ME153 – Engineering Computation Laboratory
Class 1 - MATLAB - Introduction

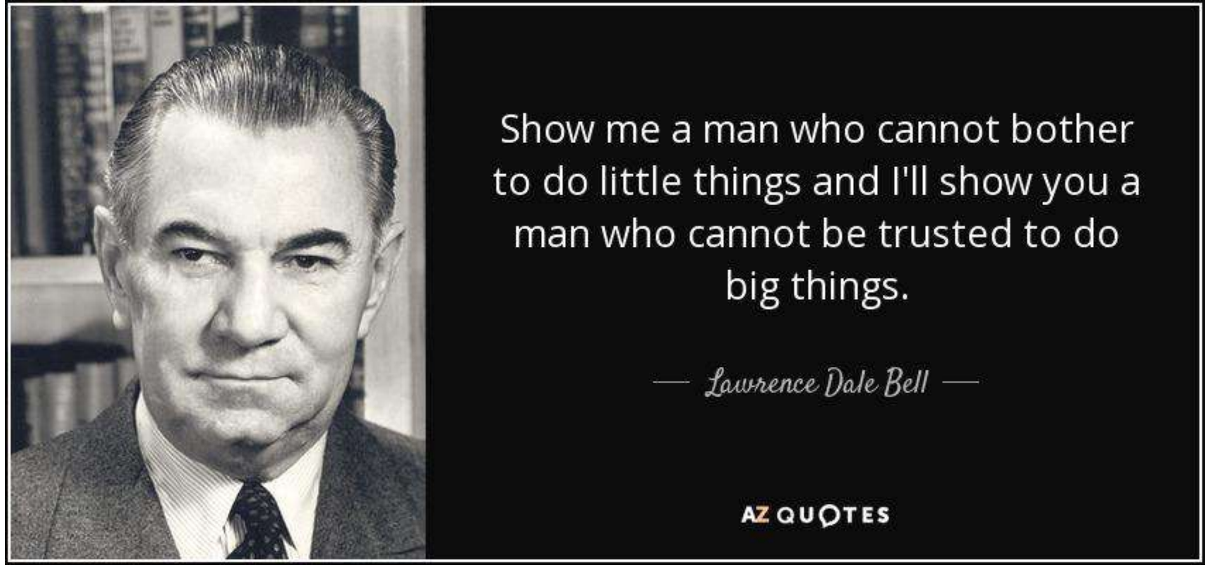
G.R.K.Gupta, MED, NITW
Mobile: 9392231833
Email: gupta@nitw.ac.in
05-04-2022, 1100 AM


HELLO
EVERYBODY


Welcome a new day with a smile on your lips
love in your heart and gratitude in your spirit
GOOD MORNING
eventstodayz.com

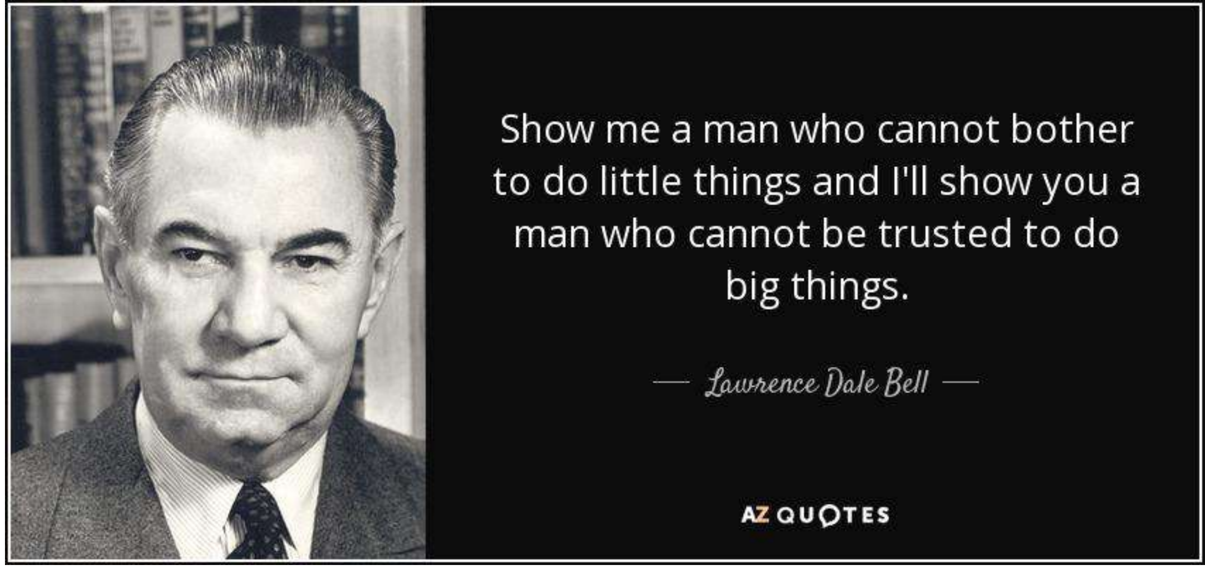
Today's Inspiring Quote

Lawrence Dale Bell (05-04-1894)



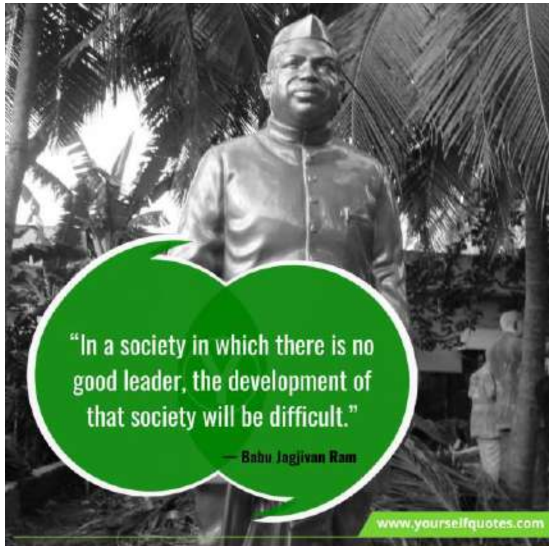
American aircraft designer and aircraft manufacturer, founder of Bell Aircraft Co., whose experimental X-1 rocket-propelled airplane in 1947 was the first to break the sound barrier in level flight. This firm also produced such significant aviation contributions as the nation's first jet propelled airplane, the world's first commercial helicopter, the world's fastest and highest flying airplane, the Bell X-1A, and the first jet vertical take-off and landing plane.

