DEPARTMENT OF
ELECTRICAL & ELECTRONICS ENGINEERING

MANIPAL INSTITUTE OF TECHNOLOGY
Manipal - 576104

LABORATORY MANUAL

DIGITAL SIGNAL PROCESSING LABORATORY
[ELE 3161]

5th Semester B. Tech. (E & E Engineering)

AUGUST – DECEMBER 2021

Digital Signal Processing (DSP) Laboratory [0 0 3 1]
ELE 3161

The Digital Signal Processing (DSP) Laboratory is primarily intended for practical implementations and understanding of the undergraduate course on DSP. In this laboratory course, students learn ‘why’ and ‘how’ to realize the theoretical concepts and applications of DSP using MATLAB. In this course students learn how to generate and perform transformation of discrete time signals in time domain and understand their spectral components in the frequency domain, use computational techniques, design frequency selective discrete time systems, especially, finite impulse response (FIR) and infinite impulse response (IIR) filters. This course provides enough depth to the practical implementation of DSP techniques augmented with MATLAB functions and examples so that the learning is consistent, logical, and enjoyable. It is assumed that the student is familiar with the fundamentals of MATLAB and discrete time signals and systems. This course is a precursor to explore DSP system design and numerous applications of DSP such as speech, audio, image, video, and biomedical signal processing.

List of Experiments

Week 1:
Introduction to the DSP LAB course. Review of MATLAB

[3]

Week 2:
Time-domain analysis of signals and systems

[3]

Week 3:
Revisit time-domain analysis of signals and systems. Implementation of challenge questions.

[3]

Week 4:
Frequency-domain analysis of signals and systems

[3]

Week 5:
Revisit frequency-domain analysis of signals and systems. Implementation of challenge questions.

[3]

Week 6:
LTI system/filter design using pole-zero technique. FIR filter design using windowing and frequency sampling techniques.

[3]

Week 7:
Revisit LTI system/filter design using pole-zero technique. FIR filter design using windowing and frequency sampling techniques. Implementation of challenge questions.

[3]

Week 8:
IIR filter design; Butterworth, Chebyshev I and Chebyshev II filters.

[3]

Week 9:
Revisit IIR filter design; Butterworth, Chebyshev I and Chebyshev II filters.

[3]

Week 10:
Introduction to simple applications of DSP in speech processing, biomedical signal processing, and image processing. Demo of mini projects.

[3]

Course Outcomes

At the end of the program the students will be able to:

CLO1	Conversion of analog signals to digital signals and vice-versa, generation of different signals
CLO2	Compute the response of discrete-time systems/filters using convolution sum technique
CLO3	Compute the frequency response of discrete-time systems/filters by applying various techniques such as Z-transform, DTFT, and DFT; demonstrate the applications of FFT
CLO4	Design and implementation of FIR and IIR filters
CLO5	Carry out simulation of DSP systems

References

1. Proakis J. G. and D. G. Manolakis, Introduction to Digital Signal Processing, PHI, 2009
2. Oppenheim A. V. and R. W. Schafer, Discrete time signal processing, Pearson, 2009.
3. Mitra S. K., DSP: A computer-based approach (2e), TMH, 2006
4. D. Hanselman and B. LittleMield, Mastering MATLAB, Prentice Hall, 2011.
5. Haykin S., Signals and Systems, Wiley, 1999.
6. Oppenheim, Willisky, and Nawab, Signals and Systems (2e), PHI, 1997.

Evaluation Scheme

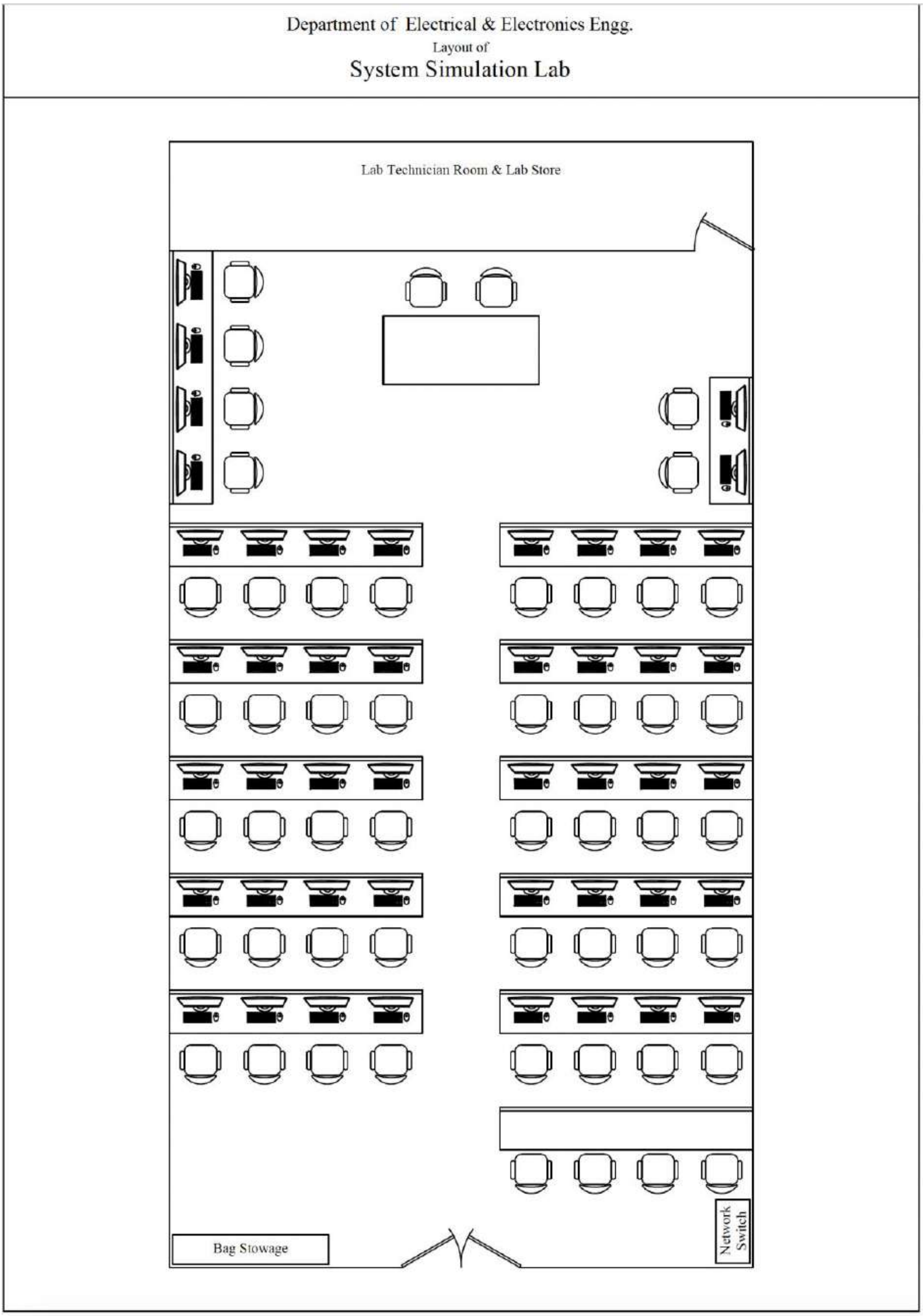
Continuous Evaluation – 60

- Preparation & Documentation - 25
- Conduction & Performance - 25
- Assignments/Quiz/Viva – 10

End Semester Lab Test – 40

- Initial write up (procedure/program) - 15
- Simulation & results - 20
- Viva voce - 05

Floor Plan


Department of Electrical & Electronics Engg.
Layout of
System Simulation Lab
Lab Technician Room & Lab Store
Bag Stowage
Network Switch

General Instructions

- Be present well before the lab session start time.
- Maintain a strict code of discipline inside the lab.
- Bring the lab manual along with a separate observation book.
- Use MATLAB help facility and learn more.
- Show the results to the instructors on completion of experiments.
- The students should not go out of the lab without permission.
- Theoretical background preparation is mandatory.
- Complete the assignments in time.
- Being absent without having genuine reason and without permission will be awarded zero marks for continuous evaluation.
- All tutorials to be completed before attending the lab examination.

Safety Instructions

- Avoid wearing indecent clothing.
- Shutdown the computer and turn off the power after finishing the experiments.
- Beware of sharp edges outside the computer case.
- Never open a power supply or a CRT monitor.
- Do not touch areas in printers that are hot or that use high voltage.
- Know where the fire extinguisher is located and how to use it.
- Do not bring food and drinks inside the lab.
- Keep your workspace clean and free of clutter.
- Adhere to the lab protocols.

