development. Let's embark on a journey to understand its capabilities and functionalities.

1. Familiarization with MATLAB and Basic Commands: MATLAB, short for Matrix Laboratory, is a high-level language that operates primarily on matrices and arrays.

To get started, some basic commands include:

- **clc** - Clear the Command Window.
- **clear** - Remove variables from the workspace.
- **help** - Display help for MATLAB functions.
- **doc** - Display detailed documentation for any MATLAB function

2. Variables and Variable Types: In MATLAB, variables are created when you assign them a value. Variable types range from numeric arrays, characters, strings, tables, and structures, to cell arrays. By default, MATLAB stores numeric variables as double-precision floating-point values.

3. Naming Convention: In MATLAB, file name & variable names must start with Alphabet. A mix of letters, numbers & Underscore “_” can be used. No space is allowed.

4. Matrix Manipulation: MATLAB excels at matrix manipulation. We can perform operations like addition, subtraction, multiplication, and division on matrices with ease. Functions like zeros, ones, and eye create matrices of zeros, ones, and identity matrices, respectively.

4. Plotting 2D Graphs: Plotting is a visual way to represent data. The plot function is used to create 2D line plots.

For example, to plot the sine function, we should use:

Code : $x = 0:\pi/100:2\pi;$
 $y = \sin(x);$


```
plot(x, y);
```

This would display a sine wave on the graph.

5. MATLAB Script/Editor: The MATLAB Editor is where we write, debug, and run our MATLAB code. Scripts are files that contain code and are executed in sequence. We can create scripts by clicking the New Script button in the Home tab or using the edit function.

6. Conditional Operators and Loops: Conditional operators (if, else, switch) and loops (for, while) control the flow of your program.

For example, a simple for loop to print numbers from 1 to 5 would look like:

```
Code: for i = 1:5
        disp(i);
    end
```

This will display numbers 1 through 5 in the Command Window.

7. Debugging Programs: Debugging is crucial for finding and fixing errors. MATLAB provides an interactive environment for debugging, where we can set breakpoints, step through code, and inspect variables. The Editor highlights lines of code where errors occur, making it easier to diagnose problems.

8. Developing User-Defined Functions: User-defined functions allow us to write reusable code blocks. A simple function to calculate the area of a circle might look like:

```
Code: function area = calcArea(radius)
        area = pi * radius^2;
    end
```

We can call this function with a radius value to get the area.

Conclusion:

MATLAB, an integrated environment designed for complex numerical computations and advanced programming. We have explored the core features of MATLAB, encompassing basic commands, variable types, matrix operations, and 2D graphical plotting. Additionally, we have examined the functionalities of the MATLAB script/editor, the application of conditional operators and loops, the process of debugging programs, and the methodology for developing user-defined functions. Equipped with this foundational knowledge, participants are now prepared to employ MATLAB for diverse applications across various domains. The competencies developed are instrumental for tasks ranging from data analysis to algorithmic development, and scientific inquiry. As we conclude, it is encouraged to persistently engage with MATLAB to refine and expand one's skill set. Continued practice will ensure proficiency and pave the way for innovative contributions to the field of MATLAB programming.

Experiment No.: 02

Experiment Name: Continuous-time and Discrete-time Representation of signals.

Objectives:

- To represent (plot) basic signals (sine, cosine, sinc, unit impulse, unit step, unit ramp, exponential signal) in MATLAB.
- To generate noise signals in MATLAB.

Required Apparatus:

- MATLAB Software

Theory:

- **Sine Wave:** A sine wave is a smooth, periodic oscillation that is mathematically described by the equation, $y = [A \sin(2\pi f t)]$

Where,

A = the amplitude.

f = the frequency in Hz.

t = the continuous time variable.

- **Cosine Wave:** A cosine wave is a smooth, periodic oscillation similar to a sine wave represented by the equation, $y = [A \cos(2\pi f t)]$

Where,

A = the amplitude.

f = the frequency in Hz.

t = the continuous time variable.

- **Sinc Wave:** The sinc wave is a function that describes a sine wave divided by its argument, commonly used in signal processing for its property of being the Fourier transform of a rectangular pulse. The equation represents the Sinc wave, $y = [A \text{sinc}(2\pi f t)]$.

- **Unit Impulse Signal:** Unit Impulse Signal is a function that is zero for negative time and one for positive time.

The equation for unit impulse signal: $y = [\text{zeros}(1, \text{abs}(a)), \text{ones}(1, 1), \text{zeros}(1, b)]$

It's defined as:

$$\delta(n) = \begin{cases} 1, & \text{for } n = 0; \\ 0, & \text{for } n \neq 0; \end{cases}$$

- **Unit Step Signal:** Unit Step Signal is a function that is zero everywhere except at zero, where it is infinitely high and has an integral of one.

The equation for unit impulse signal: $y = [\text{zeros}(1, \text{abs}(a)), \text{ones}(1,1), \text{ones}(1,b)]$;
It's defined as:

$$U(n) = \begin{cases} 1, & \text{for } n \geq 0; \\ 0, & \text{for } n < 0; \end{cases}$$

- **Unit Ramp Signal:** Unit Ramp Signal is a function that increases linearly with time from zero starting at time zero.
It's defined as:

$$U_r(n) = \begin{cases} 1, & \text{for } n \geq 0; \\ 0, & \text{for } n < 0; \end{cases}$$

- **Exponential Signal:** Exponential Signal is a signal that changes over time according to an exponential function, typically representing growth or decay.
It's defined as:

$$x(n) = a^n \quad \text{for all } n;$$

- **Noise Signal:** Noise signal is an unwanted modification that a signal may suffer during capture, storage, transmission, processing, or conversion.
For this experiment uniformly distributed white noise rand & randn function have been used.

Code:

As Continuous Time Plot:

- **Sine Wave:**

```
clc
clear
Fs=100
T=1/Fs
F=2
d=2; t=-d:T:d
y=sin(2*pi*F*t);
plot(t,y,'LineWidth',1);
xlabel('Sample Position');
ylabel('Amplitude');
title('Sine Wave Continuous Plot (Student ID)');
grid on;
```

- **Cosine Wave:**

```
clc
clear
Fs=100
T=1/Fs
F=2
d=2; t=-d:T:d
```

```

y=cos(2*pi*F*t);
plot(t,y,'LineWidth',1);
xlabel('Sample Position');
ylabel('Amplitude');
title('Cosine Wave Continuous Plot (Student ID)');
grid on;

```

▪ **Sinc Wave:**

```

clc
clear
Fs=100
T=1/Fs
F=2
d=2; t=-d:T:d
y=sinc(2*pi*F*t);
plot(t,y,'LineWidth',1);
xlabel('Sample Position');
ylabel('Amplitude');
title('sinc Wave Continuous Plot (Student ID)');
grid on;

```

As Discrete Time Stem:

▪ **Sine Wave:**

```

clc
clear
Fs=100
T=1/Fs
F=2
d=2; t=-d:T:d
y=sin(2*pi*F*t);
stem(t,y,'LineWidth',1);
xlabel('Sample Position');
ylabel('Amplitude');
title('Sine Wave Discrete Plot (Student ID)');
grid on;

```

▪ **Cosine Wave:**

```

clc
clear
Fs=100
T=1/Fs
F=2
d=2; t=-d:T:d
y=cos(2*pi*F*t);
stem(t,y,'LineWidth',1);
xlabel('Sample Position');
ylabel('Amplitude');
title('Cosine Wave Discrete Plot (Student ID)');
grid on;

```

▪ **Sinc Wave:**

```
clc
clear
Fs=100
T=1/Fs
F=2
d=2; t=-d:T:d
y=sinc(2*pi*F*t);
stem(t,y,'LineWidth',1);
xlabel('Sample Position');
ylabel('Amplitude');
title('Sinc Wave Discrete Plot (Student ID)');
grid on;
```

▪ **Unit Impulse Signal:**

```
clc
clear
a=-5
b=10
n=a:b
y=[zeros(1,abs(a)),ones(1,1),zeros(1,b)];
stem(n,y,'LineWidth',1,'Color','b');
title('Unit Impulse Signal (Student ID)');
xlabel('Sample position');
ylabel('Amplitude');
```

▪ **Unit Step Signal:**

```
clc
clear
a=-5
b=10
n=a:b
y=[zeros(1,abs(a)),ones(1,1),ones(1,b)];
stem(n,y,'LineWidth',1,'Color','b');
title('Unit Step Signal (Student ID)');
xlabel('Sample position');
ylabel('Amplitude');
```

▪ **Unit Ramp Signal:**

```
clc
clear;
n=0:10
y=n
stem(n,y,'LineWidth',1,'Color','b');
title('Unit Ramp Signal (Student ID)');
xlabel('Sample position');
ylabel('Amplitude');
```

▪ **Exponential Signal:**

```

clc
clear
n=0:100
a1=0.9
y1=a1.^n
subplot(221)
stem(n,y1,'b')
title('Exponential sequence when  $0 < a < 1$ ')
xlabel('Sample position (n)')
ylabel('Amplitude')
a2=1.1
y2=a2.^n
subplot(2,2,2)
stem(n,y2,'b')
title('Exponential sequence when  $a > 1$ ')
xlabel('Sample position (n)')
ylabel('Amplitude')
a3=0.9
y3=a3.^n
subplot(223)
stem(n,y3,'b')
title('Exponential sequence when  $1 < a < 0$ ')
xlabel('Sample position (n)')
ylabel('Amplitude')
a4=1.1
y4=a4.^n
subplot(2,2,4)
stem(n,y4,'b')
title('Exponential sequence when  $a < 1$ ')
xlabel('Sample position (n)')
ylabel('Amplitude')

```

▪ **Noise Signal Generation:**

```

clc
clear
Fs=1000 % sampling frequency
Ts=1/Fs % sampling period or time step
n=0:Ts:1-Ts % signal duration
f1=9; f2=39; f3=25; % Frequency of the three Sine Signals
y1=10*sin(2*pi*f1*n)
y2=10*sin(2*pi*f2*n)
y3=10*sin(2*pi*f3*n)
y=y1+y2+y3
ny=length(y)
N=2^nextpow2(ny)
yfft=fft(y,N)
yfft2=yfft(1:N/2)
xfft=Fs*(0:N/2-1)/N

```

```
%Plots
subplot(3,3,1); plot(n,y1,'b'); title('Sinwave 1');
subplot(3,3,2); plot(n,y2); title('Sinwave2');
subplot(3,3,3); plot(n,y3); title('Sinwave3');
subplot(3,3,[4 5 6]); plot(n,y);
title('Time domain signal (sum of all sin waves)');
subplot(3,3,[7 8 9]); plot(xfft,abs(yfft2)/max(abs(yfft2)));
title('Frequency domain Signal');
grid on;
```

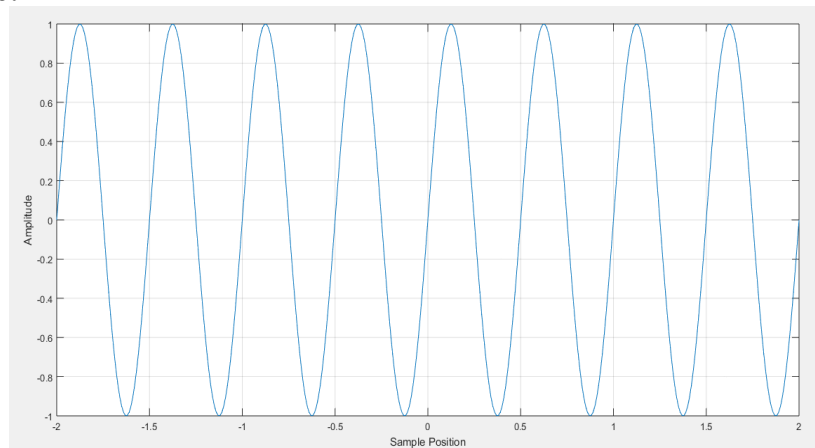
Adding noise to an image:

```
clc
Clear
image=imread('img1.jpg'); image=rgb2gray(image);
noise=6+25*randn(size(image)); subplot(1,3,1);
imshow(image); subplot(1,3,2);
imshow(uint8(noise));
noisy_image=uint8(double(image)+noise); subplot(1,3,3);
imshow(noisy_image);
```