Lawrence Dale Bell (05-04-1894)



American aircraft designer and aircraft manufacturer, founder of Bell Aircraft Co., whose experimental X-1 rocket-propelled airplane in 1947 was the first to break the sound barrier in level flight. This firm also produced such significant aviation contributions as the nation's first jet propelled airplane, the world's first commercial helicopter, the world's fastest and highest flying airplane, the Bell X-1A, and the first jet vertical take-off and landing plane.

Babu Jagjivan Ram (05-04-1908)



He was known popularly as Babuji, an Indian independence activist and politician from Bihar. He was instrumental in the foundation of the All India Depressed Classes League, an organisation dedicated to attaining equality for untouchables, in 1935 and was elected to Bihar Legislative Assembly in 1937, after which he organised the rural labour movement. In 1946, he became the youngest minister in Jawaharlal Nehru's interim government, the first cabinet of India as a Labour Minister and also a member of the Constituent Assembly of India, where he ensured that social justice was enshrined in the Constitution. He went on to serve as a minister with various portfolios for more than forty years as a member of the Indian National Congress (INC). Most importantly, he was the Defence Minister of India during the Indo-Pak war of 1971, which resulted in the creation of Bangladesh. He later served as the Deputy Prime Minister of India (1977–79);

Course Outcomes

CO1 - Develop code for visualization and plotting using MATLAB

CO2 - Develop programs in MATLAB to solve problems involving loops, functions and scripts

CO3 - Develop code for solving mathematical models involving linear equations, nonlinear equations and Ordinary Differential Equations

CO4 - Apply features of MATLAB and Python programming for numerical computations in engineering

Detailed Syllabus

Matlab:

1. Introduction to MATLAB.
2. Practice session on handling basic arithmetic etc.
3. Writing codes with control loops, functions and scripts.
4. Developing codes for visualization and plotting.
5. Solving problems involving linear and nonlinear equations.
6. Solving problems involving curve fitting and interpolations.
7. Solving problems involving ordinary differential equations.

Detailed Syllabus – contd.

Python:

8. Practice programs using different types of variables and expressions.
9. Practice programs using loops and iterations.
10. Practice programs using conditional code, and functions.

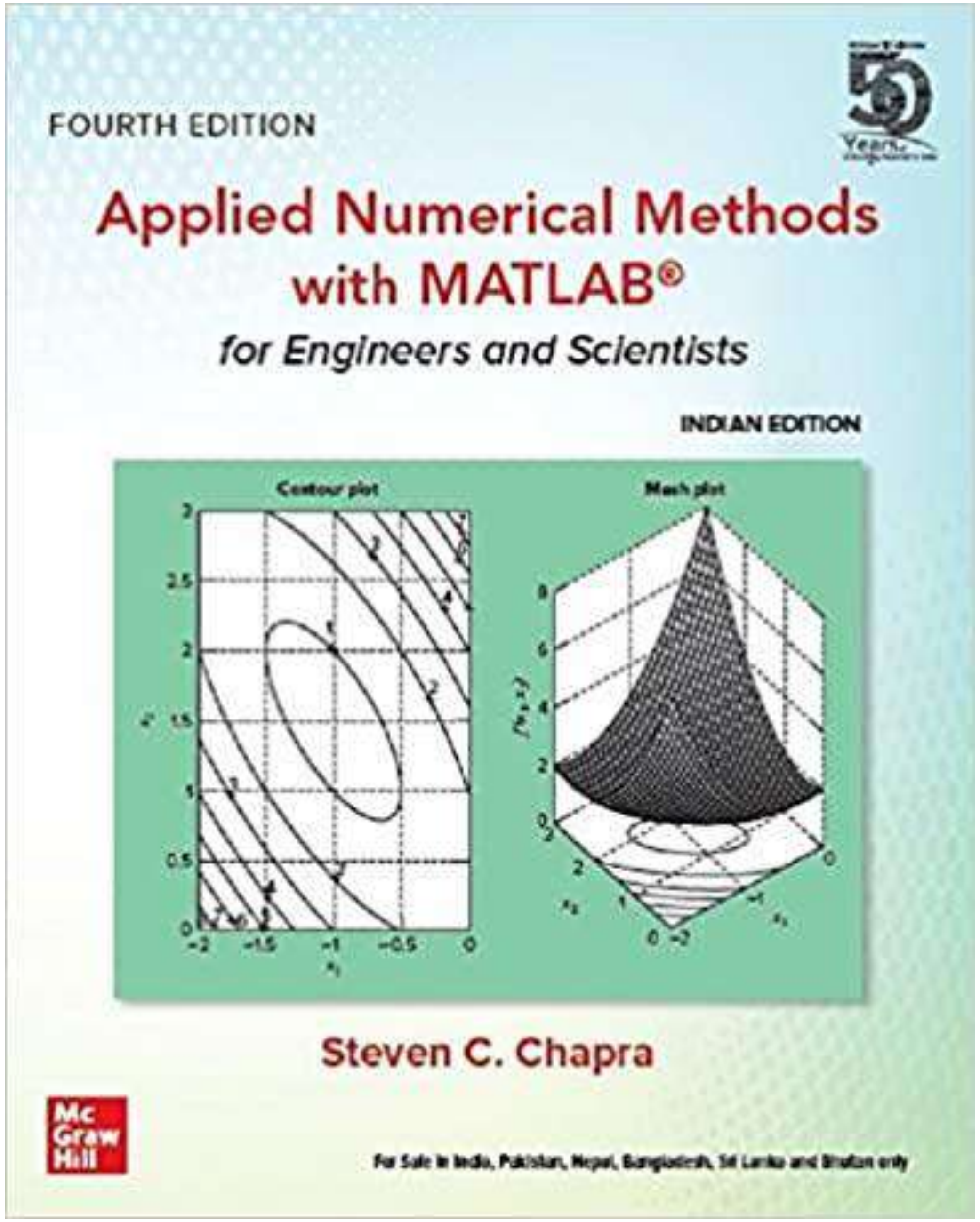
Case-Study:

