Using MATLAB

MATLAB introduction

- MATLAB is a program for doing numerical computation.
- It was originally designed for solving linear algebra type problems using matrices.
- It's name is derived from MATrix LABoratory.
- MATLAB has since been expanded and now has built-in functions for solving problems requiring data analysis, signal processing, optimization, and several other types of scientific computations.
- It also contains functions for 2-D and 3-D graphics and animation.

MATLAB System

- MATLAB system consists of five main parts
  - Development Environment
    - Set of tools and facilities that help you use MATLAB functions and files.
  - MATLAB Mathematical Function Library
    - Collection of functions like sum, sine, cosine, and complex arithmetic, matrix inverse, matrix eigenvalues, and fast Fourier transforms.

MATLAB System

– The MATLAB Language

  - High-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features.

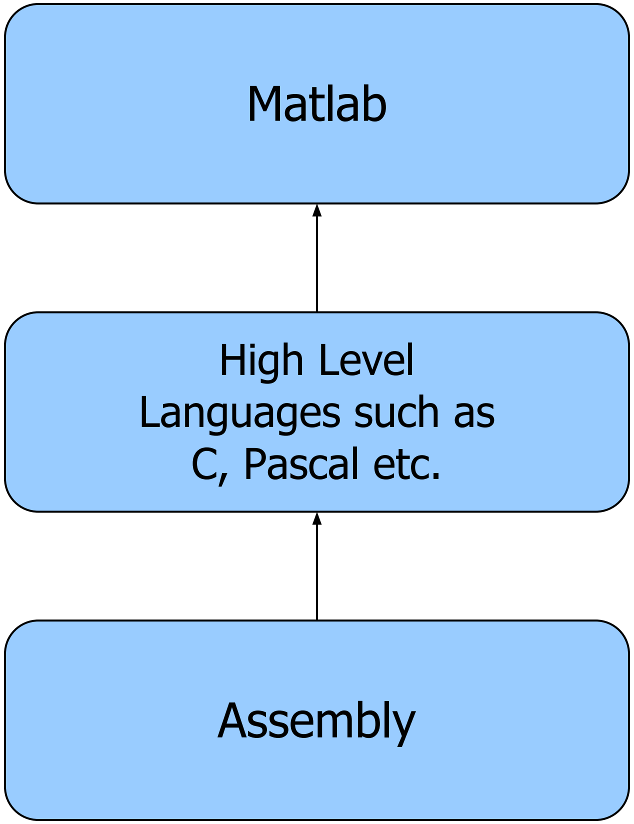
– Graphics

  - Provides extensive facilities for displaying vectors and matrices as graphs, as well as annotating and printing these graphs.

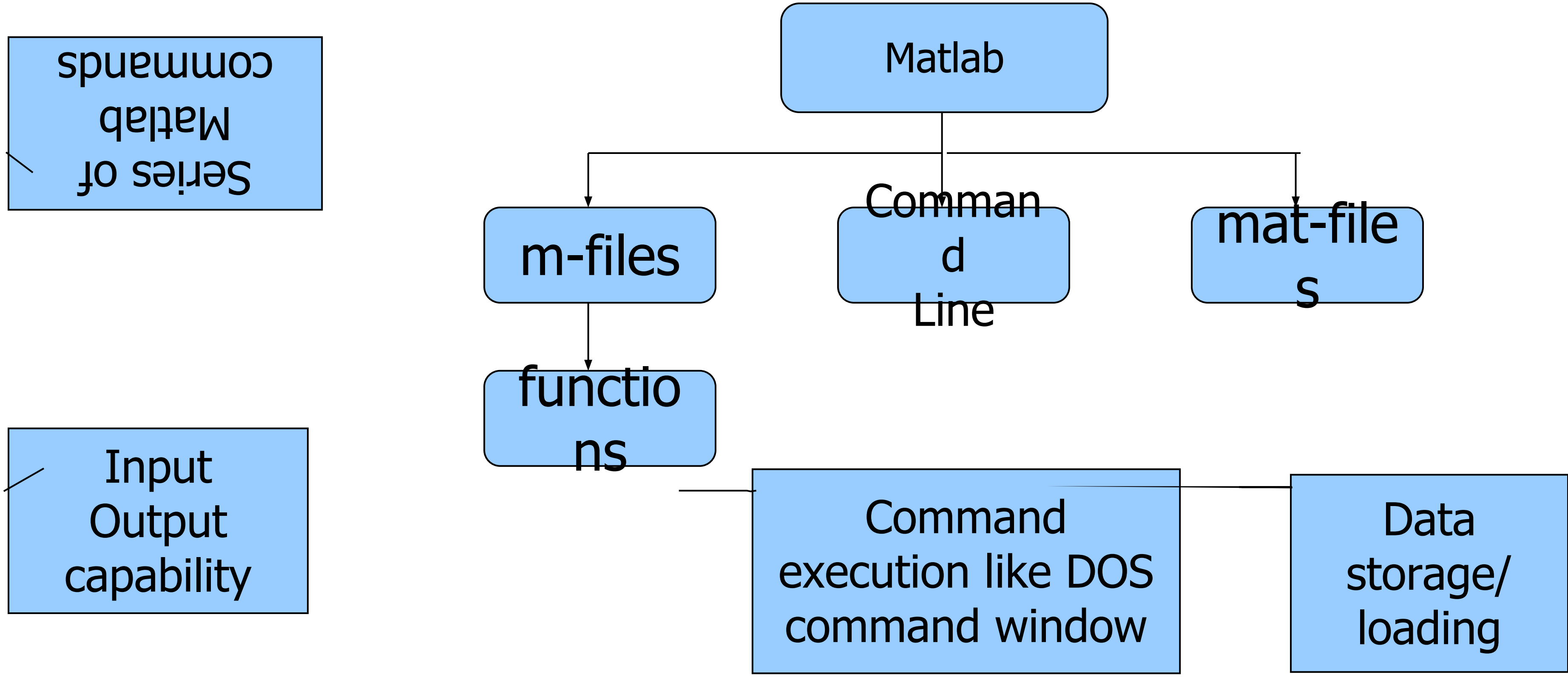
– Application Program Interface (API)

MATLAB

- Matlab is basically a high level language which has many specialized toolboxes for making things easier for us
- How high?



Matlab components



Matlab

m-files

Comman
d
Line

mat-file
s

functio
ns

Command
execution like DOS
command window

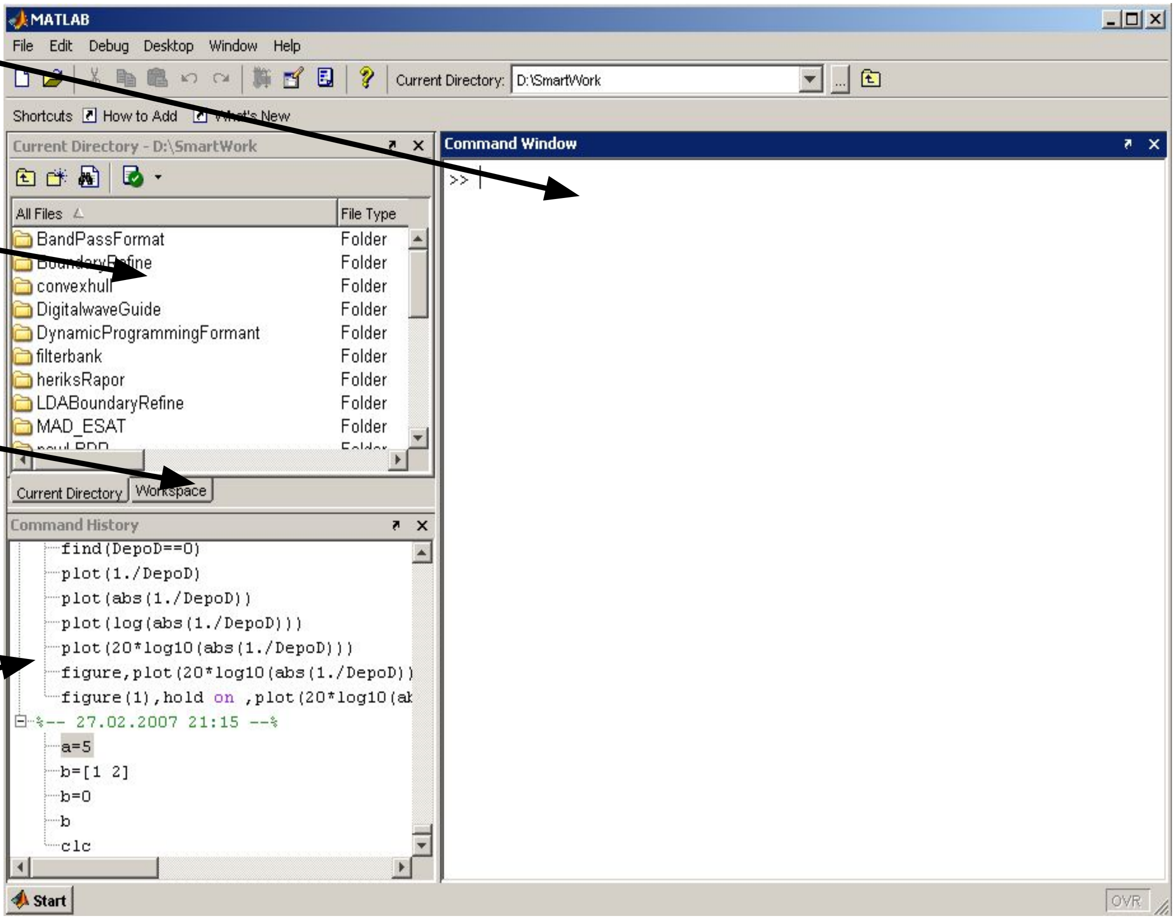
Data
storage/
loading

Series of
Matlab
commands

Input
Output
capability

Matlab Screen

- Command Window
  - type commands
- Current Directory
  - View folders and m-files
- Workspace
  - View program variables
  - Double click on a variable to see it in the Array Editor
- Command History
  - view past commands
  - save a whole session using diary

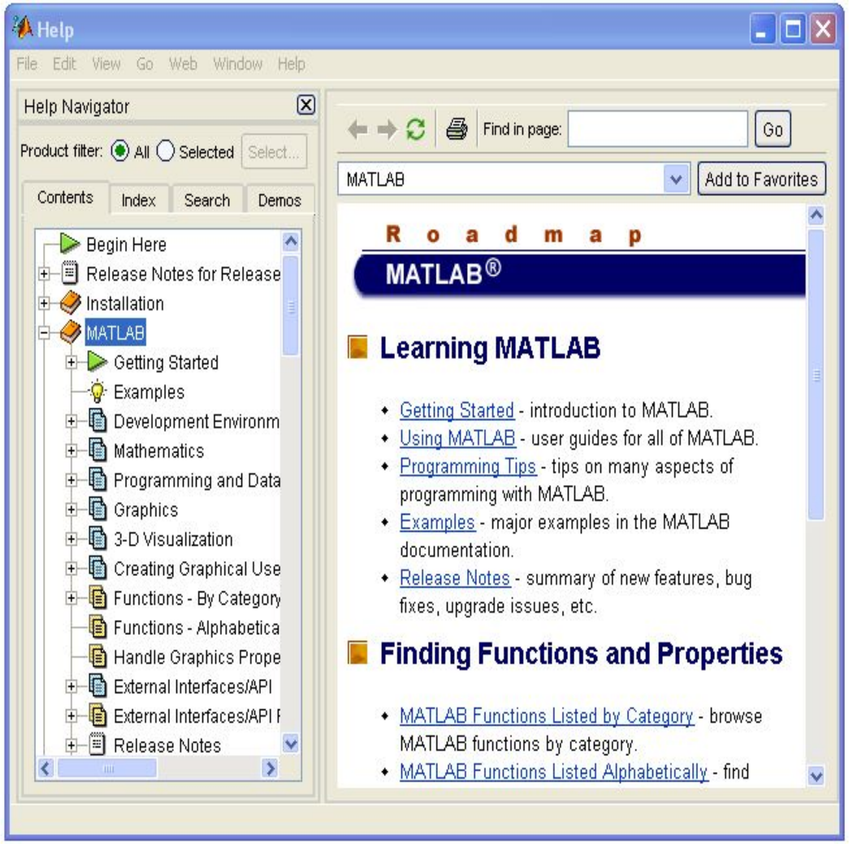

MATLAB
File Edit Debug Desktop Window Help
Current Directory: D:\SmartWork
Shortcuts How to Add What's New
Current Directory - D:\SmartWork
Command Window
Current Directory
Workspace
Command History
Start
OVR

All Files	File Type
BandPassFormat	Folder
BoundaryRefine	Folder
convexhull	Folder
DigitalwaveGuide	Folder
DynamicProgrammingFormant	Folder
filterbank	Folder
heriksRapor	Folder
LDABoundaryRefine	Folder
MAD_ESAT	Folder
nouLDDD	Folder

```
find(DepoD==0)
plot(1./DepoD)
plot(abs(1./DepoD))
plot(log(abs(1./DepoD)))
plot(20*log10(abs(1./DepoD)))
figure,plot(20*log10(abs(1./DepoD))
figure(1),hold on ,plot(20*log10(ab
%-- 27.02.2007 21:15 --%
a=5
b=[1 2]
b=0
b
clc
```

```
>>
```

MATLAB Help



Roadmap

MATLAB®

Learning MATLAB

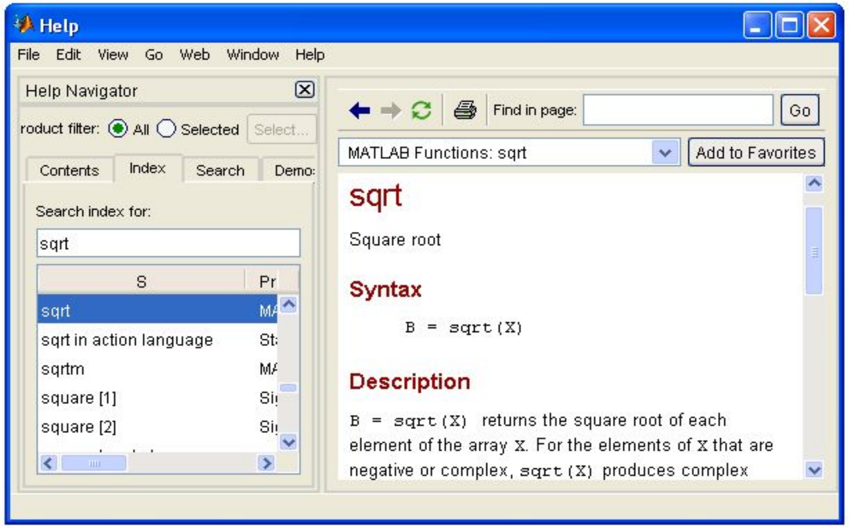
- Getting Started - introduction to MATLAB.
- Using MATLAB - user guides for all of MATLAB.
- Programming Tips - tips on many aspects of programming with MATLAB.
- Examples - major examples in the MATLAB documentation.
- Release Notes - summary of new features, bug fixes, upgrade issues, etc.