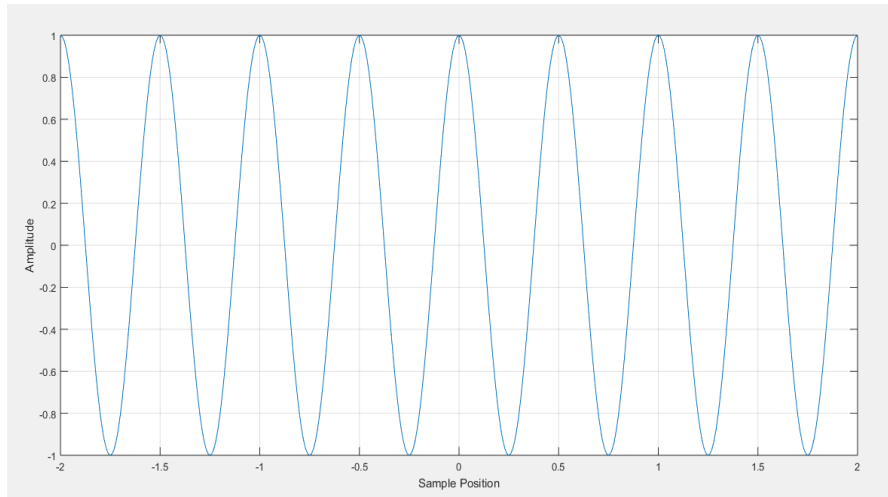
Results:

As Continuous Time Plot:

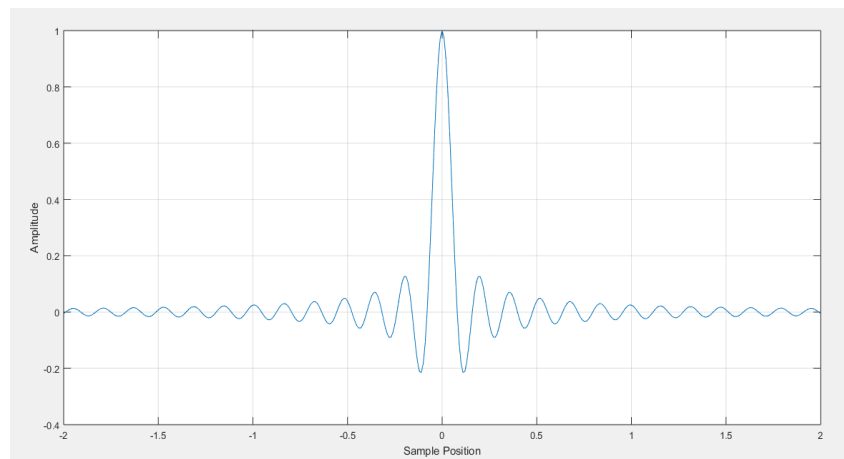
▪ Sine Wave:



▪ Cosine Wave:

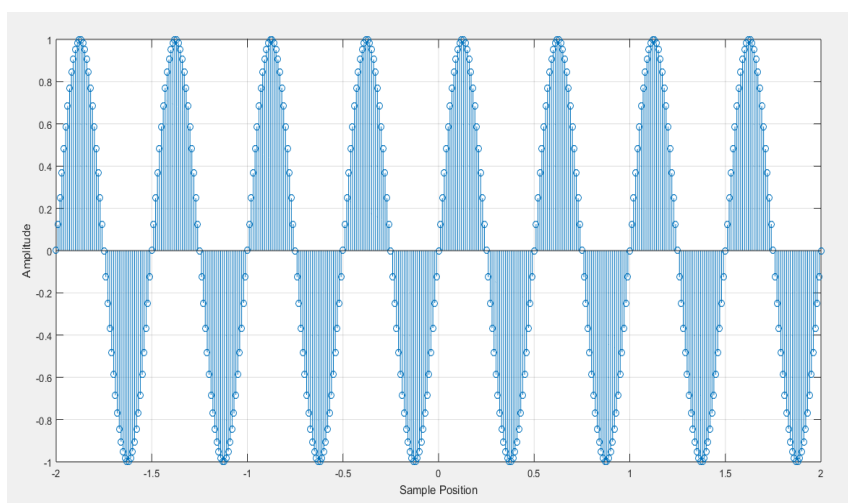


- **Sinc Wave:**

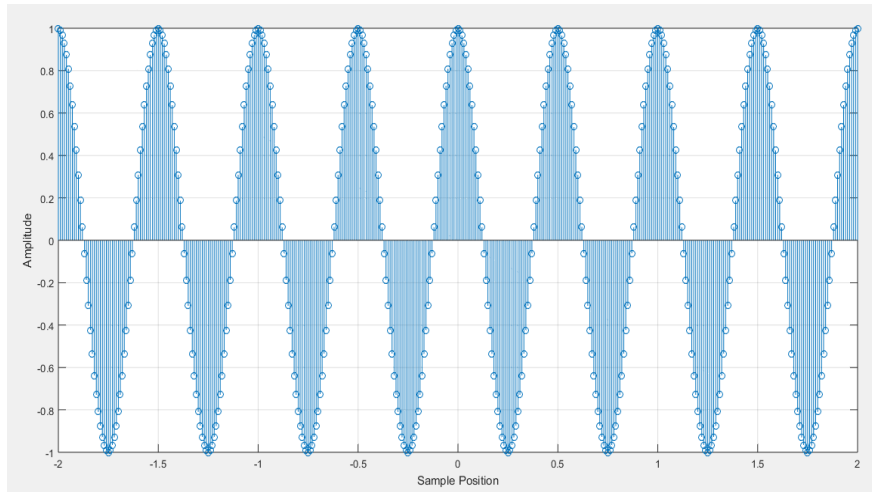


As Discrete Time Stem:

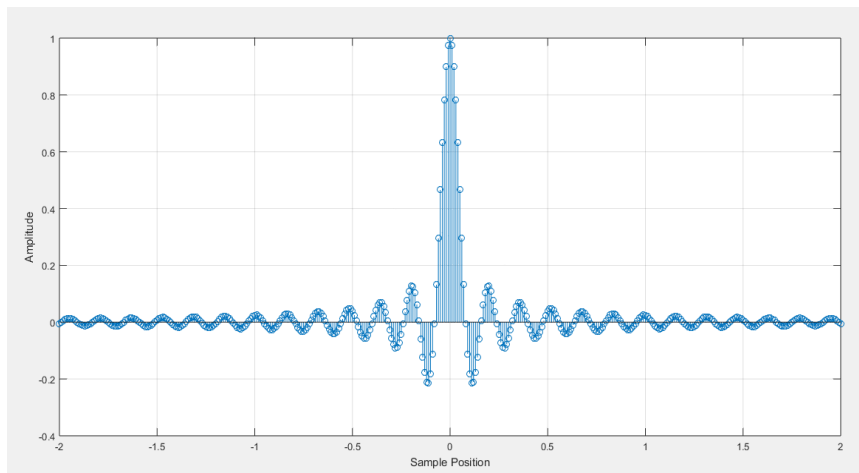
- **Sine Wave:**



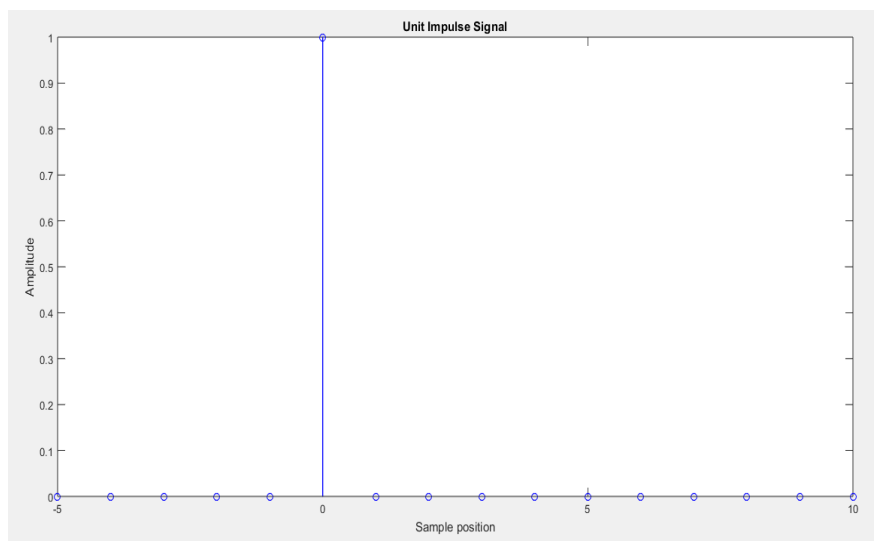
- **Cosine Wave:**



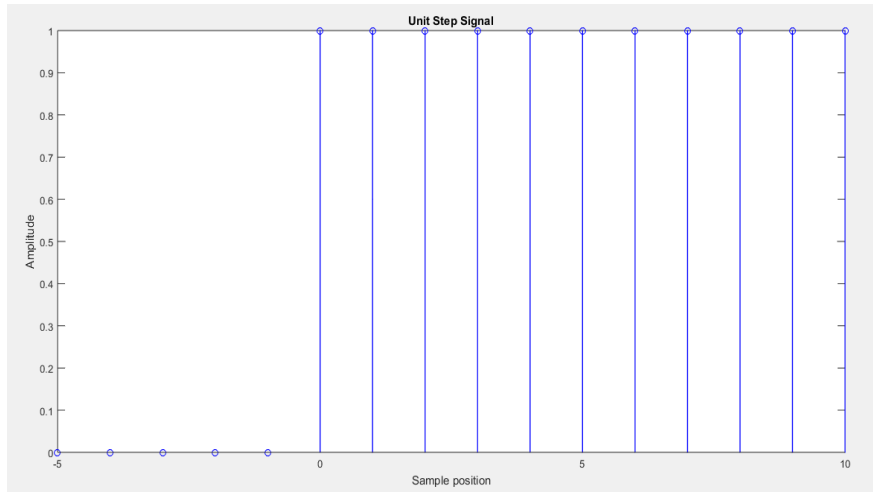
■ Sinc Wave:



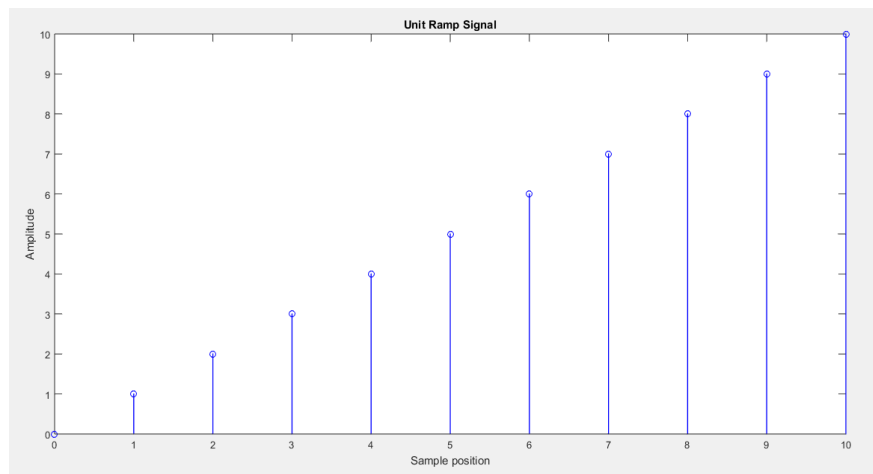
■ Unit Impulse Signal:



