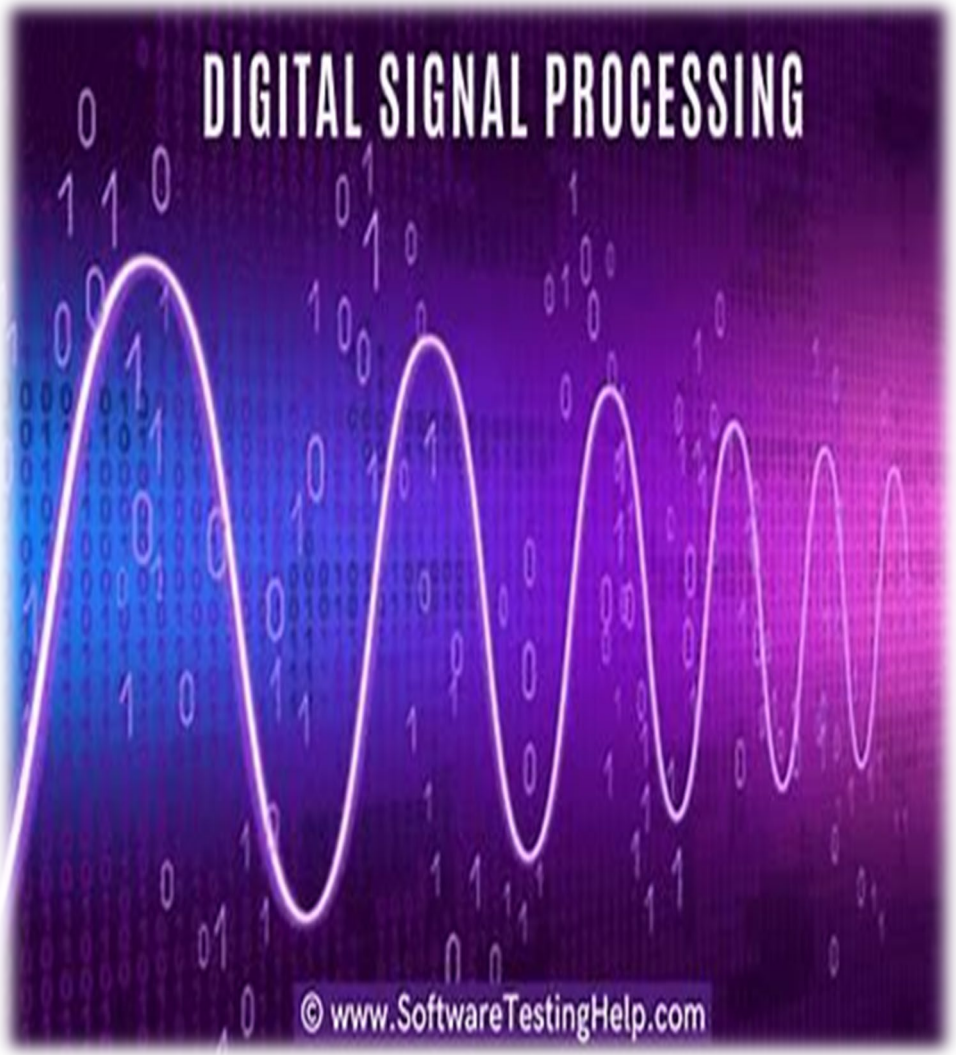
University of Global Village (UGV), Barishal
Dept. of Electrical and Electronic Engineering (EEE)

Digital Signal Processing Sessional
EEE 0714-3108

Noor Md Shahriar
Senior Lecturer, Deputy Head of Dept.
Dept. of Electrical & Electronic Engineering
University of Global Village, (UGV), Barishal
Contact: 01743-500587
E-mail: noor.shahriar1@gmail.com

'Imagination is more important than knowledge' – Albert Einstein

Basic Course Information



Course Title	Digital Signal Processing Sessional
Course Code	EEE- 0714-3108
Credits	01
Marks	50
Course Type	Sessional
Level	5th Semester
Academic Session	Winter 2025

ASSESSMENT PATTERN

CIE- Continuous Internal Evaluation (20 Marks)

Bloom's Category Marks	Tests (20)	Quiz (10)	External Participation in Curricular/Co-Curricular Activities (20)
Imitation	06	09	Bloom's Affective Domain: (Attitude or will) Attendance: 10 Viva-Voca: 5 Report Submission: 5
Manipulation	04	06	
Precision	06		
Articulation	02		
Naturalization	02		

SEE- Semester End Examination (30 Marks)

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

COURSE LEARNING OUTCOME (CLO)

Course learning outcomes (CLO): After successful completion of the course students will be able to -

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.

SYNOPSIS / 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 Objective

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 OUTLINE

Sl.	Content of Course	Hrs	CLOs
1	Introduction to MATLAB: Basics of MATLAB environment and initial programming setup	2	CLO1, CLO2
2	Continuous-Time and Discrete-Time Representation of Signals: Visualization and basic operations	2	CLO2, CLO3
3	Analog to Digital Conversion: Sampling, quantization, and coding	2	CLO2, CLO3
4	Manipulation of Discrete-Time (DT) Signals: Shifting, scaling, and inversion	2	CLO2, CLO3

COURSE OUTLINE

Sl.	Content of Course	Hrs	CLOs
5	Convolution and Correlation of Discrete-Time Sequences: Understanding DSP fundamentals	5	CLO3, CLO4
6	z-Transform in MATLAB: Analysis and computation of z-domain representation	6	CLO4, CLO5
7	Frequency Domain Analysis of DT Signals: Fourier analysis and spectral representation	7	CLO4, CLO5
8	Design of Finite Impulse Response (FIR) Filter: Techniques and implementation in MATLAB	8	CLO4, CLO5

COURSE SCHEDULE

Week	Topic	Teaching Learning Strategy	Assessment Strategy	Corresponding CLOs
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

COURSE SCHEDULE

Week	Topic	Teaching Learning Strategy	Assessment Strategy	Corresponding CLOs
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

COURSE SCHEDULE

Week	Topic	Teaching Learning Strategy	Assessment Strategy	Corresponding CLOs
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

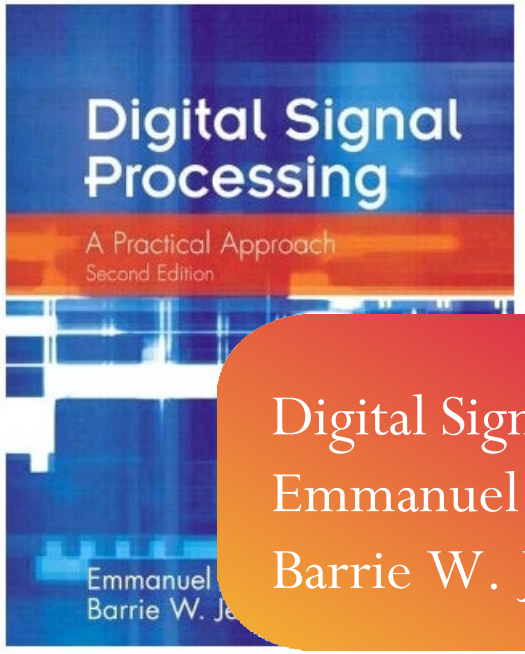
COURSE SCHEDULE

Week	Topic	Teaching Learning Strategy	Assessment Strategy	Corresponding CLOs
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

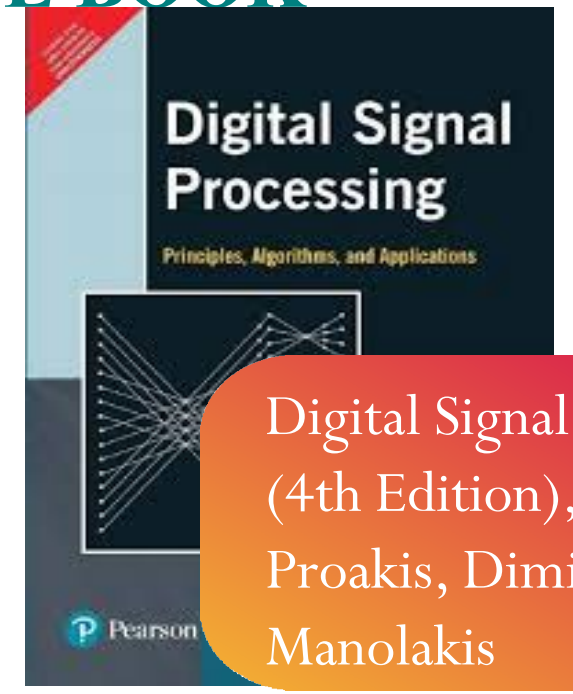
References

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/**

REFERENCE BOOK



Digital Signal Processing - Emmanuel C. Ifeachor, Barrie W. Jervis



Digital Signal Processing (4th Edition), John G. Proakis, Dimitris K Manolakis

YouTube

Video Lecture Playlist
https://youtube.com/playlist?list=PLhdVEDm7SZObfQ1O1RzBGhqh1NxVDoCW&si=zdIIK56sTaLlO4gW

University of Global Village (UGV), Barishal
Dept. of Electrical and Electronic Engineering

Experiment-1
Introduction to MATLAB

Course Code: EEE-307/CSE-309
Course Title: Digital Signal Processing Sessional

Prepared By
Noor Md Shahriar
Senior Lecturer, Dept. of EEE, UGV

Objectives:

a) To familiarize with MATLAB and some basic commands of MATLAB.

b) To know about variable and variable types in MATLAB.

c) To know about Matrix manipulation in MATLAB.

d) To learn about plotting 2D graphs in MATLAB.

e) To become familiar with MATLAB script/editor.

f) To become familiar with conditional operators and loops in MATLB.

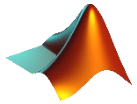
g) To learn about debugging program in MATLAB.

h) To know how to develop an user defined function in MATLAB

What is MATLAB?

- MATLAB Stands for MATrix LABoratory.
- MATLAB is a programming and numeric computing environment used by millions of engineers and scientists to analyze data, develop algorithms, and create models.
- It has many in built functions, toolboxes (signal and image processing, control systems, wireless communications, computational finance, robotics, deep learning and AI etc) and apps.

History of MATLAB



- Invented by **Prof. Cleve Moler** (American mathematician and computer programmer specializing in numerical analysis.) to make programming easy for his students.
  - >> Late 1970.
  - >> University of New Mexico.
- The MathWorks, Inc. was formed in 1984
  - >> By Moler and Jack Little.
  - >> One Product: MATLAB.
- Today
  - >> 100 products
  - >> As of 2020, MATLAB has more than 4 million users worldwide.
  - >> Taught in 5,000 universities (2015).
- Matlab Version (Release history)
  - >> 1st version: MATLAB 1.0 in 1984.
  - >> Latest Version: MATLAB R2024a (2024).

Some useful MATLAB Commands

```
>> version          % this will tell you the running MATLAB version
ans = 9.0.0.341360 (R2015a)
>> help             % lists available packages/toolboxes on system.
>> help elfun       % lists functions in elementary functions package
>> help sin         % instructions on the sine function
>> lookfor sine     %  if you don't know the function name …
>> doc       % start matlab help documentation
>> doc sin          % for full details of function
>> Ctrl+C     (Press 'Ctrl+C' to stop execution of instruction)
>> quit             % to quit MATLAB
```

Some useful MATLAB Commands (Cont.)

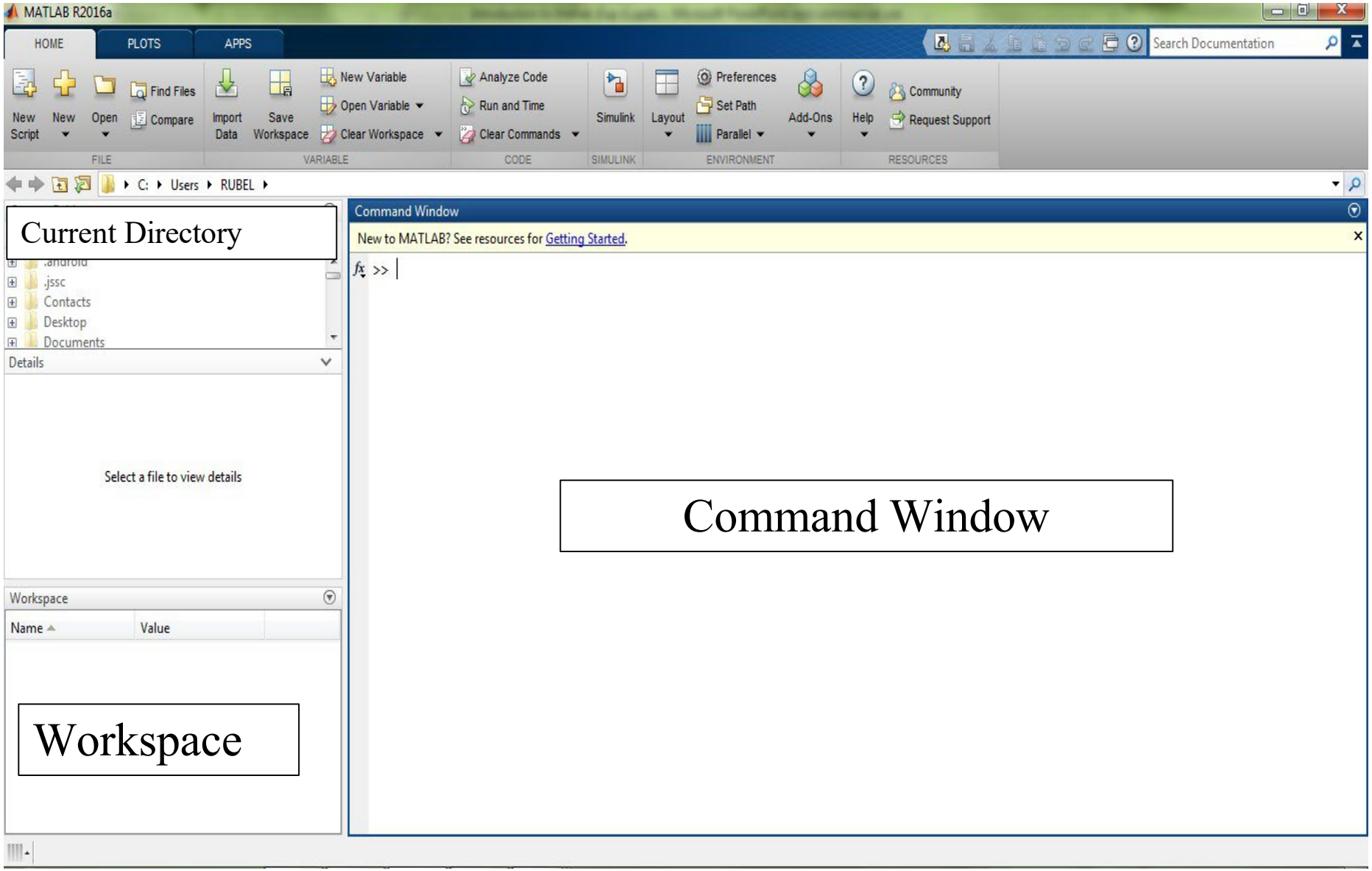
```
>> format loose      % line space increased in command window
>> format compact    % line space decreased in command window
>> format long       % displays more digits after decimal points
>> format short      % displays less digits after decimal points
```

Note: 'format long/short' has no effect on accuracy during the calculation. The commands just show larger or smaller no of digits after decimal point in the display.

```
>> exist('name of variable/function')    % Check if variables or functions are
```

defined.

MATLAB Preview



Variables

- Don't have to declare type, is case sensitive
- variable begins with a letter, e.g., A2z or a2z
- can be a mix of letters, digits, and underscores (e.g., vector_A)
- Variable name can be up to 63 characters
- Don't even have to initialise
- Just assign in command window

```
>>

>> a=12; % variable a is assigned 12
```

Try the same line without the semicolon and comments

Matlab prompt

assign operator

suppress command output

comment operator

Size of Variables

- All numerical variables in MATLAB are matrices, a mathematical data type corresponding to a two-dimensional array of numbers.

```
>> m=3;
>> size(m)
ans =
     1     1
>> a=[1,2,3];
>> size(a)
  ans =
     1     3
```

Remember these terms

Scalar: Single element variable like 1,5,42 etc.

Vector: If you group (row or column wise) a number of scalars together you end up with a vector.

Example: a=[1, 2, 3];
b=[6 ,7 ,8];

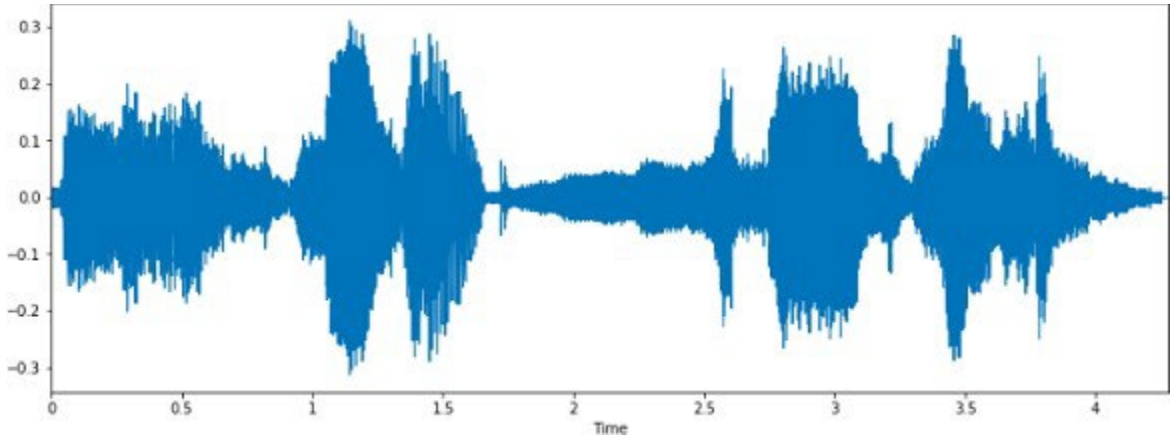
Matrix: A list of equal sized vector.

Example: A= [a;b]

Tensor (Array): Three or more dimensional matrix

Example: Color Picture.

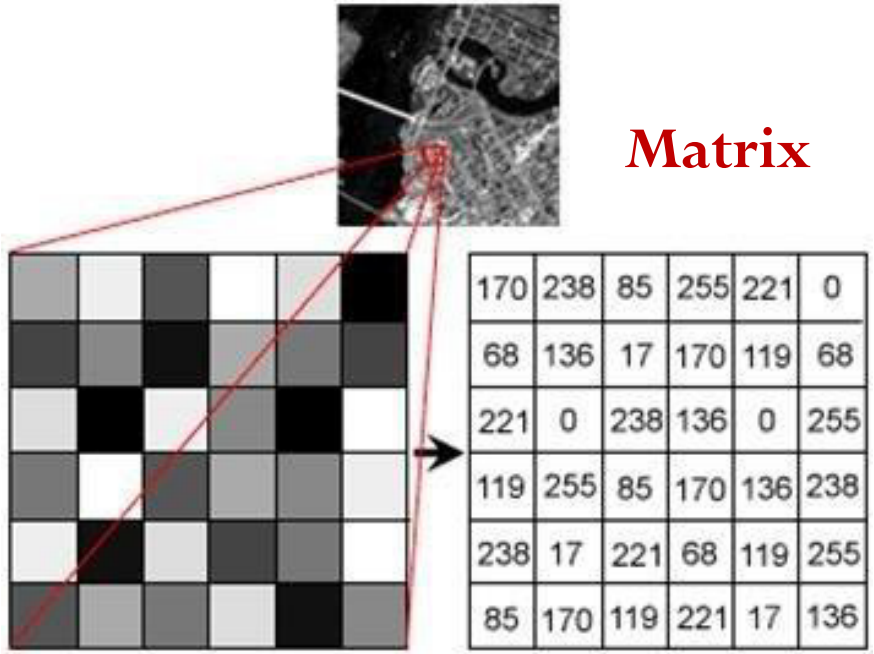
Audio Signal



Vector

{a1,a1,a3,a4,……}

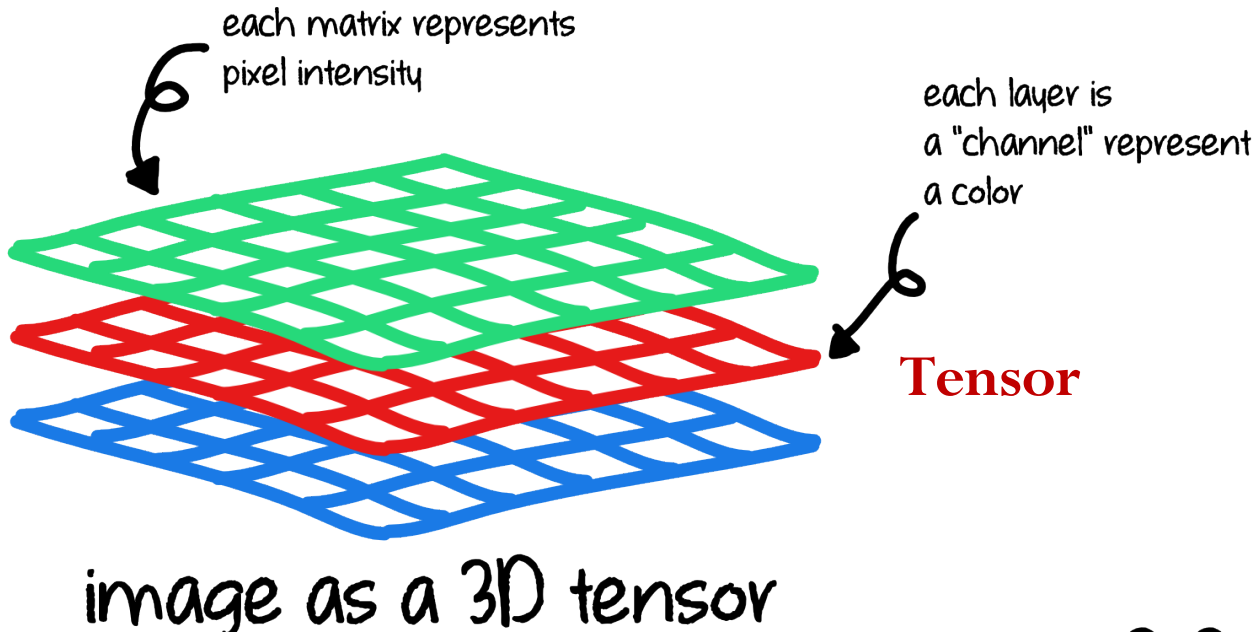
Gray Scale Image



170	238	85	255	221	0
68	136	17	170	119	68
221	0	238	136	0	255
119	255	85	170	136	238
238	17	221	68	119	255
85	170	119	221	17	136

Matrix

Color Image



Workspace

- The workspace is Matlab's memory.
- Displaying contents of workspace.

```
>> a=12;
>> b=10;
>> c=a+b;
>> whos
  Name      Size            Bytes  Class     Attributes
    a.      1x1                 8  double
    b.      1x1                 8  double
    c.      1x1                 8  double
```

- Delete variable(s) from workspace

```
>> clear a b; % delete a and b from workspace
>> whos
>> clear all; % delete all variables from workspace
>> whos
>> clc   % clear command window (workspace remain unchange)
```

Workspace (Cont.)

>> save % save workspace variable to current directory. (before closing data one can save data for using in the next session)

>> load % reload data.

>> save my_file a b % create a new file named ‘my_file’in current directory and save variable a and b in that file.

>> load my_file % load data from my_file to workspace.

Numeric Variable Types

- Floating Point Numbers (Double Precision and Single precision)
- Integer Numbers (signed and unsigned integer of 8,16,32,64-bits)

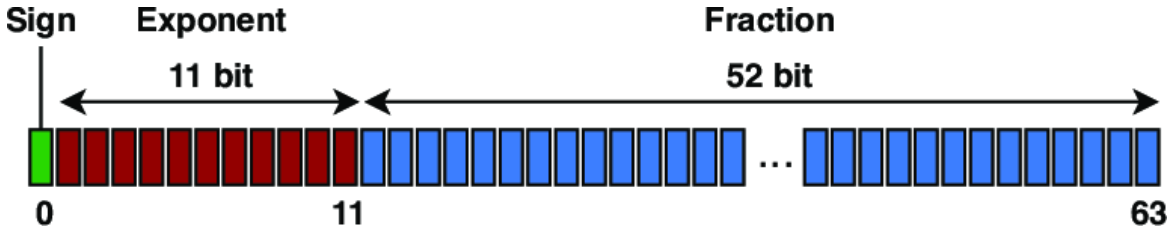
Floating-Point Numbers

MATLAB® represents floating-point numbers in either double-precision or single-precision format. The default is double precision.

Double-Precision Floating Point

- ✓ MATLAB constructs the double-precision (or double) data type according to **IEEE® Standard 754** for double precision.
- ✓ Any value stored as a double requires 64 bits.

Bits	Usage
63	Sign (0 = positive, 1 = negative)
62 to 52	Exponent, biased by 1023
51 to 0	Fraction f of the number 1.f



$$Value = (-1)^S \times 2^{E-bias} \times (1.f)_2$$

realmax or realmax('double')
realmin or realmin('double')

$$Value = (-1)^S \times 2^{E-1023} \times (1.f_{51}f_{50} \dots f_0)_2$$

Double-precision examples [edit]

0 01111111111 0000000000000000000000000000000000000000000000000000$_2$ ≙ 3FF0 0000 0000 0000$_{16}$ ≙ $+2^0 \times 1 = 1$

0 01111111111 0000000000000000000000000000000000000000000000000001$_2$ ≙ 3FF0 0000 0000 0001$_{16}$ ≙ $+2^0 \times (1 + 2^{-52}) \approx 1.0000000000000002$, the smallest number > 1

0 01111111111 0000000000000000000000000000000000000000000000000010$_2$ ≙ 3FF0 0000 0000 0002$_{16}$ ≙ $+2^0 \times (1 + 2^{-51}) \approx 1.0000000000000004$

0 10000000000 0000000000000000000000000000000000000000000000000000$_2$ ≙ 4000 0000 0000 0000$_{16}$ ≙ $+2^1 \times 1 = 2$

1 10000000000 0000000000000000000000000000000000000000000000000000$_2$ ≙ C000 0000 0000 0000$_{16}$ ≙ $-2^1 \times 1 = -2$

0 10000000000 1000000000000000000000000000000000000000000000000000$_2$ ≙ 4008 0000 0000 0000$_{16}$ ≙ $+2^1 \times 1.1_2 = 11_2 = 3$

0 10000000001 0000000000000000000000000000000000000000000000000000$_2$ ≙ 4010 0000 0000 0000$_{16}$ ≙ $+2^2 \times 1 = 100_2 = 4$

0 10000000001 0100000000000000000000000000000000000000000000000000$_2$ ≙ 4014 0000 0000 0000$_{16}$ ≙ $+2^2 \times 1.01_2 = 101_2 = 5$

0 10000000001 1000000000000000000000000000000000000000000000000000$_2$ ≙ 4018 0000 0000 0000$_{16}$ ≙ $+2^2 \times 1.1_2 = 110_2 = 6$

0 10000000011 0111000000000000000000000000000000000000000000000000$_2$ ≙ 4037 0000 0000 0000$_{16}$ ≙ $+2^4 \times 1.0111_2 = 10111_2 = 23$

0 01111111000 1000000000000000000000000000000000000000000000000000$_2$ ≙ 3F88 0000 0000 0000$_{16}$ ≙ $+2^{-7} \times 1.1_2 = 0.00000011_2 = 0.01171875$ (3/256)

$$Value = (-1)^S \times 2^{E-1023} \times (1.f_{51}f_{50} \dots f_0)_2$$

0 00000000000 0000000000000000000000000000000000000000000000000001$_2$ ≙ 0000 0000 0000 0001$_{16}$ ≙ $+2^{-1022} \times 2^{-52} = 2^{-1074} \approx 4.9406564584124654 \times 10^{-324}$ (Min. subnormal positive double)

0 00000000000 1111111111111111111111111111111111111111111111111111$_2$ ≙ 000F FFFF FFFF FFFF$_{16}$ ≙ $+2^{-1022} \times (1 - 2^{-52}) \approx 2.2250738585072009 \times 10^{-308}$ (Max. subnormal double)

0 00000000001 0000000000000000000000000000000000000000000000000000$_2$ ≙ 0010 0000 0000 0000$_{16}$ ≙ $+2^{-1022} \times 1 \approx 2.2250738585072014 \times 10^{-308}$ (Min. normal positive double)

0 11111111110 1111111111111111111111111111111111111111111111111111$_2$ ≙ 7FEF FFFF FFFF FFFF$_{16}$ ≙ $+2^{1023} \times (1 + (1 - 2^{-52})) \approx 1.7976931348623157 \times 10^{308}$ (Max. Double)

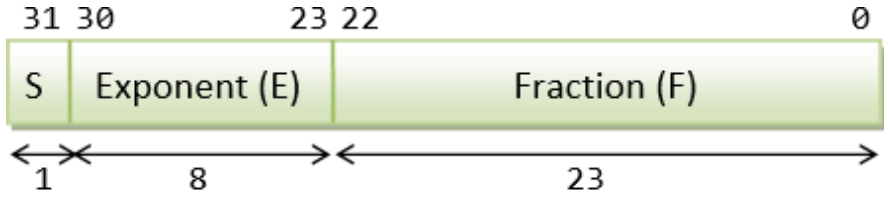
0 00000000000 0000000000000000000000000000000000000000000000000000$_2$ ≙ 0000 0000 0000 0000$_{16}$ ≙ +0

Numeric Variable Types (Cont.)

Single-Precision Floating Point

- ✓ MATLAB constructs the single-precision (or single) data type according to IEEE Standard 754 for single precision.
- ✓ Any value stored as a single requires 32 bits.

Bits	Usage
31	Sign (0 = positive, 1 = negative)
30 to 23	Exponent, biased by 127
22 to 0	Fraction f of the number 1.f

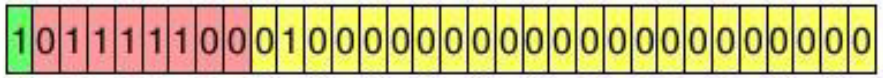


32-bit Single-Precision Floating-point Number

$$\boldsymbol{Value = (-1)^S \times 2^{E-bias} \times (1.f)_2}$$

$$\boldsymbol{Value = (-1)^S \times 2^{E-127} \times (1.f_{22}f_{21} \ldots f_0)_2}$$

- What is the decimal value of this Single Precision float?



- Solution:
  - Sign = 1 is negative
  - Exponent = $(01111100)_2 = 124$, $E - \text{bias} = 124 - 127 = -3$
  - Significand = $(1.0100 \ldots 0)_2 = 1 + 2^{-2} = 1.25$ (1. is implicit)
  - Value in decimal = $-1.25 \times 2^{-3} = -0.15625$

Numeric Variable Types (Cont.)

Integer Type Variables (Signed and Unsigned)

int8	8-bit signed integer arrays
int16	16-bit signed integer arrays
int32	32-bit signed integer arrays
int64	64-bit signed integer arrays
uint8	8-bit unsigned integer arrays
uint16	16-bit unsigned integer arrays
uint32	32-bit unsigned integer arrays
uint64	64-bit unsigned integer arrays

Related functions:

intmax('type'), Example: intmax('uint8')

intmin('type'), Example: intmin('uint8')

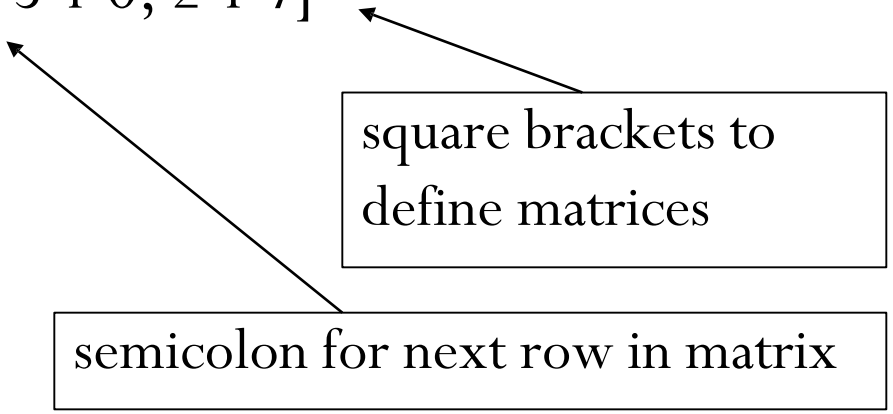
Range: $-2^7\ to\ (2^7-1) \gg -128\ to\ +127$

Range: $0\ to\ (2^8-1) \gg 0\ to\ 255$

Matrices and Access to matrix elements

- Don't need to initialise type, or dimensions

```
>>A = [3 2 1; 5 1 0; 2 1 7]
A =
     3     2     1
     5     1     0
     2     1     7
```



- Access elements of a matrix

```
>> A = [3 2 1; 5 1 0; 2 1 7]
>>A(1,2)
ans= 2
```

- Remember Matrix(row,column)
- Naming convention: Matrix variables start with a capital letter while vectors or scalar variables start with a simple letter.

The colon (:) Operator and Matrices (Accessing Parts of Matrix)

- The colon is one of the most useful operators in MATLAB. It can create vectors, subscript arrays, and specify for iterations.

Use of colon (:) operator

✓ Create Unit-Spaced Vector

i : k

```
>> 1:10
ans =
    1    2    3    4    5    6    7    8    9   10
```

✓ Create Vector with Specified Increment

i : j : k

```
>> 1:2:10
ans =
    1    3    5    7    9
```

✓ Subscript vector- A(j:k) equivalent to the vector [A(j), A(j+1), …, A(k)].

```
>> m=[2   6   3   1   8   9   0   2   4];
>> p=m(2:7)
p = 6    3    1    8    9    0
```

The colon (:) Operator and Matrices (Accessing Parts of Matrix)

✓ Index Matrix Rows and Columns

- A(:,n) is the nth column of matrix A.
- A(m,:) is the mth row of matrix A.
- A(:) reshapes all elements of A into a single column vector. This has no effect if A is already a column vector.
- A(:,j:k) includes all subscripts in the first dimension but uses the vector j:k to index in the second dimension

```
A =
     3     2     1
     5     1     0
     2     1     7
```

```
>>A(:,2)
ans =
     2
     1
     1
```

```
>>A(3,2:3)
ans =
     1     7
>> A(3,:)
ans =
     2     1     7
```

```
>> A(:)
ans =3
     5
     2
     2
     1
     1
     1
     0
     7
```

What'll happen if you type

A(:,:) ?

A(1:end, 1) ?

A(end-1, end-2) ?

sum(A(:))

Manipulating Matrices

```
>> A '          % transpose
>> B*A          % matrix multiplication
>> B.*A         % element by element multiplication (Array Multiplication)
>> B/A          % matrix division
>> B./A         % element by element division (B over A)
>> B.\A         % element by element division (B under A)
>> [B A]        % Join matrices (horizontally)
>> [B; A]       % Join matrices (vertically)
```

A =

3	2	1
5	1	0
2	1	7

B =

1	3	1
4	9	5
2	7	2

Task: Create matrices A and B and try out the matrix operators in this slide

MATLAB Graphics

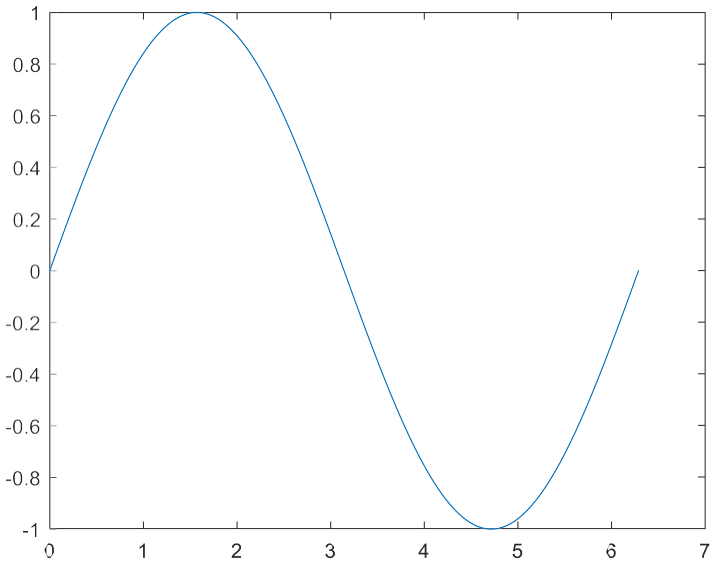
- Line plot
- Bar graph
- Surface plot
- Contour plot
- MATLAB has 2D, 3D visualization tools as well as other graphics packages.

MATLAB Graphics: 2D-Line Plot

plot(X,Y) creates a 2-D line plot of the data inY versus the corresponding values in X.

- ✓ If X andY are both vectors, then they must have equal length. The plot function plotsY versus X.
- ✓ If X andY are both matrices, then they must have equal size. The plot function plots columns ofY versus columns of X.

```
>> t = 0:pi/100:2*pi;
>> y = sin(t);
>> plot(t,y)
```



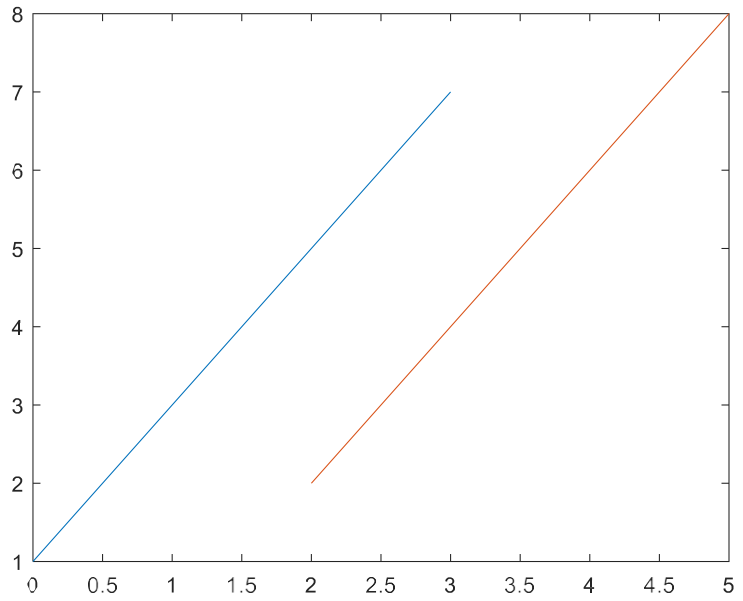
A =

1	2
3	4
5	6
7	8

B =

0	2
1	3
2	4
3	5

```
>> A=[1 2; 3 4; 5 6; 7 8]
>> B=[0 2; 1 3; 2 4; 3 5]
>> plot(B,A)
```

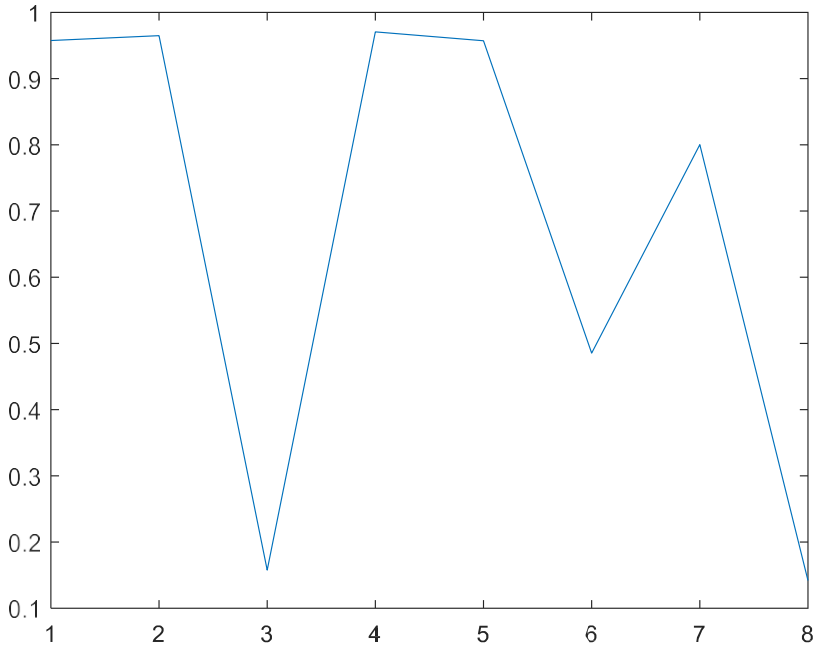


MATLAB Graphics: 2D-Line Plot (Cont.)

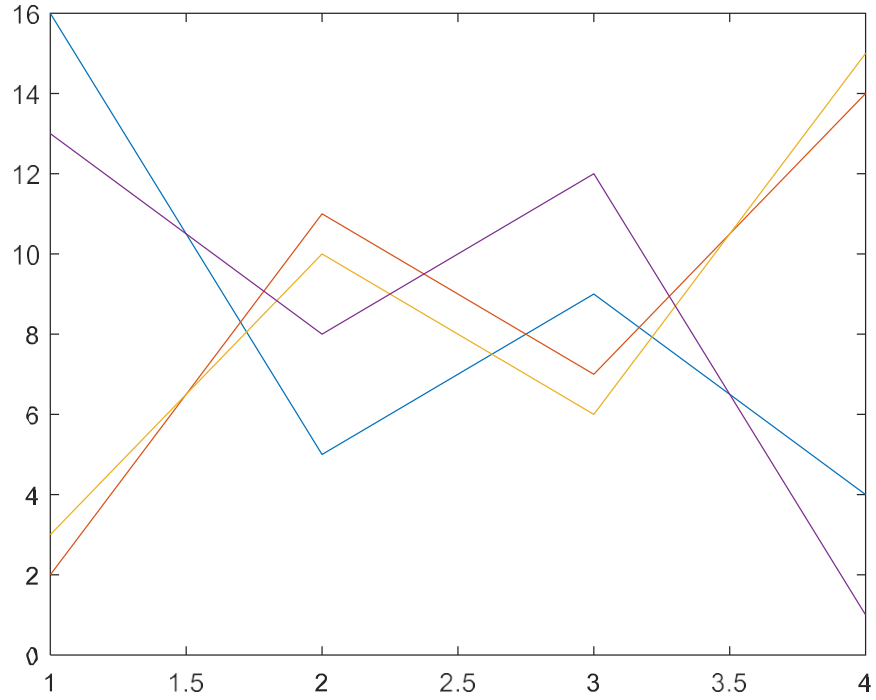
plot(Y) creates a 2-D line plot of the data inY versus the index of each value.