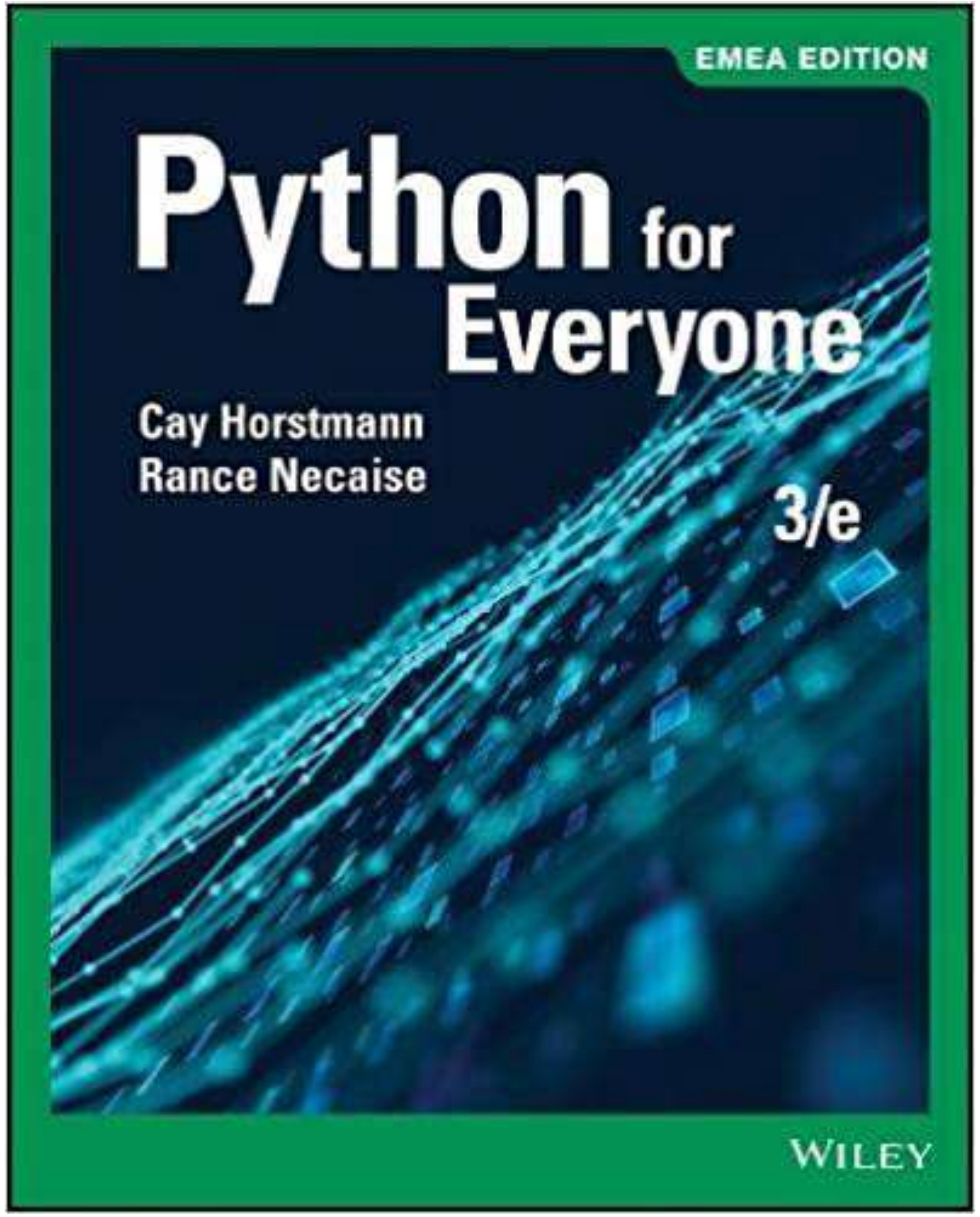
11. Case study with application of MATLAB /Python programming for topics from Engineering Mechanics.
12. Case study with application of MATLAB /Python programming for topics from Kinematics of machinery.

Detailed Syllabus – contd.

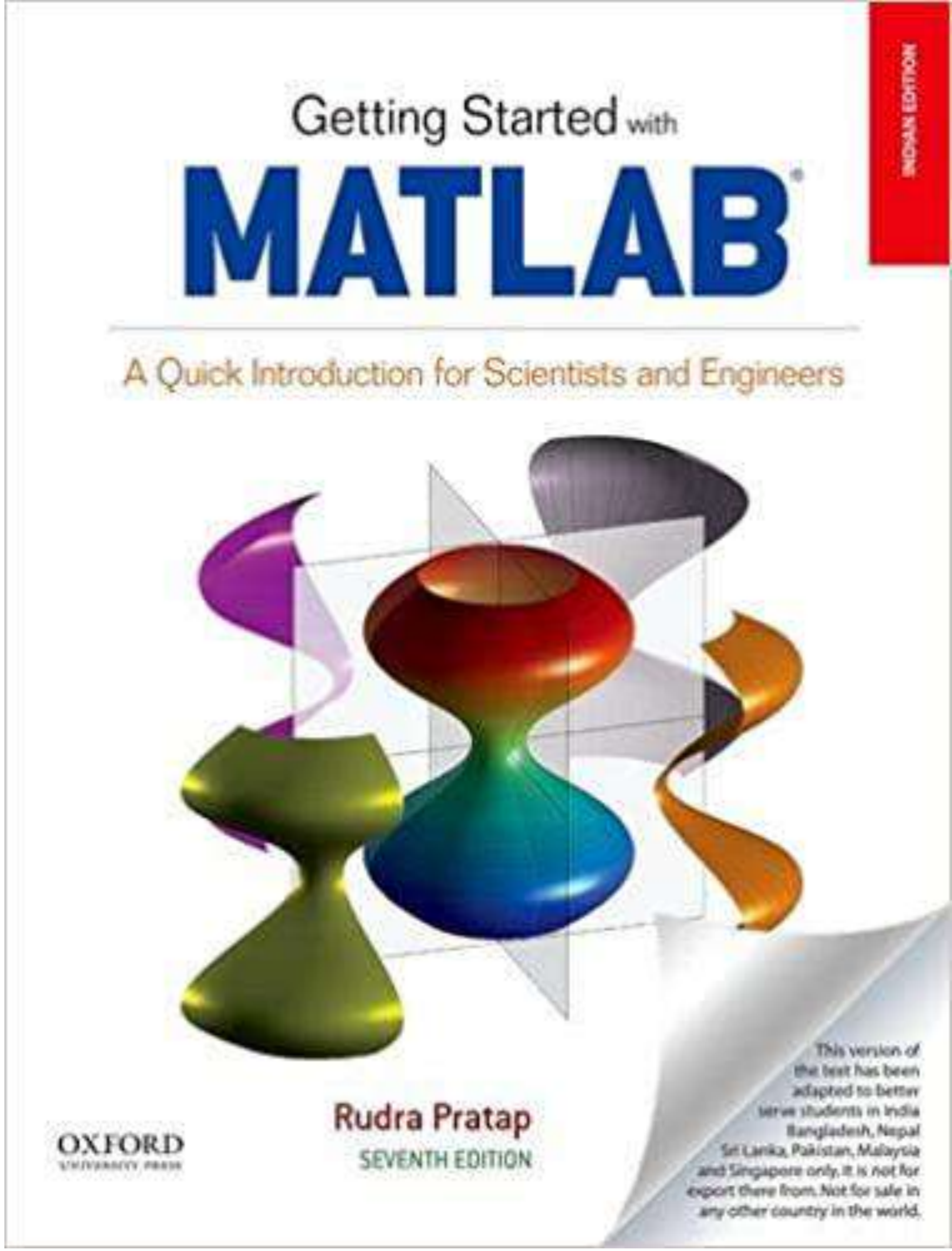
Text Books:

1. Applied Numerical Methods with MATLAB for Engineers & Scientists, Steven Chapra, McGraw-Hill, 2018, 4th edition.
2. Python for Everyone, Cay S. Horstmann, Rance D. Necaise, Wiley, 2019, 3rd edition


FOURTH EDITION
Applied Numerical Methods
with MATLAB®
for Engineers and Scientists
INDIAN EDITION
Contour plot
Mesh plot
Steven C. Chapra
Mc Graw Hill
For Sale in India, Pakistan, Nepal, Bangladesh, Sri Lanka and Bhutan only


EMEA EDITION
Python for
Everyone
Cay Horstmann
Rance Necaise
3/e
WILEY

Text Book for Today's Class

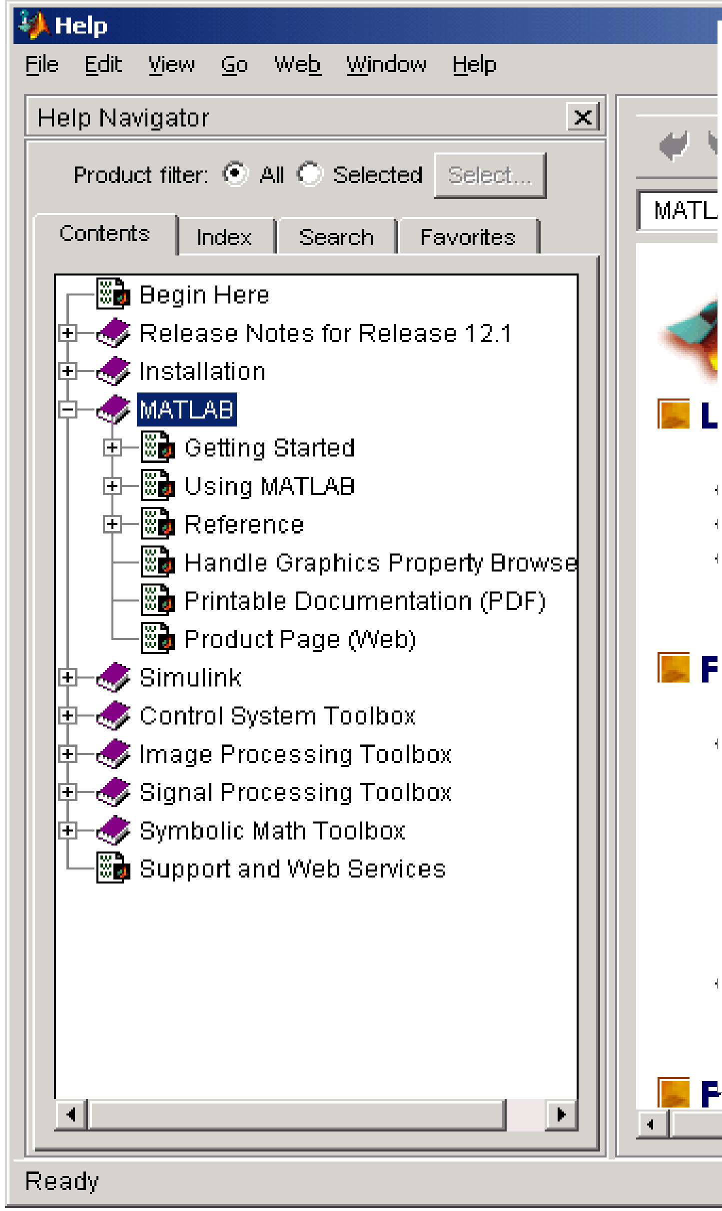


How to Install MATLAB onto your System

Nitw.ac.in - > Facilities - > Centres - > Computer Centre - > Softwares Available - > MATLAB Click on
(Matlab software, installation guide)

Then follow the procedure described there.

Getting MATLAB Help



- Type one of the following commands in the command window:
  - >>**help** – lists all the help topics
  - >>**help** *topic* – provides help for the specified topic
  - >>**help** *command* – provides help for the specified command
  - >>**helpwin** – opens a separate help window for navigation
  - >>**Lookfor** *keyword* – search all M-files for *keyword*
- Online resource

MATLAB Variables

- The MATLAB environment is command oriented somewhat like UNIX. A prompt appears on the screen and a MATLAB statement can be entered. When the <ENTER> key is pressed, the statement is executed, and another prompt appears.
- If a statement is terminated with a semicolon (;), no results will be displayed. Otherwise results will appear before the next prompt.
- Variable names ARE case sensitive.
- Variable names can contain up to 63 characters (as of MATLAB 6.5 and newer).
- Variable names must start with a letter followed by letters, digits, and underscores.
- Variable names and their types do not have to be declared in MATLAB.
- Any variable can take real, complex, and integer values.
- The name of variable is not accepted if it is reserved word.

