MATLAB

Lesson 1

MATLAB BASICS

The LINSPACE operator

linspace(1,3,5) will give you output as [1 1.5 2 2.5 3]

here linspace **(x, y, m)** means x is starting value, y is maximum possible value. and m is number of points you want to display with equal spacing.

Subscripting matrices

in matrix a = [3 2 1; 6 5 4];
a(2,1) means 6

a(:,3) means all 3rd column numbers from all rows i.e.
ans = 1
4

a(:,[3 1]) will give you output,
ans = 1 3
4 6

similarly
a (1,end) is 1
a (end,1) is 6
a (end,end) is 4
now create a 10x10 matrix and check the following command
a (3,1:4:end)

Matrices and Magic squares

In the MATLAB environment, a matrix is a rectangular array of numbers. Special meaning is sometimes attached to 1-by-1 matrices, which are scalars, and to matrices with only one row or column, which are vectors. MATLAB has other ways of storing both numeric and nonnumeric data, but in the beginning, it is usually best to think of everything as a matrix. The operations in MATLAB are designed to be as natural as possible. Where other programming languages work with numbers one at a time, MATLAB allows you to work with entire matrices quickly and easily.

Entering Matrices

The best way for you to get started with MATLAB is to learn how to handle matrices. Start by entering Dürer's matrix as a list of its elements. You only have to follow a few basic conventions:

- Separate the elements of a row with blanks or commas.
- Use a semicolon';' to indicate the end of each row.
- Surround the entire list of elements with square brackets, [ ].

To enter Dürer's matrix, simply type in the Command Window

```
>>A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
```

MATLAB displays the matrix you just entered:

```
A =
    16   3   2  13
     5  10  11   8
     9   6   7  12
     4  15  14   1
```

This matrix matches the numbers in the engraving. Once you have entered the matrix, it is automatically remembered in the MATLAB workspace. You can refer to it simply as A. Now that you have A in the workspace, take a look at what makes it so interesting. Why is it magic?

Sum, Transpose and Diag

You are probably already aware that the special properties of a magic square have to do with the various ways of summing its elements. If you take the sum along any row or column, or along either of the two main diagonals, you will always get the same number. Let us verify that using MATLAB. The first statement to try is

```
>>sum(A)
```

MATLAB replies with

```
ans =
    34   34   34   34
```

You have computed a row vector containing the sums of the columns of A. Each of the columns has the same sum, the *magic* sum, 34.

How about the row sums? MATLAB has a preference for working with the columns of a matrix, so one way to get the row sums is to transpose the matrix, compute the column sums of the transpose, and then transpose the result. For an additional way that avoids the double transpose use the dimension argument for the sum function.

The magic Function

MATLAB actually has a built-in function that creates magic squares of almost any size. Not surprisingly, this function is named magic:

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

This matrix is almost the same as the one in the Dürer engraving and has all the same "magic" properties; the only difference is that the two middle columns are exchanged.
To make this B into Dürer's A, swap the two middle columns:

```
>>A = B(:,[1 3 2 4])
```

This subscript indicates that, for each of the rows of matrix B-reorder the elements in the order 1, 3, 2, 4. It produces:

```
A =
    16     3     2    13
     5    10    11     8
     9     6     7    12
     4    15    14     1
```


Expressions

Variables

Like most other programming languages, the MATLAB language provides mathematical *expressions*, but unlike most programming languages, these expressions involve entire matrices.
MATLAB does not require any type declarations or dimension statements. When MATLAB encounters a new variable name, it automatically creates the variable and allocates the appropriate amount of storage. If the variable already exists, MATLAB changes its contents and, if necessary, allocates new storage. For example,

```
num_students = 25
```

Creates a 1-by-1 matrix named num_students and stores the value 25 in its single element. To view the matrix assigned to any variable, simply enter the variable name. MATLAB is case sensitive; it distinguishes between uppercase and lowercase letters. A and a are *not* the same variable.

