Lab Manual

EEE 336

Digital Signal Processing

Department of Electrical and Electronic Engineering

Green University of Bangladesh

Table of Contents

General Instructions

1. The experiments are designed to illustrate about different areas of signal processing and image processing. Conduct the experiments with interest and an attitude of learning.
2. Students should come with thorough preparation for the experiment to be conducted. Students should come with proper dress code.
3. Students will not be permitted to attend the laboratory unless they bring the practical record fully completed in all respects pertaining to the experiment conducted in the previous class.
4. Work quietly and carefully (the whole purpose of experimentation is to make reliable measurements!) and equally share the work with your partners.
5. Be honest in recording and representing your data. If a particular output appears wrong repeat the process carefully.
6. All presentations of data, tables and graphs calculations should be neatly and carefully plotted.
7. Graphs should be plotted very carefully. Always mention legends the axes and display units.
8. If you finish early, spend the remaining time to complete the laboratory report writing. Come equipped with calculator, scales, pencils etc.
9. Handle lab computers with care.
10. Report any issue to the instructor.
11. Turn off all the computers after finishing the experiment.

Laboratory Regulations and Safety Rules

The following Regulations and Safety Rules must be observed in all concerned laboratory location.

1. Ensure that only licensed versions of MATLAB and associated toolboxes are used in the lab.
2. Handle computer hardware, peripherals, and accessories with care.
3. Log out of your user account after each session to protect your work.
4. Only work on assigned experiments or projects. Unauthorized use of lab resources for non-academic activities is prohibited.
5. Save your work frequently to avoid data loss. Use external drives or cloud services to back up data after the lab session.
6. Ensure all assignments and lab reports are submitted on time, as per the schedule provided by the instructor.

Experiment 1
Introduction to MATLAB

1.1 Objectives

1. Explain basic features configurations and applications of some signals in digital domain.
2. Familiar with the practical implementation of digital signals.
3. Explain different digital systems and their properties.
4. Explain a system through difference equation in digital domain.
5. Manipulate frequency response of a system in digital domain.
6. Justify the system transfer function through different transformation techniques in digital domain.
7. Recognize the applications of Time and Frequency domain analysis and advanced signal processing aspects.

1.2 Learning Outcomes

After completing this experiment, the students will be able to:

1. Explain different signals and systems in digital domain.
2. Explain features, configurations of various systems in digital domain.
3. Utilize system properties through Time-domain analysis and Frequency-domain analysis in digital domain.
4. Measure system transfer function from the different transformations analysis in digital domain.
5. Explain the applications of different transformations analysis in digital domain.
6. Recognize these concepts on courses like Advanced Numerical Methods, Control Systems, communication theory etc.

1.3 Theory

1.3.1 Starting MATLAB

1. You can start MATLAB R2020a on Microsoft Windows Platform (Win7/Win10) by double clicking the **MATLAB** shortcut icon on your windows desktop or simply click **MATLAB R2020a** from the start menu.
2. As an alternative method, click on the **Start** button then type **matlab** in the 'search field' then press **Enter**. MATLAB will start immediately.
3. After MATLAB starts, you can change the directory in which MATLAB saves your MATLAB files. To do this, click on **'…'** button then select the new location.

1.3.2 Desktop Tools

1. The Command Window: To enter variables, execute commands and to run M-files.
2. Menus:
    I. File: from the file menu, you can create a new M-file, figure…etc. You can also open any file and you can access the preferences of MATLAB.
    II. Edit: cut, copy, paste...etc.
    III. Desktop: to control the desktop of MATLAB.
    IV. *Tip*: to restore the default desktop go to Desktop →Desktop Layout→default
    V. Window: to get access to the windows/files e.g. the open M-files documents.
3. The Current Directory Browser: any files you want to run must either be in the current directory or on search path. The current directory browser enables you to browse all the files saved in the current directory. You can run, rename, delete…etc.
4. Command History: in the command history you can view the previously used functions and copy and execute selected lines.
5. Lunch Pad: provides easy access to tools, demos, and documentations.

6. Help Browser: to search and view documentations for all your **MathWorks** products.
7. Workspace Browser: the MATLAB workspace consists of the set of variables (named arrays) built up during a MATLAB session and stored in memory. To view the workspace and information about each variable, use the Workspace Browser, or use the commands **who** and **whos.**
8. Array Editor: double click on a variable in the Workspace Browser to see it in the Array Editor.
9. Editor/Debugger: to create and debug M-files.

1.3.3 M-Files

M-files provide an easy way to write and excite your commands and programs. For a large number of commands and complex problem-solving M-files is a must. It allows you to place MATLAB command in a simple text file and then tell MATLAB to open the file and execute the commands exactly as it would if you typed them at the MATLAB Command Window.

1.3.4 M-Files must end with the extension '.m'. For example, homework1.m

There are many ways to load M-file Editor:

1. Click on start → programs →MATLAB R2020a →M-file Editor
2. From the MATLAB, chose New from the Home tab and select Script.

1.4 List of Equipment:

1. Desktop PC
2. Software MATLAB R2020a

1.5 Procedure

First Steps in MATLAB

When MATLAB starts, the special >> prompt (the command line) appears, MATLAB is ready to receive your commands. Try to compute c=a+b, where a=10, and b=20.

Table 1.1: (General Matlab Commands)

MATLAB Command	Details
X=[1:10]	Simple matrix generation Ans: 1 2 3 4 5 6 7 8 9 10
X=[1:1:10]	single row matrix generation. [start: step: End] Ans: 1 2 3 4 5 6 7 8 9 10
X= [1:2:10]	Ans: 1 3 5 7 9
Y=linspace(0,10,5)	Single row matrix generation. Formation by [start, end, no of data]
T=2*pi*50	for MATLAB, $\pi = pi$, T=2 πf
sin(30*pi/180)	MATLAB angle values are in Radians. Radians to Degree conversion- $Degree = \frac{180}{\pi}$* Radian
asin(1/2)*180*pi	inverse trigonometry
10*exp(-2)	'exp' means exponential; $10e^{-2}$
9e-5	'e' denoted power of 10; $9 * 10^{-5}$
date	date on MATLAB
calendar	calendar command on MATLAB
(log10(10))^4	Logarithm Command on MATLAB- logBasePower; Example: $\log_{10} 10$ log10 means= $\log_e 10$ (default)
(log2(5))^4	2 Base Logarithm
(log(10))^4	e base logarithm
X= 0:2:16; Y=2*X;	Matrix Algebraic Calculations
T=linspace(0,2*pi*100);	Plotting on MATLAB

X=sin(T); Y=cos(T); plot(X) plot(Y)	plotting X and Y respect to T
subplot(3,1,1) plot(X) subplot(3,1,2) plot(Y) subplot(3,1,3) plot(X,Y)	Subplot 1,1 1,2 1,3 2,1 2,2 2,3 3,1 3,2 3,3 Formation of subplot(Row, column, position)
plot(X,'-r') hold on plot(Y,'-b') hold on plot(X,Y,':k')	Styling of graph- plot(X,'linestyle_charecter_color_code') "magenta"----------"m" "Yellow"---------- "y" "black"---------- "k"
legend('sin','cos',circle,0)	Graph notation

Matrix Operations

To type a matrix into MATLAB you must:

- Begin with a "square bracket ([ )".
- Separate elements in a row with commas or spaces.
- Use a semicolon (;) to separate rows
- End the matrix with another "square bracket ( ])".

Table 1.2: Matlab Commands for Matrix

Operation	Matrix Command	Details
Simple Matrix	a=[1,-2,0;10,-6,2;1,11,-9] or a=[1 -2 0; 10 -6 2; 1 11 -9]	$\begin{matrix} 1 & -2 & 0 \\ 10 & -6 & 2 \\ 1 & 11 & -9 \end{matrix}$

Matrix Operation	A=[1 10 200 ; -50 0 -20 ; 44 25 60]; B=[1 15 -32 ; 14 20 20 ; 2 5 3]; C=[1 1 1 ; 25 -6 -6; 14 89 300]; A+B+C A+B-C A+B-C-10 A/C A*B*C 2*A A^2 A.^2	
Matrix Generation	zeros(3,3) ones(3,2) rand(3,3)	
Matrix Index	B=[0 20 90; 12 -34 45] B(2,3) B(1,2) B(2,2)	
Transpose, Determinant and Inverse Matrix	W=[1 2 3; -4 -5 -6; 0 1 0] W' det(W) inv(W)	W'=Transpose Matrix det(W)=Determinant of W inv(W)= inverse matrix of W
For Loop	for i = 1:2:10 x = i^2 end	*For index = expression* *Statement group x* *End* The output of this code: x = 1, 9,25,49,81.

Nested For Loop	m=3;
	n=5;
	for i=1:m
	for j=1:n
	f(i,j)=i;
	end
	end
	f
		The output of this code: f =
		1 1 1 1 1
		2 2 2 2 2
		3 3 3 3 3

<table>
<tr>
<td>While Loop</td>
<td>i=1
d=0
while i<5 % i>1
d=i+1
i=d
end
disp('loop ended')
d
i</td>
<td>Constant value 1 (i)
constant value 2 (d)
while condition
operation
end

The output of this code: d=5, i = 5</td>
</tr>
<tr>
<td>Relation
Operator & If
Else
Statement</td>
<td>a=5;
b=6;
if a~=b
disp('unequal')
else
disp('equal')
end</td>
<td>
<table>
<tr><td>Rational</td><td>description</td></tr>
<tr><td><</td><td>Less than.</td></tr>
<tr><td><=</td><td>Less than or equal.</td></tr>
<tr><td>></td><td>Greater than.</td></tr>
<tr><td>>=</td><td>Greater than or</td></tr>
<tr><td>= =</td><td>Equal to.</td></tr>
<tr><td>~ =</td><td>Not equal to.</td></tr>
</table>
</td>
</tr>
<tr>
<td>Logical
Operator</td>
<td>x=4; y=6;
if x<1 & y<1
z=0
elseif x>1 & y<1
z=1
elseif x>1 | y<1
z=2
end</td>
<td><u>Logical Operators with IF Else Statement:</u>
Symbol Meaning
& AND
| OR
~ NOT

The output of this code: z = 2</td>
</tr>
<tr>
<td>Solving
Linear
Equation</td>
<td>$3x + 10y - z = 0$
$-2x + y - 10z = 2$
$x + y - z = 3$

Method 1
A=[3 10 -1;-2 1 -10; 1 1 -1];
b=[0;-2;3];
x=inv(A)*b
x=A\b

Method 2
syms x y z
[x,y,z]=solve([3*x+10*y-z==0,-2*x+y-10*z==-2,x+y-z==3],[x,y,z])</td>
<td>The output of this code: x = 3.5000, -1.1111, -0.6111</td>
</tr>
</table>

1.6 Student Works

1. Think about A and B are the last two digits of your ID-
   a. $\sin(AB°) + \cot(BA°) + \tan^{-1}\frac{B}{A} + Be^{-BA} + A * 10^{-A} + (\log_{10} B)^{AB} + (\log_e BA)^A$
2. Solve the following equation-
   a. $17x_1+2x_2+1x_3+5x_4=4$

      $5x_1+6x_2+7x_3+1x_4=-1$

      $9x_1-10x_2+11x_3+12x_4=10$

      $13x_1+14x_2+15x_3-9x_4 = 6$

   Ans: $x_1 = 0.5353$, $x_2 = -0.9497$, $x_3 = 0.3914$, $x_4 = -0.7184$

1.7 Report Questions

1. $\sin(225°) + \cot(30°) + \tan^{-1}\left(\frac{1}{2}\right) + 10e^{-10} + 9 * 10^{-2} + (\log_{10} 10)^3 + (\log_e 10)^5$
2. Try the following commands for the matrix W

$$W = \begin{bmatrix} 10 & -12 & 30 \\ 55 & 95 & 200 \\ -70 & 5 & 2 \end{bmatrix}$$

   A=W(3,2)

   B=W(1,1)

   C=W(3,3)

   D=W(2,2)

   E=A+B

   $F = \sqrt{ln|C - D|} + \sin(B)$

1.8 References

1. Digital Signal Processing: Principles, Algorithms, and Applications, 4th Edition, Dimitris G. Manolakis and John G Proakis
2. Oppenheim, A. V., & Schafer, R. W. (2010). *Discrete-Time Signal Processing*. Pearson.

