Lecture 1
Introduction to MATLAB

Dr. Mahesha Narayana

Intended Learning Outcomes

At the end of this lecture, students will be able to:

- Understand Matlab desktop environment
- Understand basic syntaxes in Matlab
- Do calculations at the command window
- Define & manipulate variables and characters
- Use built-in functions and define new functions
- Use script files
- Plot simple graphs

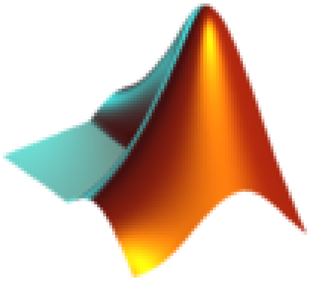
Topics

- MATLAB Features
- MATLAB Environment
- Basic syntax
- Basic operators
- MATLAB In-built and user-defined functions
- Programming and Script
- Basic: Matrix
- Plotting of functions

MATLAB

- MATLAB – MAT-rix LAB-orartory
- High-level language and interactive environment for numerical computation, visualization, and programming
- Developed by: Cleve Moler (Univ. of New Mexico, 1970)
- Proprietary commercial software developed by MathWorks (USA)
- Initial release: 1984
- Present stable release: 2015, R2015a
- Operating system: Cross-platform
  - Linux
  - Microsoft Windows
  - Mac OS X

MATLAB LOGO

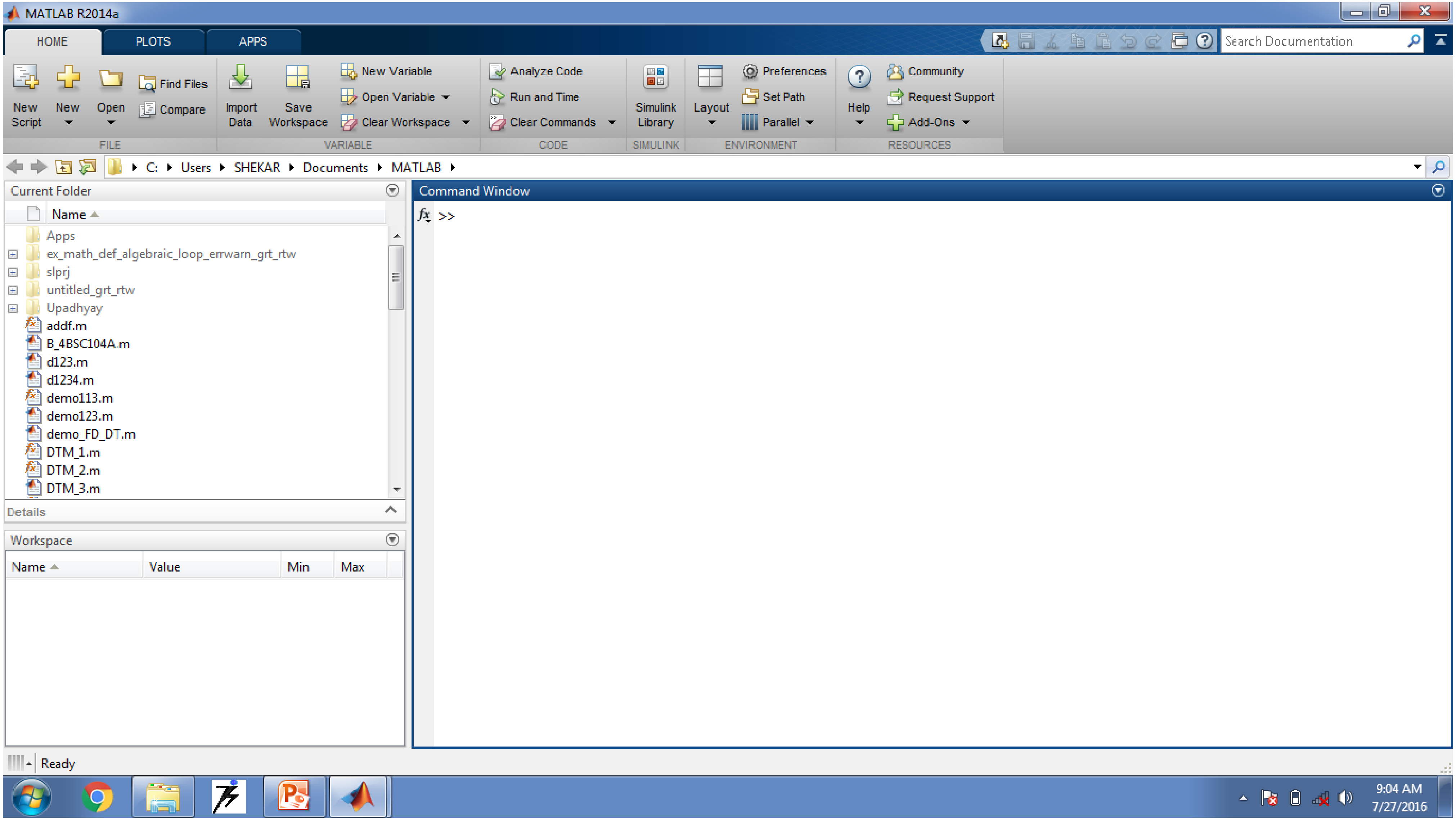


The logo is an eigenfunction of the wave equation and represents the first vibrational mode of a thin L-shaped membrane, clamped at the edges.

MATLAB Features

- Matrices and Arrays
- Calculus and Differential Equations
- Numerical Calculations
- Differentiation and Integration
- 2-D and 3-D Plotting and graphics
- Linear Algebra
- Algebraic Equations
- Non-linear Functions
- Statistics
- Data Analysis
- Develop algorithms, and create models and applications
- Signal Processing
- Image and Video Processing
- Interfacing with programs written in other languages, including C, C++, Java, Fortran and Python
- Many more ….

Understanding the MATLAB Environment



C: ▸ Users ▸ SHEKAR ▸ Documents ▸ MATLAB ▸

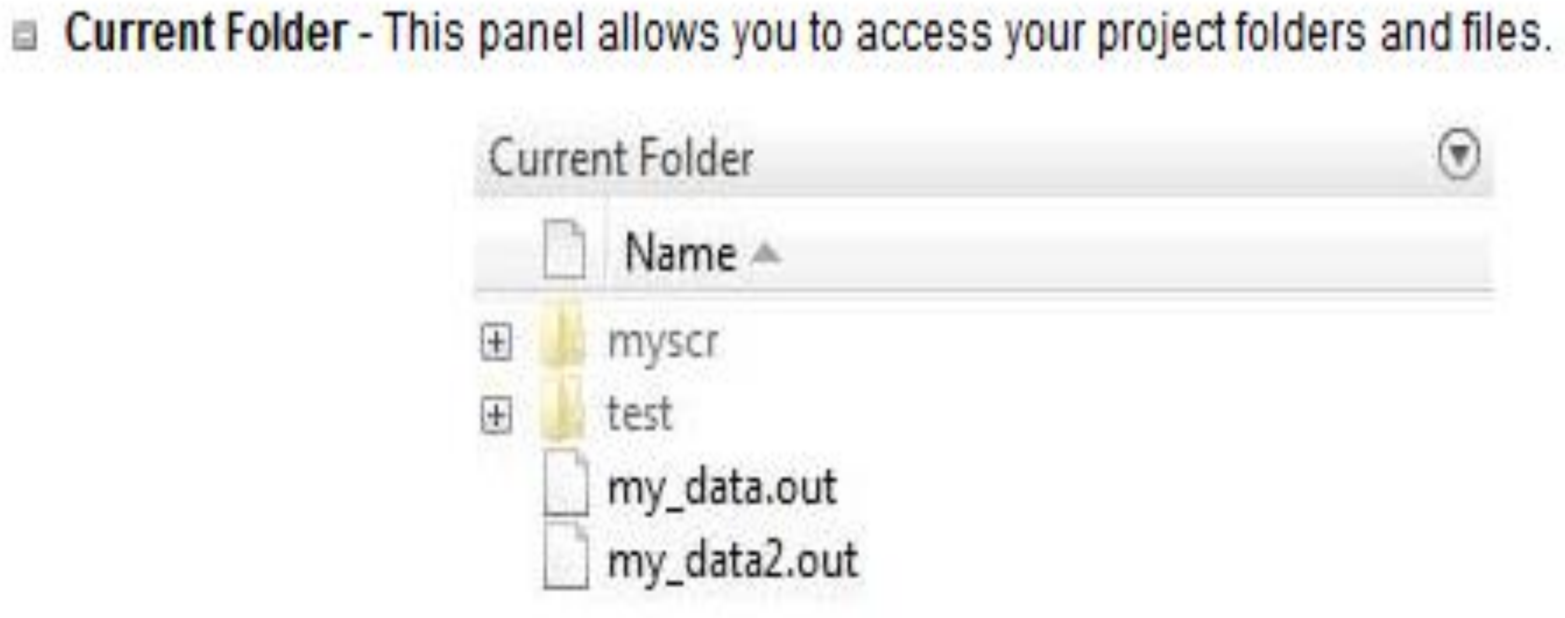
Apps
ex_math_def_algebraic_loop_errwarn_grt_rtw
slprj
untitled_grt_rtw
Upadhyay
addf.m
B_4BSC104A.m
d123.m
d1234.m
demo113.m
demo123.m
demo_FD_DT.m
DTM_1.m
DTM_2.m
DTM_3.m

Name	Value	Min	Max

```
fx >>
```

Ready

Understanding the MATLAB Environment



- **Current Folder** - This panel allows you to access your project folders and files.