Numbers

MATLAB uses conventional decimal notation, with an optional decimal point and leading plus or minus sign, for numbers. Scientific notation uses the letter e to specify a power-of-ten scale factor. Imaginary numbers use either i or j as a suffix. Some examples of legal numbers are

```
3             -99                        0.0001
9.6397238     1.60210e-20 6.02252e23     1i -3.14159j 3e5i
```

All numbers are stored internally using the *long* format specified by the IEEE® floating-point standard. Floating-point numbers have a finite *precision* of roughly 16 significant decimal digits and a finite *range* of roughly 10-308 to 10+308.

Ind=find(x,k) or Ind=find(x,k,'first'): K is positive integer. This returns at most the first k indices corresponding to the non zero element.

```
>>Ind=find(x,3)
Ind=
     1 3 4
```

Ind=find(x,k,'last'): K is positive integer. This returns at most the last k indices corresponding to the non zero element.

```
>>Ind=find(x,3,'last')
Ind=
     4 8 9
```

Controlling Command Window Input and Output

The Format Function

The format function controls the numeric format of the values displayed. The function affects only how numbers are displayed, not how MATLAB software computes or saves them. Here are the different formats, together with the resulting output produced from a vector x with components of different magnitudes.


```
>>format short
>>x = [4/3 1.2345e-6]
x=
    1.3333    0.0000
```

```
>>format long
>>x = [4/3 1.2345e-6]
x=
    1.33333333333333    0.00000123450000
```

```
>>format bank
>>x = [4/3 1.2345e-6]
x=
    1.33        0.00
```

```
>>format rat
>>x = [4/3 1.2345e-6]
x=
    4/3        1/810045
```

Logical Expressions

A logical expressionis one that evaluates to either true or false.

For example, v > 0is a logical expression that will be true if the variable v is greater than zero and false otherwise.

Logical expressions can be assigned to Boolean variables. For example s = v > 0stores the value of the logical expression v > 0 in a Boolean variable s. is v is greater than 0 then s will store the value 1 else 0.

Relational operators in MATLAB

Symbol	Relation
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
= =	Equal to
~=	Not equal to

Note:- = = is relational operator in MATLAB which checks for equality where = sign assigns the value to variable. i.e. c = 5

Logical operators in MATLAB

Complex logical expressions can be created by combining simpler logical expressions using the **logical operators: AND, OR, NOT**and **XOR**(eXclusiveOR)

A	B	AND	OR	XOR	NOT	
Symbol for scalars		A && B	A || B	XOR(A, B)	~A	~B
Symbols for vectors		A & B	A | B	XOR(A, B)	~A	~B
0	0	0	0	0	1	1
0	1	0	1	1	1	0
1	0	0	1	1	0	1
1	1	1	1	0	0	0

MATLAB

Lesson 5

Random Data and Functions

MATLAB

Lesson 7

Image processing

Image processing

MATLAB works with images and any colored or black and white image is consists of arrays of large numbers. So MATLAB saves image in the form of numbers and process the data accordingly.

First check whether your MATLAB version has license for image processing toolbox using following command
license('test','image_toolbox') % 1 means yes.

now we will see how to read the image as data matrix for processing in MATLAB
A = imread('image1.jpg');

A is always three dimensional array in above case. above command will create three matrices for red green and blue color. to see the image in MATLAB which is currently in the form of array,
imshow(A);

to see the different color matrices, commands are
A(:,:1) % for red color
A(:,:2) % for green color
A(:,:3) % for blue color

the original color image is called RGB image(Red-Green-Blue). to convert it into gray,
B = rgb2gray(A); % A is matrix of some image as above
imshow(B);

Similarly to convert colored image into Black and white,
B = im2bw(A); % A is matrix of some image as above
imshow(B);

(Note:
1. For gray image number zero is for black color and 255 is for white. remaining numbers inbetween for gray
2. For black and white image only two values: 1 for white, 0 for black.
3. For color image, if a red pane consists number 255, is for red.
   i.e. (255,0,0), in green pane, 255 is for green, and in blue,255 for blue.)

example 1:
Suppose that A matrix contains the image. Write MATLAB code to colour the top border of the image in red and display the image.