Tutorial - I

A brief introduction to MATLAB

General Instruction:
This is a hands-on tutorial. MATLAB commands for you to type are printed in **bold letters (blue in color)**. Bold letters are also used to make MATLAB expressions that are in lower case more visible when found in a sentence.

This tutorial assumes you can open the MATLAB GUI. We are using **R2017/R2018** version of MATLAB.

Part-1: The basics:

1) Start MATLAB by double clicking on the MATLAB icon on Desktop. The MATLAB window should come up on your screen. It looks like as shown in the Figure 1.1.

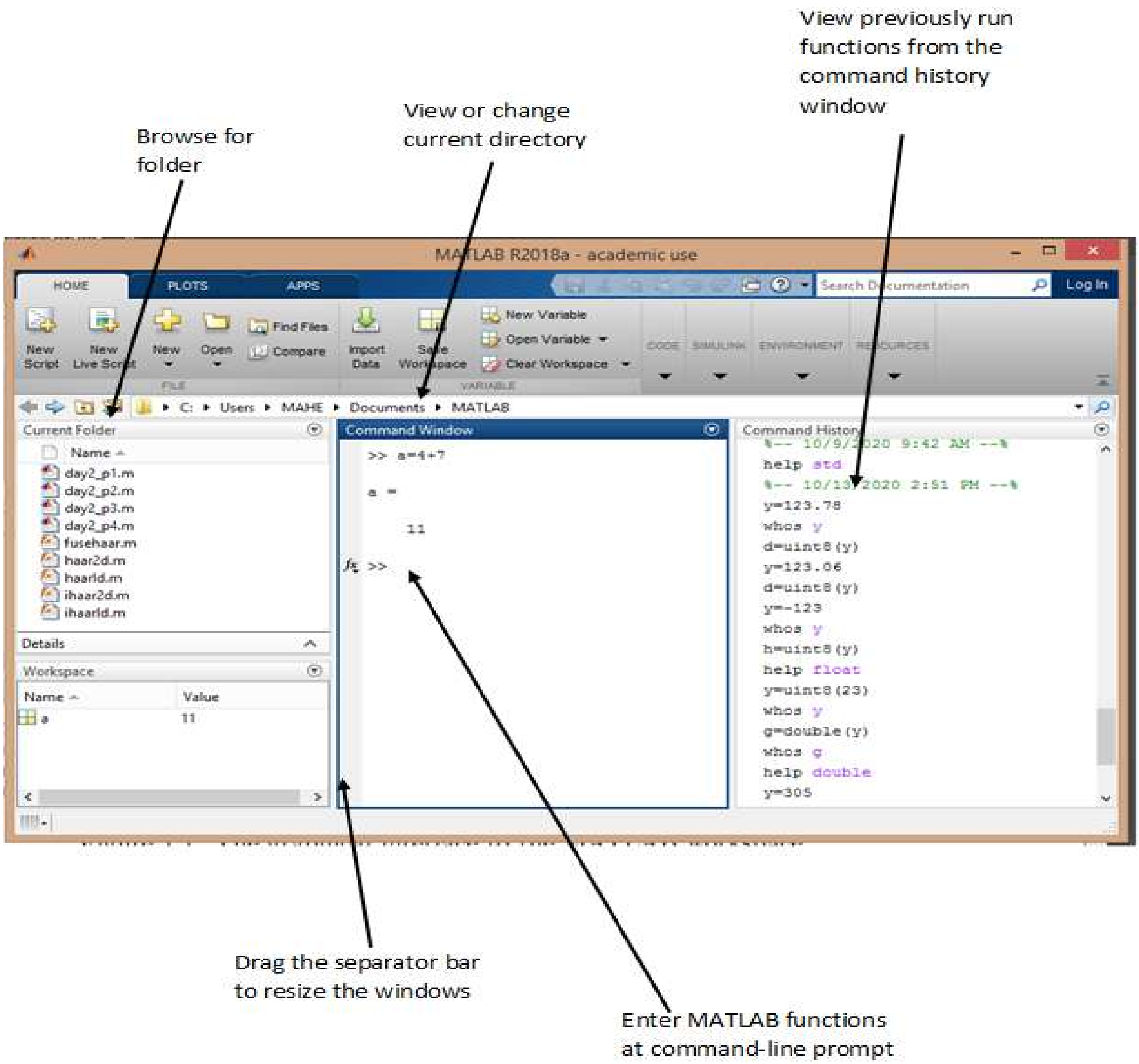


Figure 1.1 The graphical interface to the MATLAB workspace

This is the window in which you interact with MATLAB. The main window on the middle is called the Command Window. You can see the command prompt in this window, which looks like >>. If this prompt is visible, MATLAB is ready for you to enter a command. In the figure, you can see that we typed in the command a=4+7. In the bottom left corner, you can view the Workspace window. The Workspace window will show you all variables that you are using in your current MATLAB session. In this example, the workspace contains the variable 'a'. When you first start up MATLAB, the workspace is empty.

In the right side of the window you can see the Command History window, which simply gives a chronological list of all MATLAB commands that you used, the Current Directory shows you the location of the directory you are currently working in and the Current Folder shows you the contents.

You can change the layout of the MATLAB window:

2) To change the layout of the MATLAB window, click on *Layout*. Different layout styles can be chosen.

Note that MATLAB is case-sensitive, so

3) Make sure that the 'Caps Lock' is switched off!

During the MATLAB sessions you will create files to store programs or workspaces.

4) Create an appropriate folder to store this lab session's files.
5) Click on Browse for folder. Select the folder you just created so that MATLAB will automatically save files in this folder.
6) Now, let's try a simple command. Next to the prompt in the Command Window type **a=4+7** and then press 'enter' to activate this command. On the screen you should see

```
a=

        11
```

Notice how MATLAB carries out this command immediately, and gives you the prompt >> for your next command. Here, a variable called 'a' is created by MATLAB and assigned the value of 11. MATLAB stores the variable 'a' in its *workspace* until you exit MATLAB or tell MATLAB to delete the variable.

7) Go to the Workspace window and check that the variable 'a' is in your workspace.

You will see in this window that 'a' is stored as a double and that it has size 1x1.

8) In the Workspace window, double click on the name 'a'. A small window will pop up displaying a’s value. This is a handy feature. We can change the value of the variable.
9) Try **a = 16**, followed by 'enter' to change the value of a to 16. Then type **b=sqrt(a)** to find the square root of a. Press 'enter' to enter the command (in future hit ‘enter’ will not be mentioned when there is a MATLAB command). Now you should see

```
b=

        4
```

10) Go back to the Workspace window and check that 'b' is indeed in your workspace. Also check by double- clicking on 'a' that a’s value has changed.

Part-2:The MATLAB on-line help facility

MATLAB has handy on-line help facilities. There are several ways to get help. You can go to Help on the menu (Under Home toolbar Help menu is present) and select any of the available help facilities listed there. There are simpler help commands as well that work in all versions of MATLAB.

1) Type the command help in the Command Window to find a long list of all different categories for which there are MATLAB commands. Each of the listed categories contains more detailed information about available MATLAB functions. For example, type **help ops** to find a list of MATLAB operators. You can see from this list, for example, that more information about addition can be found in **help arith**.
2) If you want to know more about a MATLAB command type **help** followed by MATLAB command. For example type **help plot**

Alternatively, you can launch the help window:

3) Type **helpwin** to launch the help window.

In this window, you can get help on operators, for example, by double clicking on 'ops'.

Another very useful MATLAB command is the **lookfor** command. You can use this to search for commands related to a keyword.

4) Find out which MATLAB commands are available for plotting using the command lookfor plot in the command window. It may take a bit of time to get all commands on the screen. You can stop the search at any time by typing Ctrl-c (the control and the c key simultaneously).

Important
In the MATLAB **help** descriptions, the MATLAB commands and functions are often given in capital letters. But: always type commands in lower case. MATLAB is case sensitive and will generally not recognize commands typed in capital letters! Note that because of this case sensitivity the variables 'A' and 'a', for example, are different.

Part-3: Data representation in MATLAB

MATLAB stands for 'MATrix LABoratory'. This title is appropriate because the structure for the storage of all data in MATLAB is a matrix. The MATLAB matrix-variables may have any number of rows and columns. Scalars like the variables 'a' and 'b' that you worked with above are also stored as matrix variables with 1 row and 1 column.

So beware, a matrix-variable can be any variable in MATLAB, that is, it could be a scalar, a vector or a matrix of any size.

Entering variables:

An mxn ('*m* by *n*') MATLAB matrix-variable has *m* rows and *n* columns. For example, the variable $A = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$ is a 2 x 3 matrix.