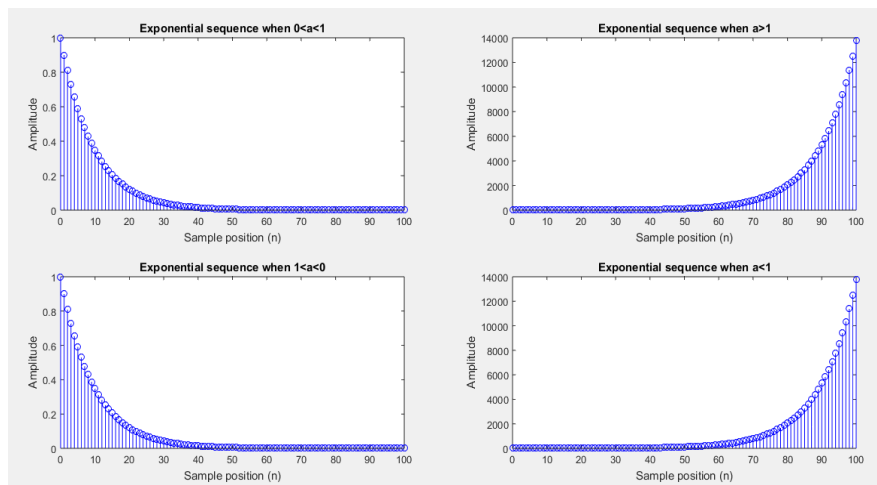
■ Unit Step Signal:



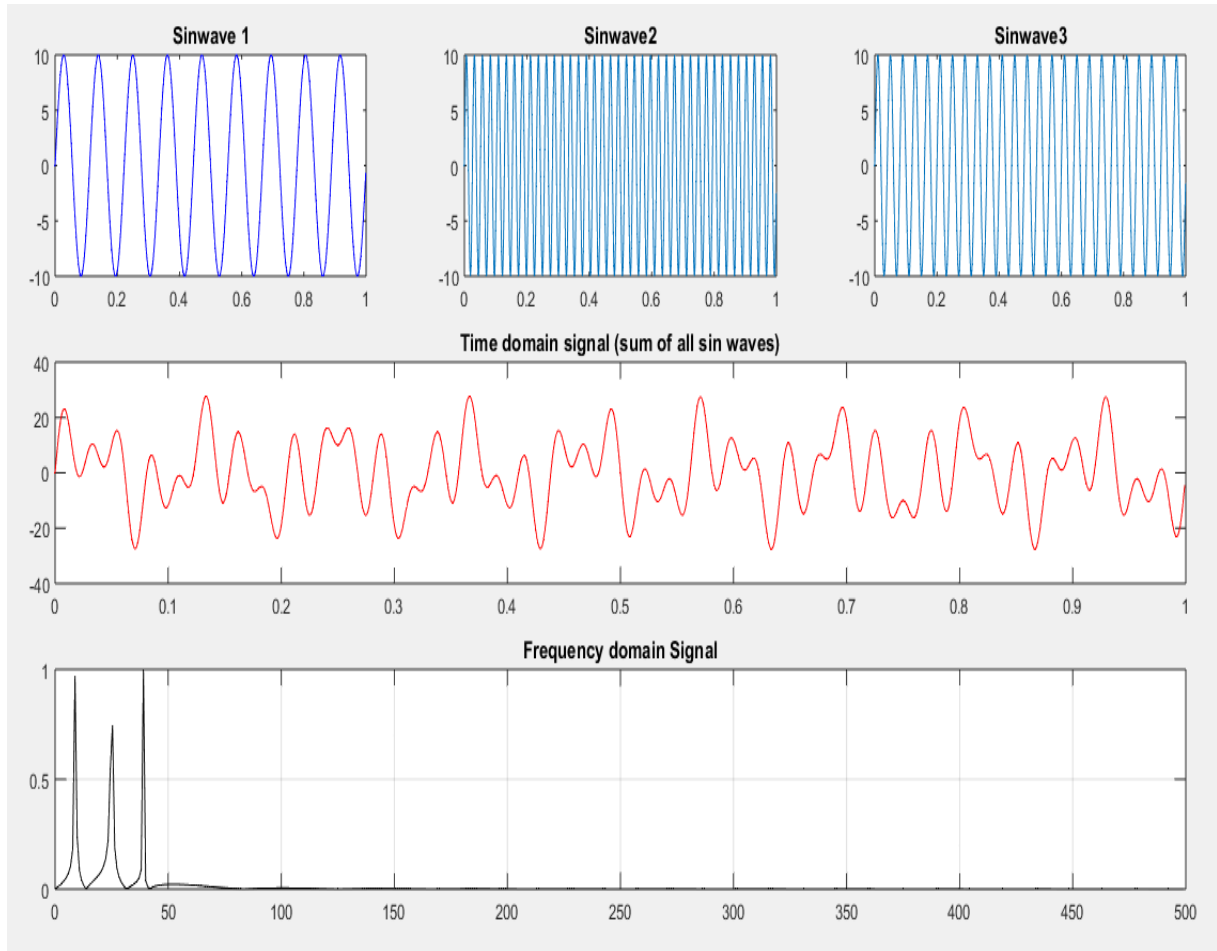
■ Unit Ramp Signal:



■ Exponential Signal:



■ Noise:



Conclusion:

In conclusion, the ability to represent and plot basic signals such as sine, cosine, sinc, unit impulse, unit step, unit ramp, and exponential signals in MATLAB is fundamental to understanding and analyzing systems in Digital Signal Processing (DSP). Moreover, generating noise signals in MATLAB is essential for simulating real-world scenarios and studying the effects of randomness and disturbances on signal integrity. This knowledge serves as a cornerstone for further exploration and application in various engineering and technology fields.

Experiment No.: 03

Experiment Name: Analog Signal to Digital Signal Conversion.

Objectives:

- To understand the analog-to-digital conversion process: Sampling, Quantization and Coding.
- To convert continuous time signal into digital signal using MATLAB code.
- To analyze the effect of sampling rate and quantization level in A/D conversion.

Required Apparatus:

- MATLAB Software.

Theory:

An analog signal is a continuous signal that represents variations in a physical quantity, such as sound or light, and can vary infinitely within a certain range. It's often used in contrast to a digital signal, which is discrete and represents information in binary form.

Analog Signals to Digital Signals conversion is Essential for several reasons:

- **Improved Performance:** Digital signals are less susceptible to noise and distortion, leading to clearer and more accurate communication.
- **Power Efficiency:** Digital systems generally use less power than their analog counterparts.
- **Ease of Encryption:** Digital signals can be encrypted more easily, enhancing security during transmission.
- **Simpler Storage:** Digital data can be stored more compactly and is less prone to degradation over time.
- **Processing Capabilities:** Digital signals can be processed and manipulated using algorithms for tasks like filtering, compression, and error correction.
- **Transmission:** Digital signals can be transmitted over long distances without significant loss of quality.

There are three steps for converting an analog signal to digital.

- **Sampling:** Converts Continuous Time (CT) signal into Discrete Time (DT) Signal.
- **Quantization:** Converts continuous-valued signal into discrete-valued signal.
- **Coding:** Converts discrete value into Binary Code.

Code:

```
clc
clear
Fs= 1200; % Vary the value of
Fs=150,400,1200,1600,3000,5000
bit=3; % Vary the value of
bit=2,3,4,8,16
dt= 1/Fs; StopTime= 0.025;
t=0:0.00001:StopTime; n=0:1:(StopTime/dt); Fc =100;
```

```

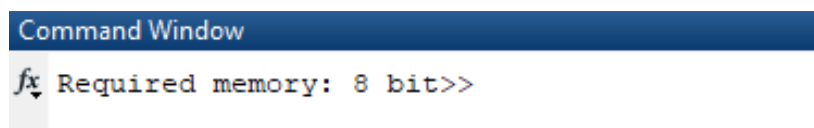
Ya=5*sin(2*pi*Fc*t)+3*cos(2*pi*6*Fc*t);
y_max=max(Ya); y_min=min(Ya);
y = 5*sin(2*pi*(Fc/Fs)*n)+3*cos(2*pi*6*(Fc/Fs)*n);
ns=length(y); q_out=zeros(1,ns);
del=(y_max-y_min)/(2^bit);
low=y_min+(del/2); high=y_max-(del/2);
for h=low:del:high
    for b=1:ns
        if((h-del/2)<y(b)) && ((y(b)<=(h+del/2)))
            q_out(b)=h;
        end
    end
end
end
q_error=q_out-y; r_memory=bit*ns;
fprintf('Required memory: %d bit',r_memory);
%Plot
subplot(321); plot(t,Ya,'r','Linewidth',1);
xlabel('Time'); ylabel('amplitude'); title('Analog Signal');
subplot(322); stem(n,y,'c','Linewidth',1);
xlabel('sample number'); ylabel('amplitude');
title('Discrete Time Continuous Valued Signal');
subplot(323); stem(n,q_out,'m','Linewidth',1);
xlabel('Sample number'); ylabel('amplitude');
title('Discrete Time Discrete Valued Signal');
subplot(324); plot(n,q_error,'b','Linewidth',1);
xlabel('Sample Number'); ylabel('amplitude');
title('Quantization Noise');
subplot(3,2,5); plot(n.*dt,q_out,'g','Linewidth',1);
xlabel('Time'); ylabel('amplitude');
title('Reconstructed Analog Signal');
subplot(3,2,6); plot(t,Ya,'r',n.*dt,q_out,'g','Linewidth',1);
xlabel('Time'); ylabel('amplitude');
title('Reconstructed Analog Signal Vs Analog Signal');
hold on;

```

Result:

- For,
 $F_s = 150$;
 $\text{Bit} = 2$;

Command Window:



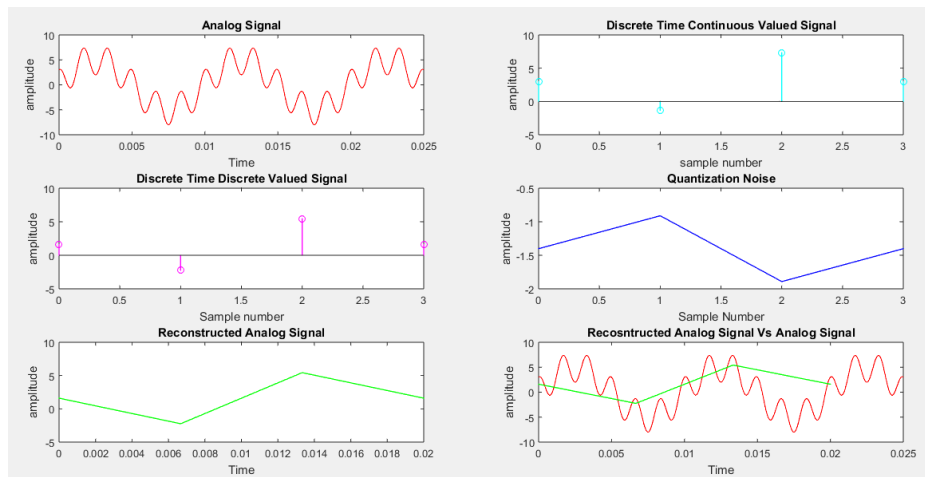
Command Window

```

fx Required memory: 8 bit>>