A(1:100,:,1) = 255;
A(1:100,:,2) = 0;
A(1:100,:,3) = 0;
imshow(A);
example 2:

Write a Matlab script to reproduce the image on the left. a normal RGB image and also gray image(Assume matrix A contains the image)

```
B = rgb2gray(A);
imshow([fliplr(B) B]);
```

```
A1 = A(:,:,1);
A2 = A(:,:,2);
A3 = A(:,:,3);
B(:,:,1) = fliplr(A1);
B(:,:,2) = fliplr(A2);
B(:,:,3) = fliplr(A3);
imshow([A B]);
```

example 3:
try this and find out what function flipdim is

```
imshow([flipdim(A,2) A])
```

example 4:
write a code to make the full image in green

```
A(:,:,2) = 255;
imshow(A);
```

example 5:
Construct by hand an image that looks like a simple drawing of a house.

To produce such images now we will see some commands

example 2:
Make the point go up and stay there forever. The movement along the edges should be visible!
solution:

```
figure,hold on
h = plot(0,0,'r.','markersize',100);
plot([0 1 1 0 0],[0 0 1 1 0],'k-')
axis([-0.3 1.3 -0.3 1.3])
axis square
grid on
steps = linspace(0,1,100);
for i = 1:100, set(h,'YData',steps(i)),pause(0.01),end
```

example 3:
Make the point visit all 4 corners of the unit square clockwise. The movement along the edges should be visible!
solution:

```
figure,hold on
h = plot(0,0,'r.','markersize',100);
plot([0 1 1 0 0],[0 0 1 1 0],'k-')
axis([-0.3 1.3 -0.3 1.3])
axis square
grid on
steps = linspace(0,1,100);
for i = 1:100, set(h,'YData',steps(i)),pause(0.01),end
for i = 1:100, set(h,'XData',steps(i)),pause(0.01),end
for i = 1:100, set(h,'YData',steps(101-i)),pause(0.01),end
for i = 1:100, set(h,'XData',steps(101-i)),pause(0.01),end
```


now try this yourself

1. Make the point visit all 4 corners of the unit square anticlockwise. The movement along the edges should be visible!

2. make the point move upward and downward 50 times. make the whole process slower than above example without changing pause

3. Make the point move along the diagonal.

4. Make three points move simultaneously as shown in the figure.

- Heater subsystem
- Thermostat subsystem
- Thermodynamic model subsystem

The most effective way to build a model of this system is to consider each of these subsystems independently.

Identifying System Components

The second step in the modeling process is to identify the system components. Three types of components define a system:

- **Parameters** — System values that remain constant unless you change them
- **States** — Variables in the system that change over time
- **Signals** — Input and output values that change dynamically during the simulation

In Simulink, parameters and states are represented by blocks, while signals are represented by the lines that connect blocks.

For each subsystem that you identified, ask yourself the following questions:

- How many input signals does the subsystem have?
- How many output signals does the subsystem have?
- How many states (variables) does the subsystem have?
- What are the parameters (constants) in the subsystem?
- Are there any intermediate (internal) signals in the subsystem?

Once you have answered these questions, you should have a comprehensive list of the system components, and are ready to begin modeling the system.

Modeling the System with Equations

The third step in modeling a system is to formulate the mathematical equations that describe the system.

For each subsystem, use the list of system components you identified to describe the system mathematically. Your model may include:

- Algebraic equations
- Logical equations
- Differential equations, for continuous systems
- Difference equations, for discrete systems

You use these equations to create the block diagram in Simulink.