1) Create the matrix A by typing **A = [1 3 5;2 4 6]** followed by the enter. Make sure that you separate the elements 1,3 and 5, and 2, 4 and 6 with a space. Use square brackets instead of parentheses. Instead of a space, you can also use a comma to separate the elements. The semi-colon (;) in this context is used to separate the elements of one row from another, but you can also use a line break to separate rows as shown below.

2) Remove the matrix A by typing clear A. Then recreate A using

```
>> A = [1,3,5        (now hit enter for the line break)
        2,4,6]
```

<u>Error messages:</u>

If your command is invalid MATLAB gives you explanatory error messages (luckily!). Read them carefully. Normally MATLAB points to the exact position of where things went wrong.

3) Try to change A to $\begin{bmatrix} 17 & 15 \\ 16 & 18 \end{bmatrix}$ using the incorrect command **A=[17 15 16;18]**

The command has failed. MATLAB tells you that dimensions of arrays being concatenated are not consistent. You put the semi-colon in the wrong place.

<u>Row and column vectors:</u>

A row vector has one row and any number of columns. Similarly, a column vector has one column and any number of rows.

4) Type **v=[1, 2, 3]** . Now use the size function to check that v has 1 row and 3 columns by typing **n= size(v)**

The first number returned by the **size** function gives the number of rows in the variable and the second number gives the number of columns.

5) Enter the column vector $B = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$ by typing **B=[1;2;3]** and check that is has 3 rows and 1 column.

<u>Changing matrices:</u>

You can change a matrix simply by defining it to be something else. We already unsuccessfully tried this when we tried to change A. Let's get it right this time.

6) Type **A=[11 22;33 44]** to make A become the 2x2 matrix given earlier.

The 2x3 matrix that A used to be has now been lost. The variable A still exists of course in MATLABs workspace but now has a different size and different values for its elements.

Part-4: Scrolling

Suppose you want to repeat or edit an earlier command. Instead of typing it again, there is a way to do it: MATLAB lets you search through your previous commands using the up-arrow and down-arrow keys. This is a very convenient facility, which can save a considerable amount of time.

1) Suppose you regret the last command, and that you want A still to be the 2x3 matrix you originally defined. Press the up-arrow key until the earlier command.

   A=[1 2 3;4 5 6] appears in the Command Window (the down-arrow key can be used if you go back too far). Now press enter. The matrix A is back to what it was originally.

You can speed up the scroll if you remember the first letters of the command you are interested in. For example, if you quickly want to jump back to the command **v = [1, 2, 3]** just type **v =** and then press the up-arrow-key. MATLAB will display the most recent command starting with **v =**.

You can also use the Command History window to jump to a previous command:

2) Go to the Command History window. Jump back to the command **v=[1,2,3]** by double-clicking on this line in the Command History window. The command will be carried out immediately.

Part-5: Basic arithmetic operations in MATLAB

1) Solve and verify that the product of the matrices $A = \begin{bmatrix} \mathbf{1} & \mathbf{2} \\ \mathbf{0} & \mathbf{1} \end{bmatrix}$ and $\boldsymbol{B} = \begin{bmatrix} \mathbf{4} & \mathbf{2} \\ \mathbf{3} & \mathbf{5} \end{bmatrix}$ is equal to $AB = \begin{bmatrix} 10 & 12 \\ 3 & 5 \end{bmatrix}$. Also, find $BA = \begin{bmatrix} 4 & 10 \\ 3 & 11 \end{bmatrix}$.
2) Create the matrices A and B in MATLAB.
3) Type **A*B** to perform the first multiplication. Because you did not give the result of this expression a name MATLAB calls it 'ans' (which is short for answer). Note that multiplication of matrices in MATLAB as in A*B is the *matrix multiplication (product)* operation.
4) Type **B*A** to perform the second matrix multiplication. The matrix 'ans' has now changed to be the result of this expression. In MATLAB 'ans' is the result of the last unnamed MATLAB command.
5) Type **A+B** to add the two matrices. Type **A-B** to subtract the matrices. Type **3*A** to multiply A by 3.

`A+B` or `B+A`	is valid if `A` and `B` are of the same size
`A*B`	is valid if `A`'s number of column equals `B`'s number of rows
`A^2`	is valid if `A` is square and equals `A*A`
α`*A` or `A*`α	multiplies each element of `A` by α

Array arithmetic operations:

Array arithmetic operations or array operations for short, are done element-by-element.

.*	Element-by-element multiplication
./	Element-by-element division
.^	Element-by-element exponentiation

Example: **k= A.*B**

Part-6: Keeping in touch with your variables

You can see/check all the variables created in the Workspace window. Alternatively you can use commands in the Command window to do the same:

1) Type **who** to see all your matrices.
2) Type **whos** to see a more complete description, including sizes, of all stored data.

Deleting matrices:

3) Remove the matrix B and the vector v by typing **clear B v**. Alternatively, you can remove B and v by selecting them in the Workspace window and then clicking the 'Delete' button. Remember that you don't have to clear matrices to use the same names again. You can just set up a new matrix with the same name.
4) Change the matrix A by typing **A=2**.
5) Check that A is now a *scalar* - the previous A matrix is lost. Again, we changed the size and the value of the elements of the matrix-variable A. The size is now 1x1.
6) *Remove all* variables by just typing **clear** without any names following.
7) Check that there are is nothing left in your workspace. Careful with this command!

Part-7: Generating variables

MATLAB provides functions to create several basic matrices automatically without having to type or read in each of the elements. The most important functions are

zeros **zeros(m,n)** creates an mxn matrix whose elements are equal to zero.
ones **ones(m,n)** creates an mxn matrix whose elements are equal to one.
eye **eye(m,n)** creates an mxn identity matrix
rand **rand(m,n)** creates an mxn matrix whose elements are all random number between 0 and 1

1) Create a 5x5 matrix with random numbers. Now add to it a diagonal matrix with the diagonal elements equal to 2. You can create such a diagonal matrix using the command **2*eye(5,5)** .
2) Type **h=10:2:20**. You should see that you have created a row vector h with elements starting at 10 and increasing in steps of 2 up to 20. We could also have created this vector with **h=[10 12 14 16 18 20]**. Note that we do NOT need brackets in **h=10:2:20**

3) If the step size is negative the sequence decreases. So **i=9:-1:5** sets i to be [9 8 7 6 5].
4) Type **r=50:55**. You should have the vector [50 51 52 53 54 55].

Suppression of output:

Up till now you have always seen the results of each command on the screen. This is often not required or even desirable; the output might clutter up the screen, or be long and take a long time to print. MATLAB suppresses the output of a command if you finish the command with a semi-colon **;.**

5) Create an vector by typing **rr=1:50;**. Observe that output of the command is supressed.

Building larger and sub matrices from original ones:

6) Delete all variables using **clear**.
7) Create the matrices $A = \begin{bmatrix} 5 & 3 \\ 8 & 9 \end{bmatrix}$ and $B = \begin{bmatrix} -3 & 0 & 4 \\ 2 & 5 & 0 \end{bmatrix}$
8) Now type **C=[A B]** or **C=[A,B]** and you should see that MATLAB put A and B side-by-side to get another matrix C.
9) Type **D=[A B;B A]** and you should see that MATLAB placed the matrix [B A] below the matrix [A B]
10) Now type **H=[A;B]**

This gives an error message, which tells you that this operation cannot be done. The number of columns in A and B are not equal.

11) To extract a submatrix P consisting of rows 2 and 3 and columns 1 and 2 of the matrix D, type **P = D([2 3],[1 2])**

Changing values of individual elements:

Individual elements in a variable can be identified using index numbers. For example the element 8 in the matrix A above can be referred to as A(2,1) because it is the first element of the second row of A. Note that the row and column indices are separated by a comma.

12) Change this element to have the value of 1 by typing **A(2,1)=1**
13) Type **A2 =A(1:2,1)** to extract the first column of A and store it in A2.

We could also achieved this by typing **A2 = A(:,1)** . The colon operator identifies all of the first column (taking the first element in ALL of the rows). Similarly **A(2,:)** identifies the whole of the second row of A. Note that MATLAB will give error if you use negative or zero indices for matrices.

However if you use an index which is above the maximum dimension the matrix is enlarged and all the intervening elements are set to zero.

14) Type **A(4,4)=2**.

 See how the extra elements of the matrix have been created, but only A(4,4) has been set to 2, the other extra elements are set to zero by default.

15) Swap the first and second columns of B by typing **B = [B(:,2) , B(:,1), B(:,3)]** .