- ✓ IfY is a vector, then the x-axis scale ranges from 1 to length(Y).
- ✓ IfY is a matrix, then the plot function plots the columns ofY versus their row number.The x-axis scale ranges from 1 to the number of rows inY.

```
>> y=rand(1,8)
>> plot(y)
```




```
>> A=magic(4)

A =

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
>> plot(A)
```



MATLAB Graphics: Line Plot (Cont.)

```
>> xlabel('t');
>> ylabel('sin(t)');
>> title('The plot of t vs sin(t)');
>> axis([0,12,-10,20]);   % axis([XMIN XMAX YMIN YMAX]) sets scaling for the x- and y-axes
                            on the current plot.
>> grid on
>> grid MINOR
>> grid off
>> axis off
>> axis on
>> close(1)
>> close all
```

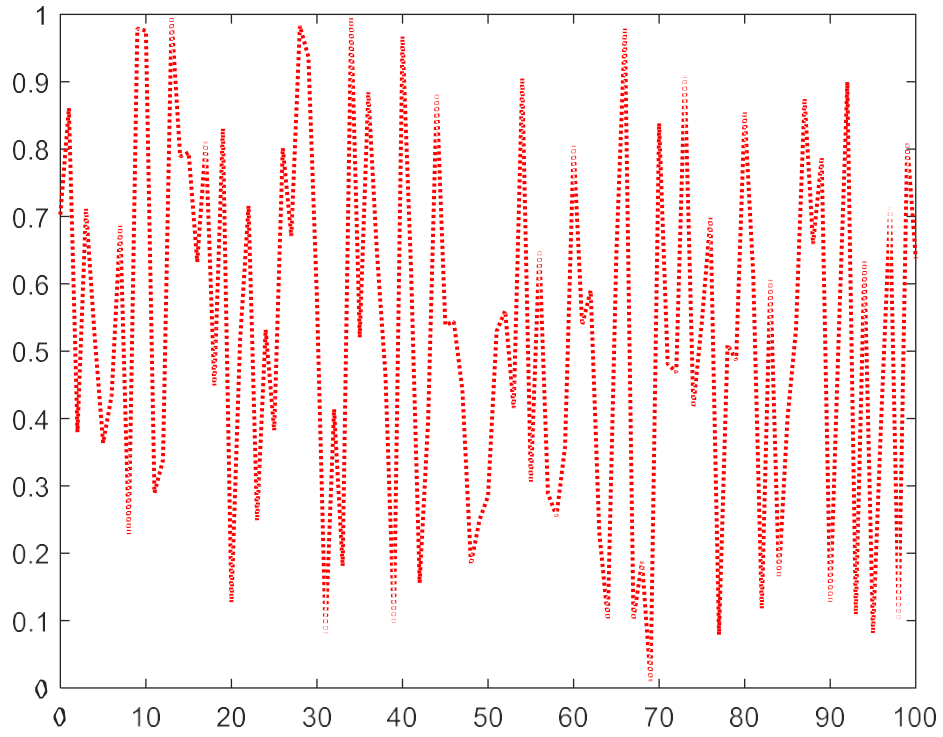
MATLAB Graphics: 2D-Line Plot (Cont.)

plot(X,Y,LineSpec)

plot(Y,LineSpec)

LineSpec: sets the line appearance and behavior (style, marker symbol, and color)

```
>> x=0:100;
>> y=rand(1,length(x));
>> plot(x,y,':r','LineWidth',1.5)
```



LineStyle — Line style

'-' (default) | '--' | ':' | '-.' | 'none'

Value	Description	Result
'-'	Solid line	————
'--'	Dashed line	- — — — —
':'	Dotted line	.....
'-.'	Dash-dotted line	—.—.—.—.-
'none'	No line	No line

LineWidth — Line width

0.5 (default) | positive value

Color — Line color

[0 0 0] (default) | RGB triplet | color string | 'none'

Long Name	Short Name	RGB Triplet
'yellow'	'y'	[1 1 0]
'magenta'	'm'	[1 0 1]
'cyan'	'c'	[0 1 1]
'red'	'r'	[1 0 0]
'green'	'g'	[0 1 0]
'blue'	'b'	[0 0 1]
'white'	'w'	[1 1 1]
'black'	'k'	[0 0 0]

MATLAB Graphics: 2D-Line Plot (Cont.)

```
>> t=0:0.01:1;

>> y=sin(2*pi*3*t);

>>plot(t,y,'--gs','LineWidth',2,'MarkerSize',6,'MarkerEdgeColor','b','MarkerFaceColor',[0.5,0.5,0.5])
```

Marker — Marker symbol

'none' (default) | 'o' | '+' | '*' | '.' |

Value	Description
'o'	Circle
'+'	Plus sign
'*'	Asterisk
'.'	Point
'x'	Cross
'square' or 's'	Square
'diamond' or 'd'	Diamond

MarkerSize — Marker size

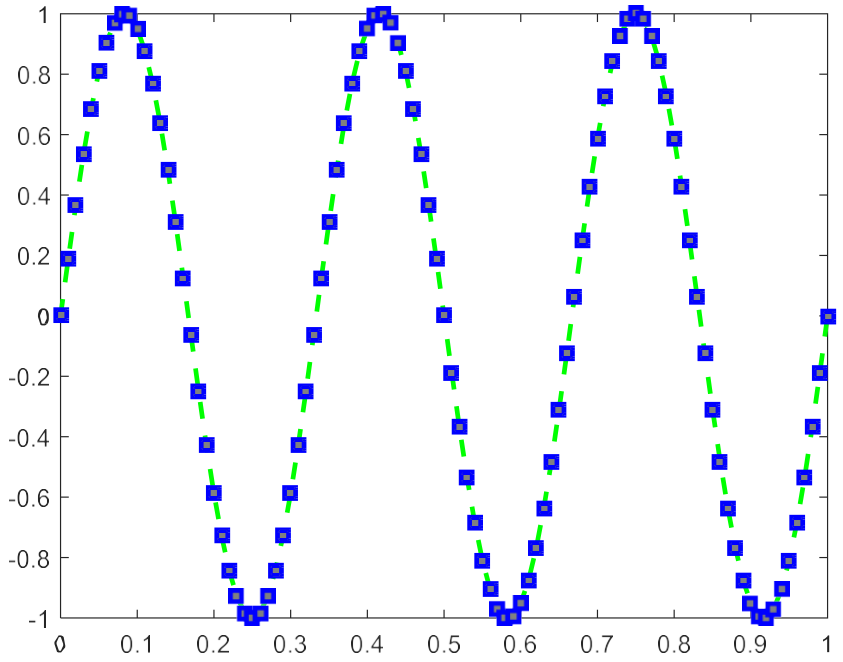
6 (default) | positive value

MarkerEdgeColor — Marker outline color

'auto' (default) | 'none' | RGB triplet | character vector

MarkerFaceColor — Marker fill color

'none' (default) | 'auto' | RGB triplet | character vector

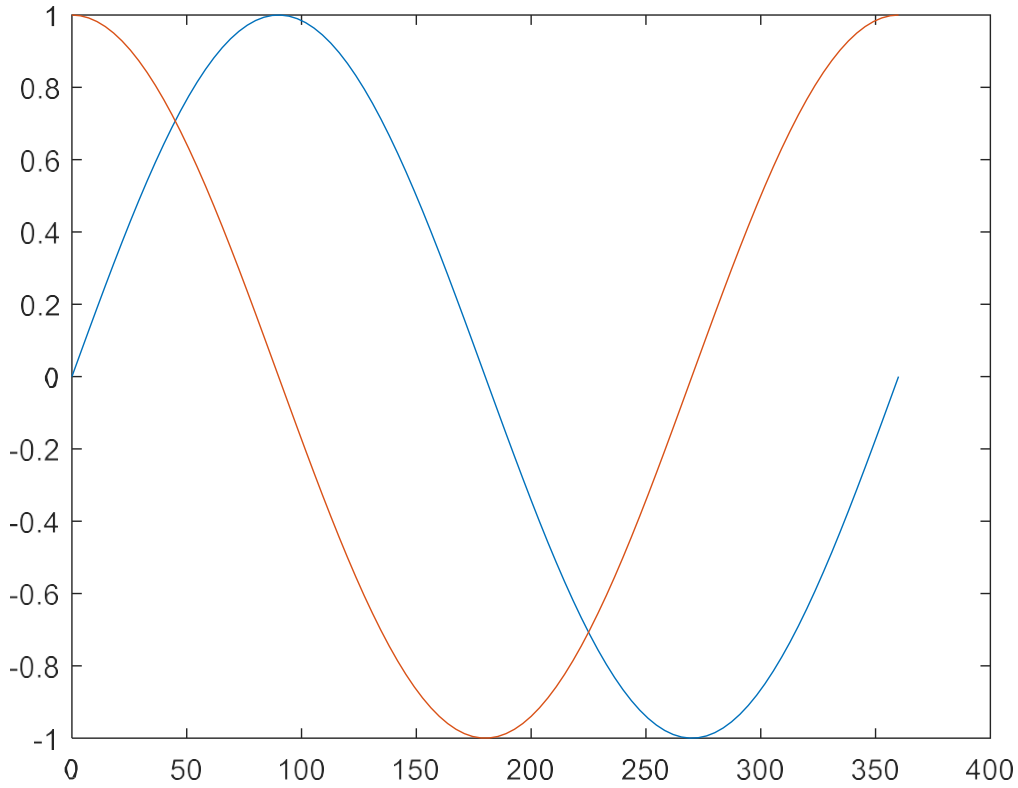


MATLAB Graphics: 2D-Line Plot (Cont.)

Multiple Vectors in Single Plot/Combine Plots in Same Axes (Method-1)

plot(X1,Y1,…,Xn,Yn) plots multiple X,Y pairs using the same axes for all lines.

```
>> x = linspace(0,360);
>> y1=sind(x);
>> y2=cosd(x);
>> plot(x,y1,x,y2)
```



Linearly spaced vector.

linspace(X1, X2) generates a row vector of 100 linearly equally spaced points between X1 and X2.

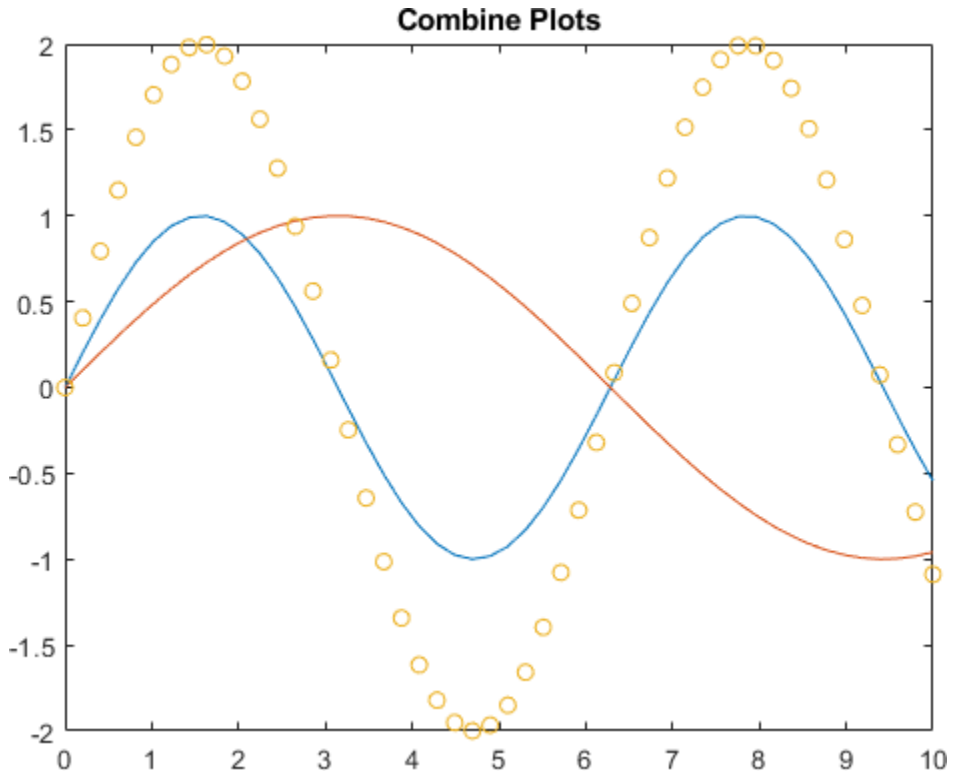
MATLAB Graphics: 2D-Line Plot (Cont.)

Multiple Vectors in Single Plot/Combine Plots in Same Axes (Method-2)

hold ON holds the current plot and all axis properties, including the current color and linestyle.

hold OFF returns to the default mode whereby PLOT commands erase the previous plots and reset all axis properties before drawing new plots.

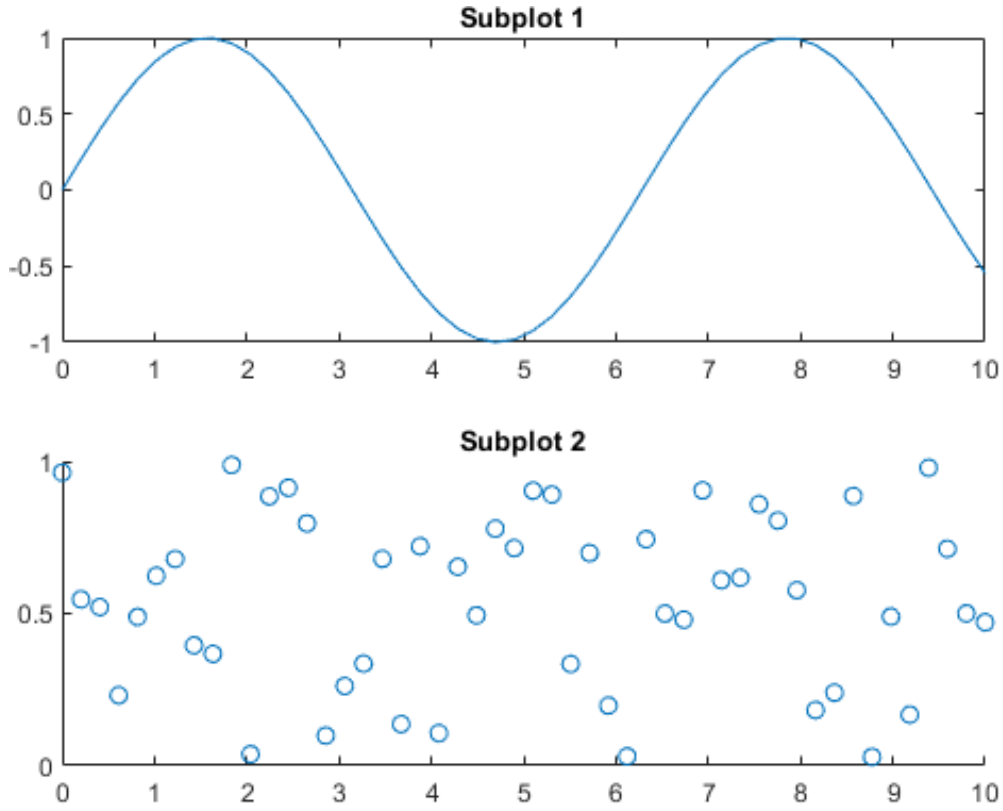
```
x = linspace(0,10,50);
y1 = sin(x);
plot(x,y1)
title('Combine Plots')
hold on
y2 = sin(x/2);
plot(x,y2)
y3 = 2*sin(x);
scatter(x,y3)
hold off
```



Combine Multiple Plots (Cont.)

Create Multiple Axes in Figure Using Subplots

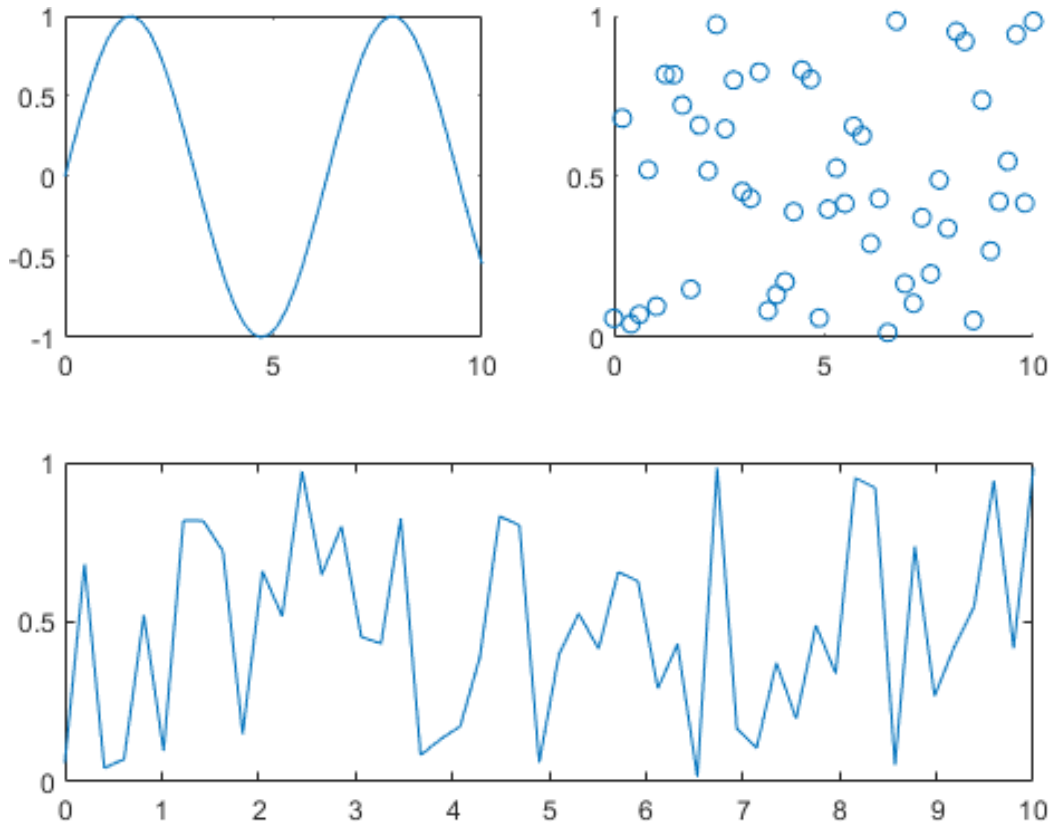
```
subplot(2,1,1);
x = linspace(0,10,50);
y1 = sin(x);
plot(x,y1)
title('Subplot 1')

subplot(2,1,2);
y2 = rand(50,1);
scatter(x,y2)
title('Subplot 2')
```



Combine Multiple Plots (Cont.)

Create Subplot that Spans Multiple Grid Positions

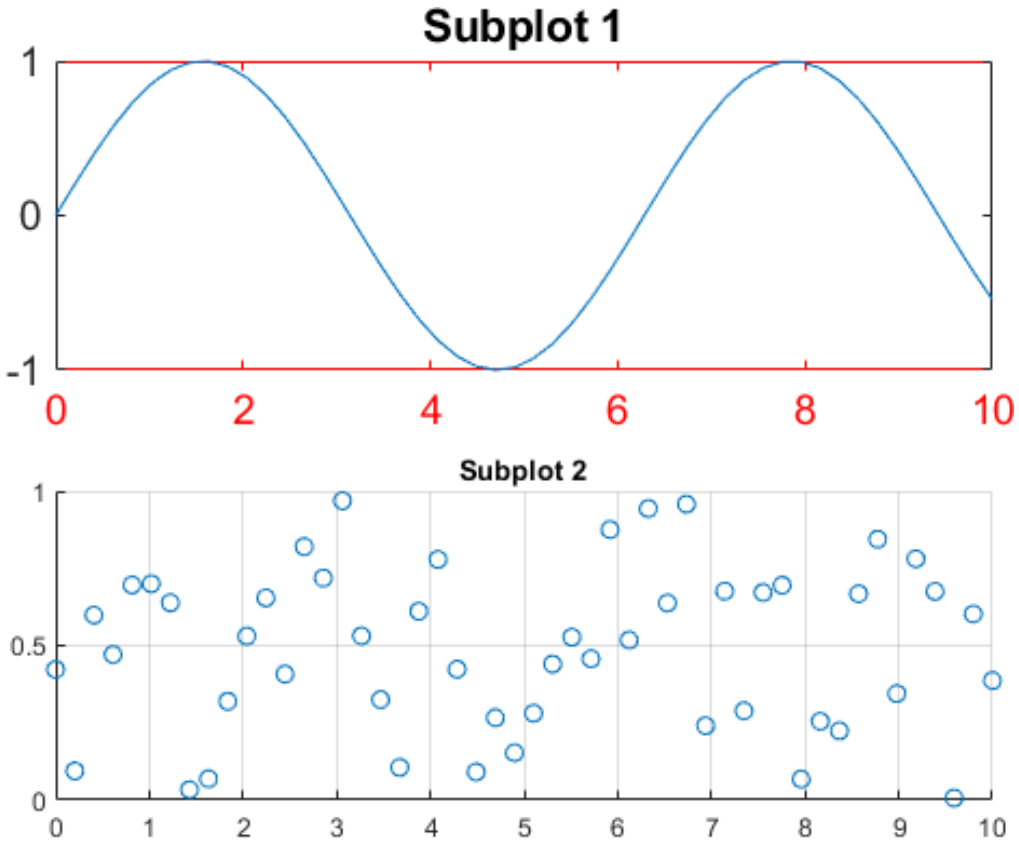
```
figure
subplot(2,2,1);
x = linspace(0,10,50);
y1 = sin(x);
plot(x,y1)

subplot(2,2,2);
y2 = rand(50,1);
scatter(x,y2)

subplot(2,2,[3 4]);
y3 = rand(50,1);
plot(x,y2)
```



Combine Multiple Plots (Cont.)

Modify Subplot Appearance

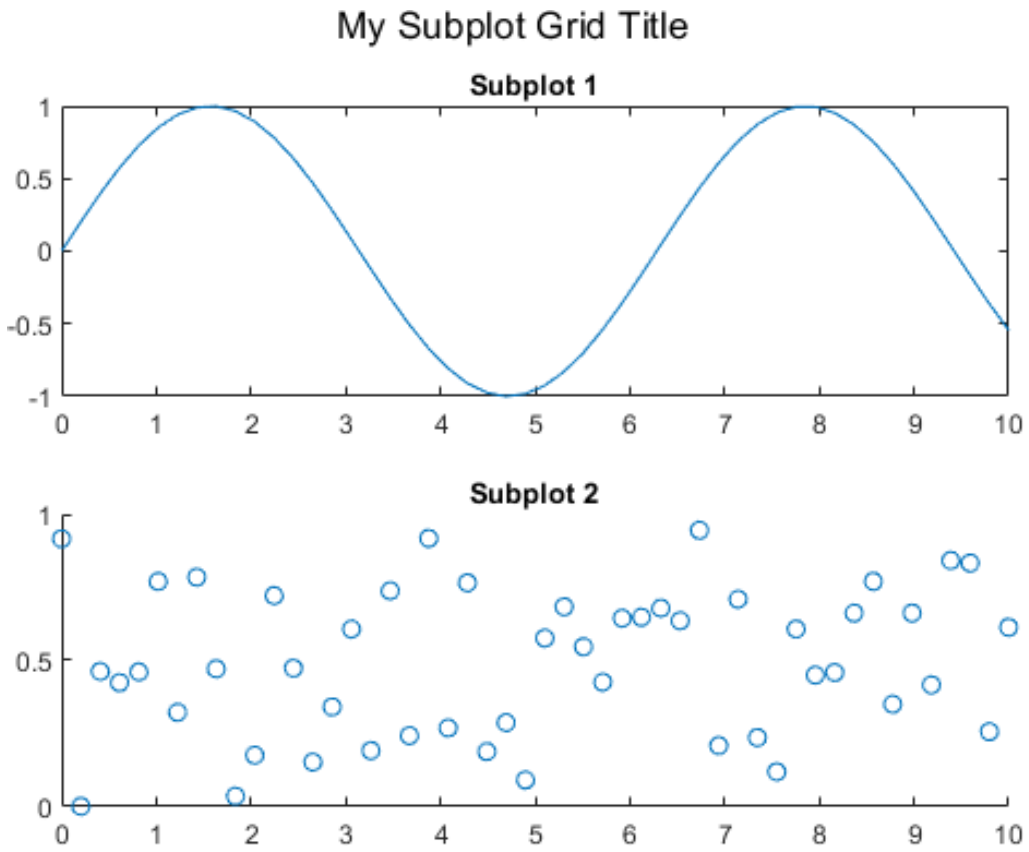
```
figure
ax1 = subplot(2,1,1);
x = linspace(0,10,50);
y1 = sin(x);
plot(ax1,x,y1)
title(ax1,'Subplot 1')
ax1.FontSize = 14;
ax1.XColor = 'red';

ax2 = subplot(2,1,2);
y2 = rand(50,1);
scatter(ax2,x,y2)
title(ax2,'Subplot 2')
grid(ax2,'on')
```



Combine Multiple Plots (Cont.)

Add SuperTitle to Grid of Subplots

```
subplot(2,1,1);
x = linspace(0,10,50);
y1 = sin(x);
plot(x,y1)
title('Subplot 1')

subplot(2,1,2);
y2 = rand(50,1);
scatter(x,y2)
title('Subplot 2')

sgtitle('My Subplot Grid Title')
```



University of Global Village (UGV), Barishal
Dept. of Electrical and Electronic Engineering (EEE)

Experiment-1
Introduction to MATLAB

Course Code: EEE-307/CSE-309
Course Title: Digital Signal Processing

Prepared By
Noor Md Shahriar
Senior Lecturer, Dept. of EEE, UGV

Objectives:

a) To familiarize with MATLAB and some basic commands of MATLAB.

b) To know about variable and variable types in MATLAB.

c) To know about Matrix manipulation in MATLAB.

d) To learn about plotting 2D graphs in MATLAB.

e) To become familiar with MATLAB script/editor.

f) To become familiar with conditional operators and loops in MATLB.

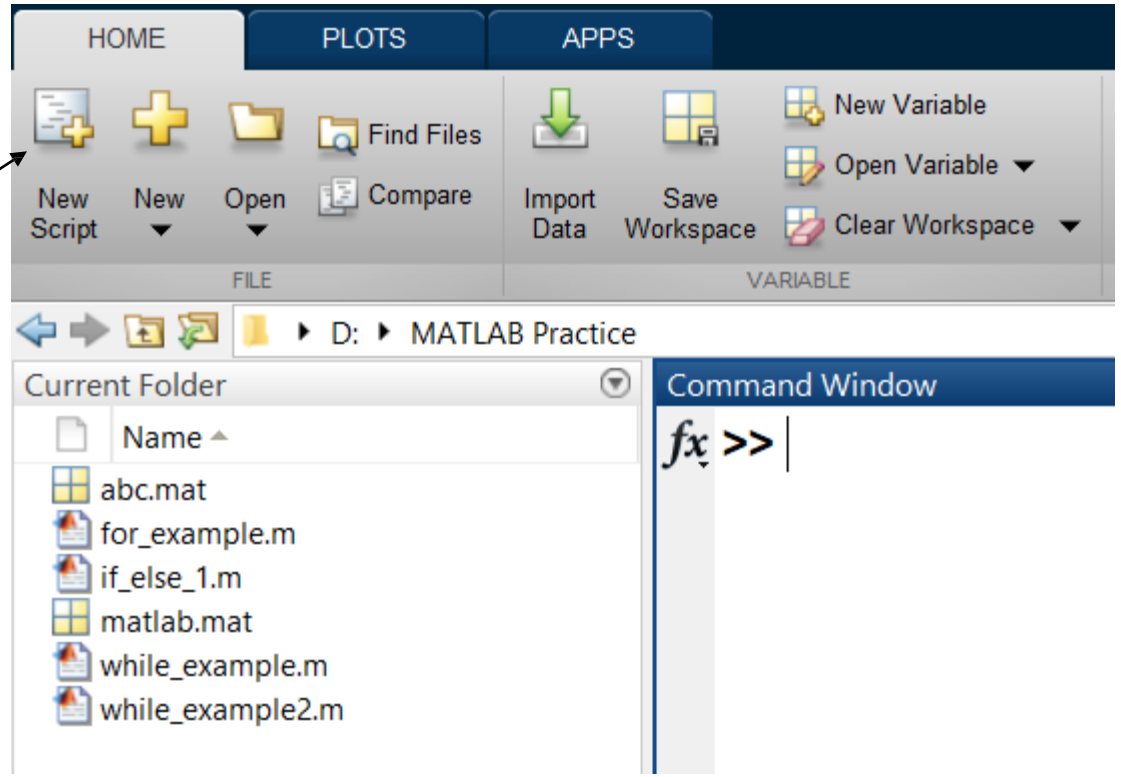
g) To learn about debugging program in MATLAB.

h) To know how to develop an user defined function in MATLAB

Scripts

- Matlab editor: easy to save, edit, run and debug program.
- Used to develop function.
- Use scripts to execute a series of Matlab commands.

Press to create new m-file in the matlab editor

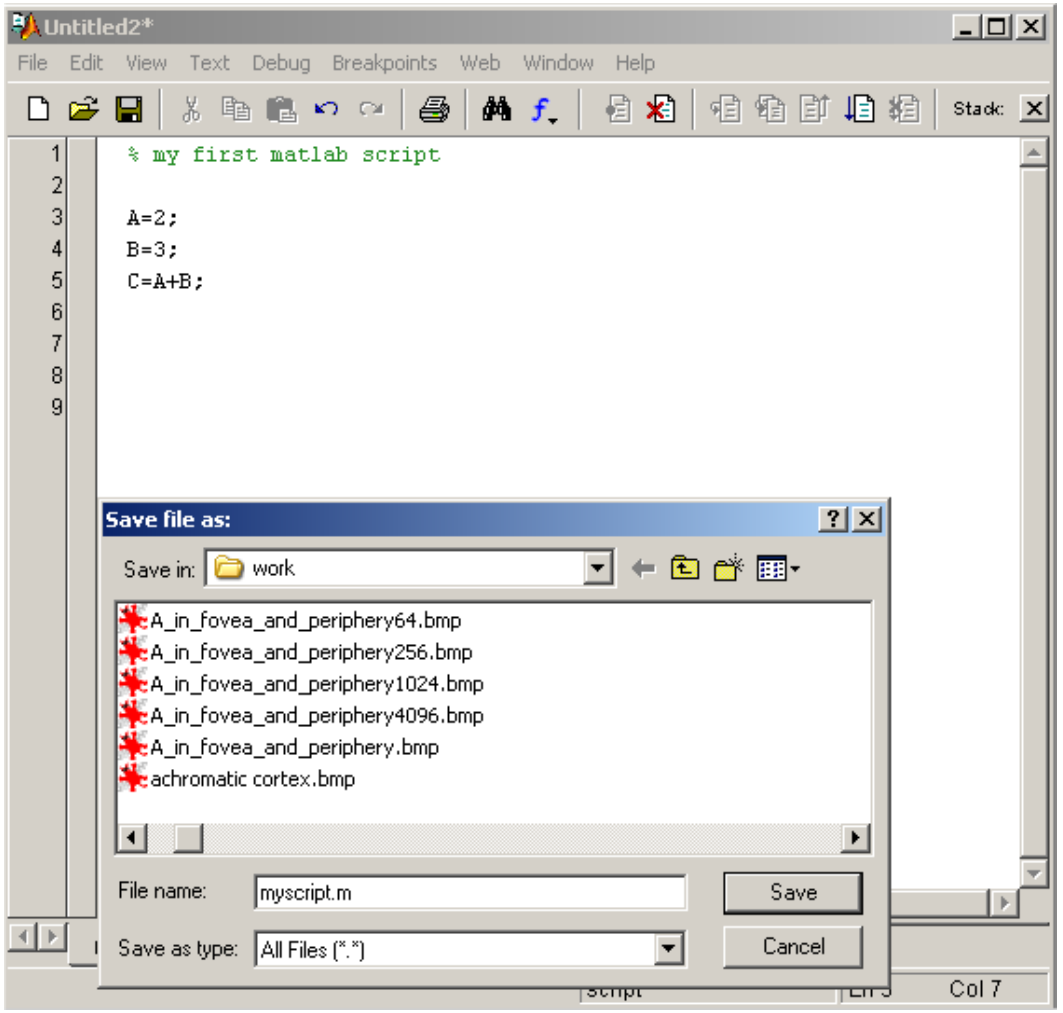


Scripts (Cont.)

- Scripts will manipulate and store variables and matrices in the Matlab Workspace (memory).
- They can be called from the Matlab command line by typing the (case sensitive!) filename of the script file.

  >> myscript
- Scripts can be opened in the editor by the following

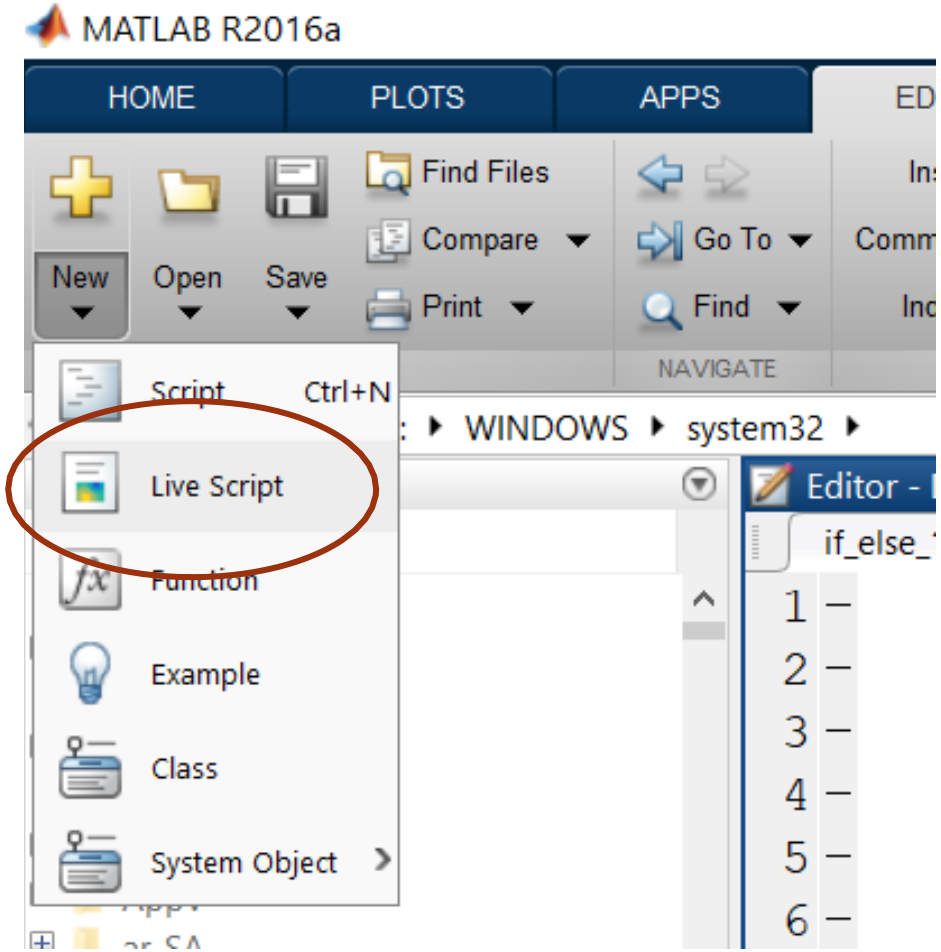
  >> open myscript



Highlight a few lines of your script by left- clicking and dragging the mouse over the lines. Right-click the highlighted lines and select Evaluate Selection.

Live Scripts

- Instead of writing code and comments in plain text, you can use formatting options in *live scripts* to enhance your code.
- Live scripts allow you to view and interact with both code and output and can include formatted text, equations, and images.



If-Else Conditions

Syntax-1

```
if   condition
   block
end
```

Syntax-2

```
if   condition
   block 1
else
   block 2
end
```

Syntax-3

```
if   condition
   block 1
elseif condition
   block 2
…
elseif condition
   block N
end
```

Syntax-4

```
if   condition
   block 1
elseif condition
   block 2
…
elseif condition
   block N
else
   block
end
```

Relational Operator:

== (Determine equality)
~= (Determine inequity)
>= (Determine greater than or equal to)
<= (Determine less than or equal to)
< (Determine less than)
> (Determine greater than)

Logical Operator:

&& (AND)
|| (OR)
~ (NOT)

If-Else Conditions

Example: Checking whether a number is even or odd.

```
number=input('Enter Number; ');
if  (rem(number,2)==0)
    disp('Number is even');
else
    disp('Number is odd');
end
```

Printing/Displaying Output in MATLAB

a) leaving the semicolon off after instruction.

b) "disp" function: It can show variable or text.

c)"fprintf" function: which accepts a C printf-style formatting string.

Example: Finding the type of a MATLAB variable.

```
A=[3,1];
[r,c]=size(A);
if r==1 && c==1
    disp('A is scaler');
elseif  (r==1 && c>1)
        disp('Row Vector');
elseif (r>1 && c==1)
            disp('Column Vector');
       else
            disp('A is MATRIX');
end
```

Switch Statement

Syntax:

```
switch switch_expression
  case case_expression
    statements
  case case_expression
    statements
   …
  otherwise
    statements
end
```

Example-1

```
n = input('Enter a number: ');

switch n
    case -1
        disp('negative one')
    case 0
        disp('zero')
    case 1
        disp('positive one')
    otherwise
        disp('other value')
end
```

Switch Statement (Cont.)

Example-2

```
x = [12 64 24];
plottype = 'pie3';

switch plottype
    case 'bar'
      bar(x)
       title('Bar Graph')
   case {'pie','pie3'}
       pie3(x)
       title('Pie Chart')
   otherwise
       warning('Unexpected plot type. No plot created.')
end
```

Example-3

```
result = 52;

switch(result)
  case 52
     disp('result is 52')
  case {52, 78}
     disp('result is 52 or 78')
end
```

For loop

Syntax:

```
for index = values
   statements
end
```

index=initVal:endVal

Increment the index variable from initVal to endVal by 1, and repeat execution of statements until index is greater than endVal.

```
for index = 0:10
    statements
end
```

index=initVal:step:endVal

Increment index by the value step on each iteration, or decrements index when step is negative.

```
for v = 1:-0.2:0
   statements
end
```

Example: Finding average of numbers

```
clc % clear command window
clear all  % clear workspace
disp('Find average');
n=input('How many numbers: ');
sum=0;
for i=1:1:n
   number(i)=input('');
   sum=sum+number(i);
end
fprintf('The average is: %f',sum/n);
```

While loop

```
while expression
   statements
end
```

N.B. while loop to repeat when expression/condition is true.

Example: Finding the factorial of a number.

```
n = input('Enter the Number:'); f
= n;
while n > 1 n
   = n-1; f =
   f*n;
end
fprintf('The factorial of %d is %d \n', n,f);
```

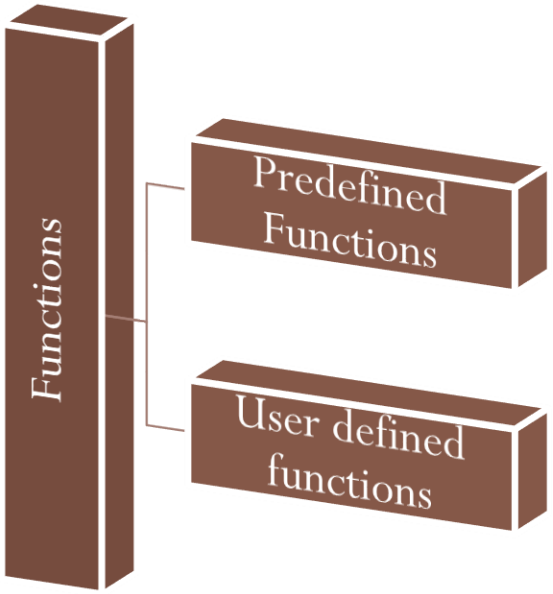
Example: Construction of Sawtooth wave.

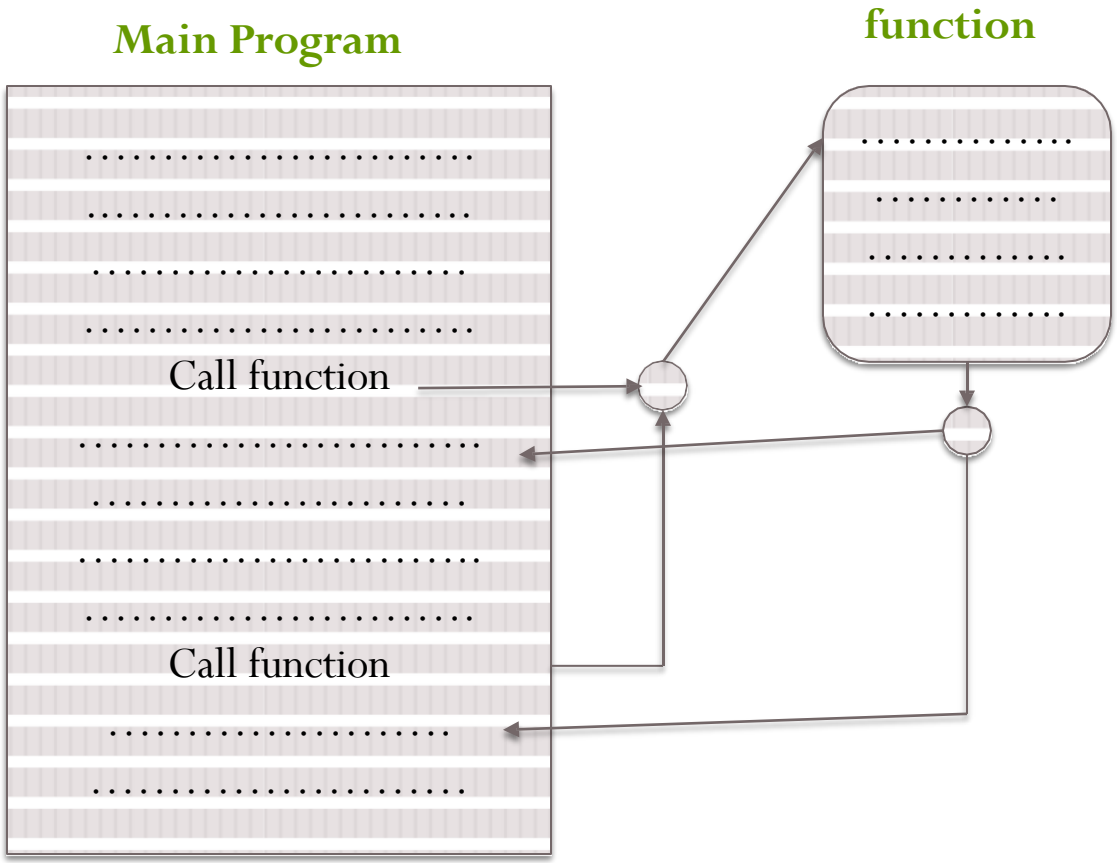
$$y = \frac{4}{\pi}\sin(2\pi f t) + \left(\frac{-4}{2\pi}\right)\sin(2\pi(2f)t) + \frac{4}{3\pi}\sin(2\pi(3f)t) + \left(\frac{-4}{4\pi}\right)\sin(2\pi(4f)t) + \ldots\ldots$$

```
m=input('Number of harmonics:');
t=0:0.001:2;
y=0;
n=1;
while n<=m
y=y+(4/(n*pi))*sin(2*pi*n*t);
n=n+1;
end
plot(t,y,'g','linewidth',3);
```

Function

- ✓ A function is a group of statements that together perform a task.
- ✓ Functions allow the programmer to break down large, complex problems to smaller and more manageable pieces.
- ✓ A function is a reusable set of instructions.





Functions in MATLAB

- A MATLAB function is a special type of M-file that runs in its own workspace.
- Users can write functions which can be called from the command line.
- Functions can accept input variable(s)/matrice(s) and will output variable(s)/matrice(s).
- Functions will **not** manipulate variable(s)/matrice(s) in the Matlab Workspace.
- Remember that the filename of a function will be its calling function name.
- Don't overload any built-in functions by using the same filename for your functions or scripts!
- Functions can be opened for editing using the **open** command. Many built-in Matlab functions can also be viewed using this command.

Syntax of User defined function in MATLAB

```
function [out_arg1, out_arg2, …]= function_name (in_arg1, in_arg2, …)

Function body

end
```

Syntax of Functions

Having no input and output arguments

```
function name

end
```

Example

```
function txt
disp('This is a sample program');
end
```

Having input arguments and no output arguments

```
function    name(in_arg1, in_arg2, …)

end
```

Example

```
function txt(a)
switch a
case 1
disp('1st text is showing function');
case 2
disp('2nd  text is showing function');
case 3
disp('3rd  text is showing function');
otherwise
disp('Nothing to show');
end
```

Having both input and output arguments

```
function    [out_arg1,  …]= ]=name(in_arg1,  …)

end
```

Example

```
function [Y]=reshape_matrix(A,r,c)
if  r>0
A(r,:)=[];
end
if c>0
A(:,C)=[];
end
Y=A;
end
```

Sub-Function

Example

```
function [Y, Aavg, Ravg]=reshape_matrix_info(A,r,c)
Aavg=avg(A);
if  r>0
A(r,:)=[];
end
if c>0
A(:,C)=[];
End
Y=A;
Ravg=avg(Y);
End
function mean=avg(M)
[p,q]=size(M);
mean=sum(M(:))/(p*q);
end
```

University of Global Village (UGV), Barishal
Dept. of Electrical and Electronic Engineering

Experiment-2
Continuous time and Discrete time representation of signals

Course Code: EEE-307/CSE-309
Course Title: Digital Signal Processing Sessional

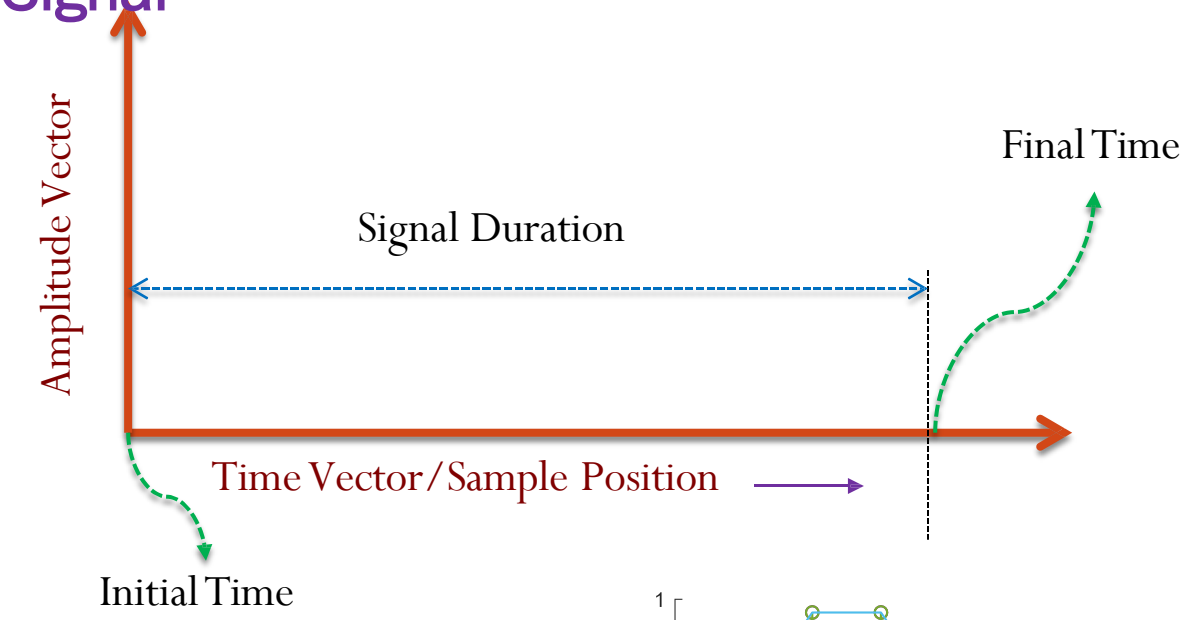
Prepared By
Noor Md Shahriar
Senior Lecturer, Dept. of EEE, UGV

Objectives

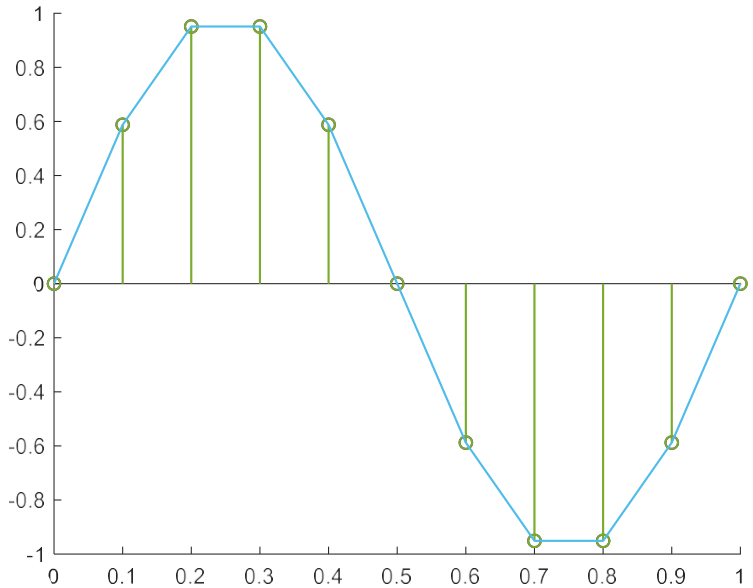
a) To represent (plot) basic signals (sine, cosine, sinc, unit impulse, unit step, unit ramp, exponential signal) in MATLAB.

b) To generate noise signal in MATLAB.

Representation of basic Signal




```
t=0:0.1:1;
y=sin(2*pi*t);
scatter(t,y,"LineWidth",1)
hold on
stem(t,y,"LineWidth",1)
hold on
plot(t,y,"LineWidth",1)
```



Time Vector/Sample Position

t= initial_time: sampling_interval:final_time

or

t= initial_time: (1/Fs):final_time

n= t/sampling_interval

or

n= 0: length(t)-1

Signal duration: Initial time and final time defines the duration of signal.