MATLAB Variables – cont'd

- Special variables:
  - **ans**: default variable name for the result.
  - **pi**: π = 3.1415926 ……
  - **eps**: ε= 2.2204e-016, smallest value by which two numbers can differ
  - **inf**: ∞, infinity
  - **NAN** or **nan**: not-a-number
- Commands involving variables:
  - **who**: lists the names of the defined variables
  - **whos**: lists the names and sizes of defined variables
  - **clear**: clears all variables
  - **clear** *name*: clears the variable *name*
  - **clc**: clears the command window
  - **clf**: clears the current figure and the graph window
  - **Ctrl+C**: Aborts calculation

Termination

`^c` (Control-c)	local abort, kills the current command execution
`quit`	quits MATLAB
`exit`	same as `quit`

MATLAB Variables – cont’d

- *Variable is a name given to a reserved location in memory.*

```
>>x = 111;
>>number_of_students = 75;
>>name = ‘University College Cork’;
>>radius = 5;
>>area = pi * radius^2;
>>x_value=23
x_value=23
```

MATLAB Arithmetic Operators

Operator	Description
+	Addition
-	Subtraction
.*	Multiplication (element wise)
./	Right division (element wise)
.\	Left division (element wise)
=	Assignment operator,e.g. a = b,(assign b to a)
:	Colon operator (Specify Range)
.^	Power (element wise)
'	Transpose
*	Matrix multiplication
/	Matrix right division
\	Matrix left division
;	Row separator in a Matrix
^	Matrix power

Logical Operators in MATLAB

Operator	Description
&	Returns 1 for every element location that is true (nonzero) in both arrays, and 0 for all other elements.
|	Returns 1 for every element location that is true (nonzero) in either one or the other, or both, arrays and 0 for all other elements.
~	Complements each element of input array, A.
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
~=	Not equal to

Calculations at the Command Line / Workspace

MATLAB as a calculator

```
» -5/(4.8+5.32)^2
ans =
   -0.0488
» (3+4i)*(3-4i)
ans =
    25
» cos(pi/2)
ans =
  6.1230e-017
» exp(acos(0.3))
ans =
    3.5470
```

Assigning Variables

```
» a = 2;
» b = 5;
» a^b
ans =
    32
» x = 5/2*pi;
» y = sin(x)
y =
     1
» z = asin(y)
z =
    1.5708
```

Semicolon suppresses screen output

Results assigned to "ans" if name not specified

() parentheses for function inputs

Arithmetic Operations on scalars

```
>> 2 + 2

ans =

     4
```

Enter `2+2` and hit the return/enter key. Note that the result of an unassigned expression is saved in the default variable *ans*.

```
>> x = 2 + 2

x =

     4
```

You can also assign the value of an expression to a variable.

```
>> y = 2^2 + log(pi)*sin(x);
>> y

y =

   3.1337
```

A semicolon at the end suppresses screen output. MATLAB remembers *y*, though. You can recall the value *y* by simply typing *y*.

Output format

Though computations inside MATLAB are performed using double precision, the appearance of floating point numbers on the screen is controlled by the output format in use. There are several different screen output formats. The following table shows the printed value of l01r in seven different formats.

format short	31.4159
format short e	3.1416e+001
format long	31.41592653589793
format long e	3.141592653589793e+001
format short g	31.416
format long g	31.4159265358979
format hex	403f6a7a2955385e
format rat	3550/113
format bank	31.42

```
>> theta = acos(-1)

theta =

    3.1416
```

MATLAB knows trigonometry. Here is arccosine of -1.

```
>> format short e
>> theta

theta =

   3.1416e+000

>> format long
>> theta

theta =

   3.141592653589793
```

The floating point output display is controlled by the `format` command. Here are two examples. More information will be provided on this later.

Quit MATLAB. You can also quit by

Arrays (Creation and Operations)

```
>> x = [1 2 3]

x =
     1     2     3
```

x is a row vector with three elements.

```
>> y = [2; 1; 5]

y =
     2
     1
     5
```

y is a column vector with three elements.

```
>> z = [2 1 0];
>> a = x + z

a =
     3     3     3
```

You can add (or subtract) two vectors of the same size.

Creating Vectors

```
v = InitialValue:Increment:FinalValue
```

Examples:

`a = 0:10:100` produces $a = [\ 0 \quad 10 \quad 20 \quad \ldots \quad 100\ ]$,

`b = 0:pi/50:2*pi` produces $b = [\ 0 \quad \frac{\pi}{50} \quad \frac{2\pi}{50} \quad \ldots \quad 2\pi\ ]$, i.e., a linearly spaced vector from 0 to 2π spaced at $\pi/50$,

`u = 2:10` produces $a = [\ 2 \quad 3 \quad 4 \quad \ldots \quad 10\ ]$.

Built-in Functions to Generate Vectors

`linspace(a,b,n)` generates a linearly spaced vector of length n from a to b.

Example: `u=linspace(0,20,5)` generates `u=[0 5 10 15 20]`.

Thus, `u=linspace(a,b,n)` is the same as `u=a:(b-a)/(n-1):b`.

`logspace(a,b,n)` generates a logarithmically spaced vector of length n from 10^a to 10^b.

Example: `v=logspace(0,3,4)` generates `v=[1 10 100 1000]`.

Thus, `logspace(a,b,n)` is the same as `10.^(linspace(a,b,n))`. (The array

Working with Matrices

```
>> A=[1 2 3; 4 5 6; 7 8 8]

A =

     1     2     3
     4     5     6
     7     8     8
```

Matrices are entered row-wise. Rows are separated by semicolons and columns are separated by spaces or commas.

```
>> A(2,3)

ans =

     6
```

Element A_{ij} of matrix A is accessed as `A(i,j)`.

```
>> A(3,3) = 9

A =

     1     2     3
     4     5     6
     7     8     9
```

Correcting any entry is easy through indexing.

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

B =

     4     5     6
     7     8     9
```

Any submatrix of A is obtained by using range specifiers for row and column indices.

```
>> B = A(2:3,:)

B =

     4     5     6
     7     8     9
```

The colon by itself as a row or column index specifies all rows or columns of the matrix.

```
>> B(:,2)=[]

B =

     4     6
     7     9
```

A row or a column of a matrix is deleted by setting it to a null vector `[]`.

```
>> A = [1 2 3; 4 5 6; 7 8 9];

>> x = A(1,:)'

x =

     1
     2
     3
```

Matrices are transposed using the single right-quote character ('). Here x is the transpose of the first row of A.

```
>> x'*x

ans =

    14
```

Matrix or vector products are well-defined between compatible pairs. A row vector (x') times a column vector (x) of the same length gives the inner product, which is a scalar, but a column vector times a row vector of the same length gives the outer product, which is a matrix.

```
>> x*x'

ans =

     1     2     3
     2     4     6
     3     6     9
```

```
>> A*x

ans =

    14
    32
    50
```

Look how easy it is to multiply a vector with a matrix, compared with Fortran or Pascal.

```
>> A^2

ans =

    30    36    42
    66    81    96
   102   126   150
```

You can even exponentiate a matrix if it is a square matrix. A^2 is simply A*A.