Finding Functions and Properties

- MATLAB Functions Listed by Category - browse MATLAB functions by category.
- MATLAB Functions Listed Alphabetically - find

- MATLAB Help is an extremely powerful assistance to learning MATLAB
- Help not only contains the theoretical background, but also shows demos for implementation
- MATLAB Help can be opened by using the HELP pull-down menu

MATLAB Help (cont.)


Help
File Edit View Go Web Window Help
Help Navigator
roduct filter: ● All ○ Selected Select...
Contents Index Search Demo:
Find in page: Go
Add to Favorites
S Pr

Search index for: sqrt

- sqrt
- sqrt in action language
- sqrtm
- square [1]
- square [2]

MATLAB Functions: sqrt

sqrt

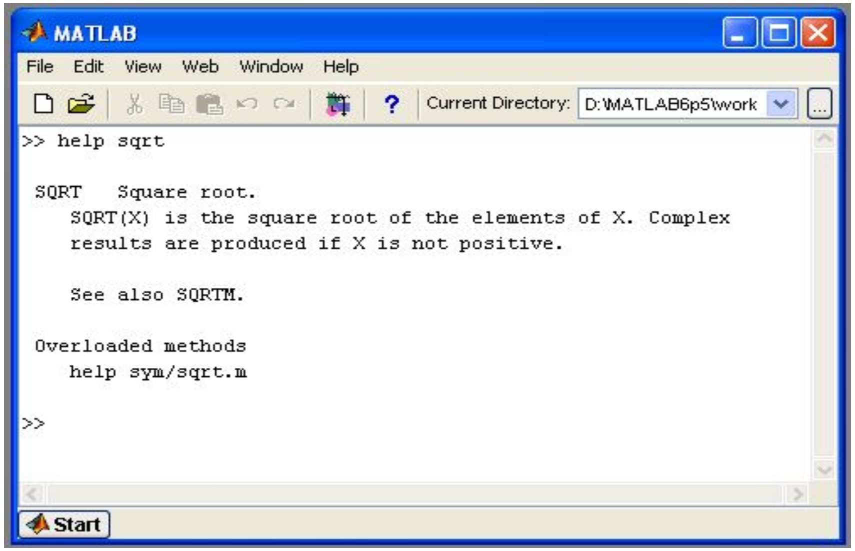
Square root

Syntax

```
B = sqrt(X)
```

Description

`B = sqrt(X)` returns the square root of each element of the array `X`. For the elements of `X` that are negative or complex, `sqrt(X)` produces complex



Current Directory: D:\MATLAB6p5\work

```
>> help sqrt

 SQRT   Square root.
    SQRT(X) is the square root of the elements of X. Complex
    results are produced if X is not positive.

    See also SQRTM.

 Overloaded methods
    help sym/sqrt.m

>>
```

- Any command description can be found by typing the command in the search field
- As shown above, the command to take square root (`sqrt`) is searched
- We can also utilize MATLAB Help from the command window as shown

MATLAB Toolboxes

- Statistics Toolbox
- Optimization Toolbox
- Database Toolbox
- Parallel Computing Toolbox
- Image Processing Toolbox
- Bioinformatics Toolbox
- Fuzzy Logic Toolbox
- Neural Network Toolbox
- Data Acquisition Toolbox
- MATLAB Report Generator
- Signal Processing
- Communications
- System Identification
- Wavelet   Filter Design
- Control System
- Robust Control

Connecting to MATLAB

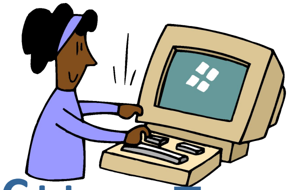
ORACLE

Microsoft Access
The Office XP database solution

Microsoft SQL Server

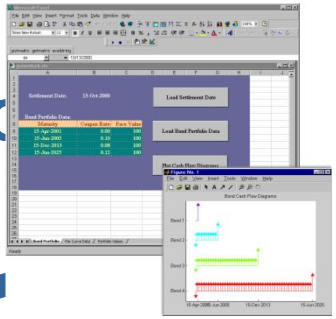
SYBASE

Database Toolbox



C/C++
Java
Perl

Excel / C

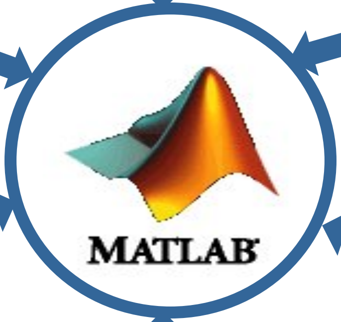


Microsoft Excel
File Edit View Insert Format Tools Data Window Help
putmatrix getmatrix evalstring
B4
10/13/2000

Settlement Date: 13-Oct-2000

Band Portfolio Data:

Maturity	Coupon Rate	Face Value
15-Apr-2001	0.00	100
15-Jun-2005	0.10	100
15-Dec-2013	0.08	100
15-Jun-2025	0.12	100



Web

The MathWorks: Developers of MATLAB and Simulink for Technical Computing - Microsoft Internet Explorer p...
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://www.mathworks.com/
Links Customize Links FAQ Page Free Hotmail Help Page Password Change Reserve
The MathWorks
index | search | contact us
products consulting training/events support store
Site Search: go
Worldwide Offices: USA/Canada
Done
Local intranet

increasing the scope and productivity of engineering and science

MATLAB® the language of technical computing

Simulink® for model-based and system-level design

About The MathWorks
Job openings
Press room
Downloads
Access member login
Newsletters
MATLAB Central

NEW! RELEASE 13

MATLAB & SIMULINK

The new Release 13 CD of the entire MathWorks product family is now available and shipping. The release includes 10 new products and 33 updates.

New products include:

- Aerospace Blockset
- Embedded Target for Motorola® MPC555
- Embedded Target for TI C6000™ DSP
- MATLAB COM Builder
- MATLAB Excel Builder
- MATLAB Link for Code Composer Studio
- SimMechanics for mechanical modeling
- xPC TargetBox™

See more information

News

MATLAB and Simulink now on Mac OS X

The MathWorks Delivers **xPC TargetBox Hardware System** for Desktop Rapid Control Prototyping

Featured Products

- Instrument Control Toolbox
- Data Acquisition Toolbox
- xPC Target for rapid control prototyping
- Real-Time Workshop Embedded Coder for production code

Events

Free Interactive Webinars:

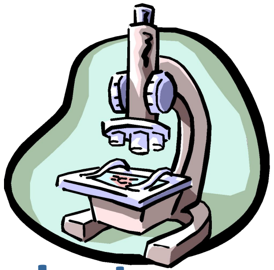
- *Controller Tuning & Plant Modeling*
- *Modeling RF Impairments with the New Communications Blockset*

Seminars:

- *Wireless Communication System Design with MATLAB and Simulink in Eatontown, NJ*

Training:

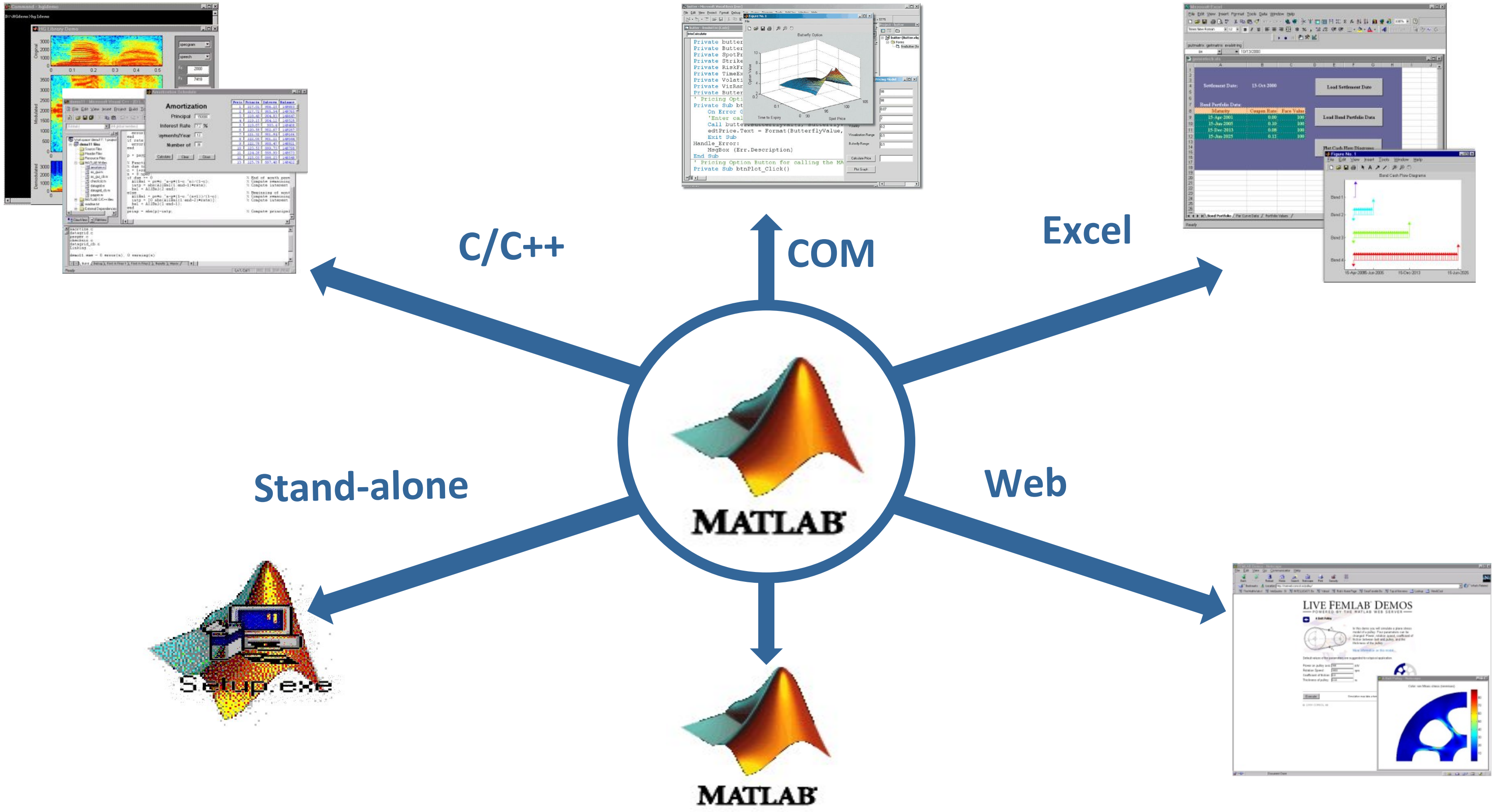
Reserve Your Seat Today - Advanced MATLAB Training in Las Vegas - November 11-14th



Instrument Control
Data Acquisition
Image Acquisition

File I/O

Deploying with MATLAB



C/C++

COM

Excel

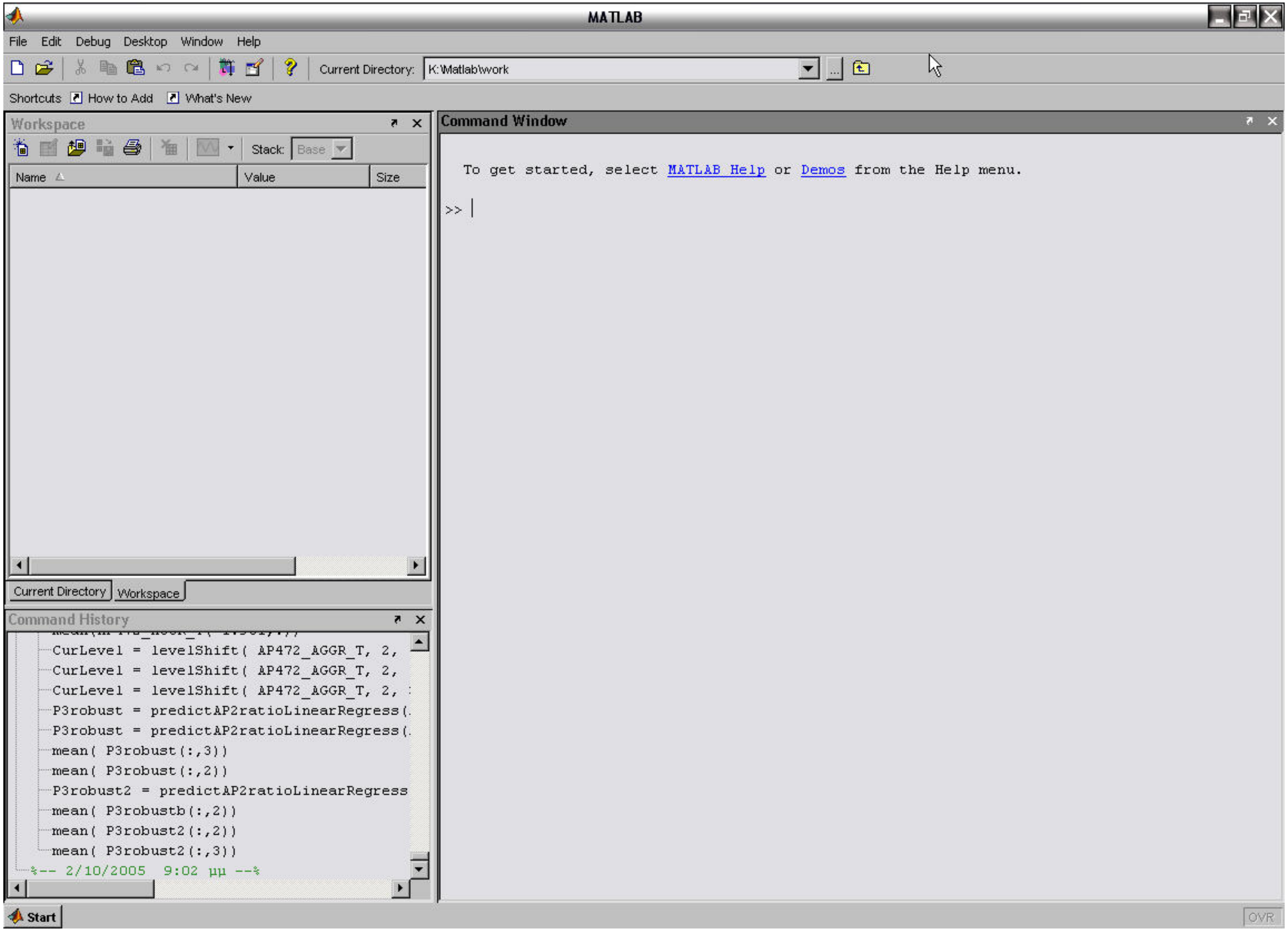
Stand-alone

Web

MATLAB

- 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.