Sampling Interval: It defines the sampling rate/ total number of sample in the signal.

Amplitude Vector

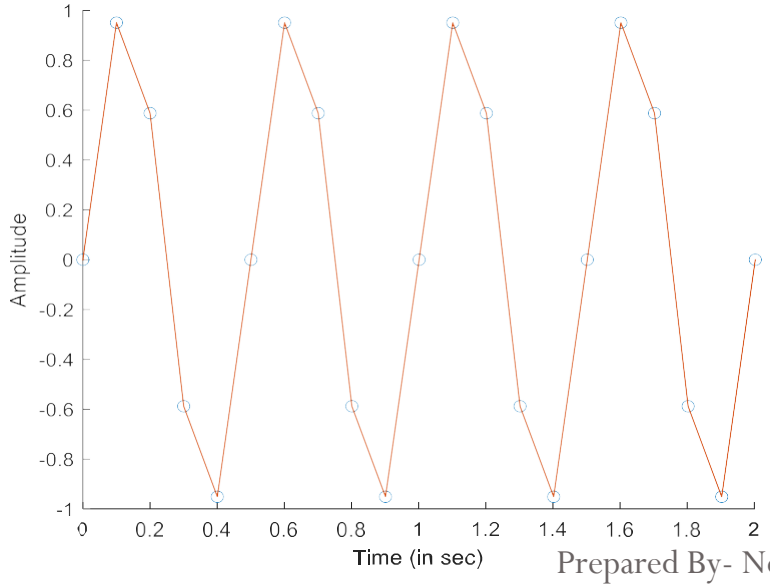
Sample Value

y=A sin(2*pi*F*t)

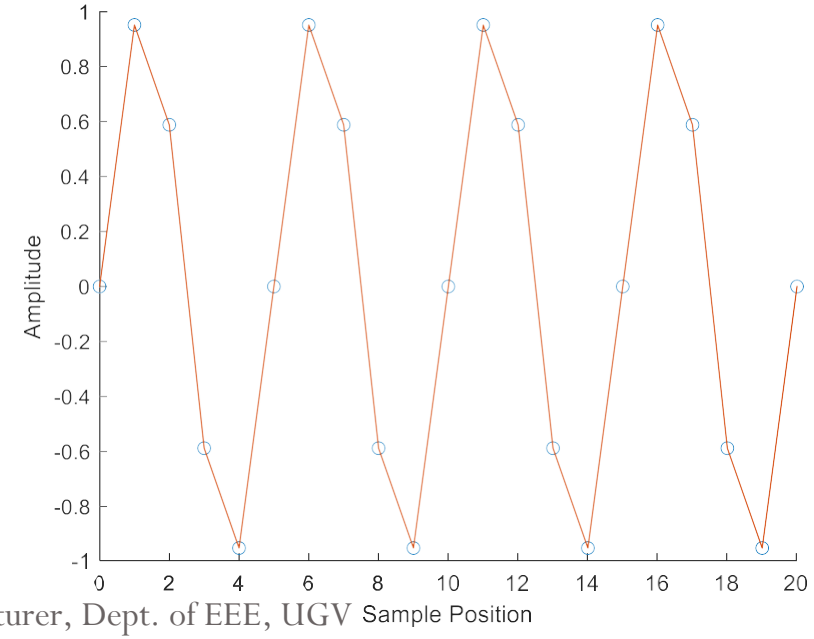
y=A cos(2*pi*F*t)

Representation of basic Signal (Cont.)

```
Fs=10;
T=1/Fs;
F=2;
duration=2;
t=0:T:duration;
y=sin(2*pi*F*t);
scatter(t,y);
hold on
plot(t,y,"LineWidth",1);
xlabel("Time (in sec)");
ylabel("Amplitude");
```




```
Fs=10;
T=1/Fs;
F=2;
duration=2;
t=0:T:duration;
n=t./T;      % n=0:length(t)-1;
y=sin(2*pi*F*t);
scatter(n,y);
hold on
plot(n,y,"LineWidth",1);
xlabel("Sample Position");
ylabel("Amplitude");
```

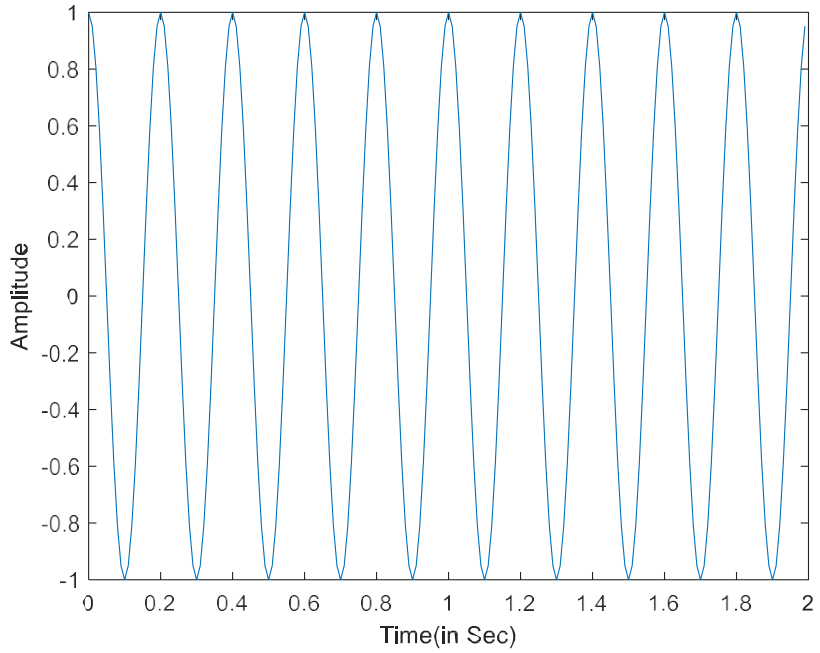


Representation of basic Signal (Cont.)

Cos

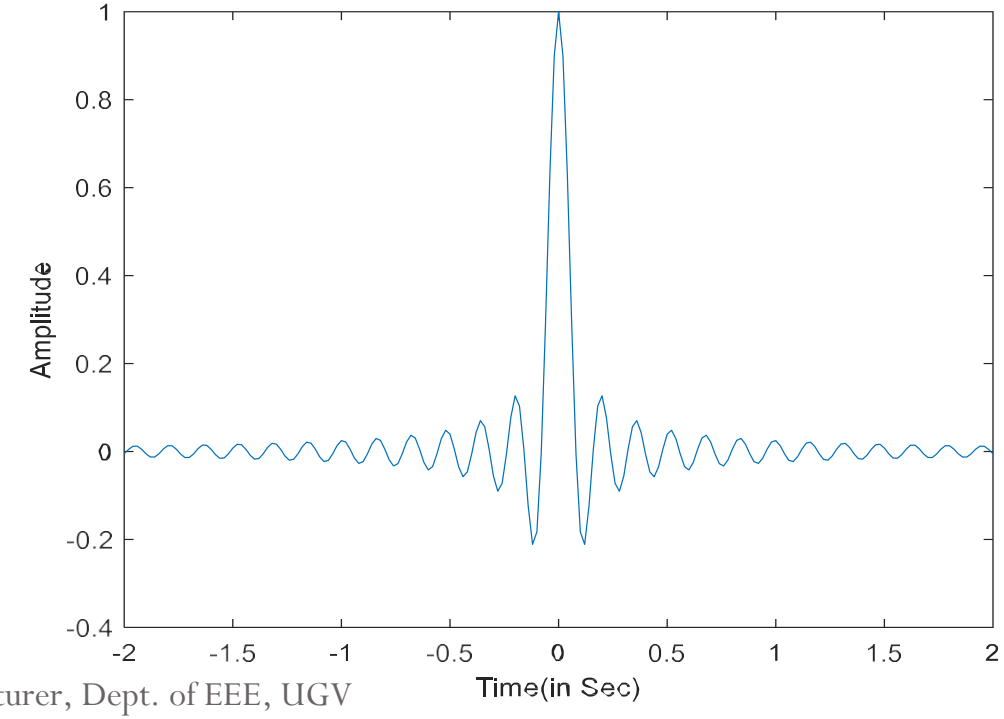
```
Fs=100;
T=1/Fs;
F=5;
duration=2;
t=0:T:duration-T;
y=cos(2*pi*F*t);
plot(t,y,"LineWidth",1);
xlabel("Time(in Sec)");
ylabel("Amplitude");
```

As continuous time plot



Sinc

```
Fs=100;
T=1/Fs;
F=2;
duration=2;
t=-duration:T:duration;
y=sinc(2*pi*F*t);
plot(t,y,"LineWidth",1);
xlabel("Sample Position");
ylabel("Amplitude");
```



Representation of basic Signal (Cont.)

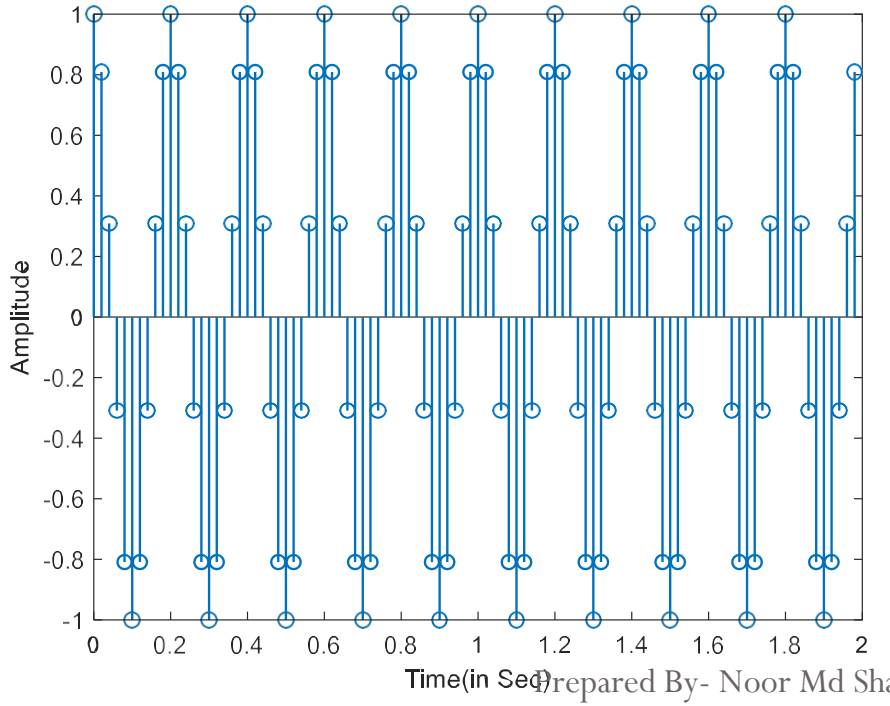
Cos

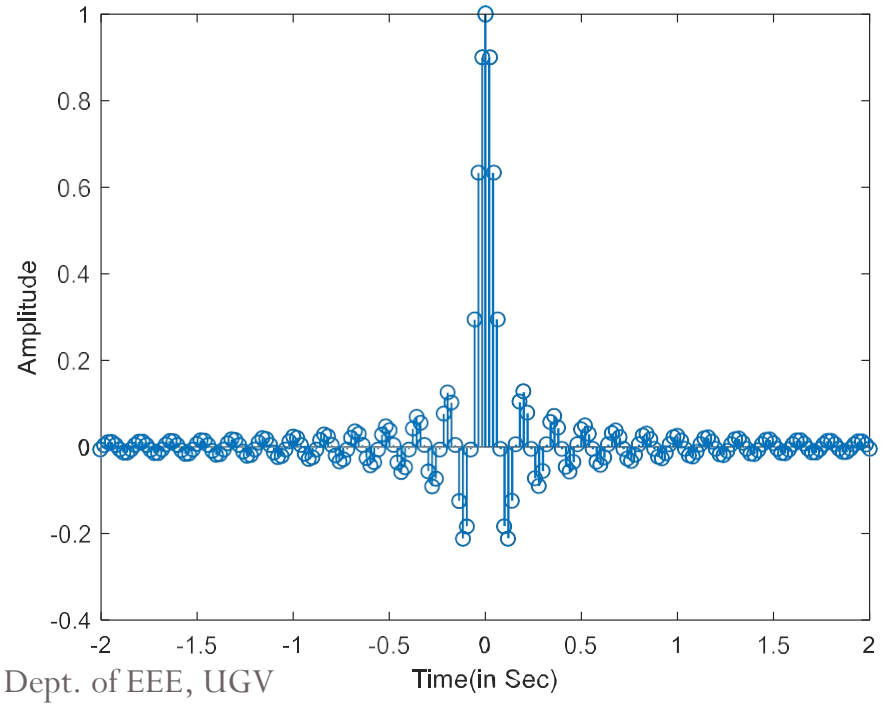
```
Fs=100;
T=1/Fs;
F=5;
duration=2;
t=0:T:duration-T;
y=cos(2*pi*F*t);
stem(t,y,"LineWidth",1);
xlabel("Time(in Sec)");
ylabel("Amplitude");
```

As discrete time plot

Sinc

```
Fs=50;
T=1/Fs;
F=2;
duration=2;
t=-duration:T:duration;
y=sinc(2*pi*F*t);
stem(t,y,"LineWidth",1);
xlabel("Time(in Sec)");
ylabel("Amplitude");
```





Representation of basic Signal (Cont.)

Unit Impulse

```
a=-5;
b=10;
n=a:b;
y=[zeros(1,abs(a)),ones(1,1),zeros(1,b)];
stem(n,y,'LineWidth',1,'Color','g');
xlabel('Sample position');
ylabel('Amplitude');
```

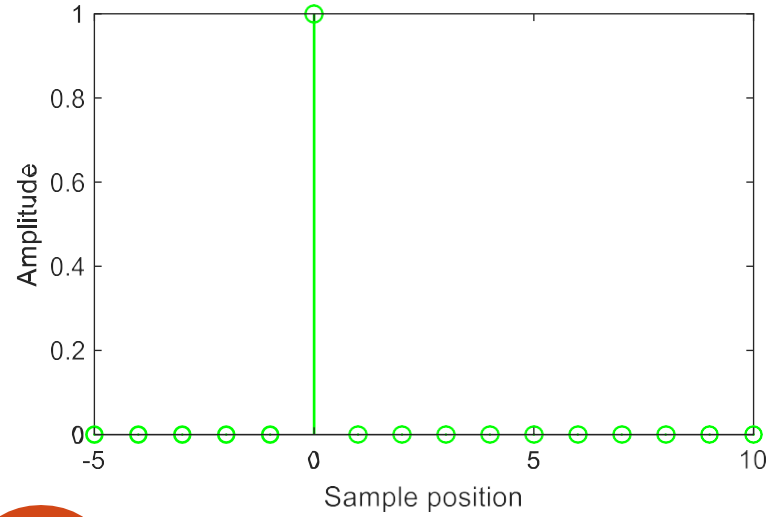
Unit Step

```
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');
xlabel('Sample position');
ylabel('Amplitude');
```

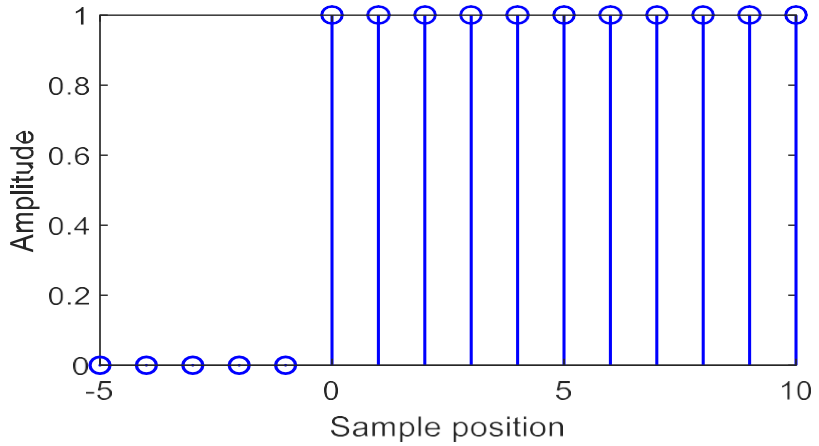
Unit Ramp

```
n=0:10;
y=n;
stem(n,y,'LineWidth',1
,'Color','m');
xlabel('Sample
position');
ylabel('Amplitude');
```

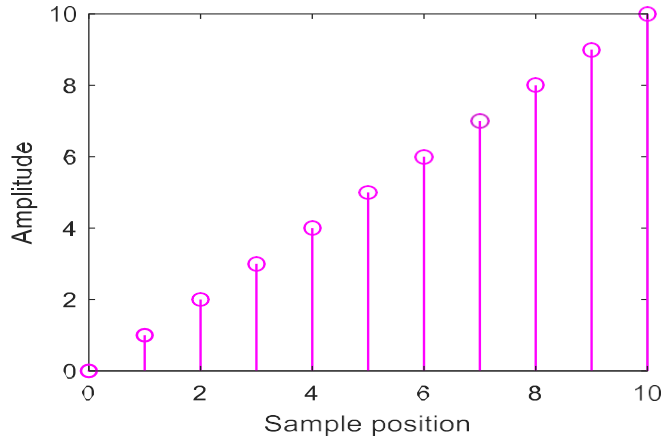
$$\delta(n) \equiv \begin{cases} 1, & \text{for } n = 0 \\ 0, & \text{for } n \neq 0 \end{cases}$$



$$u(n) \equiv \begin{cases} 1, & \text{for } n \geq 0 \\ 0, & \text{for } n < 0 \end{cases}$$



$$u_r(n) \equiv \begin{cases} n, & \text{for } n \geq 0 \\ 0, & \text{for } n < 0 \end{cases}$$

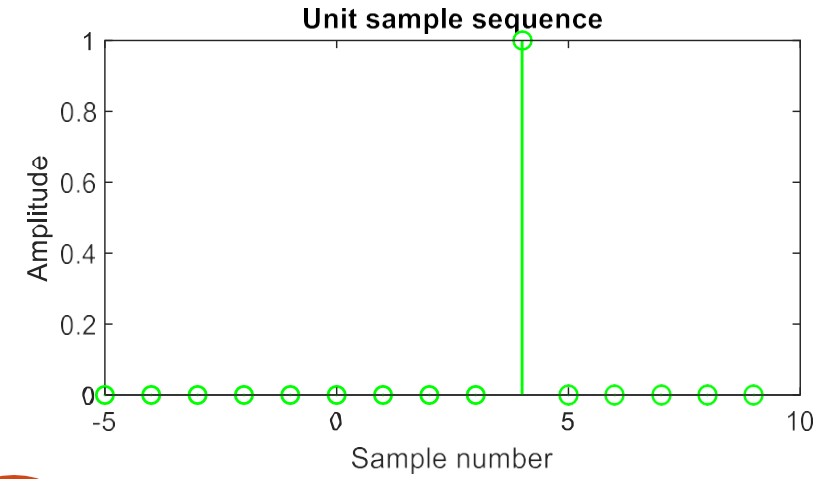


Representation of basic Signal (Cont.)

User defined function to generate Sample Sequence

```
function [x,n]=impuseq(n0,n1,n2)
n=n1:1:n2;
x=double([n==n0]);
```

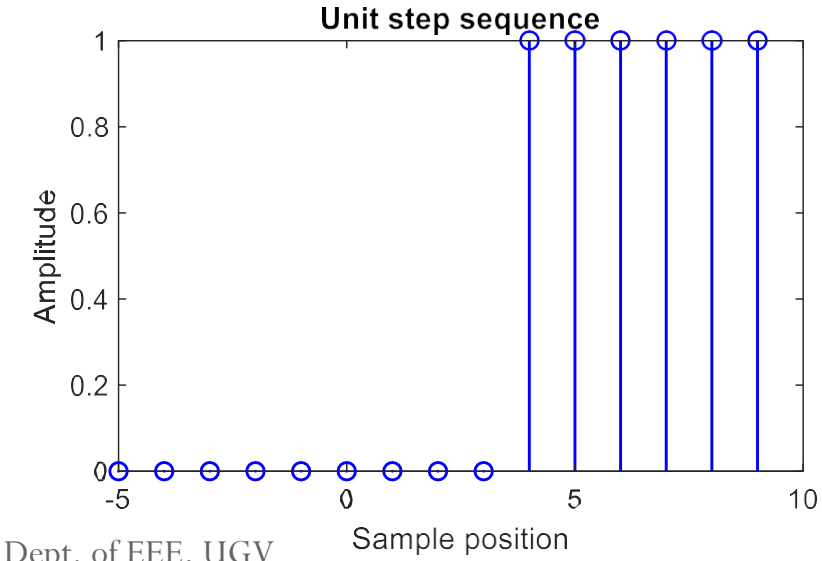
```
>> [Y,N]=impuseq(4,-5,9);
>> stem(N,Y,"LineWidth",1,"Color",'g');
>> title('Unit sample sequence');
>> xlabel('Sample number');
>> ylabel('Amplitude');
```



User defined function to generate Step Sequence

```
function [x,n]=stepseq(n0,n1,n2)
n=n1:1:n2;
x=double([n>=n0]);
```

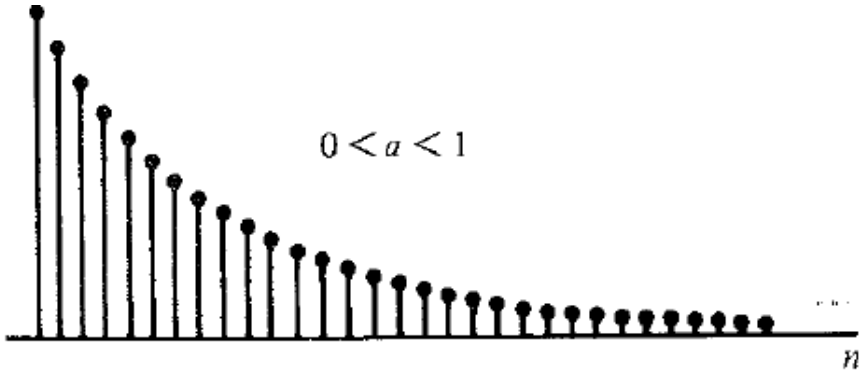
```
>> [Y,N]=stepseq(4,-5,9);
>> stem(N,Y,"LineWidth",1,"Color",'b');
>> title('Unit step sequence');
>> xlabel('Sample position');
>> ylabel('Amplitude');
```

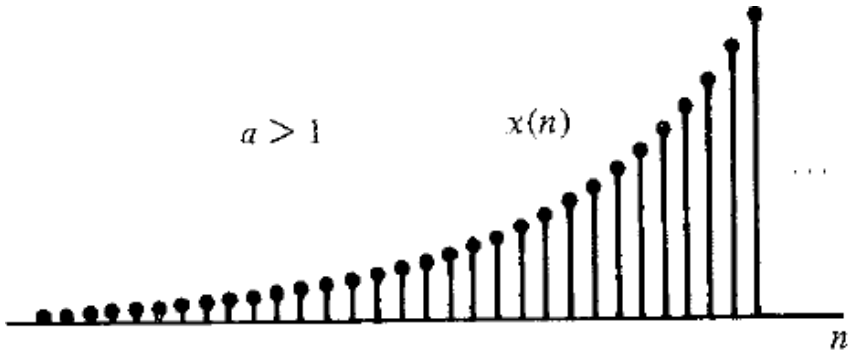


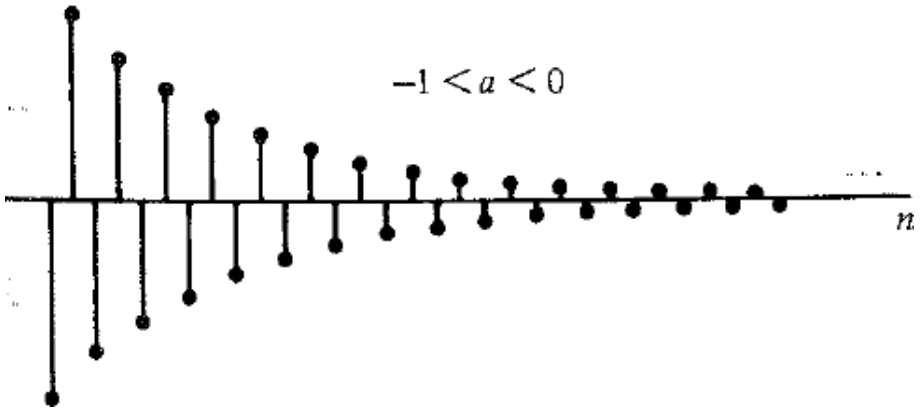
Representation of basic Signal (Cont.)

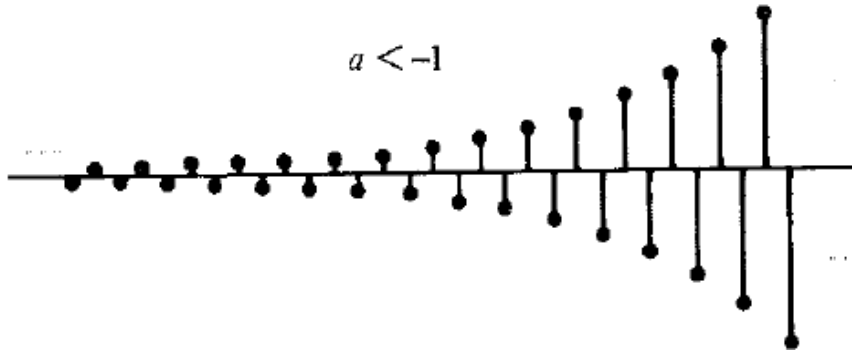
Exponential Signal

$x(n) = a^n$ for all n





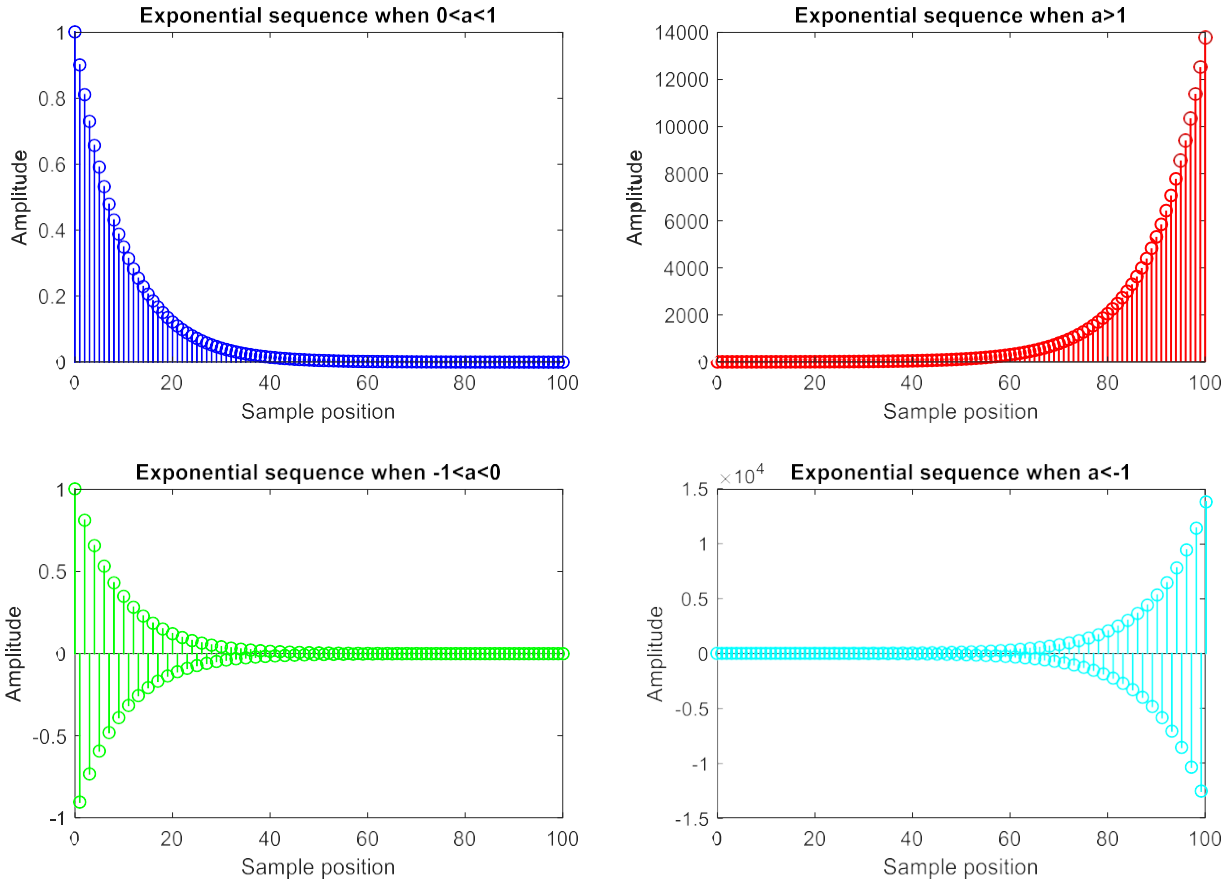




Representation of basic Signal (Cont.)

Exponential Signal

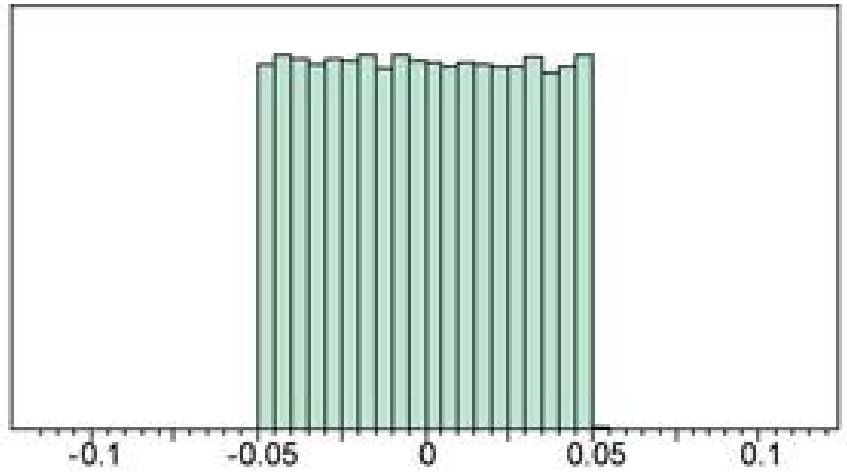
```
n=0:100;
subplot(2,2,1)
a1=0.9;
y1=a1.^n;
stem(n,y1,"LineWidth",1,"Color",'b');
title('Exponential sequence when 0<a<1');
xlabel('Sample position');
ylabel('Amplitude');
subplot(2,2,2)
a2=1.1;
y2=a2.^n;
stem(n,y2,"LineWidth",1,"Color",'r');
title('Exponential sequence when a>1');
xlabel('Sample position');
ylabel('Amplitude');
subplot(2,2,3)
a3=-0.9;
y3=a3.^n;
stem(n,y3,"LineWidth",1,"Color",'g');
title('Exponential sequence when -1<a<0');
xlabel('Sample position');
ylabel('Amplitude');
subplot(2,2,4)
a4=-1.1;
y4=a4.^n;
stem(n,y4,"LineWidth",1,"Color",'c');
title('Exponential sequence when a<-1');
xlabel('Sample position');
ylabel('Amplitude');
```



Noise (White Noise) signal generation

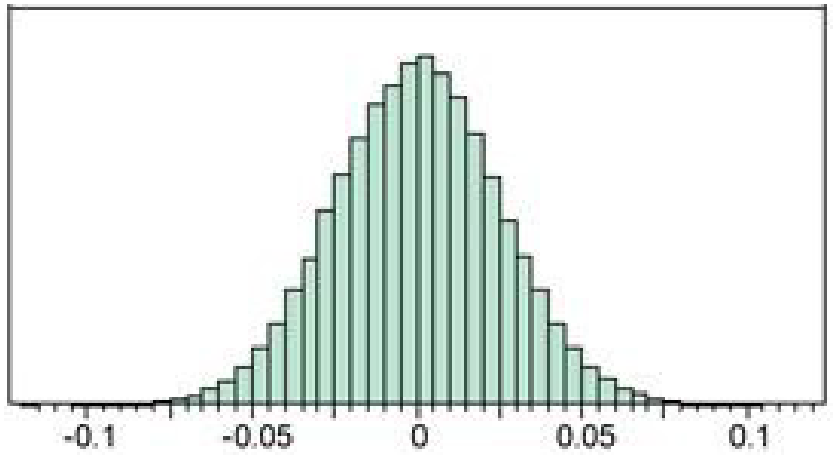
White noise is random and has *equal power over the range of frequencies*. It derives its name from *white light*, which has equal brightness at all wavelengths in the visible region.

Distribution of White Noise



Uniformly distributed:

The simplest way to understand noise is to generate it, and the simplest kind to generate is uncorrelated uniform noise (UU noise). "Uniform" means the signal contains random values from a uniform distribution; that is, every value in the range is equally likely. "Uncorrelated" means that the values are independent; that is, knowing one value provides no information about the others.



Gaussian distribution or Normally distributed random noise:

In this distribution, the most common noise errors are small

Random Sequence (Uniform Distribution)

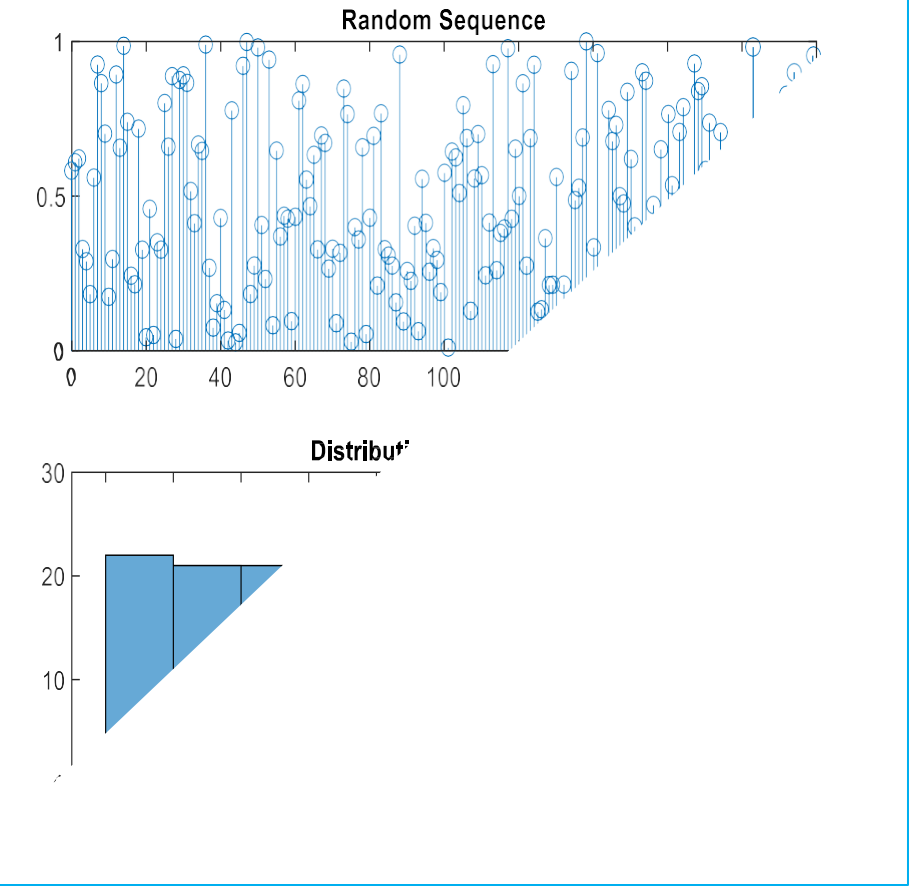
The rand function generates arrays of random numbers whose elements are uniformly distributed in the interval (0,1).

rand(1,N): Row vector
rand(N,1): Column vector
rand(N) : N × N Matrix
rand(N,M) : N × M Matrix

Mean, $\bar{x} = \frac{1}{N}\sum_{n=0}^{N-1} x(n) \approx 0.5$

$$Variance, \quad \sigma^2 = \frac{1}{N}\sum_{n=0}^{N-1}[x(n) - \bar{x}]^2 \approx 0$$

```
x=rand(1,200);
n=0:length(x)-1;
subplot(2,1,1);
stem(n,x);
title('Random Sequence');
subplot(2,1,2);
histogram(x,10);
title('Distribution of
Sequence');
m=mean(x);
v=var(x);
fprintf('Mean=%f and
Variance=%f',m,v);
```



Mean=0.480287 and Variance=0.081897

Generating a uniform distribution of random numbers on a specified interval [a,b].

```
>> a = 10;
>> b = 50;
>> x = a + (b-a) * rand(5)
```

Random Sequence (Normally distributed random numbers)

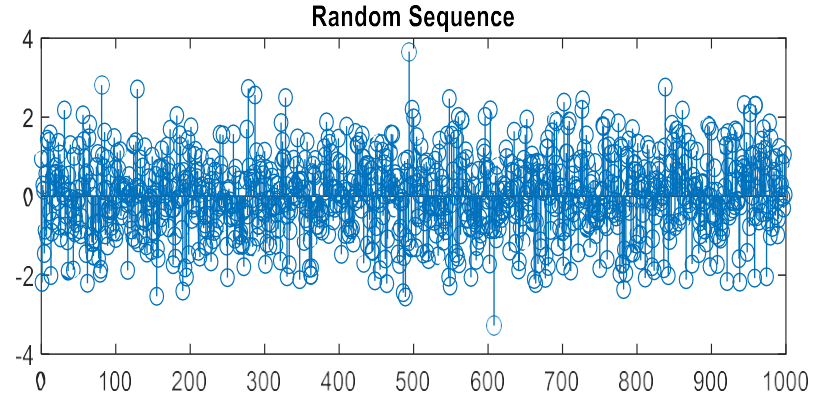
The randn function generates arrays of random numbers whose elements are normally distributed having zero mean and variance one.

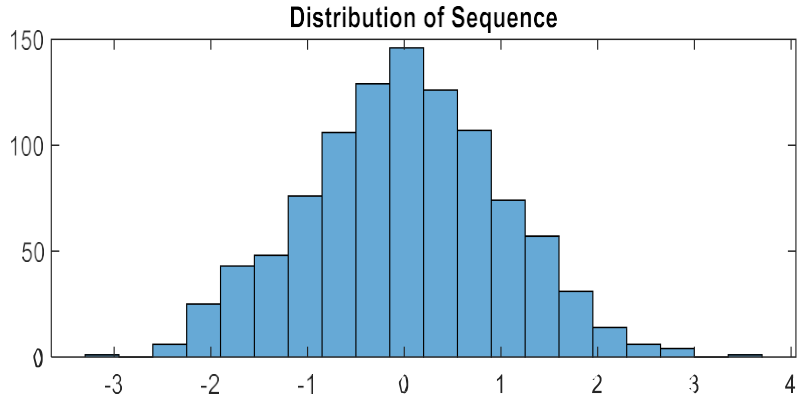
randn: Single value
randn(1,N): Row vector
randn(N,1): Column vector
randn(N) : N × N Matrix
randn(N,M) : N × M Matrix

$$\text{Mean, } \bar{x} = \frac{1}{N}\sum_{n=0}^{N-1} x(n) \approx 0$$

$$\text{Variance, } \sigma^2 = \frac{1}{N-1}\sum_{n=0}^{N} [x(n) - \bar{x}]^2 \approx 1$$

```
x=randn(1,1000);
n=0:length(x)-1;
subplot(2,1,1);
stem(n,x);
title('Random Sequence');
subplot(2,1,2);
histogram(x,10);
title('Distribution of Sequence');
m=mean(x);
v=var(x);
fprintf('Mean=%f and Variance=%f',m,v);
```

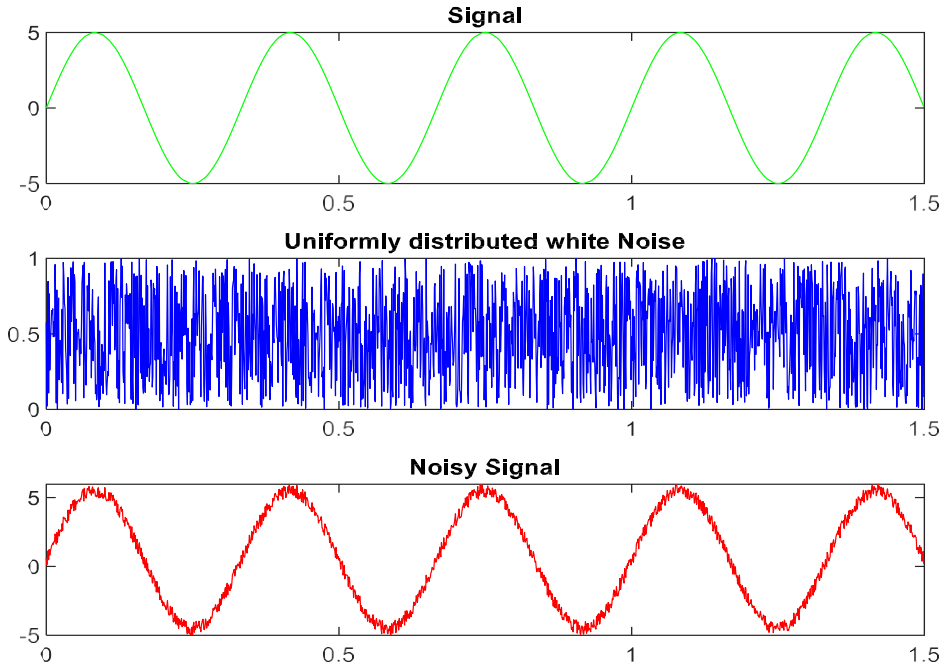


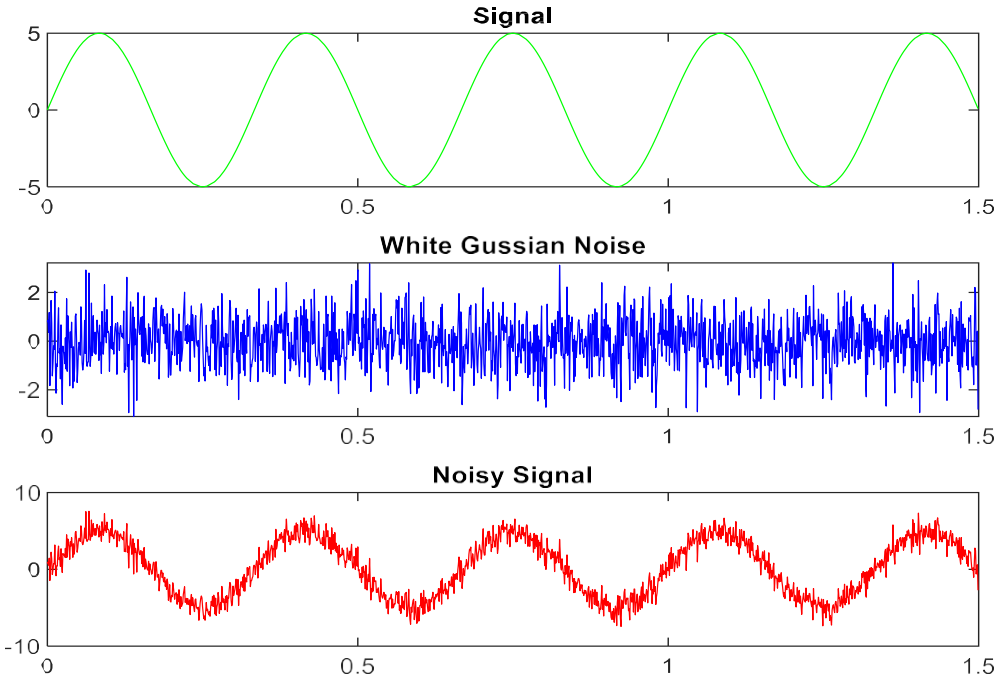


Mean=0.000357 and Variance=1.030061

Generating Noisy Signal

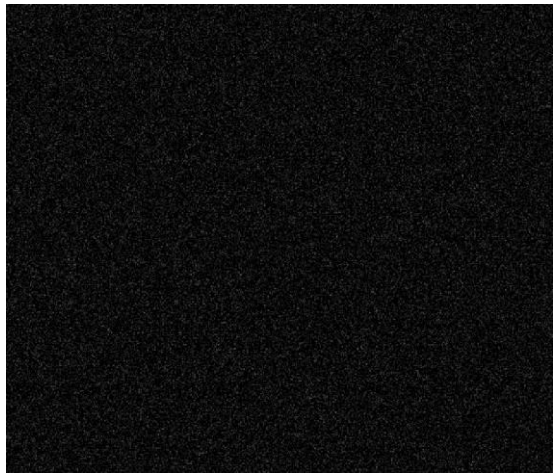
```
clc; clear;
Fs=1000; F=3; T=1/Fs;
duration=1.5;
t=0:T:duration-T;
y=5*sin(2*pi*F*t);
noise_uniform=rand(size(t));
%noise_uniform=randn(size(t));
noisy_signal=y+noise_uniform;
subplot(3,1,1); plot(t,y,'g');
title('Signal');
subplot(3,1,2);
plot(t,noise_uniform,'b');
title('Uniformly distributed white noise');
%title('White Gussian Noise');
subplot(3,1,3);
plot(t,noisy_signal,'r');
title('Noisy Signal');
```

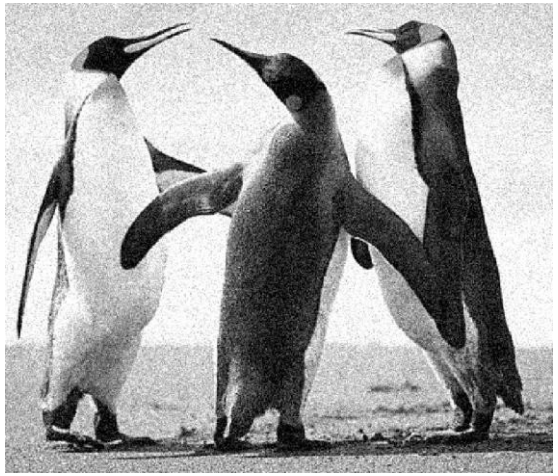




Adding noise to an image

```
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);
```





University of Global Village (UGV), Barishal
Dept. of Electrical and Electronic Engineering

Experiment-3
Analog to digital conversion

Course Code: EEE-307/CSE-309
Course Title: Digital Signal Processing Sessional

Prepared By
Noor Md Shahriar
Senior Lecturer, Dept. of EEE, UGV

Objectives

a)To understand the Analog to digital conversion process: Sampling, Quantization and Coding.