Transposing matrices and vectors:

The operator ' (single quote) finds the transpose of a variable. For example, if A is the matrix $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ then **B = A'** gives the matrix $B = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$

16) Set up a column vector of the first 10 whole numbers by writing **(1:10)'**

Part-8: Saving and retrieving your work

All variables that you created in this MATLAB session are stored in MATLAB's workspace. Upon exiting MATLAB contents of workspace is deleted. So you must save your workspace if you want to use it in later MATLAB sessions.

1) Select **Save Workspace** under **Home** menu. A save window will come up. Check that the correct folder is given. Type in the name you wish to use for the workspace file, for example 'labsession1' and save the file. MATLAB automatically selects the 'mat' extension for the workspace files.

2) Type **clear** to delete your workspace. Type **who** to check that your workspace is empty (or check your Workspace window). Retrieve the saved workspace using **Import Data** under **Home** menu. Type **who** again to check that all your variables are back in the workspace (or check the Workspace window).

 This method of saving saves all the variables in your workspace. You can select any number of variables if you save from within the Command Window instead.

3) Create two random variables by typing **x=rand**; and **y=rand**; To save these two variables in a .mat file called **'savexy.mat'** type **save savexy x y** .

 The contents of your current directory is shown in the Current Folder. But you can also find out its contents using a command:

4) The command **dir** gives you the list of all files that are in your current directory. The command **what** gives you the files in the current folder relevant to MATLAB only. Try them both.

<u>Practice:</u>

1) Create a 4x3 matrix. Interchange the 2nd and 3rd rows using one line of code.
2) Create a 8x8 matrix with random numbers between 0 and 8. Now, make all elements in the first row and first column equal to 1.
3) We would like to create the row vector [10 8 6 4 2 0 ….. 0 0 10] with a total number of elements equal to 100 (that means there are 94 zeros in the vector). Think of two ways to create this variable without typing in all the numbers.

Part-9: Script files

Let us see how to store commands in a file. Such MATLAB are known as script files. They are stored with the .m extension. In MATLAB files with the .m extension are also called M-files.

1) From the Home menu choose *New Script*. File then New then M-file.
2) Type the following lines in the new file window. Note that the file is still called '*Untitled*'.

```
% Add two matrices and display the result
clc;
A = [1, 1; 2, 2];
B = [2, 2; 3, 3];
D = A + B;
disp('The result is ');
disp(D);
```

The first line is a comment written to explain what your program does. MATLAB ignores everything on a line after a % character.

Now save this file. Saving option is there under '*Editor'* Menu.

3) *Save* the file by clicking on '***save'*** or '***save as'***. Save the file in the folder you already created. See to that '*save as type'* has *.m* extension.

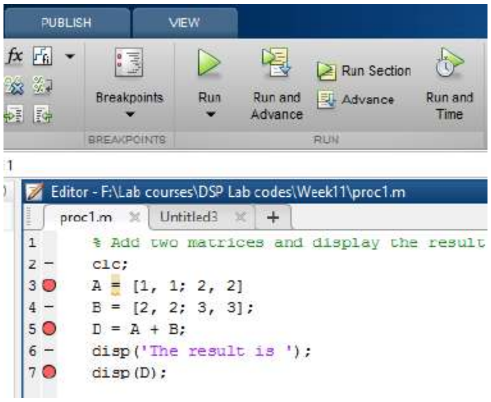
<u>Rules for naming variable/script files:</u>
Start with a letter, followed by letters or numbers or underscore, maximum 64 characters (excluding the .m extension), and must not be the same as any MATLAB reserved word

4) Run the file by clicking on '***Run'*** under Editor Menu.

<u>Running a program step-by-step:</u>

Often it is convenient to run a program step by step to carefully check that all the lines of code that you wrote are doing what you expect them to do. This is a good way to debug your code to fix the errors in your code.

5) Click on the first command line of the .m file already. Go to Breakpoints option under Editor Menu. Click on '*Set/Clear*'. Click the mouse to on the code to mention the breakpoints in .m file as shown in the figure. Remove the semi-colon from the end of command lines where breakpoints are present. Save the file.



6) Next click on '*Run*'. See the partial result on Command window. If multiple breakpoints are present, you need to click '*Run*' multiple times to see all the partial outputs. To come out of debugging mode click on '*Quit Debugging*'.
7) To remove the breakpoints click on '*clear all*' under '*Breakpoints*' Menu.

Under Breakpoints on the menu there are more options. Explore the MATLAB debugger and use it in this lab course. It is a very useful tool.

Also, know to comment out parts of your cod or uncomment using Comment menu under Editor Toolbar.

Part-10: Logical expressions

Relational operators:

eq	- Equal	==
ne	- Not equal	~=
lt	- Less than	<
gt	- Greater than	>
le	- Less than or equal	<=
ge	- Greater than or equal	>=

In MATLAB a logical expression has two possible values that are 'true' or 'false' but numeric, i.e. 1 if the expression is true and 0 if it is false.

1) Go back to the Command Window. Set up the scalars **q=10**; **w=10**; and **e=20**.
2) Type **w< e** to see that the result is given the value of 1 because w is indeed less than e. Now type **q==e**. The == operator checks if two variables have the same value. Therefore the answer is 0 in this case.

3) Relational operations are performed after arithmetic operations. For example **q == w-e** results in 0. Parentheses can be used to override the natural order of precedence. So **(q==w) - e** results in -20.
4) Create the vectors x and y by typing **x=[-1, 3, 9]** and **y=[-5, 5, 9]** . Now type **z= (x < y)**. The i-th element of z is 1 if x(i) < y(i), otherwise it is 0. So, the answer is the vector with elements 0, 1, and 0.

<u>Logical operators:</u> Please go through logical operators**.**

and	- Element-wise logical AND	&
or	- Element-wise logical OR	|
not	- Logical NOT	~

5) Give the answer for **(e>0) & (q<0)**
6) Type **(e>0) | (q<0)**. What is the answer?

<u>Part-11: Solving linear equations</u>

In linear algebra we learn that the solution to Ax = b can be written as $x = A^{-1}b$, where A^{-1} is the inverse of A.
For example, consider the following system of linear equations:

$$x + 2y + 3z = 1$$
$$4x + 5y + 6z = 1$$
$$7x + 8z = 1$$

1) To get the solution type g=inv(a)*b where $a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 0 & 8 \end{bmatrix}$ and $b = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$ or **g=a\b**

<u>Matrix Functions:</u>

Some of the Matrix based MATLAB commands are given below.

`det`	Determinant
`diag`	Diagonal matrices and diagonals of a matrix
`eig`	Eigenvalues and eigenvectors
`inv`	Matrix inverse
`norm`	Matrix and vector norms
`rank`	Number of linearly independent rows or columns

<u>Part-12: The if statements</u>

MATLAB supports the variants of ‘if’ construct.

1) **if …… end**

Example:

```
mark=30;
if mark<40
disp('F grade');
disp(mark);
end;
```

2) **if … else … end**

Example:

```
mark=80;
if mark>40
disp('P grade ');
else
disp('F grade');
end;
```

3) **if … elseif … else … end**

Example:

```
mark=80;
if mark>70
disp('Excellent ');
elseif mark>40
disp('P grade ');
else
disp('F grade');
end;
```

Part-13: The for loop

Example-1

```
angle = pi*rand(3,1);
for k=1:3
cosangle = cos(angle(k));
disp(cosangle);
end;
```

Example-2

```
x = rand(4,1);
maxval = 0;
for k=1:4
if (x(k) > maxval)
maxval = x(k);
```

```
end;
end;
disp(maxval);
```

Part-14: Nested for-loops

Example

```
A = rand(2,2);
for k=1:2
for m=1:2
exponential = exp(A(k,m));
display(exponential);
end;
end;
```

Part-15: While loops

Example

```
x = 15;
while x > 0
x = x-4;
disp(x);
end;
disp('Done');
```

Part-16: Writing your own MATLAB functions

The first MATLAB function you will write is a simple function to convert temperatures given in degrees Fahrenheit into degrees Celsius, according to the formula:

$$T_C = (T_F - 32)\frac{5}{9}$$

1) Open a new script file. Type the following code:

```
function tc = fahtocel(tf)
% converts function input 'tf' of temperatures in Fahrenheit to
% function output 'tc'of temperatures in Celsius
temp = tf-32;
tc = temp*5/9;
return;
```

The word function on the first line tells MATLAB that you are about to define a function. The next words tc = fahtocel(tf) tell MATLAB that:
- the function is called 'fahtocel'
- the function returns a function output variable called 'tc'
- the function is given a function input variable called 'tf'.