Experiment 2
Discrete-Time Signals and Systems- Part I

2.1 Objectives

1. Study and observation of basic signal generation using MATLAB.

2.2 Learning Outcomes

After completing this experiment, the students will be able to:

1. Able to generate different signals as needed in MATLAB.

2.3 Theory

Any discrete signal is normally denoted by x(n), in which the variable n is integer-valued and represents discrete instances in time. Therefore, it is also called a discrete-time signal, which is a number sequence and will be denoted by one of the following notations:

$$x(n) = \{x(n)\} = \{\ldots, x(-1), \underset{\uparrow}{x(0)}, x(1), \ldots\}$$

where the up-arrow indicates the sample at time instances, n = 0

In MATLAB a finite-duration sequence is represented by a row vector of appropriate values. However, such a vector does not have any information about sample position n. Therefore, a correct representation of x(n) would require two vectors, one each for x and n. For example, a sequence

$$x(n) = \{3, 2, -2, \underset{\uparrow}{1}, 5, -3, 6\}$$ can be represented in MATLAB by,

- n = [-3 -2 -1 0 1 2 3];
- x = [3 2 -2 1 5 -3 6];

Generally, we will use the x-vector representation alone when the sample position information is not required.

2.4 List of Equipments:

1. Desktop PC
2. Software MATLAB R2020a

2.5 Procedure

Here in this experiment we will implement some elementary sequences in MATLAB

2.5.1 Some Elementary Sequences

2.5.1.1 Unit sample sequence

In MATLAB the function *Zeros(1, N)* generates a row vector of N zeros, which can be used to implement δ(n) over a finite interval. However, the logical relation (n = = 0) is an elegant way of implementing δ(n). For example, we can use the following function to define a new function impseq to generate an impulse sequence δ(n), in the time range n1 <= n <= n2

```
function [x, n] = impseq(n0,n1,n2)
[x,n] = imseq(n0,n1,n2)
n = [n1:n2];
x = [(n-n0) == 0];
end
```

2.5.1.2 Unit step sequence:

In MATLAB the function *ones(1, N)* generates a row vector of N ones, which can be used to generate u(n) over a finite interval. Once again, the logical relation (n >= 0) is an elegant way of implementing the unit step sequence, u(n). For example, we can use the following codes to implement the unit step sequence, using the function, *stepseq()*

```
function [x, n] = impseq(n0,n1,n2)
```

```
[x,n] = imseq(n0,n1,n2)
n = [n1:n2];
x = [(n-n0) >= 0];
end
```

2.5.1.3 Real-valued exponential sequence:

To generate $x(n) = (0.9)^n$, $0 <= n <= 10$, the MATLAB script will be:

```
n = [0:10];
x = (0.9) .^ n;
```

2.5.1.4 Complex-valued exponential sequence

To generate $x(n) = \exp[(2+j3)n]$, $0 <= n <= 10$, the MATLAB script will be:

```
n = [0:10];
x = exp((2+3j) * n);
```

2.5.1.5 Sinusoidal sequence

$x(n) = \cos(\omega_0 n + \theta)$ where θ is the phase in radian

A MATLAB function cos (or sin) is used to generate sinusoidal sequences. For example, to generate $x(n) = 3\cos(0.1\pi n + \pi / 3) + 2\sin(0.5\pi n)$, $0 \le n \le 10$, the following MATLAB script can be used:

```
>>  n=[0:10];
>>  x=3*cos(0.1*pi*n + pi/3) + 2*sin(0.5*pi*n);
```

2.5.1.6 Random sequences

In MATLAB two types of random sequences are available. The *rand(1, N)*generates a length N random sequences whose elements are uniformly distributed between [0, 1]. The *randn(1, N)* generates a length N Gaussian random sequence with mean 0 and variance 1. Other random sequences can be generated using transformation of the above function.

2.5.1.7 Periodic sequence

A sequence x(n) is periodic if x(n) = x(n + N) . The smallest integer N that satisfies the above relation is called the fundamental period. To generate P periods of x%(n) from one period { x (n), 0 ≤ n ≤ N-1}, we can copy x(n) P times:

```
>> xtilde=[x,x,..., x];
```

But an elegant way approach is to use MATLAB's powerful indexing capabilities. First, we generate a matrix containing P rows of x(n) values. Then we can concatenate P rows into a long row vector using the construct (:). However, this construct works only on columns.

```
xtilde = x' * ones(1,P); % P columns of x; x is a row vector
xtilde = xtilde(:);   % long column vector
xtilde = xtilde';         % long row vector
```

2.6 Student Works

1. Generate and plot each of the following sequences over the indicated interval

   a. x (n) = 2δ (n + 2) −δ (n −4), −5 ≤ n ≤ 5.

   b. x(n) = n[u(n) −u(n −10)] +10e−0.3(n−10)[u(n-10)-u(n-20)], 0 ≤ n ≤ 20.

   c. x (n) = cos(0.04πn) +0.2w(n), 0 ≤ n ≤ 50, where w(n) is a Gaussian random sequence with zero mean and unit variance.

   x (n) = { ...,5, 4,3, 2,1,5, 4,3, 2,1,5, 4,3, 2,1,...}; −10 ≤ n ≤ 9.
   ↑

2.7 Report Questions

1. x (n) ={1, 2,3, 4,5,6,7,6,5, 4,3, 2,1}. Determine and plot the following sequence.
   ↑

   a. x1 (n) = 2 x(n −5) −3x(n +4)

   b. x2 (n) = x (3 −n) + x (n) x (n −2)

2.8 References

1. Digital Signal Processing: Principles, Algorithms, and Applications, 4th Edition, Dimitris G. Manolakis and John G Proakis
2. Oppenheim, A. V., & Schafer, R. W. (2010). *Discrete-Time Signal Processing*. Pearson.

Experiment 3
Discrete-Time Signals and Systems- Part II

3.1 Objectives

1. Study and observation of basic signal generation using MATLAB.

3.2 Learning Outcomes

After completing this experiment, the students will be able to:

1. Able to generate different signals as needed in MATLAB.

3.3 Theory

In the last laboratory experiment, we learned how to define some basic signal sequences, such as impulse sequences and unit step sequences. This experiment focuses on performing basic operations on signals, including addition, multiplication, scaling, time shifting, folding, sample summation, and sample products.

3.3.1 Signal addition

Signal addition involves a sample-by-sample addition of two signals. This is mathematically given by:

$$\{x1\ (n\ )\} + \{x2\ (n)\} = \{\ x1\ (n) + \ x2\ (n)\} \tag{3.1}$$

The important note is that the lengths of the two sequences must be the same.

3.3.2 Signal Multiplication

Signal multiplication is a sample-by-sample operation given by:

$$\{x_1\ (n\ )\}.\{x_2\ (\ n)\} = \{x_1\ (\ n) x_2\ (\ n)\} \tag{3.2}$$

3.3.3 Scaling or Multiplication of a Sequence by a Scalar

Scaling multiplies each sample value of the sequence by a scalar α:

$$\alpha\{x(n)\} = \{\alpha x(n)\} \tag{3.3}$$

3.3.4 Time Shifting of Sequences

In time shifting, each sample of x(n) is shifted by an amount k to obtain y(n):

$$y(\,n\,) = \{x(n\, -k\,)\} \tag{3.4}$$

This operation only modifies the vector n by adding k to its elements.

3.3.5 Folding of Sequences

Folding involves flipping the sequence x(n) around n=0 to obtain y(n):

$$y(\,n\,) = \{x(-n)\} \tag{3.5}$$

3.3.6 Sample Summation

Sample summation adds all sample values of x(n) between two indices n_1 and n_2:

$$\sum_{n=n_1}^{n_2} x(n) = x(n_1) + \cdots + x(n_2) \tag{3.6}$$

3.3.7 Sample Products

Sample products multiply all sample values of x(n) between two indices n_1 and n_2:

$$\prod_{n=n_1}^{n_2} x(n) = x(n_1) * \ldots * x(n_2) \tag{3.7}$$

3.3.8 Signal Energy

The energy of a sequence x(n) is computed as:

$Ex=\sum|x(n)|2$

$$x = \sum_{-\infty}^{\infty} x(n)x^*(n) = |\sum_{-\infty}^{\infty} x(n)|^2 \quad (3.8)$$

3.3.9 Signal Power

The average power of a periodic sequence x(n) with a fundamental period N is given by:

$$p = \frac{1}{N}\sum_{n=0}^{N-1}|x(n)|^2 \quad (3.9)$$

3.4 List of Equipments:

1. Desktop PC
2. Software MATLAB R2020a

3.5 Procedure

3.5.1 Signal addition

It is implemented in MATLAB by the arithmetic operator "+". However, the important thing to note is that, lengths of the two sequences must be the same. Now we will create a new function of our own, *sigadd(),* which will add two signal sequences of arbitrary duration. The steps are as follows.

```
function [y,n] = sigadd(x1,n1,x2,n2)
implements y(n) = x1(n)+x2(n)
[y,n] = sigadd(x1,n1,x2,n2)
y = sum sequence over n, which includes n1 and n2
x1 = first sequence over n1
x2 = second sequence over n2 (n2 can be different from n1)
n = min(min(n1), min(n2)) : max(max(n1),max(n2)); % duration of y(n)
  y1 = zeros(1, length(n));
y2 = y1;
```

```
y1(find((n>=min(n1)) & (n<=max(n1))==1));
y2(find((n>=min(n2)) & (n<=max(n2))==1));
  y = y1 + y2;
```

3.5.2 Signal multiplication:

When we need to multiply two signal sequences, we need sample-by-sample multiplication (or "dot" multiplication) given by

It is implemented in MATLAB by the array operator "**.***". Once again, the similar restrictions apply for the .* operator as the + operator. Therefore, we can use the following *sigmult()* function, designed by us and only to serve our purpose.

```
function [y,n] = sigmult(x1,n1,x2,n2)
implements y(n) = x1(n)*x2(n)
[y,n] = sigmult(x1,n1,x2,n2)
End
y = product sequence over n, which includes n1 and n2
x1 = first sequence over n1
x2 = second sequence over n2 (n2 can be different from n1)
n = min(min(n1), min(n2)) : max(max(n1),
y1 = zeros(1, length(n)); y2 = y1;
y1(find((n>=min(n1)) & (n<=max(n1))==1))
y2(find((n>=min(n2)) & (n<=max(n2))==1))
y = y1 .* y2;
```

3.5.3 Scaling or multiplication of a sequence with a scaler

An arithmetic operator "*" is used to implement the scaling operation in MATLAB which is given below.

```
y = alpha * x;
```

3.5.4 Time Shifting of sequences

In this operation each sample of x(n) is shifted by an amount 'k' to obtain a shifted sequence y(n). If we let m = n-k, then n = m+k and the eq. (3.4) becomes

$$y(m+k)=\{x(m)\} \tag{3.11}$$

Hence this operation has no effect on the vector x, but the vector n is changed by adding k to each element. Now we will define a function named *sigshift()* to do this shifting operation.

```
function [y, n] = sigshift(x,m,n0)
implements y(n) = x(n-n0)
[y, n] = sigshift(x,m,n0)
 n = m+n0;
 y = x;
```

3.5.5 Folding of sequences

In this operation each sample of x(n) is flipped around n = 0 to obtain a folded sequence y(n). In MATLAB this operation is implemented by *fliplr(x)* function for sample values and by *-fliplr(n)* function for sample positions as shown in the *sigfold* function of our design.

```
function [y, n] = sigfold(x,n)
implements y(n) = x(-n)
[y, n] = sigfold(x,n)
end
y = fliplr(x);
n = -fliplr(n);
```

3.5.6 Sample summation

This operation differs from signal addition operation. It adds all sample values of x(n) between n_1 and n_2. It is implemented by the *sum(x(n_1:n_2))* a built in MATLAB function.

3.5.7 Sample products

This operation also differs from signal multiplication operation. It multiplies all sample values of x(n) between n_1 and n_2. It is implemented by the *prod(x(n1:n2))* a built in MATLAB function.

3.5.8 Signal energy

The energy of a sequence x(n) is mentioned in eq. (3.9) where superscript * denotes the operation of complex conjugate. The energy of a finite duration sequence x(n) can be computed in MATLAB using the following approaches:

```
Ex = sum(x .* conj(x)); % one approach
Ex = sum(abs(x) .^2); % another approach
```

3.6 Student Work

1. Generate and plot each of the following sequences over the indicated interval. Also calculate energy and power of these signals.
   i. $x(n) = 2\delta(n+2) - \delta(n-4), \quad -5 \le n \le 5.$
   ii. $x(n) = n\,[u(n) - u(n-10)] + 10\, e^{-0.3(n-10)}\,[u(n-10) - u(n-20)],$
   iii. $x(n) = \cos(0.04\pi n) + 0.2w(n), \quad 0 \le n \le 50,$ where w(n) is a Gaussian random sequence with zero mean and unit variance. $-10 \le n \le 9.\uparrow$

3.7 Report questions

1. $x(n) = \{1, 2, 3, 4, 5, 6, 7, 6, 5, 4, 3, 2, 1\}$. Determine and plot the following sequence.
   $\uparrow$
   i. $x_1(n) = 2x(n-5) - 3x(n+4)$
   ii. $x_1(n) = 2x(n-5) - 3x(n+4)$
2. $x(n) = \{1, 2, 3, 4, 5, 6, 7, 6, 5, 4, 3, 2, 1\}$. Determine and plot the following sequence.
   $\uparrow$

   $x_2(n) = x(3-n) + x(n)\,x(n-2)$

3.8 References

1. Digital Signal Processing: Principles, Algorithms, and Applications, 4th Edition, Dimitris G. Manolakis and John G Proakis
2. Oppenheim, A. V., & Schafer, R. W. (2010). *Discrete-Time Signal Processing*. Pearson.