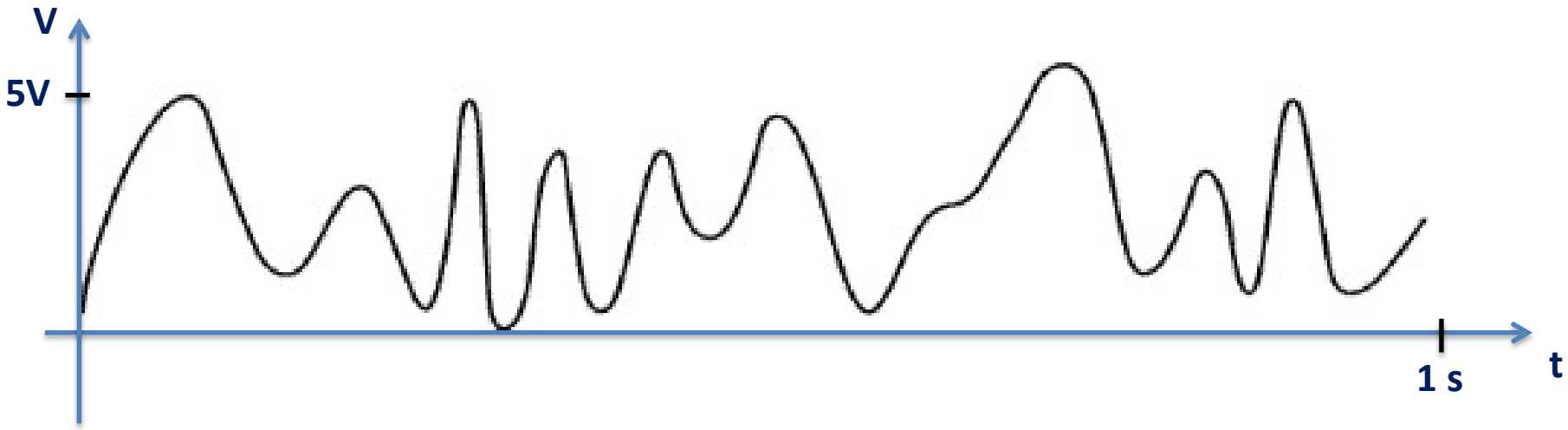
b) To convert continuous time signal into digital signal using MATLAB code.

c)To analyze the effect of sampling rate and quantization level in A/D conversion.

What is Analog Signal? Why A/D Conversion?

Signals in the real world are analog: ***light, sound, temperature, pressure, acceleration or other phenomenon***. So, real-world signals must be converted into digital, using a circuit called ADC (Analog-to-Digital Converter), before they can be manipulated by ***digital equipment***.

Continuous time and continuous valued signal is known as analog signal.



$x = \{a, b, c, d\}$

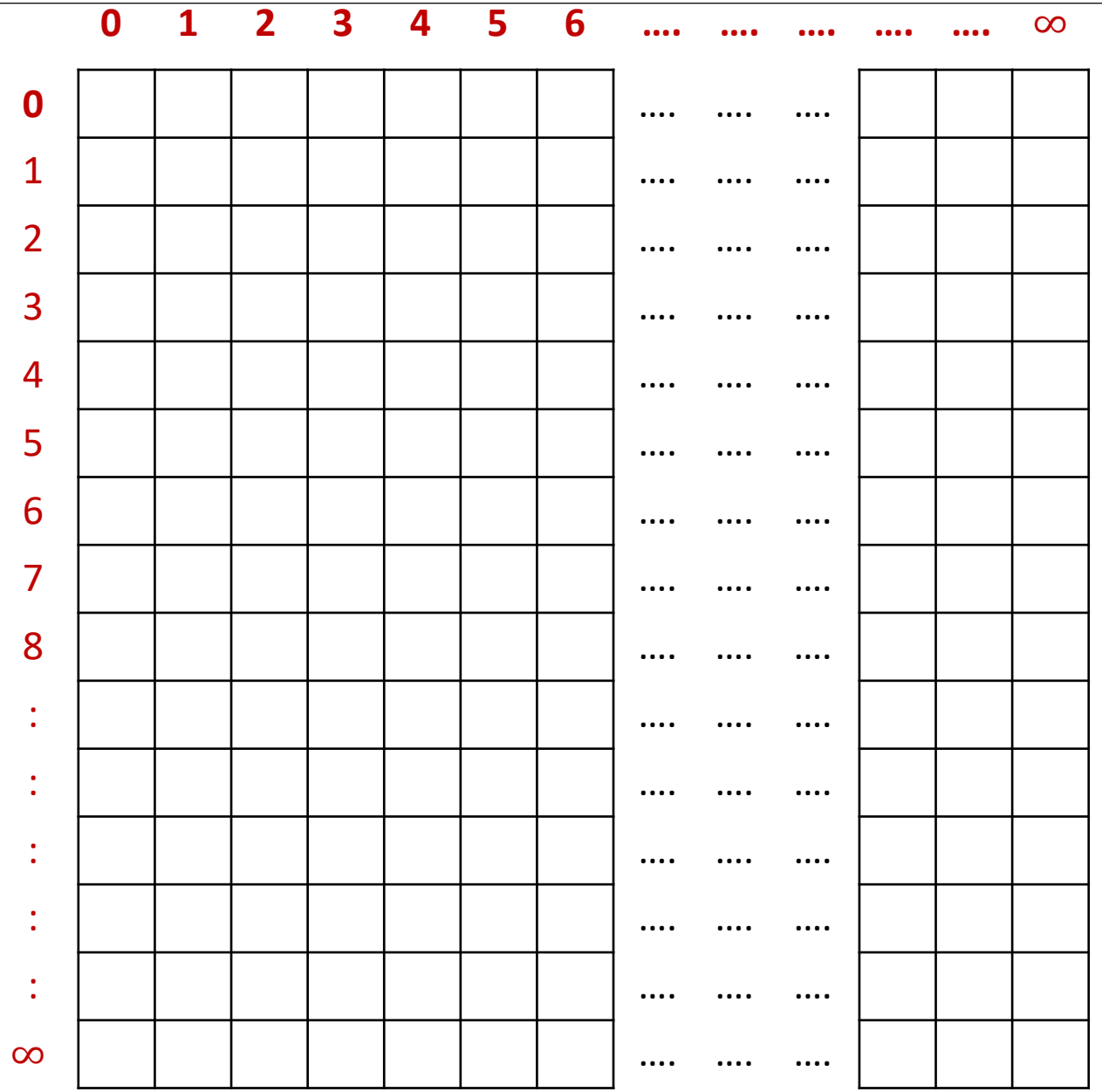
$2^n = 4 \gg n = 2$

Value	Code
a	00
b	01
c	10
d	11

$x = \{a, b, c, d, e, f, g, h\}$

$2^n = 8 \gg n = 3$

Value	Code
a	000
b	001
c	010
d	011
e	100
f	101
g	110
h	111



Any analog signal is an infinite parameter for any digital devices

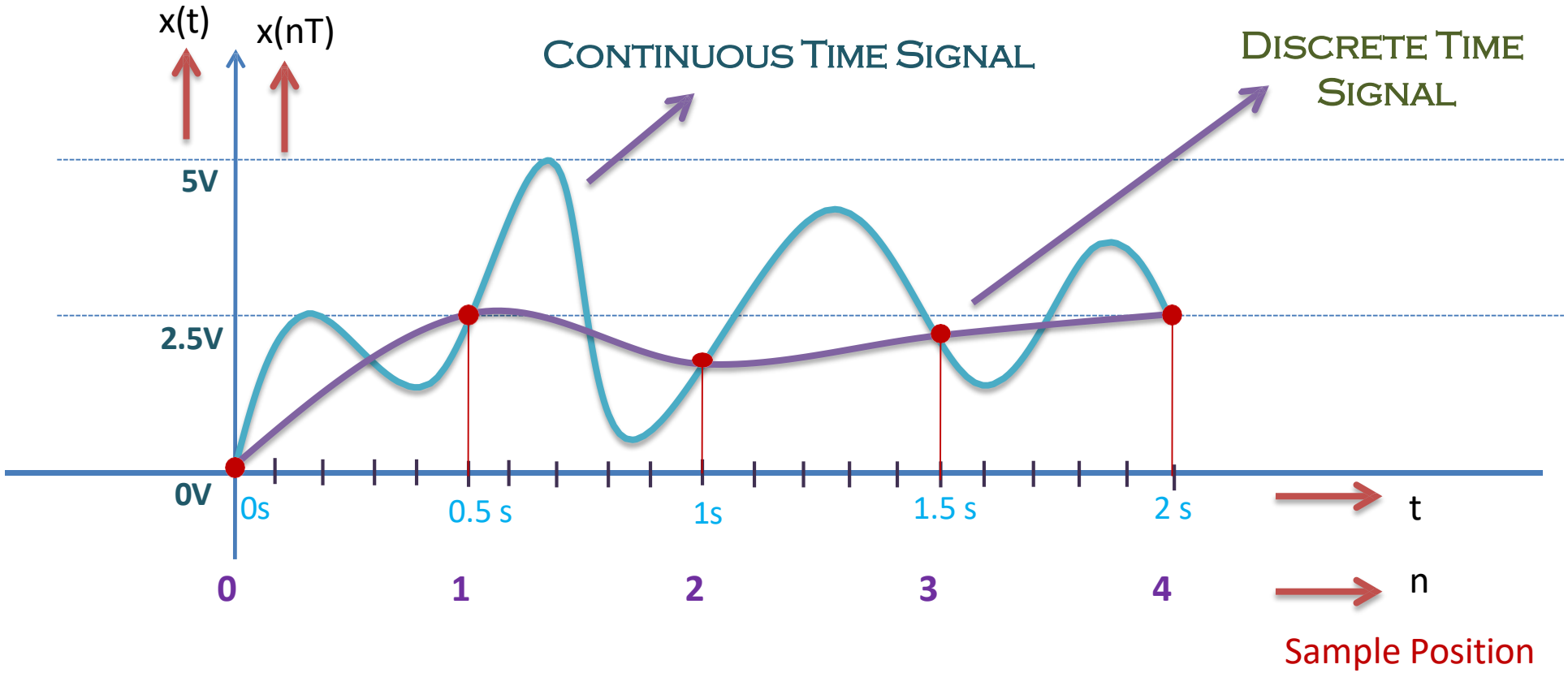
Steps of Converting Analog Signal into Digital

1. **Sampling:** Converts Continuous Time (CT) signal into Discrete Time (DT) Signal.

2. **Quantization:** Converts continuous-valued signal into discrete-valued signal.

3. **Coding:** Converts a discrete value into Binary Code.

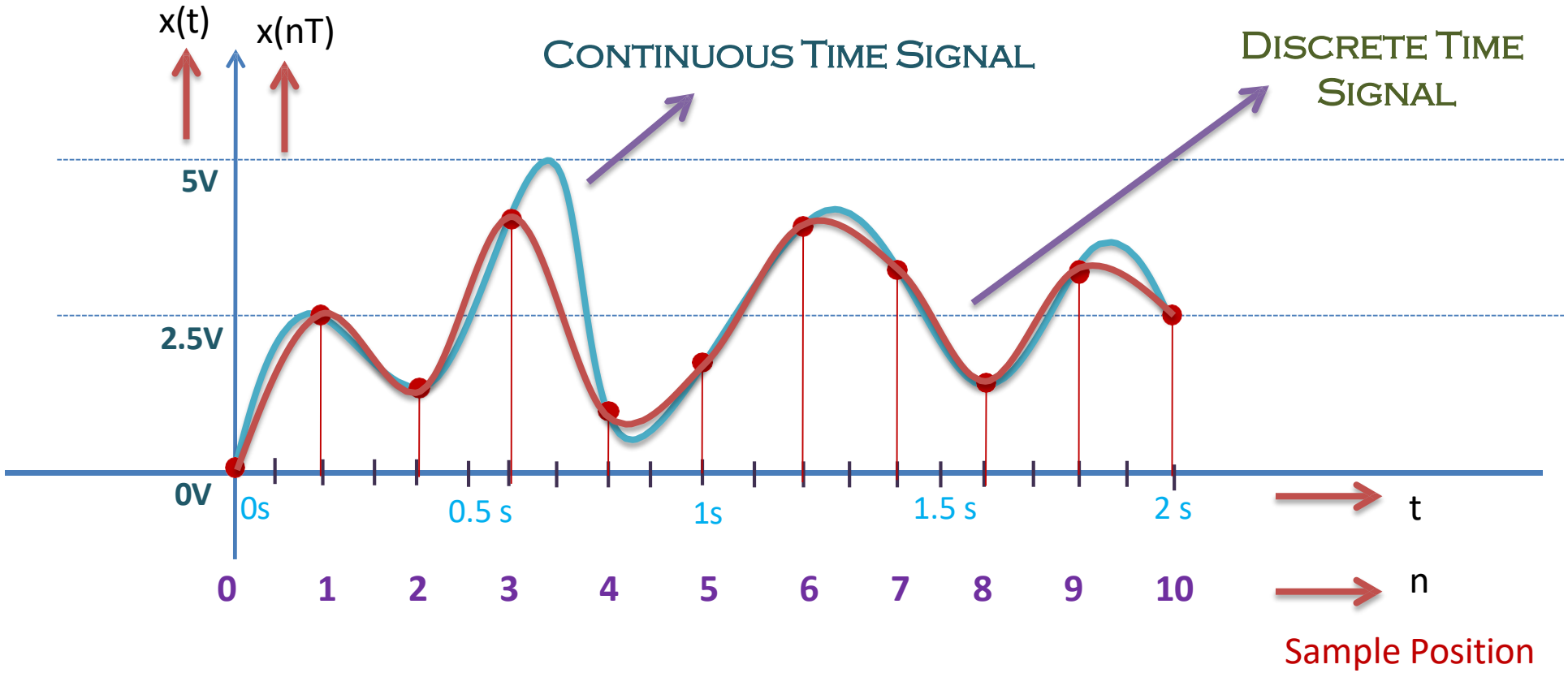
Sampling



Sampling interval (T) = 0.5

Sampling frequency (Fs) = 2 Hz

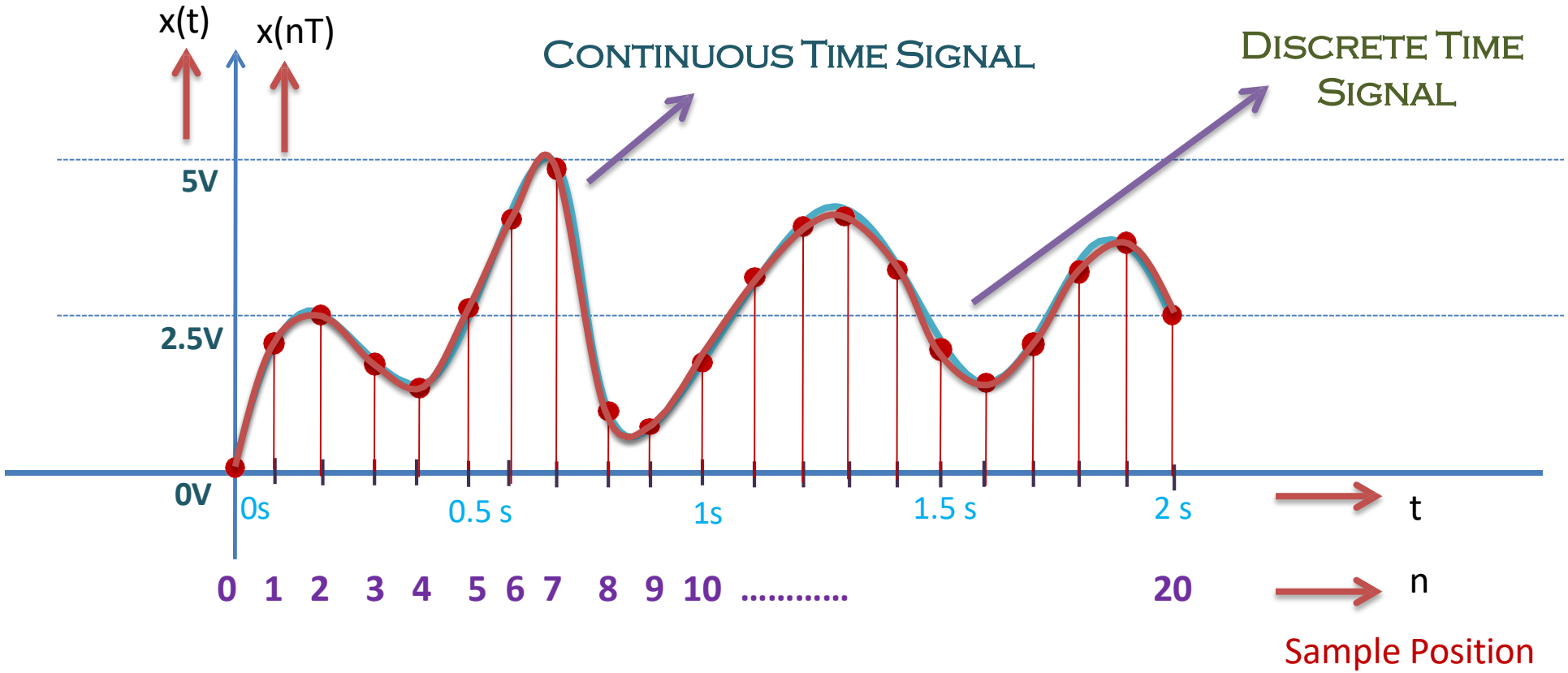
Sampling



Sampling interval (T) = 0.2

Sampling frequency (Fs) = 5 Hz

Sampling



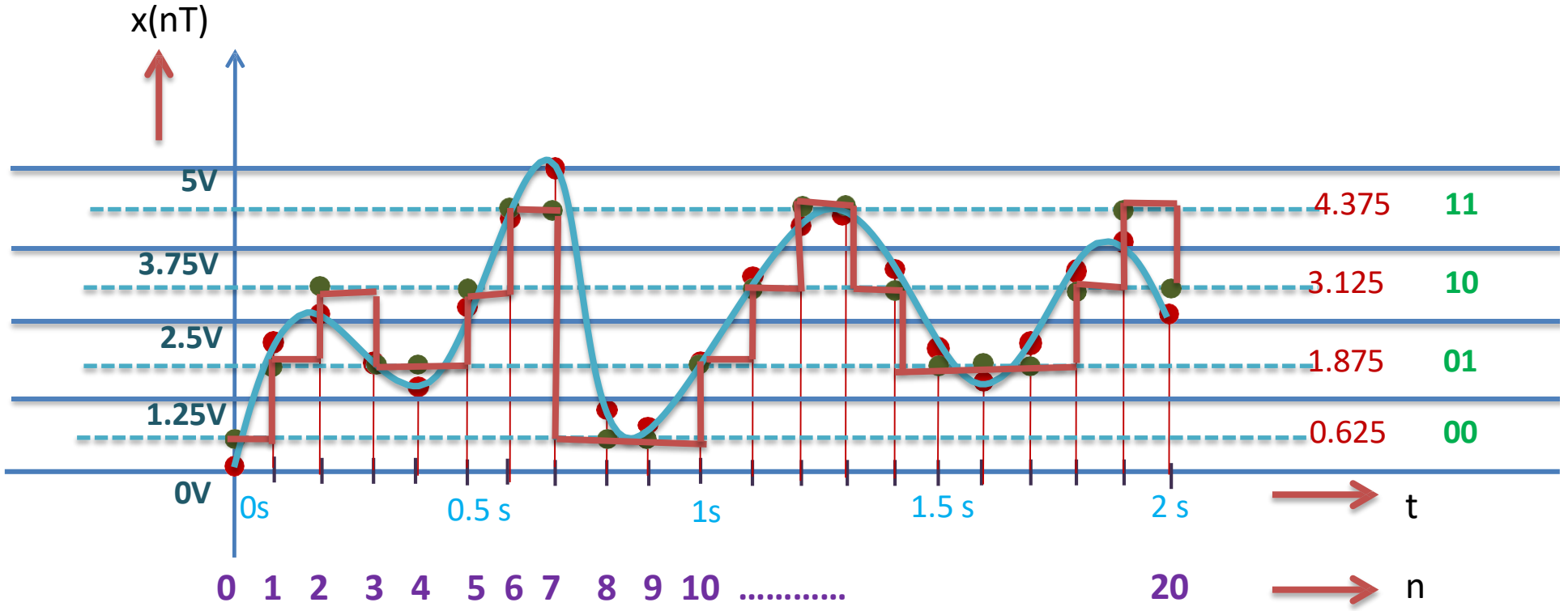
Sampling interval (T) = 0.1

Sampling frequency (Fs) = 10 Hz

Sampling theorem

$$f_s \geq 2W$$

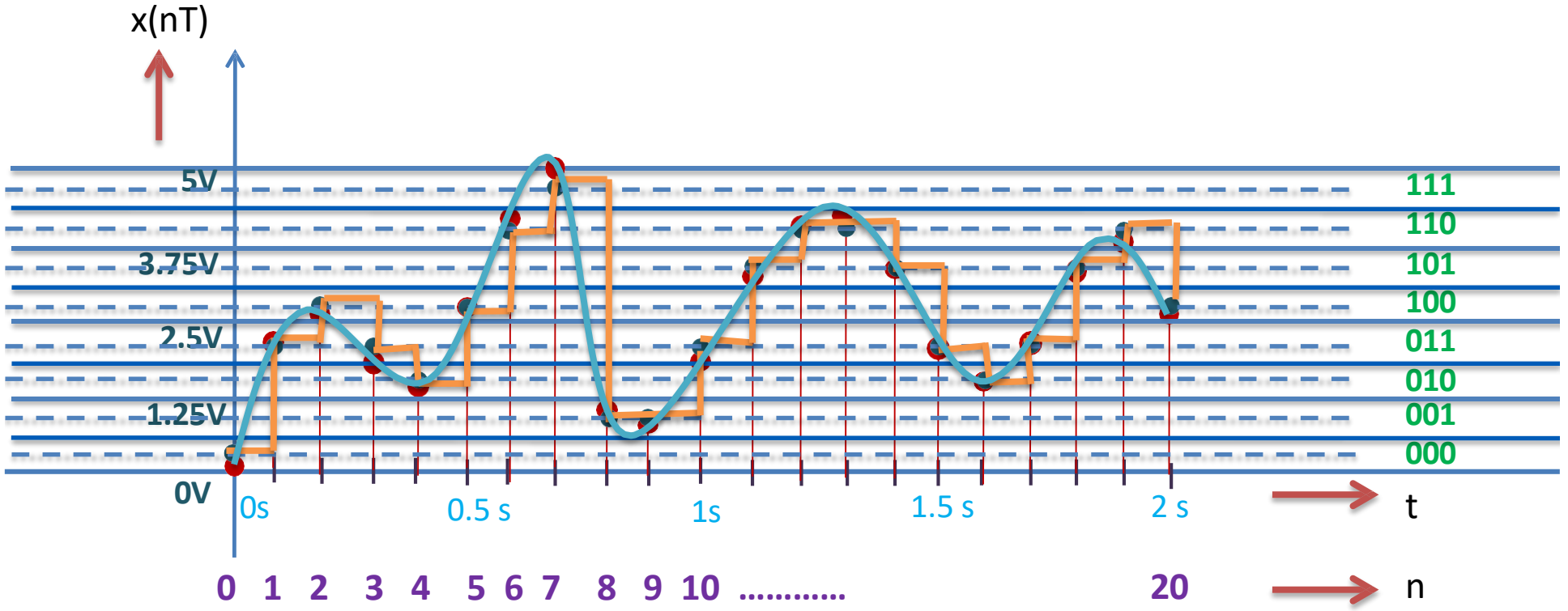
Quantization and Coding



No of Quantization Level = 4
Sample size= 2 bit
Memory Required= 2*20= 40 bit

Sample No	Sample Value	Sample No	Sample Value
0	00	7	11
1	01	8	00
2	10	9	00
3	01	10	01
4	01	.	.
5	10	.	.
6	11	.	.

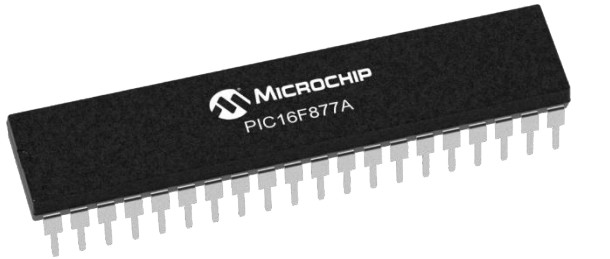
Quantization and Coding

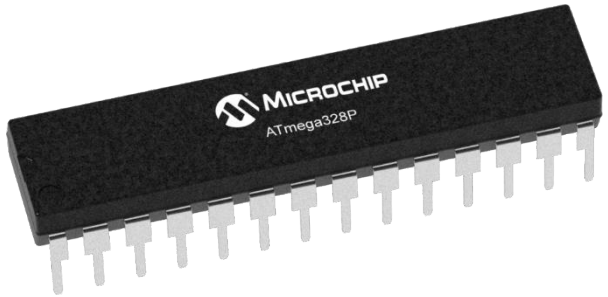


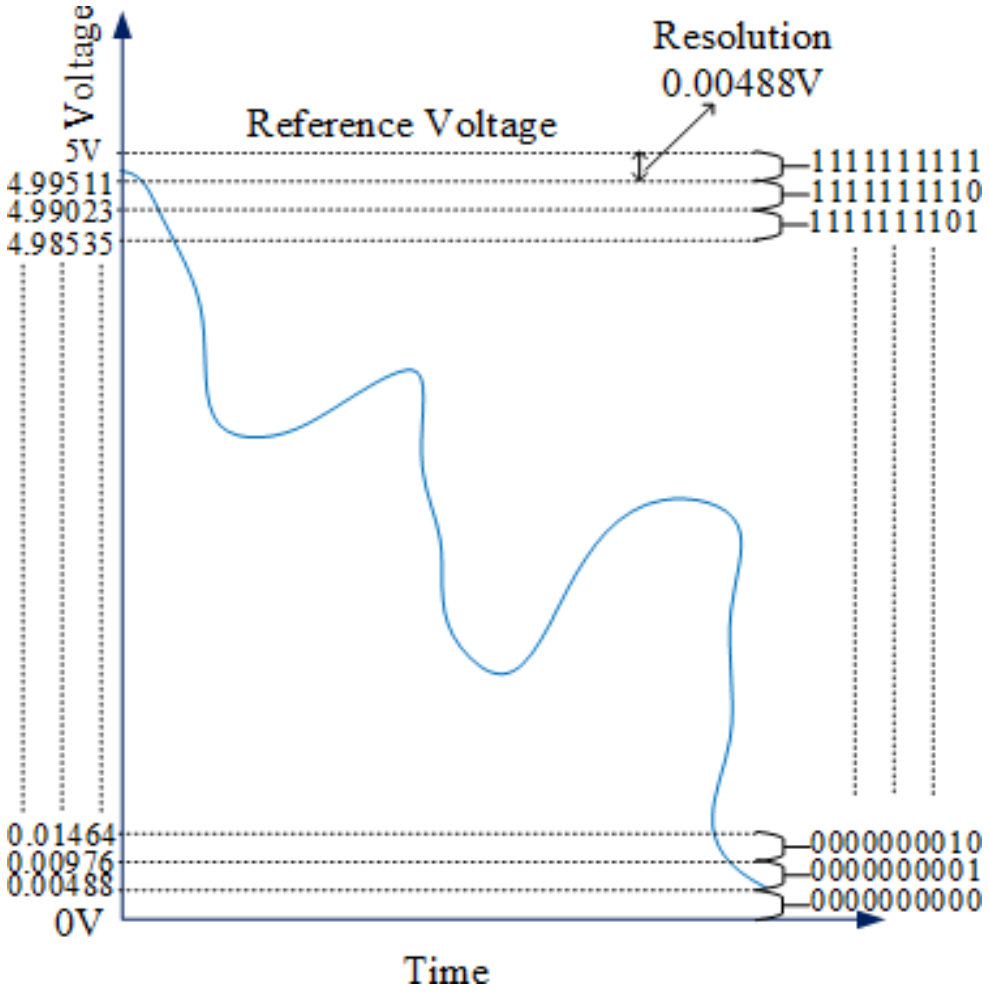
No of Quantization Level = 8
Sample size= 3 bit
Memory Required= 3*20= 60 bit

Sample No	Sample Value	Sample No	Sample Value
0	000	7	111
1	011	8	001
2	100	9	001
3	011	10	011
4	010	.	.
5	100	.	.
6	110	.	.

A/D Converter in Microcontroller







Program for converting analog signal to digital signal

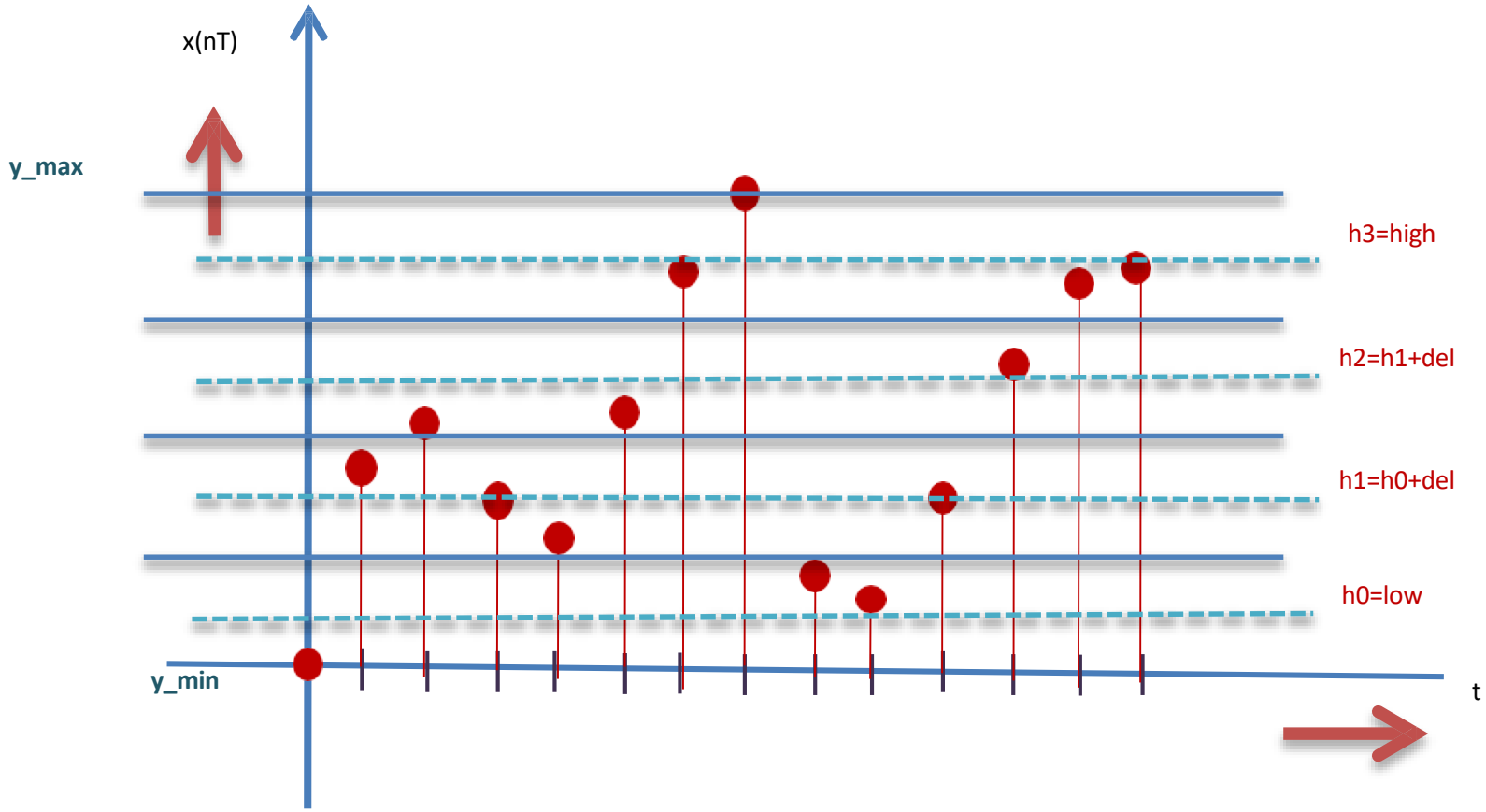
Code:

```
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
q_error=q_out-y; r_memory=bit*ns;
```

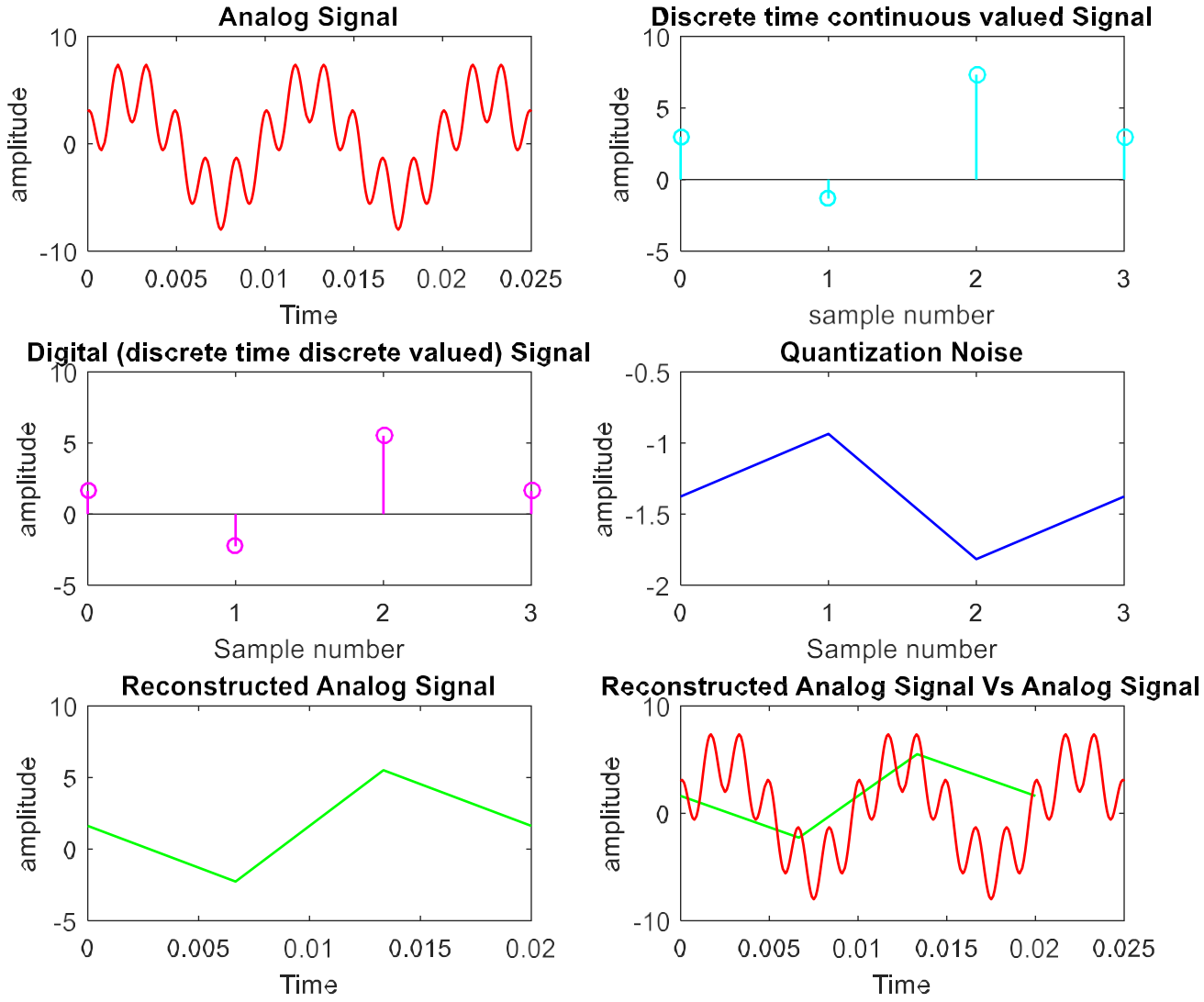
Program for converting analog signal to digital signal (Cont.)

```
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('Digital (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); hold on;
```

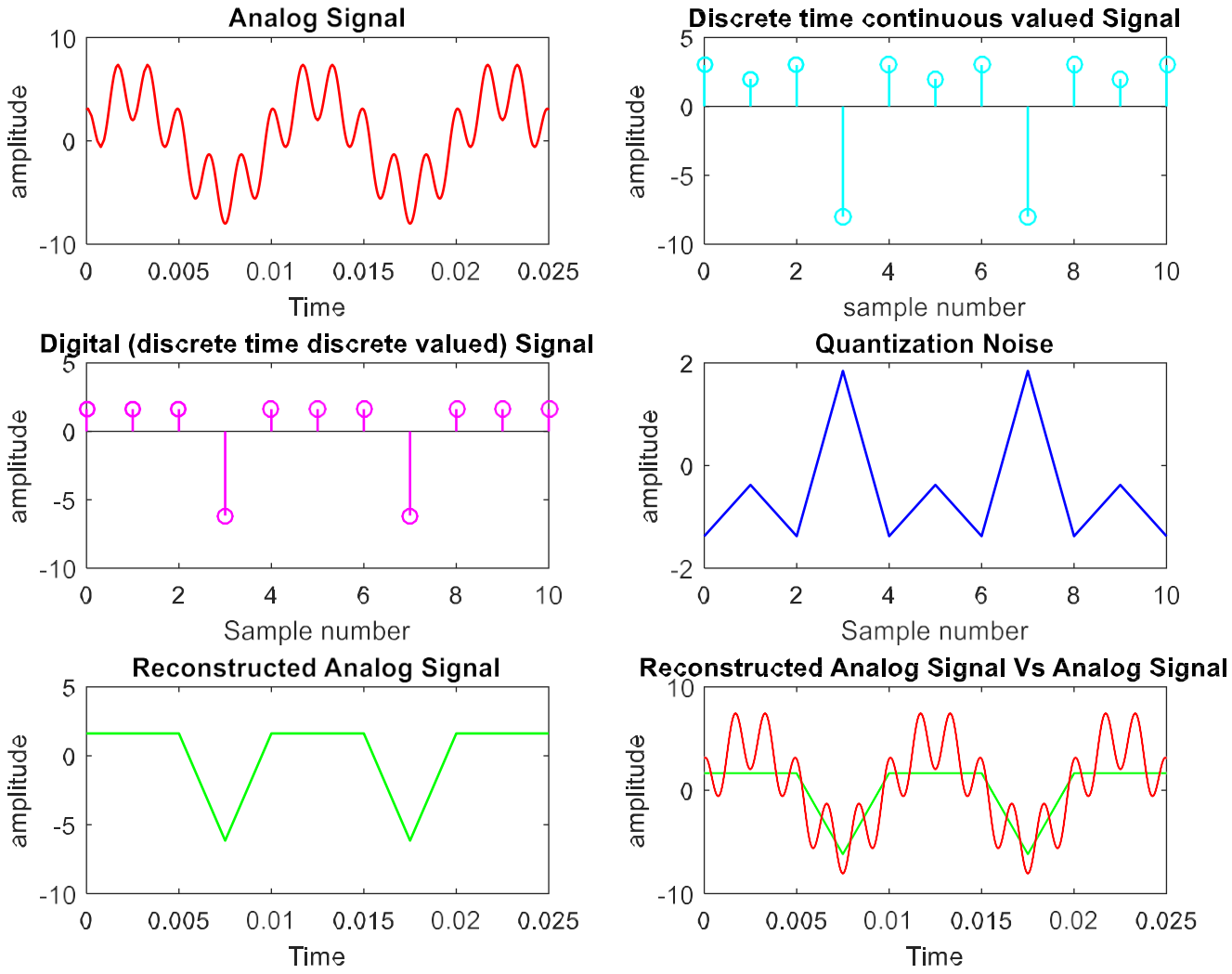
Program for converting analog signal to digital signal (Cont.)