Current Folder

Name

myscr

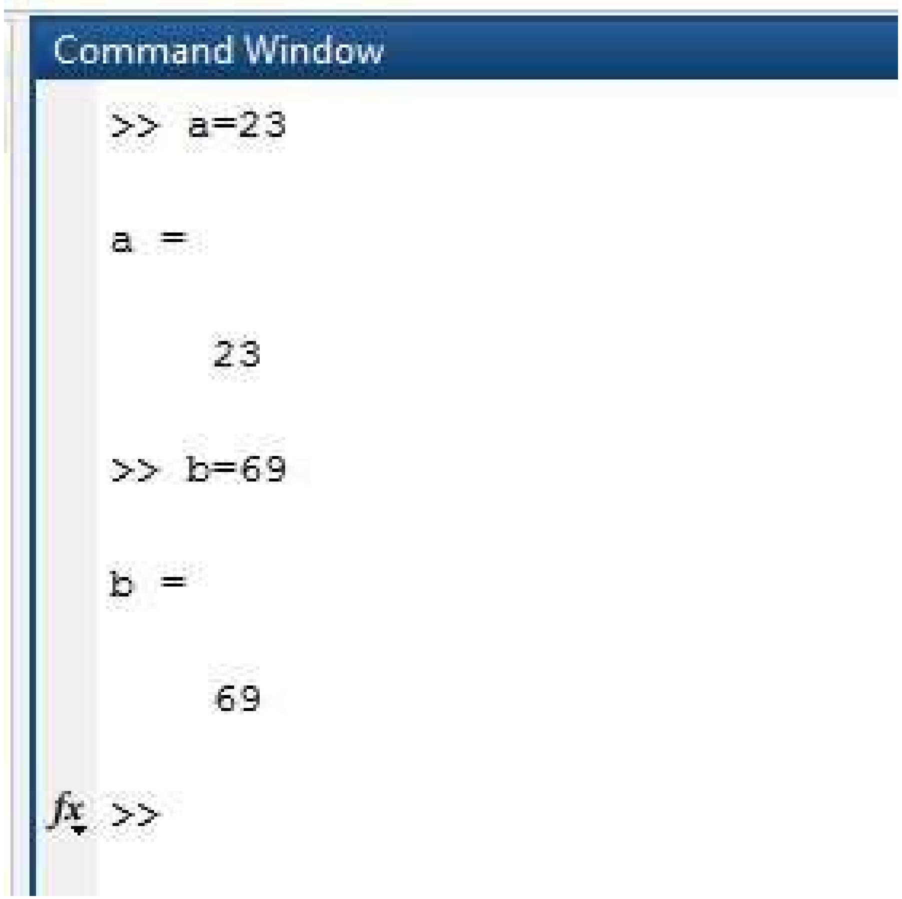
test

my_data.out

my_data2.out

Understanding the MATLAB Environment

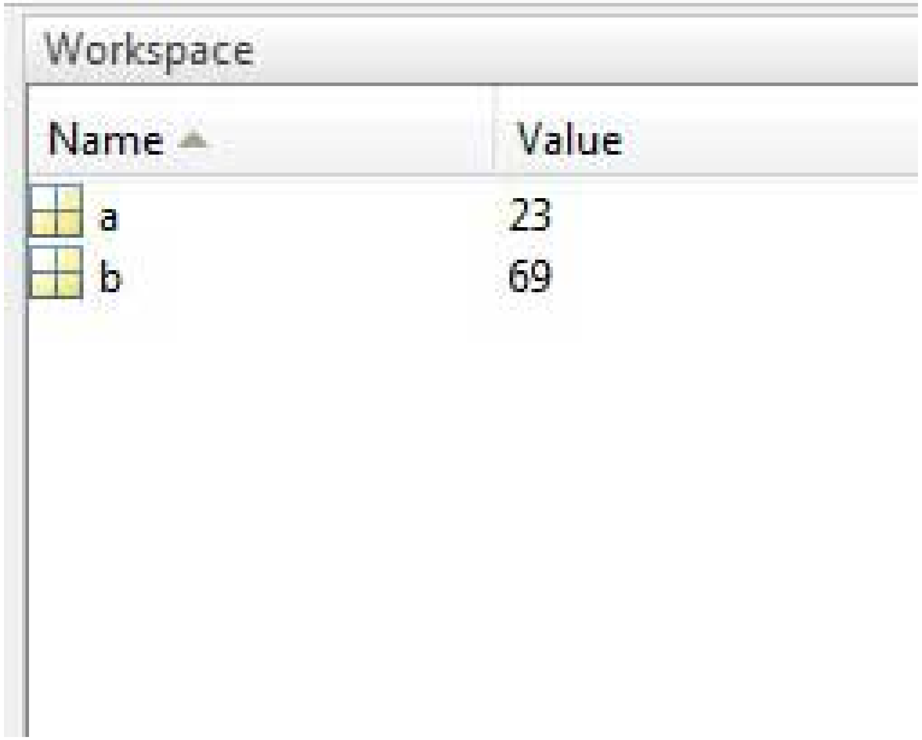
- **Command Window** - This is the main area where you enter commands at the command line, indicated by the command prompt (>>).



```
>> a=23

a =

    23

>> b=69

b =

    69

fx >>
```

Understanding the MATLAB Environment

- **Workspace** - The workspace shows all the variables you create and/or import from files.

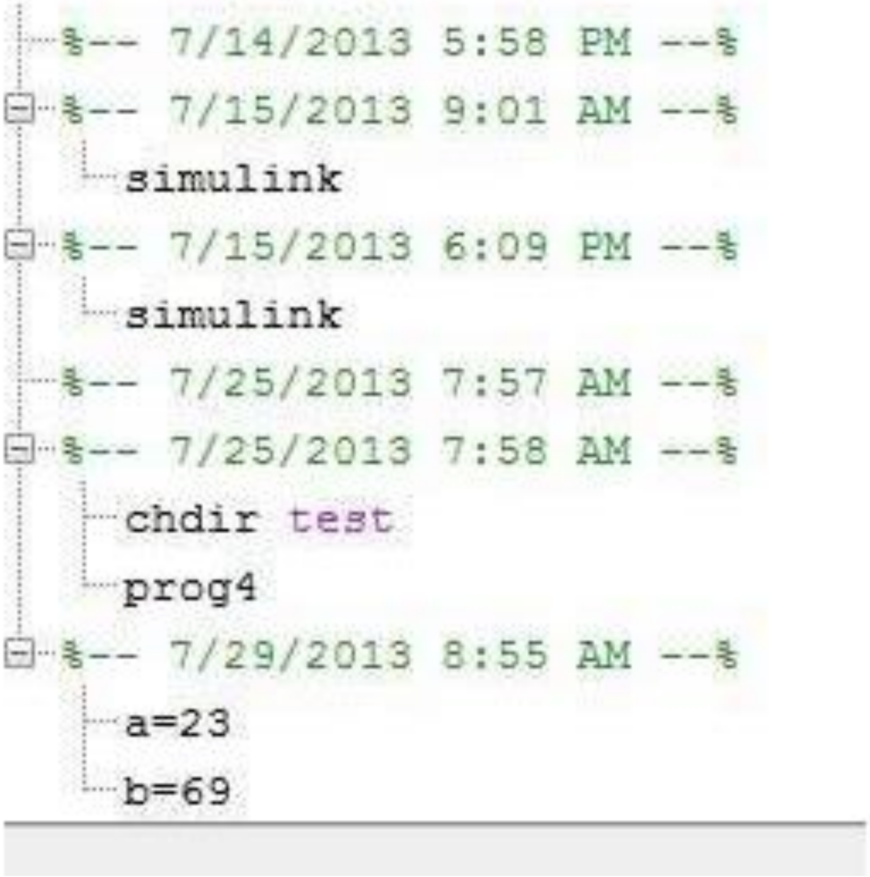


Workspace

Name	Value
a	23
b	69

Understanding the MATLAB Environment

- **Command History** - This panels shows or rerun commands that you entered at the command line.



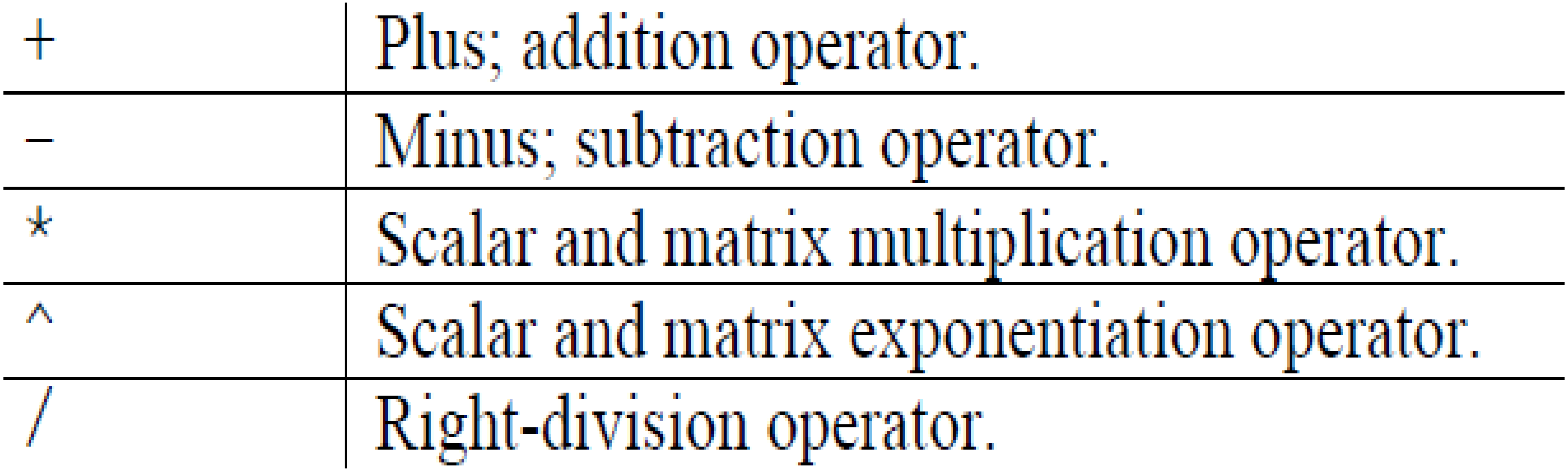
```
%-- 7/14/2013 5:58 PM --%
%-- 7/15/2013 9:01 AM --%
   simulink
%-- 7/15/2013 6:09 PM --%
   simulink
%-- 7/25/2013 7:57 AM --%
%-- 7/25/2013 7:58 AM --%
   chdir test
   prog4
%-- 7/29/2013 8:55 AM --%
   a=23
   b=69
```

Basic Syntax - Using MATLAB as a Calculator

- MATLAB environment behaves like a highly advanced calculator
- Enter your commands at the >> (command prompt) and press ENTER key (↲)
- Examples:

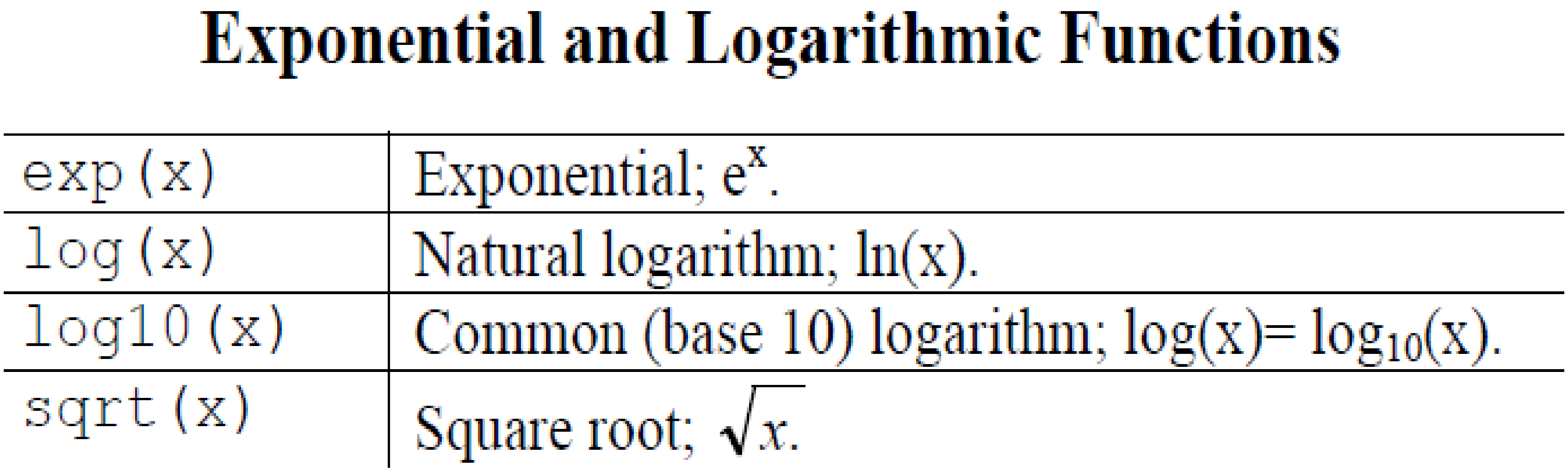
$\gg 5+5$ $ans = 10$ $\gg 6 * 30$ $ans = 180$	$\gg 3\text{^}2$ $ans = 9$ $\gg pi$ $ans = 3.1416$ $\gg \sin(pi/2)$ $ans = 1$

Basic Operators



+	Plus; addition operator.
−	Minus; subtraction operator.
*	Scalar and matrix multiplication operator.
^	Scalar and matrix exponentiation operator.
/	Right-division operator.