The MATLAB User Interface



```
To get started, select MATLAB Help or Demos from the Help menu.

>> |
```

```
CurLevel = levelShift( AP472_AGGR_T, 2,
CurLevel = levelShift( AP472_AGGR_T, 2,
CurLevel = levelShift( AP472_AGGR_T, 2, :
P3robust = predictAP2ratioLinearRegress(.
P3robust = predictAP2ratioLinearRegress(.
mean( P3robust(:,3))
mean( P3robust(:,2))
P3robust2 = predictAP2ratioLinearRegress
mean( P3robustb(:,2))
mean( P3robust2(:,2))
mean( P3robust2(:,3))
%-- 2/10/2005  9:02 μμ --%
```

More about the Workspace

- `who`, `whos` – current variables in the workspace
- `save` – save workspace variables to *.mat file
- `load` – load variables from *.mat file
- `clear` – clear workspace variables

- CODE

MATLAB

Everything in MATLAB is a matrix !

identifiers

- Identifiers are all the words that build up the program
- An identifier is a sequence of letters, digits and underscores "_"
- Maximal length of identifiers is 63 characters
- Can't start with a digit
- Can't be a reserved word

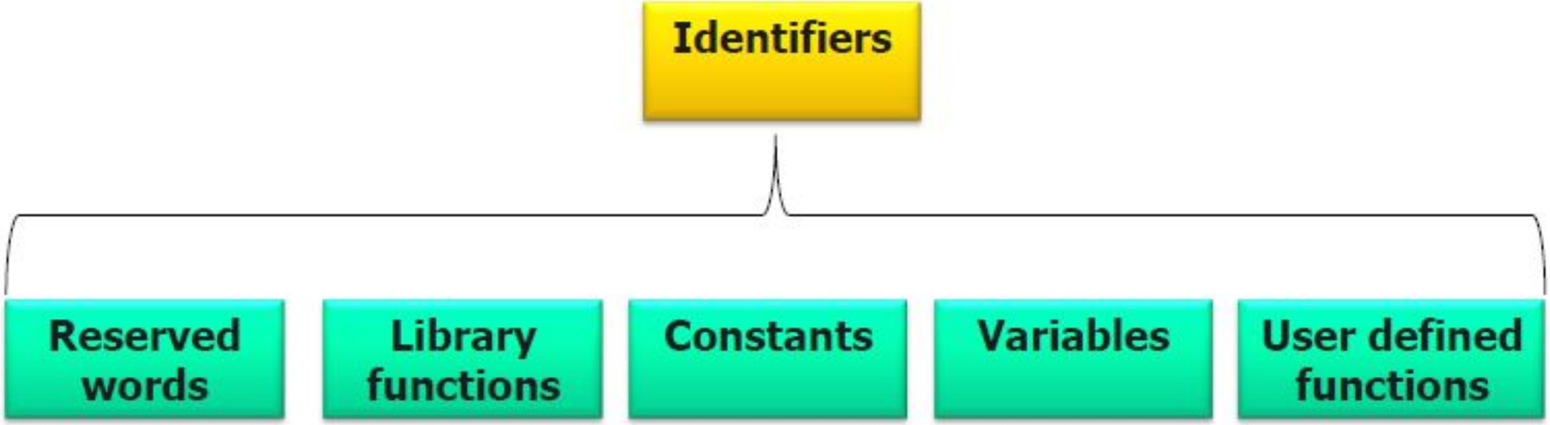
Examples of Legal identifiers:

- time
- day_of_the_week
- bond007
- findWord

Examples of illegal identifiers:

- 007bond
- #time
- ba-baluba
- if
- while

An overview of the main players in a program



Identifiers

Reserved words

Library functions

Constants

Variables

User defined functions

Reserved words (keywords)

- Words that are part of the Matlab language
  - There are 17 reserved words:
    - for
    - function
    - otherwise
    - try
    - break
    - end
    - return
    - switch
    - catch
    - if
    - elseif
    - continue
    - global
    - while
    - case
    - else
    - persistent

- Do **NOT** try to redefine their meaning!
- Don **NOT** try to redefine their library function names either!

Constants

- The value of a constant is fixed and does not change throughout the program

Numbers

100

0.3

Arrays

[1 2 3 4 5]

Matrices

[5 3
4 2]

Chars

‘c’

Strings

‘I like to eat sushi’

‘1 + 2’

Variables

- **Why do we need variables?**

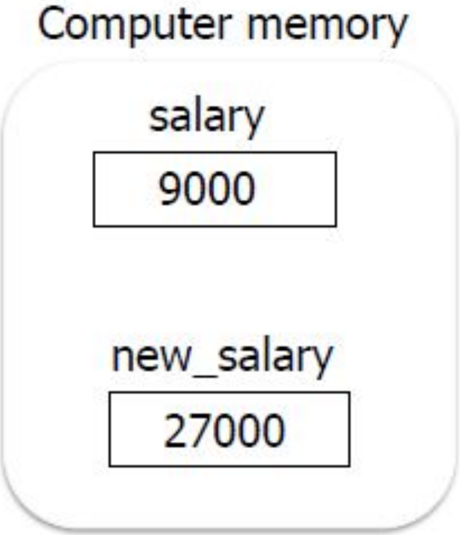
- **Example:**

```
>> salary = 9000;
>> new_salary = salary * 3;
>> disp(new_salary);

   27000
```

constant

variable

Library functions



Computer memory

salary

9000

new_salary

27000

If we update salary, new_salary will NOT be updated automatically

Variables

- **Another example:**

```
price_bamba = 3
```

What happens if you omit the **';' ?**

The Matlab Console

```
price_bamba =
     3
```

Variables

- **Another example:**

```
price_bamba = 3
n_bamba     = 2;
```

What happens when we add the **';'** ?

The Matlab Console

```
price_bamba =
     3
```

Variables

- **Another example:**

```
price_bamba = 3
n_bamba     = 2;
price_bisly = 5
n_bisly     = 3;
```

The Matlab Console

```
price_bamba =
     3

price_bisly =
     5

total_price =
    21

n_bamba =
     5

total_price =
    21
```

How can we fix it?

Redefine total_price

```
total_price = price_bamba * n_bamba + price_bisly * n_bisly
n_bamba     = 5
total_price
```

Variables

- **Tip #1**: Give your variables meaningful names.

```
a  = 9000
b = 100
```

are a bad choice for naming variables that store your working hours and salary!

A more meaningful choice of names would

```
salary = 9000;
hours  = 5;
```

Variables

- **Tip #2**: Don't make variable names too long

```
salary_I_got_for_my_work_at_the_gasoline_station = 9000;

salary_I_got_for_my_work_in_the_bakery = salary_I_got_for_my_work_at_the_gasoline_station * 3;

disp(salary_I_got_for_my_work_in_the_bakery);
```

Very bad choice of variable name!!!

- When should I use capital letters ?
- **Tip #3**: Whatever you do - be consistent.

Variables Types

- Each variable has a *type*
- Why do we need variable types?

- Different types of variable store different types of data

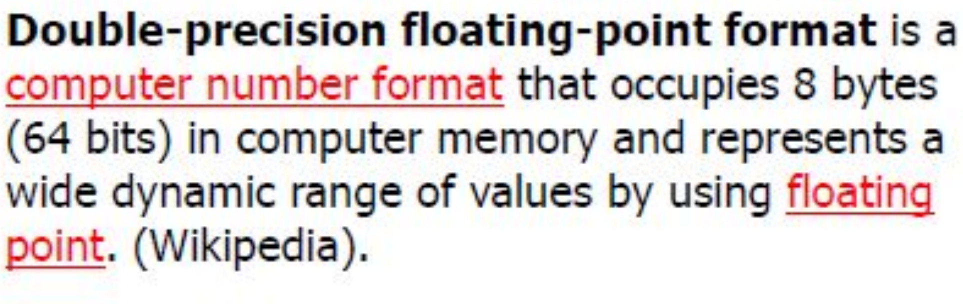
```
>> a = 10
a =
   10

>> class(a)
ans =
double
```

Returns the type of a variable

The default variable type in Matlab is double

Variables Types

- Double



Double-precision floating-point format is a computer number format that occupies 8 bytes (64 bits) in computer memory and represents a wide dynamic range of values by using floating point. (Wikipedia).

- Allows representation of very large numbers (size of a galaxy) to very small numbers (subatomic particles).

```
a = magic(4);
b = single(a);

whos
  Name      Size           Bytes  Class

  a         4x4              128  double array
  b         4x4               64  single array
```

Double-Precision Floating Point

Bits	Usage
63	Sign (0 = positive, 1 = negative)
62 to 52	Exponent, biased by 1023
51 to 0	Fraction f of the number 1.f

Single-Precision Floating Point

Bits	Usage
31	Sign (0 = positive, 1 = negative)
30 to 23	Exponent, biased by 127
22 to 0	Fraction f of the number 1.f

```
x = 25.783;
```

Variables Types

- Each variable has a *type*

- Why do we need variable types?

- Different types of variable store different types of data

```
>> a = 10
a =
   10

>> class(a)
ans =
double
```

```
>> b = 10.56
b =
   10.5600

>> class(b)
ans =
double
```

```
>> c = 'Bush'
c =
Bush

>> class(c)
ans =
char
```

```
>> d = true
d =
     1

>> class(d)
ans =
logical
```

Variables Types

- Different variable types require different memory allocations

```
>> a = 10.4 %double requires 8 bytes
a =
   10.4
```

```
        1                    2                    3                          8
|1|0|0|0|1|1|0|0|   |1|0|1|1|1|0|0|0|   |0|0|0|0|1|0|0|0|  ...   |1|0|0|0|1|0|0|0|
```

```
>> b = 'B'  %char requires 2 bytes
b =
B
```

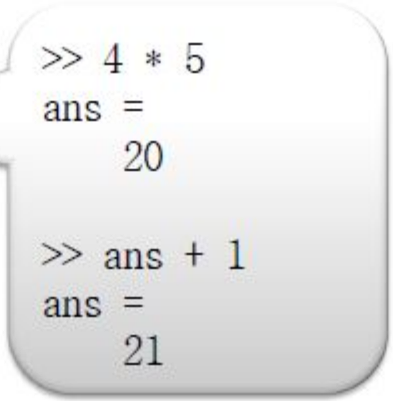
```
         1              2
 |1|0|0|0|1|1|0|0|   |1|0|1|1|1|0|0|0|
```

Memory allocation and release is done automatically in Matlab

- How many bytes are required to store this variable: `c = 'Bush'` ?

Special variables

- ans



```
>> 4 * 5
ans =
    20

>> ans + 1
ans =
    21
```

Special variables

- ans
- pi
- inf

```
>> 2 * inf
ans =
   Inf

>> 1 / 0
Warning: Divide by zero.
ans =
   Inf
```

Special variables

- ans
- pi
- inf
- NaN
- In the tutorial you'll see more…

```
>> 0 / 0
Warning: Divide by zero.
ans =
   NaN

>> NaN + 1
ans =
   NaN
```

Data types

double	**Convert to double precision**
single	**Convert to single precision**
int8	**Convert to 8-bit signed integer**
int16	**Convert to 16-bit signed integer**
int32	**Convert to 32-bit signed integer**
int64	**Convert to 64-bit signed integer**
uint8	**Convert to 8-bit unsigned integer**
uint16	**Convert to 16-bit unsigned integer**
uint32	**Convert to 32-bit unsigned integer**
uint64	**Convert to 64-bit unsigned integer**

Data types

Class	Range of Values	Conversion Function
Signed 8-bit integer	-2^7 to 2^7-1	int8
Signed 16-bit integer	-2^{15} to 2^{15}-1	int16
Signed 32-bit integer	-2^{31} to 2^{31}-1	int32
Signed 64-bit integer	-2^{63} to 2^{63}-1	int64
Unsigned 8-bit integer	0 to 2^8-1	uint8
Unsigned 16-bit integer	0 to 2^{16}-1	uint16
Unsigned 32-bit integer	0 to 2^{32}-1	uint32
Unsigned 64-bit integer	0 to 2^{64}-1	uint64

Starting MATLAB

```
 To get started, type one of these commands:
 helpwin, helpdesk, or demo


» a=5;
» b=a/2

b =

   2.5000

»
```

Variables

- No need for types. i.e.,

~~int a;~~
~~double b;~~
~~float c;~~

- All variables are created with double precision unless specified and they are matrices.