Experiment 4
Sampling and Reconstruction of Analog Signals

4.1 Objectives

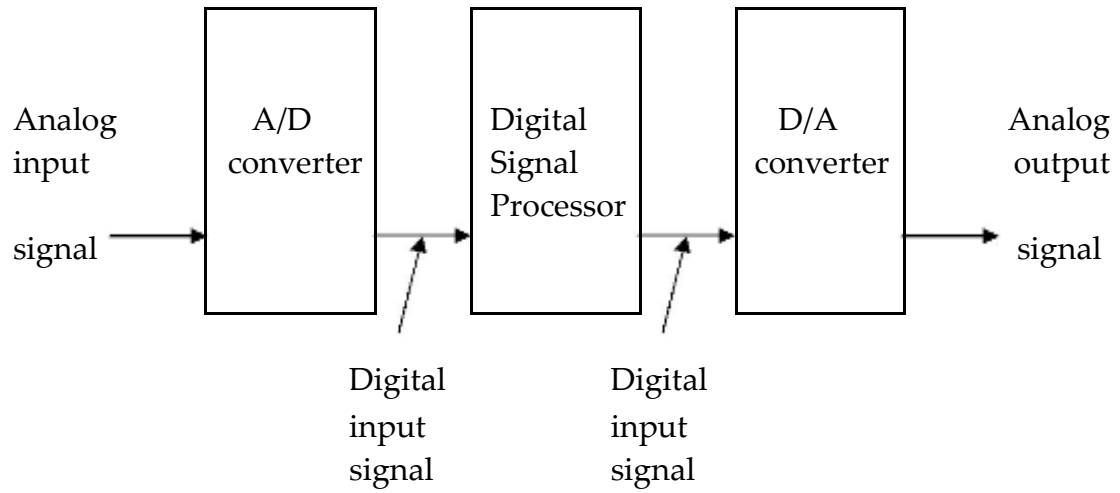
1. In this experiment we will learn few digital filter design techniques

4.2 Learning Outcomes

After completing this experiment, the students will be able to:

1. To be able to design digital filter for a given specification

4.3 Theory



In many DSP applications, real world analog signals are converted into discrete signals using sampling and quantization operations (collectively called analog-to-digital conversion or ADC). These discrete signals are processed by digital signal processors, and the processed signals are converted into analog signals using a reconstruction operation (called digital-to-analog conversion or DAC). To understand how the DSP system works, we need to know the relation between an analog signal and its discrete time sampled version. In time domain, relation between an analog signal and a sampled discrete time signal is given by

$$x(n) = x_a(nT_s) \tag{4.1}$$

where T_s is sampling interval. However, in the frequency domain, the relation between spectra of an analog signal and its discretized version is more complicated. Here we can use using Fourier analysis to explain this relation and then address the reconstruction operation as follows:

The continuous time Fourier transform is given by,

$$X_a(F) = \int_{-\infty}^{\infty} x_a(t) e^{-j2\pi Ft} dt \tag{4.2}$$

The inverse continuous time Fourier transform is given by

$$X_a(F) = \int_{-\infty}^{\infty} X_a(t) e^{-j2\pi Ft} dt \tag{4.3}$$

The discrete time Fourier transform is given by

$$X(f) = \sum_{n=-\infty}^{\infty} x(n) e^{-j2\pi fn} \tag{4.4}$$

The inverse discrete time Fourier transform is given by

$$x_a(t) = \int_{-\infty}^{\infty} X_a F \, e^{-j2\pi Ft} dF \tag{4.5}$$

The relation between $X_a(F)$ and $X(f)$ is given by

$$X(f) = F_s \sum_{k=-\infty}^{\infty} X_a(f - k) F_s \tag{4.6}$$

where F_s is a sampling frequency = $1/T_s$. In other words, $X(f)$ consists of infinite numbers of copies of scaled $X_a(F)$ separated by frequency interval $f = 1$. From the relation between discrete time and analog frequencies

$$f = \frac{F}{Fs} \tag{4.7}$$

we get,

$$X\left(\frac{F}{Fs}\right) = Fs \sum Xa(F - kFs) \tag{4.8}$$

4.3.1 Sampling Principle

In order to avoid aliasing, a band-limited signal $x_a(t)$ with bandwidth B can be reconstructed from its sample values $x(n) = x_a(nT_s)$ if the sampling frequency $F_s = 1/T_s$ is greater than twice the bandwidth B of $x_a(t)$.

$$F_s > 2B \tag{4.9}$$

Otherwise, aliasing would result in $x(n)$. The sampling rate of $2B$ for an analog band-limited signal is called the Nyquist rate.

4.3.2 Reconstruction

From the sampling theorem and the above examples, it is clear that if we sample band-limited $x_a(t)$ above its Nyquist rate, then we can reconstruct $x_a(t)$ from its samples x(n). Using an interpolation formula:

$$x_a(t) = \sum_{n=-\infty}^{\infty} x(n)\ sinc\big(F_s(t - nT_s)\big) \tag{4.10}$$

Where, $sinc(x) = \frac{\sin(\pi x)}{\pi x}$ is an interpolating function derived from an ideal low pass reconstruction filter. However, since an ideal low pass reconstruction filter cannot be implemented, we usually estimated the ideal low pass filter by the following methods:

4.3.2.1 Zero-order-hold (ZOH) interpolation

1. In this interpolation a given sample value is held for the sample interval until the next sample is received.

$$x_a(t) = x(n),$$
$$nT_s \le t < (n+1)T_s$$

which can be obtained by filtering the impulse train through an interpolating filter of the form which is a rectangular pulse. The resulting signal is a piecewise-constant

(staircase) waveform which requires an appropriately designed analog post-filter for accurate waveform reconstruction.

2. First-order-hold (FOH) interpolation

(a) In this case the adjacent samples are joined by straight lines. This can be obtained by filtering the impulse train through,

$$h(t) = 1 + \frac{t}{Ts} \qquad ; 0 \le t \le Ts$$

$$h(t) = 1 - \frac{t}{Ts} \qquad ; Ts \le t \le 2Ts$$

otherwise, $h(t) = 0$

4.4 List of Equipments:

1. Desktop PC
2. Software MATLAB R2020a

4.5 Procedure

To study the effect of sampling on the spectrum of the discrete signal, we will sample at two different sampling frequencies and then reconstruct the signals as follows:

Let $x_a(t) = e^{-10\,t}$. The continuous time Fourier transform is given by

$$X_a(F) = \frac{2 * 10}{10 + (2\pi F)^2} \qquad (4.11)$$

Sampling and Reconstruction procedures for $x_a(t) = e^{-10|t|}$.

1. Sampling $x_a(t)$ at F_s = 50 samples/sec. :

```
clear all
tmin = -1;
tmax = 1;
% Analog signal
```

```
t = tmin:0.001:tmax;
xa = exp(-10*abs(t));
Sampling rate (sample/second) Fs = 50
Sample period
Ts = 1/Fs
Discrete time signal n = tmin/Ts:tmax/Ts;
x = exp(-10*abs(n*Ts)); %Display signals in time domain figure(1)
subplot(211)
plot(t,xa)
title('Analog and discrete time signals') xlabel('time (sec)')
ylabel('Analog signal x(t)') subplot(212)
stem(n,x)
xlabel('n')
ylabel('Discrete time signal x(n)')
Computing Fourier transform
Analog frequency (Hert) F = -100:0.1:100;
W = (2*pi*F);
%Discrete time frequency (Circle/sample) f = F/Fs;
w = 2*pi*f;
%Analog spectrum for continuous time Fourier transform XaF =
2.*(10./(10^2+W.^2));
%Discrete time fourier transform XF = x*exp(-j*n'*w);
%Display spectra in frequency domain figure(2)
subplot(311)
plot(F,abs(XaF)) title('Spectra of signals') xlabel('Freq (circle/sec)')
ylabel('Original Xa(F)') subplot(312) plot(F,abs(XF)) xlabel('Freq
(circle/sec)') ylabel('X(F/Fs)') subplot(313) plot(f,abs(XF))
xlabel('Freq (circle/sample)') ylabel('X(f)')
Display spectra in the fundamental range
```

```
figure(3)
subplot(211)
plot(F,abs(XF))
title('Spectra in the fundamental range') xlabel('Freq (circle/sec)')
ylabel('X(F/Fs)')
v = axis;
v(1:2) = [-Fs/2 Fs/2]; axis(v)
subplot(212)
plot(f,abs(XF))
xlabel('Freq (circle/sample)') ylabel('X(f)')
v = axis;
v(1:2) = [-1/2 1/2]; axis(v)
```

Explain whether experimental results are consistent with theoretical results.

2. Reconstruction of xa(t):

```
t = tmin:0.001:tmax;
figure(4)
subplot(211)
hold on
stem(n*Ts,x,'r')
for i = 1:size(x,2)
xsinc(i,:) = x(i)*sinc(Fs*(t -(i+min(n)-1)*Ts)); plot(t,xsinc(i,:))
end
title('Signal reconstruction') xlabel('time (second)')
ylabel('x(n)*Sinc(Fs*(t-nTs))')
hold off
xar = sum(xsinc);
subplot(212)
plot(t,xar,'b-',t,xa,'r:')
```

```
legend('Reconstructed signal','Original signal') ylabel('Reconstructed signal xa(t)')
xlabel('time (second)')
maxerror = max(abs(xa - xar));
```

Explain the property of a *sinc* function that can use to reconstruct the signal without interfering other *sinc* functions.

3. Repeat step 1 and 2 using Fs = 10 samples/sec. Explain that why using Fs = 50 samples/sec is better than using Fs = 10 samples/sec. Locate the area where the spectra are most likely to overlap other resulting in aliasing.

For each T_s plot x(n). Reconstruct the analog signal $y_a(t)$ from the samples x(n) using the *sinc* interpolation (use $\Delta t = 0.001$) and determine the frequency in $y_a(t)$ from your plot. Comment on your results.

4.6 Student Works

Consider an analog signal $x_a(t) = \sin(20\pi t)$, $0 \le t \le 1$. It is sampled at Ts = 0.01, 0.03, 0.05 and 0.1 sec intervals to obtain x(n). For each T_s plot x(n). Reconstruct the analog signal $y_a(t)$ from the samples x(n) using the *sinc* interpolation (use $\Delta t = 0.001$) and determine the frequency in $y_a(t)$ from your plot. Comment on your results.

4.7 Report questions

1. What is the MATLAB function that would be used to plot a staircase (ZOH) interpolation of the analog signal.
2. What is the MATLAB function that would be used to plot a linear (FOH) interpolation of the analog signal.
3. From the experiment, describe why the minimum sampling rate must be at least twice the bandwidth of an analog signal

4.8 References

1. Digital Signal Processing: Principles, Algorithms, and Applications, 4th Edition, Dimitris G. Manolakis and John G Proakis
2. Oppenheim, A. V., & Schafer, R. W. (2010). *Discrete-Time Signal Processing*. Pearson.

Experiment 5
Introduction to Z-transform

5.1 Objectives

1. To learn how to convert a discrete signal into Z domain
2. To learn how to analyze the stability of a discrete system
3. To learn how to find the region of convergence from MATLAB plot

5.2 Learning Outcomes

After completing this experiment, the students will be able to:

1. Able to convert a discrete signal into Z domain and find it's ROC using MATLAB
2. Able to comment on the stability of a discrete system

5.3 Theory

The z-transform of a sequence x(n) is given by

$$x(z) = Z[x(n)] = \sum_{n=-\infty}^{\infty} x(n)\, z^{-n} \tag{5.1}$$

where z is a complex variable. The set of z values for which X(z) exists is called the region of convergence (ROC). The inverse z-transform of a complex function X(z) is given by

$$x(n) = Z^{-1}[X(z)] = \frac{1}{2\pi j}\oint X(z) z^{n-1}\, dz \tag{5.2}$$

Where C is a counter clockwise contour encircling the origin and lying in the ROC.

1. The roots of the denominator of X(z) indicate the poles, and the roots of the numerator are the zeros.
2. The z-plane plot visualizes the poles and zeros, aiding in the analysis of system stability and behavior.

3. The impulse response of a system can be found by performing the inverse z-transform or by using MATLAB's impz function.
4. Partial Fraction Expansion helps express X(z) in simpler terms for inverse z-transform computation.
5. The frequency response of a system can be analyzed using MATLAB's freqz function.