The return statement indicates the end of the function and asks MATLAB to return to the place from which the function was called. You can call functions from the command window of course, or from script files, or even from other functions.

Function M-files must always start with the function statement and the name of the function M-file must always be the same as the name of the function. So here, we must save this M-file as 'fahtocel.m'.

2) Save the M-file as 'fahtocel.m'. MATLAB will suggest the name fahtocel.m to you automatically and all you have to do is press the save button.

Calling the function from a script file:

3) Open a new script file. Type the following code:

```
tempf=1000;
tempc=fahtocel(tempf)
```

4) Save and run the file.

Functions with more than one output variable:

5) Type the below function file

```
function [vol, surf] = cuboid(len,br,dep)
% Function to return the volume and surface area of a cuboid given
% the length len, breadth br and depth dep
vol = len*br*dep;
surf = 2*(len*br + len*dep + br*dep);
return;
```

6) Write the script file:

```
% script file for cuboid
length =[ 5, 10];
breadth = [4, 10];
depth = [3, 8];
% call the function
[volume, surface] = cuboid(length,breadth,depth)
```

7) Save and run the file.

Part-17: Plotting using MATLAB

Basic Two-Dimensional Plot:

```
t=0:0.01:2;
x=2*sin(2*pi*50*t);
plot(t,x)
xlabel('Time')
ylabel('signal amplitude')
title('first plot')
```

Multiple plots in single window:

```
x=0:pi/16:2*pi;
y1=sin(x);
y2=cos(x);
plot(x,y1,'* -',x,y2,'r s -')
xlabel('x')
ylabel('sin(x), cos(x)')
title('Trig Functions')
legend('sin','cos')
```

Subplots:

```
x=0:pi/16:2*pi;
y1=sin(x);
y2=cos(x);
subplot(2,1,1)
plot(x,y1,'* -')
xlabel('x')
ylabel('sin(x)')
subplot(2,1,2)
plot(x,y2,'r s -')
xlabel('x')
ylabel('cos(x)')
```

Tutorial - II

Time Domain Analysis of Discrete-time LTI Systems

1. Plotting discrete time signals:

Plot the discrete-time signal 0.5^{n} for $-10 \leq n \leq 10$

```
clc;
clf;
clear
n=-10:10;
x1= 0.5.^n;
stem(n,x1)
```

2. Generation of signals:

A small set of elementary signals is needed in this exercise. To begin, create the following signals of length 16 ($0 \leq n \leq 15$)

- b[n], a 16-point block sequence with unit amplitude.
- r[n), the first 16 points of the ramp function, defined as nu[n].
- t[n], 16 points of a periodic triangular wave with period 8, a maximum value of one, a minimum value of 0, and starting point n = 0.
- e[n], the first 16 points of the one-sided exponential, (5/6)nu[n).

Write the MATLAB code to plot the above signals. Using the elementary signals just created, write the code to plot the following new signals:

i. $v[n] = r[n-6]u[n]$
ii. $z[n] = t[n](u[n] - u[n-10])$

```
%% Workspace Initialization.
 clc; clear; close all;

%% Generate the basic signals of common length 16.
N = 16;
n = 0:N-1;

b = ones(1,N);  % Block of ones.
r = n;              % Ramp function.

P = 8;  % Triangular wave period.
n1 = 0:P/2-1;
n2 = P/2:P-1;
P1 = P*ones(1,length(n2));
A = 1;

tri_block = [2*A*n1/P 2*A*(P1-n2)/P]  ;
t = [tri_block tri_block]; % % Periodic Triangular wave
```

```
e = (5/6).^n; % One sided exponential.

figure(1)
stem(n,b);
figure(2)
stem(n,r);
figure(3)
stem(n,t);
figure(4)
stem(n,e);

%% i. Create and display  r[n-6]*u[n].
figure('Name', 'Tutorial-2. Elementary Signals');
stem(n,r);
grid;
hold on;
stem(n+6,r,'r*');
title('r[n] (blue) and v[n]=r[n-6]*u[n] (red)');

%% ii. Create and display z[n]=t[n]*(u[n]-u[n-10]).
z = [t(1:10) zeros(1,6);];
figure('Name',' Tutorial-2. Elementary Signals');
stem(n,t);
grid;
hold on;
stem(n,z,'r*');
title('t[n] (blue) and z[n]=t[n]*(u[n]-u[n-10]) (red)');
```

3. Check the output of the following code:

i)
```
t=-10:0.01:10;
f=heaviside(t);
plot(t,f)
```

ii)
```
t=-10:0.01:10;
g=heaviside(t-3);
figure(1)
plot(t,g)
axis([-15 15 -1 2])
```

iii)
```
t=-10:0.01:10;
l=heaviside(-2*t+2);
figure(4)
plot(t,l)
axis([-15 15 -1 2])
```

Addition & Subtraction of signals:

4. Perform the operations $x_1 + x_2$ and $x_1 - x_2$ where $x_1 = 0.5^{\mathrm{n}}$ and $x_2 = 0.8^{\mathrm{n}}$ for $-10 \le n \le 10$

```
clf;
clear
n=-10:10;
x1= 0.5.^n;
x2= 0.8.^n;
x=x1+x2;
 y=x1-x2;
 subplot(221)
stem(n,x1)
 subplot(222)
stem(n,x2)
subplot(223)
stem(n,x)
 subplot(224)
 stem(n,y)
```

5. Compute and plot the output response of the LTI system whose impulse response
   h[n] = 1 ; 0≤ n ≤ 5
   0 ; otherwise for the input, x[n] = {1 4 2 6}
   ↑

```
clf;
clear
x=input('enter input sequence')  %x = [1 4 2 6];
h=input('enter h(n)')   %h = [1 1 1 1 1 1];
x1=length(x);
h1=length(h);
s1=x1+h1-1; n=0:s1-1;
y = conv(x,h)
stem(n,y)
```

6. For the LTI systems described by the following difference equations, generate its impulse response, and unit step response.

   i. *y[n] = x[n] + 2x[n-1]*
   ii. *y[n] = 0.9 y[n-1] + x[n]* Also find the analytical expression.
   iii. *y[n]-0.3695y[n-1]+0.1958y[n-2]=0.2066x[n]+0.4131x[n-1] +0.2066x[n-2]*

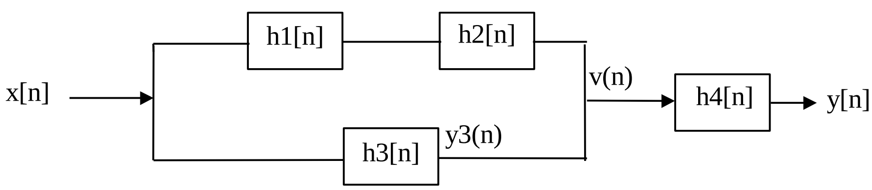
Sample Solution (iii)

```
% Time domain response of difference equations
% y[n] - 0.3695 y[n-1]+0.1958y[n-2]=0.2066x[n]+0.4131x[n-1]+0.2066x[n-2]
```

```
b1 = [0.2066   0.4131   0.2066];
a1 = [1   –0.3695    0.1958];
%    Impulse response of the system
        impz(b1,a1,20);
        title('Impulse response');

%    Step response of the system
     step_n = [ones(1,20)];
     y = filter(b1,a1,step_n);
     stem(y);
     title('Step response');
```

7. Compute the overall impulse response of the system shown in figure.



h1[n] = (½)n $0 \le n \le 5$
0   otherwise

h2[n] = 1 $0 \le n \le 5$
0   otherwise

y3[n] = 0.25 x[n] + 0.5x[n-1] + 0.25 x[n-2]

y[n] = 0.9 y[n-1] – 0.81 y[n-2] + v[n] + v[n-1]

Code to find the overall impulse response of the system given:

```
n=0:5;
h1=(0.5).^n; h2=ones(1,6);
h5=conv(h1,h2);
% y3(n) = 0.25 x(n)+0.5 x(n-1)+0.25 x(n-2);
h3 = impz([0.25 0.5 0.25],1,max(size(h5)));
h6 = h5 + h3';
% y(n) = 0.9 y(n-1) - 0.81 y(n-2)+v(n)+v(n-1);
n4=[1 1 0]; d4=[1 -0.9 0.81];
 h4 = impz(n4,d4,20);
h = conv (h6, h4); x=0:29;
stem(x,h); grid on; title('Overall Impulse Response')
```