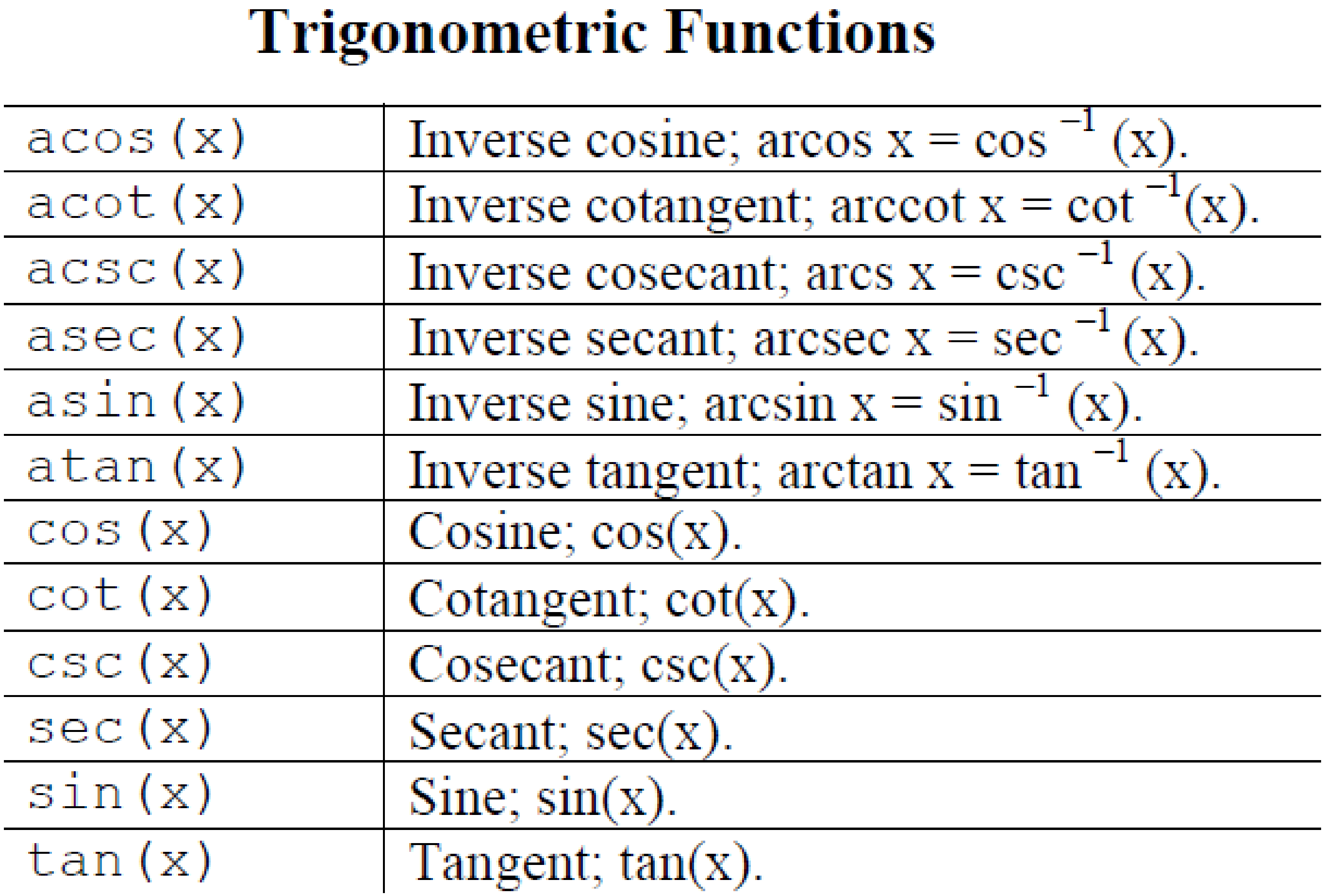
In-built Basic Functions



Exponential and Logarithmic Functions

`exp(x)`	Exponential; e^x.
`log(x)`	Natural logarithm; ln(x).
`log10(x)`	Common (base 10) logarithm; log(x)= $\log_{10}(x)$.
`sqrt(x)`	Square root; $\sqrt{x}$.

In-built Functions

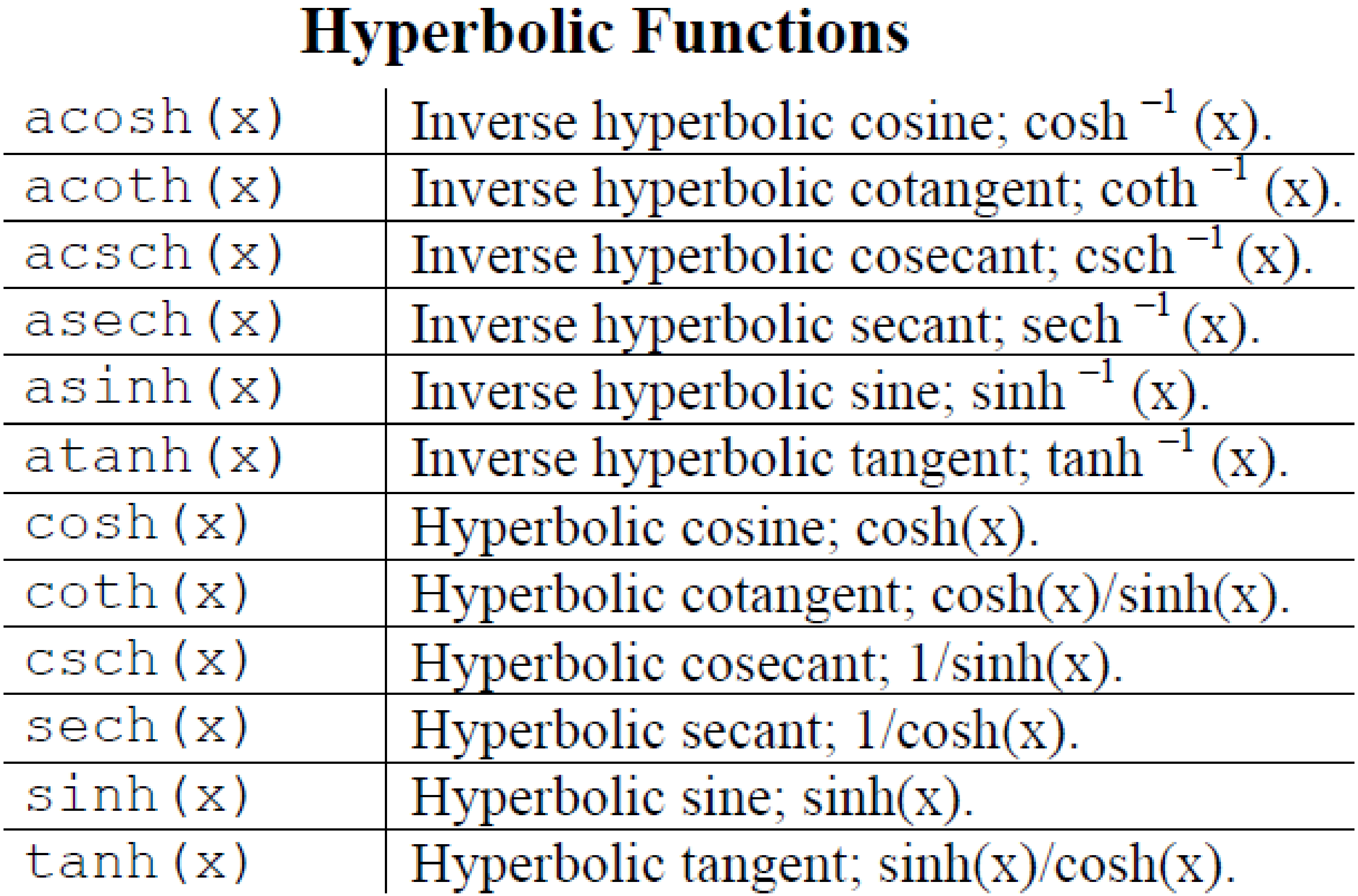


Trigonometric Functions

`acos(x)`	Inverse cosine; arcos x = cos $^{-1}$ (x).
`acot(x)`	Inverse cotangent; arccot x = cot $^{-1}$(x).
`acsc(x)`	Inverse cosecant; arcs x = csc $^{-1}$ (x).
`asec(x)`	Inverse secant; arcsec x = sec $^{-1}$ (x).
`asin(x)`	Inverse sine; arcsin x = sin $^{-1}$ (x).
`atan(x)`	Inverse tangent; arctan x = tan $^{-1}$ (x).
`cos(x)`	Cosine; cos(x).
`cot(x)`	Cotangent; cot(x).
`csc(x)`	Cosecant; csc(x).
`sec(x)`	Secant; sec(x).
`sin(x)`	Sine; sin(x).
`tan(x)`	Tangent; tan(x).

- In the above commands, angle X is in radians.
- For angles in degrees use: sind(), cosd(),…

In-built Functions



Hyperbolic Functions

`acosh(x)`	Inverse hyperbolic cosine; cosh $^{-1}$ (x).
`acoth(x)`	Inverse hyperbolic cotangent; coth $^{-1}$ (x).
`acsch(x)`	Inverse hyperbolic cosecant; csch $^{-1}$ (x).
`asech(x)`	Inverse hyperbolic secant; sech $^{-1}$ (x).
`asinh(x)`	Inverse hyperbolic sine; sinh $^{-1}$ (x).
`atanh(x)`	Inverse hyperbolic tangent; tanh $^{-1}$ (x).
`cosh(x)`	Hyperbolic cosine; cosh(x).
`coth(x)`	Hyperbolic cotangent; cosh(x)/sinh(x).
`csch(x)`	Hyperbolic cosecant; 1/sinh(x).
`sech(x)`	Hyperbolic secant; 1/cosh(x).
`sinh(x)`	Hyperbolic sine; sinh(x).
`tanh(x)`	Hyperbolic tangent; sinh(x)/cosh(x).