Example:

```
>>x=5;
>>x1=2;
```

- After these statements, the variables are 1x1 matrices with double precision

MATLAB Variable Names

- 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.

MATLAB Special Variables

1.	ans	Default variable name for results
2.	pi	Value of π
3.	eps	Smallest incremental number
4.	inf	Infinity
5.	NaN	Not a number e.g. 0/0
6.	i and j	i = j = square root of -1
7.	realmin	The smallest usable positive real number
8.	realmax	The largest usable positive real number

Different format

```
>> e=1/3
e =
   0.3333                                    %default
>> format long
>> e
e =
  0.333333333333333                          %long decimal
>> format short e
>> e
e =
  3.3333e-001                          %long exponential
```

To clear a variable

```
» who

              Your variables are:

              D     ans    rho
              NRe     mu     v

» clear D
» who

              Your variables are:

              NRe     ans     mu     rho     v

»
```

Complex Numbers

Complex number i or j stands for $\sqrt{-1}$

```
» i
ans =
      0 + 1.0000i

» c1 = 2+3i
c1 =
   2.0000 + 3.0000i
»
```

Complex Numbers

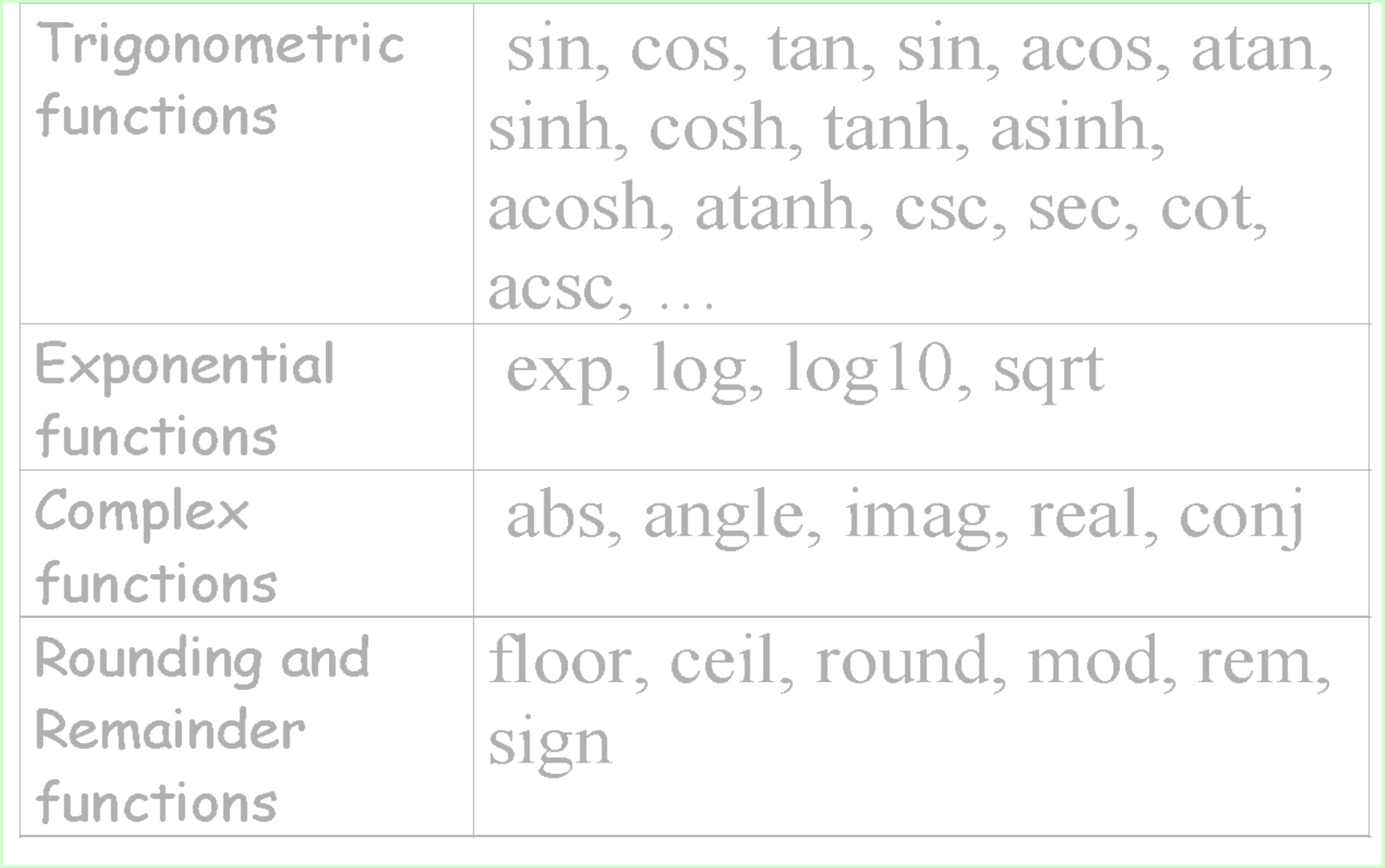
Some functions deal with complex number

```
>> c=1-2i
                    c =   1.0000 - 2.0000i

>> abs(c)
                    ans =    2.2361

>> real(c)
                    ans =     1

>> imag(c)
                    ans = -2

>> angle(c)
                    ans =   -1.1071
```

Mathematical Functions

```
» x=sqrt(2)/2

x =

   0.7071

» y=sin(x)

y =

   0.6496
»
```

Built-in Functions



Trigonometric functions	sin, cos, tan, sin, acos, atan, sinh, cosh, tanh, asinh, acosh, atanh, csc, sec, cot, acsc, …
Exponential functions	exp, log, log10, sqrt
Complex functions	abs, angle, imag, real, conj
Rounding and Remainder functions	floor, ceil, round, mod, rem, sign

Math & Assignment Operators

Power ^ or .^ a^b or a.^b
Multiplication * or .* a*b or a.*b
Division / or ./ a/b or a./b
or \ or .\ b\a or b.\a
NOTE: 56/8 = 8\56

- (unary) + (unary)
Addition + a + b
Subtraction - a - b
Assignment = a = b (assign b to a)

Other MATLAB symbols

>>	prompt
. . .	continue statement on next line
,	separate statements and data
%	start comment which ends at end of line
;	(1) suppress output
	(2) used as a row separator in a matrix
:	specify range

MATLAB Relational Operators

- MATLAB supports six relational operators.

Less Than	<
Less Than or Equal	<=
Greater Than	>
Greater Than or Equal	>=
Equal To	==
Not Equal To	~=

MATLAB Logical Operators

- MATLAB supports three logical operators.

```
not       ~   % highest precedence
and       &  % equal precedence with or
or      |  % equal precedence with and
```

MATLAB Matrices

- MATLAB treats all variables as matrices. For our purposes a matrix can be thought of as an array, in fact, that is how it is stored.

- Vectors are special forms of matrices and contain only one row OR one column.

- Scalars are matrices with only one row AND one column

MATLAB Matrices

- A matrix with only one row AND one column is a scalar. A scalar can be created in MATLAB as follows:

```
» x=23

  x =

  23
```

MATLAB Matrices

- A matrix with only one row is called a row vector. A row vector can be created in MATLAB as follows (note the commas):

```
» rowvec = [12 ,  14 , 63]

rowvec =

   12    14    63
```

MATLAB Matrices

- A matrix with only one column is called a column vector. A column vector can be created in MATLAB as follows (note the semicolons):

```
» colvec = [13 ; 45 ; -2]

colvec =

   13
   45
   -2
```

MATLAB Matrices

- A matrix can be created in MATLAB as follows (note the commas AND semicolons):

```
» a = [1 , 2 , 3 ; 4 , 5  ,6 ; 7 , 8 , 9]        Or
 » a =   [1  2  3 ; 4  5  6 ; 7  8  9]           Or
          >> a=[1 2 3
                4 5 6
               7 8 9]
 a =


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

Extracting a Sub-Matrix

- A portion of a matrix can be extracted and stored in a smaller matrix by specifying the names of both matrices and the rows and columns to extract. The syntax is:

```
sub_matrix = matrix ( r1 : r2 , c1 : c2 ) ;
```

where r1 and r2 specify the beginning and ending rows and c1 and c2 specify the beginning and ending columns to be extracted to make the new matrix.

MATLAB Matrices

- A column vector can be extracted from a matrix. As an example we create a matrix below:

```
» matrix=[1,2,3;4,5,6;7,8,9]

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

- Here we extract column 2 of the matrix and make a column vector:

```
» coltwo=matrix( : , 2)

coltwo =

    2
    5
    8
```

MATLAB Matrices

- A row vector can be extracted from a matrix. As an example we create a matrix below:

```
» matrix=[1,2,3;4,5,6;7,8,9]

matrix =

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

- Here we extract row 2 of the matrix and make a row vector. Note that the 2:2 specifies the second row and the 1:3 specifies which columns of the row.

```
» rowvec=matrix(2 : 2 , 1 : 3)

rowvec =

    4     5     6
```

Special Matrices

$$eye(3)=\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix} \qquad \mathrm{zeros}(3,2)=\begin{bmatrix}0 & 0\\ 0 & 0\\ 0 & 0\end{bmatrix}$$

$$ones(3)=\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} \qquad \mathrm{ones}(2,4)=\begin{bmatrix}1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1\end{bmatrix}$$

Special Matrices functions

```
>> a=magic(4)                         %magic matrix

a =

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

>> b=rand(5)                    %random matrix

b =

    0.8147    0.0975    0.1576    0.1419    0.6557
    0.9058    0.2785    0.9706    0.4218    0.0357
    0.1270    0.5469    0.9572    0.9157    0.8491
    0.9134    0.9575    0.4854    0.7922    0.9340
    0.6324    0.9649    0.8003    0.9595    0.6787
```

Some matrix building functions

```
>> a
a =
    1    2    3
    4    5    6
    7    8    9
```

```
>> diag(a)
ans =
    1
    5
    9
```

```
>> triu(a)
ans =
    1    2    3
    0    5    6
    0    0    9
```

```
>> tril(a)
ans =
    1    0    0
    4    5    0
    7    8    9
```

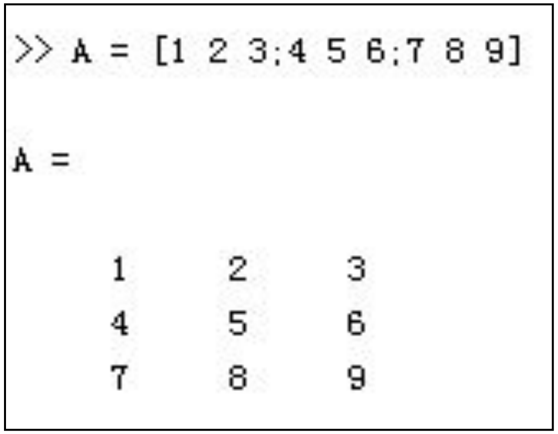
Concatenation of Matrices

- x = [1 2], y = [4 5]

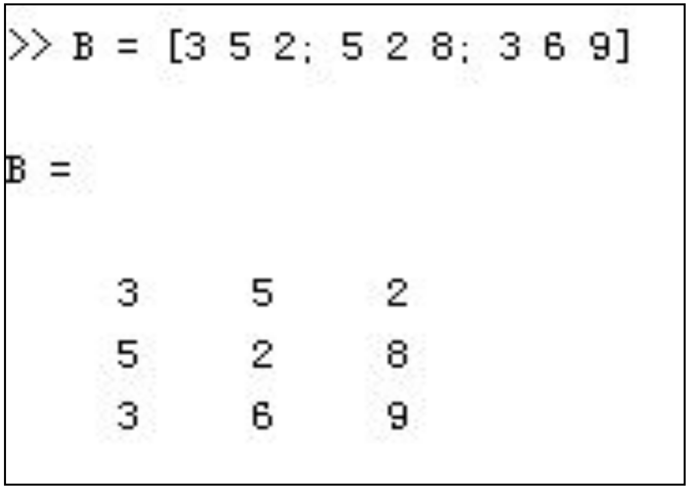
```
  A = [ x y]
     1   2   4   5


B = [x ; y]
    1 2
    4 5
```

Matrices Operations

Given A and B:

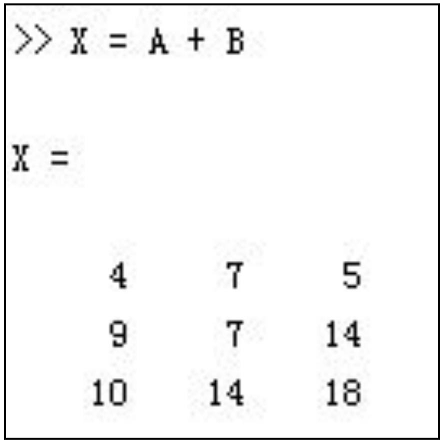


```
>> A = [1 2 3;4 5 6;7 8 9]