Transform domain representation

$$\boldsymbol{H(z) = \frac{1 - 5z^{-1} + 6z^{-2}}{1 - 2.5z^{-1} + z^{-2}}}$$

- *To enter a transfer function*

```
%    Z - domain
%    H(z) = (1-5z^(-1)+6z^(-2))/(1-2.5z^(-1)+z^(-2))
       b2 = [1 -5 6]; a2 = [1 -2.5 1];
       h  = [b2,a2];
       sys=tf(b2,a2,1e-3)
%    To display H(z) as a rational function of ‘z’
       printsys(b2,a2,’z’);
```

- *Transfer function to zero-pole conversion (tf2zp)*

```
[z,p,k] = tf2zp(b2,a2)
```

- *To obtain the pole-zero map*

```
zplane(b2,a2);
title(‘ Pole- zero plot of H(z) ‘)
```

- *To find the Partial Fractions of the Transfer function*

```
[r,p,k] = residuez(b2,a2)
```

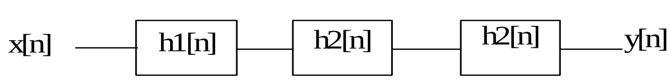
Questions for practice:

1) A small set of elementary signals is needed in this exercise. To begin, create the following signals of length 16 ($0 \le n \le 15$)
   - b[n], a 16-point block sequence with unit amplitude.
   - e[n], the first 16 points of the one-sided exponential, $\left(\frac{5}{6}\right)^n u[n]$.

Write the MATLAB code to plot the above signals. Using the elementary signals just created, write the code to plot the following new signals:

i. $y[n] = e[n + 10]b[n]$
ii. $e_e[n] = even\{e[n]\}(u[n + 5] - u[n - 5])$

2) A cascade of three LTI systems is shown below



If h2[n] = U[n] – U[n-2] and overall impulse response is {1 5 10 11 8 4 1} starting at n=0, find h1[n] and verify the result analytically. (use MATLAB function deconv) Also find the response of the overall system to the input x[n]= δ[n] - δ[n-1].

3) Transform the system described by y[n]-0.3695y[n-1]+0.1958y[n-2]=0.2066x[n] +0.4131x[n-1] +0.2066x[n-2] to zero-pole form and residue form. Plot pole-zero map and comment on stability.

4) Compute the causal inverse of $H(z) = \dfrac{z^{-1} + 0.5z^{-2}}{1 - 0.6z^{-1} + .08z^{-2}}$

TUTORIAL 3

Objectives:

Frequency domain representation of discrete-time signals – DFS, DTFT, DFT

Note: Students need to revise the theory covered in this tutorial
Write the code in the observation book (Calculation not required)

Frequency domain response

1. Write a function that can synthesise a waveform in the form

$$x(t) = \mathrm{Re}\left\{\sum_{k=1}^{N} X_k e^{j2\pi k f_o t}\right\}$$ where f_o is the fundamental frequency.

For f_o = 25Hz, $X_k = j4/k\pi$ for k odd and 0 for k even, plot $x(t)$ for N=5,10 and 25. Explain what happens when $N\rightarrow\infty$. repeat the synthesis with f_o=1kHz and listen to the cases N=1,2,3,4,5,10. Ensure that the sampling frequency fs in sound(x,fs) is high to prevent aliasing.

```
clear; clc; clf;
T = 0.12; sum = 0; f0 = 25; fs = 1000;
%Change f0 and N values
N=25;
t = 0; i = 1;
while t < T
    sum = 0;
    k = 1;
    while k <= N
        X = j*4/(k*pi);
        sum = sum+X*exp(j*2*pi*k*f0*t);
        k=k+2;
    end;
    x(i)=real(sum);
    t=t+1/fs;
    i=i+1;
end;
t= 0:1/fs:T-1/fs;
plot(t,x)
title('Wave form for N=25')
```

2. Find the DFS coefficients of the signal $x[n]=1+sin(\pi n/12 + 3\pi/8)$

```
clear; clc;
n= 0:23; phi=3*pi/8;
x=1+sin(pi*n/12 + phi);
X=fft(x,24)/24;
```

3. For the LTI systems described by the following difference equations, generate its frequency response. Comment on the type of response.
   i. *y[n] = 0.5x[n] + 0.5x[n-1]*
   ii. *y[n] = 0.9 y[n-1] + x[n]*
   iii. *y[n]-0.3695y[n-1]+0.1958y[n-2]=0.2066x[n]+0.4131x[n-1] +0.2066x[n-2]*

   Sample Solution (iii)

```
  % Frequency domain response of
  % difference equations
  % y[n]-0.3695y[n-1]+0.1958y[n-2] =
  % 0.2066x[n]+0.4131x[n-1]+
  % 0.2066x[n-2]

b1 = [0.2066 0.4131 0.2066];
a1 = [1 -0.3695 0.1958];
    freqz(b1,a1,64);
    title(‘ Frequency response’)
```

Write the programme for the remaining three bits (i and ii) and plot the response

4. Determine and plot the real and imaginary parts and the magnitude and phase spectra of the following DTFT for various values of *r* and *θ*.

$$G(e^{j\omega}) = \frac{1}{1 - 2r\cos(\theta)e^{-j\omega} + r^2 e^{-j2\omega}} \qquad 0 < r < 1$$

```
clear; clc;
theta=60*pi/180;
r=0.5
w= -pi:pi/100:pi;
den = 1 -2*r*cos(theta)*exp(-j*w) +r^2*exp(-j*2*w);
G=1./den;
subplot (221), plot(w, real(G));
title('Real part')
subplot (222), plot(w, imag(G));
title('Imaginary part')
subplot (223), plot(w, abs(G));
title('Magnitude')
subplot (224), plot(w, angle(G));
title('Phase')
```

5. Compute and plot the DTFT of the following sequence and observe the properties
   $s[n] = A\cos(2\pi f_0 n + \phi)$ Try for f_o=100Hz, $\phi=\pi/6$ and different lengths of sequence.

```
clear; clc; clf;
A = 1; f0 = 100; phi = pi/6; fs = 2000;
N= input('Enter the  length of sequence:');
```

```
n = 0: 1/fs : (N-1)/fs;
sig = A * cos (2*pi*f0*n + phi);
w_axis = linspace(-1, 1, 1024);
time_axis=0:length(sig)-1;
F=fftshift(fft(sig,1024));
sig_dtft=abs(F);
subplot(2,1,1), stem(time_axis,sig);
title('Signal');
xlabel('Index');
subplot(2,1,2), plot(w_axis,sig_dtft);
title('DTFT Magnitude');
xlabel('Digital Frequency');
```

6. *Three domains:* Relation between location of poles and zeroes in z plane, impulse response and frequency response.
   - **i.** **y(n) = 0.77y(n-1)+x(n)+x(n-1)**
   - **ii.** **y(n)=0.77y(n-1)+0.77x(n)-x(n-1)**
   - **iii.** **$H(z)=1-z^{-1}/1+0.77z^{-1}$**
   - **iv.** **$H(z)=1-z^{-1}+z^{-2}-z^{-3}+z^{-4}-z^{-5}$**
   - **v.** **y(n)=x(n)+x(n-1)+x(n-2)+x(n-3)+x(n-4)+x(n-5)**
   - **vi.** **$H(z)=3-3z^{-1}$**

```
%solution for 6(i)
clear; clc; clf;
num=[1 1]; den=[1 -0.77];
[z,p,k]=tf2zp(num,den);
% pole-zero plot
figure(1);
zplane(z,p);
title('Pole-zero plot');
% impulse response plot
figure(2);
impz(num,den,20);
title('Impulse response plot');
% Frequency response plot
figure(3);
freqz(num,den,64);
title('Frequency response plot');
```

Write the programme for the remaining three bits (ii, iii, iv, v, and vi) and plot the response

7. Compute and plot the DFT of the following sequences and observe their properties
   - i. Unit impulse signal; *xi = {1 0 0 0 0 0 0 0}*
   - ii. All ones; *x1 = {1 1 1 1 1 1 1 1}*
   - iii. Three point boxcar; *xb = {1 1 1 0 0 0 0 0}*
   - iv. Symmetric boxcar; *xbsy = {1 1 0 0 0 0 0 1}*

```
%solution for 7(iii)
N = 8; nn = 0:(N-1); kk = nn;
xb = [1 1 1 0 0 0 0 0];
Xb = fft(xb,N);

subplot(221), stem(nn,xb);
title(' x(n) '); xlabel(' Index (n) ');
axis([0 7 0 1]);
subplot(222), stem(kk,real(Xb));
title(' Real part of DFT ');
xlabel(' Index (k) ');
axis([0 7 -1 4]);
subplot(224), stem(kk,imag(Xb));
title(' Imag part of DFT ');
xlabel(' Index (k) ');
axis([0 7 -2 2]);
```