5.4 List of Equipments

1. Desktop PC
2. Software MATLAB R2020a

5.5 Procedure

5.5.1 Example 1

Assume we have a transfer function in the z-domain given by and the partial fraction form X(z) can be written as

$$X(z) = \frac{z^{-1}}{1 - 0.25z^{-1} - 0.375z^{-2}} = \frac{0.8z}{z - 0.75} + \frac{-0.8z}{z + 0.5}$$

The inverse z-transform of this is thus

$$x(n) = [(0.8)(0.75)^n + (-0.8)(-0.5)^n]u(n)$$

There are several MATLAB functions that could assist with calculating and analyzing these results. We can find the roots of the denominator polynomial using-

```
den = [1 -0.25 -0.375];
roots(den)
ans =
0.7500
 -0.5000
```

We can then plot the zeros and poles either with

- zeros and poles in ***column*** vectors

```
z = [0]
```

p = [0.75; -0.5]

zplane(z,p)

- numerator and denominator coefficients in ***row*** vectors

num = [0 1 0]

num = 0 1 0

den = [1 -0.25 -0.375]

den = 1.0000 -0.2500 -0.3750

zplane(num,den)

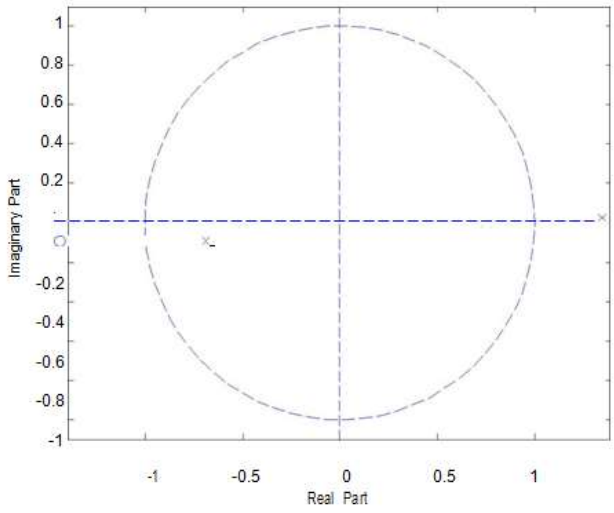


Figure 5.1: z-plane

5.5.2 Example 2

We can find the impulse response (or inverse z-transform) of the polynomial based on the power series expansion method using the *"impz"* function

h = impz(num,den,N) where N is the number of terms or coefficients to compute-

>> h = impz(num,den,10)

h =

0

1.0000

0.2500

0.4375

0.2031

0.2148

0.1299

0.1130

0.0770

0.0616

MATLAB can also be used to help found the partial fraction expansion

Assume you have a polynomial of

$$X(z) = \frac{z}{z^2 - 0.25z - 0.375} \tag{5.3}$$

so, in MATLAB we would enter

num = [0 1];

den = [1 -0.25 -0.375];

[R,P,K]=residuez(num,den)

Now we will get in the command script –

=

0.8000

-0.8000

=

0.7500

-0.5000

= [ ]

which corresponds to a partial fraction expansion of

$$X(z) = \frac{0.8z}{z - 0.75} \pm \frac{0.8z}{z + 0.5} + 0 \tag{5.4}$$

$$x(n) = [(0.8)(0.75)^n + (-0.8)(-0.5)^n]u(n)$$

5.5.3 Example 3

The same residue function can used to convert back a partial fraction form into a

rational form also. Let us consider the partial function form expansion of the X(z) defined as

$$X(z) = \frac{0.8z}{z - 0.75} \pm \frac{0.8z}{z + 0.5} + 0 \qquad (5.5)$$

If we use MATLAB to find the corresponding rational function we should use

```
P=[0.8 -0.8]
R=[0.7500  -0.5000]
>> [b,a]=residuez(R,P,K)
b =
0         1   0
a =
1.0000  -0.2500  -0.3750
```

So we will get,

$$X(z) = \frac{z}{z^2 - 0.25z - 0.375}$$

5.5.4 Example 4

We can also plot the frequency response of a particular polynomial with freqz

$$X(z) = \frac{z}{z^2 - 0.25z - 0.375} \qquad (5.6)$$

```
>> num = [0 1 0];
>> den = [1 -0.25 -0.375];
>> freqz(num, den, 512)
```

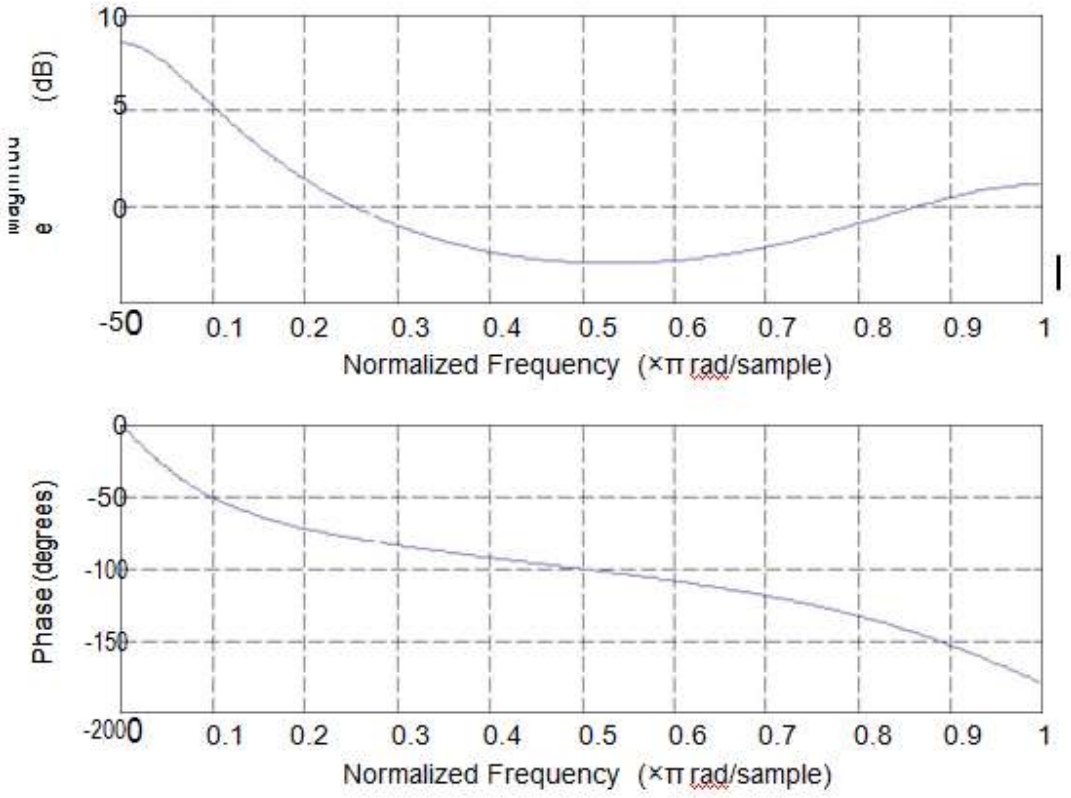


Figure 5.2: Magnitude and Phase Curve

Here we have used the form *freqz(num,den,N)*

where 'num' are the numerator coefficients, 'den' are the denominator coefficients, and 'N' is the number of points to use in the plot which goes from 0 to π. An alternative form is to use

[H,f]=freqz(num,de,N,Fs)

plot(f, abs(H))

'num' and 'den' are the same. 'Fs' is the sampling frequency. 'N' values between 0 and Fs are calculated. The response data versus frequency are stored in H.

```
num = [0 1 0];
den = [1 -0.25 -0.375];
    [H,f] = freqz(num,den,512,8000);
plot(f,abs(H))
```

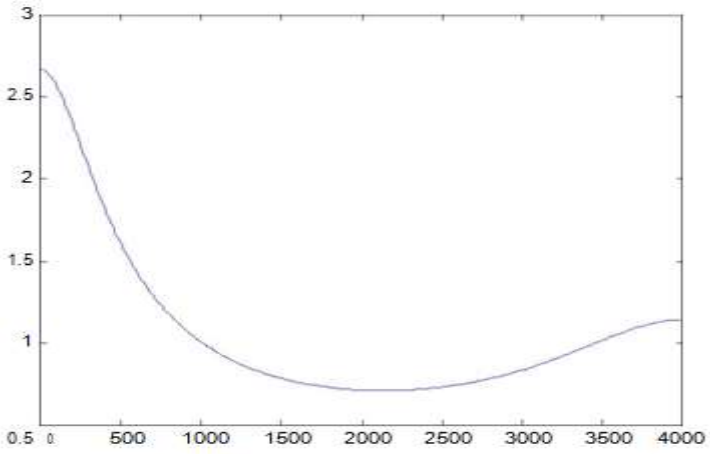


Figure 5.3: frequency curve

5.6 Student Works

Plot the frequency response of a particular polynomial with freqz

$$X(z) = \frac{z}{z^2 - 1.5z - 0.475} \tag{5.7}$$

5.7 Report questions

1. Find the partial inverse z-transform of the H(z) through partial fraction expansion by MATLAB.

$$X(z) = \frac{1 - \frac{7}{4z} + \frac{1}{4z^2}}{1 - \frac{3}{4z} + \frac{1}{8z^2}} \tag{5.8}$$

ROC: |z|> ½

2. Draw the pole-zero plot of *X(z)*
3. Draw the magnitude and phase plot of $X(e^{jw})$

5.8 References

1. Digital Signal Processing: Principles, Algorithms, and Applications, 4th Edition, Dimitris G. Manolakis and John G Proakis.

2. Oppenheim, A. V., & Schafer, R. W. (2010). *Discrete-Time Signal Processing*. Pearson.

Experiment 6
Discrete Time Fourier Transform

6.1 Objectives

1. To learn how to compute and interpret the Discrete Time Fourier Transform (DTFT) of a discrete signal
2. To investigate the properties of DTFT

6.2 Learning Outcomes

After completing this experiment, the students will be able to:

1. Able to perform a DTFT of a discrete signal
2. Know the interpretation of a DTFT performed
3. To use the properties of DTFT and their significance

6.3 Theory

If x(n) is absolutely summable, this is, $\sum_{n=-\propto}^{\propto}[x(n)] <\propto$ then its DTFT is given by

$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty}[x(n)]e^{-j\omega n} \tag{6.1}$$

The inverse Discrete Time Fourier Transform (IDFT) is defined as

$$x(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} x(e^{j\omega})e^{j\omega n}d\omega \tag{6.2}$$

The DTFT transforms a discrete signal x(n) into a complex valued continuous functions $X(e^{jw})$ of a real vaiarble w, called a digital frequency , which is measured in radians.

Some of the important properties of the DTFT are as follows:

Periodicity

The Discrete Time Fourier Transform (DTFT) of a signal x(n) is periodic in with a period of 2π.

$$X(e^{j\omega}) = X(e^{j(\omega+2\pi)} \tag{6.3}$$

Symmetry

For a real valued signal x(n) , its DTFT $X(e^{j\omega})$ is a conjugate symmetric

$$(e^{j\omega}) = X^{*}e^{j\omega} \tag{6.4}$$

Linearity

The DTFT is a linear transformation; If,

$$x_1(n) \leftrightarrow X_1(e^{j\omega}) \tag{6.5}$$

$$x_1(n) \leftrightarrow X_2(e^{j\omega}) \tag{6.6}$$

$$\alpha x_1 + \beta x_2(n) \leftrightarrow \alpha X_1(e^{j\omega}) + \beta X_2(e^{j\omega}) \tag{6.7}$$

Time Shifting

A shift in the time domain corresponds to the phase shifting

$$(n-k) \leftrightarrow X(e^{j\omega})e^{-j\omega k} \tag{6.8}$$

Frequency Shifting

Multiplication by a complex exponential corresponds to a shift in the frequency domain.

$$x(n)e^{j\omega_o n} \leftrightarrow X(e^{j(\omega-\omega_o)}) \tag{6.9}$$

The students can easily verify the periodicity property and symmetry property from Example 1 and Example 2. We will now verify linearity property, time shifting property and frequency shifting property.