A =

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



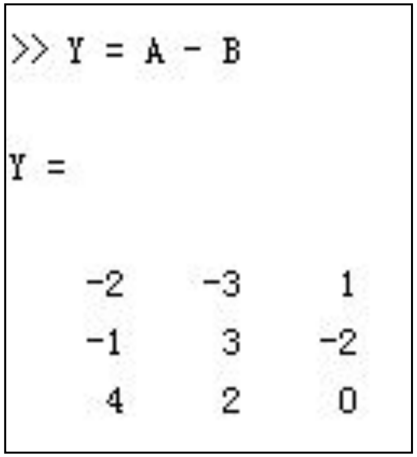
```
>> B = [3 5 2; 5 2 8; 3 6 9]

B =

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

Addition



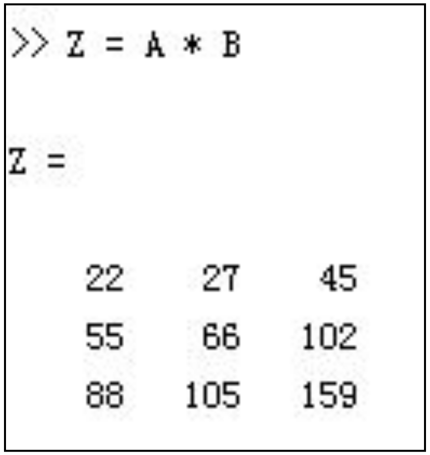
```
>> X = A + B

X =

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

Subtraction



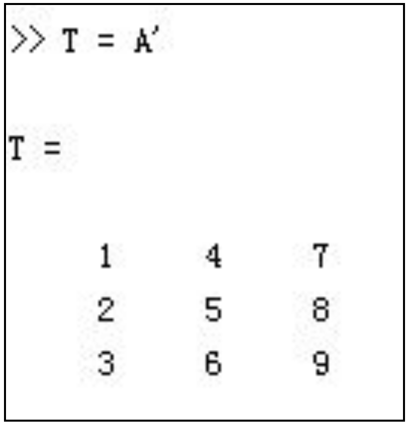
```
>> Y = A - B

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
```

Scalar - Matrix Addition

```
» a=3;
» b=[1, 2, 3;4, 5, 6]
b =
    1     2     3
    4     5     6

» c= b+a   % Add a to each element  of b
c =
    4     5     6
    7     8     9
```

Scalar - Matrix Subtraction

```
» a=3;
» b=[1, 2, 3;4, 5, 6]
b =
    1     2     3
    4     5     6

» c = b - a              %Subtract a from each element of b
c =
   -2    -1     0
    1     2     3
```

Scalar - Matrix Multiplication

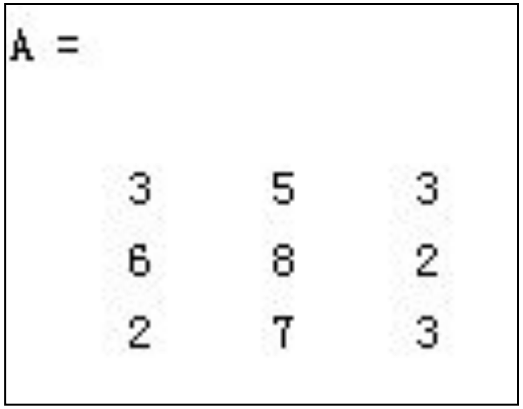
```
» a=3;

» b=[1, 2, 3; 4, 5, 6]
b =
    1    2    3
    4    5    6

» c = a * b            % Multiply each element of b by a
c =
    3    6    9
   12   15   18
```

Scalar - Matrix Division

```
» a=3;
» b=[1, 2, 3; 4, 5, 6]
b =
    1     2     3
    4     5     6

» c = b / a                % Divide each element of b by a
c =
   0.3333    0.6667    1.0000
   1.3333    1.6667    2.0000
```

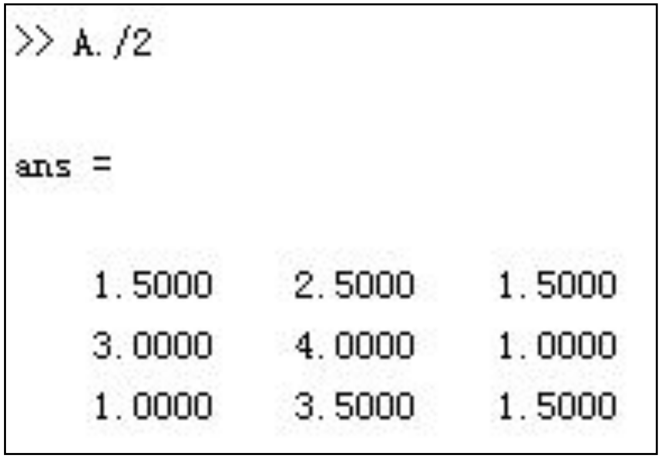
The use of "." – "Element" Operation

Given A:



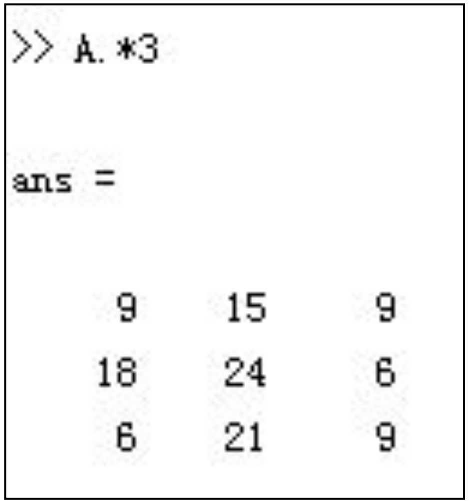
```
A =

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

Divide each element of A by 2



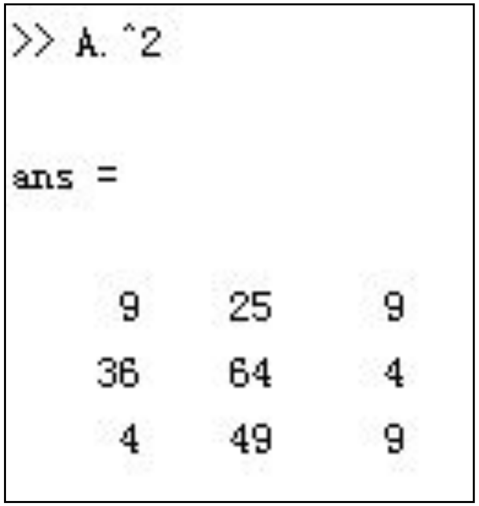
```
>> A./2

ans =

    1.5000    2.5000    1.5000
    3.0000    4.0000    1.0000
    1.0000    3.5000    1.5000
```

Multiply each element of A by 3



```
>> A.*3

ans =

     9    15     9
    18    24     6
     6    21     9
```

Square each element of A



```
>> A.^2

ans =

     9    25     9
    36    64     4
     4    49     9
```

Mean and Median

Mean: Average or mean value of a distribution
Median: Middle value of a sorted distribution

M = mean(A),   M = median(A)
M = mean(A,dim),   M = median(A,dim)

M = mean(A), M = median(A): Returns the mean or median value of vector A.
If A is a multidimensional mean/median returns an array of mean values.

Example:

```
A = [ 0 2 5 7 20]             B = [1 2 3
                                3 3 6
                            4 6 8
                            4 7 7];
```

mean(A) = 6.8
mean(B) = 3.0000 4.5000 6.0000 (column-wise mean)
mean(B,2) = 2.0000 4.0000 6.0000 6.0000 (row-wise mean)

Mean and Median

Examples:

```
A = [ 0 2 5 7 20]                B = [1 2 3
                                3 3 6
                            4 6 8
                            4 7 7];
```

Mean**:**

mean(A) = 6.8

mean(B) = 3.0 4.5 6.0 (column-wise mean)

mean(B,2) = 2.0 4.0 6.0 6.0 (row-wise mean)

Median:

median(A) = 5

median(B) = 3.5 4.5 6.5 (column-wise median)

```
median(B,2) = 2.0
              3.0
              6.0
              7.0 (row-wise median)
```

Standard Deviation and Variance

- Standard deviation is calculated using the std() function
- std(X) : Calcuate the standard deviation of vector x
- If x is a matrix, std() will return the standard deviation of each column
- Variance (defined as the square of the standard deviation) is calculated using the var() function
- var(X) : Calcuate the variance of vector x
- If x is a matrix, var() will return the standard deviation of each column

```
X = [1 5 9;7 15 22]
s = std(X)
s = 4.2426  7.0711   9.1924
```

Histograms

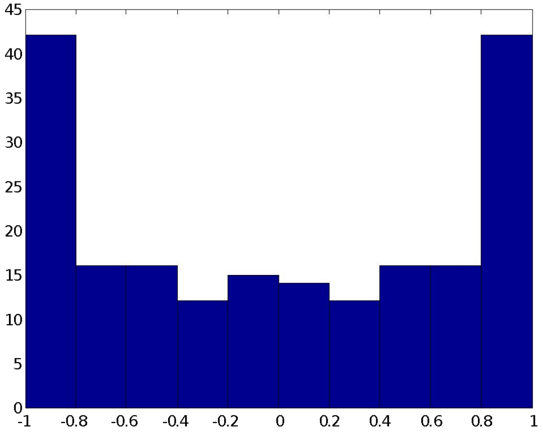
- Histograms are useful for showing the pattern of the whole data set
- Allows the shape of the distribution to be easily visualized

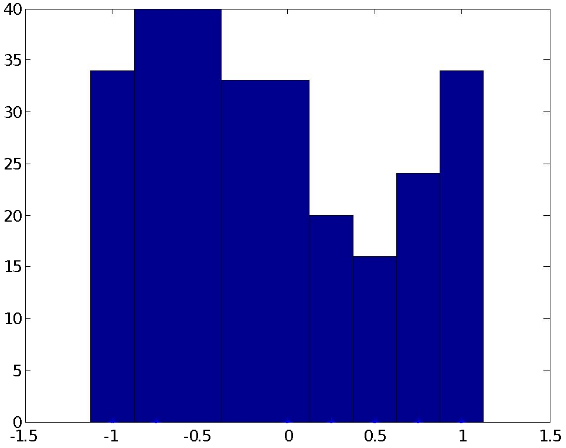
Histograms

- Matlab hist(y,m) command will generate a frequency histogram of vector y distributed among m bins
- Also can use hist(y,x) where x is a vector defining the bin centers

Example:

```
  >>b=sin(2*pi*t)
>>hist(b,10);            >>hist(b,[-1 -0.75 0 0.25 0.5 0.75 1]);
```





Histograms

The histc function is a bit more powerful and allows bin edges to be defined

[n, bin] = histc(x, binrange)

x = statistical distribution

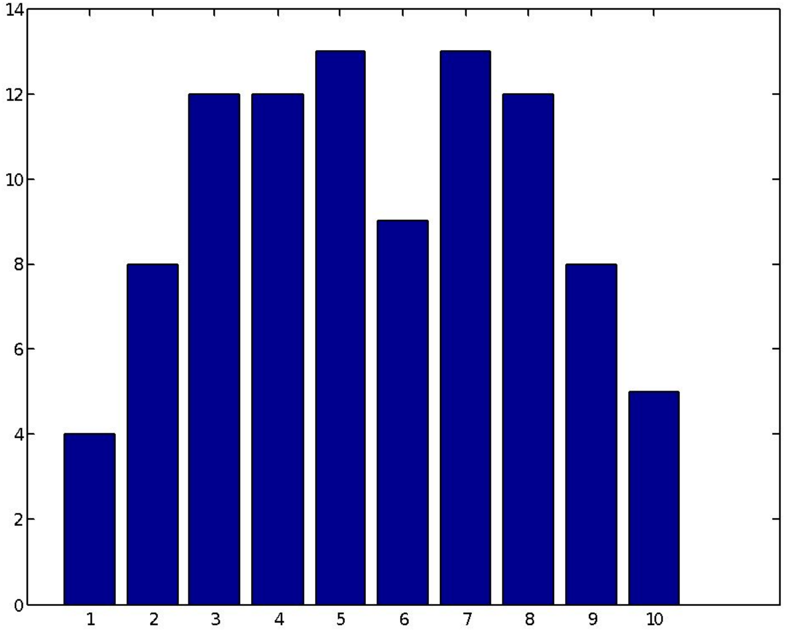
binrange = the range of bins to plot e.g.: [1:1:10]
n = the number of elements in each bin from vector x
bin = the bin number each element of x belongs

- Use the bar function to plot the histogram

Histograms

Example:

```
>> test = round(rand(100,1)*10)
>> histc(test,[1:1:10])
>> Bar(test)
```



Some Useful MATLAB commands

- **who** List known variables
- **whos** List known variables plus their size
- **help** >> help sqrt Help on using sqrt
- **lookfor** >> lookfor sqrt Search for keyword sqrt in m-files
- **what** >> what a: List MATLAB files in a:
- **clear** Clear all variables from work space
- **clear x y** Clear variables x and y from work space
- **clc** Clear the command window

Some Useful MATLAB commands

- what List all m-files in current directory
- dir List all files in current directory
- ls Same as dir
- type test Display test.m in command window
- delete test Delete test.m
- cd a: Change directory to a:
- chdir a: Same as cd
- pwd Show current directory

MATLAB Logical Functions

- **MATLAB also supports some logical functions.**

xor (exclusive or)   Ex: xor (a, b)

Where a and b are logical expressions. The xor operator evaluates to true <u>if and only if</u> one expression is true and the other is false. True is returned as 1, false as 0.

any (x)	**returns 1 if any element of x is nonzero**
all (x)	**returns 1 if all elements of x are nonzero**
isnan (x)	**returns 1 at each NaN in x**
isinf (x)	**returns 1 at each infinity in x**
finite (x)	**returns 1 at each finite value in x**

Input and fprintf

```
>> x=input('please enter a number')
please enter a number100
x =
   100
>> fprintf('%d',x)
100>>
>>
```

Text string,error message

- Text string are entered into matlab surrounded by single quotes
- s=‘this is a text’
- Text string can be displayed with
- disp(‘this is message’)
- Error message are best display with
- error(‘sorry, this is error’)
- error Display message and abort function.
- disp Display array.

Flow Control

- if
- for
- while
- switch case
- break
- ….

Control Structures

- If Statement Syntax