```
>> A.^2

ans =

     1     4     9
    16    25    36
    49    64    81
```

When a dot precedes the arithmetic operators `*`, `^`, and `/`, MATLAB performs array operation (element-by-element operation). So, `A.^2` produces a matrix with elements $(a_{ij})^2$.

Arithmetic Operations

$+$	addition
$-$	subtraction
$*$	multiplication
$/$	division
^ (caret)	exponentiation

`A+B` or `A-B`	is valid if A and B are of the same size,
`A*B`	is valid if A's number of columns equals B's number of rows,
`A/B`	is valid and equals $A \cdot B^{-1}$ for same-size square matrices, and
`A^2`	which equals `A*A`, makes sense only if A is square.

Array Operations

.*	element-by-element multiplication
./	element-by-element left division
.\	element-by-element right division
.^	element-by-element exponentiation
.'	nonconjugated transpose

Examples: `u.*v` produces $[u_1v_1 \;\; u_2v_2 \; u_3v_3 \;\; \ldots]$,

`u./v` produces $[u_1/v_1 \;\; u_2/v_2 \; u_3/v_3 \;\; \ldots]$, and

`u.^v` produces $[u_1^{v_1},\; u_2^{v_2},\; u_3^{v_3},\; \ldots]$.

```
>> b = x + y
```

But you cannot add (or subtract) a row vector to a column vector.

```
??? Error using ==> plus
Matrix dimensions must agree.
```

```
>> a = x.*z

a =
     2     2     0
```

You can multiply (or divide) the elements of two same-sized vectors term by term with the *array operator* `.*` (or `./`) .

```
>> b = 2*a

b =
     4     4     0
```

But multiplying a vector with a scalar does not need any special operation (no dot before the `*`).

```
>> x = linspace(0,10,5)

x =
         0    2.5000    5.0000    7.5000   10.0000
```

Create a vector x with 5 elements *lin*early *space*d between 0 and 10.

Built-in Functions

```
>> y = sin(x);

>> z = sqrt(x).*y

z =

         0    0.9463   -2.1442    2.5688   -1.7203
```

Trigonometric functions `sin`, `cos`, etc., as well as elementary math functions `sqrt`, `exp`, `log`, etc., operate on vectors term by term.

Mathematical Functions of MATLAB-1

Elemantary Mathematical (Trigonometric) Functions	
Trigonometric functions	Remarks
sin(x) cos(x) tan(x) asin(x) acos(x) atan(x) atan2(y,x) sinh(x) cosh(x) tanh(x) asinh(x) acosh(x) atanh(x)	- pi/2 ≤ atan(x) ≥ pi/2, Same as atan(y/x) but –pi ≥ atan(y,x) ≥ pi

Mathematical Functions of MATLAB-2

Other elemantary functions	Remarks
abs(x)	Absolute value of x
angle(x)	Phase angle of complex value: If x = real, angle = 0. If $x = \sqrt{-1}$, angle = pi/2
sqrt(x)	Square root of x
real(x)	Real part of complex value x
imag(x)	Imaginary part of complex value x
conj(x)	Complex conjugate x
round(x)	Round to do nearest integer
fix(x)	Round a real value toward zero
floor(x)	Round x toward $- \infty$
ceil(x)	Round x toward $+ \infty$
sign(x)	+1 if $x > 0$; -1 if $x < 0$
exp(x)	Exponential base e
log(x)	Log base e
log10(x)	Log base 10
factor(x)	1 if x is a prime numb

And there are many many more !

Plotting 2-D Graphs

`>> theta = linspace(0,2*pi,100);`	Create a linearly spaced 100-elements-long vector θ.
`>> x = cos(theta);`	Calculate x- and y-coordinates.
`>> y = sin(theta);`	
`>> plot(x,y)`	Plot x vs. y (see Section 6.1).
`>> axis('equal');`	Set the length scales of the two axes to be the same.
`>> xlabel('x')`	Label the x-axis with `x`.
`>> ylabel('y')`	Label the y-axis with `y`.
`>> title('Circle of unit radius')`	Put a title on the plot.
`>> print`	Print on the default printer.

Creating, Saving, and Executing a Script File

```
% CIRCLE - A script file to draw a unit circle
% File written by Rudra Pratap. Last modified 5/28/98
% ----------------------------
theta=linspace(0,2*pi,100);      % create vector theta
x=cos(theta);                    % generate x-coordinates
y=sin(theta);                    % generate y-coordinates
plot(x,y);                       % plot the circle
axis('equal');                   % set equal scale on axes
title('Circle of unit radius')   % put a title
```

Creating and Executing a Function File

```
function [x,y] = circlefn(r);
% CIRCLEFN - Function to draw a circle of radius r.
% File written by Rudra Pratap on 9/17/94. Last modified 7/1/98
% Call syntax:    [x,y] = circlefn(r);  or just:  circlefn(r);
% Input:         r = specified radius
% Output: [x,y] = the x- and y-coordinates of data points
theta=linspace(0,2*pi,100);     % create vector theta
x = r*cos(theta);               % generate x-coordinates
y = r*sin(theta);               % generate y-coordinates
plot(x,y);                      % plot the circle
axis('equal');                  % set equal scale on axes
title(['Circle of radius r =',num2str(r)])
                                % put a title with the value of r
```

```
>> R = 5;
>> [x,y] = circlefn(R);
```

Specify the input and execute the function with an explicit output list.

```
>> [cx,cy] = circlefn(2.5);
```

You can also specify the value of the input directly.

```
>> circlefn(1);
```

If you don't need the output, you don't have to specify it.

```
>> circlefn(R^2/(R+5*sin(R)));
```

Of course, the input can also be a valid MATLAB expression.

Creating and Printing Simple Plots

`plot`	creates a 2-D line plot,
`axis`	changes the aspect ratio of the x-axis and the y-axis,
`xlabel`	annotates the x-axis,
`ylabel`	annotates the y-axis,
`title`	puts a title on the plot, and
`print`	prints a hard copy of the plot.

Interactive Computation

```
>> A = rand(4,3)

A =
     0.8147     0.6324     0.9575
     0.9058     0.0975     0.9649
     0.1270     0.2785     0.1576
     0.9134     0.5469     0.9706
```

Create a 4 x 3 random matrix *A*.

```
>> A(3:4, 2:3)

ans =
     0.2785     0.1576
     0.5469     0.9706
```

Get those elements of *A* that are located in rows 3 to 4 and columns 2 to 3.

```
>> A(:,4) = A(:,1)

A =
     0.8147     0.6324     0.9575     0.8147
     0.9058     0.0975     0.9649     0.9058
     0.1270     0.2785     0.1576     0.1270
     0.9134     0.5469     0.9706     0.9134
```

Add a fourth column to *A* and set it equal to the first column of *A*.