Write the programme for the remaining three bits (i, ii, and iv) and document the result

8. Generate a real-valued test signal *v[n]* using the MATLAB function rand of length *N=15 or 16*. Compute DFT of *v[n]* to get *V[k]* and then try the following for even and odd lengths
   a) Compute the even and odd parts of *v[n]*,
   $v_e[n]$ = 0.5 [v(n) + v[-n mod N]]
   v(-n mod N) = [v(1), v(N:-1:2)]
   $v_o[n]$ = 0.5 [v[n] - v[-n mod N]]
   b) Compare DFT $v_e[n]$ with *Re V[k]* and DFT $v_o[n]$ with *Im V[k]*

```
clear;clc;clf;
N=15;nn=0:(N-1);kk=nn;
v=rand(1,N);
V=fft(v,N);
vfold=[v(1) v(N:-1:2)];
veven=0.5*(v+vfold);
vodd=0.5*(v-vfold);
Veven=fft(veven,N);
Vodd=fft(vodd,N);

subplot(321), stem(nn,veven);
title('veven(n)');
axis([0 N-1 0 1]);
subplot(322), stem(nn,vodd);
title('vodd(n)');
axis([0 N-1 -1 1]);
subplot(323), stem(kk,real(V));
title('ReV(k)');
axis([0 N-1 -2 10]);
subplot(324), stem(kk,Veven);
title('Ve(k)');
axis([0 N-1 -2 10]);
subplot(325), stem(kk,imag(V));
```

```
title('ImV(k)');
axis([0 N-1 -2 2]);
subplot(326), stem(kk,imag(Vodd));
title('Im(Vo(k))');
axis([0 N-1 -2 2]);
```

9. Generate a signal *s[n]* with three sinusoidal components at 50, 120 and 240Hz corrupted by AWGN. Plot the spectrum and identify the signal components.

```
% identification of sinusoids in noise
fs=2000;
t = (0:199)/fs;
s = sin(2*pi*50.*t) + sin(2*pi*120.*t) +sin(2*pi*240.*t);

awgn = (0.5*randn(1,200)+.25);   % N(0.25, 0.25)
sn = s+awgn;
subplot(211), plot(t,sn);
title(' Sinusoid with noise'); grid;

Sn = fft(sn,200);
f = 0:10:990;
sfmag = abs(Sn);
subplot(212), plot(f,sfmag(1:100));
title(' Spectral estimation'); grid;
```

TUTORIAL 4

Objective:

Design of FIR filters using windowing, frequency sampling technique

Note: Students need to revise the theory covered in this tutorial
Write the code in the observation book (Calculation not required)

Filter structures

1. Determine the Cascade and Parallel form structure for the filter with transfer function

$$H(z) = \frac{0.44z^2 + 0.362z + 0.02}{z^3 + 0.4z^2 + 0.18z - 0.2}$$

```
num = [0 0.44 0.362 .02];
den = [1 0.4 0.18 -0.2];
[cascade_sos,G] = tf2sos(num,den);
[r p k] = residuez(num,den)
```

2. Determine the lattice form structure for the filter transfer functions given below. *Use MATLAB function poly2rc.* Also check the stability of the filters.

$$a)\ H(z) = 1 + 1.2z^{-1} + 1.12z^{-2} + 0.12z^{-3} - 0.08z^{-4}$$

$$b)\ H(z) = \frac{1 + 1.6z^{-1} + 0.6z^{-2}}{1 - z^{-1} - 0.25z^{-2} + 0.25z^{-3}}$$

```
% Sample Solution (2a)

num2a = [1 1.2 1.12 0.12 -0.08];
den2a = [1];
k2a = poly2rc(num2a);
% Sample Solution (2b)
num2b = [1 1.6 0.6];
den2b = [1 -1 -0.25 0.25];
[k2b,v2b] = tf2latc(num2b,den2b);
```

FIR Filters

3. Determine the impulse response of an ideal lowpass filter with linear phase characteristics. Truncate the impulse response at different lengths, say N=11, 21, 31, 41 and observe the magnitude response of the filters. *Gibbs phenomenon*

```
clear;clc;clf;
M=input('enter the length of the filter:');
w_c=pi/3;
Mby2=(M-1)/2;
n=0:M-1;
h_d = sin(w_c*(n-Mby2))./(pi*(n-Mby2));
h_d(Mby2+1) = w_c/pi;
[H,w] = freqz(h_d,1);
```

```
subplot(211), stem(n,h_d);
subplot(212), plot(w/pi,abs(H));
figure
freqz(h_d,1);
```

4. Consider a low pass filter with ω_p=0.2π and ω_s =0.3π. Design a FIR filter using frequency sampling method. Plot the frequency response of the designed filter and determine the ripple in the passband (R_p) and minimum stopband attenuation (A_s). Try for filter length M=20 and M=40.

```
w_p=0.2*pi; w_s=0.3*pi;
M = input('Enter order of filter:');
pass=fix(w_p*M/(2*pi))+1; % kp
stop=fix(w_s*M/(2*pi))+1; % ks
trans=stop-pass;
if rem(M,2)==0,U=M/2 -1;else U=(M-1)/2;end;
if trans==1,
Hr=[ones(1 ,pass),zeros( 1 ,U-pass+1)];
else
tk = pass+ 1 :stop;
trans_mag=0.5*(1+cos(pi*(tk-pass)/trans)); % raised
coosine in transition band
     Hr=[ones(1,pass),trans_mag, zeros(1,U-stop+1)];
end;
k=0:U;
G=(-1).^k.*Hr;
if rem(M,2)==0,
G_M=[G 0 -G(U+1:-1 :2)]; else
G_M=[G -G(U+1:-1 :2)];
end;
I=0:M-1 ;
H=G_M.*exp(pi*I*j/M);
h=ifft(H);
Mag = abs(fft(h,512));
w=[0:255]*pi/256; plot(w,Mag(1:256));
figure;
freqz(h,1,512);
```

5. Design the above filter using Hanning window, Hamming window, Blackmann window, and Bartlett window. Plot the impulse response, amplitude response and zero locations of the designed filter and compare their performance.

Characteristics of commonly used window functions

Window function	*Approximate Transition width $\Delta\omega$*	*Exact Transition width $\Delta\omega$*	*Minimum stop band attenuation A_s dB*
Rectangular	4π/M	1.8π/M	21
Hamming	8π/M	6.2π/M	44
Hanning	8π/M	6.6π/M	53
Bartlett	8π/M	6.1π/M	25
Blackmann	12π/M	11π/M	74

```
% lowpass design using window functions clear;close all;
        w_p=0.2*pi;
        M = input('Enter order of filter:');
        h_hann=fir1 (M,w_p/pi,hann(M+1));
        h_hamm=fir1 (M,w_p/pi,hamming(M+1));
        h_blackman=fir1 (M,w_p/pi,blackman(M+1));
        h_bartlett=fir1 (M,w_p/pi,bartlett(M+1));
        freqz(h_hann, 1,512);title(['Hanning window, M='
        ,int2str(M)]);
        figure;
        freqz(h_hamm,1,512);title(['Hamming window, M='
        ,int2str(M)]);
        figure;
        freqz(h_blackman,1,512);title(['Blackman window,
        M=',int2str(M)]);
        figure;
        freqz(h_bartlett,1 ,512);title(['Bartlett window,
        M=' ,int2str(M)]);
        figure;
        n=0:M;
        subplot(2,2,1 ),stem(n,h_hann);
        title(['Hanning window, M=',int2str(M)]);
        subplot(2,2,2),stem(n,h_hamm);
        title(['Hamming window, M=' ,int2str(M)]);
        subplot(2,2,3),stem(n,h_blackman);
        title(['Blackman window, M=' ,int2str(M)]);
        subplot(2,2,4) ,stem(n ,h - bartlett);
        title(['Bartlett window, M=' ,int2str(M)]);
        figure;
        subplot(2,2,1 ),zplane(h_hann,1 );
        title(['Hanning window, M=' ,int2str(M)]);
        subplot(2,2,2),zplane(h_hamm,1 );
        title(['Hamming window, M=' ,int2str(M)]);
        subplot(2,2,3),zplane(h_blackman,1 );
        title(['Blackman window, M=' ,int2str(M)]);
        subplot(2,2,4 ),zplane(h_bartlett,1 );
        title(['Bartlett window, M=' ,int2str(M)]);
```