```

Figure:

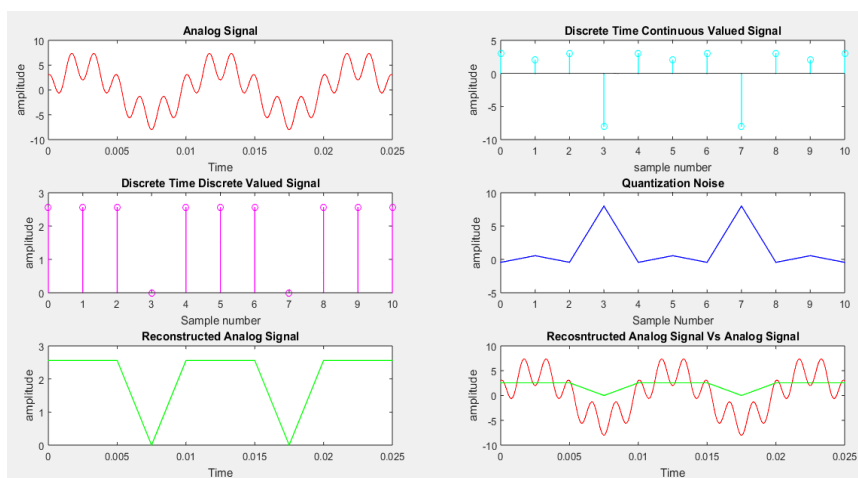


- For,
 $F_s = 400$;
 $\text{Bit} = 3$;

Command Window:

```
Command Window
fx Required memory: 33 bit>>
```

Figure:

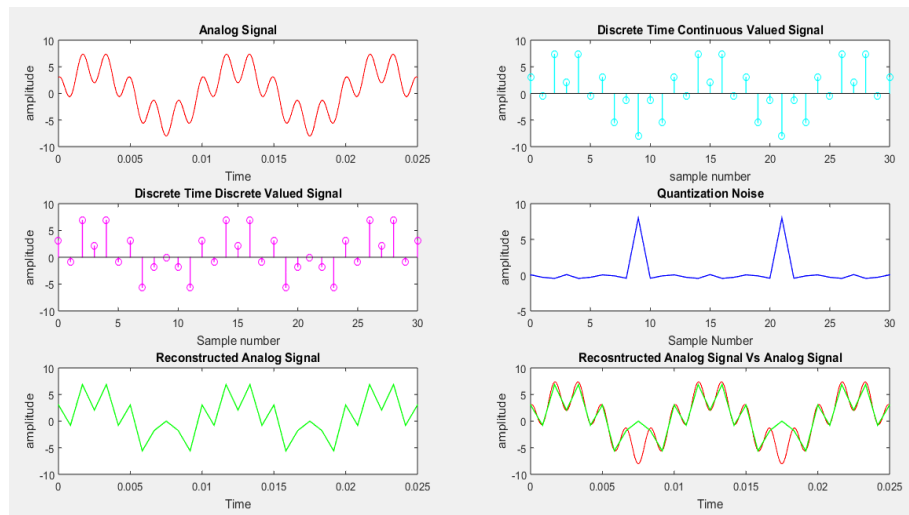


- For,
 $F_s = 1200$;
 $\text{Bit} = 4$;

Command Window:

```
Command Window
fx Required memory: 93 bit>> |
```

Figure:

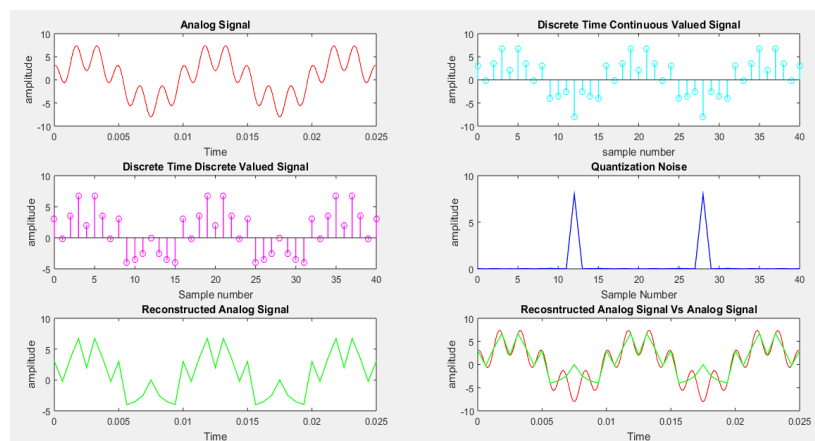


- For,
 $F_s = 1600$;
 $\text{Bit} = 8$;

Command Window:

```
Command Window
fx Required memory: 124 bit>> |
```

Figure:



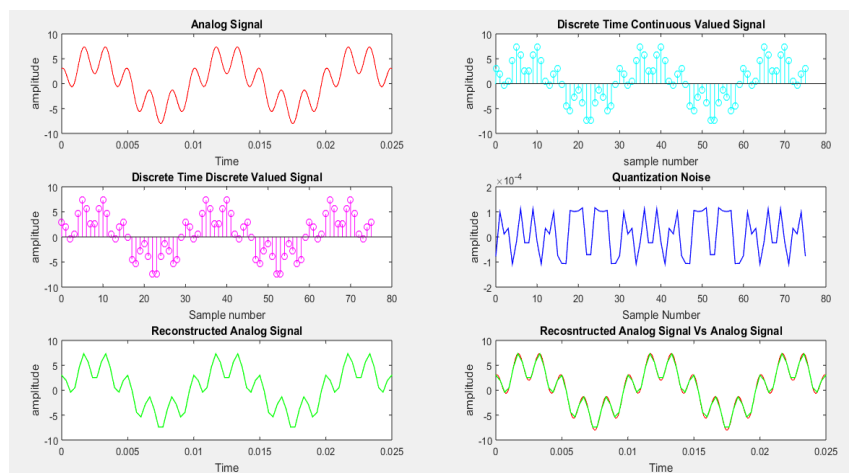
- For,
 $F_s = 3000$;
 $\text{Bit} = 16$;

Command Window:

Command Window

```
fx Required memory: 328 bit>>
```

Figure:



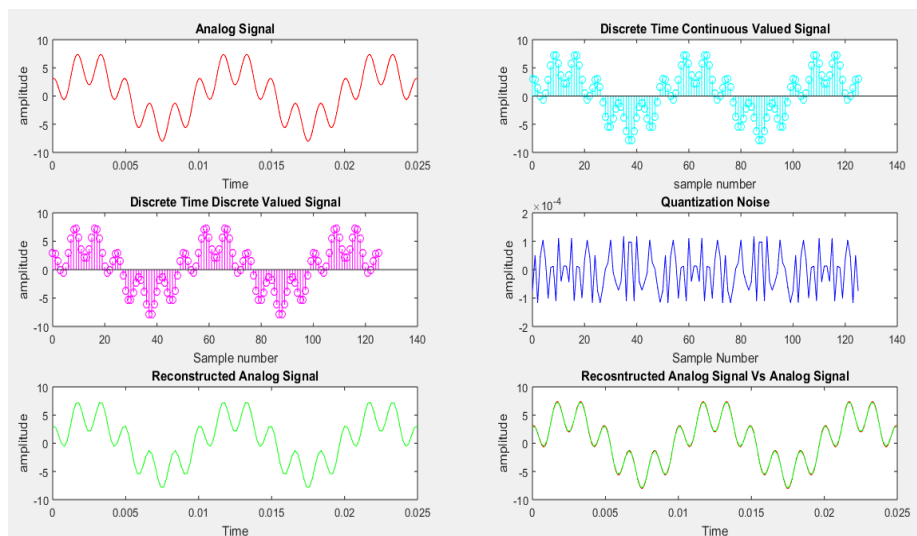
- For,
 $F_s = 5000$;
 $\text{Bit} = 16$;

Command Window:

Command Window

```
fx Required memory: 2016 bit>> |
```

Figure:



Conclusion:

The exploration of the analog-to-digital conversion process, including sampling, quantization, and coding, is a pivotal aspect of modern signal processing. Through MATLAB coding exercises, we've gained practical experience in converting continuous-time signals into their digital counterparts. Additionally, we've observed firsthand the impact of varying sampling rates and quantization levels on the fidelity of A/D conversion. This understanding is crucial for designing efficient digital systems that accurately represent real-world analog signals, ensuring high-quality signal reconstruction and reliable data transmission in various applications.

Experiment No.: 04

Experiment Name: Manipulation of Discrete-Time (DT) signals.

Objectives:

- To develop MATLAB program to add/subtract/multiply two discrete time sequences.
- To develop function to shift and fold the DT sequence.
- To develop the function to find the symmetric (even) and anti-symmetric (odd) part of DT signal.

Required Apparatus:

- MATLAB Software.

Theory:

1. Addition/Subtraction/Multiplication of two Discrete-Time Sequences:

Operations on discrete-time sequences are performed element-wise.

- **Addition:** The addition of two signals is the process of combining them by adding their corresponding amplitudes at each point in time or sample index.
Equation: $z[n] = x[n] + y[n]$
- **Subtraction:** The subtraction of two signals involves calculating the difference between their corresponding amplitudes at each point in time or sample index.
Equation: $z[n] = x[n] - y[n]$
- **Multiplication:** The subtraction of two signals involves calculating the difference between their corresponding amplitudes at each point in time or sample index.
Equation: $z[n] = x[n] .* y[n]$

These operations require the sequences to be of the same length or appropriately padded

2. Shifting and Folding Discrete-Time Sequences:

Shifting involves moving the sequence in time.

- **Right Shift:** The right shift of a signal refers to delaying the signal in time. In discrete-time signal processing, it means that each sample of the signal is moved to the right by a certain number of sample intervals.
Equation: $y[n] = x[n-k]$
Here;
 k = Samples of delay.
- **Left Shift:** The right shift of a signal refers to advancing the signal in time. In discrete-time signal processing, it means that each sample of the signal is moved to the left by a certain number of sample intervals.
Equation: $y[n] = x[n+k]$
Here;

k= Samples of advance.

- **Folding:** Folding, or time-reversal, is an operation in signal processing where a signal is flipped around the vertical axis. It mirrors the original signal.

Equation: $y[n] = x[-n]$

Shifting and Folding are used to analyze system properties like time-invariance.

3. Symmetric (Even) and Anti-Symmetric (Odd) Parts of a Discrete-Time Signal:

Any discrete-time signal can be decomposed into symmetric and anti-symmetric parts.

- **Even Symmetric:** Even symmetric signals are those that are identical to their mirror image about the vertical axis, like a cosine wave.

Equation: $x(t) = x(-t)$

- **Odd Symmetric:** Odd symmetric signals are antisymmetric about the vertical axis, like a sine wave.

Equation: $x(t) = -x(-t)$

This decomposition is useful for analyzing signals and systems, especially in the context of Fourier series and transforms.