6.4 List of Equipments

1. Desktop PC
2. Software MATLAB R2020a

6.5 Procedure

6.5.1 Example 1

Determine the DTFT of the following discrete signal

$$x(n) = (0.5)^n u(n)$$

Solution:

The sequence x(n) is absolutely summable; therefore it's DTFT exists. The DTFT of x(n) is defined as

$$X(e^{j\omega}) = \sum_{-\infty}^{\infty} x(n)e^{-j\omega n} = \sum_{-\infty}^{\infty} 0.5^n e^{-j\omega n}$$

$$X(e^{j\omega}) = \sum_{0}^{\infty} (0.5e^{-j\omega})^n = 1 + 0.5e^{-j\omega} + (0.5e^{-j\omega})^2$$

Multiplying both side of $0.5e^{-j\omega}$,we got the following

$$X(e^{j\omega})0.5e^{-j\omega} = 0.5e^{-j\omega} + (0.5e^{-j\omega})^2 + (0.5e^{-j\omega})^3$$

By subtracting these equation, we get-

$$X(e^{j\omega}) = \frac{1}{1 - 0.5e^{-j\omega}} = \frac{e^{j\omega}}{e^{j\omega} - 0.5}$$

The following code will compute the DTFT of x(n)

```
w = [0:1:500]*pi/500; % [0, pi] axis divided into 501 % points.
X = exp(j*w) ./ (exp(j*w)-0.5*ones(1,501));
magX = abs(X);
angX = angle(X);
realX = real(X);
imagX = imag(X);
subplot(2,2,1); plot(w/pi, magX);
xlabel('frequency in pi units');
ylabel('Magnitude');
title('Magnitude Part'); subplot(2,2,2);
plot(w/pi, angX); grid
```

```
xlabel('frequency in pi units');
ylabel('Radians');
title('Angle Part');
subplot(2,2,3);
plot(w/pi, realX);
xlabel('frequency in pi units');
ylabel('Real Value');
title('Real Part');
subplot(2,2,4);
plot(w/pi, imagX);
xlabel('frequency in pi units');
ylabel('Imaginary Value');
title('Imaginary Part');
```

In this example the the DTFT of x(n) has been found for 501 equispaced points between [0, π]. Please examine the plots produced by the above piece of MATLAB Code.

6.5.2 Example 2

Find the DTFT of the signal defined by

$$x(n) = \left(0.9e^{\frac{j\pi}{3}}\right)^n \qquad \text{Where} \quad 0 \le n \le 10$$

Solution:

The DTFT of the x(n) is defined as

$$X\left(e^{j\omega}\right) = \sum_{n=-\infty}^{\infty} x(n)e^{-j\omega n} = \sum_{n=0}^{10} \left(0.9\, e^{\frac{j\pi}{3}}\right)^n e^{-j\omega n} = \sum_{n=0}^{10} \left(0.9\, e^{\frac{j\pi}{3}}\right)^n e^{-j\omega n}$$

This code will compute DTFT of x(n) and plot it

```
n = 0:10;
x = (0.9*exp(j*pi/3)).^n;
k = -200:200;
w =(pi/100)*k;
```

```
X = x*(exp(-j*pi/100)).^(n'*k); magX = abs(X); angX = angle(X);
subplot(2,1,1); plot(w/pi, magX); grid
xlabel('frequency in pi units'); ylabel('Magnitude'); title('Magnitude Part');
subplot(2,1,2);
plot(w/pi, angX/pi); grid
xlabel('frequency in pi units'); ylabel('Radians/pi'); title('Angle Part');
```

6.5.3 Example 3

In this example we will verify the linearity property using real-valued finite duration sequences. Let x1(n) and x2(n) be two random sequences uniformly distributed between [0,1] over 0≤n≤10. By using the following piece of MATLAB code, we can verify the linearity property of DTFT.

```
x1=rand(1,11); % generate random number x1
x2=rand(1,11); % generate random number x2
alpha=2;
beta=3;
x3=alpha*x1+beta*x2 % generate the third signal and x3 is a linear combintaion
of x1 and x2
compute the DTFT of x1,x2 and x3 n=0:10;
k = 0:500;
w = (pi/500)*k; m=n'*k
X11=(exp(-j*pi/500)).^m
X1=x1*X11 % DTFT of x1
X2=x2*X11 % DTFT of x2
X3=x3*X11 % DTFT of x3
X_check=alpha*X1+beta*X2; % linear combination of DTFT  of x1 and DTFT of x2
```

```
error=max(abs(X3-X_check)) % check the error
```

6.6 Student Works

1. Using MATLAB, modify the given code to find the DTFT of

$$x(n) = sin\left(\frac{\pi n}{4}\right) u(n)$$

2. Experimentally verify the frequency-shifting property for x(n) = {1,2,3,4} with a shift of $\omega_o=\pi/4$

6.7 Report questions

1. Find the DTFT of the following two discrete signal given by

   a. x(n)={1 3 5 7 9 11 13 15 17}
   ↑

   b. $h(n)_2$={1 -2 3 -2 1}
   ↑

2. verify the time shifting property, we generate a random sequence uniformly distributed between [0,1] over 0 ≤n≤10 and we will generate another signal y(n)=x(n-2). Then we will verify the sample shift property. Use the following piece of MATLAB code

6.8 References

1. Digital Signal Processing: Principles, Algorithms, and Applications, 4th Edition, Dimitris G. Manolakis and John G Proakis
2. Oppenheim, A. V., & Schafer, R. W. (2010). *Discrete-Time Signal Processing*. Pearson.

Experiment 7
Study of Sampling, Quantization and Encoding

7.1 Objectives

The objectives of this experiment is to,

1. Learn the importance of sampling, quantization, and encoding in digital signal processing.
2. Explain the Nyquist sampling theorem and its significance in preventing aliasing.

7.2 Learning Outcomes

After completing this experiment, the students will be able to:

1. Determine appropriate sampling rates for various signals.
2. Analyze the effects of quantization error and how it relates to signal-to-noise ratio (SNR).
3. Understand how encoding affects data rate and bandwidth in communication systems.

7.3 Theory

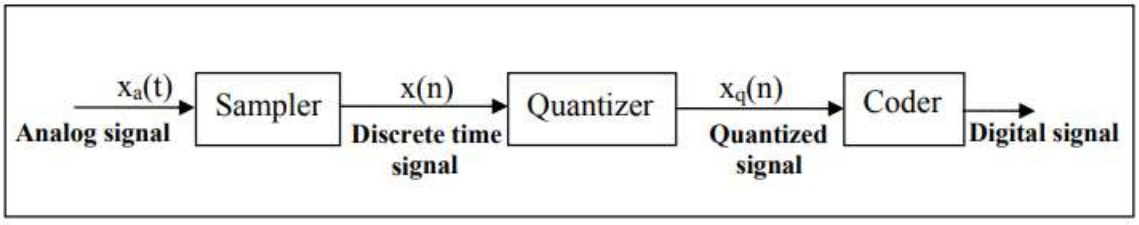


Fig.7.1. Analog to digital conversion of signals

An analog signal is sampled at regular intervals, with each sample taken once every T_s seconds, producing a sequence of sampled data. The sampler is considered ideal, meaning it captures the signal's value at a single, infinitesimally small moment in time.

The key parameter in the sampling process is the sampling period T_s , which determines the sampling frequency or sampling rate. The sampling frequency, expressed as $f_s = \frac{1}{T_s}$, where f_s is measured in units of "samples per second" or "hertz (Hz)."

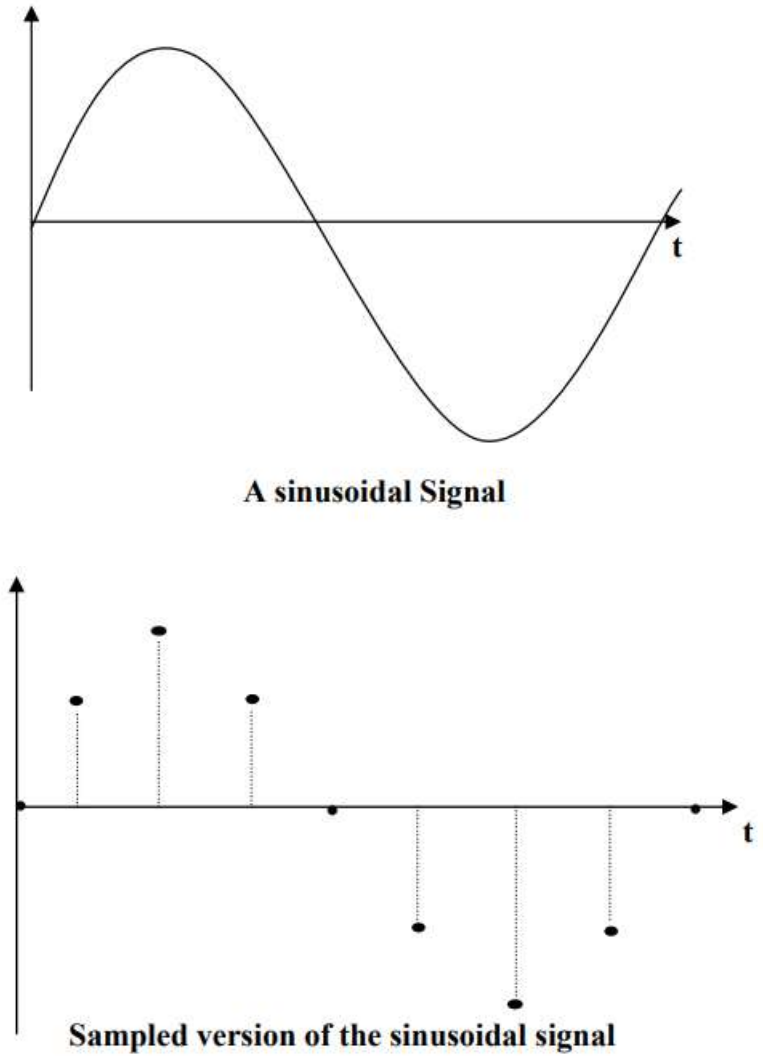


Fig.7.2. Sampling of a sinusoidal signal

Sampling theorem:

The sampling theorem states that a continuous-time signal must be sampled at a rate at least twice the highest frequency of the signal to ensure accurate reconstruction. This minimum sampling rate allows the original signal to be fully recovered using

simple low-pass filtering. If the sampling frequency is insufficient, the signal cannot be accurately reconstructed.

If a continuous time signal contains no frequency components higher than W Hz, then it can be completely determined by uniform samples taken at a rate sample f_s per second where $f_s \geq 2W$.

Or, in terms of sampling period,

$$T_s \leq \frac{1}{2W}$$

Quantization:

This process converts a continuous-valued signal into a digital signal with discrete values x(n). Each signal sample is assigned to the nearest value from a fixed set of possible values. This assignment, known as quantization, is done by either rounding or truncating the sample.

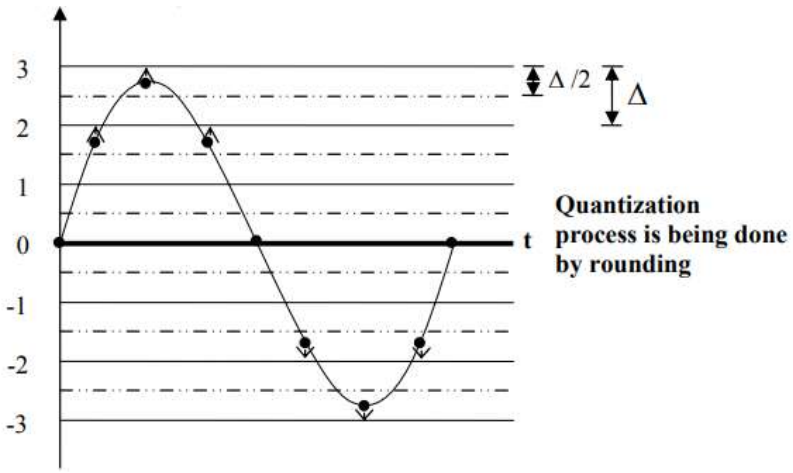


Fig.7.3. Quantization by rounding

In the case of rounding, the step size is divided into upper and lower halves. The value in the upper half are stepped into the next level and the value in the lower half remains in that level.

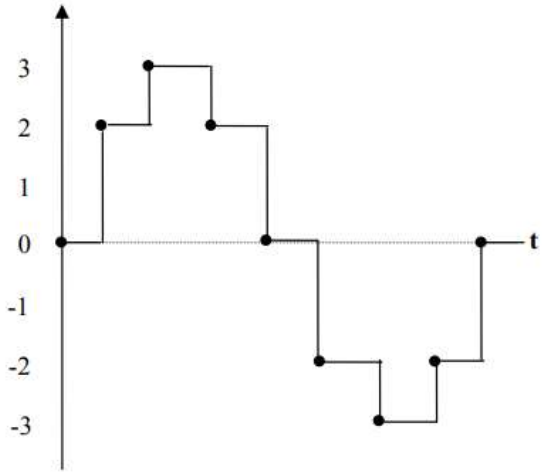


Fig.7.4. Quantized Signal

When the quantization operation is applied to a signal x(n), the resulting output is called the quantized signal $x_q(n)$. The difference between the original unquantized sample and the quantized output is referred to as the quantization error. The limited range of possible output values is known as quantization levels, and the gap between two consecutive levels is called the quantization step size or resolution (Δ).