Basic Syntax - Complex Numbers

- To represent the imaginary part of complex numbers, use either i or j

$$\gg i$$

$$ans = 0 + 1.0000\ i$$

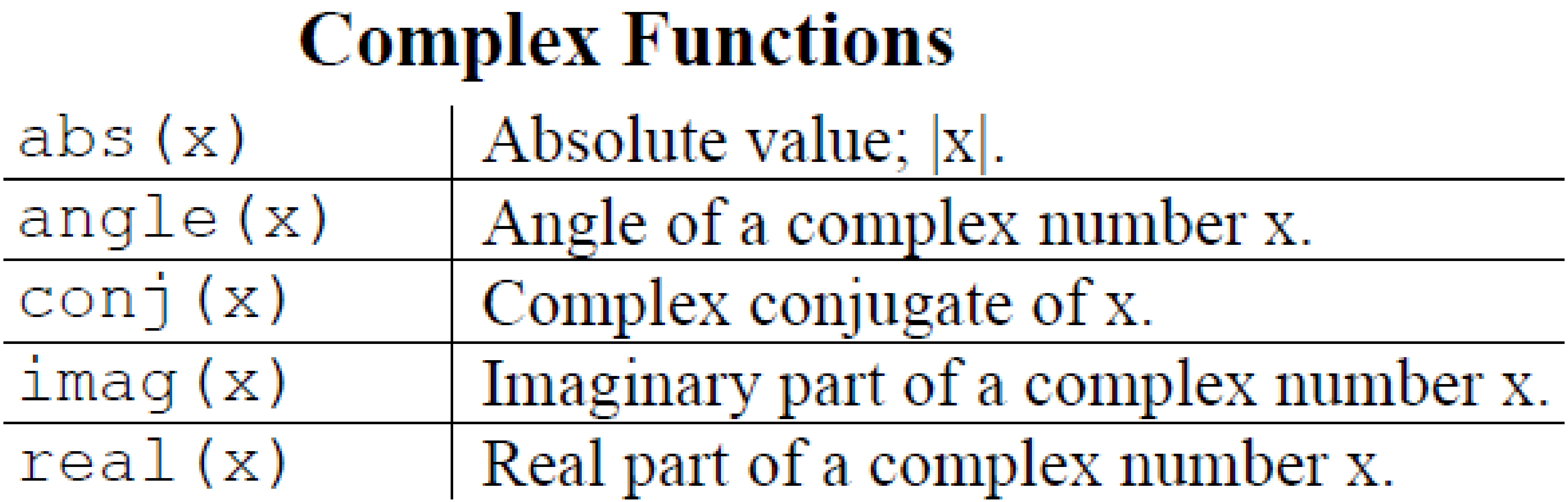
$$\gg j$$

$$ans = 0 + 1.000\ i$$

$$\gg (5 + 6i) * (2 + 9i)$$

$$ans = -44.0000\ + 57.0000i$$

In-built Functions



Complex Functions

`abs(x)`	Absolute value; |x|.
`angle(x)`	Angle of a complex number x.
`conj(x)`	Complex conjugate of x.
`imag(x)`	Imaginary part of a complex number x.
`real(x)`	Real part of a complex number x.

help function

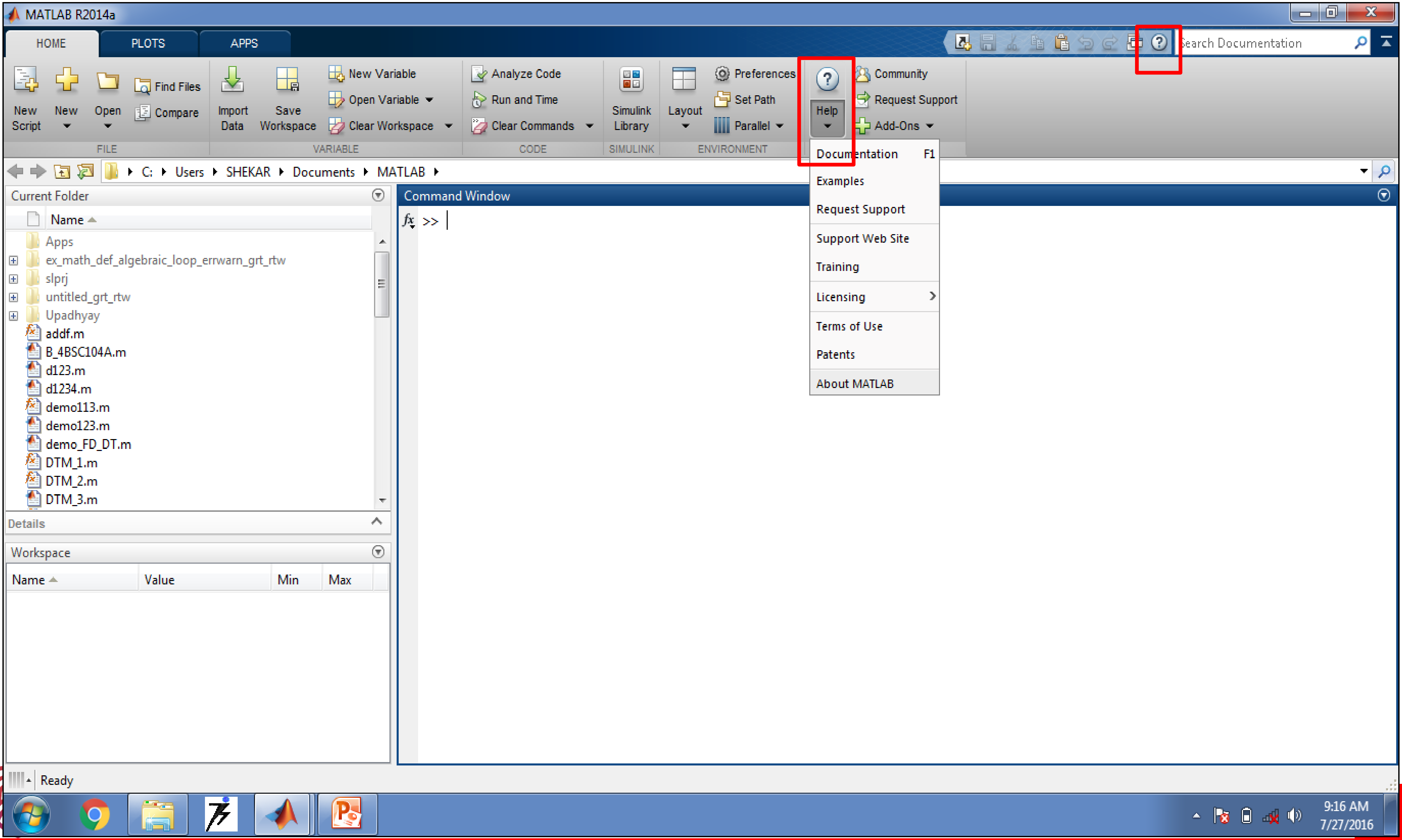
- If you type help followed by the name of a Matlab command, you'll get details about that command
- Very useful to see how a function works or what type of input it expects
- Syntax:

```
>> help command_name
```

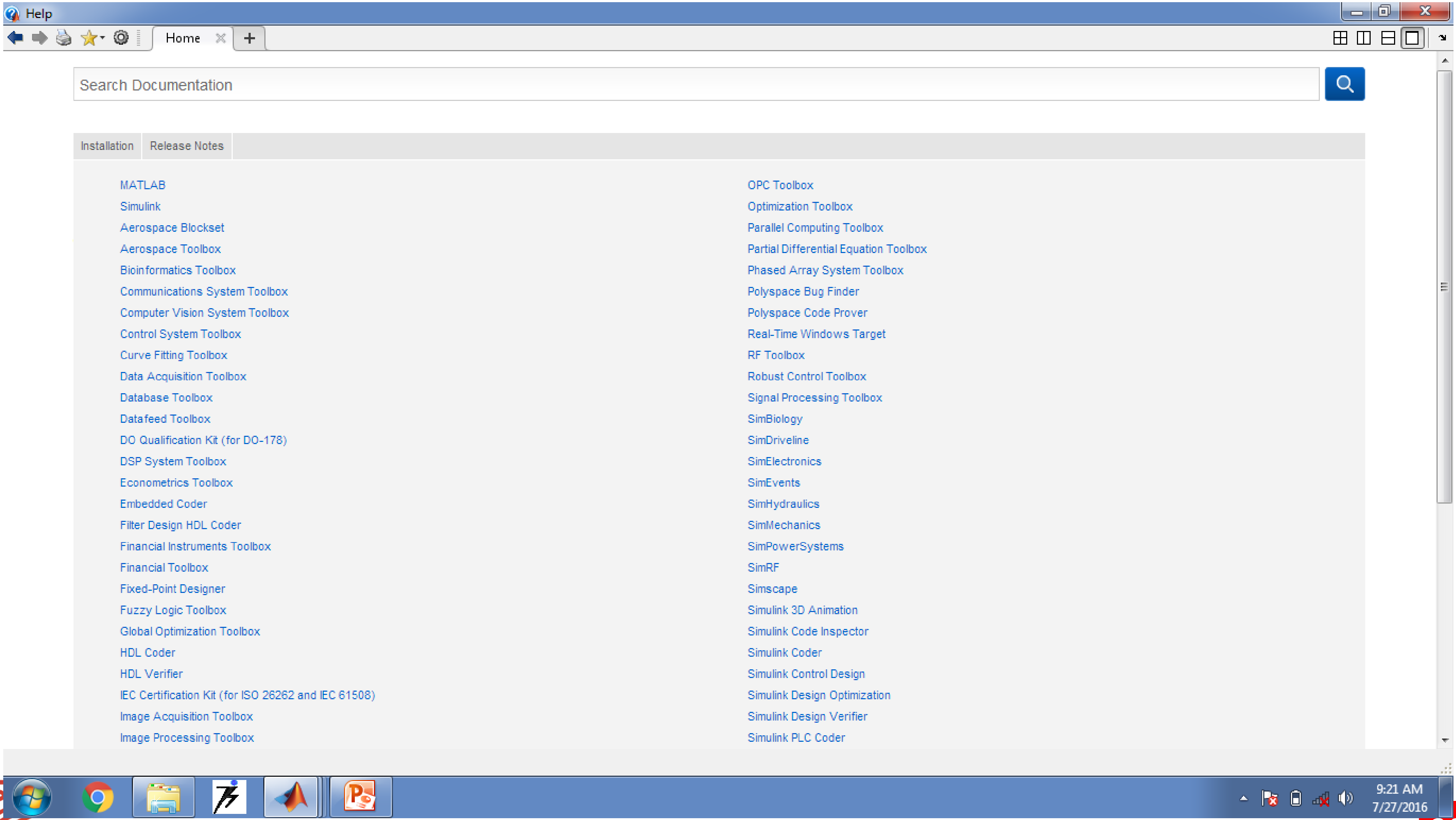
- Examples:

```
>> help sin
>> help round
>> help nthroot
```

Help Using the Help Menu



Help Using the Help Menu



MATLAB
Simulink
Aerospace Blockset
Aerospace Toolbox
Bioinformatics Toolbox
Communications System Toolbox
Computer Vision System Toolbox
Control System Toolbox
Curve Fitting Toolbox
Data Acquisition Toolbox
Database Toolbox
Datafeed Toolbox
DO Qualification Kit (for DO-178)
DSP System Toolbox
Econometrics Toolbox
Embedded Coder
Filter Design HDL Coder
Financial Instruments Toolbox
Financial Toolbox
Fixed-Point Designer
Fuzzy Logic Toolbox
Global Optimization Toolbox
HDL Coder
HDL Verifier
IEC Certification Kit (for ISO 26262 and IEC 61508)
Image Acquisition Toolbox
Image Processing Toolbox

OPC Toolbox
Optimization Toolbox
Parallel Computing Toolbox
Partial Differential Equation Toolbox
Phased Array System Toolbox
Polyspace Bug Finder
Polyspace Code Prover
Real-Time Windows Target
RF Toolbox
Robust Control Toolbox
Signal Processing Toolbox
SimBiology
SimDriveline
SimElectronics
SimEvents
SimHydraulics
SimMechanics
SimPowerSystems
SimRF
Simscape
Simulink 3D Animation
Simulink Code Inspector
Simulink Coder
Simulink Control Design
Simulink Design Optimization
Simulink Design Verifier
Simulink PLC Coder

Basic Syntax - Assigning Values to Variables

- For example, create a variable named b by typing this statement at the command line and press ENTER:

$$\gg b = 5.5$$

- MATLAB adds variable b to the workspace and displays the result in the Command Window:

$$b = 5.5000$$

- Once a variable is entered into the system, you can refer or call it later (unless you quit without saving the file).
- For eg. type b at the command prompt. MATLAB will return the value of b at the prompt.