Code:

1. Addition/Subtraction/Multiplication of two Discrete-Time Sequences:

- **Addition:**

Function

```
function [y,n]=sigadd(x1,n1,x2,n2)
n=min(min(n1),min(n2)):max(max(n1),max(n2));
y1=zeros(1,length(n));
y2=y1; y1((n>=min(n1)) & (n<=max(n1)))=x1;
y2((n>=min(n2)) & (n<=max(n2)))=x2; y=y1+y2;
```

Script

```
clc
clear
x1=input('x1(n):');
n11=input('Starting point of x1(n):');
n1=n11:n11+length(x1)-1;
x2=input('x2(n):');
n22=input('Starting point of x2(n):');
n2=n22:n22+length(x2)-1;
[y,n]=sigadd(x1,n1,x2,n2);
subplot(311);
stem(n1,x1);
title('Signal x1(n1)');
subplot(312);
stem(n2,x2);
title('Signal x2(n2)');
subplot(313);
```

```
stem(n,y);
title('Signal y(n)=x1(n) + x2(n)');
```

▪ Subtraction:

Function

```
function [y,n]=sigsub(x1,n1,x2,n2)
n=min(min(n1),min(n2)):max(max(n1),max(n2));
y1=zeros(1,length(n));
y2=y1; y1((n>=min(n1)) & (n<=max(n1)))=x1;
y2((n>=min(n2)) & (n<=max(n2)))=x2; y=y1+y2;
```

Script

```
n1=-3:2;
x1=[1,5,4,3,7,0];
n2=-5:5;
x2=[0 1 2 3 3 9 8 5 5 4 4];
[y,n]=sigsub(x1,n1,x2,n2);
subplot(311);
stem(n1,x1);
title('Signal x1(n1)');
subplot(312);
stem(n2,x2);
title('Signal x2(n2)');
subplot(313);
stem(n,y);
title('Signal y(n)=x1(n) - x2(n)');
```

▪ Multiplication:

```
clc
clear
% Define two discrete-time signals
n = 0:10; % Sample index from 0 to 10
signal1 = sin(2*pi*0.1*n); % A sine wave with frequency of 0.1 Hz
signal2 = cos(2*pi*0.1*n); % A cosine wave with frequency of 0.1 Hz
% Subtract the two signals
subtractedSignal = signal1 - signal2;
% Plot the original signals and the subtracted signal figure;
subplot(3,1,1);
stem(n, signal1, 'filled');
title('Signal 1 : Sine Wave');
subplot(3,1,2);
stem(n, signal2, 'filled');
```

```

title('Signal 2 : Cosine Wave');
subplot(3,1,3);
stem(n, subtractedSignal, 'filled', 'r');
title('Subtracted Signal : Signal 1 - Signal 2');
xlabel('Sample Index');
ylabel('Amplitude');

```

2. Shifting and Folding Discrete-Time Sequences:

▪ Shift:

Function:

```

function [y,n]=sigshift(x,m,k)
%implement y(n)=x(n-k)
n=m+k;
y=x;

```

Script:

```

clc
clear
x1=input('x(n):')
n1=input('Starting point of x(n)')
n=n1:n1+length(x1)-1;
[y1,n11]=sigshift(x1,n,-5);
[y2,n22]=sigshift(x1,n,3);
subplot(311);
stem(n1,x1);
title('Signal x1(n1)');
subplot(312);
stem(n11,y1);
title('Left shift of x1(n1)');
subplot(313);
stem(n22,y2);
title('Right shift of x1(n1)');

```

▪ Folding:

Function:

```

function [y,n]=sigfold(x,m)
% implementation of y(n)=x(-n)
y=fliplr(x);
n=-fliplr(m);

```

Script:

```

clc
clear
x1=input('x(n):')

```

```

n1=input('Starting point of x(n)')
n=n1:n1+length(x1)-1;
[y,n]=sigfold(x2,n2);
subplot(211);
stem(n2,x2);
title('Signal x2(n2)');
subplot(212);
stem(n,y);
title('Signal y=x2(-n2)');

```

3. Symmetric (Even) and Anti-Symmetric (Odd) Parts of a Discrete-Time Signal:

Function:

```

function [xe,ne,xo,no]=even_odd(x,n)
[X,N]=sigfold(x,n);
[xe,ne]=sigadd(x,n,X,N);
xe=0.5*xe;
[xo,no]=sigsub(x,n,X,N);
xo=0.5*xo;

```

Script:

```

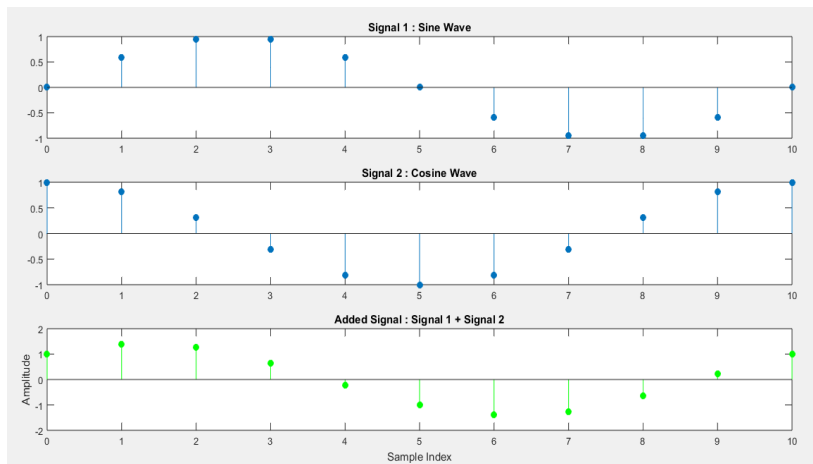
clc
clear
x1=input('x(n):')
n1=input('Starting point of x(n)')
n=n1:n1+length(x1)-1;
[xe,ne,xo,no]=even_odd(x,n);
subplot(2,2,[1 2]);
stem(n,x);
title('Original Signal');
subplot(2,2,3);
stem(ne,xe);
title('Even Component');
subplot(2,2,4);
stem(no,xo);
title('Odd Component');

```

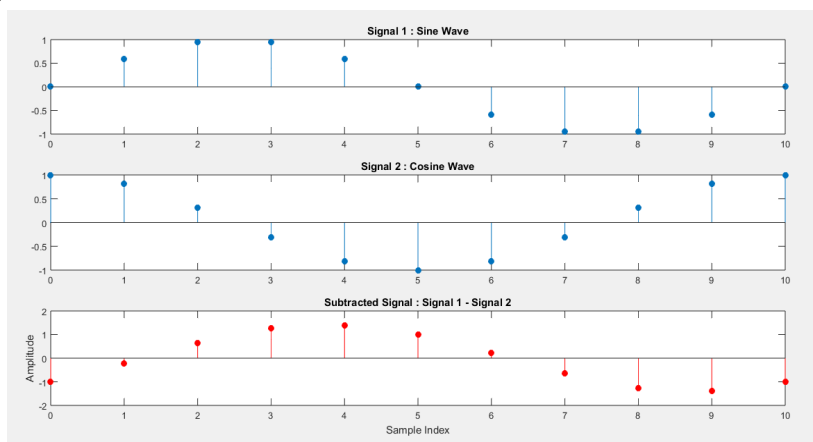
Result:

1. Addition/Subtraction/Multiplication of two Discrete-Time Sequences:

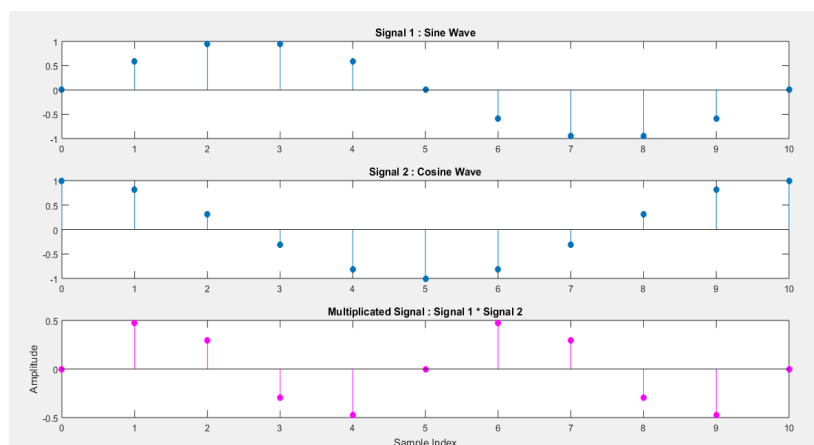
▪ Addition:



■ Subtraction:

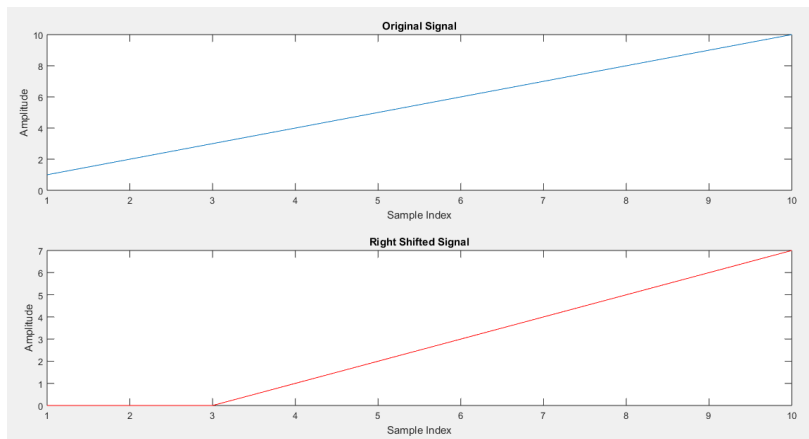


■ Multiplication:

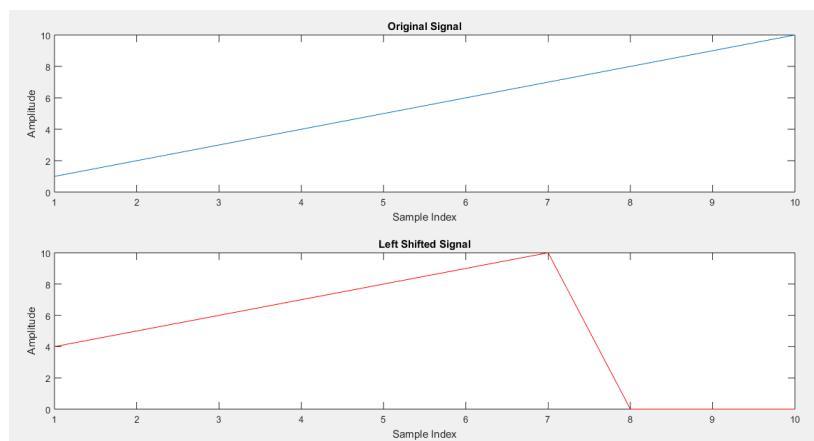


2. Shifting and Folding Discrete-Time Sequences:

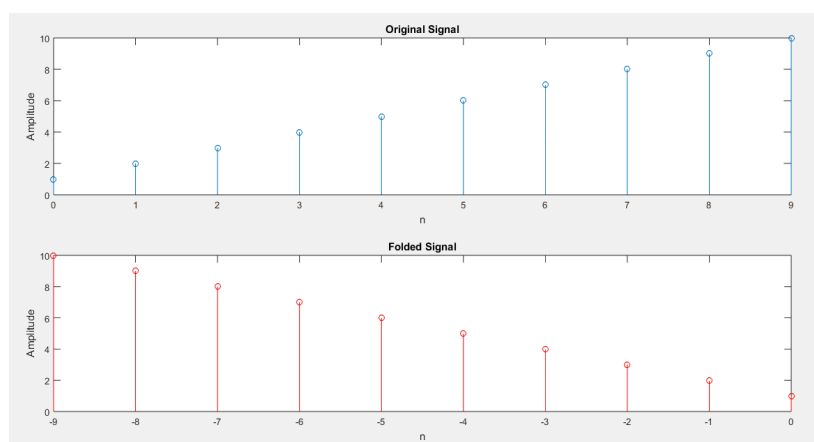
■ Right Shift:



■ Left Shift:

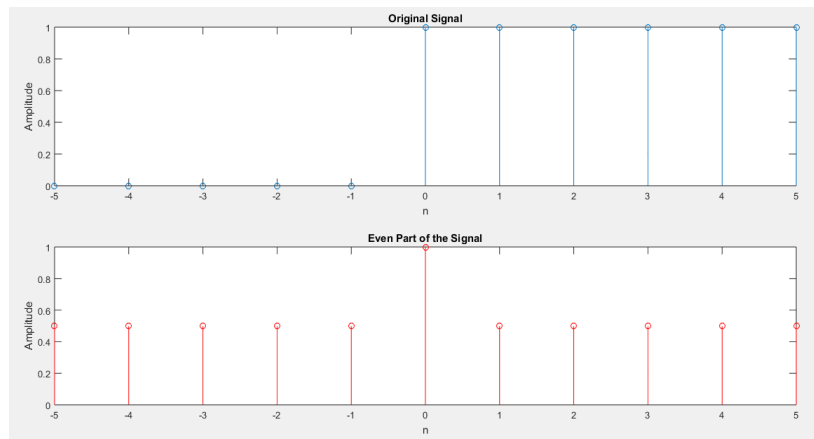


■ Folding:

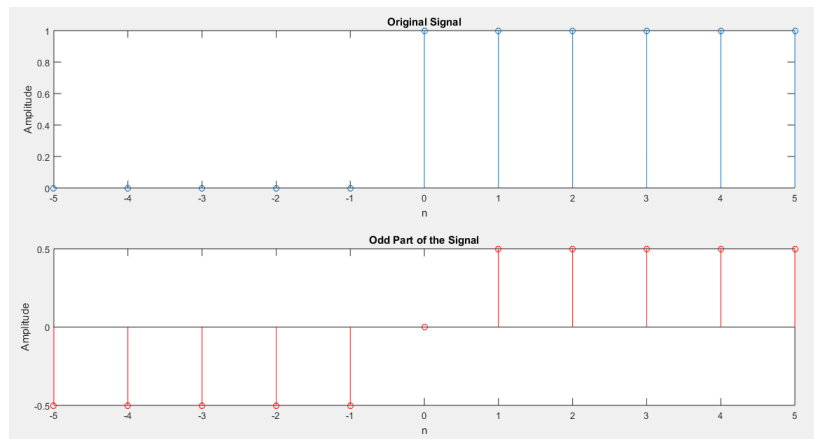


3. Symmetric (Even) and Anti-Symmetric (Odd) Parts of a Discrete-Time Signal:

■ Even Symmetric:



- **Odd Symmetric:**



Conclusion:

The development of MATLAB programs to perform arithmetic operations on DT sequences such as addition, subtraction, and multiplication allows for the manipulation and analysis of signals in various applications. The ability to shift and fold DT sequences further extends this capability, enabling the examination of signal properties in different time frames and orientations. Moreover, the functions to extract symmetric (even) and anti-symmetric (odd) parts of a DT signal are crucial for understanding the inherent characteristics of signals, which can be pivotal in areas like system analysis and filter design. These operations form the building blocks for more complex signal processing tasks and are instrumental in the field of digital signal processing.

Experiment No.: 05

Experiment Name: Convolution and Correlation of Discrete-Time sequences.

Objectives:

- To develop MATLAB program to find the convolution sum of two discrete-time sequence.
- To develop program to calculate the correlation of two DT sequence.
- Use of MATLAB built-in function to find the convolution and correlation sequence of two DT sequence.
- To observe the application of correlation in active sonar application.

Required Apparatus:

- MATLAB Software.

Theory:

- **Convolution:** Convolution is a mathematical operation that combines two functions to produce a third function, which represents how the shape of one function is modified by the other. In the context of signal processing, it's often used to apply filters to signals or to find the overlap between two signals.

Equation:

$$y(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k)$$

- **Correlation:** Correlation refers to the measure of similarity between two signals. It quantifies how much one signal resembles another over time. In signal processing, there are two main types of correlations;

1. **Cross-correlation:** This measures the similarity between two different signals, often used to find the time delay between them or to match a known pattern within a signal.

Equation:

$$r_{xy} = \sum_{n=-\infty}^{\infty} x(n)y(n)$$

2. **Auto-correlation:** This measures the similarity of a signal with a delayed version of itself, which can reveal repeating patterns, such as periodic signals, or identify the fundamental frequency of a signal.

Equation:

$$r_{xy} = \sum_{n=-\infty}^{\infty} x(n)x(n)$$

Both types of correlation can provide valuable insights into the characteristics and behavior of signals in various applications, such as communications, radar, and audio processing.

Code:

▪ Convolution:

```
clc;
clear
x=input('Input Sequence x(n):');
n1= input('Input Starting Point of x(n):');
h=input('Input Sequence h(n):');
m1= input('Input Starting Point h(n):');
n=n1:n1+length(x)-1;
m=m1:m1+length(h)-1;
ny=(min(n) + min(m)) : (max(n) + max(m));
y=conv(x,h); % Built in Function for Convolution
disp('convolution of sequence x(n) & h(n):');
disp(y);
% Plot
subplot(311); stem(n,x,'b');
title('Signal x(n)'); grid on; xlabel('n'); ylabel('x(n)');
subplot(312); stem(m,h,'g');
title('Signal y(n)'); grid on; xlabel('n'); ylabel('h(n)');
subplot(313); stem(ny,y,'m');
title('Convolution');
grid on; xlabel('n'); ylabel('x(n)*h(n)');
```

▪ Deconvolution:

▪ Cross-Correlation:

```
clc;
clear;
n=0:6;
x=[0.1 0.2 -0.1 4.1 -2 1.5 -0.1];
m=0:6;
y=[0.1 4 -2.2 1.6 0.1 0.1 0.2];
[Y,N]=xcorr(x,y)
subplot(311); stem(n,x);
title('Signal x(n)');
subplot(312); stem(m,y);
title('Signal y(n)');
subplot(313); stem(N,Y/max(Y));
title('Normalized Crosscorrelation of sequence x(n) and y(n)');
```

▪ Auto-Correlation:

```
clc;
clear;
n = 0:6;
x = [0.1 0.2 -0.1 4.1 -2 1.5 -0.1];
```

```
[Rxx, lags] = xcorr(x, 'biased');
subplot(211);
stem(n, x); % Plot the original signal x(n)
title('Signal x(n)');
subplot(212); stem(lags, Rxx, 'm');
title('Auto-Correlation of sequence x(n)');
```

Result:

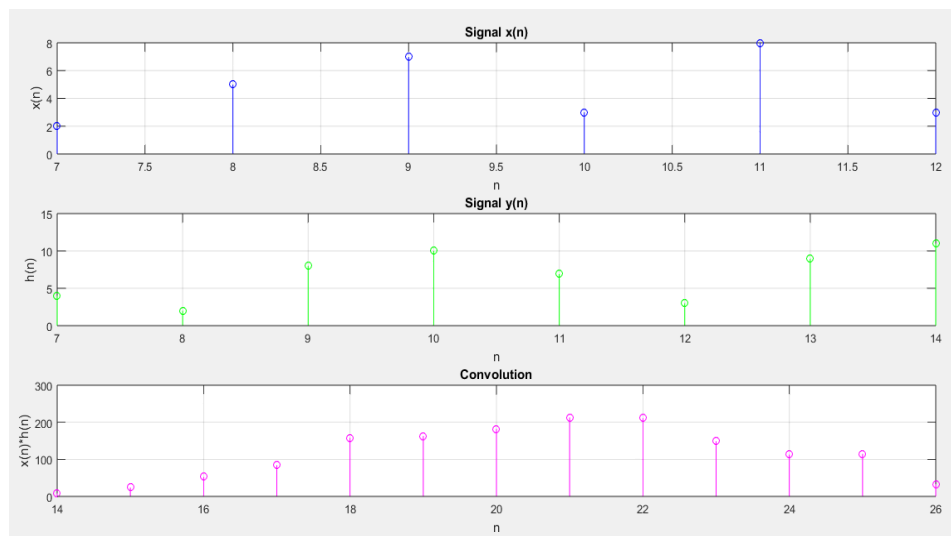
▪ Convolution:

Command Window:

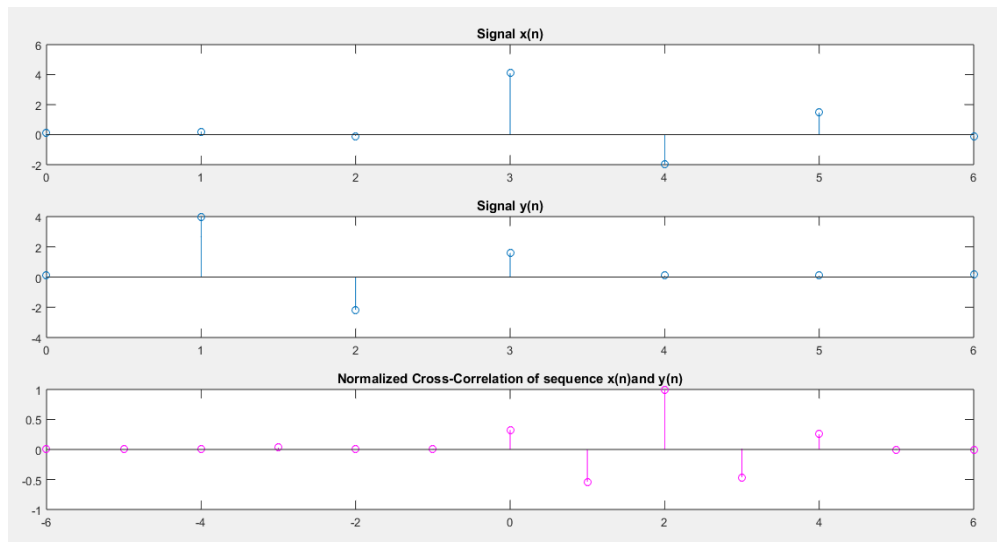
```
Command Window

Input Sequence x(n): [ 2 5 7 3 8 3 ]
Input Starting Point of x(n):7
Input Sequence h(n):[ 4 2 8 10 7 3 9 11]
Input Starting Point h(n):7
convolution of sequence x(n) & h(n):
      8      24      54      86      158      163      182      213      213      149      114      115      33
```

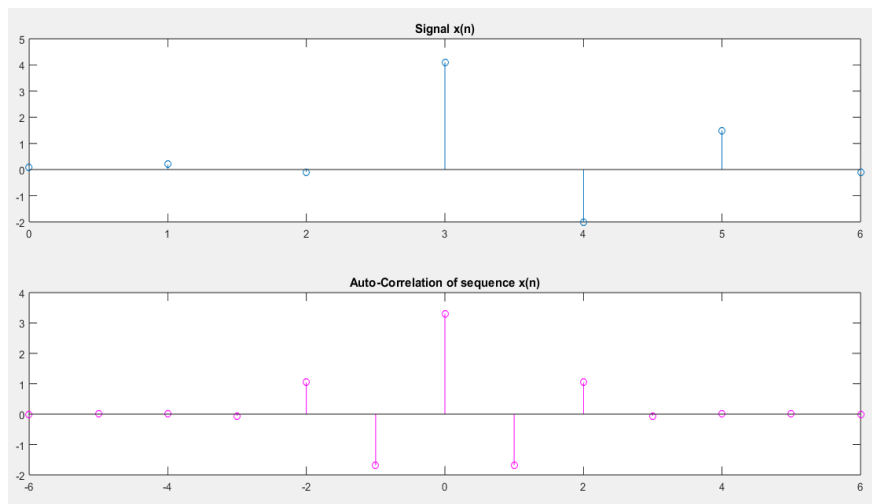
Figure:



▪ Cross-Correlation:



■ Auto-Correlation:



Conclusion:

Creating MATLAB programs for the convolution and correlation of signals is essential for signal processing. Convolution reveals the output response of systems to signals, while correlation assesses signal similarities, crucial in applications like active sonar. MATLAB's conv and xcorr functions facilitate these computations, aiding in tasks such as target detection in sonar. These operations underscore the practicality of signal processing in technological advancements and automation in MATLAB enhances this further.