The quantization error $e_q(n)$ is limited to,

$$-\frac{\Delta}{2} \quad to \quad +\frac{\Delta}{2}$$

If x_{min} and x_{max} represent the minimum and maximum value of x(n) and L is the number of the quantization levels, then

$$\Delta = \frac{x_{max} - x_{min}}{L-1}$$

If x_{max} is +V and x_{min} = - V and b is the number of bits,

$$\Delta = \frac{2V}{2^b - 1}$$

The SNR can be derived as,

SNR (dB) = 1.76+6b. Here b is the number of bits.

7.4 List of Equipments

1. Desktop PC.
2. Matlab R2020a.

7.5 Procedure

1. Generate a signal x(t) = sin ($2\pi 10t + 10^6$)
2. Take the samples of this signal at a sampling rate of 20Hz.
3. Reconstruct the analog signal. [use interp1() function].
4. Repeat steps 2 and 3 at a sampling rate of 50Hz, 100Hz and 10 Hz.

7.6 Student Works

1. Generate another signal y(t) = sin(2π10t) + sin(2π50t) + sin(2π100t)

   For successful reconstruction of the signal find the sampling frequency.

   Verify it through the program.

7.7 Report Questions

Take the signal of in **student works** section and sample it by 200 Hz sampling frequency. Quantize the signal by a 3-bit uniform quantizer. Obtain the quantization noise power. Obtain the SNR using the equation mentioned in the theory part.

7.8 References

1. https://www.mathworks.com/help/fixedpoint/ref/embedded.quantizer.quantize.html

Experiment 8
Tumor Detection from US Image using MATLAB

8.1 Objectives

1. To learn biomedical image processing using MATLAB.
2. To learn image processing command.

8.2 Learning Outcomes

1. Auto segmentation of Tumor region using MATLAB.

8.3 Theory

Medical imaging plays a crucial role in modern diagnostics, enabling the visualization and analysis of internal body structures to identify abnormalities such as tumors. Among the various imaging modalities, ultrasound (US) is widely used due to its non-invasive nature, real-time imaging capability, and cost-effectiveness. Detecting and segmenting tumors in ultrasound images is critical for diagnosing diseases, planning treatments, and monitoring patient outcomes. This experiment focuses on using MATLAB for tumor detection in ultrasound images. MATLAB provides robust tools and functions for image processing, making it ideal for automating segmentation tasks and enhancing the accuracy of diagnostic processes.

Ultrasound imaging operates on the principle of transmitting high-frequency sound waves into the body and capturing their echoes as they reflect off internal structures. While effective, ultrasound images often suffer from noise, low contrast, and speckle artifacts, making tumor detection a challenging task. To address these challenges, this experiment employs image processing techniques in MATLAB to automatically segment tumor regions.

Steps in Tumor Detection

1. **Image Acquisition**: The ultrasound image is read from the dataset using MATLAB's imread function. Visualization is facilitated through functions such as imshow and imagesc.
2. **Preprocessing**: The image is converted into an envelope image to reduce noise and enhance the features of interest. The envelope image is further transformed into a B-Mode image, which is the standard grayscale representation of ultrasound images.

3. **Binarization**: The B-Mode image is converted into a binary image using thresholding techniques (imbinarize), which segments the image into background and potential regions of interest.
4. **Region Selection**: The user manually identifies the tumor region using interactive tools such as imfreehand. This step ensures that the region of convergence (ROC) is accurately defined.
5. **Boundary Detection and Visualization**: The detected tumor boundary is highlighted using MATLAB's boundary visualization tools. The segmented region can then be saved for medical purposes or further analysis.
6. **Postprocessing**: Operations like morphological filtering (imfill) and hole filling enhance the accuracy of the segmented region, ensuring that only the most relevant areas are retained.

8.4 List of Equipments

1. Desktop PC
2. Software MATLAB R2020a

8.5 Procedure

1. Read US image from data set using imread function
2. Show the US image using imagesc, imshow function.
3. Convert the US image to envelope image using envelope function.
4. Convert the envelope image to uint8 image using imzvint8 function in order to get B_Mode image.
5. Convert B_Mode image to Binary image using imbinarize function.
6. Manually select the region of convergence (ROC) using imfreehand function.
7. Save auto boundary image for medical purpose.

8.5.1 Steps

Code	Details
clc;clear all; close all; pwd; CurrentFolder=pwd; pid=imread([CurrentFolder,'\DATA_Student_2\000015.png']); figure(1) image(pid) title('Original Image Read')	

Code	Details

figure(2)
imagesc(pid);
imshow(pid);
imagesc(pid);
title('Original Image Read using Imagesc')
colormap turbo	

Envelope_RF1=envelope(pid);
B_Mode_full = im2uint8(mat2gray(log10(Envelope_RF1))); % B-mode image
figure(3)
imagesc(B_Mode_full);
imshow(B_Mode_full);
imagesc(B_Mode_full);
title('B-Mode-full Image')	

thresholdLevel=graythresh(B_Mode_full);
grayImage = imbinarize(B_Mode_full,thresholdLevel);
figure(4)
imagesc(grayImage);
imshow(grayImage);
imagesc(grayImage);
title('Gray of B-Mode full Image')
 	
binaryImage = bwareafilt(~grayImage, 1);
binaryImage = imfill(binaryImage, 'holes');	

s=2
while s==2
figure(5)
imagesc(grayImage);
imshow(grayImage);
imagesc(grayImage);
title('Binary of B-Mode full Image')
 	
hold on
imfreehand
pause
 	
s = input('Exactly Done?:\n 1.Yes\n 2.No\n Enter your choice: ');
	Select Lesion Region

Code	Details

% if s==2
% close figure(5)
if s==1
  clear g1
  g1=input('Enter The Region of Lesion: ');
  break
end
% close all
end	

Region_Lesion=g1;
v=round(Region_Lesion);
maximum=max(v);
minimum=min(v);
width=maximum(1)-minimum(1);
height=maximum(2)-minimum(2);
size_ellipse=size(v);
f=round(Region_Lesion);
ep=size(Region_Lesion);
 	
match_1=f(1,1);match_2=f(1,2);match_3=[f(1,1) f(1,2)];
for i=1:ep(1)
  if f(i,1)~=match_1 || f(i,2)~=match_2
  match_3(i,1)=f(i,1);
  match_3(i,2)=f(i,2);
  match_1=f(i,1);
  match_2=f(i,2);
  end
end
match_3;
ep=size(match_3);
match_4=[];match_5=[];
for i=1:ep(1)
  if match_3(i,1)~=0 || match_3(i,2)~=0
  match_4=[match_4 match_3(i,1)];
  match_5=[match_5 match_3(i,2)];
  end
end
match_4;
match_5;
ep=size(match_4);match_6=[];
for i=1:ep(2)
	Filtering of Redundant Coordinates and Convert Float to Integer

Code	Details

match_6(i,1)=match_4(1,i);
match_6(i,2)=match_5(1,i);
end
match_6;
ep=size(match_6);
 	
match_7=[];match_8=[];match_9=[];match_10=[];
 	
for j=1:ep(1)
count_1=0;count_2=0;
for i=1:ep(1)
if match_6(j,1)==match_6(i,1) & match_6(j,2)==match_6(i,2)
count_1=count_1+1;
end
end
for i=1:length(match_7)
if match_6(j,1)==match_7(1,i) & match_6(j,2)==match_8(1,i)
count_2=count_2+1;
end
end
if count_1==1 & count_2==0
match_7=[match_7 match_6(j,1)];
match_8=[match_8 match_6(j,2)];
elseif count_1>1 & count_2==0
match_7=[match_7 match_6(j,1)];
match_8=[match_8 match_6(j,2)];
elseif count_1>1 & count_2>0
match_9=[match_9 match_6(j,1)];
match_10=[match_10 match_6(j,2)];
end
 	
end
x_co_ordinate_Lesion=match_7;
y_co_ordinate_Lesion=match_8;
match_9;
match_10;
clear ep
ep=size(match_7);match_11_Lesion=[];
for i=1:ep(2)
match_11_Lesion(i,1)=match_7(1,i);
match_11_Lesion(i,2)=match_8(1,i);	

Code	Details
end match_11_Lesion;	

for i=1:length(match_11_Lesion)
  Region_Lesion_x(i)=match_11_Lesion(i,1);
  Region_Lesion_y(i)=match_11_Lesion(i,2);
end
 	
pk2=13;
MAT_File_Size_B_Mode=size(B_Mode_full);
if (min(Region_Lesion_x)-pk2)>0
  Min_x=min(Region_Lesion_x)-pk2;
else
  Min_x=min(Region_Lesion_x)-(min(Region_Lesion_x)-6);
end
if (max(Region_Lesion_x)+pk2)<MAT_File_Size_B_Mode(1,2)
  Max_x=max(Region_Lesion_x)+pk2;
else
 	
Max_x=max(Region_Lesion_x)+((MAT_File_Size_B_Mode(1,2)-max(Region_Lesion_x))-6);
end
if (min(Region_Lesion_y)-pk2)>0
  Min_y=min(Region_Lesion_y)-pk2;
else
  Min_y=min(Region_Lesion_y)-(min(Region_Lesion_y)-6);
end
if (max(Region_Lesion_y)+pk2)<MAT_File_Size_B_Mode(1,1)
  Max_y=max(Region_Lesion_y)+pk2;
else
 	
Max_y=max(Region_Lesion_y)+((MAT_File_Size_B_Mode(1,1)-max(Region_Lesion_y))-6);
end
 	
B_Mode_image_Reload=B_Mode_full;
	Reshaping B-Mode Image

clear MAT_File_Size;
MAT_File_Size=size(B_Mode_image_Reload);
  for ss=MAT_File_Size(1,1):-1:Max_y
  B_Mode_image_Reload(ss,:)=[];
	Reducing B-Mode Image

Code	Details

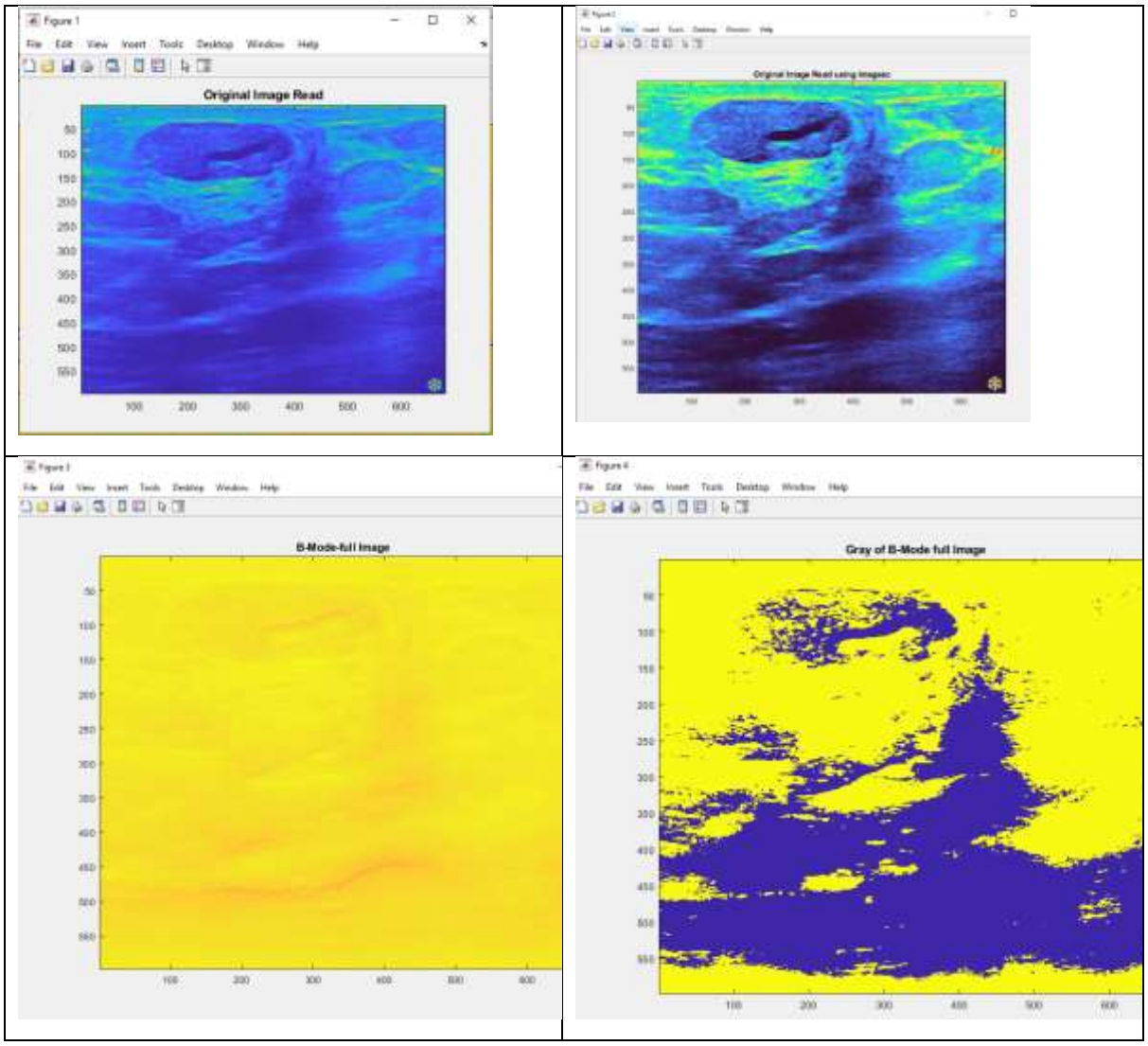
end
MAT_File_Size=size(B_Mode_image_Reload);
for ss=Min_y:-1:1
B_Mode_image_Reload(ss,:)=[];
end
MAT_File_Size=size(B_Mode_image_Reload);
for ss=MAT_File_Size(1,2):-1:Max_x
B_Mode_image_Reload(:,ss)=[];
end
MAT_File_Size=size(B_Mode_image_Reload);
for ss=Min_x:-1:1
B_Mode_image_Reload(:,ss)=[];
end
MAT_File_Size=size(B_Mode_image_Reload);
% disp('Binary Image Reload Size')
sz=size(B_Mode_image_Reload);
mat_size_1=sz(1);
mat_size_2=sz(2);	