6. Design a FIR low pass filter of order 20 with the following frequency response using Remez exchange algorithm

$$H(\omega) = 1;\ 0 \le \omega \le 0.4\pi.$$
$$0;\ 0.5\pi \le \omega \le \pi.$$

Problem may be extended for the design of high pass, band pass and band reject filters.

```
% FIR filter design using Remez Exchange method
n = 20;   % length of filter
f = [0  0.4  0.5  1];    % filter specs
m = [1  1    0    0];
bfir = remez(n,f,m)
[hfir,wfir] = freqz(bfir);
plot(f,m,wfir/pi,abs(hfir),'-');
title(' n=20 FIR LPF');
```

7. Design a linear phase FIR bandpass filter to satisfy the following specifications:

Passband	8-12 kHz
Stopband ripple	0.001
Peak passband ripple	0.0015
Sampling frequency	44.14 kHz
Transition width	3 kHz

Obtain the filter coefficients and compare the frequency response for the filter using (a) window method (b) frequency sampling method and (c) optimal method

```
clear; clc; close all;
h=fir1(82,[0.3624 0.5437],blackman(83));
freqz(h,1,512)
figure
% mod5_7b
f = [0 5 8 12 15 22.07]/22.07;
m = [0 0 1 1 0 0];
b = fir2(80,f,m);
[h,w] = freqz(b,1,128);
plot(f,m,w/pi,abs(h));
legend('ldeal','fir2 Designed')
title('Comparison of Frequency Response
Magnitudes');
figure
% mod5_7c
rp = 0.0015;   % Passband ripple
rs = 0.001;  % Stopband ripple
fs = 44140; % Sampling frequency
f = [5000 8000 12000 15000]; % Cutoff frequencies
a = [0 1 0 ]; % Desired amplitudes
dev=[rs rp rs]; % deviations
[n, fo, mo, w]=remezord(f, a, dev, fs);
hopt = remez(n,fo,mo,w);
[H ,W]=freqz(hopt,1,1024);
```

Tutorial-5

IIR Filters

Objective:

Design of IIR filters using Butterworth, Chebyshev, Elliptical approximations, and direct method.

1. Design an IIR low pass filter with passband edge at 1000Hz and stopband edge at 1500Hz for a sampling frequency of 8000Hz. The filter is to have a passband ripple of 0.5 dB and a stopband ripple below 30 dB. Plot the frequency response and compare the performance of the filters designed with Butterworth, Chebyshev I, Chebyshev II and Elliptic approximations and using both impulse invariant and bilinear transformations.

```
clear; clc; close all;
Fs=8000;
wp=1000*2/Fs; ws=1500*2/Fs;
wpa=1000*2*pi; wsa=1500*2*pi;
Rp=0.5; Rs=30;
 % Butterworth filter
% 1. Bilinear transformation
[Nb1,wn_b1]=buttord(wp,ws,Rp,Rs);
[num_but1,den_but1]=butter(Nb1,wn_b1);

% 2. Impulse invariant transformation
[Nb2,wn_b2]=buttord(wpa,wsa,Rp,Rs,'s');
[num_but2a,den_but2a]=butter(Nb2,wn_b2,'s');
[num_but2,den_but2]=impinvar(num_but2a,den_but2a,Fs);

f=0:10:Fs/2;
hb1=freqz(num_but1,den_but1,f,Fs);
hb2=freqz(num_but2,den_but2,f,Fs);
plot(f,20*log10(abs(hb1)),f,20*log10(abs(hb2)));
Title('Butterworth filter');axis([0 4000 -60 1]);
figure; plot(f,abs(hb1),f,abs(hb2));
Title('Butterworth filter');

% Chebyshev I filter
% 1. Bilinear transformation
[Nc11,wn_c11]=cheb1ord(wp,ws,Rp,Rs);
[num_c11,den_c11]=cheby1(Nc11,Rp,wn_c11);

% 2. Impulse invariant transformation
[Nc12,wn_c12]=cheb1ord(wpa,wsa,Rp,Rs,'s');
[num_c12a,den_c12a]=cheby1(Nc12,Rp,wn_c12,'s');
[num_c12,den_c12]=impinvar(num_c12a,den_c12a,Fs);
```

```
f=0:10:Fs/2;
hc11=freqz(num_c11,den_c11,f,Fs);
hc12=freqz(num_c12,den_c12,f,Fs);
figure;plot(f,20*log10(abs(hc11)),f,20*log10(abs(hc12)
));
Title('Chebyshev I filter');axis([0 4000 -60 1]);
figure; plot(f,abs(hc11),f,abs(hc12));
Title('Chebyshev I filter');

% Chebyshev II filter
% 1. Bilinear transformation
[Nc21,wn_c21]=cheb2ord(wp,ws,Rp,Rs);
[num_c21,den_c21]=cheby2(Nc21,Rs,wn_c21);

% 2. Impulse invariant transformation
[Nc22,wn_c22]=cheb2ord(wpa,wsa,Rp,Rs,'s');
[num_c22a,den_c22a]=cheby2(Nc22,Rs,wn_c22,'s');
[num_c22,den_c22]=impinvar(num_c22a,den_c22a,Fs);

f=0:10:Fs/2;
hc21=freqz(num_c21,den_c21,f,Fs);
hc22=freqz(num_c22,den_c22,f,Fs);
figure;plot(f,20*log10(abs(hc21)),f,20*log10(abs(hc22)
));
Title('Chebyshev II filter');axis([0 4000 -60 1]);
figure; plot(f,abs(hc21),f,abs(hc22));
Title('Chebyshev II filter');

% Elliptic filter
% 1. Bilinear transformation
[Ne1,wn_e1]=ellipord(wp,ws,Rp,Rs);
[num_e1,den_e1]=ellip(Ne1,Rp,Rs,wn_e1);

% 2. Impulse invariant transformation
[Ne2,wn_e2]=ellipord(wpa,wsa,Rp,Rs,'s');
[num_e2a,den_e2a]=ellip(Ne2,Rp,Rs,wn_e2,'s');
[num_e2,den_e2]=impinvar(num_e2a,den_e2a,Fs);

f=0:10:Fs/2;
he1=freqz(num_e1,den_e1,f,Fs);
he2=freqz(num_e2,den_e2,f,Fs);
figure;plot(f,20*log10(abs(he1)),f,20*log10(abs(he2)))
;
Title('Elliptic filter');axis([0 4000 -60 1]);
figure; plot(f,abs(he1),f,abs(he2));
Title('Elliptic filter');
```

2. Design an IIR low pass filter for the above specifications using direct method. *Use MATLAB function* yulewalk.

```
clear; clc; close all;
Fs=8000;
wp=1000*2/Fs; ws=1500*2/Fs;
wpa=1000*2*pi; wsa=1500*2*pi;
Rp=0.5; Rs=30;
% Direct method
f = [0 wp ws 1];
m = [1 1 0 0];

[b,a] = yulewalk(10,f,m);
[h,w] = freqz(b,a,128);
plot(f,m,w/pi,abs(h),'--')
legend('Ideal','yulewalk Designed')
title('Comparison of Frequency Response Magnitudes')
```

3. **Design a FIR filter for the same specifications and compare the characteristics of IIR filters with that of FIR filters.**

4. **Generate a signal s(n) with three sinusoidal components at 5, 15 and 30Hz and sampled at 100Hz. Design an bandpass elliptic filter to keep the 15Hz sinusoid and eliminate the 5 and 30Hz harmonics.**

Sample Solution

```
fs=100; t=(1:100)/fs;
s= sin(2*pi*5*t)+sin(2*pi*15*t)+sin(2*pi*30*t);
plot(t,s); grid;
title('Time domain waveform (5,15,30 Hz)');

%  Design filter to keep 15 Hz and remove others
wp1=10/50; wp2=20/50; ws1=5/50; ws2=25/50;
wp=[wp1,wp2]; ws=[ws1,ws2]; rp=0.1; rs=40;

[n,wn]=ellipord(wp,ws,rp,rs)
[b,a] = ellip(n,rp,rs,wn);
freqz(b,a);

[H,w] = freqz(b,a);
plot(w*fs/(2*pi),abs(H)); grid;

sf = filter(b,a,s);
subplot(211); plot(t,sf); grid;
title('Filtered signal 15Hz');
```

```
S= fft(s,512); SF=fft(sf,512);
f=(0:255)/256*(fs/2);
subplot(212); plot(f, abs([S(1:256)', SF(1:256)'])); grid
```

5. Develop a sine-cosine generator and plot the first 50 samples of its two output sequences. Scale the outputs so that they both have maximum amplitude of ±1. Take 24 samples in one cycle.

6. Execute the filterDesigner program by typing **filterDesigner** at MATLAB command prompt. Study the characteristics of the different filters.