```
>> A(2:4,2:4) = eye(3)

A =
    0.8147    0.6324    0.9575    0.8147
    0.9058    1.0000         0         0
    0.1270         0    1.0000         0
    0.9134         0         0    1.0000
```

Replace the last 3 x 3 submatrix of A (rows 2 to 4, columns 2 to 4) by a 3 x 3 identity matrix.

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

A =
    0.9058    1.0000         0         0
    0.9134         0         0    1.0000
```

Delete the first and third rows of A.

```
>> A = round(A)

A =
     1     1     0     0
     1     0     0     1
```

Round off all entries of A.

```
>> A(:)'

ans =
     5     1     1     0     0     1
```

String out all elements of A in a row (note the transpose at the end).

Utility Matrices

`eye(m,n)`	returns an m by n matrix with ones on the main diagonal,
`zeros(m,n)`	returns an m by n matrix of zeros,
`ones(m,n)`	returns an m by n matrix of ones,
`rand(m,n)`	returns an m by n matrix of random numbers,
`randn(m,n)`	returns an m by n matrix of normally distributed numbers,
`diag(v)`	generates a diagonal matrix with vector v on the diagonal,
`diag(A)`	extracts the diagonal of matrix A as a vector, and
`diag(A,1)`	extracts the first upper off-diagonal vector of matrix A.

```
>> eye(3)

ans =

     1     0     0
     0     1     0
     0     0     1
```

`eye(n)` creates an n x n identity matrix. The commands `zeros`, `ones`, and `rand` work in a similar way.

```
>> B = [ones(3) zeros(3,2); zeros(2,3) 4*eye(2)]

B =

     1     1     1     0     0
     1     1     1     0     0
     1     1     1     0     0
     0     0     0     4     0
     0     0     0     0     4
```

Create a matrix B using submatrices made up of elementary matrices: ones, zeros, and the identity matrix of the specified sizes.

```
>> diag(B)'

ans =

     1     1     1     4     4
```

This command pulls out the diagonal of B in a row vector. Without the transpose, the result would obviously be a column vector.

```
>> diag(B,1)'

ans =

     1     1     0     0
```

The second argument of the command specifies the off-diagonal vector to be pulled out. Here we get the first upper off-diagonal vector. A negative value of the argument specifies the lower off-diagonal vectors.

```
>> d = [2 4 6 8];
>> d1 = [-3 -3 -3];
>> d2 = [-1 -1];
```

Create vectors d, $d1$, and $d2$ of length 4, 3, and 2, respectively.

```
>> D = diag(d) + diag(d1,1) + diag(d2,-2)

D =

     2    -3     0     0
     0     4    -3     0
    -1     0     6    -3
     0    -1     0     8
```

Create a matrix D by putting d on the main diagonal, $d1$ on the first upper diagonal, and $d2$ on the second lower diagonal.

`rot90`	rotates a matrix by 90^o,
`fliplr`	flips a matrix from left to right,
`flipud`	flips a matrix from up to down,
`tril`	extracts the lower triangular part of a matrix,
`triu`	extracts the upper triangular part of a matrix, and
`reshape`	changes the shape of a matrix.

Relational Operations

$<$	less than
$<=$	less than or equal
$>$	greater than
$>=$	greater than or equal
$==$	equal
~=	not equal

Examples: If $x = [1 \ \ 5 \ \ 3 \ \ 7]$ and $y = [0 \ \ 2 \ \ 8 \ \ 7]$, then

`k = x < y`	results in $k = [0\ 0\ 1\ 0]$	because $x_i < y_i$ for $i = 3$,
`k = x <= y`	results in $k = [0\ 0\ 1\ 1]$	because $x_i \leq y_i$ for $i = 3$ and 4,
`k = x > y`	results in $k = [1\ 1\ 0\ 0]$	because $x_i > y_i$ for $i = 1$ and 2,
`k = x >= y`	results in $k = [1\ 1\ 0\ 1]$	because $x_i \geq y_i$ for $i = 1, 2,$ and 4
`k = x == y`	results in $k = [0\ 0\ 0\ 1]$	because $x_i = y_i$ for $i = 4$, and
`k = x ~= y`	results in $k = [1\ 1\ 1\ 0]$	because $x_i \neq y_i$ for $i = 1, 2,$ and 3

Logical Operations

`&`	logical AND
`|`	logical OR
`~`	logical complement (NOT)
`xor`	exclusive OR

Examples: For two vectors $x = [0\ \ 5\ \ 3\ \ 7]$ and $y = [0\ \ 2\ \ 8\ \ 7]$,

`m = (x>y)&(x>4)`	results in $m = [0\ 1\ 0\ 0]$, because the condition is true only for x_2,
`n = x|y`	results in $n = [0\ 1\ 1\ 1]$, because either x_i or y_i is nonzero for $i = [2\ 3\ 4]$,
`m = ~(x|y)`	results in $m = [1\ 0\ 0\ 0]$, which is the logical complement of `x|y`, and
`p = xor(x,y)`	results in $p = [0\ 0\ 0\ 0]$, because there is no such index i for which x_i or y_i, but not both, is nonzero.

Elementary Math Functions

Trigonometric functions

`sin`, `sind`	sine,	`sinh`	hyperbolic sine,
`asin`, `asind`	inverse sine,	`asinh`	inverse hyperbolic sine,
`cos`, `cosd`	cosine,	`cosh`	cyperbolic cosine,
`acos`, `acosd`	inverse cosine,	`acosh`	inverse hyperbolic cosine,
`tan`, `tand`	tangent,	`tanh`	hyperbolic tangent,
`atan`, `atand`	inverse tangent,	`atanh`	inverse hyperbolic tangent,
`atan2`	four-quadrant $\tan^{-1}$,		
`sec`, `secd`	secant,	`sech`	hyperbolic secant,
`asec`, `asecd`	inverse secant,	`asech`	inverse hyperbolic secant,
`csc`, `cscd`	cosecant,	`csch`	hyperbolic cosecant,
`acsc`, `acscd`	inverse cosecant,	`acsch`	inverse hyperbolic cosecant,
`cot`, `cotd`	cotangent,	`coth`	hyperbolic cotangent,
`acot`, `acotd`	inverse cotangent, and	`acoth`	inverse hyperbolic cotangent.

The angles given to these functions as arguments must be in *radians* for `sin`, `cos`, etc., and in *degrees* for `sind`, `cosd`, etc. Thus, `sin(pi/2)` and `sind(90)` produce the same result. All of these functions, except `atan2`, take a single scalar, vector, or matrix as input argument. The function `atan2` takes two input arguments, `atan2(y,x)`, and produces the four-quadrant inverse tangent such that $-\pi \leq \tan^{-1}\frac{y}{x} \leq \pi$. This gives the angle a rectangular to polar conversion.