```
if (Condition_1)
        Matlab Commands
elseif (Condition_2)
        Matlab Commands
elseif (Condition_3)
        Matlab Commands
else
        Matlab Commands
end
```

Some Dummy Examples

```
if ((a>3) & (b==5))
    Some Matlab Commands;
end
```

```
if (a<3)
    Some Matlab Commands;
elseif (b~=5)
    Some Matlab Commands;
end
```

```
if (a<3)
    Some Matlab Commands;
else
    Some Matlab Commands;
end
```

Control Structures

- **For loop syntax**

```
for i=Index_Array
    Matlab Commands
end
%..............................
for i=start:inc_value:stop
    Matlab Commands
end
```

Some Dummy Examples

```
for i=1:100
    Some Matlab Commands;
end
```

```
for j=1:3:200
    Some Matlab Commands;
end
```

```
for m=13:-0.2:-21
    Some Matlab Commands;
end
```

```
for k=[0.1 0.3 -13 12 7 -9.3]
    Some Matlab Commands;
end
```

Control Structures

- While Loop Syntax

```
while (condition)
    Matlab Commands
end
```

Dummy Example

```
while  ((a>3) & (b==5))
     Some Matlab Commands;
end
```

switch

- switch – Switch among several cases based on expression
- The general form of SWITCH statement is:

```
switch switch_expr
     case case_expr,
          statement, …, statement
     case {case_expr1, case_expr2, case_expr3, …}
          statement, …, statement
          …
     otherwise
          statement, …, statement
end
```

switch (cont.)

- Note:
  - Only the statements between the matching CASE and the next case, otherwise, or end are executed
  - Unlike C, the switch statement does not fall through (so breaks are unnecessary)

- CODE

Some Examples

```
>> x=20
              x =    20


>> y=30
              y =    30


>> if x>y
'greater x'
else
'greater y'
end


              ans =greater y


>>
```

Some Examples

```
>> for p=1:10
fprintf('%d\t',p)
end
1 2  3    4  5 6    7 8 9 10 >>
```

```
>> for p=1:2:10
fprintf('%d\t',p)
end
1       3       5       7       9       >>
```

Reading Data from files

- MATLAB supports reading an entire file and creating a matrix of the data with one statement.

```
>> load fcmdata.dat;         % loads file into matrix.
          % The matrix may be a scalar, a vector, or a
          %  matrix with multiple rows and columns.  The
          %  matrix will be named mydata.
>> size (fcmdata)          % size will return the number
                           % of rows and number of
                           % columns in the matrix
>> length (fcmdata)          % length will return the total
                             % no. of elements in myvector
```

M-Files

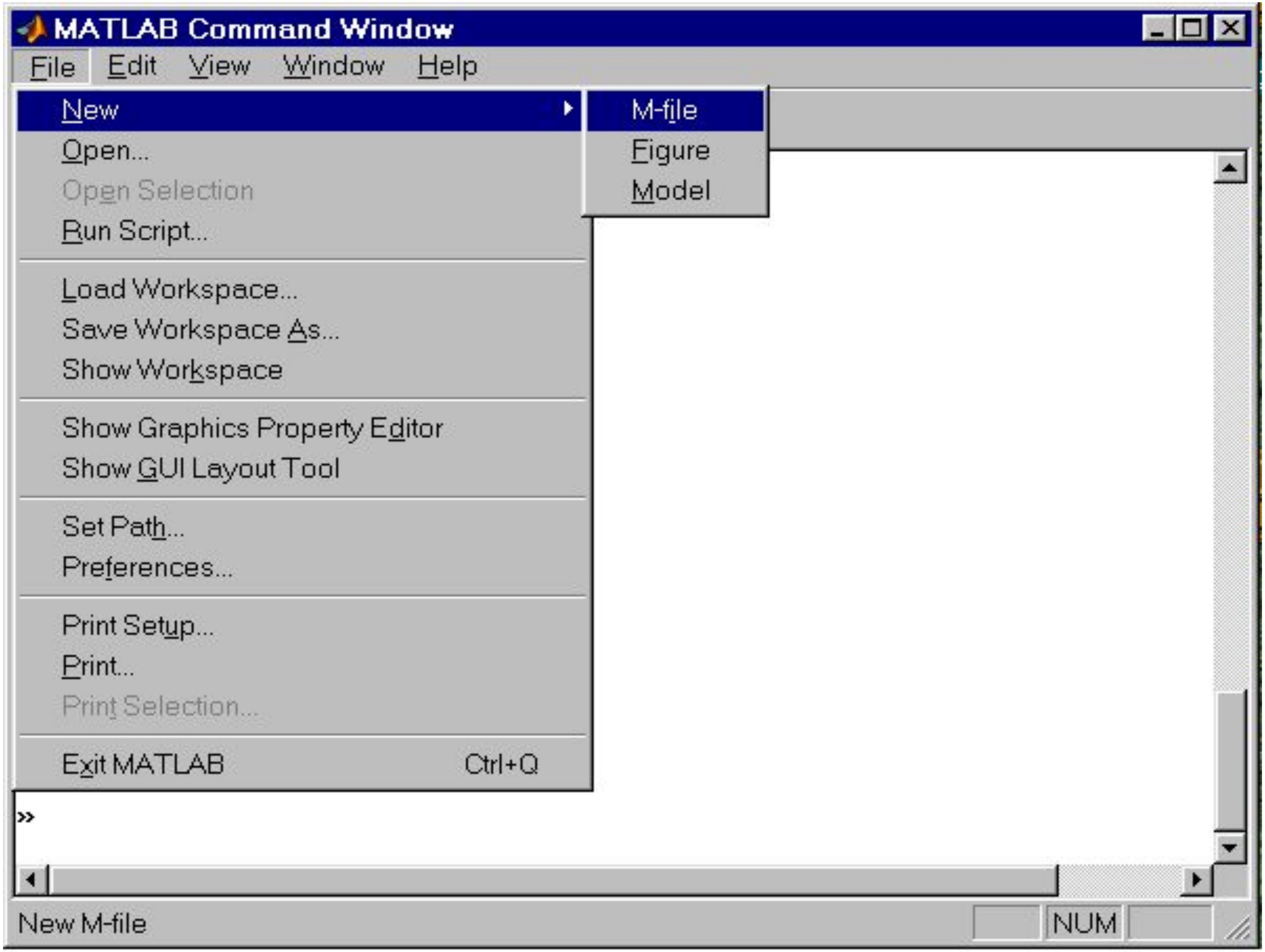
- Script file: a collection of MATLAB commands

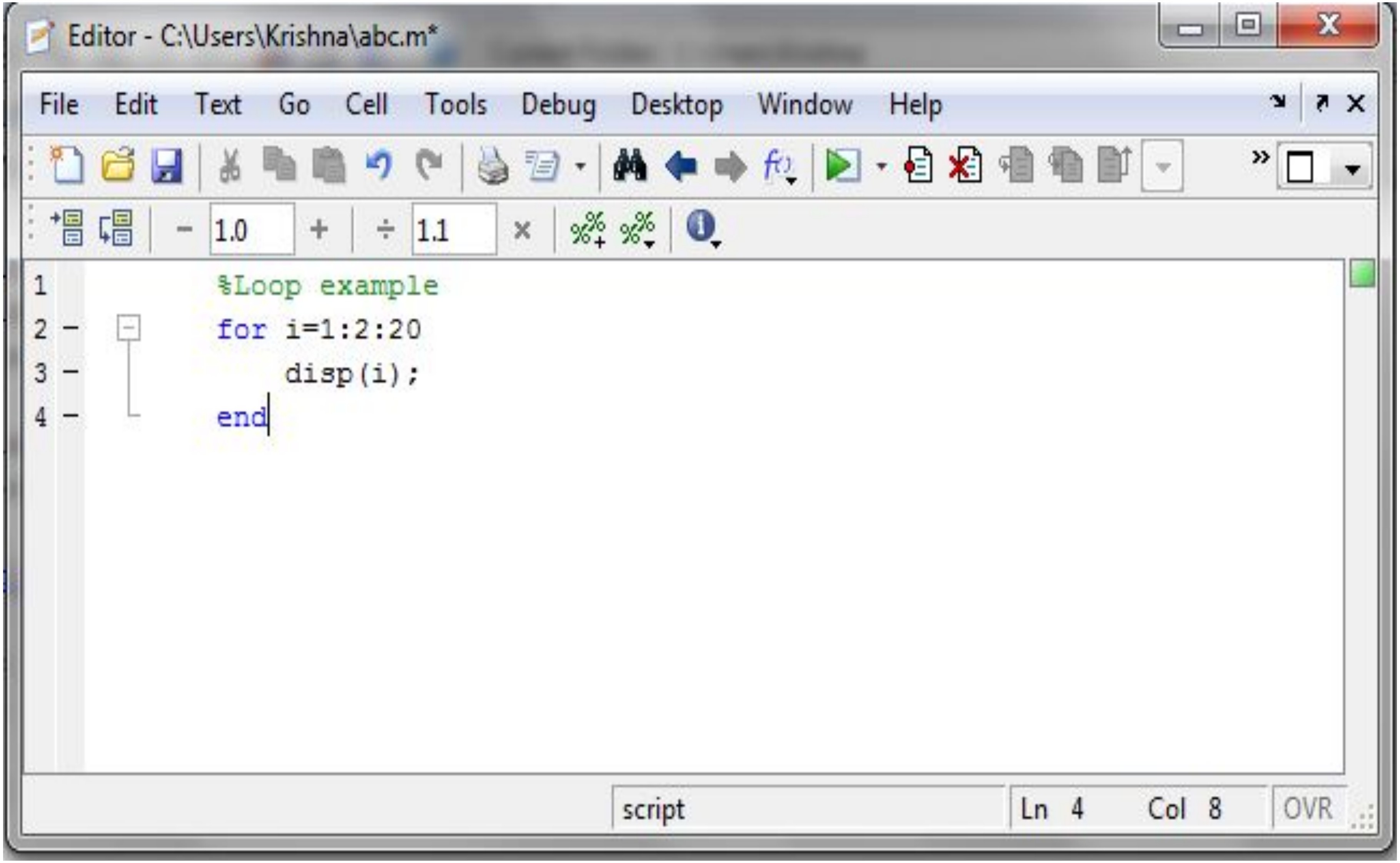
- Function file: a definition file for one function

Script Files

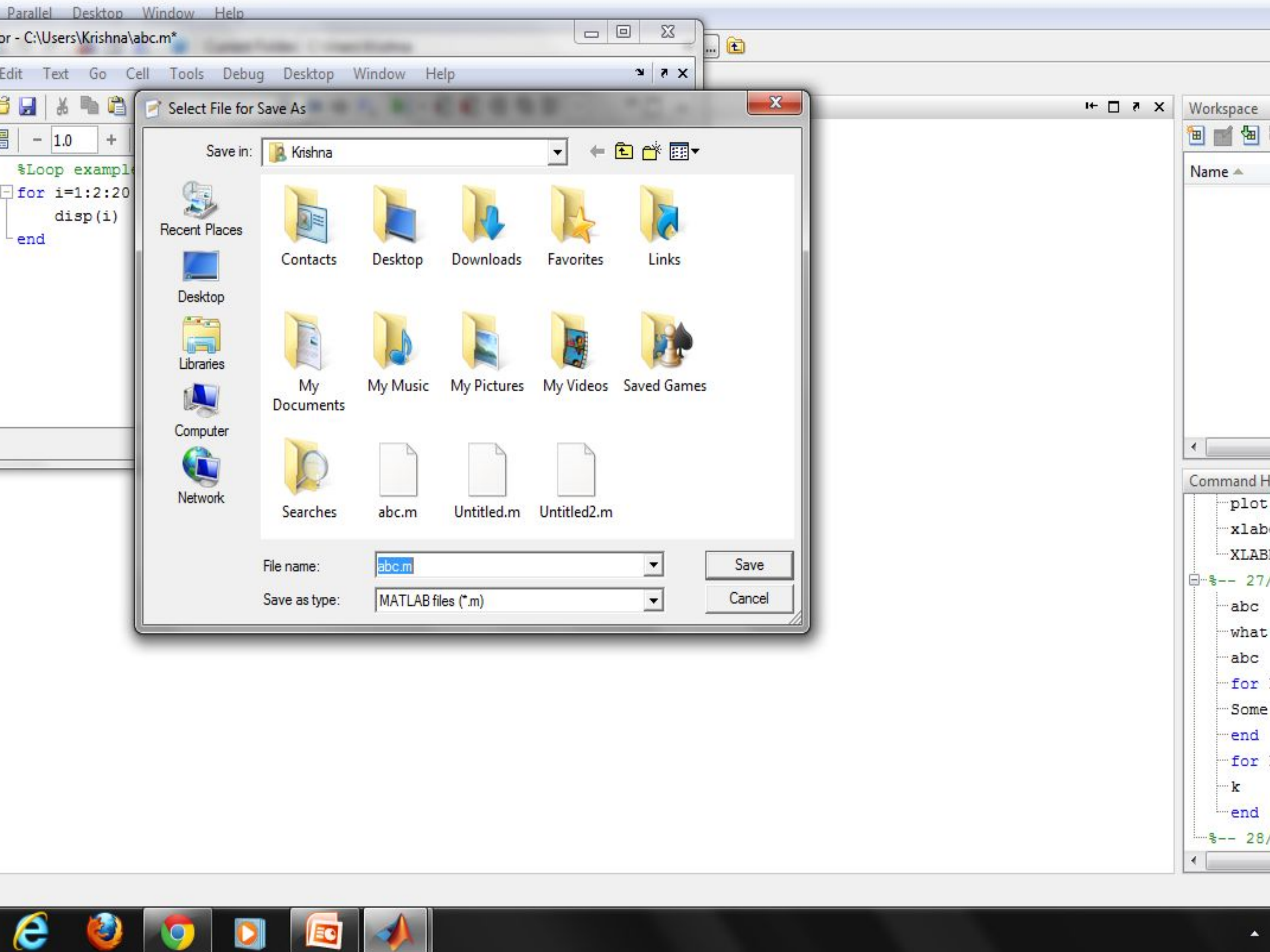
- **Any valid sequence of MATLAB commands can be in the script files.**

- **Variables defined/used in script files are global, i.e., they present in the workspace.**


MATLAB Command Window
File
Edit
View
Window
Help
New
Open...
Open Selection
Run Script...
Load Workspace...
Save Workspace As...
Show Workspace
Show Graphics Property Editor
Show GUI Layout Tool
Set Path...
Preferences...
Print Setup...
Print...
Print Selection...
Exit MATLAB
Ctrl+Q
M-file
Figure
Model
>>
New M-file
NUM


Editor - C:\Users\Krishna\abc.m*
File Edit Text Go Cell Tools Debug Desktop Window Help
1.0
1.1
script
Ln 4 Col 8
OVR