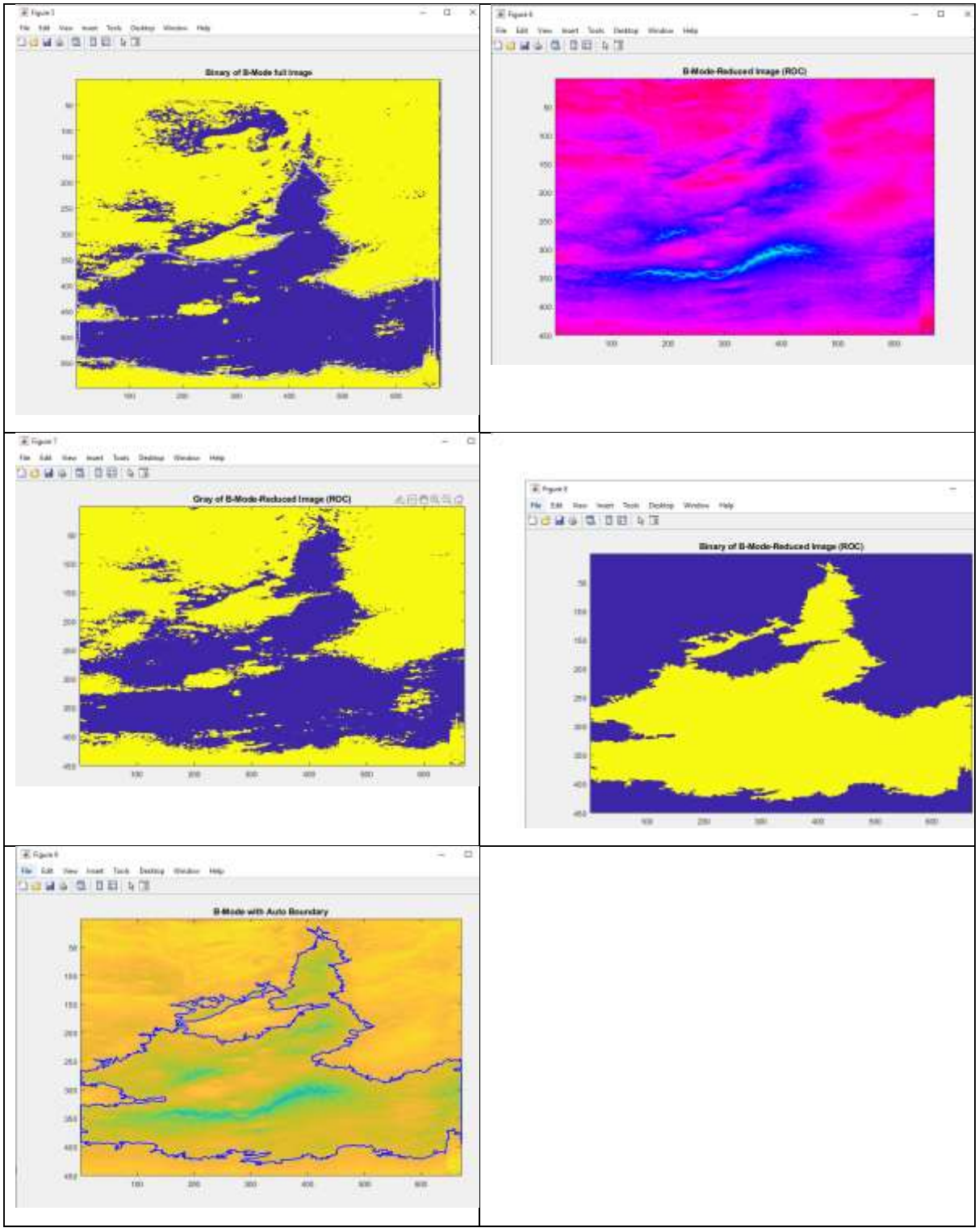
B_Mode = B_Mode_image_Reload;
figure(6)
imagesc(B_Mode);
imshow(B_Mode);
imagesc(B_Mode);
title('B-Mode-Reduced Image (ROC)')
colormap HSV
 	
thresholdLevel=graythresh(B_Mode);
grayImage = imbinarize(B_Mode,thresholdLevel);
figure(7)
imagesc(grayImage);
imshow(grayImage);
imagesc(grayImage);
title('Gray of B-Mode-Reduced Image (ROC)')
 	
binaryImage = bwareafilt(~grayImage, 1);
binaryImage = imfill(binaryImage, 'holes');
figure(8)
imagesc(binaryImage);
imshow(binaryImage);
imagesc(binaryImage);
title('Binary of B-Mode-Reduced Image (ROC)')	

Code	Details

figure(9)
imagesc(B_Mode);
imshow(B_Mode);
imagesc(B_Mode);
title('B-Mode with Auto Boundary');
axis on;
hold on;
% [B,~,~,rgbImage] = bwboundaries(binaryImage);
boundaries = bwboundaries(binaryImage);
visboundaries(boundaries, 'Color', 'b');
hold off
	Save B-Mode Image with Auto Boundary

8.5.2 Result





8.6 Student Works

1. Replace graythresh (Otsu's method) with other thresholding methods like adaptive thresholding or manual thresholding to observe how it impacts segmentation quality.

2. Use edge detection methods such as Canny, Sobel, or Prewitt to detect tumor boundaries.

3. Experiment with Noise Reduction Methods

8.7 Report Questions

1. Explain the significance of preprocessing in ultrasound tumor detection.
2. Discuss the role of binarization and morphological operations in segmentation.
3. Describe the use of manual region selection in tumor segmentation.

8.8 References

1. Gonzalez, R. C., & Woods, R. E. (2018). *Digital Image Processing* (4th Edition). Pearson.
2. Otsu, N. (1979). "A Threshold Selection Method from Gray-Level Histograms." *IEEE Transactions on Systems, Man, and Cybernetics*.

Experiment 9

Noise Reduction in Audio Signals using Adaptive Filtering

9.1 Objectives

To implement an adaptive noise cancellation algorithm to remove noise from a recorded audio signal.

9.2 Learning Outcomes

After completing this experiment the students will be able to:

1. Adaptive Filtering Basics: Gain practical experience in implementing adaptive filtering algorithms using MATLAB to remove noise from audio signals.
2. Signal Processing Proficiency: Develop skills in signal processing techniques, frequency domain analysis and parameter optimization for noise reduction.
3. Real-World Relevance: Understand the importance of noise reduction in industries like telecommunications and audio processing, while refining MATLAB programming abilities.

9.3 Theory

Adaptive Filtering: Adaptive filtering is a technique where the parameters of a filter are adjusted in real-time to optimize its response based on the characteristics of the input signal. In this context, an adaptive filter is used to model and reduce the noise present in an audio signal.

LMS Algorithm: The Least Mean Squares (LMS) algorithm is a widely used method for adaptive filtering. It updates filter coefficients iteratively to minimize the mean squared error between the filter's output and the desired output.

Algorithm Steps: Initialization: The adaptive filter coefficients are initialized to zeros. These coefficients determine the filter's behavior.

Iterative Update: The algorithm processes the input noisy audio signal iteratively.

Filter Output: For each iteration, the filter output is computed by taking the weighted sum of the current and past input samples.

Error Calculation: The difference between the noisy input and the filter's output is calculated. This difference represents the noise component.

Coefficient Update: The filter coefficients are updated using the LMS update rule. The update aims to minimize the error by adjusting the filter's weights.

Convergence Control: The step size parameter controls how much the filter coefficients are adjusted in each iteration. A small step size may lead to slow convergence, while a large step size may cause instability.

Repeat: The process is repeated for each sample in the input signal.

Filtered Output: The output of the adaptive filter represents an attempt to minimize the noise present in the original noisy signal. The filtered output ideally retains the desired audio content while attenuating noise.

Frequency Domain Analysis: The experiment includes a frequency domain analysis to visualize the effectiveness of noise reduction. Power Spectral Density (PSD) is calculated for both the original noisy signal and the filtered output. The PSD plots provide insights into how the noise reduction algorithm affects different frequency components of the signal. Comparing the PSDs helps assess the extent of noise reduction achieved.

9.4 List of Equipments

1. Desktop PC

2. Software MATLAB R2020a

9.5 Procedure:

9.5.1 Steps

1. Load the noisy audio signal and its sampling frequency using audioread.
2. Define the length of the adaptive filter.
3. Define LMS algorithm parameters
4. Initialize the filtered signal
5. Apply adaptive filtering using the LMS algorithm:
6. Save the filtered signal:
7. Perform frequency domain analysis:
8. Plot the PSD

The codes are given here:

```
>> % Load the noisy audio signal
>> [y, fs] = audioread('noisy_audio.wav');
```

%This line reads the audio file 'noisy_audio.wav' and stores the audio signal in the variable y and the %sampling frequency in fs.

```
>> % Define the length of the adaptive filter
>> filterLength = 128;
```

%Sets the length of the adaptive filter to 128 samples. This determines the number of past samples used %to predict the current output.

```
>> % Initialize the adaptive filter weights
>> w = zeros(filterLength, 1);
```

%Initializes the adaptive filter weights (coefficients) as a column vector of zeros

```
>> % Define LMS algorithm parameters
```

```
>> mu = 0.01;      % Step size
>> delta = 1e-6;   % Small constant to avoid division by zero
```

%Sets parameters for the LMS algorithm: mu is the step size, controlling the rate of weight updates; delta %is a small constant added to the denominator to prevent division by zero.

```
>> % Initialize the filtered signal
>> filteredSignal = zeros(size(y));
```

%Initializes the vector filteredSignal as zeros, which will store the output of the adaptive filtering process.

```
>> % Apply adaptive filtering using LMS
>>  for n = filterLength : length(y)
>>    % Extract a segment of the noisy signal
>>    inputSegment = y(n:-1:n-filterLength+1);
```

%A loop iterates over each sample in the noisy signal from filterLength to the end. inputSegment is a %vector containing the current filterLength samples of the noisy signal in reverse order.

```
>>    % Compute the filter output
>>    output = w' * inputSegment;
```

%Computes the output of the adaptive filter by taking the dot product of filter weights (w) and the input %segment.

```
>>    % Compute the error signal
>>    error = y(n) - output;
```

%Calculates the error by comparing the current noisy signal sample (y(n)) with the filter's output.

```
>>    % Update the filter weights using LMS algorithm
>>    w = w + (mu / (inputSegment' * inputSegment + delta)) * inputSegment * error;
```
%Updates the filter weights using the LMS algorithm's update rule. The weights are adjusted in the %direction that minimizes the error. The update includes a step size factor (mu), normalized by the energy %of the input segment.

```
>>    % Store the filtered output
>>    filteredSignal(n) = output;
>> end
```
%Stores the current output of the adaptive filter in the filteredSignal vector.

```
>>% Save the filtered signal
>>audiowrite('filtered_audio.wav', filteredSignal, fs);
```
%Saves the filtered output as a WAV file named 'filtered_audio.wav'.

```
>>% Frequency domain analysis
>>nfft = 1024;  % Number of FFT points
>>% Compute the power spectral density (PSD) of the original noisy signal
>>[P_noisy, f_noisy] = pwelch(y, hamming(nfft), [], nfft, fs);
```
%Performs a frequency domain analysis using the pwelch function. Calculates the power spectral density %(PSD) of the original noisy signal. nfft is the number of FFT points used for spectral analysis. f_noisy %contains the corresponding frequency values.