Basic Syntax – Assigning Values to Variables

- In MATLAB every variable is an array or a matrix.
- For eg., the above statement $b = 5.5$, creates a 1×1 matrix named b and stores the value 5.5 in it.
- Variable name is case sensitive
- Any name can be used – but do not use pre-defined constant/function names like i, j, pi, sin,…
- Do not begin variable name with numerical
- Do not add space in between variable name, rather use under-score(_) for large variable name

Examples

$$\gg bb = 2.45$$
$$bb = 2.4500$$

$$\gg df = \sin(15)$$
$$df = 0.65029$$

$$\gg zx = bb * df$$
$$zx = 1.5932$$

$$var1 = \exp(bb/df)$$
$$var1 = 43.274$$

Semi-colon (;)

- To suppress and hide the output after the execution of a command, add a semicolon (;) after the command
- If a semi-colon is present after an expression, MATLAB will execute the command, but the result will not be displayed on the screen
- Example:

$$\gg x = 5.7;$$

$$\gg y = \cos(12.3)\,;$$

$$\gg new = x * \exp(y)$$

$$new = 14.957$$

$$\gg y$$

$$y = 0.96473$$

Comment (%)

- One can add short descriptions in the code by using the % symbol called comment
- Anything written right to % is ignored by MATLAB and will not be executed

$$\gg g = 9.8;\ \%\text{ acceleration due to gravity}$$

$$\gg eng = 2.3 * 1.6 * 10^{-19};\ \%\text{ energy in Joules}$$

- Increases the readability of the program
- Useful tool for debugging (fixing problems) programs
- To eliminate a few lines of code temporarily, you can ‘comment them out’ rather than cutting and pasting them into and out of your program.

clc, clear, clear all

- clc – Clear Command Window

```
>> clc
```

- Clears all input and output from the Command Window display, giving a "clean screen."
- clc only clears the command window, the variables defined are not cleared/deleted
- clear - Deletes variables from workspace/memory

```
>> clear
```

- clear all - Clears all objects (variables, functions)

```
>> clear all
```

- To clear a particular variable say x1, type

```
>> clear x1
```

save and load

- Variables defined at the command prompt (workspace variables) gets deleted once you quit or exit MATLAB
- Variables can be saved for later use using save command

```
>> save file_name
```

- Saving preserves the workspace in your current working folder in a compressed file with a .mat extension, called a MAT-file.
- To restore data from the saved file to the workspace use the load command

```
>> load file_name
```

Character Strings

- Strings can be concatenated using strcat
- strcat(s1,...,sN) - concatenates strings s1,...,sN.
- Strings can also be concatenated using square brackets

```
≫ aa = ‘good’;
≫ bb = ‘morning’;
≫ cc = strcat(aa,bb)
cc = goodmorning
≫ my_string = [aa bb]
≫ my_string = goodmorning
```

Character Strings

```
≫ aa = ‘name’;
≫ bb = ‘mail’;
≫ cc = ‘com’;
≫ id = [aa,’@’,bb,’.’cc]
≫ id = name@mail.com
```

- num2str or int2str - convert numeric values to strings

```
≫ g = 9.8;
≫ f = 2*g;
≫ f_value = [‘force on the particle is’,num2str(f),’ Newton’]
f_value = force on the particle is 19.6 Newton
```

Anonymous Functions

- Anonymous functions provide an easy way to specify a function.
- It lets you develop an analytical expression of one or more inputs and either assign that expression to a variable or pass it as an argument to a function.
- The basic syntax is

  ≫ function_name = @(variables) matlab_expression;
- Function_name is called the function handle.
- @ operator creates the handle
- Parentheses () immediately after the @ operator include the function input arguments
- matlab_expression is the expression to be evaluated

Anonymous Functions

```
>>  myfunc = @(x) sin(x)/x;
>>  myfunc(2)
ans =  0.45465

>>  cube = @(x,y) x^3+y^3;
>>  cube(3,2)
ans = 35

>> a = 1.3;
>> b = .2;
>> c = 30;
>> parabola = @(x) a*x.^2 + b*x + c;
>> parabola(1)
```

Programming and Scripts

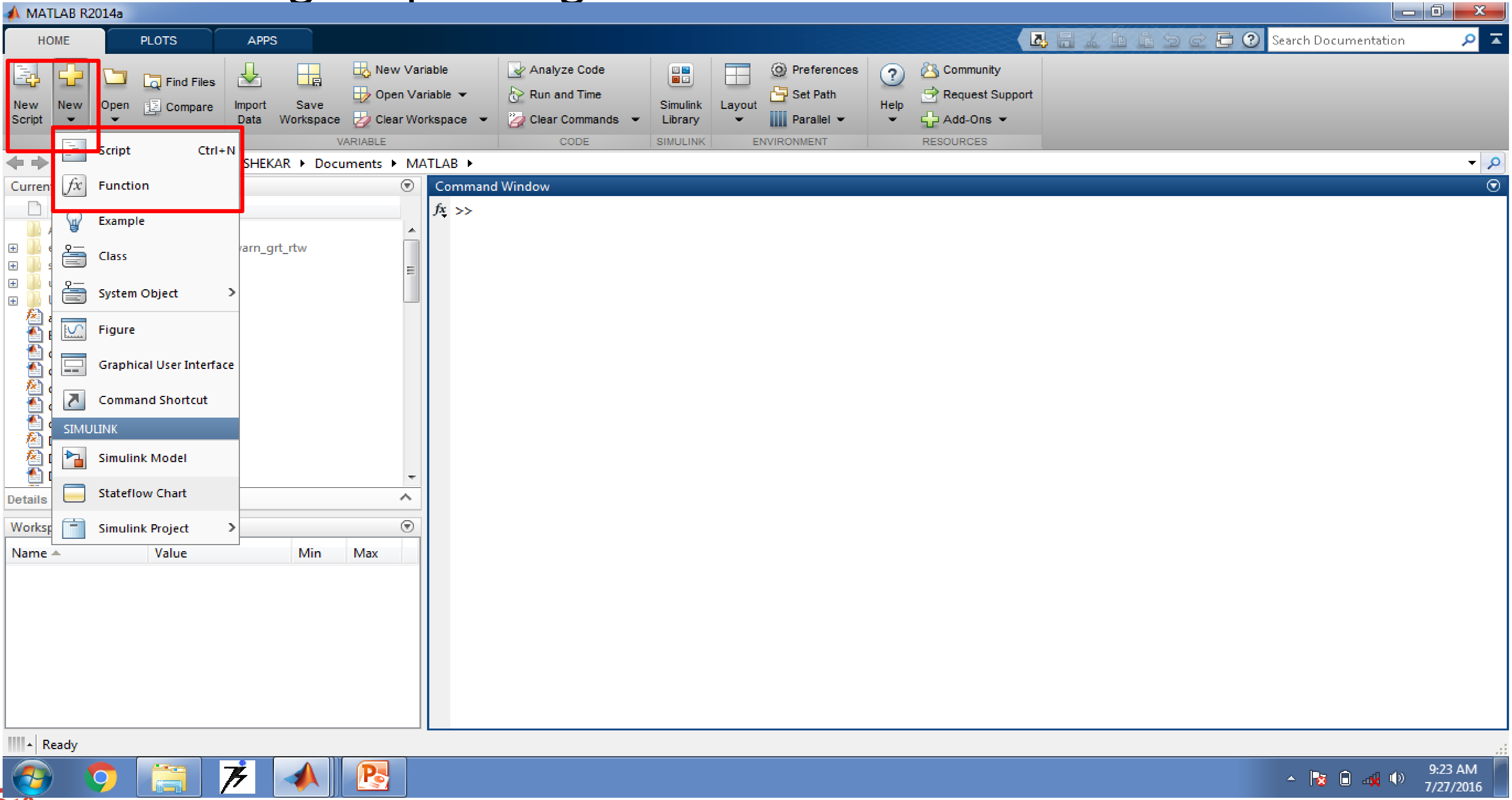
- The simplest type of MATLAB program is called a script.
- A script is a file with a .m extension that contains multiple sequential lines of MATLAB commands and function calls.
- You can run a script by typing its name at the command line.
- To create a script, use the edit command

```
>> edit file_name
```

- This opens a blank file with .m extension in MATLAB editor
- Enter the code and then save