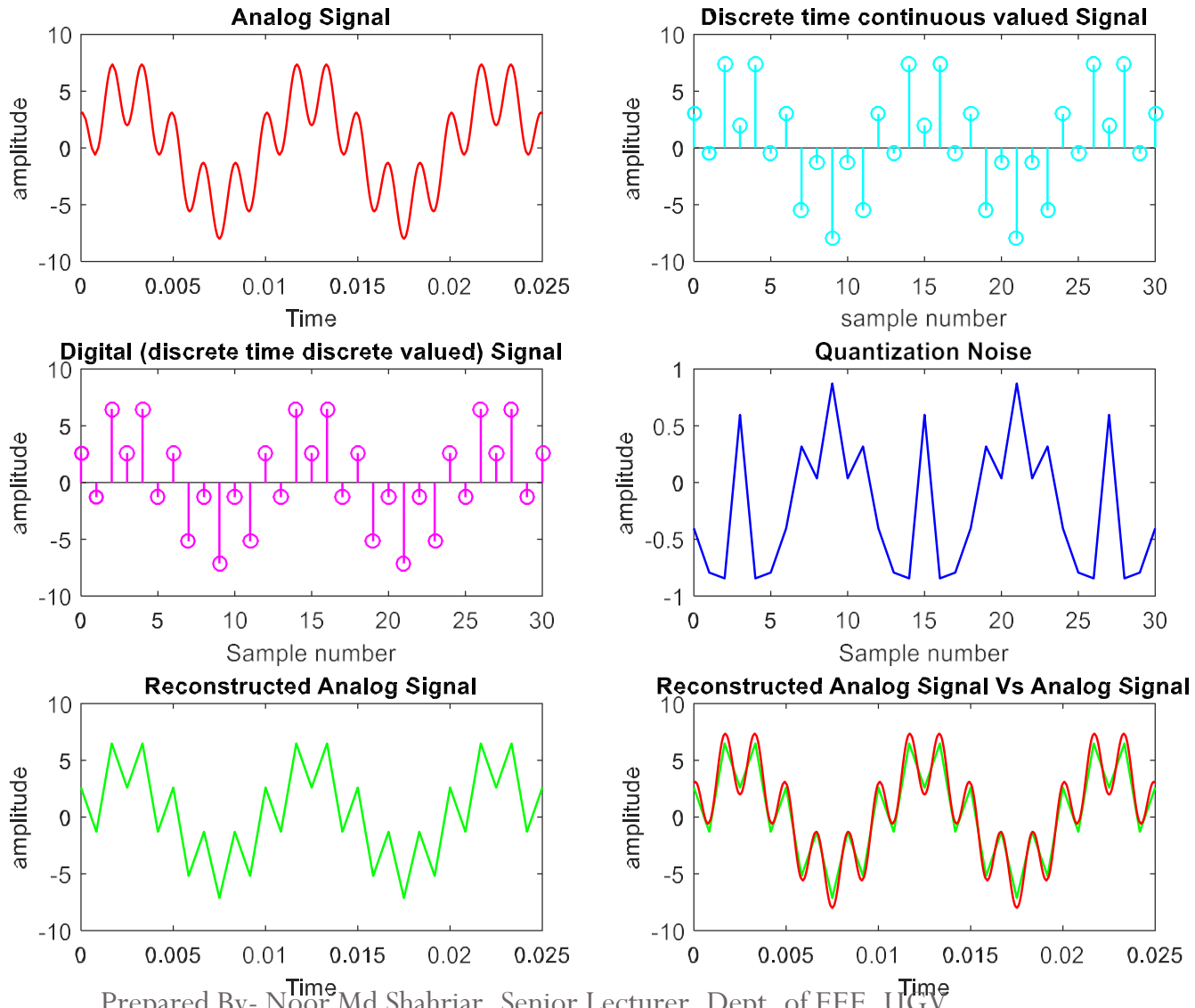
Output: Low sampling rate=150 Hz (< Nyquist rate) and low resolution (bit=2)



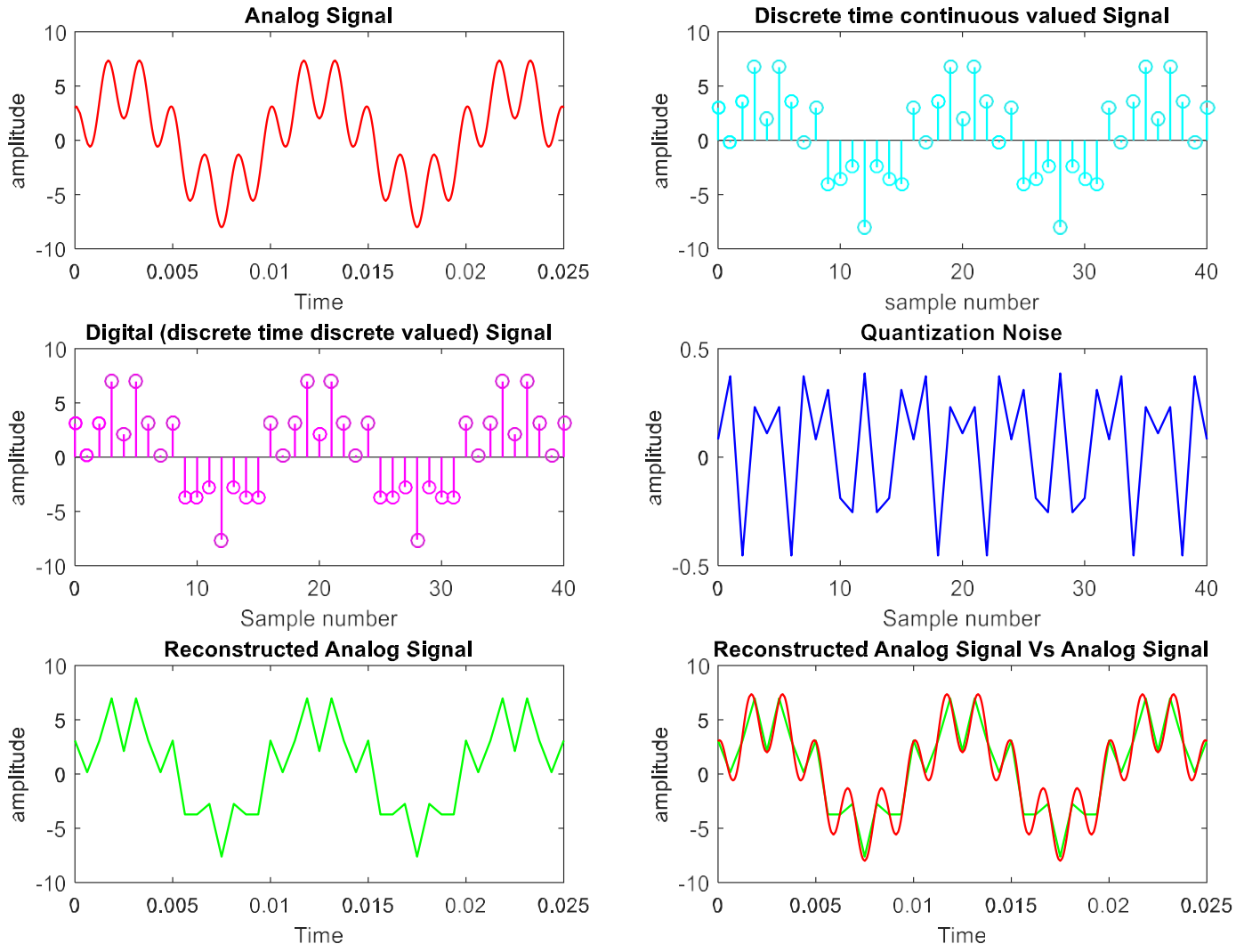
Output: Low sampling rate 400 Hz (< Nyquist rate) and low resolution (bit=2)



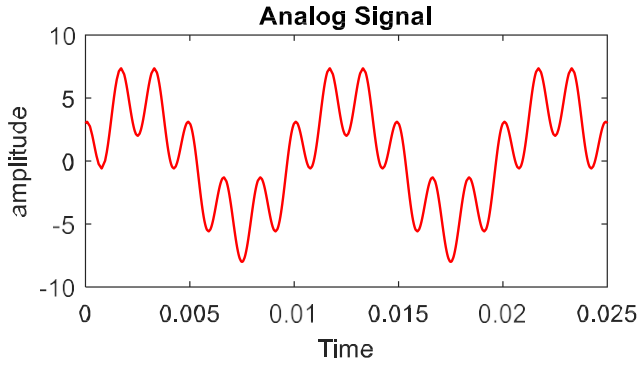
Output: Low sampling rate 1200 Hz (= Nyquist rate) and low resolution (bit=3)

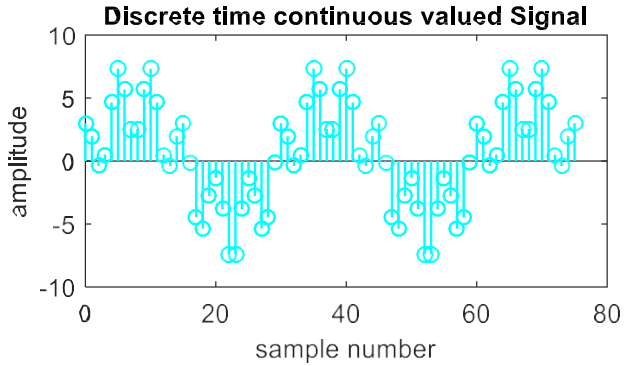


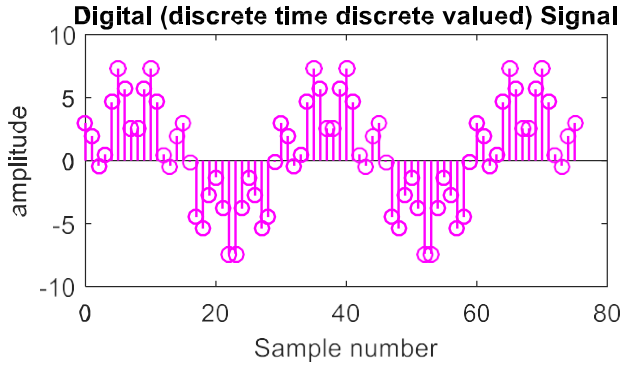
Output: Sampling rate (1600 Hz) and Low Resolution (bit=4)

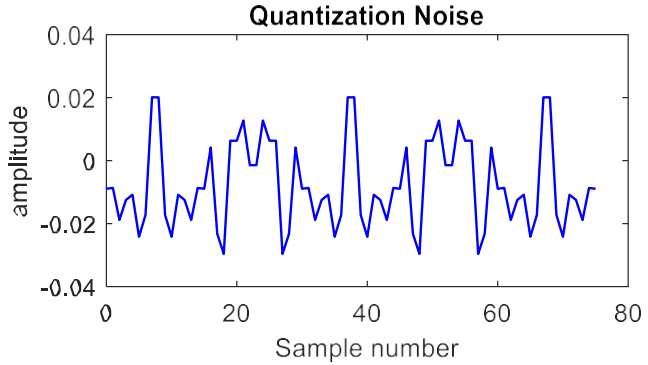


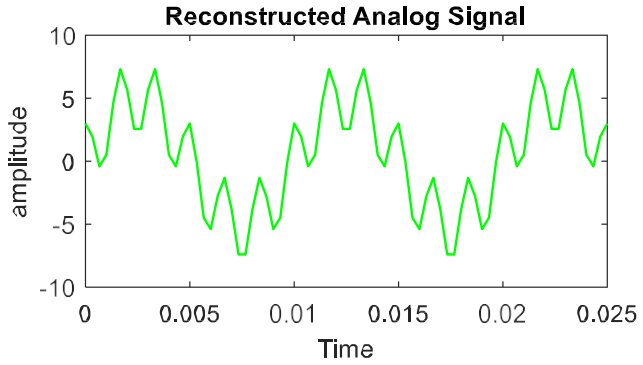
Output: Sampling rate (3000 Hz) and Resolution (bit=8)

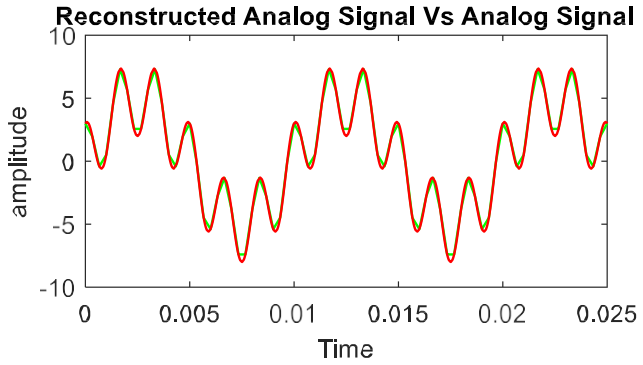




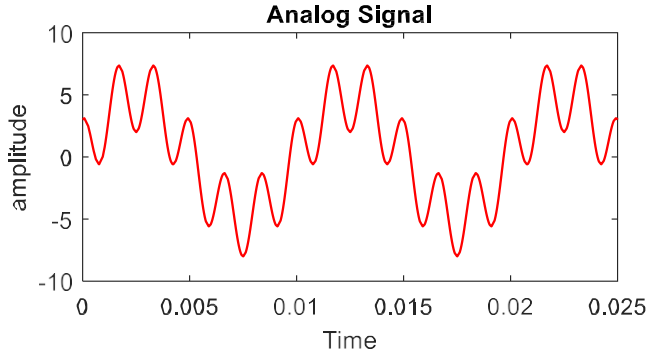


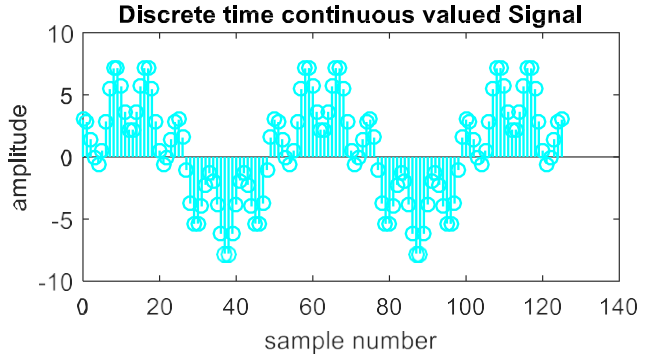


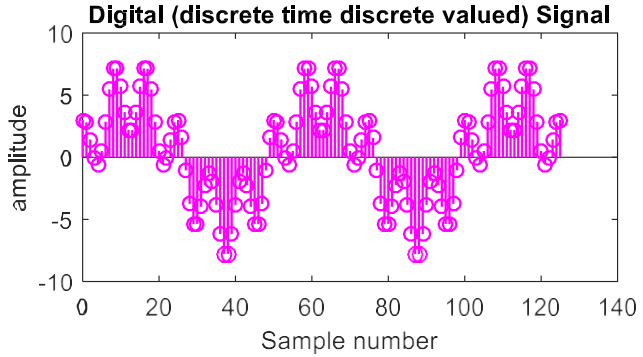


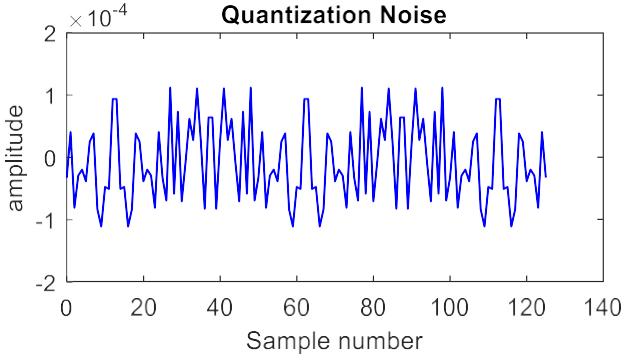


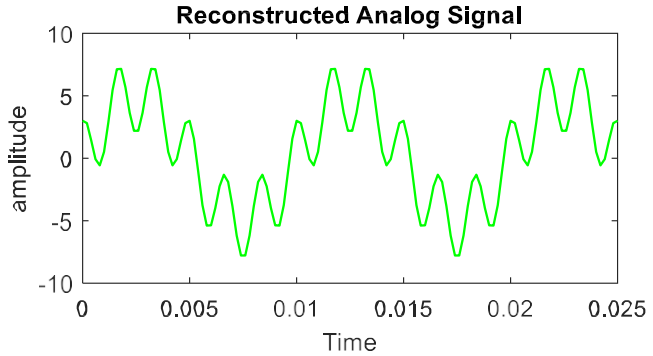
Output: Sampling rate (5000 Hz) and Resolution (bit=16)

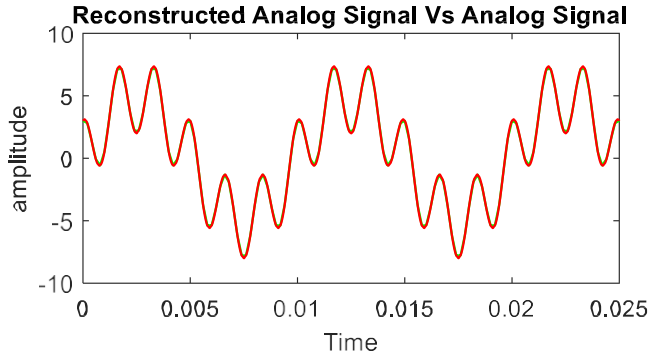












University of Global Village (UGV), Barishal
Dept. of Electrical and Electronic Engineering

Experiment-4
Manipulation of Discrete-Time (DT) signals

Course Code: EEE-307/CSE-309
Course Title: Digital Signal Processing Sessional

Prepared By
Noor Md Shahriar
Senior Lecturer, Dept. of EEE, UGV

Objectives

a) To develop MATLAB program to add/subtract/multiply two discrete time sequence.

b)To develop function to shift and fold DT sequence.

c)To develop the function to find the symmetric (even) and anti-symmetric (odd) part of DT signal.

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;
```

Command Window

```
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

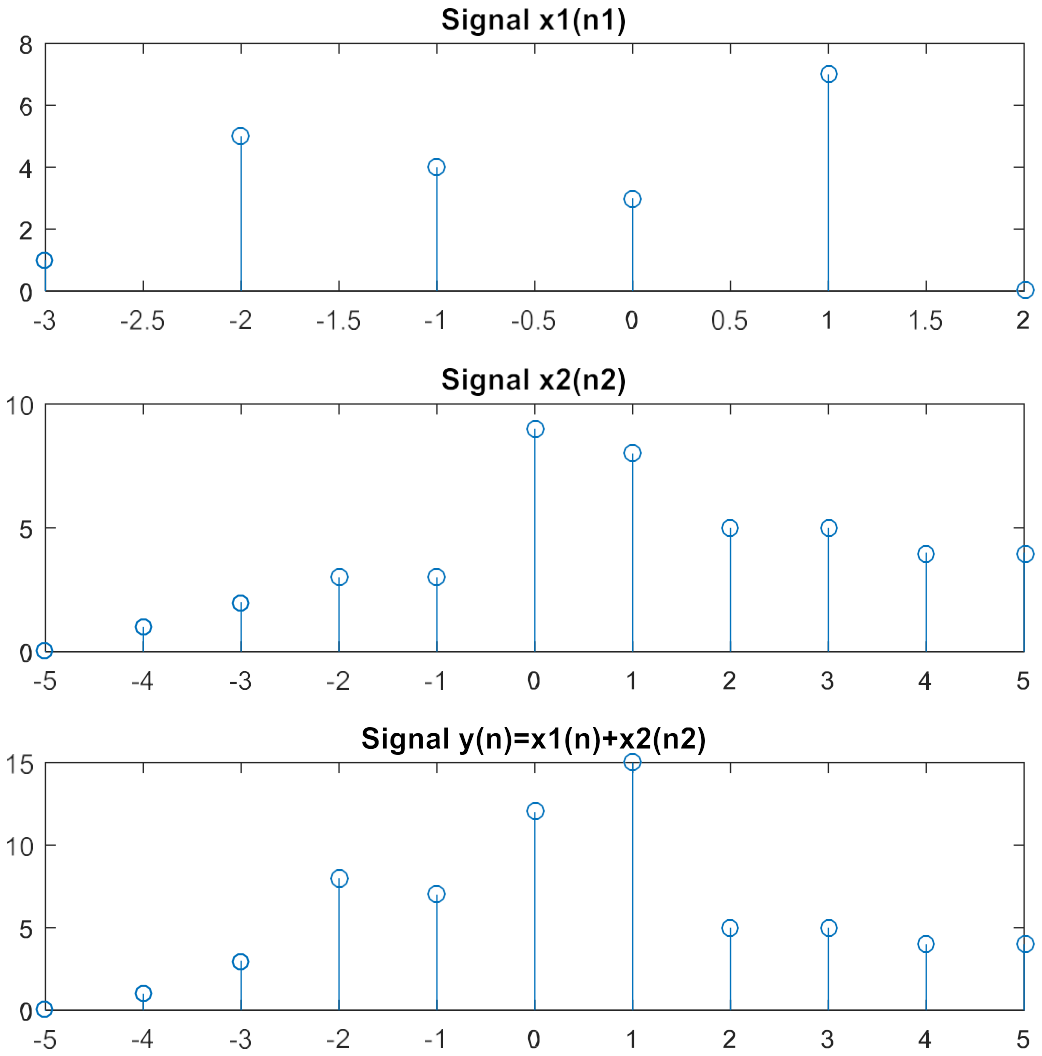
Function

```
function [y,n]=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;
```

Command Window

```
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]=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)');
```

Addition: Output



Shift

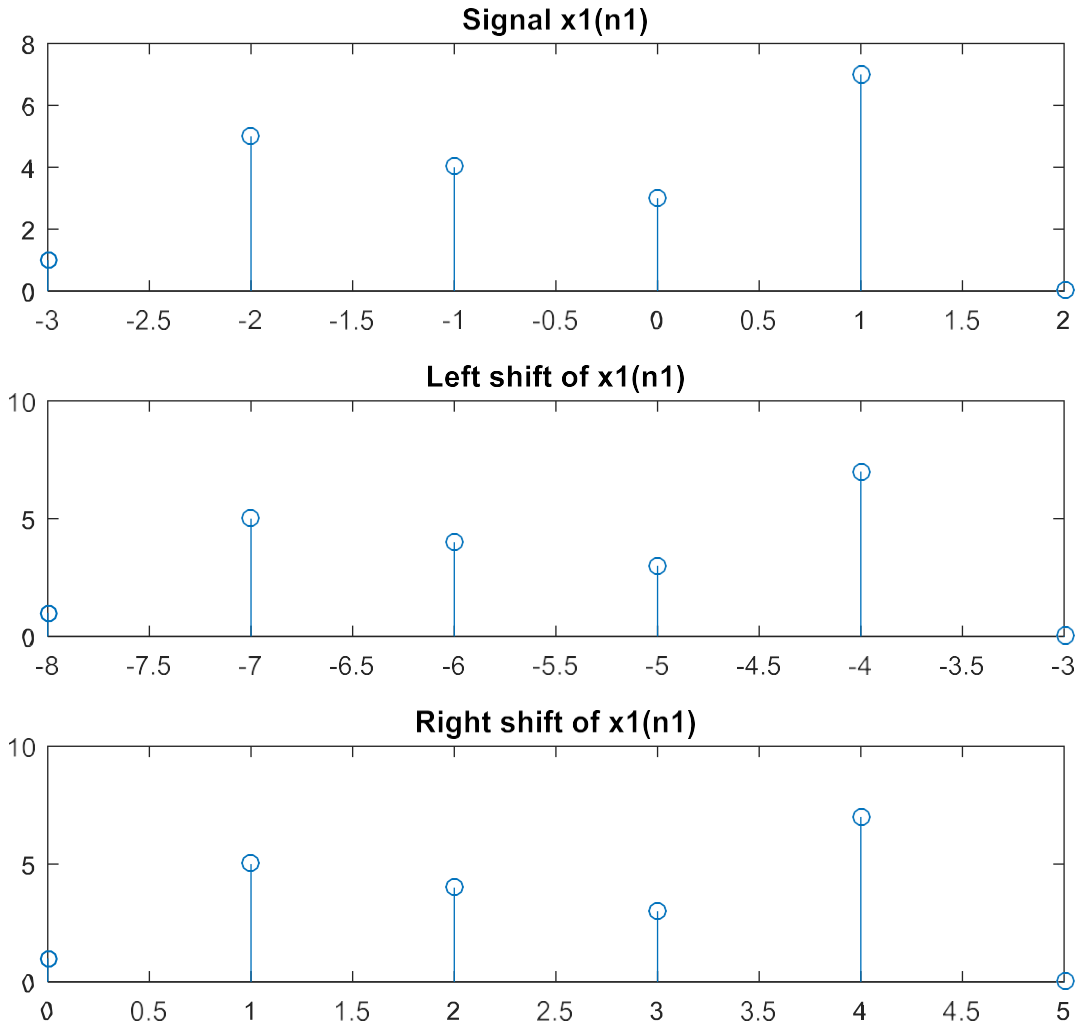
Function

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

Command Window

```
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)');
```

Shift: Output



Folding

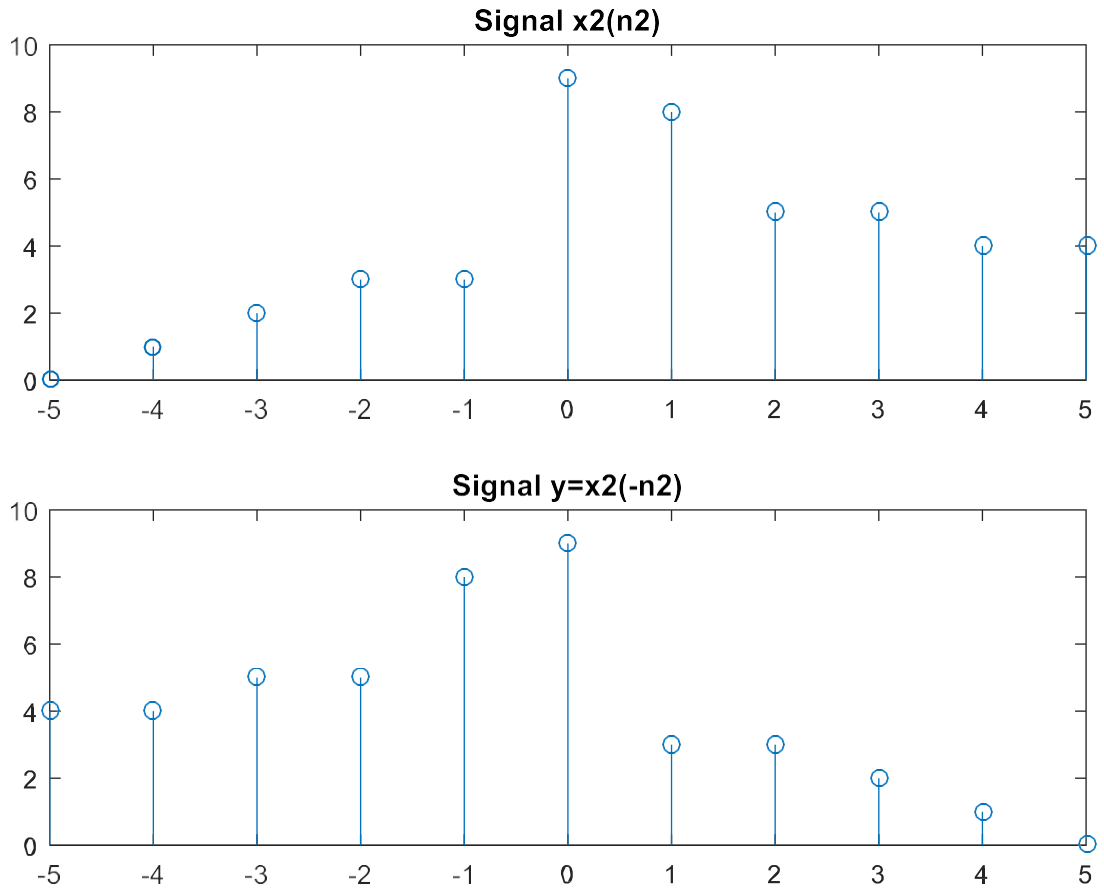
Function

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

Command Window

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

Folding: Output



Finding even and odd part of DT signals

Many signal are neither even nor odd. Any arbitrary signal can be expressed as the sum of two signal components, one of which is even and the other odd.

The even signal components is expressed as

$$x_e(n) = \frac{1}{2}[x(n) + x(-n)]$$

The odd signal components is expressed as

$$x_o(n) = \frac{1}{2}[x(n) - x(-n)]$$

The signal x(n) is expressed as-

$$x(n) = x_e(n) + x_o(n)$$

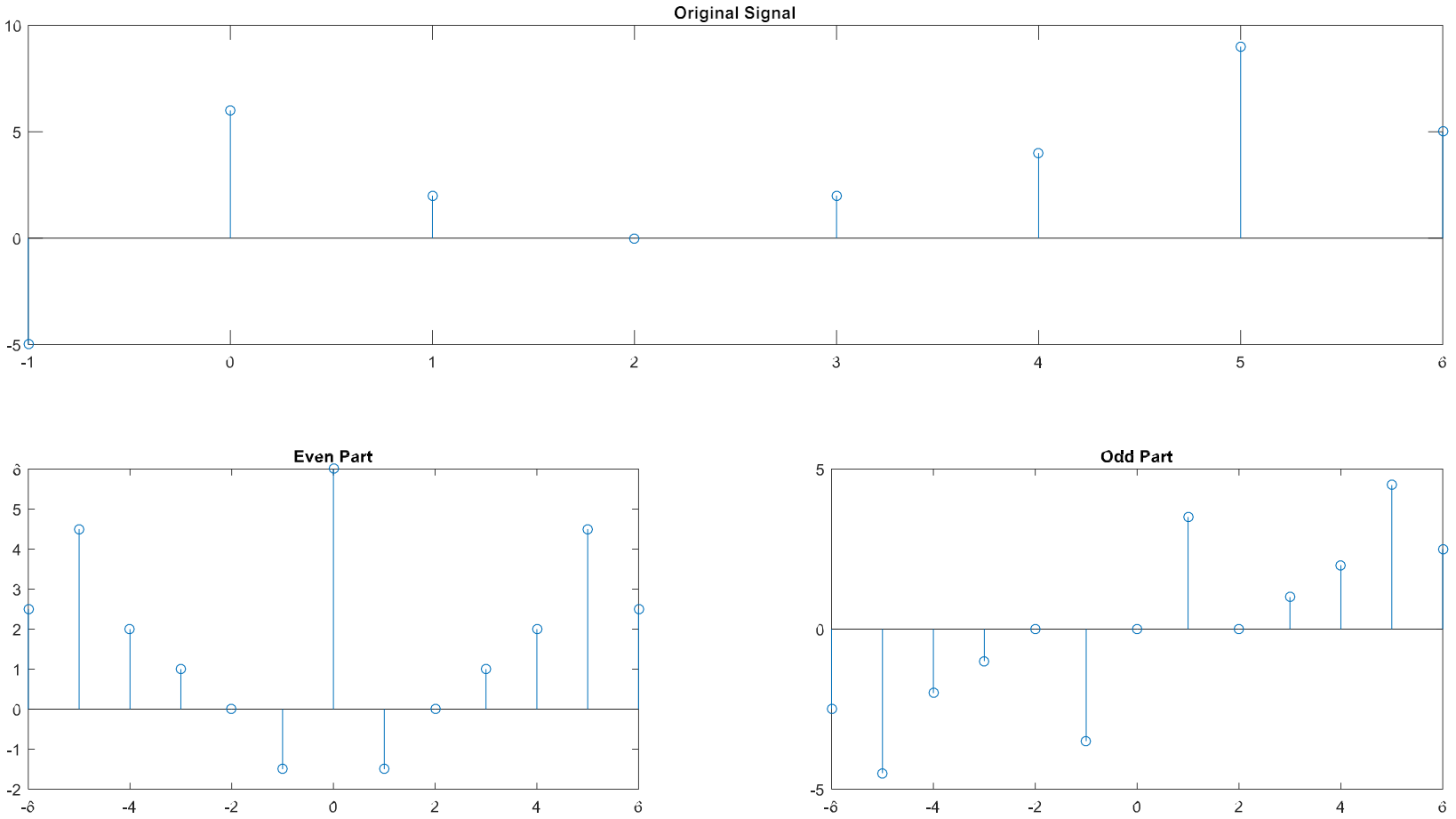
Function

```
function [xe,ne,xo,no]=even_odd(x,n)
[X,N]=sig_fold(x,n);
[xe,ne]=sig_add(x,n,X,N);
xe=0.5*xe;
[xo,no]=sig_sub(x,n,X,N);
xo=0.5*xo;
```

Command Window

```
clc
clear
x=input('x(n):');
n1=input('Starting point of x(n):');
n=n1:n1+length(x)-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 Part');
subplot(2,2,4);
stem(no,xo);
title('Odd Part');
```

Output


Original Signal
10
5
0
-5
-1
0
1
2
3
4
5
6
Even Part
6
5
4
3
2
1
0
-1
-2
-6
-4
-2
0
2
4
6
Odd Part
5
0
-5
-6
-4
-2
0
2
4
6

University of Global Village (UGV), Barishal
Dept. of Electrical and Electronic Engineering

Experiment-5
Convolution and correlation of Discrete-Time sequences

Course Code: EEE-307/CSE-309
Course Title: Digital Signal Processing Sessional

Prepared By
Noor Md Shahriar
Senior Lecturer, Dept. of EEE, UGV

Objectives

a)To develop MATLAB program to find the convolution sum of two discrete-time sequence.

b) To develop program to calculate the correlation of two DT sequence.

c)Use of MATLAB built-in function to find the convolution and correlation sequence of two DT sequence.

d) To observe the application of correlation in active sonar application.

The Convolution Sum

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

The above equation that gives the response y(n) of the LTI system as a function of the input signal x(n) and unit sample (impulse) response h(n) is called a convolutional sum.

The convolution sum is used to compute the output of a LTI system for a given input x[n] and impulse response h[n].

Convolution: Easy Method

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

$$h(n) = \{1,\underset{\uparrow}{2},1, -1\}$$

1	2	1	-1
2	4	2	-2
3	6	3	-3
1	2	1	-1

y(-1)=1
y(0)=2+2=4
y(1)=3+4+1=8
y(2)=1+6+2-1=8
y(3)=2+3-2=3
y(4)=1-3=-2
y(5)= -1

$y(n)=x(n)* h(n)= \{1,\underset{\uparrow}{4},8,8,3,-2,-1\}$

Convolution: Developing Algorithm

$$x(n) = \{x_1, x_2, x_3, x_4\}$$

$$h(n) = \{h_1, h_2, h_3, h_4\}$$

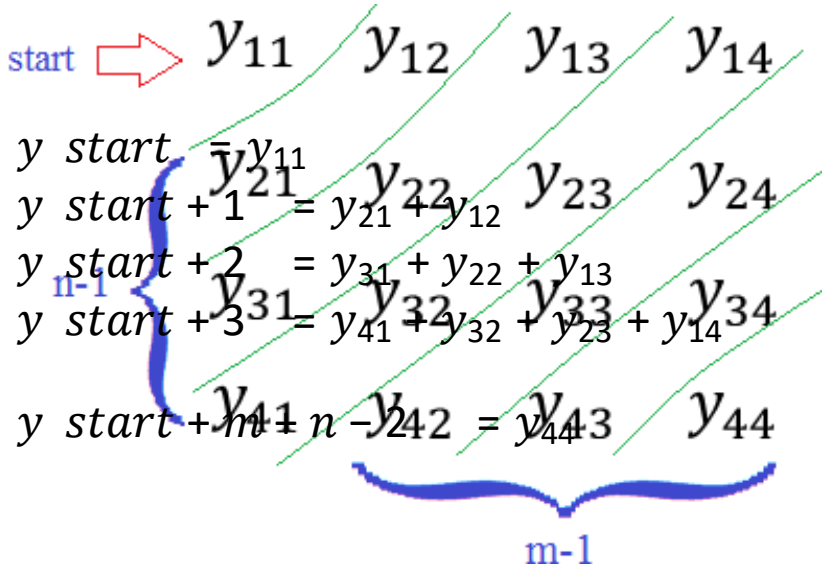
Formation of Matrix by x(n) and h(n)

$$\begin{matrix} y_{11} & y_{12} & y_{13} & y_{14} \\ y_{21} & y_{22} & y_{23} & y_{24} \\ y_{31} & y_{32} & y_{33} & y_{34} \\ y_{41} & y_{42} & y_{43} & y_{44} \end{matrix}$$

Where,

$$\begin{matrix} y_{11} = x_1h_1 & y_{12} = x_1h_2 & y_{13} = x_1h_3 & y_{14} = x_1h_4 \\ y_{21} = x_2h_1 & y_{22} = x_2h_2 & y_{23} = x_2h_3 & y_{24} = x_2h_4 \\ y_{31} = x_3h_1 & y_{32} = x_3h_2 & y_{33} = x_3h_3 & y_{34} = x_3h_4 \\ y_{41} = x_4h_1 & y_{42} = x_4h_2 & y_{43} = x_4h_3 & y_{44} = x_4h_4 \end{matrix}$$

Convolution: Developing Algorithm



$y\ start = y_{11}$

$y\ start + 1 = y_{21} + y_{12}$

$y\ start + 2 = y_{31} + y_{22} + y_{13}$

$y\ start + 3 = y_{41} + y_{32} + y_{23} + y_{14}$

$y\ start + m + n - 2 = y_{44}$

( )

( )

( )

( )

( )

start= min (min(n),min(m))

N=start+m+n-2

for NN=1 to N

  for i= 1 to NN

    j= NN-i+1

    $Y(NN) = Y(NN) + y(i, j)$

Convolution: User-defined function

Function

```
function [Y,N]=convf(x,n,h,m)
start=min(min(n),min(m));
N=start:(start+length(n)+length(m)-2);
for i=1:length(m)
   YY(:,i)=x*h(i);
end
Y=zeros(1,length(N));
for NN=1:length(N)
   for i=1:NN
      j=NN-i+1;
      if(i<=length(n) && j<=length(m))
         Y(NN)=Y(NN)+YY(i,j);
      end
   end
end
```

Instructions at Command Window

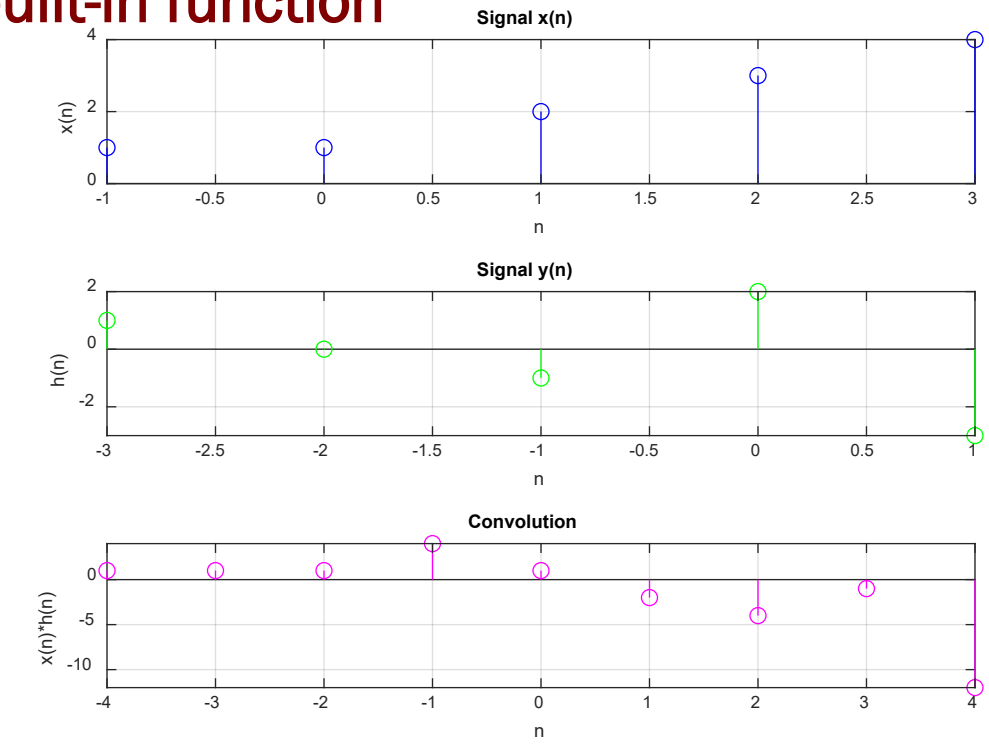
```
>> n=[0 1 2 3];
>> x=[1 2 3 1];
>> m=[-1 0 1 2];
>> h=[1 2 1 -1];
>> [Y,N]=convf(x,n,h,m);
>> subplot(311);
>> stem(n,x);
>> title('The input signal x(n)');
>> subplot(312);
>> stem(m,h);
>> title('Impulse response of system');
>> subplot(313);
>> stem(N,Y);
>> title('Output of the System');
```

Convolution: Output

Convolution: Using Matlab built-in function

Code:

```
clc; clear;
x=input('Input Sequence x(n):');
nl= input('Input Starting Point of x(n):');
h=input('Input Sequence h(n):');
ml= input('Input Starting Point h(n):');
n=nl:nl+length(x)-1;
m=ml:ml+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)');
```



Example (in Workspace):

Input Sequence x(n):[1 1 2 3 4]
Input Starting Point of x(n):-1
Input Sequence h(n):[1 0 -1 2 -3]
Input Starting Point h(n):-3
convolution of sequence x(n) & h(n):
1 1 1 4 1 -2 -4 -1 -12

Correlation of DT signals

- A mathematical operation that closely resembles convolution is correlation. In convolution the input and impulse response are involved whereas in correlation two signal sequences are involved.
- The correlation between the two signals is to measure the degree to which the two signals are similar and thus to extract some information that depends to a large extent on the application.
- Correlation of signals is often uncounted in radar, sonar, digital communications, geology and other areas in science and engineering.

$$r_{xy}(l) = \sum_{n=-\infty}^{\infty} x(n)y(n-l), \qquad l = 0, \pm 1, \pm 2, \dots \dots$$

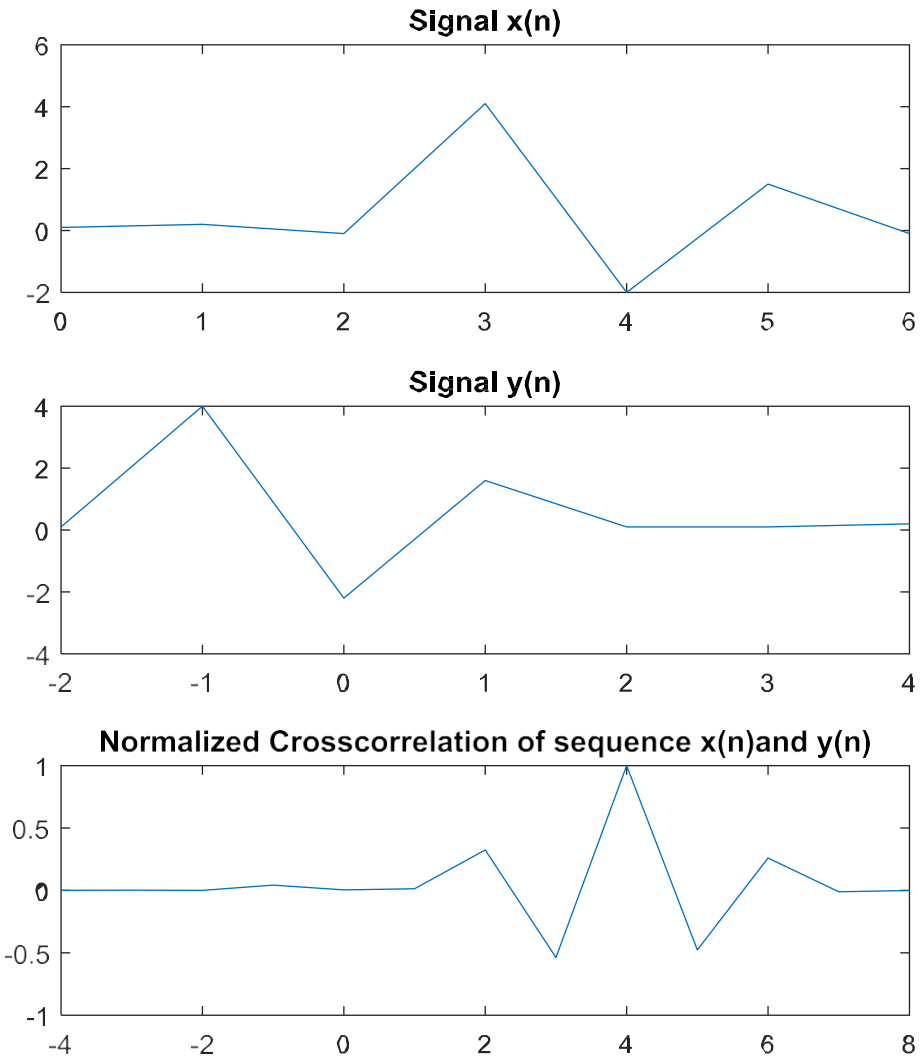
$$r_{xy}(l) = x(l) * y(-l)$$

Correlation: User-defined function

```
function [Y,N]=corlnf(x,n,h,m)
[r,c]=size(h);
if c==1
   h=flipud(h);
else
   h=fliplr(h);
end
[r,c]=size(m);
if c==1
   m=-flipud(m);
else
   m=-fliplr(m);
end
start=min(min(n),min(m));
N=start:(start+length(n)+length(m)-2);
for i=1:length(m)
  YY(:,i)=x*h(i);
end
Y=zeros(1,length(N));
for NN=1:length(N)
   for i=1:NN
      j=NN-i+1;
      if(i<=length(n) && j<=length(m))
         Y(NN)=Y(NN)+YY(i,j);
      end
   end
end
```

Correlation: Output

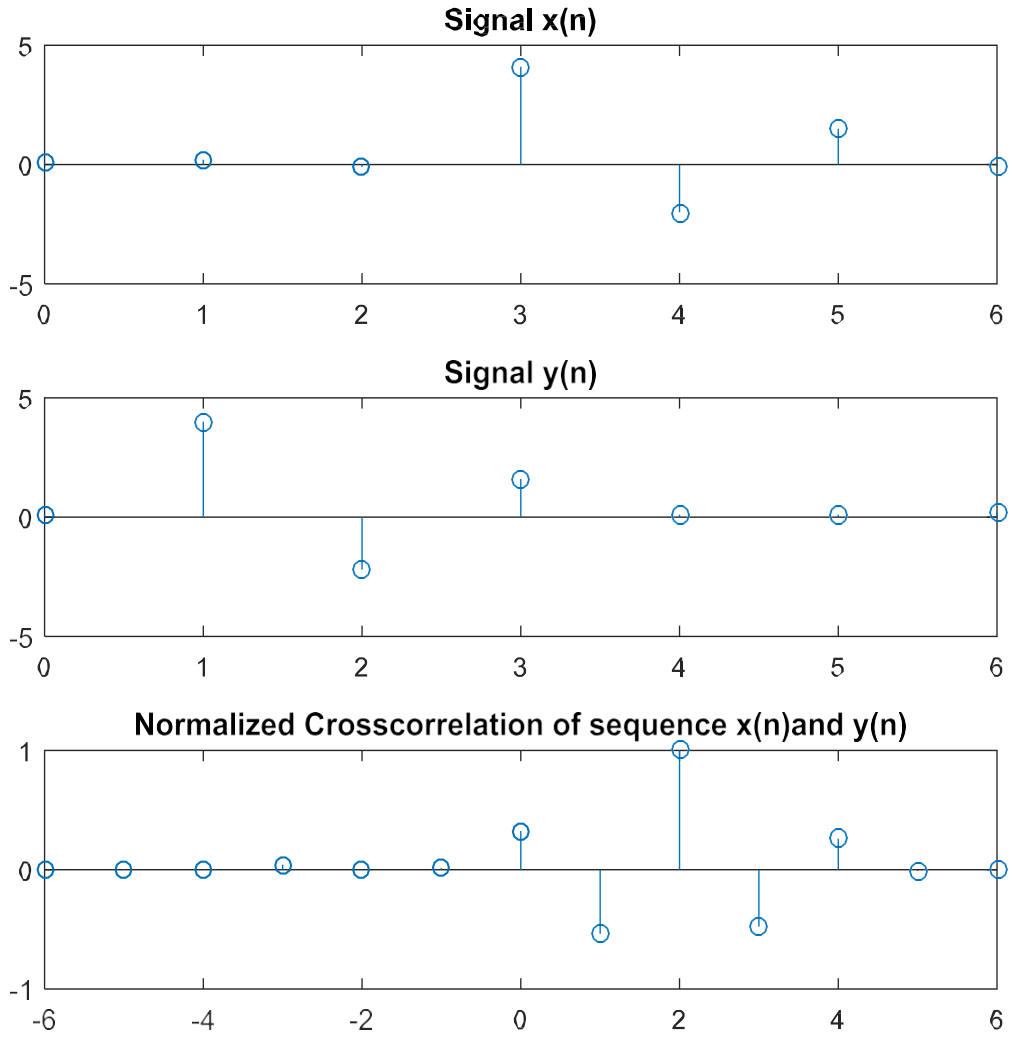
```
>> n=0:6;
>> x=[0.1 0.2 -0.1 4.1 -2 1.5 -0.1];
>> m=-2:4;
>> y=[0.1 4 -2.2 1.6 0.1 0.1 0.2];
>> [Y,N]=corlnf(x,n,y,m);
>> subplot(311);
>> plot(n,x);
>> title('Signal x(n)');
>> subplot(312);
>> plot(m,y);
>> title('Signal y(n)');
>> subplot(313);
>> plot(N,Y/max(Y));
>> title('Normalized Crosscorrelation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



Maximum similarities is obtained when y(n) shifted 4 positions (correlation at lag of 4)

Correlation: Using Matlab built-in function