```
%Loop example
for i=1:2:20
    disp(i);
end
```



or - C:\Users\Krishna\abc.m*

```
%Loop exampl
for i=1:2:20
    disp(i)
end
```

Select File for Save As

Save in: Krishna

File name: abc.m

Save as type: MATLAB files (*.m)

```
plot
xlab
XLAB
%-- 27/
abc
what
abc
for
Some
end
for
k
end
%-- 28/
```

Using Script M-files

```
» what

M-files in the current directory
C:\WINDOWS\Desktop\Matlab-Tutorials

abc abc1

» abc
1
3
5
.
.
.
```

File Name

M-file Example

%test.m

```
for i=1:5
    for j=1:i
        fprintf('*');
    end
    fprintf('\n');
end
```

%output

```
>> test
*
**
***
****
*****
```

Writing User Defined Functions

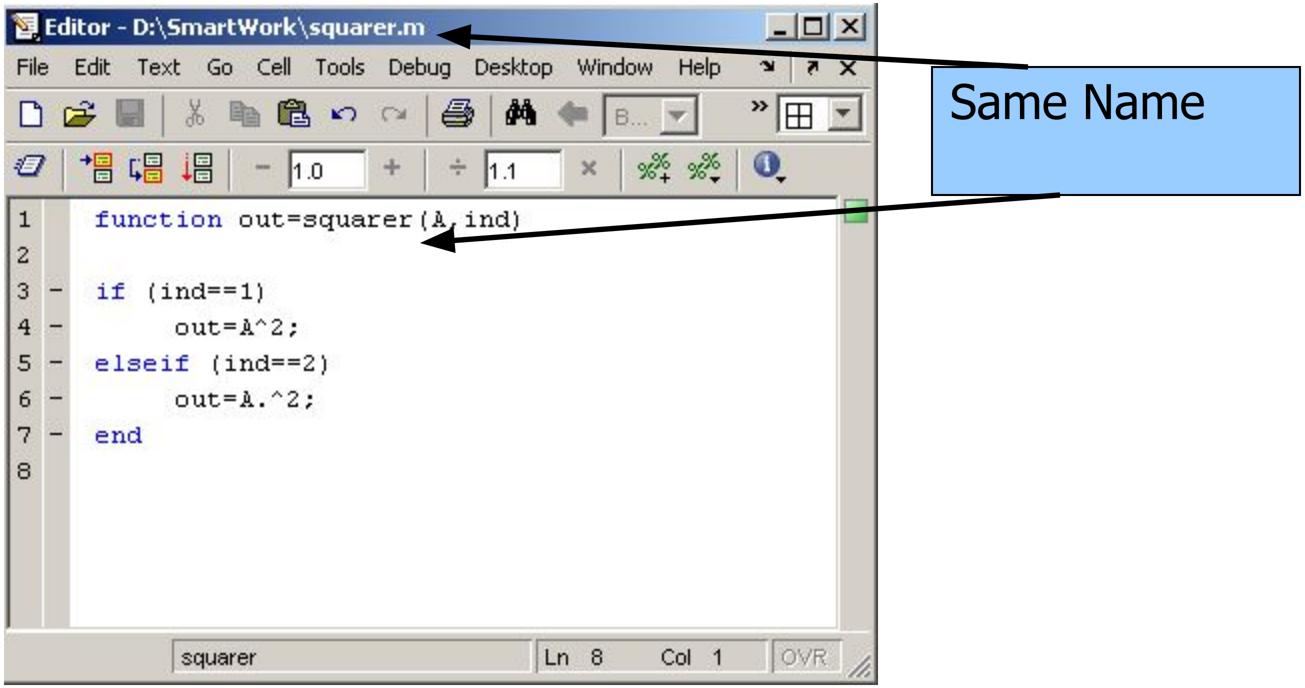
- Functions are m-files which can be executed by specifying some inputs and supply some desired outputs.
- The code telling the Matlab that an m-file is actually a function is

```
function out1=functionname(in1)
function out1=functionname(in1,in2,in3)
function [out1,out2]=functionname(in1,in2)
```

- You should write this command at the beginning of the m-file and you should save the m-file with a file name same as the function name

Writing User Defined Functions

- Examples
  - Write a function : out=squarer (A, ind)
    - Which takes the square of the input matrix if the input indicator is equal to 1
    - And takes the element by element square of the input matrix if the input indicator is equal to 2



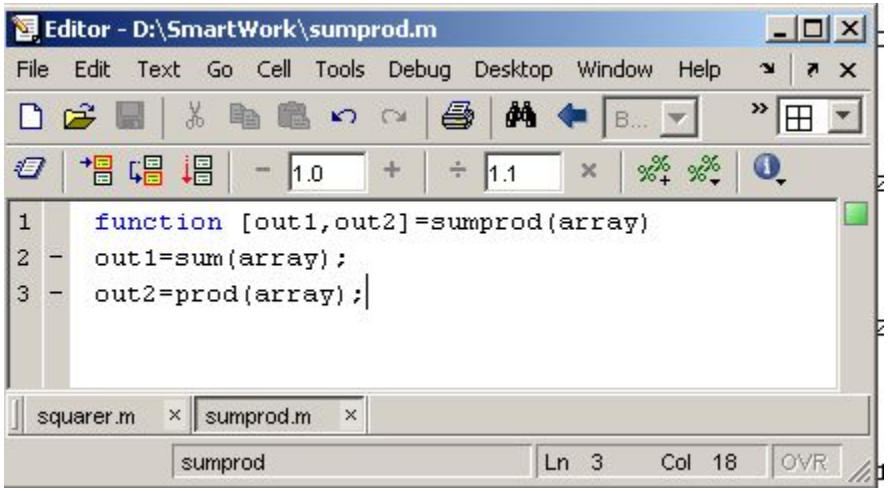
Editor - D:\SmartWork\squarer.m
File Edit Text Go Cell Tools Debug Desktop Window Help
1.0
1.1
squarer
Ln 8 Col 1
OVR

```
function out=squarer(A,ind)

if (ind==1)
        out=A^2;
elseif (ind==2)
        out=A.^2;
end

```

Same Name

Writing User Defined Functions

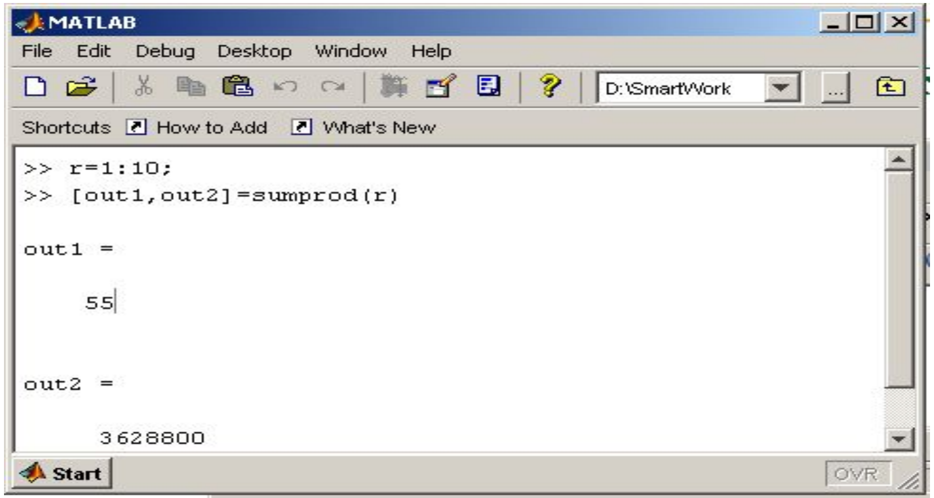
- Another function which takes an input array and returns the sum and product of its elements as outputs



Editor - D:\SmartWork\sumprod.m
File Edit Text Go Cell Tools Debug Desktop Window Help
1.0
1.1
squarer.m
sumprod.m
sumprod
Ln 3
Col 18
OVR

```
function [out1,out2]=sumprod(array)
out1=sum(array);
out2=prod(array);
```

- The function sumprod(.) can be called from command window or an m-file as



```
>> r=1:10;
>> [out1,out2]=sumprod(r)

out1 =

    55


out2 =

     3628800
```

Function Example1

multiply.m

```
function y = multiply(a,b)
y=a*b;
 end
```

output

```
>> multiply(23,3)

ans =

    69
```

Function Example2

function1.m

```
function [out1,out2] = function1(a,b)
out1=sin(a);
out2=sin(b);
end
```

output

```
[a,b]=function1(2,4)
a =
    0.9093

b =
   -0.7568
```

Commands for Navigating Folders

```
pwd
```

displays current folder

```
cd c:/menke/docs/eda/ch01
```

change to a folder in a specific place

```
cd ..
```

change to the parent folder

```
cd ch01
```

change to the named folder that within the current one

```
dir
```

display all the files and folders in the current folder

Vectors and Matrices

$$\mathbf{r} = \begin{bmatrix} 2 & 4 & 6 \end{bmatrix} \text{ and } \mathbf{c} = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 5 \end{bmatrix}^T \text{ and } \mathbf{M} = \begin{bmatrix} 1 & 2 & 4 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

```
r = [2, 4, 6];
c = [1, 3, 5]’;
M =[ [1, 4, 7]', [2, 5, 8]', [3, 6,
9]'];
```

Transpose Operator

Swap rows and columns of an array, so that

$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$ becomes [1, 2, 3, 4] (and vice versa)

Standard mathematical notation: $\mathbf{a}^{\mathrm{T}}$

***MatLab* notation: $\mathbf{a}'$**

Vector Multiplication

Let’s define some vectors and matrices

$$\mathbf{a} = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} \text{ and } \mathbf{b} = \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix} \text{ and } \mathbf{M} = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 0 \\ 2 & 0 & 1 \end{bmatrix} \text{ and } \mathbf{N} = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 2 & 0 \\ -1 & 0 & 3 \end{bmatrix}$$

```
a = [1, 3, 5]’;
c = [3, 4, 5]’;
M =[ [1, 0, 2]', [0, 1, 0]', [2, 0,
1]'];
N =[ [1, 0,-1]', [0, 2, 0]', [-1,0,
3]'];
```

Inner (or Dot) Product

$$s = \mathbf{a}^{\mathbf{T}}\mathbf{b} = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix}^{T} \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 5 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix} = 2\times1 + 3\times4 + 5\times6 = 44$$

```
s = a'*b;
```

Outer (or Tensor) Product

$$\mathbf{T} = \mathbf{a}\mathbf{b}^{\mathbf{T}} = \begin{bmatrix}1\\3\\5\end{bmatrix}\begin{bmatrix}2\\4\\6\end{bmatrix}^{T} = \begin{bmatrix}2\times1 & 4\times1 & 6\times1\\2\times3 & 4\times3 & 6\times3\\2\times5 & 4\times5 & 6\times5\end{bmatrix} = \begin{bmatrix}2 & 4 & 6\\6 & 12 & 18\\10 & 20 & 30\end{bmatrix}$$

```
T = a*b’;
```

Product of a Matrix and a Vector

$$\mathbf{c} = \mathbf{Ma} = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 0 \\ 2 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} = \begin{bmatrix} 1\times1 + \ 0\times3 + \ 2\times5 \\ 0\times1 + 1\times3 + \ 0\times5 \\ 2\times1 + \ 0\times3 + \ 1\times5 \end{bmatrix} = \begin{bmatrix} 11 \\ 3 \\ 7 \end{bmatrix}$$

```
c = M*a;
```

Product of a Matrix and a Matrix

$$\mathbf{P} = \mathbf{MN} = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 0 \\ 2 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \\ 0 & 2 & 0 \\ -1 & 0 & 3 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 5 \\ 0 & 2 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

```
P = M*N;
```

Element Access

$$\mathbf{a} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \text{ and } \mathbf{M} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

$$s = a_2 = 2 \quad \text{and} \quad t = M_{23} = 6 \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} M_{12} \\ M_{22} \\ M_{32} \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \\ 8 \end{bmatrix}$$

```
s = a(2);
t = M(2,3);
b = M(:,2);
```

Element Access

$$\mathbf{a} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \text{ and } \mathbf{M} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

$$\mathbf{c} = \begin{bmatrix} M_{21} & M_{22} & M_{23} \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} \text{ and } \mathbf{T} = \begin{bmatrix} M_{22} & M_{23} \\ M_{32} & M_{33} \end{bmatrix} = \begin{bmatrix} 5 & 6 \\ 8 & 9 \end{bmatrix}$$

```
c = M(2,:)';
T = M(2:3,2:3);
```

Another Example of a FOR Loop

swap the columns of $\mathbf{M} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$ to form $\mathbf{N} = \begin{bmatrix} 3 & 2 & 1 \\ 6 & 5 & 4 \\ 9 & 8 & 7 \end{bmatrix}$

without looping

```
N = fliplr(M);
```

with looping

```
for i = [1:3]
for j = [1:3]
N(i,4-j) = M(i,j);
end
end
```

Matrix Inverse

$$\mathbf{A}^{-1}\,\mathbf{A} = \mathbf{A}\,\mathbf{A}^{-1} = \mathbf{I}$$