Examples: If `q=[0 pi/2 pi]`, `x=[1 -1 -1 1]`, and `y=[1 1 -1 -1]`, then

`sin(q)`	gives `[0 1 0]`,
`sinh(q)`	gives `[0 2.3013 11.5487]`,
`atan(y./x)`	gives `[0.7854 -0.7854 0.7854 -0.7854]`, and
`atan2(y,x)`	gives `[0.7854 2.3562 -2.3562 -0.7854]`.

Exponential Functions

`exp`	exponential, *Example:* `exp(A)` produces a matrix with elements $e^{(A_{ij})}$. So how do you compute e^A? See the next section.
`log`	natural logarithm, *Example:* `log(A)` produces a matrix with elements $\ln(A_{ij})$.
`log10`	base 10 logarithm, *Example:* `log10(A)` produces a matrix with elements $\log_{10}(A_{ij})$.
`sqrt`	square root, *Example:* `sqrt(A)` produces a matrix with elements $\sqrt{A_{ij}}$. But what about $\sqrt{A}$? See the next section.
`nthroot`	real *n*th root of real numbers, *Example:* `nthroot(A,3)` produces a matrix with elements $\sqrt[3]{A_{ij}}$.

In addition, `log2`, `pow2`, `nextpow2`, `realpow`, `reallog`, `realsqrt`, `log1p` (for $\log(1+x)$), and `exp1m` (for $e^x - 1$) functions exist in MATLAB. Clearly, these are array operations. You can, however, also compute matrix exponential e^A, matrix square root $\sqrt{A}$, etc. See Section 3.2.5.

Round-off Functions

`fix`	round toward 0, *Example:* `fix([-2.33 2.66])` $= [-2 \quad 2]$.
`floor`	round toward $-\infty$, *Example:* `floor([-2.33 2.66])` $= [-3 \quad 2]$.
`ceil`	round toward $+\infty$, *Example:* `ceil([-2.33 2.66])` $= [-2 \quad 3]$.
`mod`	modulus after division; `mod(a,b)` is the same as `a-floor(a./b)*b`, *Example:* `mod(26,5)` $= 1$ and `mod(-26,5)` $= 4$.
`round`	round toward the nearest integer, *Example:* `round([-2.33 2.66])` $= [-2 \quad 3]$.
`rem`	remainder after division, `rem(a,b)` is the same as `a-fix(a./b)*b`, *Example:* If `a=[-1.5 7]`, `b=[2 3]`, then `rem(a,b)` $= [-1.5 \; 1]$.
`sign`	signum function, *Example:* `sign([-2.33 2.66])` $= [-1 \quad 1]$.

Matrix Functions

`expm(A)`	finds the exponential of matrix A, e^A,
`logm(A)`	finds $\log(A)$ such that $A = e^{\log(A)}$, and
`sqrtm(A)`	finds $\sqrt{A}$.

```
>> A = [1 2; 3 4]

A =

     1     2
     3     4

>> asqrt = sqrt(A)

asqrt =

    1.0000    1.4142
    1.7321    2.0000

>> Asqrt = sqrtm(A)

Asqrt =

   0.5537 + 0.4644i   0.8070 - 0.2124i
   1.2104 - 0.3186i   1.7641 + 0.1458i
```

`sqrt` is an array operation. It gives the square root of each element of A as is evident from the output here.

`sqrtm`, on the other hand, is a true matrix function, i.e., it computes $\sqrt{A}$. Thus `[Asqrt]*[Asqrt]` $=[A]$.

```
>> exp_aij = exp(A)

exp_aij =

    2.7183    7.3891
   20.0855   54.5982

>> exp_A = expm(A)

exp_A =

   51.9690   74.7366
  112.1048  164.0738
```

Similarly, `exp` gives an element-by-element exponential of the matrix, whereas `expm` finds the true matrix exponential e^A. For information on other matrix functions, type `help matfun`.

Linear Algebra

Solving a Linear System

$$
\begin{aligned}
5x - 3y + 2z &= 10 \\
-3x + 8y + 4z &= 20 \\
2x + 4y - 9z &= 9
\end{aligned}
$$

```
>> A = [5 -3 2; -3 8 4; 2 4 -9];    % Enter matrix A
>> b = [10; 20; 9];                 % Enter column vector b
>> x = A\b                          % Solve for x
x =
    3.4442
    3.1982
    1.1868

>> c = A*x                         % check the solution
c =
    10.0000
    20.0000
     9.0000
```

The backslash (\) or the left division is used to solve a linear system of equations $[\mathbf{A}]\{\mathbf{x}\} = \{\mathbf{b}\}$. For more information, type: `help slash`.

Eigen Values and Eigen Vectors

```
>> A = [5 -3  2; -3  8  4; 4  2  -9];
>> [V,D]  =  eig(A)

V =

    0.1725     0.8706    -0.5375
    0.2382     0.3774     0.8429
   -0.9558     0.3156    -0.0247


D =

  -10.2206          0          0
         0     4.4246          0
         0          0     9.7960
```

Here **V** is a matrix containing the eigenvectors of **A** as its columns. For example, the first column of **V** is the first eigenvector of **A**.

D is a matrix that contains the eigenvalues of **A** on its diagonal.

Today’s Inspiring Story

Story of a Soap Box

One of the most memorable case studies on Japanese management was the case of the empty soap box, which happened in one of Japan's biggest cosmetics companies. The company received a complaint that a consumer had bought a soap box that was empty.

Immediately the authorities isolated the problem to the assembly line, which transported all the packaged boxes of soap to the delivery department. For some reason, one soap box went through the assembly line empty.

Management asked its engineers to solve the problem. Post-haste, the engineers worked hard to devise an X-ray machine with high- resolution monitors manned by two people to watch all the soap boxes that passed through the line to make sure they were not empty.

No doubt, they worked hard and they worked fast but they spent whoopee amount to do so. But when a workman was posed with the same problem, did not get into complications of X-rays, etc but instead came out with another solution.

He bought a strong industrial electric fan and pointed it at the assembly line. He switched the fan on, and as each soap box passed the fan, it simply blew the empty boxes out of the line.

Moral of the story: Always look for simple solutions. Devise the simplest possible solution that solves the problem. So, learn to focus on solutions not on problems. “**If you look at what you do not have in life, you don't have anything; if you look at what you have in life, you have everything.**”


Thank You
grammarly