```
clc;
clear
x=input('Input Sequence x(n):');
nl= input('Input Starting Point of x(n):');
h=input('Input Sequence h(n):');
ml= input('Input Starting Point h(n):');
n=nl:nl+length(x)-1;
m=ml:ml+length(h)-1;
[ny,y]=xcorr(x,h); % Built in Function for Cross-
correlation
disp('Cross-Correlation 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('Cross-Correlation');
grid on; xlabel('n'); ylabel('x(n)*h(n)');
```



Application of Correlation

Correlation in radar and active sonar applications

x(n) is the transmitted signal and **y(n)** is the received signal.

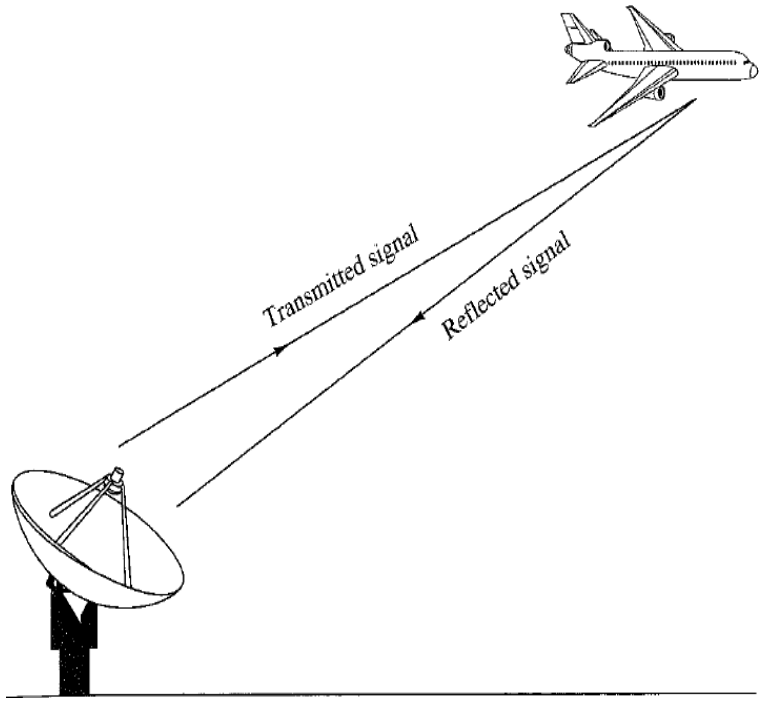
If a target is present y(n) will be

$$y(n) = \alpha x(n - D) + w(n)$$

Where α is some attenuation factor representing the signal loss involved in the round-trip transmission of the signal x(n), D is the round trip delay and w(n) represents additive noise that is picked up by the antenna and any noise generated by the electronic components and amplifier.

If there is no target,

y(n)=w(n)



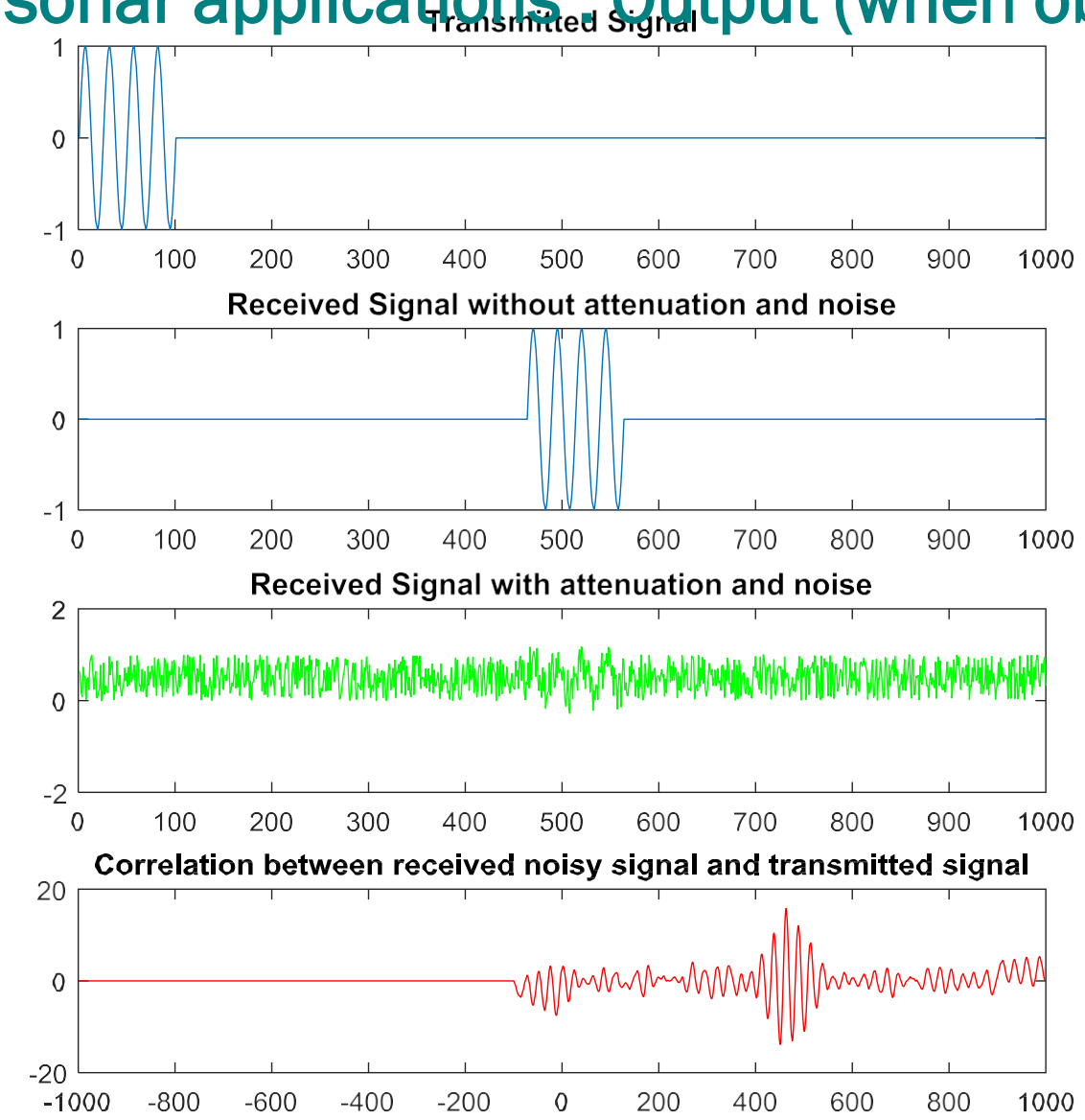
Comparing two signal x(n) and y(n) radar detects whether a target is present or not and also calculate the distance if target is present. In practice, the signal x(n-D) is heavily corrupted by the additive noise to the point where a visual inspection of y(n) does not reveal the presence or absence of the desired signal reflected from the target. Correlation provides us with a means for extracting this important information from y(n).

Correlation in active sonar applications : Program

```
clc
clear all
N=0:0.01:1;
delay_position=uint16(100+(1000-100)*rand(1,1));
attenuation=0.3;
sampling_Time=1/1000;
y=sin(2*pi*4*N);
tx=zeros(1,1000);
tx(N<=101)=y;
rx=zeros(1,1000);
for i=1:1:101
   rx(delay_position+i)=y(i);
end
noise=rand(1,1000);
rx_noise=(rx.*attenuation)+noise;
[YY,NN]=xcorr(rx_noise,tx);
[c,t]=max(YY);
if(c>10)
distance=(340*(t-1000)*sampling_Time)/2;
distance
else
   disp('There is no object');
end
subplot(411);
plot(tx);
title('Transmitted Signal');
subplot(412);
plot(rx);
title('Received Signal without attenuation
and noise');
subplot(413);
plot(rx_noise,'g');
title('Received Signal with attenuation and
noise');
subplot(414);
plot(NN,YY,'r');
title('Correlation between received noisy
signal and transmitted signal');
```

Correlation in active sonar applications : Output (when object is pres

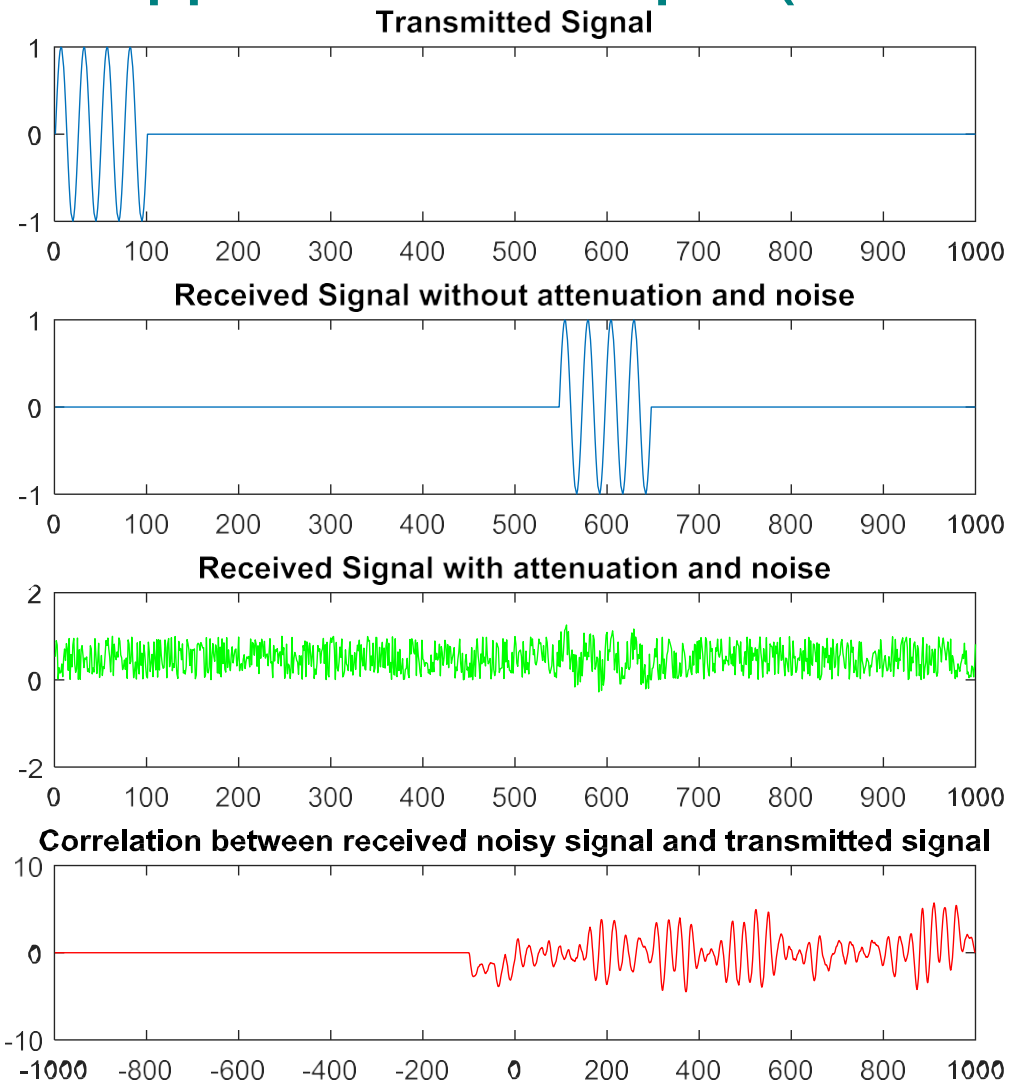
Correlation sequence has maximum value (15.78) at n= 463
i.e. after 463 unit delay an echo arrives.
So, if sampling frequency is 1KHz, the distance of object is 78.7100 m.
(Considering sound velocity 340 m/s)



Correlation in active sonar applications : Output (when object is not present)

Correlation sequence has maximum value =5.7181
Normally when object is present the maximum value of correlation sequence is greater than 12.

Decision: There is no object



University of Global Village (UGV), Barishal
Dept. of Electrical and Electronic Engineering

Experiment-6
z-Transform in Matlab

Course Code: EEE-307/CSE-309
Course Title: Digital Signal Processing Sessional

Prepared By
Noor Md Shahriar
Senior Lecturer, Dept. of EEE, UGV

Objectives

a) To realize the significance of z-transform

b) To determine the z-transform and inverse z-transform of discrete time signal and systems in MATLAB

c) To find the pole-zero plot and impulse response of DT system.

d) To implement moving average filter by z-transform.

Significance of z^{-n}

The z-transform of a discrete-time signal x(n) is defined as the power series

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

Where z is a complex variable. The relation sometimes called the direct z-transform because it transforms the time-domain signal x(n) into its complex-plane representation X(z).

$$\mathbf{z = re^{j\omega}}$$

$$\boldsymbol{z^{-n} = r^{-n}e^{-j\omega n}}$$

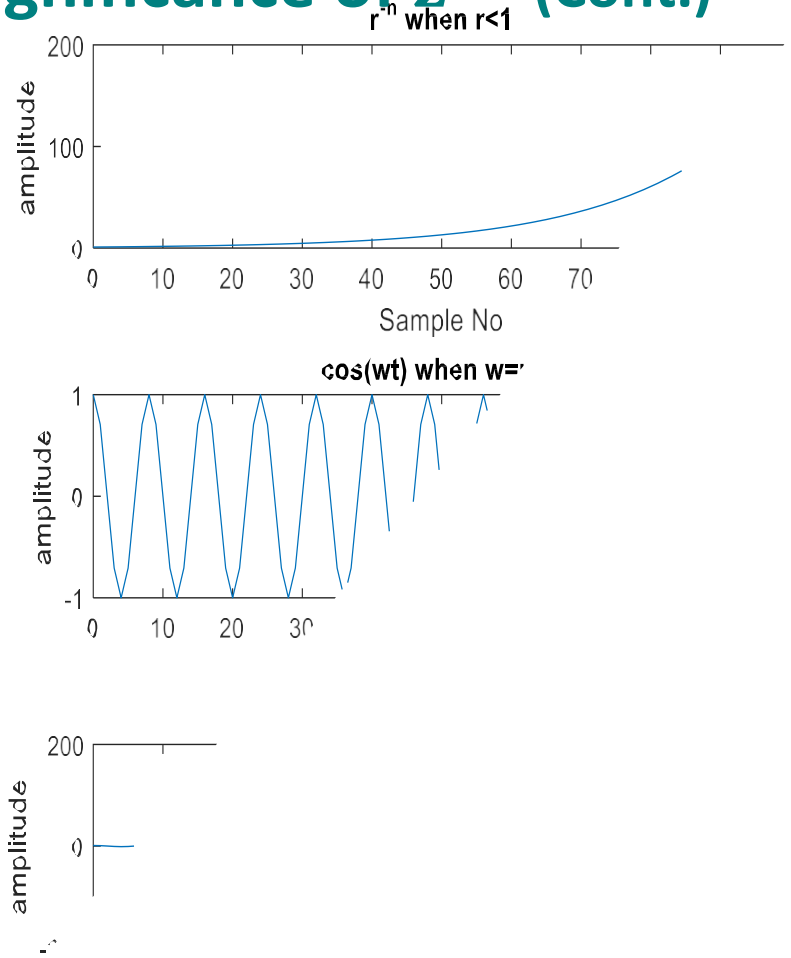
$$\boldsymbol{= r^{-n}[cos\ \omega n) - jsin(\omega n)]}$$

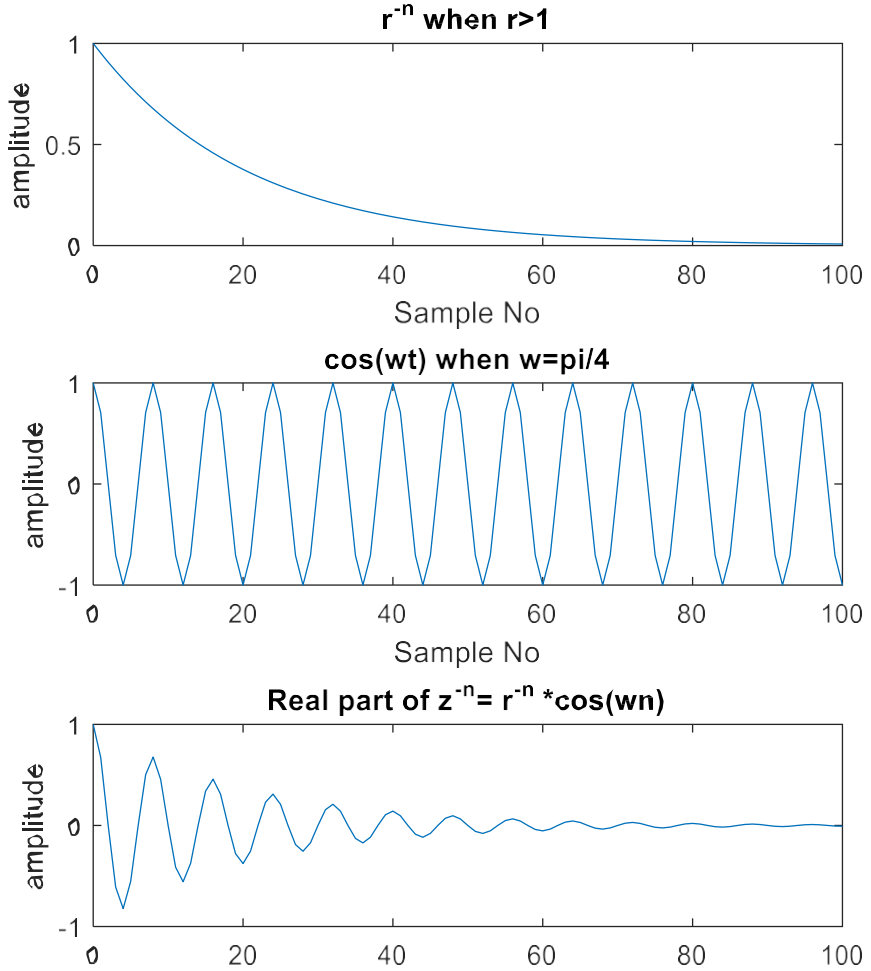
Where,
'r' is a real number
$e^{j\omega}$ is Euler's Number
ω is the angular frequency in radians per sample.

Significance of z^{-n} (Cont.)

```
n=0:.1:100;
r=0.96;
w=pi/4;
y1=r.^(-n);
y2=cos(w*n)+i*sin(w*n);
y=y1.*y2;
subplot(3,1,1);
plot(n,y1);
xlabel('Sample No');
ylabel('amplitude');
title('r^-^n when r=1');
subplot(3,1,2);
plot(n,y2);
xlabel('Sample No');
ylabel('amplitude');
title('cos(wt) when w=pi/4');
subplot(3,1,3);
plot(n,real(y));
xlabel(' ');
ylabel('amplitude');
title('Real part of z^-^n= r^-^n *cos(wn)');
```

Significance of z^{-n} (Cont.)

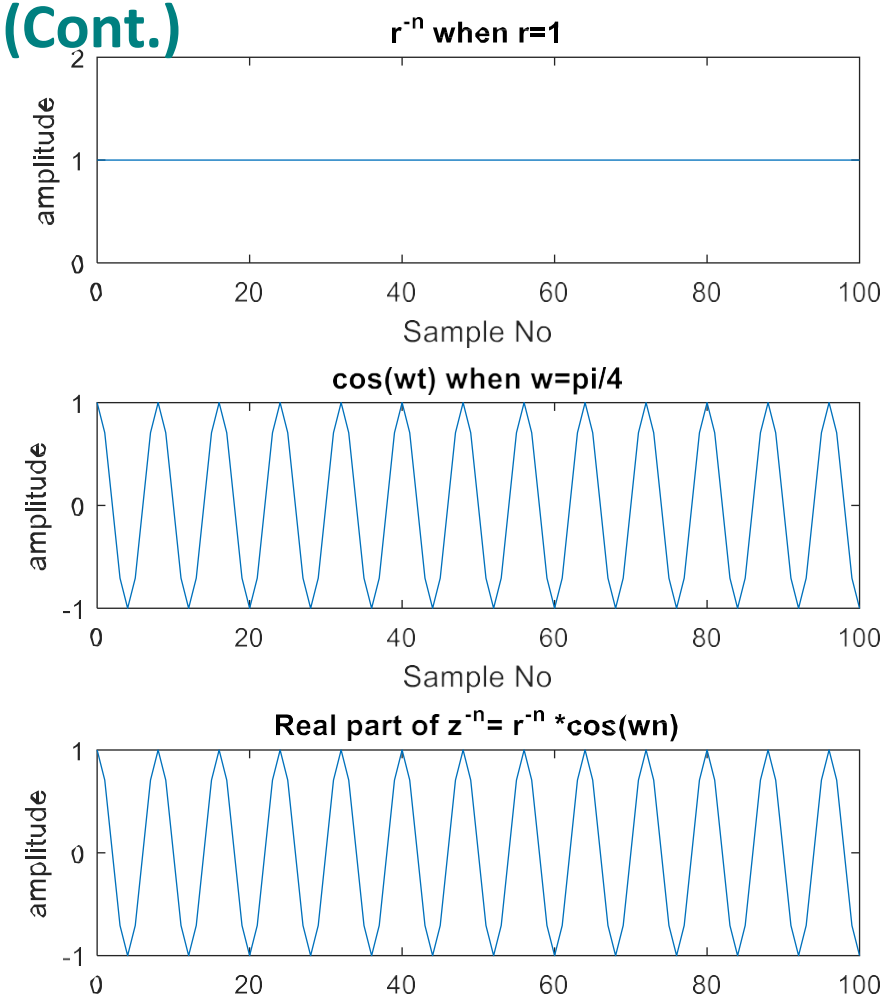




Plot of $r^{-n}cos\,(\omega n)$ when r<1 and $\omega = \frac{\pi}{4}$ per samples

Plot of $r^{-n}cos\,(\omega n)$ when r>1 and $\omega = \frac{\pi}{4}$ per samples

Significance of z^{-n} (Cont.)



Plot of $r^{-n}cos\,(\omega n)$ when r=1 and $\omega = \frac{\pi}{4}$ per samples

When,

r>1, z^{-n} has exponentially decreasing oscillation

r<1, z^{-n} has exponentially increasing oscillation

r=1, z^{-n} has oscillation of constant amplitude.

So we can say

z^{-n} represents a set of oscillating components of constant or increasing or decreasing amplitude based on value of z

Finding z-transform of a Sequence

```
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(result);
```

Example:
(Shown in Command Window)

```
Input Sequence:[-1 2 3 4 6]
Input Starting Point:-2
2*z + 4/z + 6/z^2 - z^2 + 3
```

Finding z-transform of a function

Example-1

```
>> syms a n f;
>> f=a^n;
>> ztrans(f)

ans =

-z/(a - z)
```

Example-2

```
>> syms n
>> x=1/4^n;
>> xz=ztrans(x)

xz =

z/(z - 1/4)
```

Example-3

```
>> syms w n;
>> x=sin(w*n);
>> xz=ztrans(x)

xz =

(z*sin(w))/(z^2 - 2*cos(w)*z + 1)
```

Finding inverse z-transform of a function

```
>> syms  z
>> x=2*z/(2*z-1);
>> xn=iztrans(x)

xn =

(1/2)^n
```

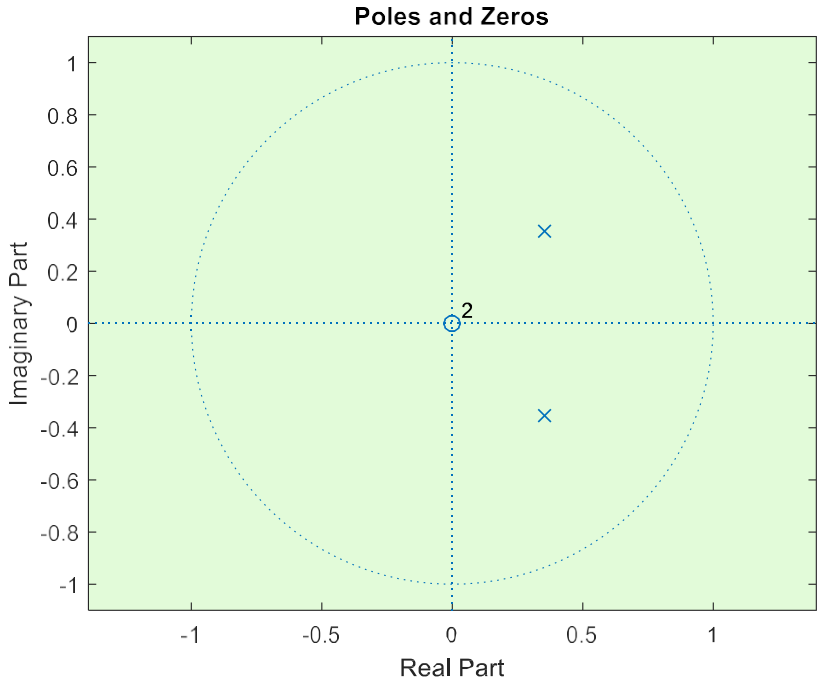
Z-Plane and Impulse response

```
function []=zplane_impulseResponse(Num,Den)
figure('Name','Z-Plane','position',[179 83 560 420]);  % [left spacing,  bottom spacing, size(x), size(y)]
zplane(Num,Den);
set(gca,'Color',[0.89 0.99 0.85]);     % gca(get current axes handle), [RGB Triplet ]
title('Poles and Zeros');
figure('Name','Impulse response','position',[751 83 560 420]);
impz(Num,Den);
set(gca,'Color',[0.89 0.92 0.92]);
title('Impulse Response')
end
```

System with complex conjugate poles

$$H(z) = \frac{z}{z^2 - 0.707z + 0.2499} = \frac{z}{\big(z - (0.3535 + j0.3535)\big)(z - (0.3535 - j0.3535))}$$

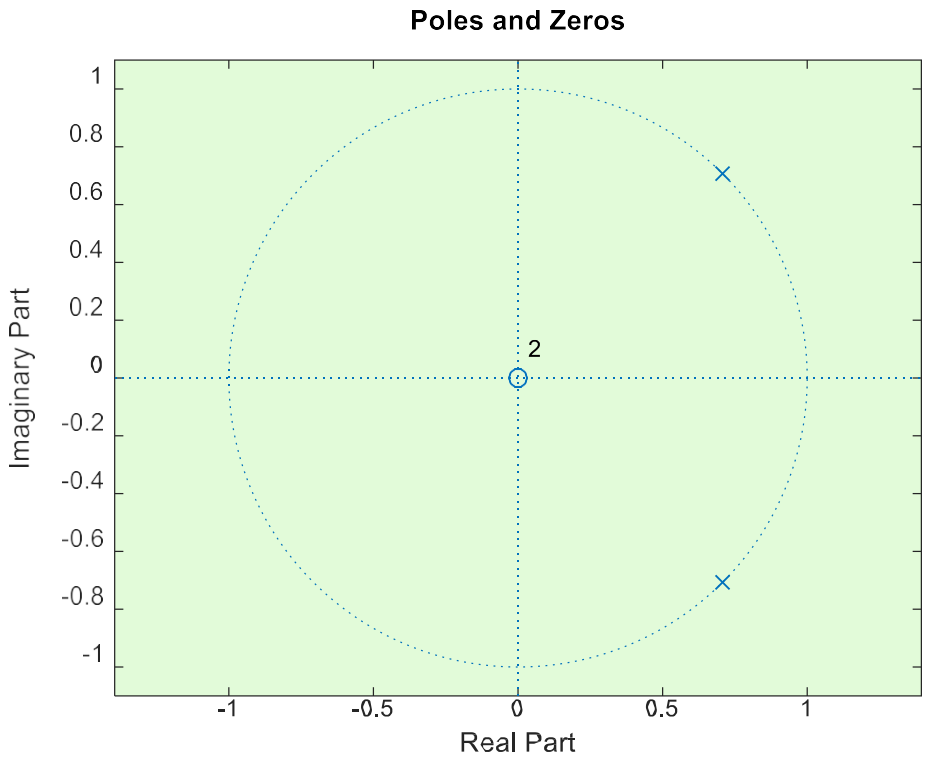
zplane_impulseResponse([1 0], [1 -0.707 .2499])

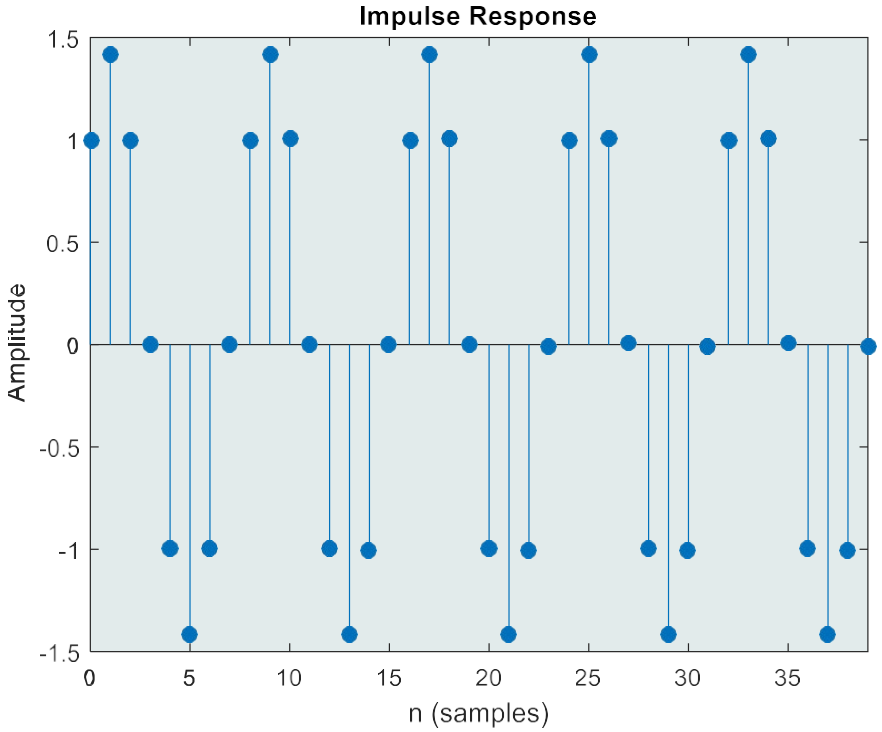


System with complex conjugate poles in unit circle

zplane_impulseResponse([1],[1 -1.4144 1]);

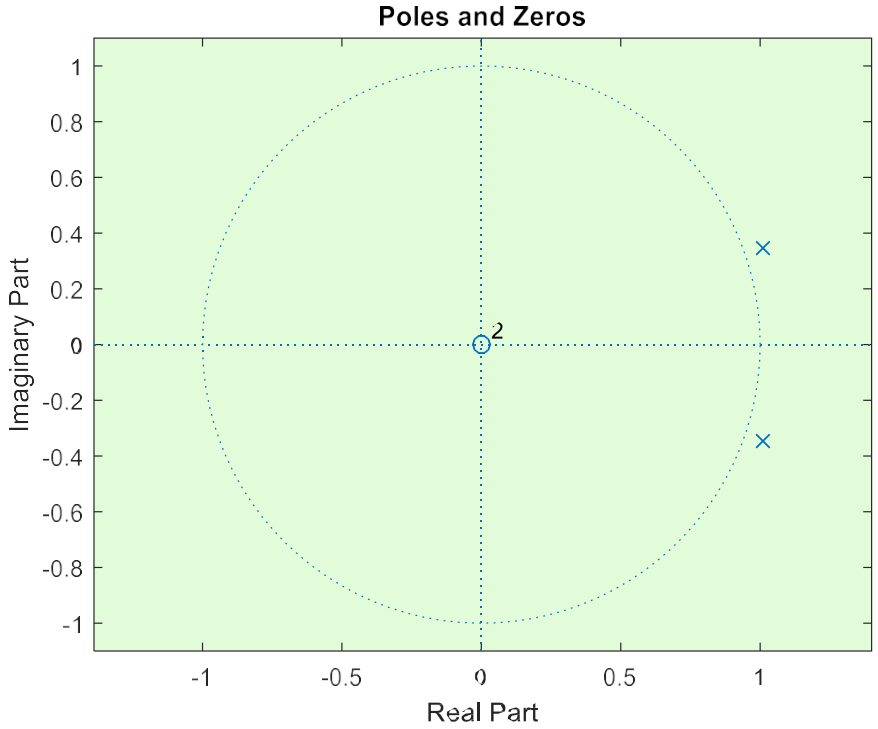
$$H(z) = \frac{z}{z^2 - 1.4144z + 1}$$

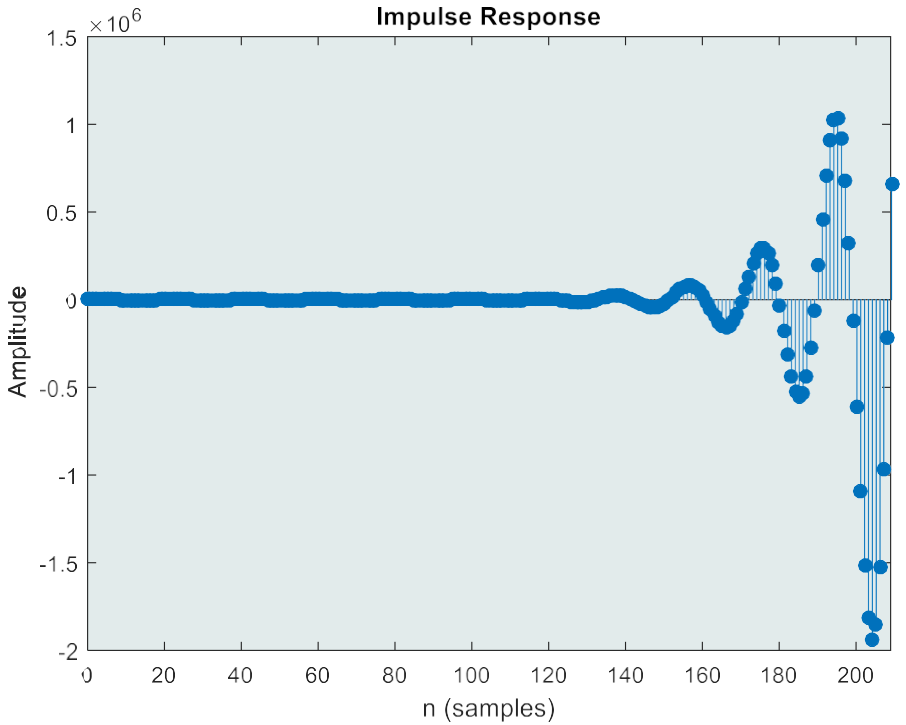




System with complex conjugate poles outside of unit circle

zplane_impulseResponse([1,0],[1 -2.02 1.14])





Finding Partial Fraction

>> [R,P,K]=residueZ(num,den);

Where R,P and K forms the partial fraction in the following forms,

$$X(z) = \frac{R_1 z}{(z - p_2)} + \frac{R_2 z}{(z - p_2)} + \cdots\cdots + K$$

Example: Find the partial fraction of following expression

$$X(z) = \frac{4 - \frac{7}{4}z^{-1} + \frac{1}{4}z^{-2}}{1 - \frac{3}{4}z^{-1} + \frac{1}{8}z^{-2}}$$

Code

```
>> num=[4  (-7/4)  (1/4)];
>> den=[1  (-3/4)  (1/8)];
>> [r,p,k]=residuez(num,den)
```

Output

```
r =3     -1
p = 0.5000      0.2500
k = 2
```

$$X(z) = \frac{3z}{(z - 0.5)} - \frac{z}{(z - 0.25)} + 2$$

Moving Average Filter

$$y(n) = \frac{1}{3}[x(n-1) + x(n) + x(n+1)]$$

$$\gg Y(z) = \frac{1}{3}X(z)[z^{-1} + 1 + z]$$

$$\gg H(z) = \frac{1}{3}\frac{1 + z + z^2}{z}$$

Num=(1/3)*[1 1 1]

Den=[1]

```
clc
clear all
close all
t=0:0.1:50;
x=5*sin(t);
n=randn(1,length(t));
x=x+n;
a=input('Enter the window
length:');
t2=ones(1,a);
num=(1/a)*t2;
den=[1];
y=filter(num,den,x);
plot(x,'b');
hold on
plot(y,'r');
```

University of Global Village (UGV), Barishal
Dept. of Electrical and Electronic Engineering

Experiment-7
Frequency domain analysis of DT signals

Course Code: EEE-307/CSE-309
Course Title: Digital Signal Processing Sessional

Prepared By
Noor Md Shahriar
Senior Lecturer, Dept. of EEE, UGV

Objectives

a)To understand the mathematics of transforming a signal from time domain to frequency domain.

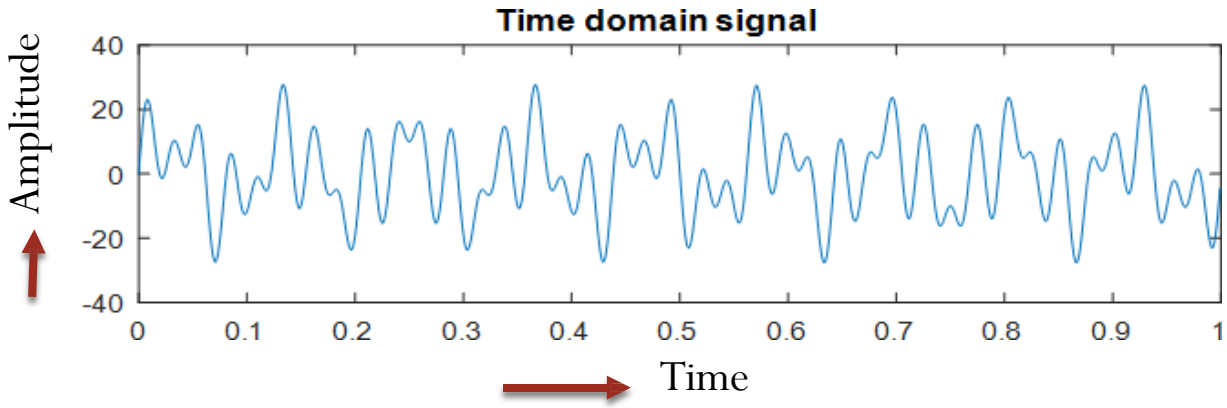
b) To transform a signal from time domain to frequency domain by DFT in MATLAB.

c) To transform a signal from time domain to frequency domain by FFT in MATLAB.

Time domain and Frequency domain Representation of Signal

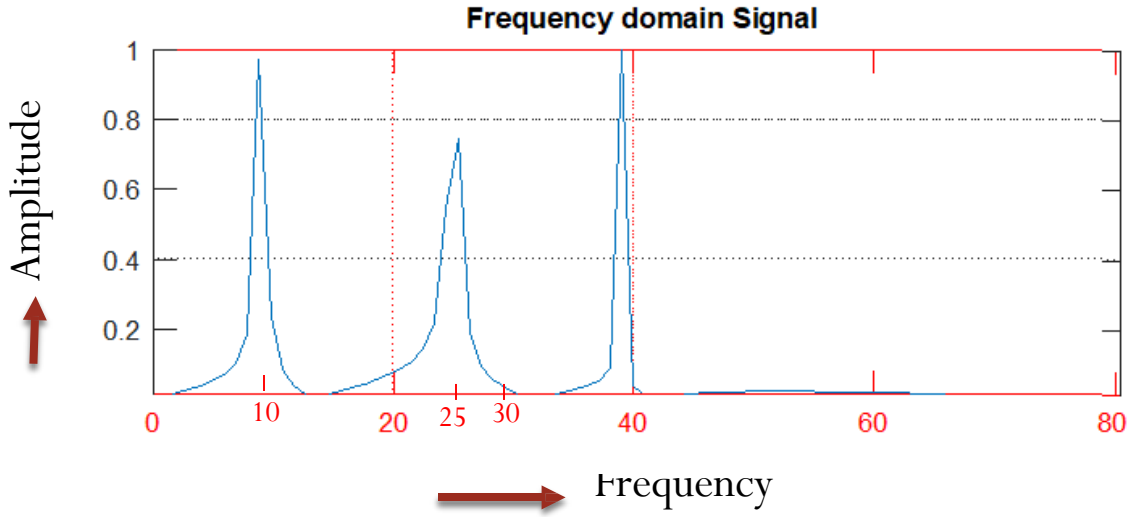
Signal in Time Domain ⟶ x(t)

The variation of amplitude with time

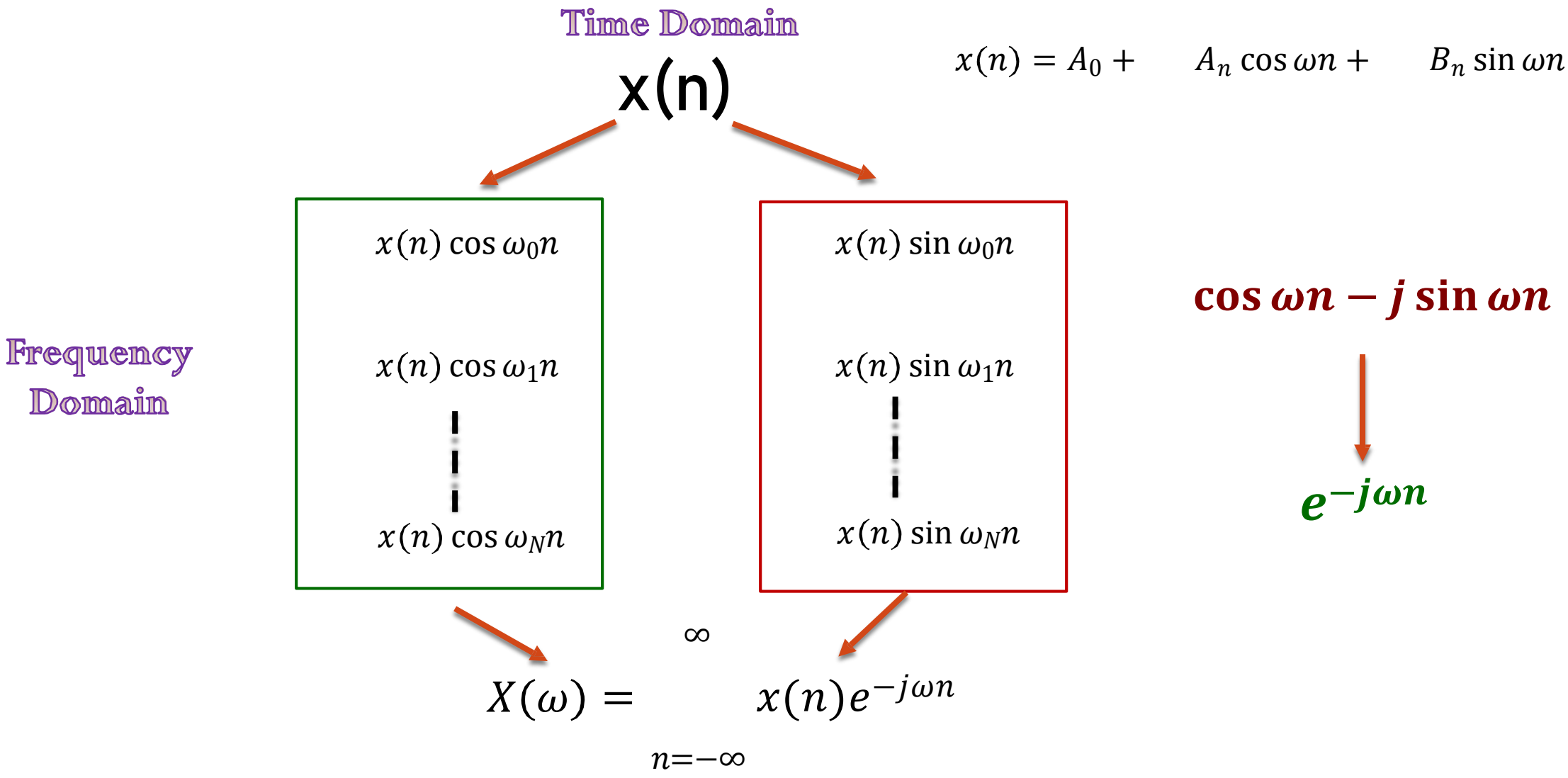


Signal in Frequency Domain ⟶ X(ω)

Amplitude of signals in various frequency components



Concept of converting time domain to frequency domain



Fourier Transform of Discrete Time Signal

Discrete Time Fourier Transform (DTFT)

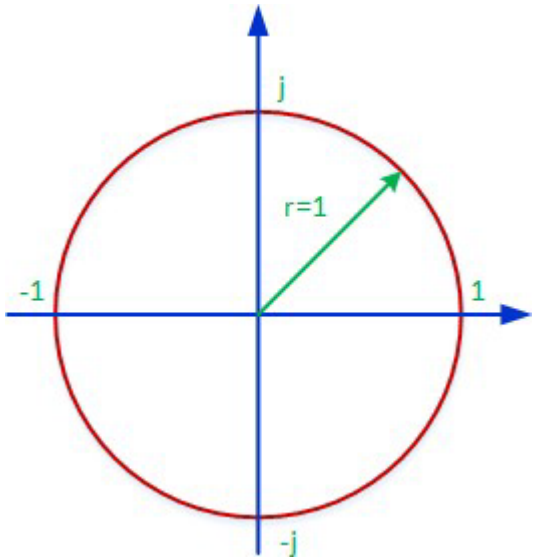
$$X(\omega) = \sum_{n=-\infty}^{+\infty} x(n)e^{-j\omega n}$$

In DTFT frequency domain (ω) is continuous.
ω changes from 0 to 2π, but it is continuous.

Discrete Fourier Transform (DFT)

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j\frac{2\pi nk}{N}}$$

In DFT frequency domain ($\omega = \frac{2\pi k}{N}$) is discrete.
ω changes from 0 to 2π taking 0 to (N-1) number of samples.

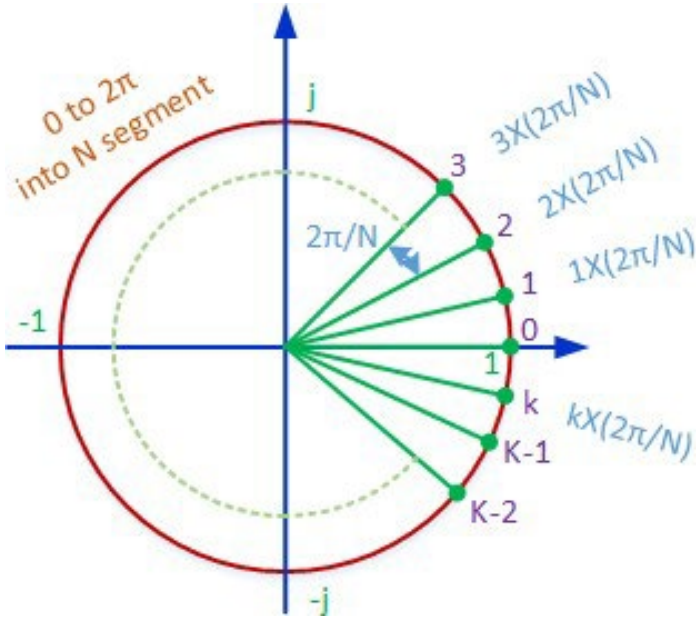


Angular Frequency

0 to 2π

$(-\pi$ to $\pi)$

$(-\frac{F_s}{2}$ to $+\frac{F_s}{2})$



Fourier Transform of Discrete Time Signal

Discrete Fourier Transform (DFT)

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi nk}{N}}$$

n= 0 to N-1 and k= 0 to N-1

$$e^{-j\frac{2\pi nk}{N}} = W_N^{nk} = \begin{bmatrix} W_N^0 & W_N^0 & W_N^0 & \cdots & W_N^0 \\ W_N^0 & W_N^1 & W_N^2 & \cdots & W_N^{N-1} \\ W_N^0 & W_N^2 & W_N^4 & & W_N^{2(N-1)} \\ & \vdots & & \ddots & \vdots \\ W_N^0 & W_N^{(N-1)} & W_N^{2(N-1)} & \cdots & W_N^{(N-1)(N-1)} \end{bmatrix}$$

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \\ . \\ . \\ X(N-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & . & . & 1 \\ 1 & W_N^1 & W_N^2 & . & . & W_N^{N-1} \\ 1 & W_N^2 & W_N^4 & . & . & W_N^{2(N-1)} \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ 1 & W_N^{N-1} & W_N^{2(N-1)} & . & . & W_N^{(N-1)(N-1)} \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ . \\ . \\ x(N-1) \end{bmatrix}$$

Discrete Fourier Transform (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);
subplot(212);
%plot(0:N-1,abs(dft));
plot((0:N-1)*(Fs/N),abs(dft));
```

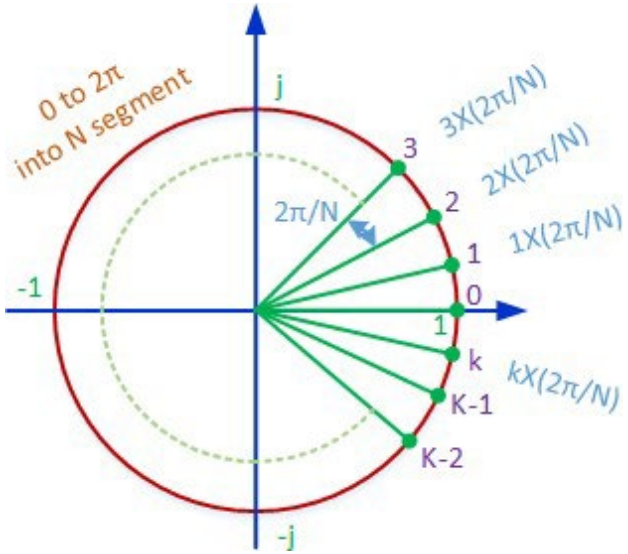
Defining Signal

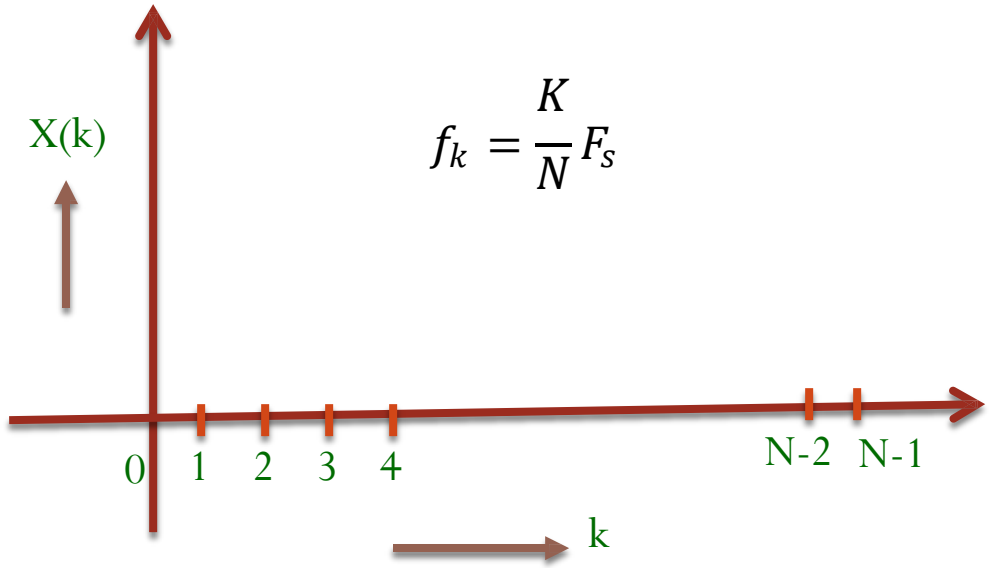
Transformation Matrix

$$e^{-j\frac{2\pi nk}{N}} = (e^{-j\frac{2\pi}{N}})^{nk} = E^{nk}$$

DFT Operation

Plotting Time-domain and frequency-domain Signal





$$f_k = \frac{K}{N} F_s$$

Fast Fourier Transform (FFT)

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \\ . \\ . \\ X(N-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & . & . & 1 \\ 1 & W_N^1 & W_N^2 & . & . & W_N^{N-1} \\ 1 & W_N^2 & W_N^4 & . & . & W_N^{2(N-1)} \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ 1 & W_N^{N-1} & W_N^{2(N-1)} & . & . & W_N^{(N-1)(N-1)} \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ . \\ . \\ x(N-1) \end{bmatrix}$$

The N-point DFT values can be computed in a total of N^2 multiplications and N(N-1) complex additions.