Experiment No.: 06

Experiment Name: Z-Transform & Inverse Z-Transform using MATLAB.

Objectives:

- To realize the significance of z-transform.
- To determine the z-transform and inverse z-transform of discrete time signal and systems in MATLAB.
- To find the pole-zero plot and impulse response of DT system.
- To implement moving average filter by z-transform.

Required Apparatus:

- MATLAB Software.

Theory:

- **Z -Transform:** The Z-transform is a mathematical technique used in signal processing and control theory to analyze discrete-time signals. It transforms a discrete-time signal, which is a sequence of real or complex numbers, into a complex frequency-domain representation, known as the Z-domain or Z-plane.

Equation:

$$X(z) = \sum_{n=-\infty}^{\infty} x[n] \cdot z^{-n}$$

- **Inverse Z -Transform:** The inverse Z-transform is a mathematical process used to determine the original discrete-time signal from its Z-transform representation. It essentially reverses the Z-transform operation, converting a function in the Z-domain back into its time-domain sequence.

Equation:

$$x[n] = Z^{-1}\{X(z)\}$$

Code:

- **Z – Transform:**

```
clc
clear
syms z
x=input('Input Sequence:');
n= input('Input Starting Point:');
n1= length(x) + n - 1;
m=1;
result=0;
for i= n:n1
result= result + x(m)*z^(-i);
m=m+1;
end
```

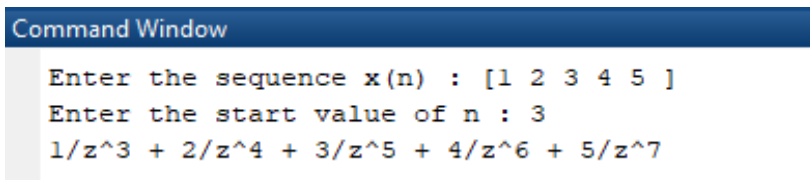
```
disp('X(Z):')
disp(result)
```

- **Inverse Z – Transform:**

```
clc
clear
syms z
Fz= input('Input Function X(Z):');
Iz=iztrans(Fz);
disp('Inverse Z-transform x(n):');
disp(Iz);
```

Result:

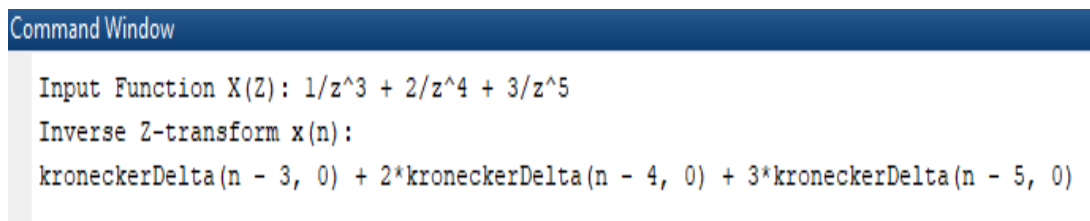
- **Z - Transform:**



Command Window

```
Enter the sequence x(n) : [1 2 3 4 5 ]
Enter the start value of n : 3
1/z^3 + 2/z^4 + 3/z^5 + 4/z^6 + 5/z^7
```

- **Inverse Z - Transform:**



Command Window

```
Input Function X(Z): 1/z^3 + 2/z^4 + 3/z^5
Inverse Z-transform x(n):
kroneckerDelta(n - 3, 0) + 2*kroneckerDelta(n - 4, 0) + 3*kroneckerDelta(n - 5, 0)
```

Conclusion:

The z-transform is essential in digital signal processing for system analysis and filter design, offering a complex frequency domain perspective of discrete-time signals. Utilized in MATLAB through `ztrans` and `iztrans`, it facilitates the examination of system dynamics and stability. The pole-zero plots and impulse response, derived from the system's transfer function, are instrumental in assessing system behavior. Additionally, the z-transform aids in implementing filters like the moving average, which smoothens data by attenuating high-frequency components.

Experiment No.: 07

Experiment Name: Frequency domain analysis of Discrete time Signal.

Objectives:

- To understand the mathematics of transforming a signal from time domain to frequency domain.
- To transform a signal from time domain to frequency domain by DFT in MATLAB.
- To transform a signal from time domain to frequency domain by FFT in MATLAB.

Required Apparatus:

- MATLAB Software.

Theory:

- **DFT:** The Discrete Fourier Transform (DFT) is a fundamental tool in Digital Signal Processing (DSP). It is used to analyze the frequency content of discrete signals and to perform operations such as filtering and spectrum analysis. The DFT converts a finite sequence of equally-spaced samples of a function into a same-length sequence of equally-spaced samples of the discrete-time Fourier transform (DTFT), which is a frequency-domain representation.

Equation:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-\frac{j2\pi}{N}kn}$$

Where;

$X[k]$ = value of the DFT at index - k ,
 $x[n]$ = n^{th} sample of the input signal,
 N = total number of samples,
 e = base of the natural logarithm,
 j = imaginary unit.

- **FFT:** The Fast Fourier Transform (FFT) is an algorithm that computes the Discrete Fourier Transform (DFT) of a sequence, or its inverse, much more rapidly than possible by direct calculation. Essentially, it transforms a signal from its original domain (often time or space) into a representation in the frequency domain and vice versa. The FFT is widely used in engineering, music, science, and mathematics for its efficiency in analyzing the frequency components of signals.

Code:

- **DFT:**
clc;
clear
T=0.005;


```

t=0:T:1; Fs=1/T;
x=sin(2*pi*50*t);
N=length(x);
E=exp(-1i*2*pi/N);
TM=zeros(N,N);
for k= 1:N
for n=1:N
TM(n,k)=E^((n-1)*(k-1));
end
end
dft=x*TM;
subplot(211); plot(t,x); title ('Time Domain Signal');
xlabel('Time');
ylabel('Amplitude');
subplot(212); plot((0:N-1)*(Fs/N),abs(dft), 'g'); title ('Frequency Domain Signal');
xlabel('Frequency');
ylabel('Amplitude');

```

■ FFT:

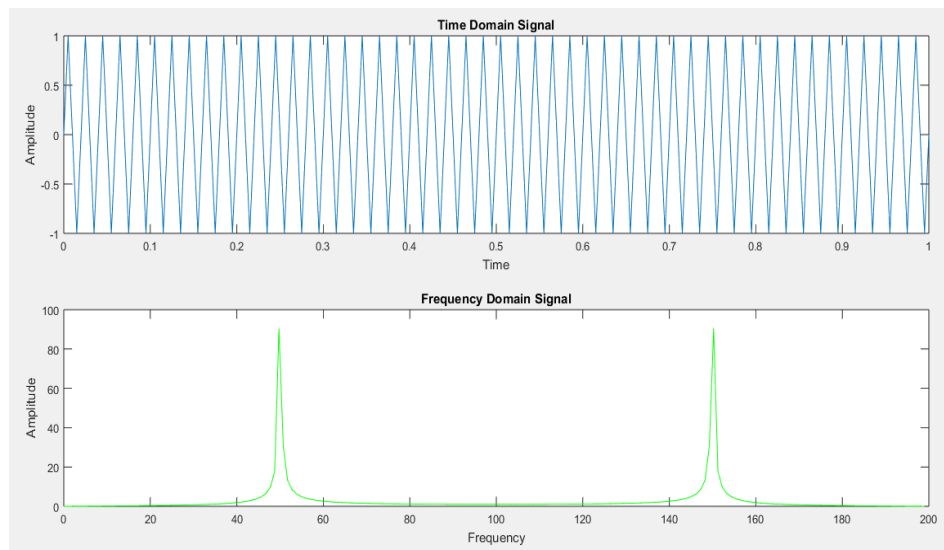
```

clc
clear
Fs=1000; % sampling frequency
Ts=1/Fs; % sampling period or time step
n=0:Ts:1-Ts; %signal duration
f1=9; f2=39; f3=25; % Frequency of the three Sine Signals
y1=10*sin(2*pi*f1*n); y2=10*sin(2*pi*f2*n);
y3=10*sin(2*pi*f3*n); y=y1+y2+y3;
ny=length(y); N=2^nextpow2(ny);
yfft=fft(y,N); yfft2=yfft(1:N/2);
xfft=Fs*(0:N/2-1)/N;
%Plots
subplot(3,3,1); plot(n,y1,'b'); title('Sinwave 1');
subplot(3,3,2); plot(n,y2); title('Sinwave2');
subplot(3,3,3); plot(n,y3); title('Sinwave3');
subplot(3,3,[4 5 6]); plot(n,y, 'r');
title('Time domain signal (sum of all sin waves)');
subplot(3,3,[7 8 9]); plot(xfft,abs(yfft2)/max(abs(yfft2)),'k');
title('Frequency domain Signal');
grid on;

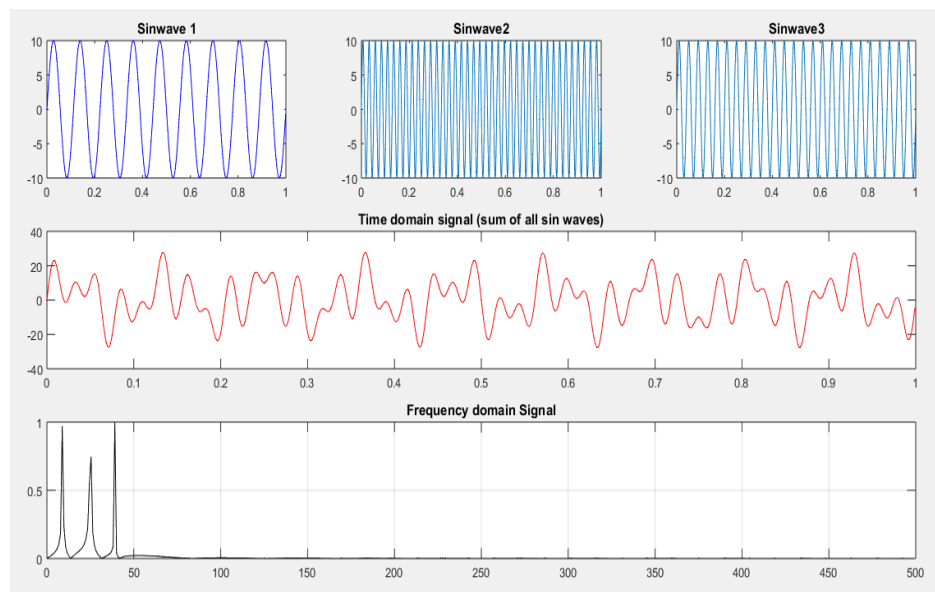
```

Result:

■ DFT:



■ FFT:



Conclusion:

The Discrete Fourier Transform (DFT) and the Fast Fourier Transform (FFT) are essential in digital signal processing for transforming signals from the time domain to the frequency domain. Using MATLAB for DFT and FFT allows for practical application and exploration of these concepts. DFT helps analyze discrete signal frequency components, crucial for audio and image processing, while FFT reduces computational complexity, enabling real-time processing of large datasets.