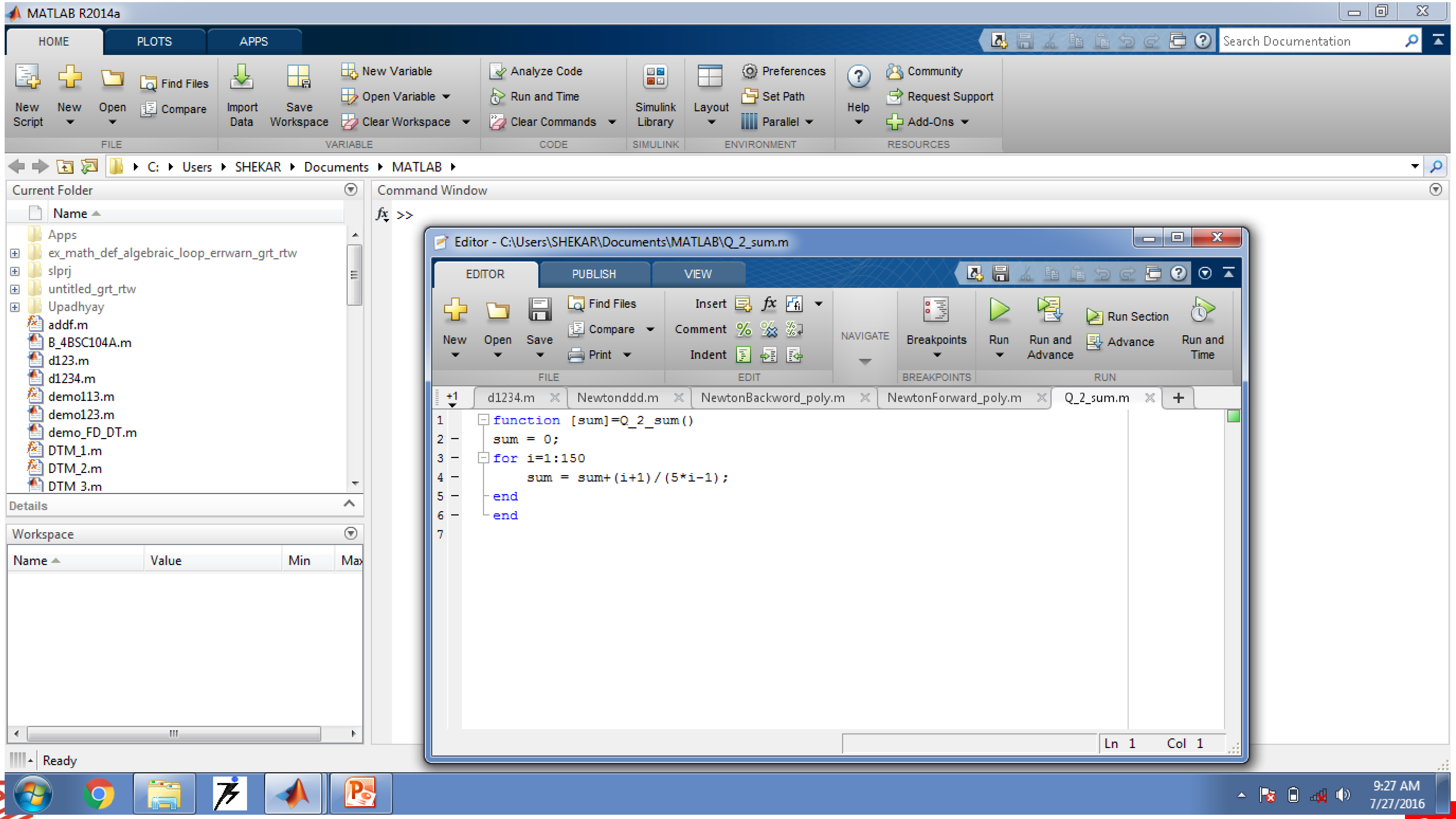
Programming and Scripts

- Creating script using GUI



- You can also use CTRL+N

Programming and Scripts



```
function [sum]=Q_2_sum()
sum = 0;
for i=1:150
    sum = sum+(i+1)/(5*i-1);
end
end
```

Programming and Scripts

- Example: Create a script file named tarea to calculate area of a triangle
- First create blank script file and enter the following commands in the script file:

```
b = 10; % base of the triangle
h = 4; % height of the triangle
area = 0.5*(b*h)
```

- Now save the script file using the save button (or file menu) in the script editor.
- Enter the file name as tarea when you save the file
- Extension .m would be added automatically

Programming and Scripts

- To run the script, type the file name at command prompt

```
>> tarea
area = 20
```

- One can also click the run icon ▶ in the MATLAB editor to run the script
- Whenever you write code, it is a good practice to add comments that describe the code
- Comments allow others to understand your code, and can refresh your memory when you return to it later

input and fprintf - example

- Calculating area of triangle. Type code in script, save and run.

```
b = input(‘Base of the triangle, b: ’)
h = input(‘Height of the triangle, h: ’)
area = 0.5*(b*h);
fprintf(‘The area of the triangle with base %f and height %f is %f\n’, b, h, area)
```

Basics: Matrix

- How to define a matrix/vector
  - A = [1 2 3 4; 4 5 6 7] ~~ [1:4; 4:7]   (!!! Comma, colon, semicolon bracket)
- Special matrix
  - zeros(m,n)
  - ones(m,n)
- diag(vec)
- Matrix operation
  - Basic arithmetic operation (!!! Period & dimensions)
  - Inverse (inv) and transpose (apostrophe)
  - Read/change matrix component (!!! parenthesis)
  - Stacking and breaking
  - Size(), length(), eig()

Array, Matrix

- a vector `x = [1 2 5 1]`

```
x =
     1     2     5     1
```

- a matrix `x = [1 2 3; 5 1 4; 3 2 -1]`

```
x =
     1     2     3
     5     1     4
     3     2    -1
```

- transpose `y = x’`

```
y =
     1
     2
     5
     1
```

Long Array, Matrix

-
  ```
  t =1:10
  ```

```
t =
      1    2    3    4   5   6    7   8    9    10
```

-
  ```
  k =2:-0.5:-1
  ```

```
k =
      2   1.5   1   0.5   0   -0.5   -1
```

-
  ```
  B = [1:4; 5:8]
  ```

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

Generating Vectors from functions

- zeros(M,N) MxN matrix of zeros

```
x = zeros(1,3)
x =
   0      0      0
```

- ones(M,N) MxN matrix of ones

```
x = ones(1,3)
x =
   1      1      1
```

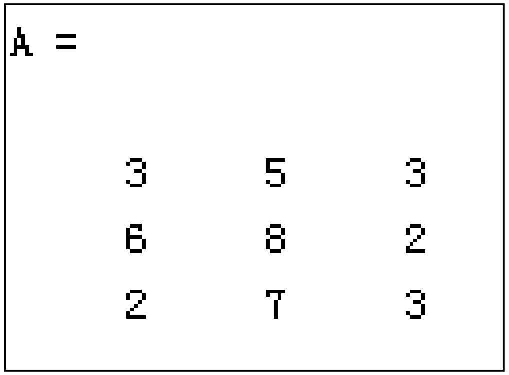
- rand(M,N) MxN matrix of uniformly distributed random numbers on (0,1)

```
x = rand(1,3)
x =
 0.9501  0.2311 0.6068
```

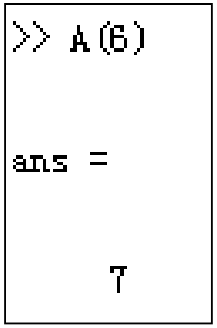
Matrix Index

- The matrix indices begin from 1 (not 0 (as in C))
- The matrix indices must be positive integer

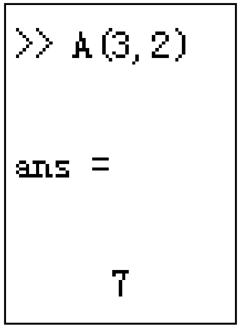
Given:



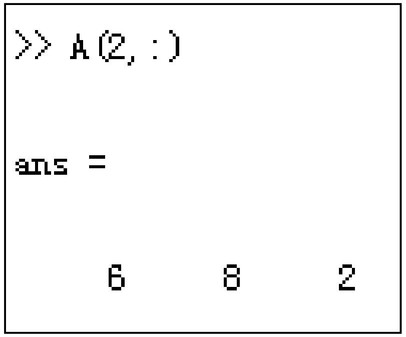
```
A =

     3     5     3
     6     8     2
     2     7     3
```



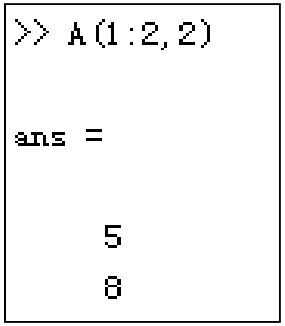
```
>> A(6)

ans =

     7
```



```
>> A(3,2)

ans =

     7
```



```
>> A(2,:)

ans =

     6     8     2
```



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

ans =

     5
     8
```

A(-2), A(0)

Error: ??? Subscript indices must either be real positive integers or logicals.

A(4,2)

Error: ??? Index exceeds matrix dimensions.

Concatenation of Matrices

-
  ```
  x = [1 2], y = [4 5], z=[ 0 0]

  A = [ x y]

      1    2    4    5

   B = [x ; y]

       1  2
       4  5
  ```

```
C = [x y ;z]
```

Error:

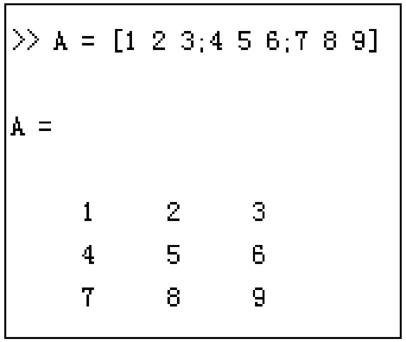
??? Error using ==> vertcat CAT arguments dimensions are not consistent.

Operators (arithmetic)

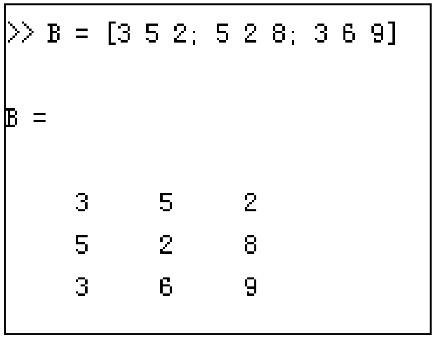
+	addition
-	subtraction
*	multiplication
/	division
^	power
‘	complex conjugate transpose

Matrices Operations

Given A and B:

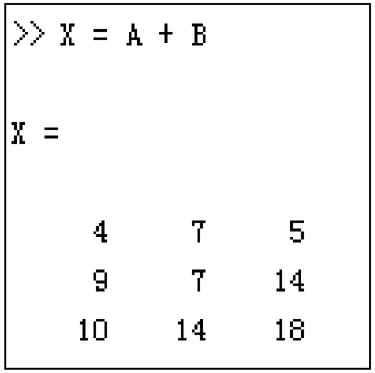


```
>> A = [1 2 3;4 5 6;7 8 9]

A =

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



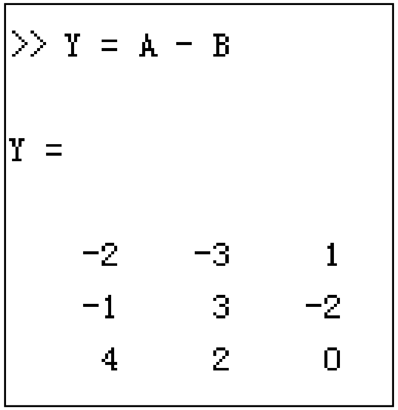
```
>> B = [3 5 2; 5 2 8; 3 6 9]

B =

     3     5     2
     5     2     8
     3     6     9
```

Addition



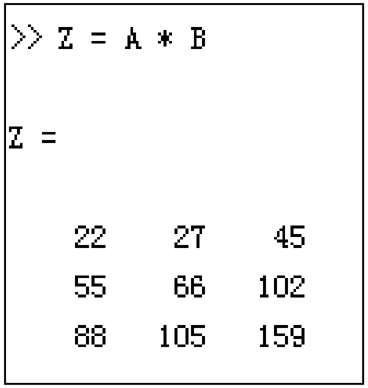
```
>> X = A + B

X =

     4     7     5
     9     7    14
    10    14    18
```