```
B = inv(A);
```

Slash and Backslash Operators

$$\mathbf{c} = \mathbf{A}^{-1}\,\mathbf{b} \quad \text{and} \quad \mathbf{D} = \mathbf{B}\,\mathbf{A}^{-1}$$

```
c = A\b;
D = B/A;
```

Loading Data Files

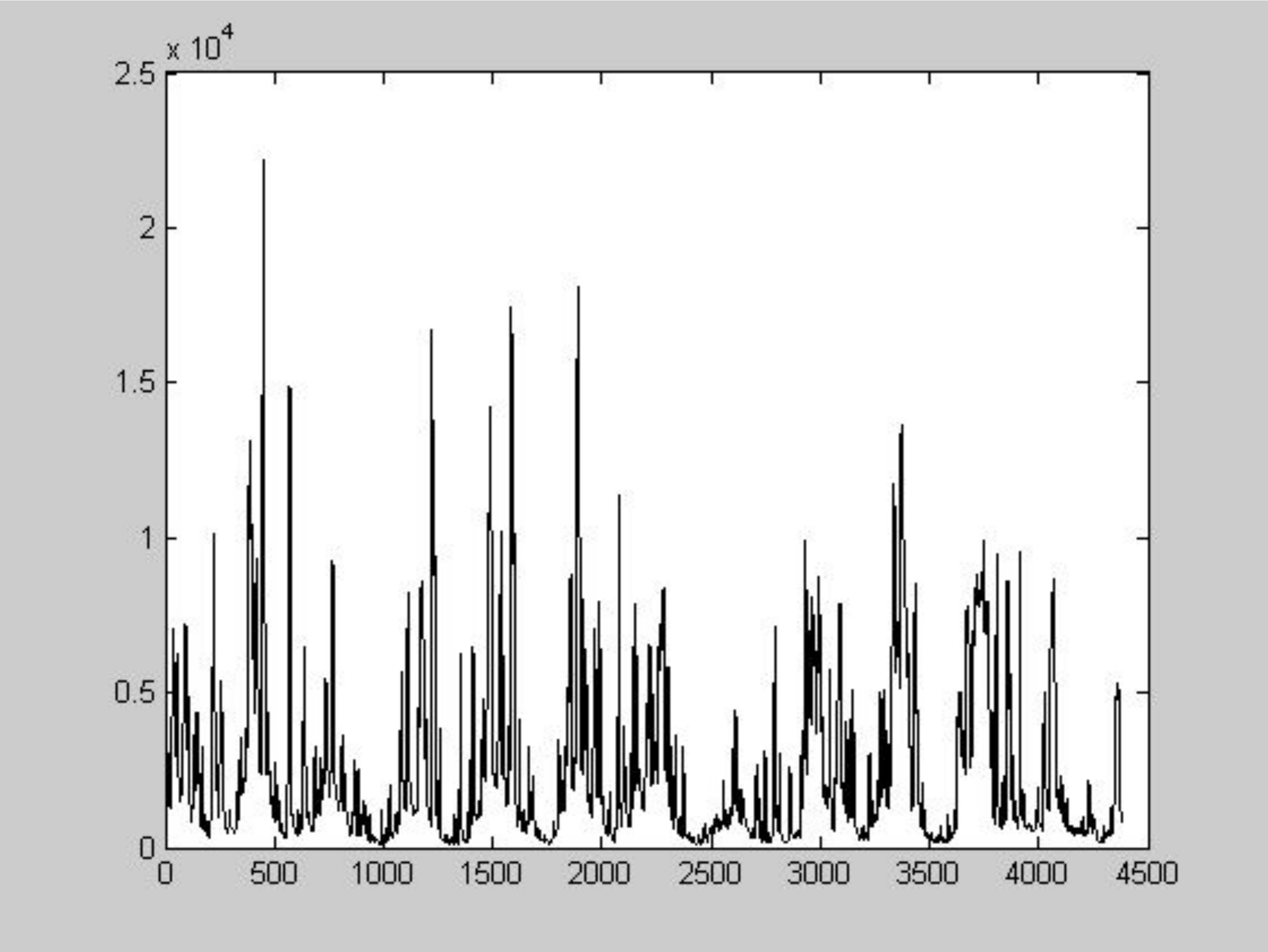
A data file, neuse.txt is used. It contains two columns of data, time (in days starting on January 1, 1974) and discharge (in cubic feet per second, cfs). The data set contains 4383 rows of data. The information about the data is saved in the file neuse_header.txt.

A text file of tabular data is very easy to load into MatLab

```
D = load(‘neuse.txt’);
t = D(:,1);
d = D(:,2);
```

A Simple Plot of Data

```
plot(t,d);
```



A Somewhat Better Controlled Plot

```
set(gca,'LineWidth',2);
```

make the axes thicker

```
plot(t,d,'k-','LineWidth',2);
```

plot black lines of width 2

```
title('Neuse River Hydrograph');
```

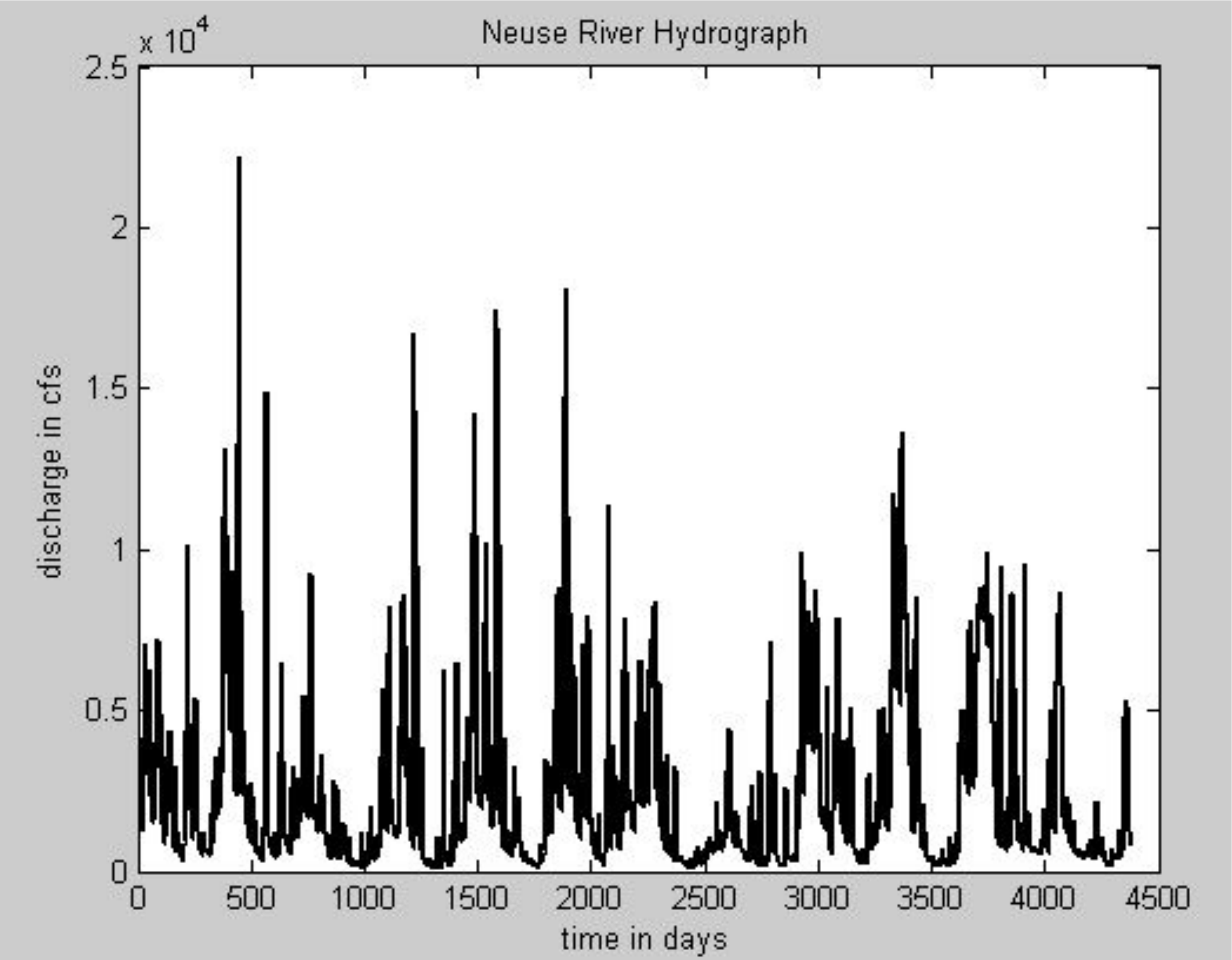
title at top of figure

```
xlabel('time in days');
```

label x axis

```
ylabel('discharge in cfs');
```

label y axis


Neuse River Hydrograph
x 10^4
2.5
2
1.5
1
0.5
0
discharge in cfs
0
500
1000
1500
2000
2500
3000
3500
4000
4500
time in days

Writing a Data File

example: convert cfs to m^3

```
f=35.3146;
dm = d/f;
Dm(:,1)=t;
Dm(:,2)=dm;
dlmwrite(‘neuse_metric.txt’,Dm,’\t’);
%dlmwrite Write ASCII delimited file.
%dlmwrite('FILENAME',M,'DLM') writes
  %matrix M into FILENAME using the
  %character DLM as the delimiter.
```

Finding and Using Documentation

Some other basic data structures in Matlab

cell arrays
structures
containers.Map
Summary

- A cell array is a general purpose matrix

- Each of the elements can contain data of a different type, size and dimension

- Cell arrays are created using the **cell** command or by using curly braces

- `>> cell_name{row,col} = data;`

- Storage is allocated dynamically

- cellplot shows a graphical depiction of a cell array

Cell Array

. Here, we might store the following data in a variable to describe the **Antoine coefficients** for benzene and the range they are relevant for **[Tmin Tmax]**

```
c = {'benzene' 6.9056 1211.0 220.79 [-16 104]}

c =
    'benzene'    [6.9056]    [1211]    [220.7900]    [1x2 double]
```

To access the elements of a cell array use curly brackets for indexing.

```
c{1}
 ans = benzene
```

Cell Array

We can also index the cell array, e.g. to get elements 2-4:

```
[A B C] = c{2:4}

A =
   6.9056

B =
       1211
C =

  220.7900
```

Cell Array

If you want to extract all the contents to variable names that are easy to read, use this syntax:

```
[name A B C Trange] = c{:}
```

```
name = benzene
A = 6.9056
B = 1211
C = 220.7900
Trange =

   -16   104
```

Structures

a structure contains named fields that can contain a variety of data types. Structures are often used to set options

```
s = struct('name','benzene','A',6.9056,'B',1211.0')
```

```
 s = name: 'benzene'
     A: 6.9056
     B: 1211
```

And we can add fields like this:

```
s.C = 220.79
s.Trange = [-16 104]
```

structures

```
s =
    name: 'benzene'
       A: 6.9056
       B: 1211
       C: 220.7900

s =
      name: 'benzene'
         A: 6.9056
         B: 1211
         C: 220.7900
    Trange: [-16 104]
```

structures

we can access the data in a struct by the field

```
s.name
s.Trange
```

```
ans = benzene
ans = -16 104
```

```
fieldnames(s)
```

```
ans = 'name'
      'A'
      'B'
      'C'
      'Trange'
```

containers.Map

A container.Map is like a dictionary, with a **key:value** relationship. You can use complicated key strings including spaces. By default, all keys must be the same type, e.g. all strings.

```
cM = containers.Map();
 cM('name') = 'benzene';
 cM('A') = 6.9056;
cM('B') = 1211.0;
cM('C') = 220.79;
 cM('Trange') = [-16 104];
cM('key with spaces') = 'random thoughts';
```

structures

and we can access the data in a map by key:

```
cM('name') cM('key with spaces')

ans =
     benzene

ans =
     random thoughts
```

Creating a Cell Array

```
emptyCell = cell(3,4,2)
emptyCell(:,:,1) =
    []    []    []    []
    []    []    []    []
    []    []    []    []

emptyCell(:,:,2) =

    []    []    []    []
    []    []    []    []
    []    []    []    []
```

Access Data in a Cell Array

```
C = {'one', 'two', 'three';
     1, 2, 3};

upperLeft = C(1:2,1:2)
upperLeft =
    'one'    'two'
    [  1]    [  2]
```

```
C(1,1:3) = {'first','second','third'}
```

replaces the cells in the first row of C with an equivalent-sized (1-by-3) cell array:

```
C = 'first' 'second' 'third' [ 1] [ 2] [ 3]
```

If cells an array contain numeric data, it can be converted to the cells to a numeric array using the cell2mat function:

```
numericCells = C(2,1:3)
numericVector = cell2mat(numericCells)
```

numericCells is a 1-by-3 cell array, but numericVector is a 1-by-3 array of type double:

```
numericCells = [1] [2] [3]
numericVector = 1 2 3
```

```
last = C{2,3}
```

creates a numeric variable of type double, because the cell contains a double value:

```
last = 3
```

Similarly, this command

```
C{2,3} = 300
```

replaces the contents of the last cell of C with a new, numeric value:

```
C = 'first' 'second' 'third' [ 1] [ 2] [ 300]
```

```
[r1c1, r2c1, r1c2, r2c2] = C{1:2,1:2}
```

returns

```
r1c1 = first
 r2c1 = 1
 r1c2 = second
 r2c2 = 2
```

Similarly,

```
nums = [C{2,:}]
```

returns

```
nums = 1 2 300
```

TRY

```
A= {rand(2,2,2), ' February', 10.28}
A =
[2x2x2 double] 'February', [10.2800]

A{1}
A{2}

B{1,1}=1:8;
B{1,2}=strvcat('Monday','Tuesday','Wednesday
','Thursday');
B{2,2}=A;
B{1,1}
cellplot(B)
```