Experiment No.: 08

Experiment Name: Design of Finite Impulse Response (FIR) Filter

Objectives:

- To understand the impact of window function in FIR filter design.
- To understand the conversion of high pass filter kernel from low pass filter kernel.
- To design of band pass and band stop filter.
- To understand the use of FIR filter in signal separation.

Required Apparatus:

- MATLAB Software.

Theory

Finite Impulse Response (FIR) filters are a type of digital filter characterized by a finite-duration impulse response. They are widely used due to their inherent stability and linear phase response.

1. Lowpass Filter: Passes frequencies below the cutoff frequency and attenuates higher frequencies.
2. High-pass Filter: Passes frequencies above the cutoff frequency and attenuates lower frequencies.
 - Achieved using spectral inversion of the lowpass filter.
3. Bandpass Filter: Passes frequencies between two cutoff frequencies.
 - Achieved by convolving high-pass and lowpass filters.
4. Band-stop Filter: Stops frequencies between two cutoff frequencies while passing the rest.
 - Achieved by summing high-pass and lowpass filters.

A Blackman window was used to smooth the frequency response, reducing side lobes in the frequency domain.

Algorithm:

The process of designing each filter is described below:

1. Lowpass Filter

1. Define the filter order M and normalized cutoff frequency ncf .
2. Compute the ideal impulse response using the sinc function.
3. Apply the Blackman window to the ideal response.

2. High-pass Filter

1. Call the lowpass filter function with the same parameters.
2. Apply spectral inversion:
 - Invert all coefficients of the lowpass filter.
 - Add 1 to the center coefficient if the filter length is odd.

3. Bandpass Filter

1. Generate a high-pass filter with a lower cutoff frequency $ncf1$.
2. Generate a lowpass filter with an upper cutoff frequency $ncf2$.
3. Convolve the high-pass and lowpass filters.
4. Truncate or adjust the length of the resulting filter.

4. Band-stop Filter

1. Generate a high-pass filter with an upper cutoff frequency $ncf2$.
2. Generate a lowpass filter with a lower cutoff frequency $ncf1$.
3. Ensure both filters have the same length by truncating if necessary.
4. Sum the high-pass and lowpass filters.

Code:

```
clc
clear
M=input('Define the length of filter kernel:');
n=0:1:M-1;
wc=0.5*pi;
hd=sin(wc*(n-(M-1)/2))./(pi*(n-(M-1)/2));
if(rem(M,2)~=0)
    hd(((M-1)/2)+1)=wc/pi;
end
w_blackman=0.42-0.5*cos((2*pi*n)/(M-1))+0.08*cos((4*pi*n)/(M-1));
w_hamming=0.54-(0.46*cos((2*pi*n)/(M-1)));
w_hanning=hanning(M)';
w_bartlett=bartlett(M)';
h_blackman=hd.*w_blackman;
h_hamming=hd.*w_hamming;
h_bartlett= hd.*w_bartlett;
h_hanning= hd.*w_hanning;
plot(n,w_blackman,n,w_hamming,n,w_hanning,n,w_bartlett,'Linewidth',1.25);
title('Shapes of different window');
legend('Blackman Window','HammingWindow','HanningWindow','Bartlett Window');
fvtool(h_blackman,1,h_hamming,1,h_bartlett,1,h_hanning,1);
legend('Blackman Window','HammingWindow','Bartlett Window','HanningWindow');
```

Lowpass Filter:

```

function h = fir_lowpass(ncf, M)

n = 0:1:M-1;           % Time index
wc = 2*pi*ncf;          % Normalized cutoff angular frequency
hd = sin(wc*(n-(M-1)./2))./(pi*(n-(M-1)./2)); % Ideal impulse response
if(rem(M,2)~=0)
    hd((M-1)/2+1)=wc/pi; % Handle division by zero for the center tap
end
% Blackman window
w_blackman = 0.42-0.5*cos((2*pi*n)./(M-1))+0.08*cos((4*pi*n)./(M-1));
h = hd.*w_blackman;    % Apply the window
end

```

Highpass Filter:

```

function h=fir_highpass(ncf,M)

if (rem(M,2)==0)
    M=M+1;
end
% Generate the lowpass filter
h_low = fir_lowpass(ncf, M);

% Apply spectral inversion
h = -h_low; % Invert all coefficients
h((M-1)/2 + 1) = h((M-1)/2 + 1) + 1; % Add 1 to the center tap
end

```

Bandpass Filter:

```

function h = fir_bandpass(ncf1, ncf2, M)

% Generate the highpass and lowpass filters
h_high = fir_highpass(ncf1, M);
h_low = fir_lowpass(ncf2, M);

% Convolve the filters to create a bandpass filter
h = conv(h_high, h_low);
end

```

Bandstop Filter:

```

function h = fir_bandstop(ncf1, ncf2, M)

% Generate the highpass and lowpass filters
h_high = fir_highpass(ncf2, M);
h_low = fir_lowpass(ncf1, M);
% Ensure both filters have the same length
len_low = length(h_low);
len_high = length(h_high);

if len_low > len_high
    h_low = h_low(1:len_high); % Truncate lowpass filter
elseif len_high > len_low
    h_high = h_high(1:len_low); % Truncate highpass filter
end
% Add the filters to create a bandstop filter

```

```

    h = h_high + h_low;
end

```

Comparison of Different types of Filters:

```

clc
clear

%Generation of Raw Signal
Fs=500; % sampling frequency
Ts=1/Fs; % sampling period or time step
t=0:Ts:1-Ts; %signal duration
f1=20; f2=70; f3=90;
y1=10*sin(2*pi*f1*t); y2=10*sin(2*pi*f2*t); y3=10*sin(2*pi*f3*t);
y=y1+y2+y3;
subplot(321); plot(t,y,'r'); title('Noisy Signal');

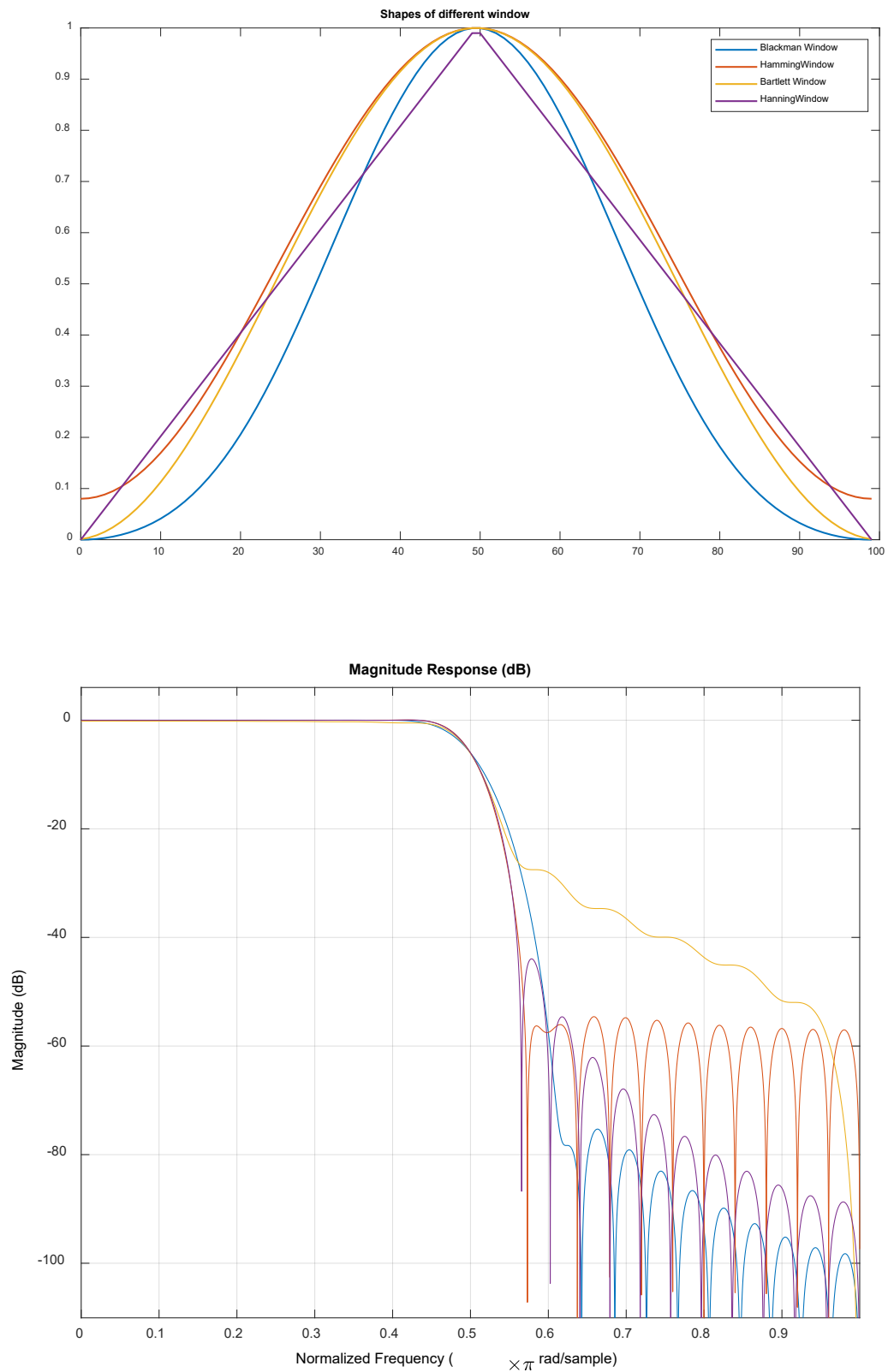
%Frequency domain representation of raw signal
N_rs=length(y);
N_rs=2^nextpow2(N_rs);
fc_rs=fft(y,N_rs);
fc_rs=fc_rs(1:N_rs/2);
f_rs=Fs*(0:N_rs/2-1)/N_rs;
subplot(322); plot(f_rs,abs(fc_rs),'r');
title('Noisy signal in frequency domain');
%Filter Design
cut_off1=65/Fs; cut_off2=75/Fs; order=256;

%h=fir_lowpass(cut_off1,order);
%h=fir_highpass(cut_off2,order);
%h=fir_bandpass(cut_off1,cut_off2,order);
h=fir_bandstop(cut_off1,cut_off2,order);

%Plot
subplot(323); stem(h);
title('Filter impulse response');
subplot(324);
[f_h,w]=freqz(h);
plot((w/(2*pi))*Fs,abs(f_h),'linewidth',1.2);
title('Filter frequency response');
%Signal Filtering
filtered_sig=conv(y,h);
subplot(325); plot(filtered_sig,'g');
title('Filtered signal');
%Frequency domain representation of filtered signal
N_fs=length(filtered_sig); N_fs=2^nextpow2(N_fs);
fc_fs=fft(filtered_sig,N_fs); fc_fs=fc_fs(1:N_fs/2);
f_fs=Fs*(0:N_fs/2-1)/N_fs;
subplot(326); plot(f_fs,abs(fc_fs),'g');
title('Filtered signal in frequency domain');

```

Figure:



Results

The frequency responses for each filter were analyzed and observed as per expectations:

- Lowpass Filter: Frequencies below the cutoff frequency were passed.
- High-pass Filter: Frequencies above the cutoff frequency were passed.
- Bandpass Filter: Frequencies between the two cutoff frequencies were passed.
- Band-stop Filter: Frequencies between the two cutoff frequencies were attenuated.

Conclusion:

1. The FIR filters for lowpass, high-pass, bandpass, and band-stop were successfully implemented using MATLAB.
2. Spectral inversion and convolution techniques were effective for high-pass and bandpass filters.
3. The Blackman window provided a smooth transition and reduced side lobes in the frequency response.