Subtraction



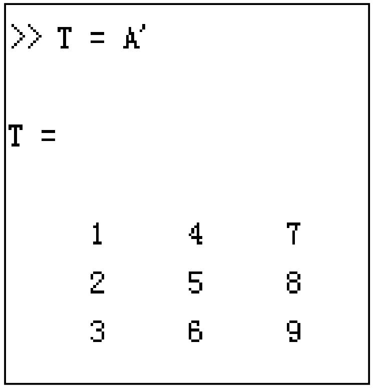
```
>> Y = A - B

Y =

    -2    -3     1
    -1     3    -2
     4     2     0
```

Product



```
>> Z = A * B

Z =

    22    27    45
    55    66   102
    88   105   159
```

Transpose



```
>> T = A'

T =

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

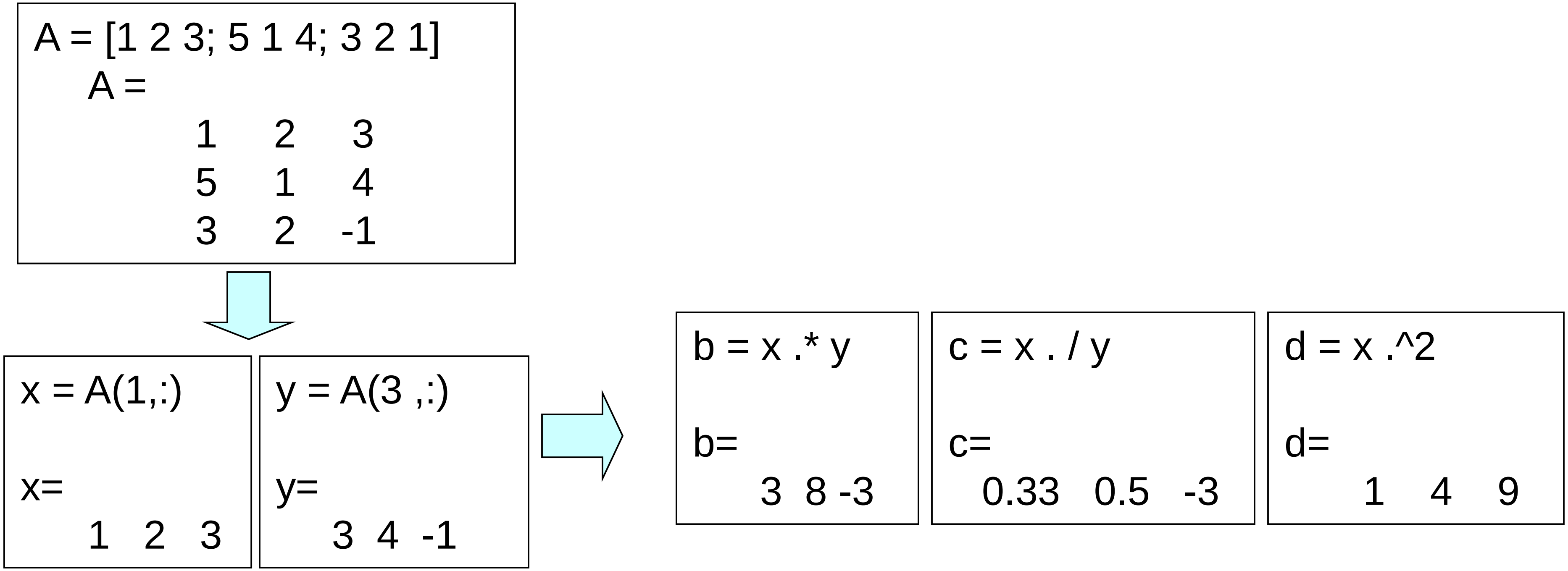
Operators (Element by Element)

.* element-by-element multiplication

./ element-by-element division

.^ element-by-element power

The use of “.” – “Element” Operation



```
A = [1 2 3; 5 1 4; 3 2 1]
     A =
              1     2     3
              5     1     4
              3     2    -1
```

```
x = A(1,:)

x=
     1   2   3
```

```
y = A(3 ,:)

y=
     3  4  -1
```

```
b = x .* y

b=
     3  8 -3
```

```
c = x . / y

c=
  0.33   0.5   -3
```

```
d = x .^2

d=
      1    4    9
```

```
K= x^2
Erorr:
 ??? Error using ==> mpower  Matrix must be square.
B=x*y
Erorr:
??? Error using ==> mtimes Inner matrix dimensions must agree.
```

Creating Matrices

- `zeros(m, n)`: matrix with all zeros
- `ones(m, n)`: matrix with all ones.
- `eye(m, n)`: the identity matrix
- `rand(m, n)`: uniformly distributed random
- `randn(m, n)`: normally distributed random
- `magic(m)`: square matrix whose elements have the same sum, along the row, column and diagonal.
- `pascal(m)` : Pascal matrix.

Matrix operations

- `^`: exponentiation
- `*`: multiplication
- `/`: division
- `\`: left division. The operation `A\B` is effectively the same as `INV(A)*B`, although left division is calculated differently and is much quicker.
- `+`: addition
- `-`: subtraction

Array Operations

- Evaluated element by element

  `.'` : array transpose (non-conjugated transpose)

  `.^` : array power

  `.*` : array multiplication

  `./` : array division

- Very different from Matrix operations

```
>> A=[1 2;3 4];
>> B=[5 6;7 8];
>> A*B
    19    22
    43    50
```

But:

```
>> A.*B
     5    12
    21    32
```

Some Built-in functions

- `mean(A)`:mean value of a vector
- `max(A), min (A)`: maximum and minimum.
- `sum(A)`: summation.
- `sort(A)`: sorted vector
- `median(A)`: median value
- `std(A)`: standard deviation.
- `det(A) :` determinant of a square matrix
- `dot(a,b)`: dot product of two vectors
- `Cross(a,b)`: cross product of two vectors
- `Inv(A)`: Inverse of a matrix A

Indexing Matrices

Given the matrix:

```
A  =                 n
        0.9501     0.6068     0.4231
 m      0.2311     0.4860     0.2774
```

Then:

```
A(1,2) = 0.6068
```

→ $A_{ij}, i = 1..m, j = 1..n$

```
A(3) = 0.6068
```

→ $index = (i-1)m + j$

```
A(:,1) = [0.9501
  1:m
           0.2311 ]

A(1,2:3)=[0.6068   0.4231]
```

Adding Elements to a Vector or a Matrix

```
>> A=1:3
A=
   1  2  3
>> A(4:6)=5:2:9
A=
   1  2  3  5  7  9

>> B=1:2
B=
   1  2
>> B(5)=7;
B=
   1  2  0  0  7
```

```
>> C=[1 2; 3 4]
C=
   1  2
   3  4
>> C(3,:)=[5 6];
C=
   1  2
   3  4
   5  6

>> D=linspace(4,12,3);
>> E=[C D’]
E=
   1  2  4
   3  4  8
   5  6  12
```

Solutions to Systems of Linear Equations

- Example: a system of 3 linear equations with 3 unknowns (x_1, x_2, x_3):

$$\begin{aligned} 3x_1 + 2x_2 - x_3 &= 10 \\ -x_1 + 3x_2 + 2x_3 &= 5 \\ x_1 - x_2 - x_3 &= -1 \end{aligned}$$

Let :

$$A = \begin{bmatrix} 3 & 2 & 1 \\ -1 & 3 & 2 \\ 1 & -1 & -1 \end{bmatrix} \qquad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad b = \begin{bmatrix} 10 \\ 5 \\ -1 \end{bmatrix}$$

Then, the system can be described as:

$$Ax = b$$

Solutions to Systems of Linear Equations (con’t…)

- Solution by Matrix Inverse:

  $Ax = b$

  $A^{-1}Ax = A^{-1}b$

  $x = A^{-1}b$

- MATLAB:

```
>> A = [ 3 2 -1; -1 3 2; 1 -1 -1];
>> b = [ 10; 5; -1];
>> x = inv(A)*b
x =
   -2.0000
    5.0000
   -6.0000
```

Answer:

$x_1 = -2,\ x_2 = 5,\ x_3 = -6$

- Solution by Matrix Division:

  The solution to the equation

  $$Ax = b$$

  can be computed using left division.

- MATLAB:

```
>> A = [ 3 2 -1; -1 3 2; 1 -1 -1];
>> b = [ 10; 5; -1];
>> x = A\b
x =
   -2.0000
    5.0000
   -6.0000
```

Answer:

$x_1 = -2,\ x_2 = 5,\ x_3 = -6$

NOTE: left division: A\b ➔ b ÷ Aright division: x/y ➔ x ÷ y

plot

- plot – command for plotting graphs
- To plot a graph the following steps are required:

1. Specify the x-axis, by specifying the range of values for the variable x, for which the function is to be plotted
2. Define the function, y = f(x), the function to be plotted
3. Use the plot(x, y) command to plot the function y against x

plot - examples

```
x = [0 1 2 3 4 5 6 7 8 9 10];
y = exp(x);
plot(x,y)
```

plot – x-axis

To create a large range of x-axis use : operator or linspace:

x = initial value : increment : final value

If increment is not specified, default increment is 1

x = 1:100 % (produces x=[1 2 3 …… 100])

x = 1:.1:10 %(produces x = [1 1.1 1.2 …… 100])

x = 0:pi/50:2*pi %(produces x = $\left[0\ \frac{\pi}{50} \frac{2\pi}{50}\ \ldots\ldots 2\pi\ \right]$)

linspace(a,b,n) – generates linearly spaced grid of length n from a to b

x = linspace(0,20,5) generates x = [0 5 10 15 20]

x = linspace(a,b,n) is same as x = a : (b-a)/(n-1) : b

plot- examples

```
x = 0:.1:10;
y = exp(x);
plot(x,y)
```

____

```
x = linspace(0,10,11);
y = exp(x);
plot(x,y)
```

____

Also see: plot(x,y,’r--’), plot(x,y,’r*’)

Plot – labelling the axes

- Use xlabel and ylabel command

```
x = linspace(0,2*pi,100);
y = sin(x);
plot(x,y)
xlabel(‘x’)
ylabel(‘sin(x)’)
```

Plot – adding title

- Use title command

```
x = linspace(0,2*pi,100);
y = sin(x);
plot(x,y)
xlabel(‘x’)
ylabel(‘sin(x)’)
title(‘Graph of Sine Function’)
```

Basic Task: Plot the function sin(x) between 0≤x≤4*π*

- Create an x-array of 100 samples between 0 and 4π.

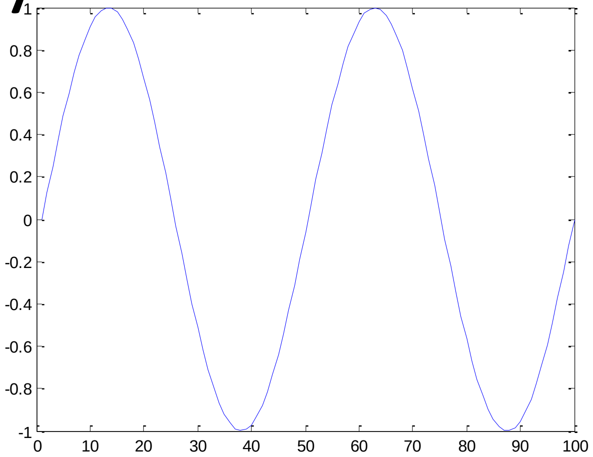
```
>>x=linspace(0,4*pi,100);
```

- Calculate sin(.) of the x-array

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

- Plot the y-array

```
>>plot(y)
```



Plot the function $e^{-x/3}$sin(x) between $0 \leq x \leq 4\pi$

- Create an x-array of 100 samples between 0 and 4π.