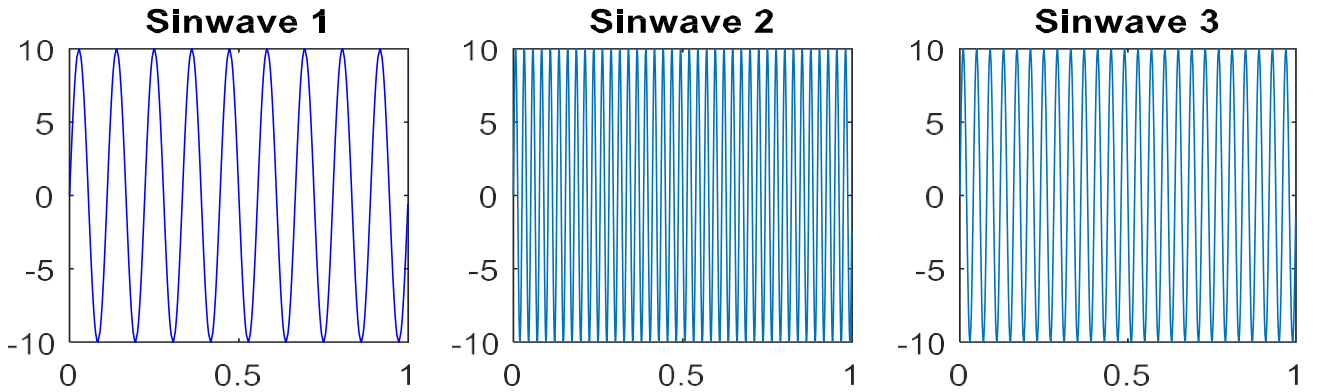
A Fast Fourier Transform (FFT) is an efficient algorithm to compute the Discrete Fourier Transform (DFT) and inverse of DFT. FFT requires a smaller number of arithmetic operations such as multiplications and addition than DFT (i.e. FFT requires lesser computation time than DFT).

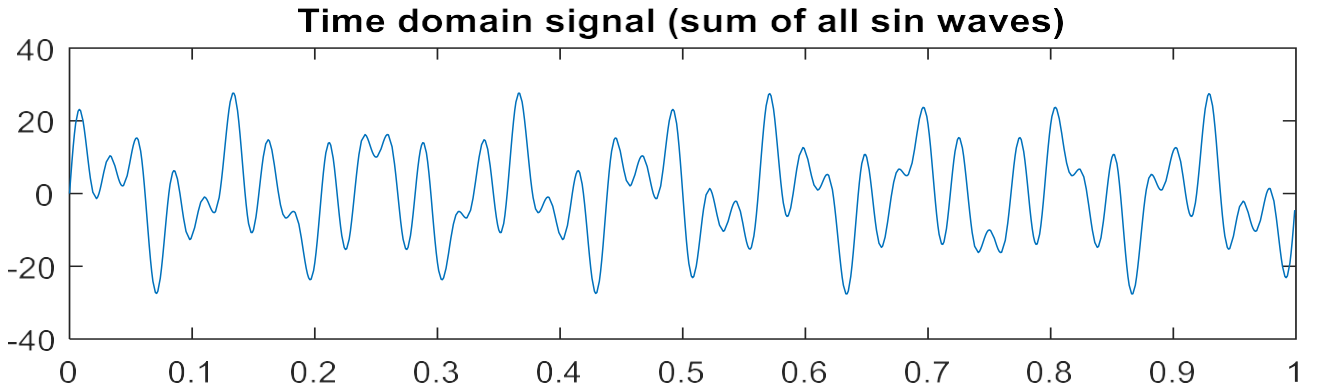
No of computations in direct DFT	Computations in FFT
Multiplications: N^2 Additions: N(N-1)	Multiplications: $\frac{N}{2} log_2(N)$ Additions: $N\ log_2(N)$

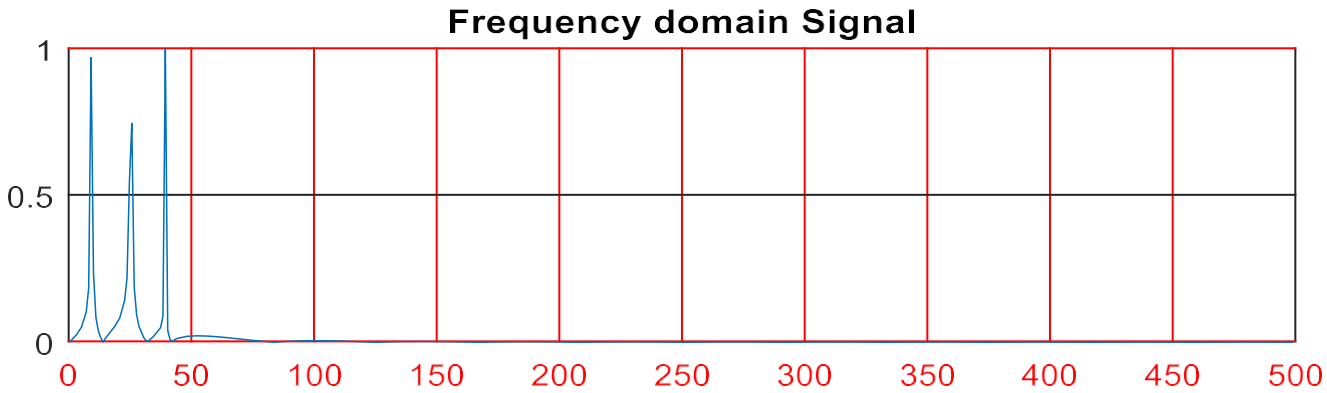
Comparison of Computational Complexity for the Direct Computation of the DFT Versus the FFT Algorithm

Fast Fourier Transform (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);
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;
```


Sinwave 1
Sinwave 2
Sinwave 3
10
5
0
-5
-10
0
0.5
1


Time domain signal (sum of all sin waves)
40
20
0
-20
-40
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Frequency domain Signal
1
0.5
0
0
50
100
150
200
250
300
350
400
450
500

University of Global Village (UGV), Barishal
Dept. of Electrical and Electronic Engineering

Experiment-8
Design of Finite Impulse Response (FIR) Filter

Course Code: EEE-307/CSE-309
Course Title: Digital Signal Processing Sessional

Prepared By
Noor Md Shahriar
Senior Lecturer, Dept. of EEE, UGV

Objectives

a) To understand the impact of window function in FIR filter design.

b) To understand the conversion of high pass filter kernel from low pass filter kernel.

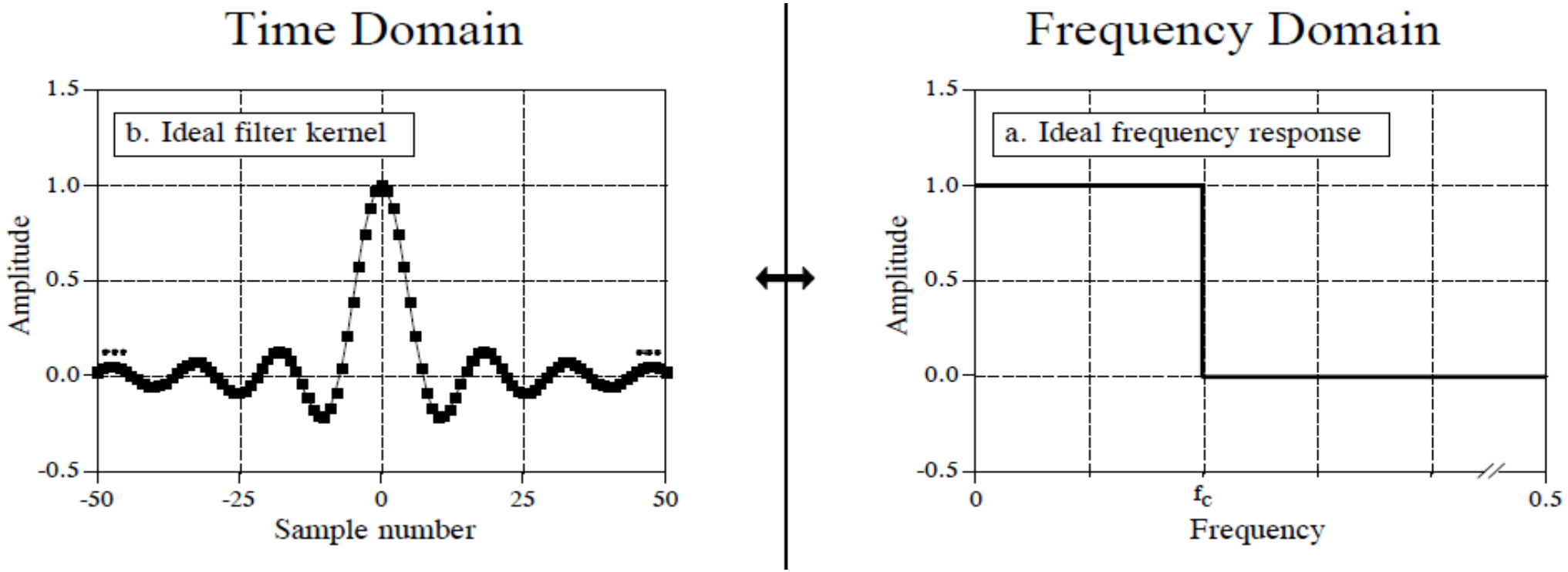
c) To design of band pass and band stop filter.

d) To understand the use of FIR filter in signal separation.

Why we need window function?

Impulse response of Ideal Low Pass Filter

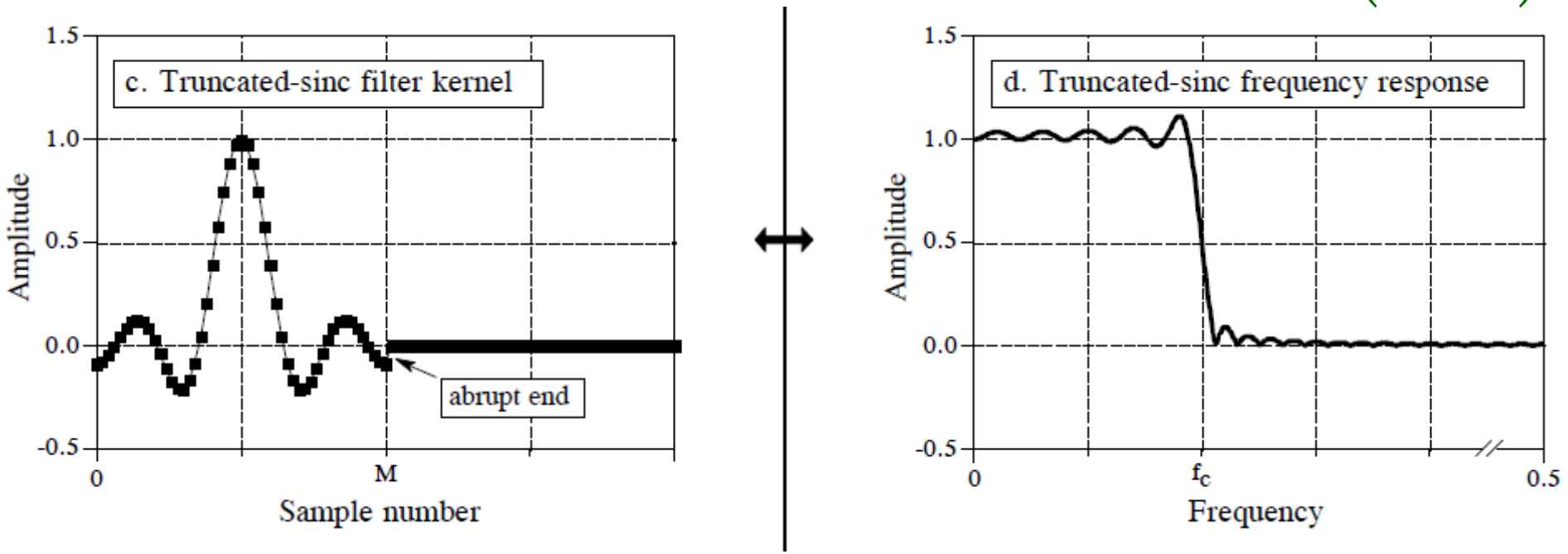
$$h(n) = \frac{\sin(\omega_c n)}{\pi n} = \frac{\sin(2\pi f_c n)}{\pi n}$$



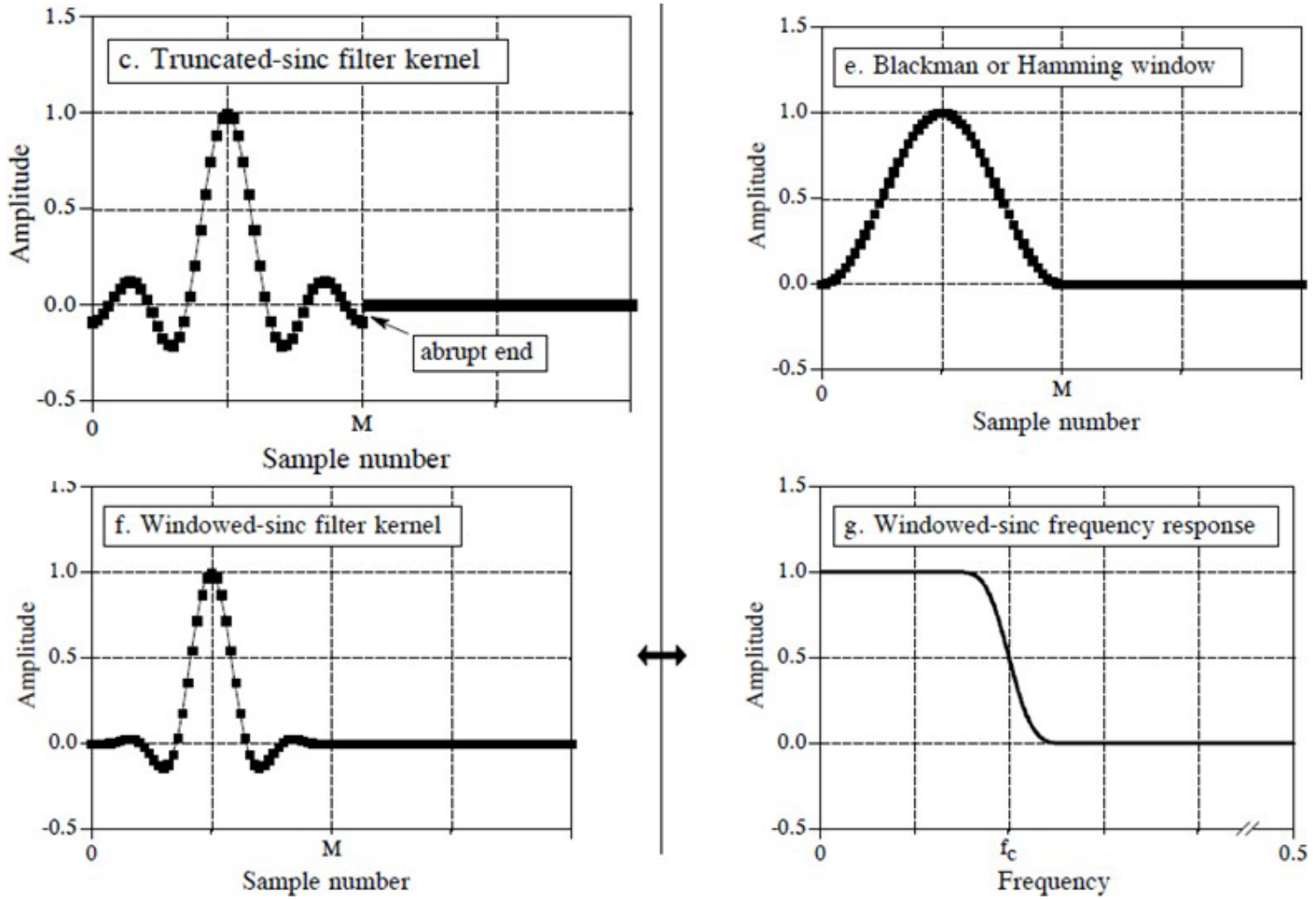
Impulse response of Practical (Realizable) Low Pass Filter

$$h(n) = \frac{\sin \omega_c(n - \frac{M-1}{2})}{\pi(n - \frac{M-1}{2})} \quad \text{Where,} \quad 0 \leq n \leq M-1 \text{ and } n \neq \frac{M-1}{2}$$

If M is selected to be odd, the value $h(n)$ *at* $n = \frac{M-1}{2}$ is: $h\left(\frac{M-1}{2}\right) = \frac{\omega_c}{\pi}$



Impulse response of Practical (Realizable) Low Pass Filter



$$h(n) = \frac{\sin \omega_c (n - \frac{M-1}{2})}{\pi (n - \frac{M-1}{2})}$$

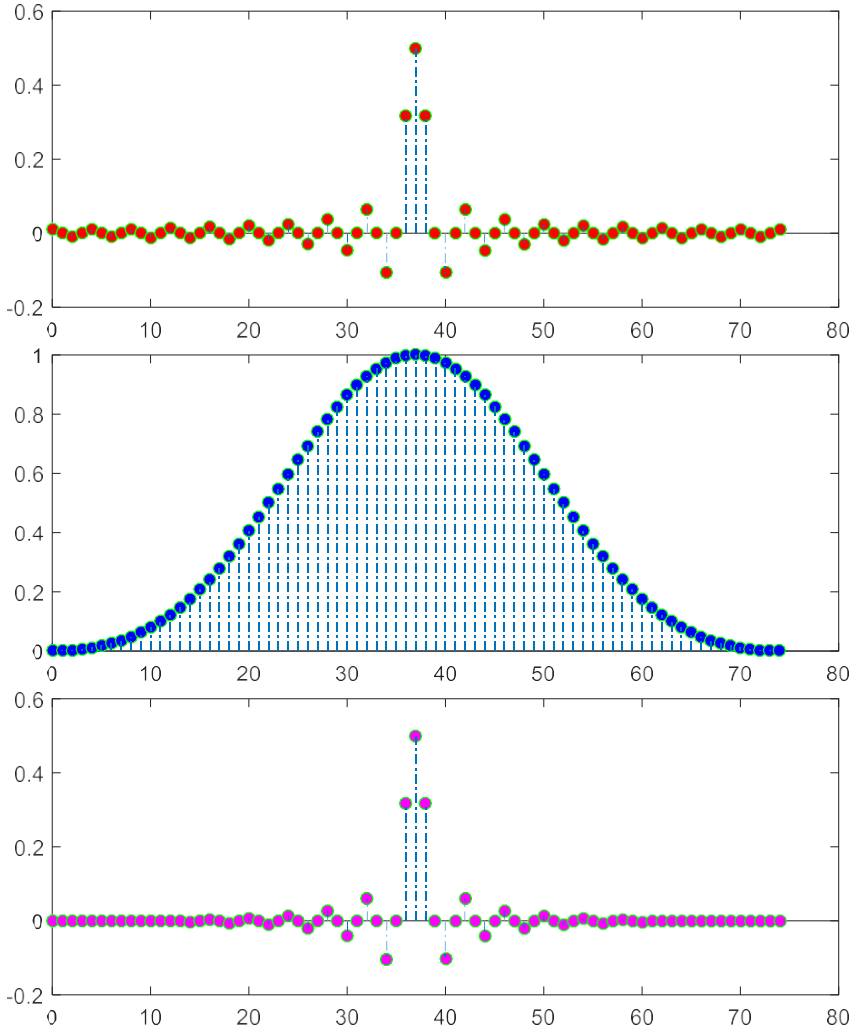
$$w(n) = 0.42 - 0.5\, cos\, \frac{2\pi n}{M-1} + 0.08\, cos\, \frac{4\pi n}{M-1}$$

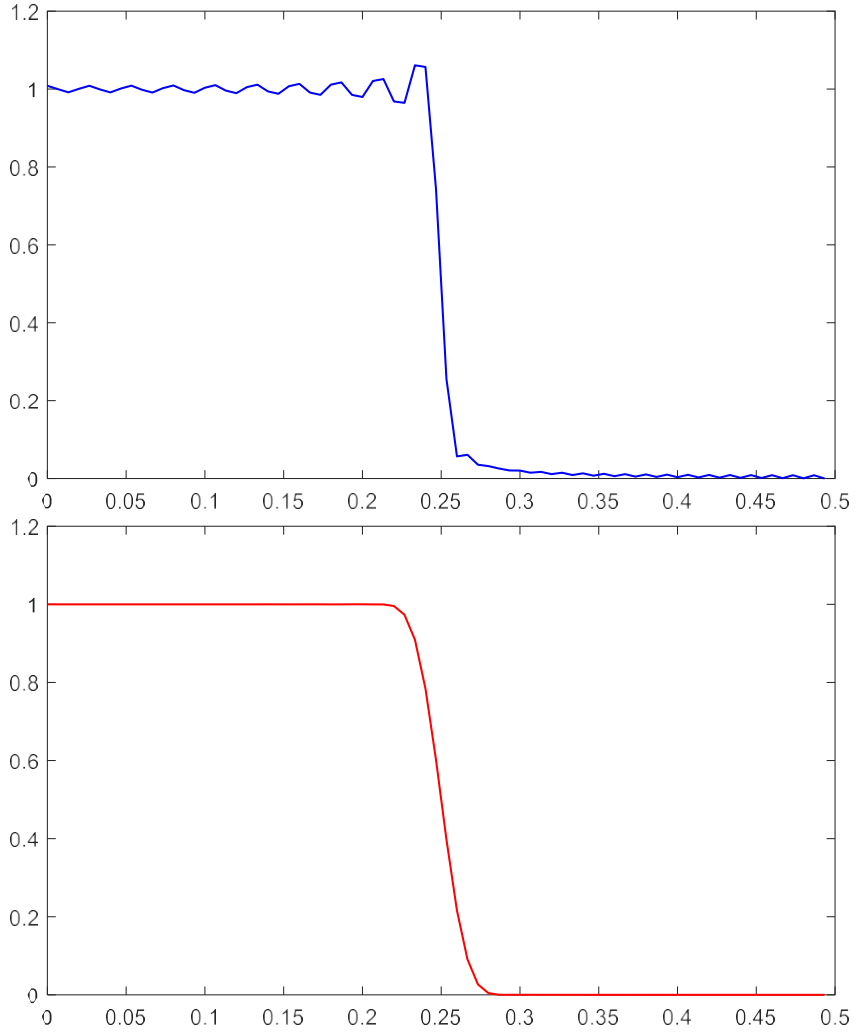
$$\boldsymbol{h(n) \times w(n)}$$

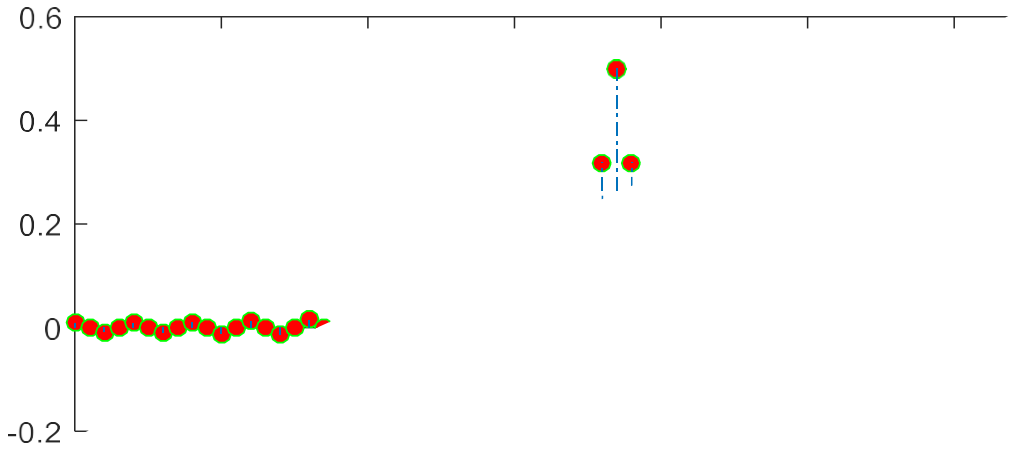
FIR filter

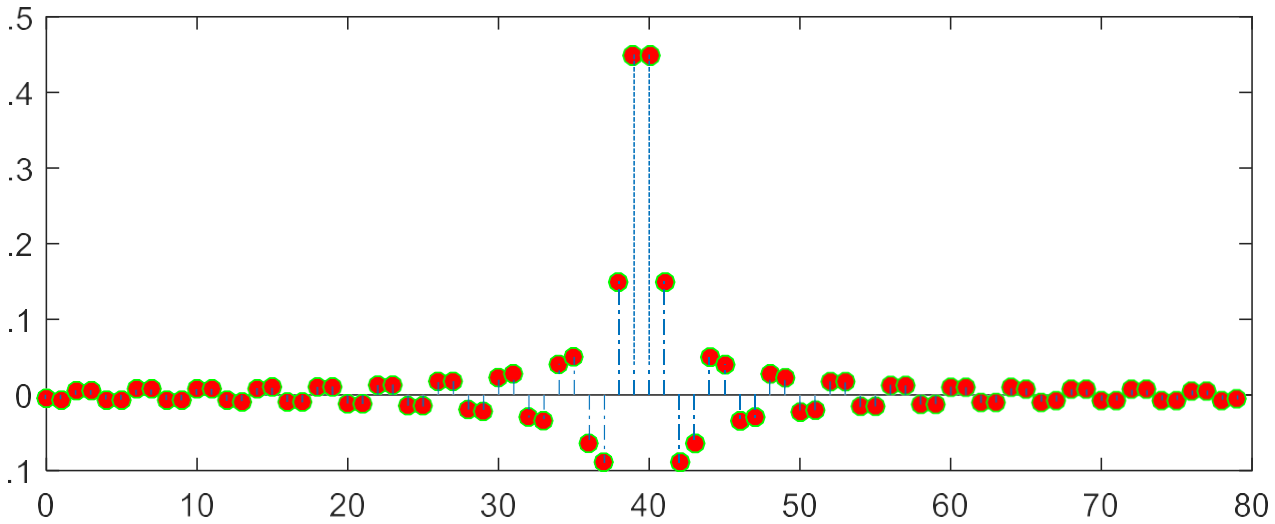
```
M=input('Define the length of filter kernel:');
n=[0:1:M-1];
wc=0.5*pi;
hd=sin(wc*(n-(M-1)./2))./(pi*(n-(M-1)./2));
if(rem(M,2)~=0)
hd(((M-1)/2)+1)=wc/pi;
end
[f_hd,w1]=freqz(hd,1,M);
w_blackman=0.42-0.5*cos((2*pi*n)./(M-1))+0.08*cos((4*pi*n)./(M-1));
h=hd.*w_blackman;
 [f_h,w2]=freqz(h,1,M);
subplot(6,2,[1 3]);
stem(n,hd,'LineStyle','-.','MarkerFaceColor','red','MarkerEdgeColor','green');
 subplot(6,2,[2 4 6]);
plot(w1./(2*pi),abs(f_hd),'b','Linewidth',1);
subplot(6,2,[5 7]);
stem(n,w_blackman,'LineStyle','-.','MarkerFaceColor','blue','MarkerEdgeColor','green');
subplot(6,2,[9 11]);
stem(n,h,'LineStyle','-.','MarkerFaceColor','m','MarkerEdgeColor','green');
subplot(6,2,[8 10 12]);
plot(w2./(2*pi),abs(f_h),'r','Linewidth',1);
fvtool(hd,1);
fvtool(h,1);
```

Output: Low pass FIR filter





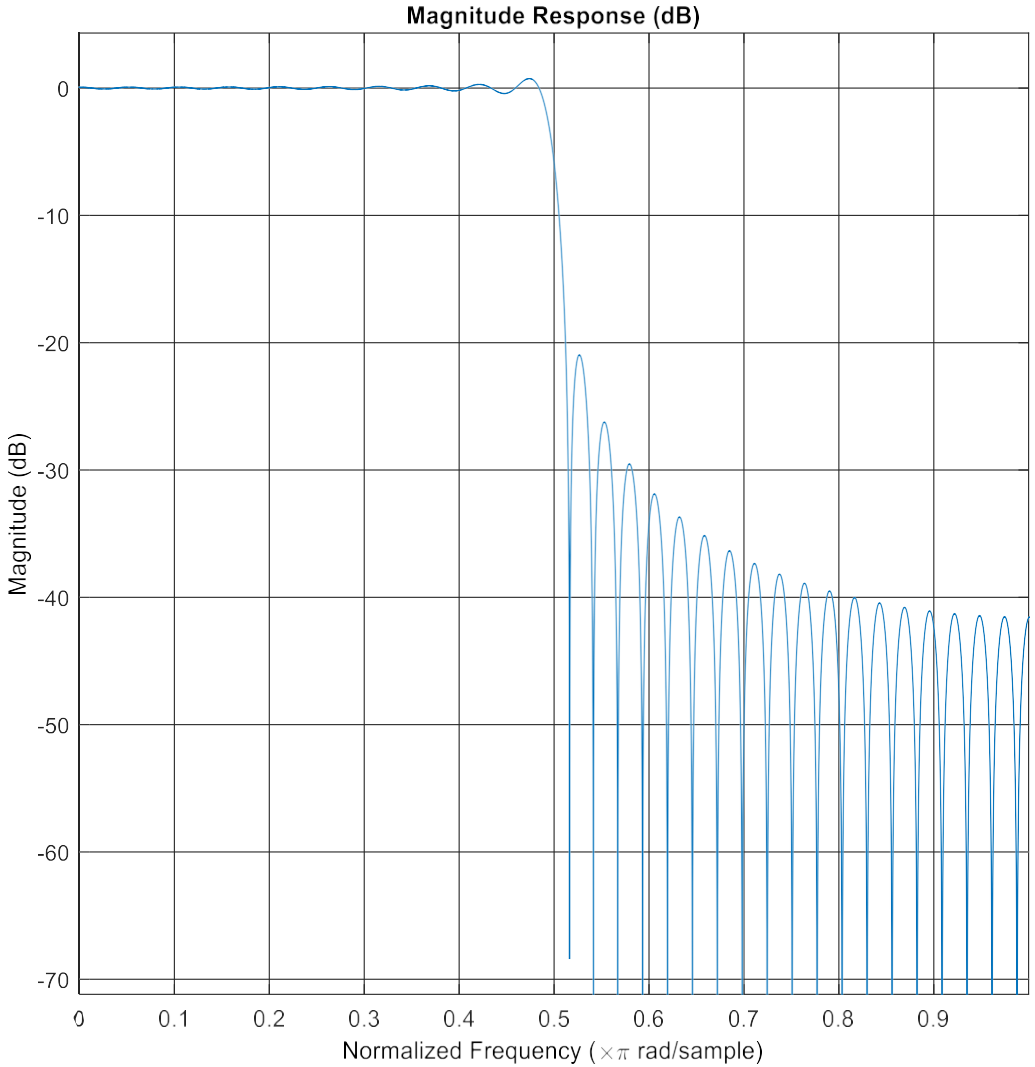


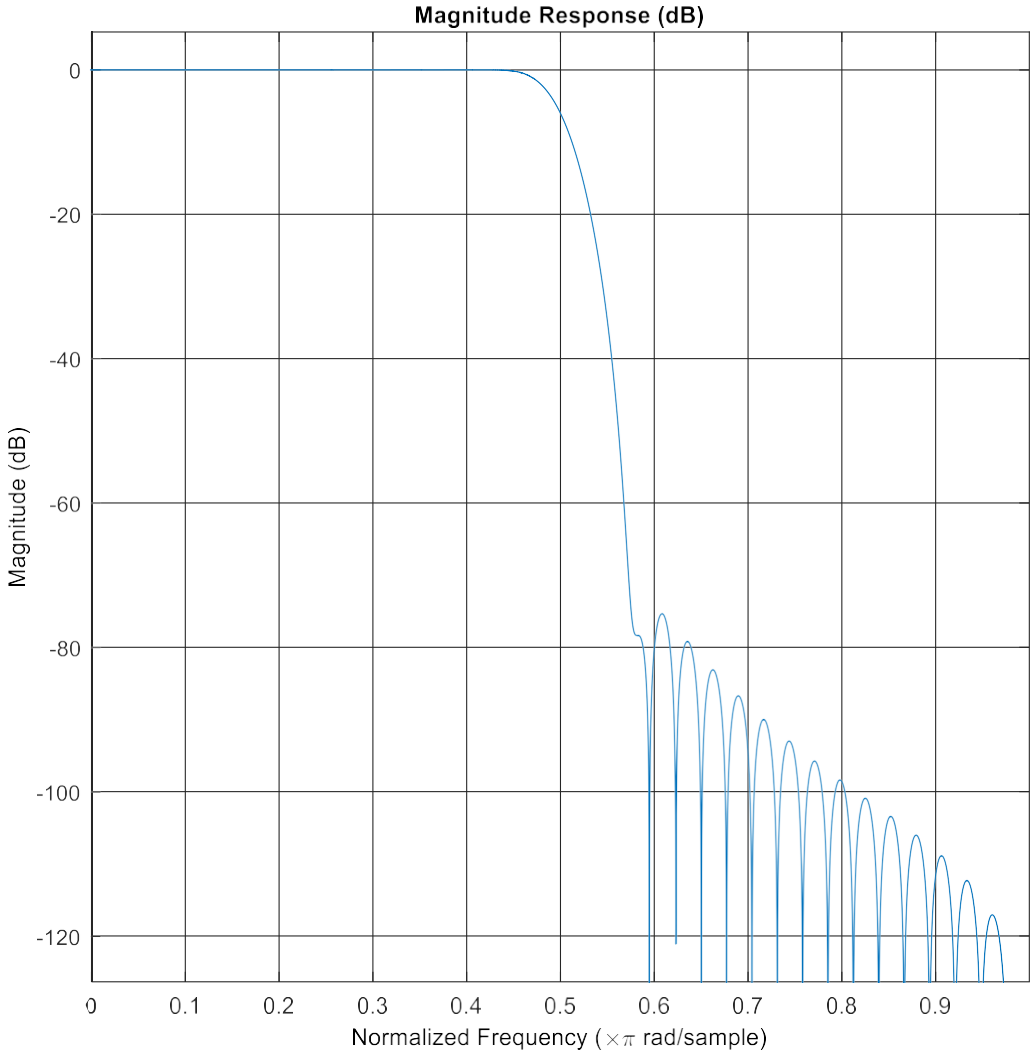


$$h(n) = \frac{\sin \omega_c(n - \frac{M-1}{2})}{\pi(n - \frac{M-1}{2})}$$

$$h\left(\frac{M-1}{2}\right) = \frac{\omega_c}{\pi}$$

Output: High pass FIR filter

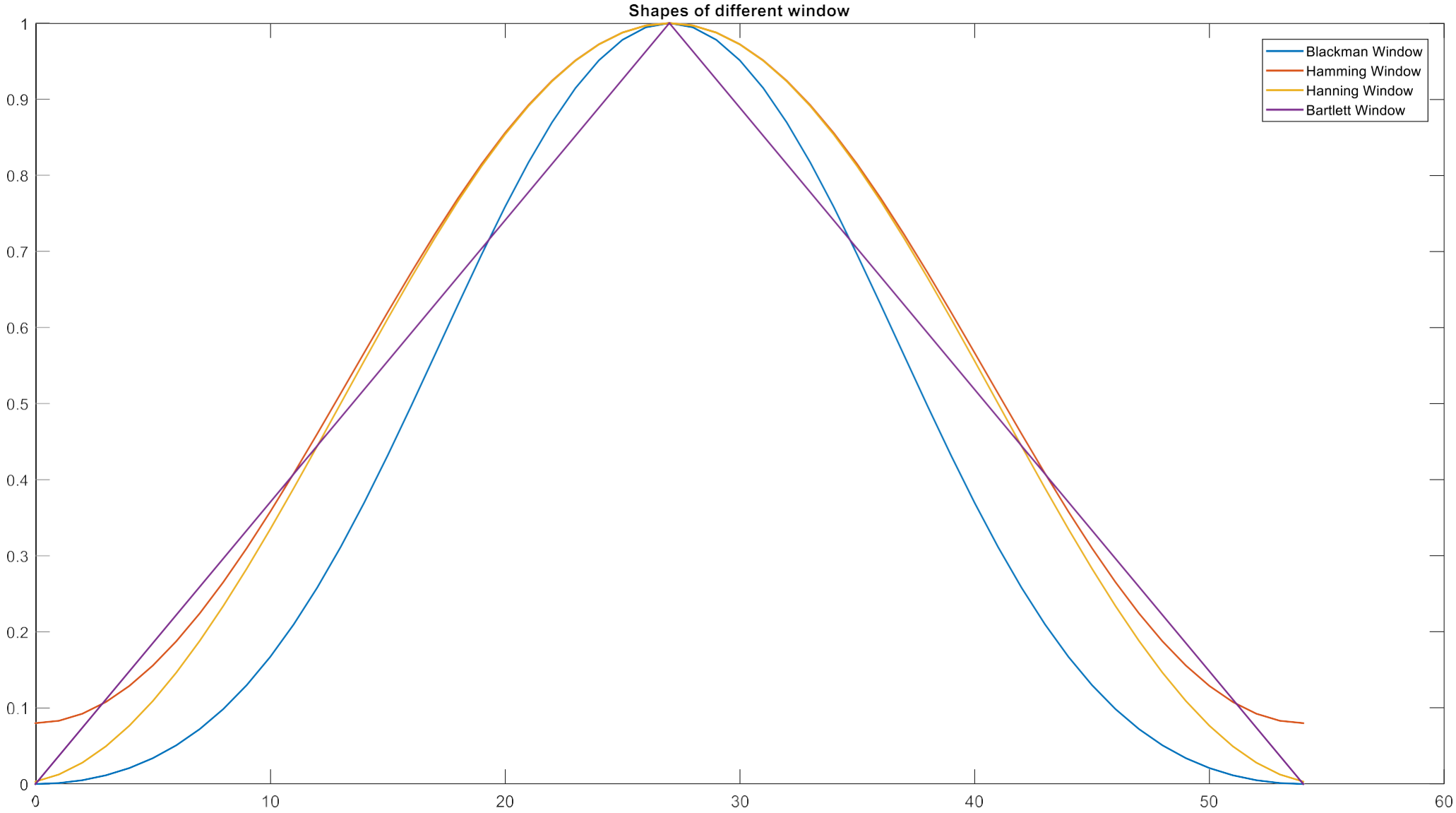


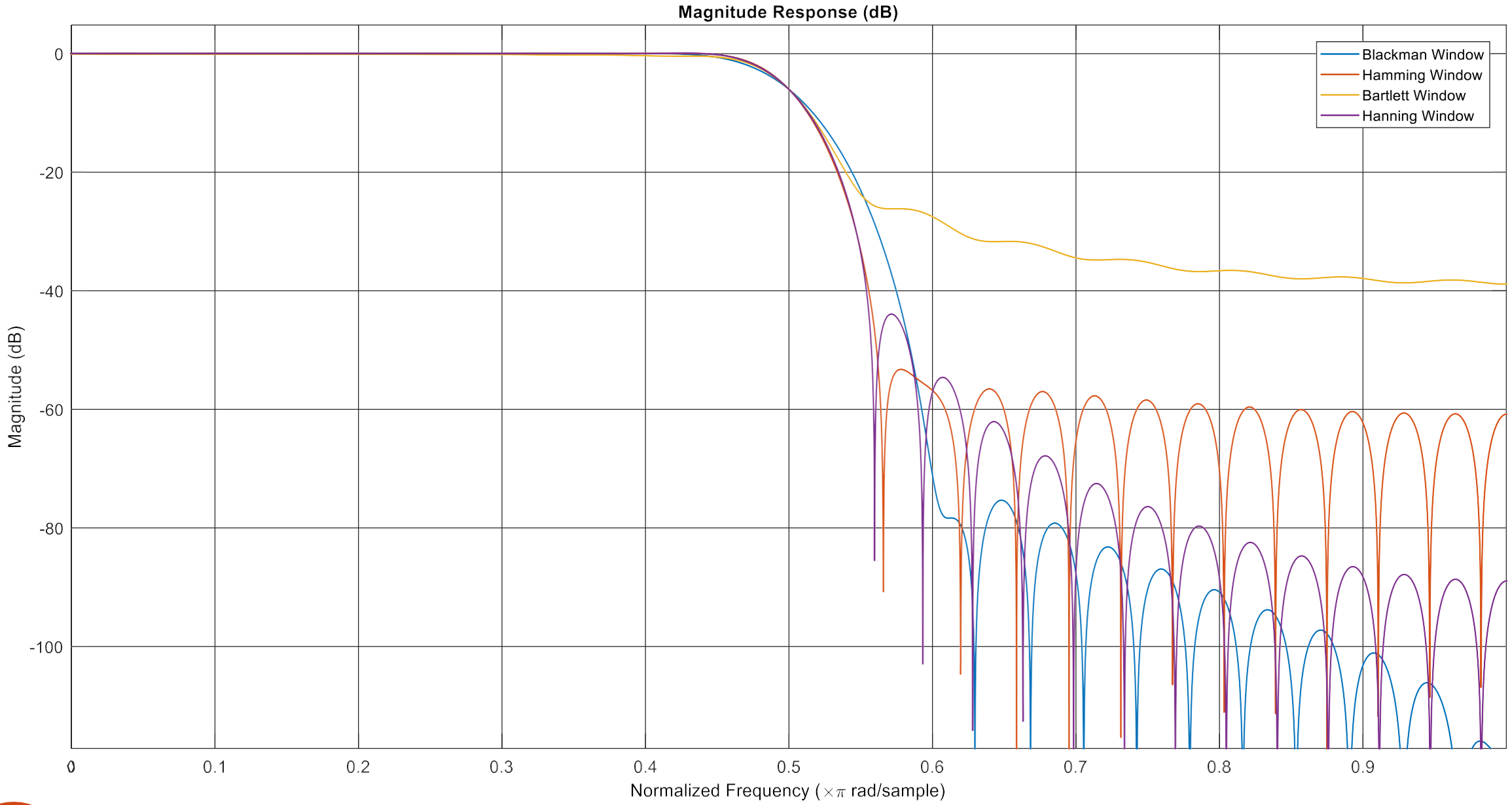


Comparison among different window functions.

Comparison among different window functions.

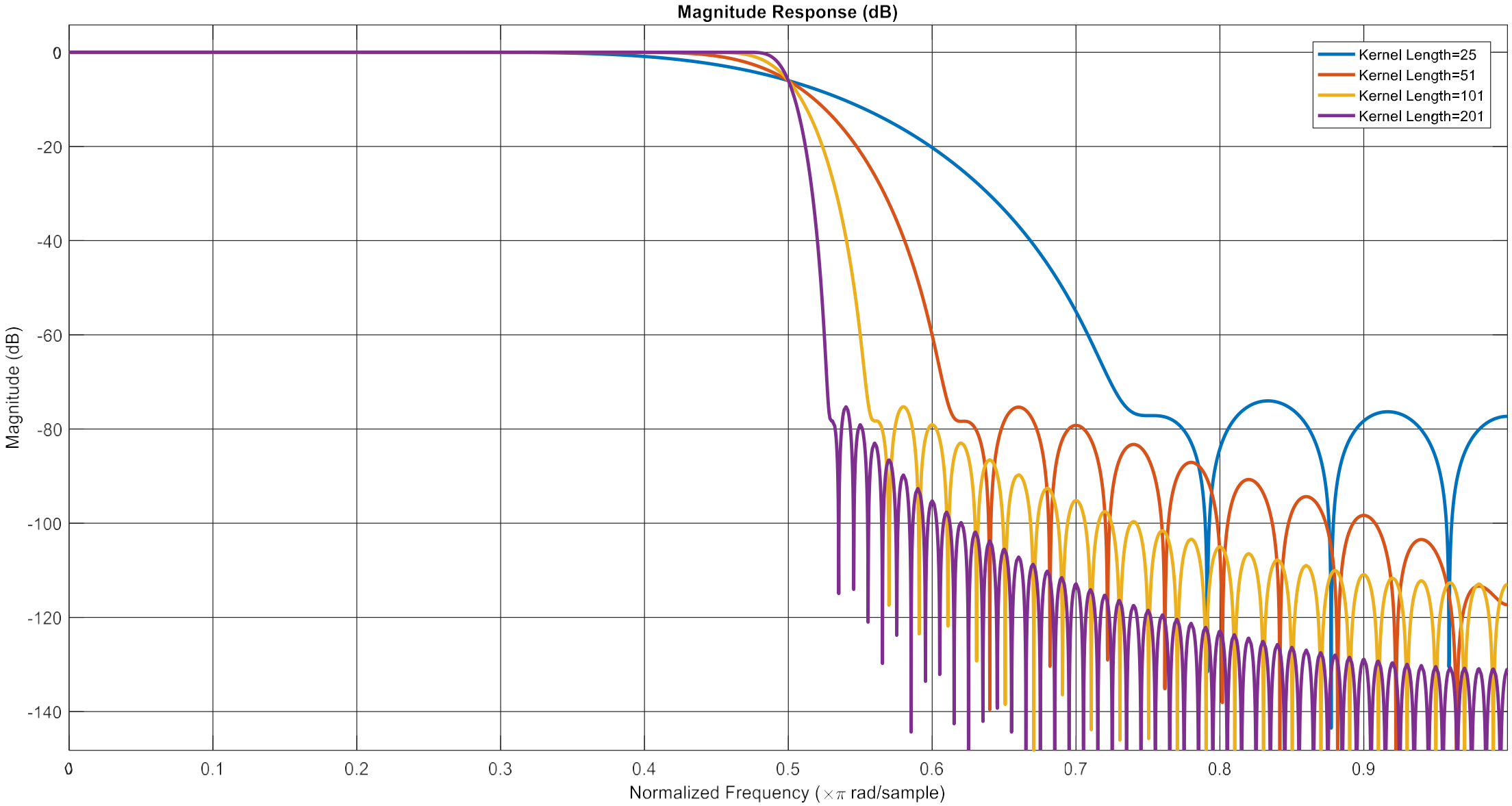
```
clc
clear
M=input('Define the length of filter kernel:');
n=0:1:M-1;
wc=0.5*pi;
hd=sin(wc*(n-(M-1)./2))./(pi*(n-(M-1)./2));
if(rem(M,2)~=0)
    hd(((M-1)/2)+1)=wc/pi;
end
w_blackman=0.42-0.5*cos((2*pi*n)./(M-1))+0.08*cos((4*pi*n)./(M-1));
w_hamming=0.54-(0.46*cos((2*pi*n)./(M-1)));
w_hanning=hanning(M)';
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');
```


Shapes of different window
Blackman Window
Hamming Window
Hanning Window
Bartlett Window
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
10
20
30
40
50
60


Magnitude Response (dB)
Blackman Window
Hamming Window
Bartlett Window
Hanning Window
0
-20
-40
-60
-80
-100
Magnitude (dB)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
Normalized Frequency (×π rad/sample)

Effect of kernel length on roll-off rate