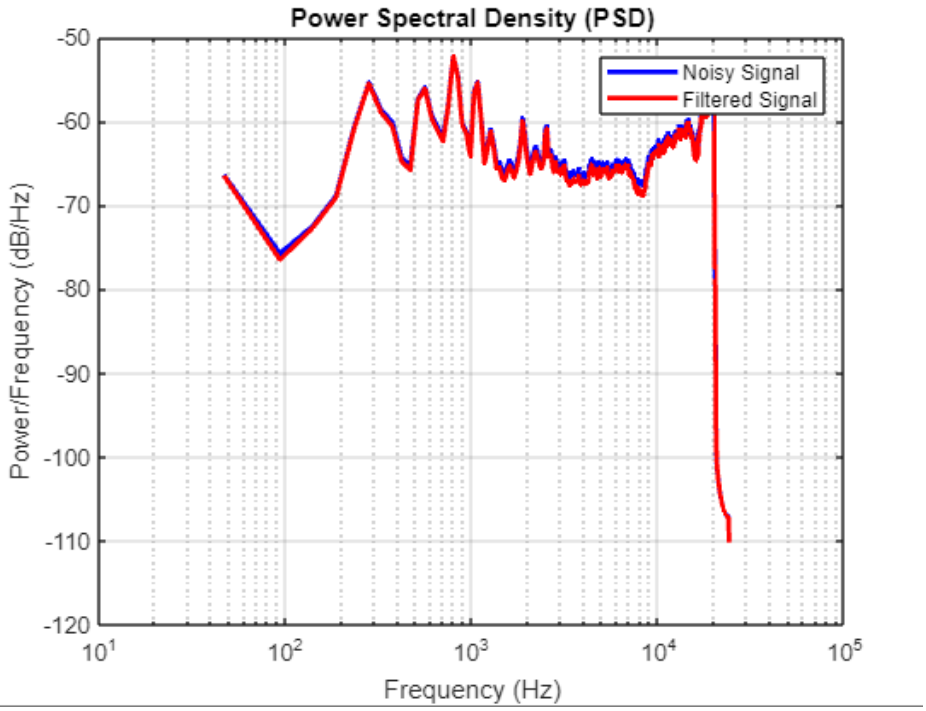
```
>>% Compute the PSD of the filtered signal
>>[P_filtered, f_filtered] = pwelch(filteredSignal, hamming(nfft), [], nfft, fs);
```
%Calculates the PSD of the filtered output using the same parameters.

```
>>% Plot the PSD
```

```
>>figure;
>>semilogx(f_noisy, 10*log10(P_noisy), 'b', 'LineWidth', 2);
>>hold on;
>>semilogx(f_filtered, 10*log10(P_filtered), 'r', 'LineWidth', 2);
>>hold off;
>>grid on;
>>title('Power Spectral Density (PSD)');
>>xlabel('Frequency (Hz)');
>>ylabel('Power/Frequency (dB/Hz)');
>>legend('Noisy Signal', 'Filtered Signal');
```

%Plots the PSDs of the original noisy signal (P_noisy) and the filtered output (P_filtered) on a %logarithmic frequency scale. The blue line represents the noisy signal's PSD, and the red line represents %the filtered output's PSD. Axis labels and a legend are added for clarity.

9.5.2 Result



9.6 Student Works

1. Implement a Wiener filter for noise reduction in the audio signal.

2. Apply median filtering to reduce impulsive noise in the audio signal.

9.7 Report Questions

1. Discuss the theoretical concepts of adaptive filtering and the LMS algorithm.
2. List three real-world applications where adaptive noise cancellation is essential.
3. What are the limitations of the LMS algorithm in adaptive filtering?

9.8 References

1. Haykin, S. (1996). *Adaptive Filter Theory*. Prentice Hall.
2. Proakis, J. G., & Manolakis, D. G. (2007). *Digital Signal Processing: Principles, Algorithms, and Applications*. Pearson Education.
3. Oppenheim, A. V., & Schafer, R. W. (2010). *Discrete-Time Signal Processing*. Pearson.

Experiment 10
Bone Fracture Detection in MATLAB

10.1 Objectives

This lab aims to introduce students to the practical use of Digital Signal Processing (DSP) in medical image analysis, focusing on bone fracture detection within X-ray images. By applying fundamental image processing techniques, students will learn to enhance images, identify edges, and potentially locate fractures. The experiment intends to familiarize students with DSP's role in medical diagnostics and highlight the synergy between technology and healthcare applications.

10.2 Learning Outcomes

After completing this experiment the students will be able to:

1. Image Processing Skills: Develop proficiency in using MATLAB for image enhancement, edge detection, and manipulation.
2. Real-world DSP Application: Understand how DSP techniques are practically employed in medical image analysis.
3. Medical Technology Insight: Gain awareness of how image processing aids in diagnosing medical conditions, fostering interest in healthcare applications

10.3 Theory

The provided MATLAB code performs bone fracture detection in X-ray images. It follows a series of steps to process the image and identify potential bone fractures. Below is an overview of the theory behind the code:

1. Image Preprocessing:

- The input image is loaded and converted to grayscale.
- A Gaussian filter is applied to the grayscale image to denoise it.

2. Edge Detection:

- Canny edge detection is applied to the denoised image to identify bone edges.
- Morphological closing operation is performed to enhance edge continuity.

3. Region of Interest (ROI) Extraction:

- The connected components of the detected edges are analyzed using region properties.
- The two largest edge regions (bones) are isolated.

4. Hough Transform:

- A Hough transform is applied to detect angles corresponding to bone orientations.
- Peaks in the Hough transform are identified as major angles.

5. Break Detection:

- For each detected bone, a convolution operation is performed using the bone's orientation.
- The convolution images are subtracted to detect potential break points.
- A threshold-based approach identifies significant break points.

6. Ellipse Fitting:

- Ellipses are fitted around the detected break points using region properties.
- Bounding ellipse coordinates are calculated using centroid, major axis, minor axis, and orientation information.

7. Visualization:

- The code generates visualizations at different stages of the process.
- The original X-ray image is displayed with ellipses drawn around detected breaks.

Overall, the code combines edge detection, Hough transform, convolution, and thresholding techniques to locate and mark potential bone fractures in X-ray images. It aims to provide a basic framework for identifying fractures, but it's important to note that real-world medical image analysis often involves more complex algorithms, validation, and expert consultation.

10.4 List of Equipments

1. Desktop PC
2. Software MATLAB R2020a

10.5 Procedure

The code of Bone Fracture Detection is given here:

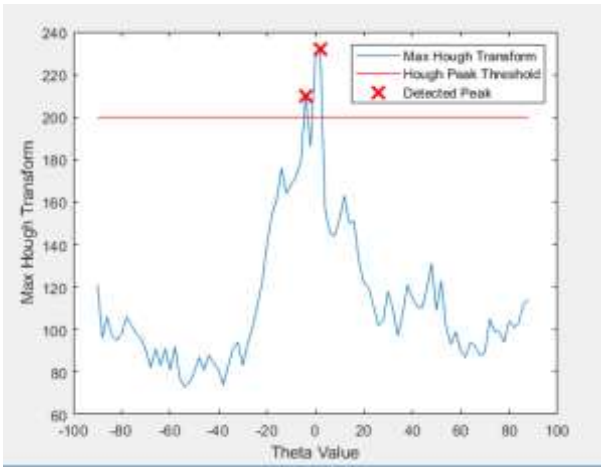
```
clc;
clear all;
close all;
img = imread('blue-bone.jpg');
ImgBlurSigma = .5; % Amount to denoise input image
MinHoughPeakDistance = 2; % Distance between peaks in Hough transform angle detection
HoughConvolutionLength = 40; % Length of line to use to detect bone regions
HoughConvolutionDilate = 2; % Amount to dilate kernel for bone detection
BreakLineTolerance = 0.15; % Tolerance for bone end detection
breakPointDilate = 6; % Amount to dilate detected bone end points

%%%%%%%%%%%%%%%%%%%%%%%
```

```
img = (rgb2gray(img)); % Load image
img = imfilter(img, fspecial('gaussian', 10, ImgBlurSigma), 'symmetric'); % Denoise
%figure(1)
%imshow(img)
% Do edge detection to find bone edges in image
% Filter out all but the two longest lines

boneEdges = edge(img, 'canny');
%figure (4)
%imshow(boneEdges)
boneEdges = bwmorph(boneEdges, 'close');
edgeRegs = regionprops(boneEdges, 'Area', 'PixelIdxList');
AreaList = sort(vertcat(edgeRegs.Area), 'descend');
edgeRegs(~ismember(vertcat(edgeRegs.Area), AreaList(1:2))) = [];
edgeImg = zeros(size(img, 1), size(img,2));
edgeImg(vertcat(edgeRegs.PixelIdxList)) = 1;
% Do hough transform on edge image to find angles at which bone pieces are
% found
% Use max value of Hough transform vs angle to find angles at which lines
% are oriented.  If there is more than one major angle contribution there
% will be two peaks detected but only one peak if there is only one major
% angle contribution (ie peaks here = number of located bones = Number of
% breaks + 1)
[H,T,R] = hough(edgeImg,'RhoResolution',1,'Theta',-90:2:89.5);
maxHough = max(H, [], 1);
HoughThresh = (max(maxHough) - min(maxHough))/2 + 1.65*min(maxHough);
[~, HoughPeaks] = findpeaks(maxHough,'MINPEAKHEIGHT',HoughThresh,
'MinPeakDistance', MinHoughPeakDistance);
```

```
% Plot Hough detection results
figure(2)
plot(T, maxHough);
hold on
plot([min(T) max(T)], [HoughThresh, HoughThresh], 'r');
plot(T(HoughPeaks), maxHough(HoughPeaks), 'rx', 'MarkerSize', 12, 'LineWidth', 2);
hold off
xlabel('Theta Value'); ylabel('Max Hough Transform');
legend({'Max Hough Transform', 'Hough Peak Threshold', 'Detected Peak'});
% Locate site of break
if numel(HoughPeaks) > 1;
   BreakStack = zeros(size(img, 1), size(img, 2), numel(HoughPeaks));
   % Convolute edge image with line of detected angle from hough transform
   for m = 1:numel(HoughPeaks);

      boneKernel = strel('line', HoughConvolutionLength, T(HoughPeaks(m)));
      kern = double(bwmorph(boneKernel.getnhood(), 'dilate',
HoughConvolutionDilate));
      BreakStack(:,:,m) = imfilter(edgeImg, kern).*edgeImg;
   end

   % Take the difference between convolution images.  Where this crosses zero
   % (within tolerance) should be where the break is.  Have to filter out
   % regions elsewhere where the bone simply ends.
   brImg = abs(diff(BreakStack, 1, 3)) < BreakLineTolerance*max(BreakStack(:)) &
edgeImg > 0;
   [BpY, BpX] = find(abs(diff(BreakStack, 1, 3)) <
BreakLineTolerance*max(BreakStack(:)) & edgeImg > 0);
```

```
   brImg = bwmorph(brImg, 'dilate', breakPointDilate);
   %imshow(brImg)
   brReg = regionprops(brImg, 'Area', 'MajorAxisLength', 'MinorAxisLength', ...
      'Orientation', 'Centroid');
   brReg(vertcat(brReg.Area) ~= max(vertcat(brReg.Area))) = [];
 % Calculate bounding ellipse
   brReg.EllipseCoords = zeros(100, 2);
   t = linspace(0, 2*pi, 100);
   brReg.EllipseCoords(:,1) = brReg.Centroid(1) + brReg.MajorAxisLength/2*cos(t -
brReg.Orientation);
   brReg.EllipseCoords(:,2) = brReg.Centroid(2) + brReg.MinorAxisLength/2*sin(t -
brReg.Orientation);

else
   brReg = [];
end
% Draw ellipse around break location
figure(3)
imshow(img)
hold on
colormap('gray')
if ~isempty(brReg)
   plot(brReg.EllipseCoords(:,1), brReg.EllipseCoords(:,2), 'r');
end
hold off
```

9.1.1 Result



10.6 Student works

1. Apply Canny edge detection on the preprocessed image.
2. Use the region properties function to extract the two largest connected components corresponding to the bones.
3. Apply the Hough transform to the edge-detected image and plot the Hough transform peaks.

10.7 Report Questions

1. Identify the limitations of this approach in real-world medical diagnostics.
2. Suggest improvements or additional techniques that could enhance bone fracture detection.
3. Explain the convolution-based approach for detecting bone breaks.

10.8 References

Reference Dataset for X-ray images:

https://drive.google.com/drive/folders/1gsMypMOg4krSHToVHXAcXvWcVHQfUHrg?usp=sharing