```
>>x=linspace(0,4*pi,100);
```

- Calculate sin(.) of the x-array

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

- Calculate $e^{-x/3}$ of the x-array

```
>>y1=exp(-x/3);
```

- Multiply the arrays y and y1

```
>>y2=y*y1;
```

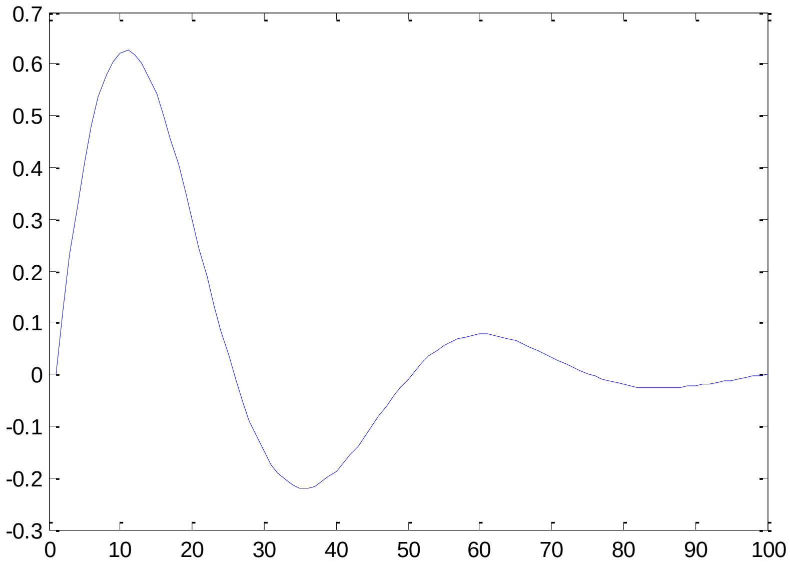
Plot the function $e^{-x/3}$sin(x) between $0\leq x\leq 4\pi$

- Multiply the arrays y and y1 correctly

```
>>y2=y.*y1;
```

- Plot the y2-array

```
>>plot(y2)
```

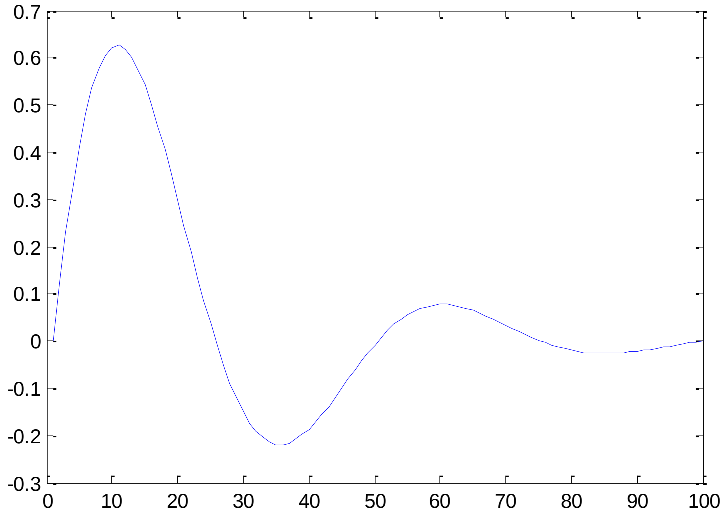


Display Facilities

- plot(.)

Example:

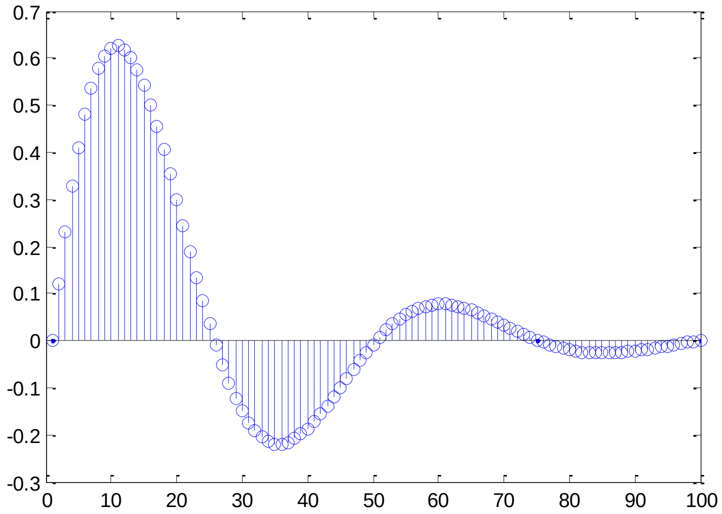
```
>>x=linspace(0,4*pi,100);
>>y=sin(x);
>>plot(y)
>>plot(x,y)
```



- stem(.)

Example:

```
>>stem(y)
>>stem(x,y)
```



Display Facilities

- title(.)

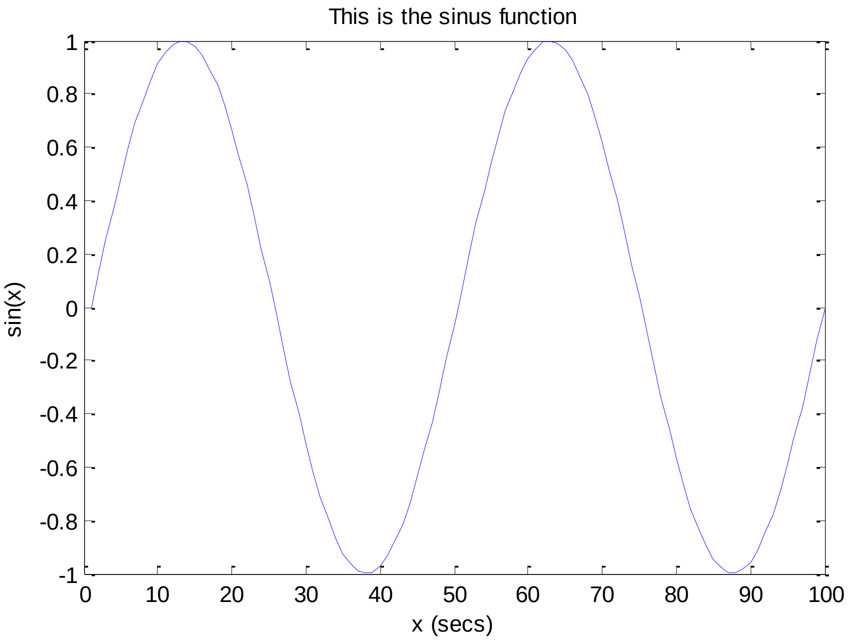
```
>>title('This is the sinus function')
```

- xlabel(.)

```
>>xlabel('x (secs)')
```

- ylabel(.)

```
>>ylabel('sin(x)')
```



Operators (relational, logical)

- == Equal to
- ~= Not equal to
- < Strictly smaller
- > Strictly greater
- <= Smaller than or equal to
- >= Greater than equal to
- & And operator
- | Or operator

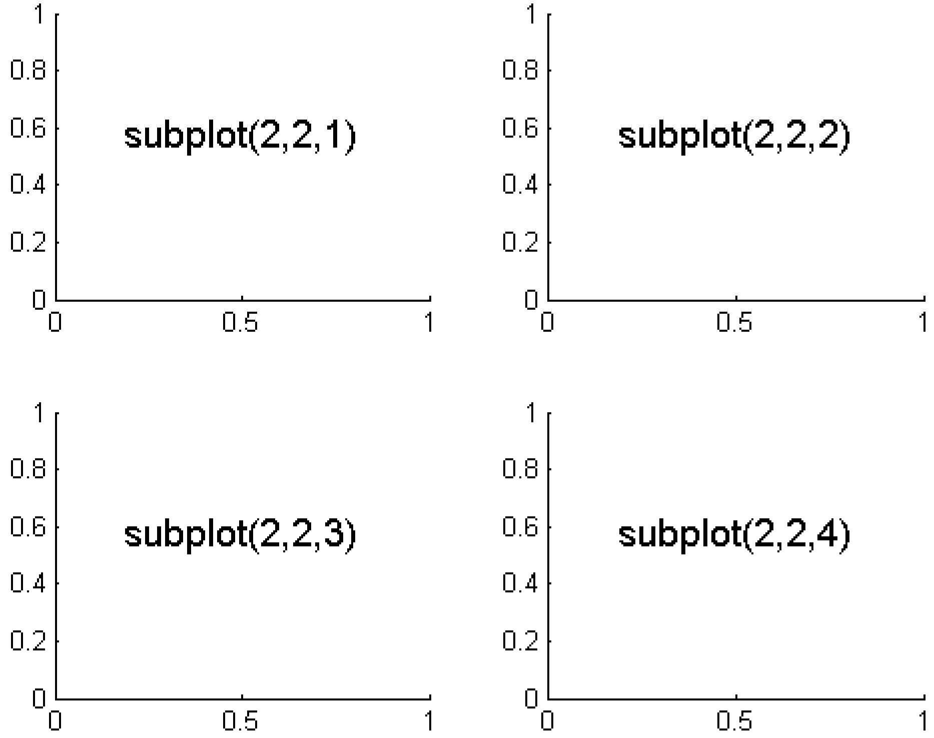
Plot – adding more graphs

- Use hold on command

```
x = 0:pi/100:2*pi;
y = sin(x);
plot(x,y)
hold on
y2 = cos(x);
plot(x,y2,':')
legend(‘sin’, ‘cos’)
```

subplot

```
x=linspace(-5,5);
y1=sin(x);
subplot(2,2,1)
plot(x,y1)
title(‘first subplot’)
y2=sin(2*x);
subplot(2,2,2)
plot(x,y2)
title(‘second subplot’)
y3=sin(3*x);
subplot(2,2,3)
plot(x,y3)
title(‘third subplot’)
y4=sin(4*x);
subplot(2,2,4)
plot(x,y4)
title(‘fourth subplot’)
```



Session Summary

- MATLAB environment behaves like a highly advanced scientific calculator
- In MATLAB every variable is an array or a matrix
- The simplest type of MATLAB program is called a script
- A script is a file with a .m extension that contains multiple sequential lines of MATLAB commands and function calls
- ^ exponentiation, *multiplication, / division, \ left division (The operation A\B is effectively the same as INV(A)*B, although left division is calculated differently and is much quicker)