```
M=[25,51,101,201]
for i=1:4
n=[0:1:M(i)-1];
wc=0.5*pi;
hd=sin(wc*(n-(M(i)-1)./2))./(pi*(n-(M(i)-1)./2));
if(rem(M(i),2)~=0)
hd(((M(i)-1)/2)+1)=wc/pi;
end
w=0.42-0.5*cos((2*pi*n)./(M(i)-1))+0.08*cos((4*pi*n)./(M(i)-1));
switch i
   case 1
      h1=hd.*w;
   case 2
      h2=hd.*w;
    case 3
       h3=hd.*w;
    case 4
       h4=hd.*w;
end
 end
 fvtool(h1,1,h2,1,h3,1,h4,1);
legend('Kernel Length=25','Kernel Length=51','Kernel Length=101','Kernel Length=201');
```


Magnitude Response (dB)
Kernel Length=25
Kernel Length=51
Kernel Length=101
Kernel Length=201
Magnitude (dB)
0
-20
-40
-60
-80
-100
-120
-140
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
Normalized Frequency (×π rad/sample)

Conversion of low-pass filter into high-pass filter

Low-pass to high-pass conversion:

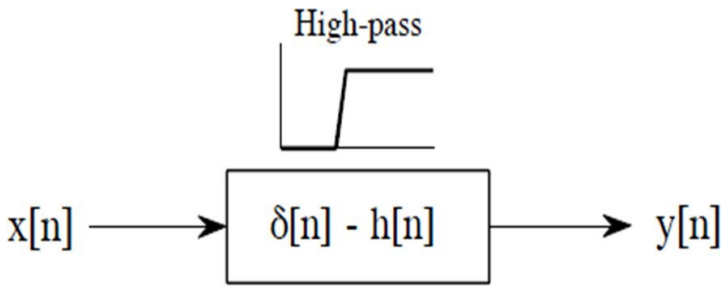
There are two methods for the low-pass to high-pass conversion:

a) **spectral inversion** and

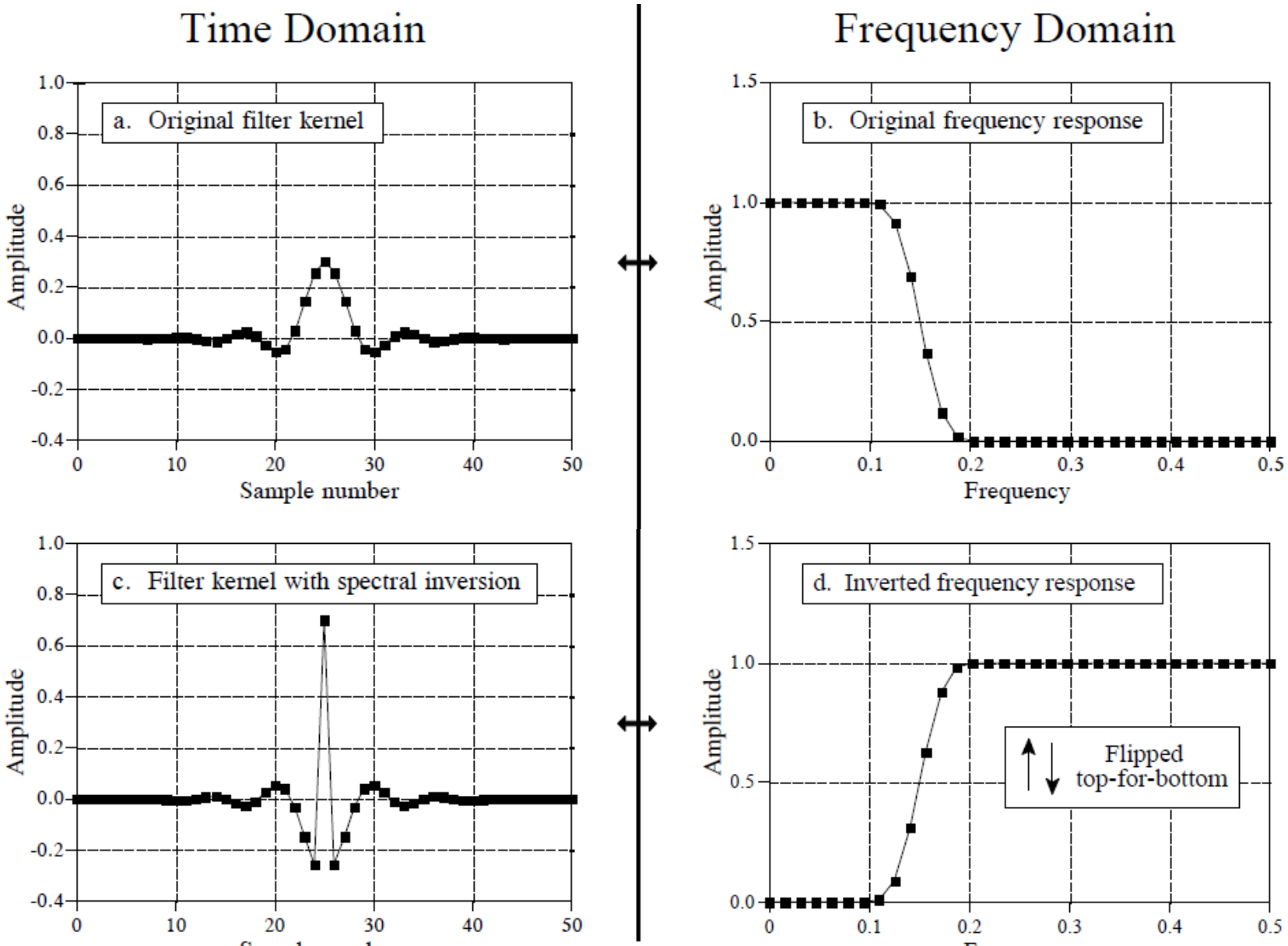
b)spectral reversal.

Both are equally useful.

Spectral Inversion



$$\delta(n) - h(n)$$



First, change the sign of each sample in the filter kernel.

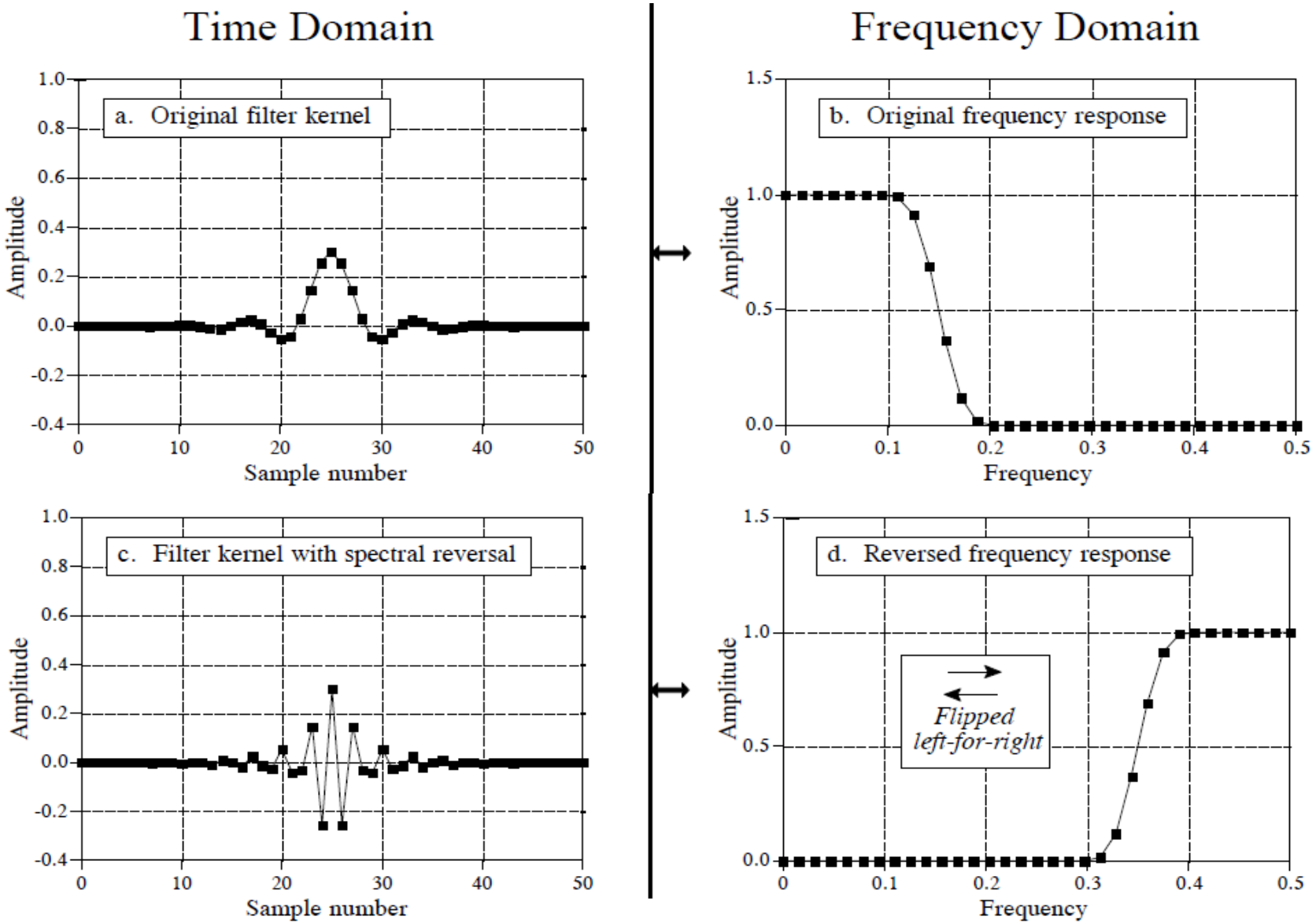
Second, add *one* to the sample at the center of symmetry.

Spectral Inversion (Program)

```
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));
 h=hd.*w_blackman;
h_high=-h;
h_high(((M-1)/2)+1)=h_high(((M-1)/2)+1)+1;
 fvtool(h,1,h_high,1);
 legend('Low-pass Filter','High-pass Filter');
```

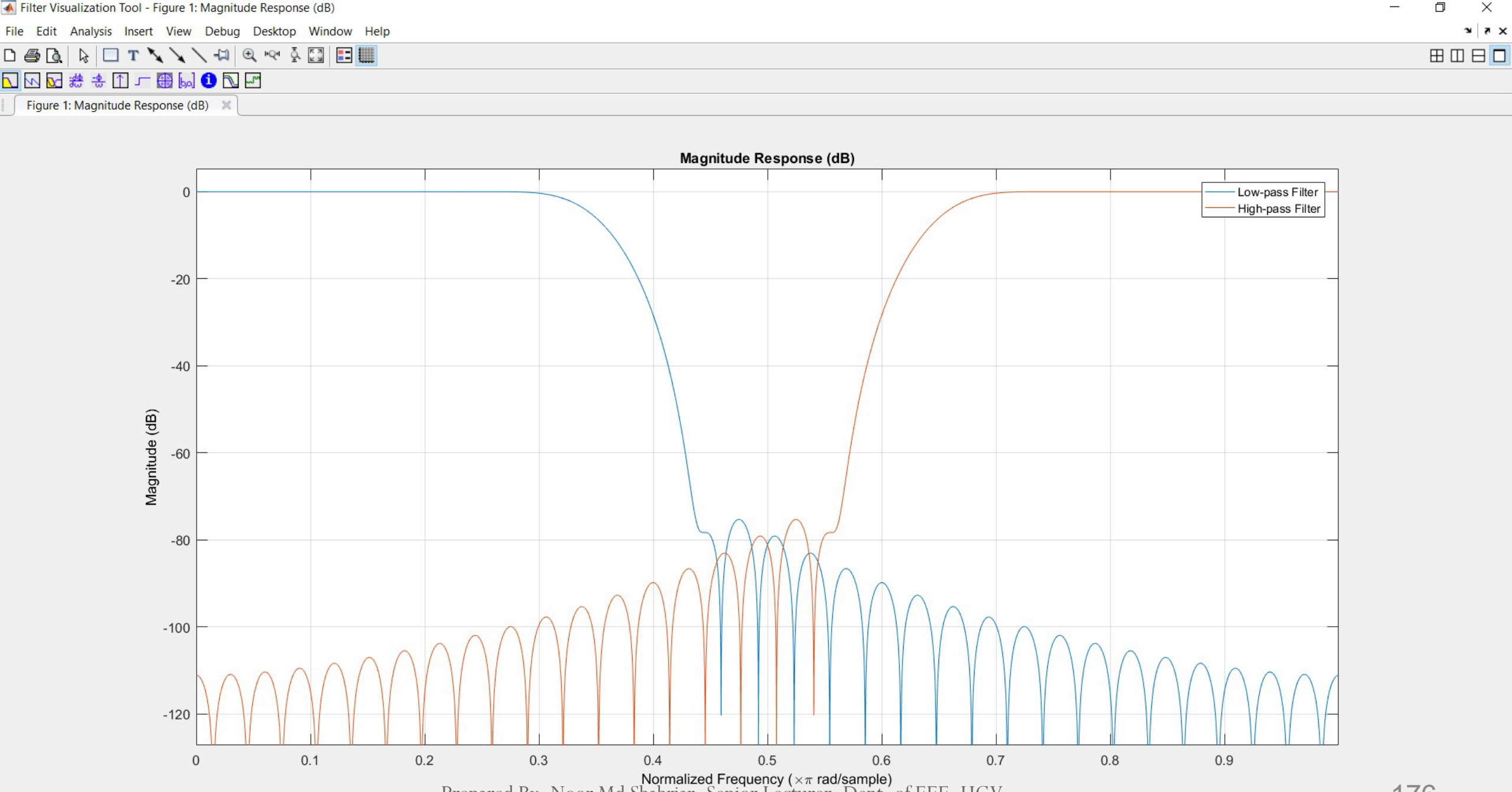
Spectral Reversal

The high-pass filter kernel, is formed by changing the sign of every other sample in low pass filter kernel.



Spectral Inversion (Program)

```
M=input('Define the length of filter kernel:');
 n=[0:1:M-1];
 wc=0.35*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
 [f_hd,w1]=freqz(hd,1,M);
 w_blackman=0.42-0.5*cos((2*pi*n)./(M-1))+0.08*cos((4*pi*n)./(M-1));
 h=hd.*w_blackman;
 h_high=h.*(-1).^n;
 fvtool(h,1,h_high,1);
 legend('Low-pass Filter','High-pass Filter');
```


Filter Visualization Tool - Figure 1: Magnitude Response (dB)
File Edit Analysis Insert View Debug Desktop Window Help
Figure 1: Magnitude Response (dB)
Magnitude Response (dB)
Low-pass Filter
High-pass Filter
Magnitude (dB)
0
-20
-40
-60
-80
-100
-120
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
Normalized Frequency (×π rad/sample)

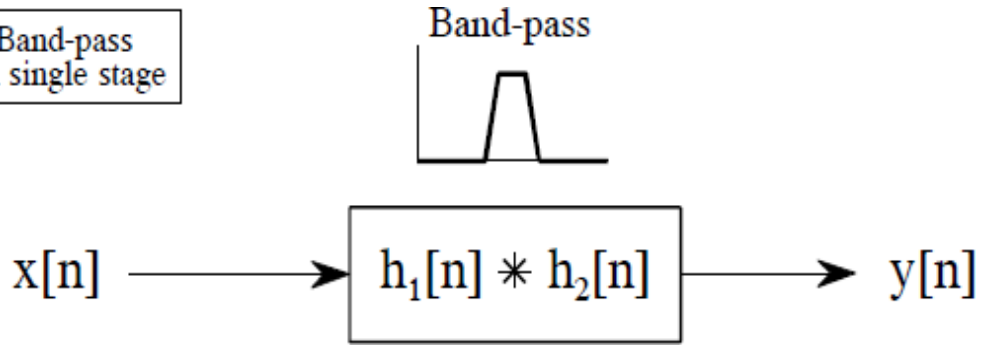
Design of Band-pass and Band-stop filter

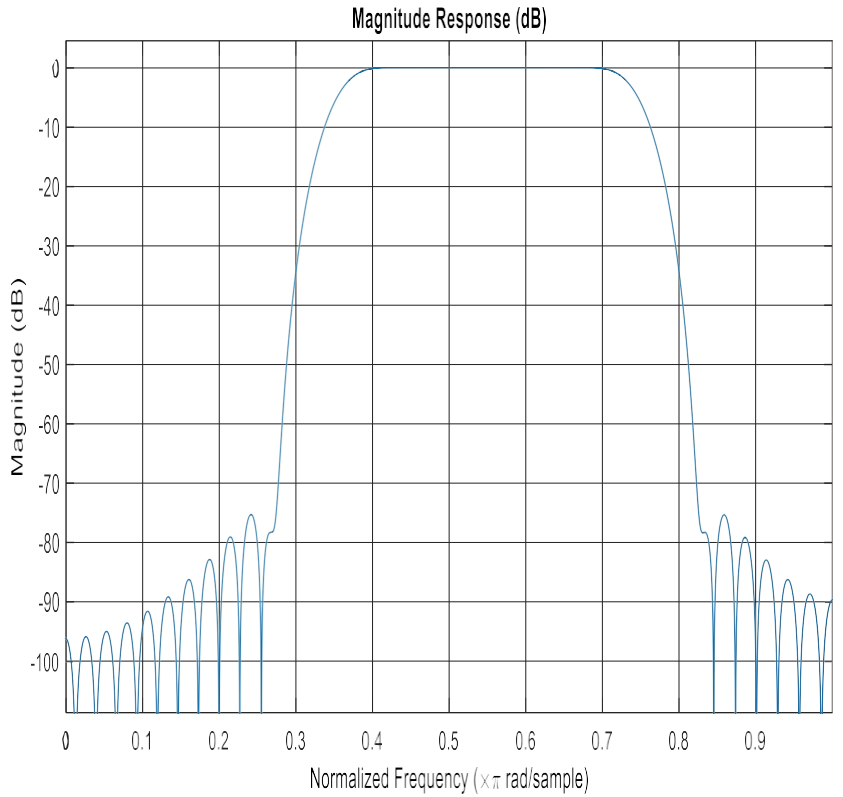
Band-pass filter from low and high pass filter kernel

Convolution of low and high pass filter kernel

```
M=input('Define the length of filter kernel:');
 n=[0:1:M-1];
 wc_L=0.75*pi;
 hd1=sin(wc_L*(n-(M-1)./2))./(pi*(n-(M-1)./2));
 if(rem(M,2)~=0)
 hd1(((M-1)/2)+1)=wc_L/pi;
 end
 w_blackman=0.42-0.5*cos((2*pi*n)./(M-1))+0.08*cos((4*pi*n)./(M-1));
 h1=hd1.*w_blackman;
 wc_H=0.35*pi;
 hd2=sin(wc_H*(n-(M-1)./2))./(pi*(n-(M-1)./2));
 if(rem(M,2)~=0)
 hd2(((M-1)/2)+1)=wc_H/pi;
 end
 w_blackman=0.42-0.5*cos((2*pi*n)./(M-1))+0.08*cos((4*pi*n)./(M-1));
 h2=hd2.*w_blackman;
h2=-h2;
h2(((M-1)/2)+1)=h2(((M-1)/2)+1)+1;
h=conv(h1,h2);
fvtool(h,1);
```

b. Band-pass in a single stage

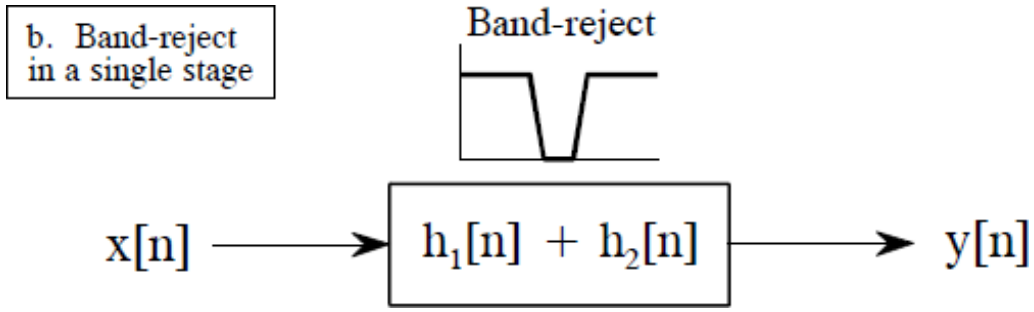


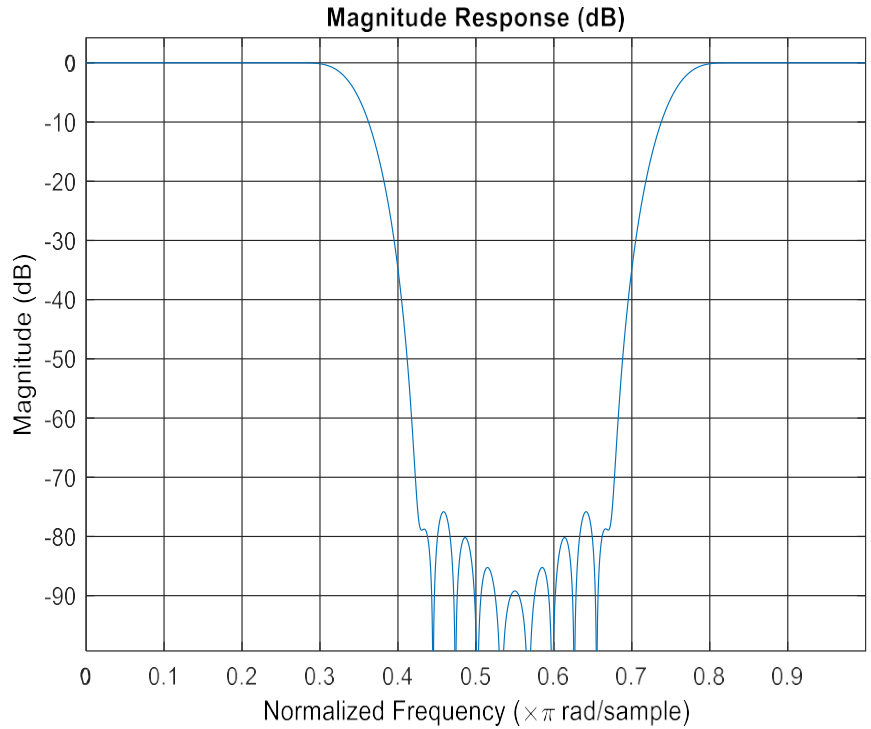


Band-reject filter from low and high pass filter kernel

Addition of low and high pass filter kernel

```
M=input('Define the length of
 filter kernel:'); n=[0:1:M-1];
 wc_L=0.35*pi;
 hd1=sin(wc_L*(n-(M-
 1)./2))./(pi*(n-(M-1)./2));
 if(rem(M,2)~=0)
 hd1(((
 M-
 1)/2)+
 1)=wc
 _L/pi;
 end
 w_blackman=0.42-0.5*cos((2*pi*n)./(M-
 1))+0.08*cos((4*pi*n)./(M-1)); h1=hd1.*w_blackman;
 wc_H=0.75*pi;
 hd2=sin(wc_H*(n-(M-
 1)./2))./(pi*(n-(M-1)./2));
 if(rem(M,2)~=0)
 hd2(((
 M-
```





Application of FIR Filter in signal separation

Application of FIR Filter in signal separation (MATLAB Code)

```
%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);
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');
```

Application of FIR Filter in signal separation (User-defined functions used in code)

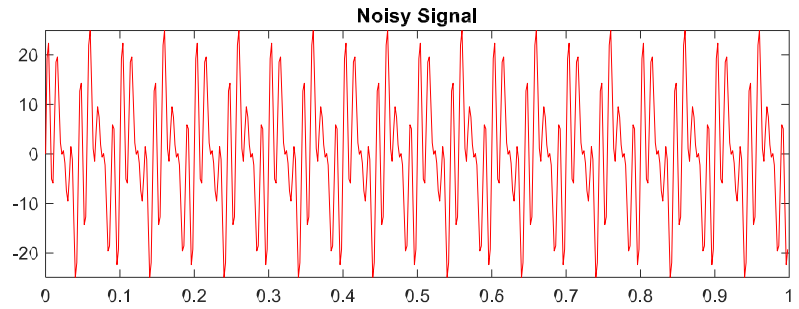
```
function h=fir_lowpass(ncf,M)
n=[0:1:M-1];
 wc=2*pi*ncf;
 hd=sin(wc*(n-(M-1)./2))./(pi*(n-(M-1)./2));
 if(rem(M,2)~=0)
 hd(((M-1)/2)+1)=wc/pi;
 end
 w_blackman=0.42-0.5*cos((2*pi*n)./(M-1))+0.08*cos((4*pi*n)./(M-1));
 h=hd.*w_blackman;


function h=fir_highpass(ncf,M)
if (rem(M,2)==0)
    M=M+1;
end
n=[0:1:M-1];
 wc=2*pi*ncf;
 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));
 h_low=hd.*w_blackman;
h=-h_low;
h(((M-1)/2)+1)=h(((M-1)/2)+1)+1;
```

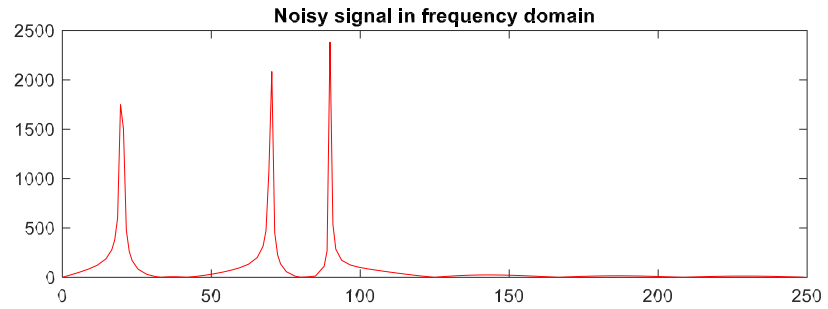
Application of FIR Filter in signal separation (User-defined functions used in code)

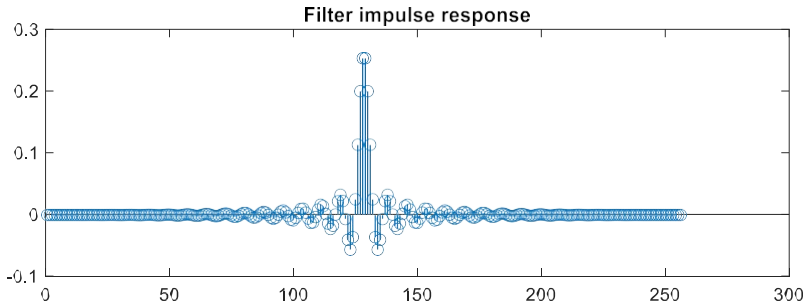
```
function h=fir_bandpass(ncf_low,ncf_high,M)
if (rem(M,2)==0)
    M=M+1;
end
%lowpass filter
n=[0:1:M-1];
 wc1=2*pi*ncf_high;
 hd=sin(wc1*(n-(M-1)./2))./(pi*(n-(M-1)./2));
 if(rem(M,2)~=0)
 hd(((M-1)/2)+1)=wc1/pi;
 end
 w_blackman=0.42-0.5*cos((2*pi*n)./(M-1))
             +0.08*cos((4*pi*n)./(M-1));
 h_low=hd.*w_blackman;

%highpass filter
 wc2=2*pi*ncf_low;
 hd=sin(wc2*(n-(M-1)./2))./(pi*(n-(M-1)./2));
 if(rem(M,2)~=0)
 hd(((M-1)/2)+1)=wc2/pi;
 end
 w_blackman=0.42-0.5*cos((2*pi*n)./(M-1))
                     +0.08*cos((4*pi*n)./(M-1));
 HL=hd.*w_blackman;
h_high=-HL;
h_high(((M-1)/2)+1)=h_high(((M-1)/2)+1)+1;
%bandpass filter
h=conv(h_low,h_high);
```

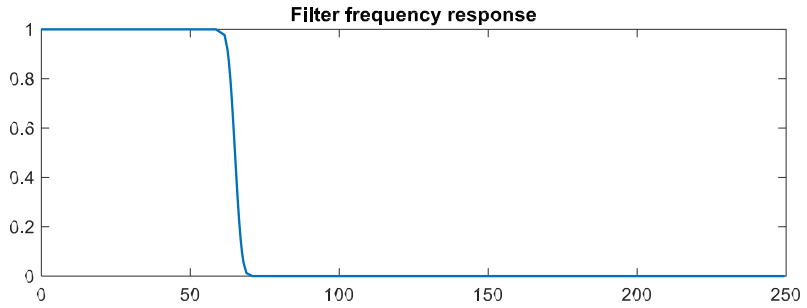
```
function h=fir_bandstop(ncf_low,ncf_high,M)
if (rem(M,2)==0)
    M=M+1;
end
%lowpass filter
n=[0:1:M-1];
 wc1=2*pi*ncf_low;
 hd=sin(wc1*(n-(M-1)./2))./(pi*(n-(M-1)./2));
 if(rem(M,2)~=0)
 hd(((M-1)/2)+1)=wc1/pi;
 end
 w_blackman=0.42-0.5*cos((2*pi*n)./(M-1))
             +0.08*cos((4*pi*n)./(M-1));
 h_low=hd.*w_blackman;
%highpass filter
 wc2=2*pi*ncf_high;
 hd=sin(wc2*(n-(M-1)./2))./(pi*(n-(M-1)./2));
 if(rem(M,2)~=0)
 hd(((M-1)/2)+1)=wc2/pi;
 end
 w_blackman=0.42-0.5*cos((2*pi*n)./(M-1))
                     +0.08*cos((4*pi*n)./(M-1));
 HL=hd.*w_blackman;
h_high=-HL;
h_high(((M-1)/2)+1)=h_high(((M-1)/2)+1)+1;
%bandstop filter
h=h_low+h_high;
```

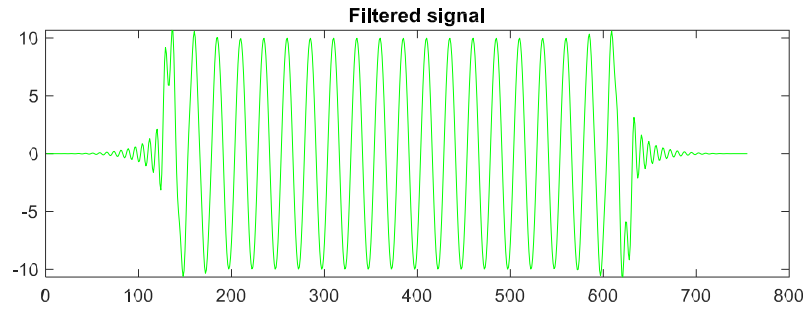
Signal Separation by FIR low-pass filter

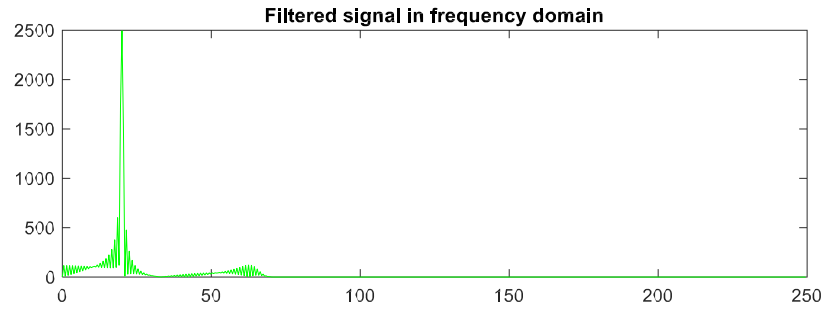




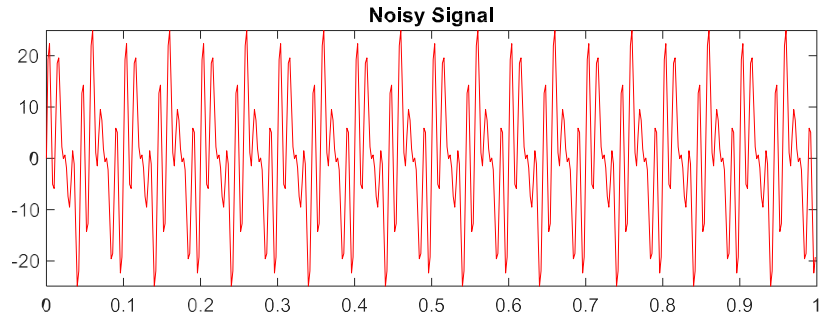


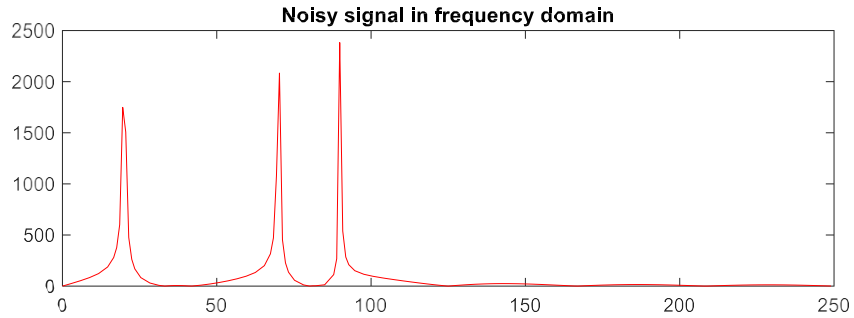


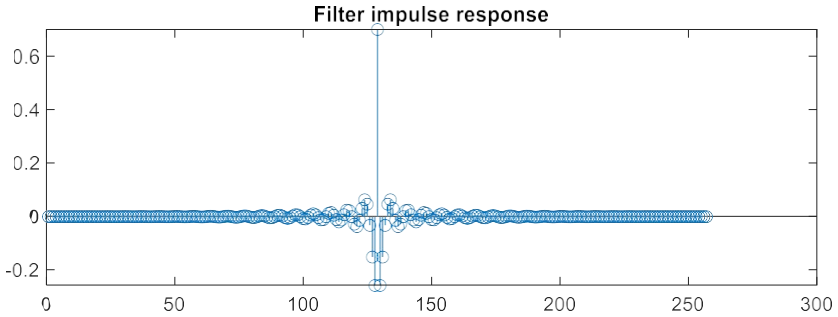


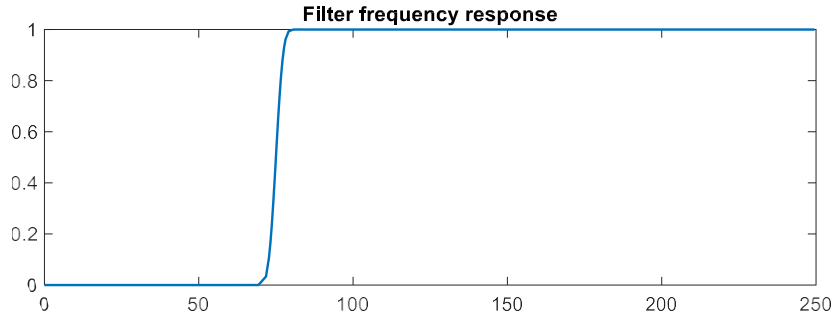


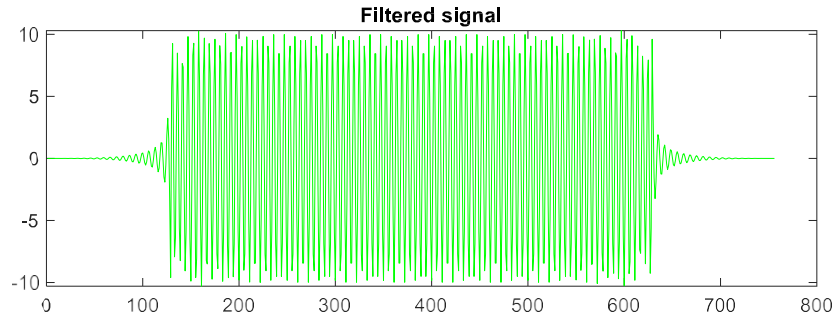
Signal Separation by FIR high-pass filter

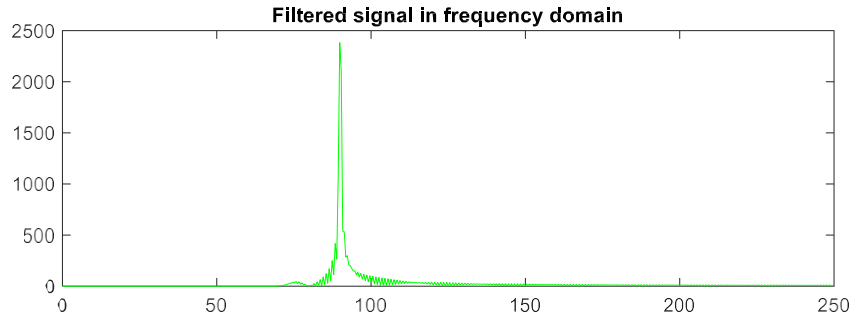




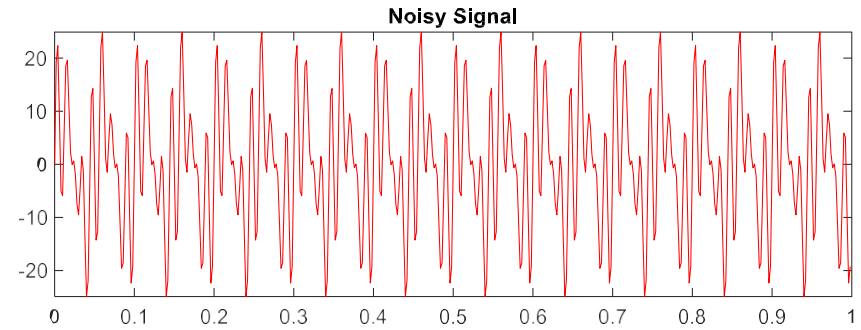


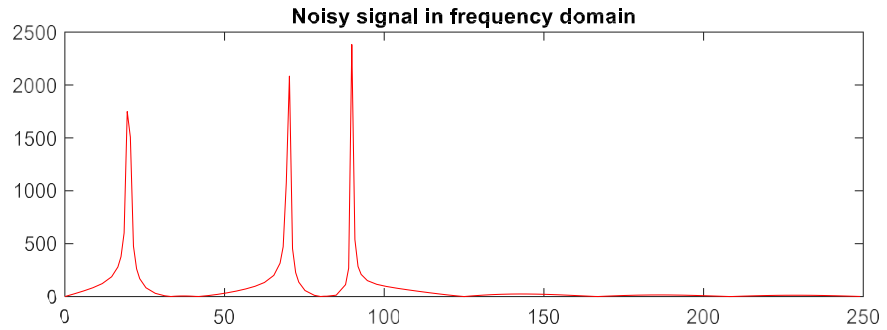


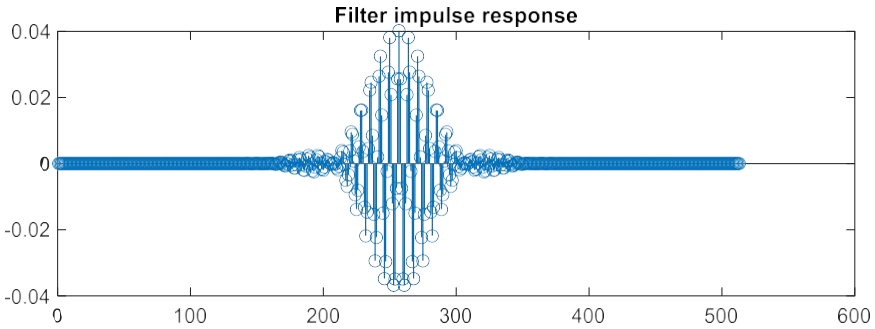


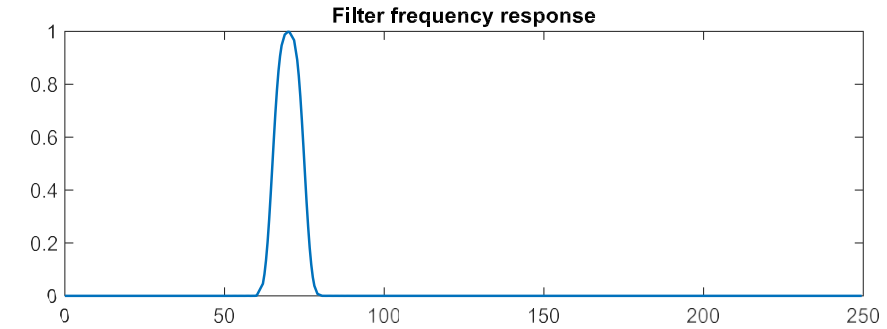


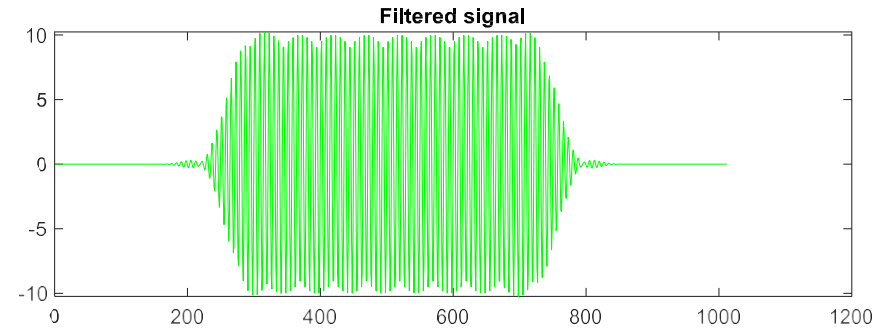
Signal Separation by FIR band-pass filter

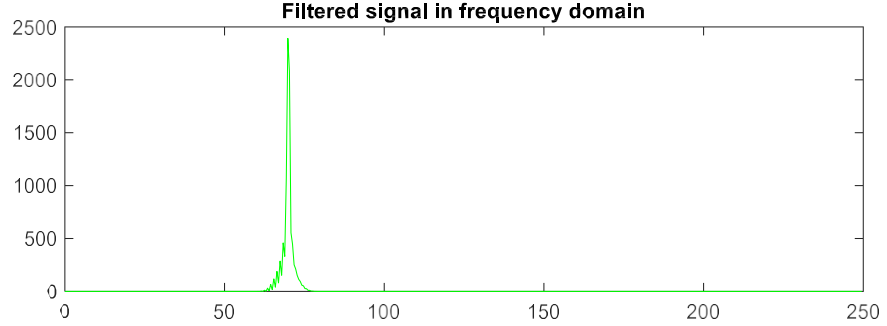












Signal Separation by FIR band-stop filter