Matlab Basics Workshop

MATLAB (Matrix Laboratory)

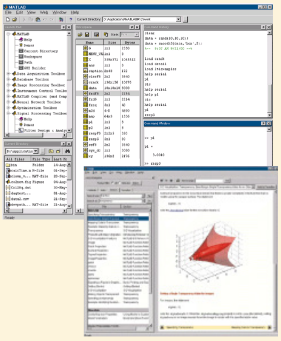
In the World

www.mathworks.com

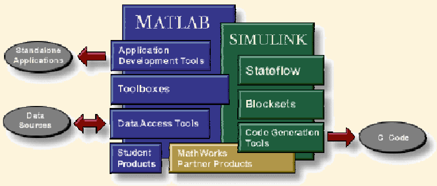
What is MATLAB

MATLAB(Matrix laboratory) is an interactive software system. It integrates mathematical computing, visualization, and a powerful language to provide a flexible environment for technical computing. Typical uses include

- **Math and computation**
- **Algorithm development**
- **Data acquisition**
- **Modeling, simulation, and prototyping**
- **Data analysis, exploration, and visualization**
- **Scientific and engineering graphics**
- **Application development, including graphical user interface building**



The MATLAB Product Family



The MathWorks offers a set of integrated products for data analysis, visualization, application development, simulation, design, and code generation. MATLAB is the foundation for all the MathWorks products.

Software Development Philosophy

- **Major software characteristics:**
  - matrix-based numeric computation
  - high-level programming language
  - graphics & visualization
  - toolboxes provide application-specific functionality
- **Multi-platform support (PC / Macintosh / Unix)**
- **Open & extensible system architecture**
- **Interfaces to other systems.**
  - Custom C, Fortran (**MATLAB** is callable)
  - Extensive data I/O facility

Outline

- **MATLAB** Fundamentals
- Plotting Fundamentals
- Programming and Application development

Matlab Basics

Matlab Fundamentals

Starting MATLAB

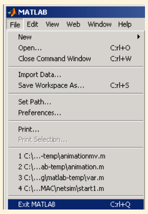
- **Windows**

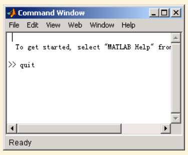
  double-click the MATLAB shortcut icon on your Windows desktop.

- **After starting MATLAB, the MATLAB desktop opens.**

Quitting MATLAB

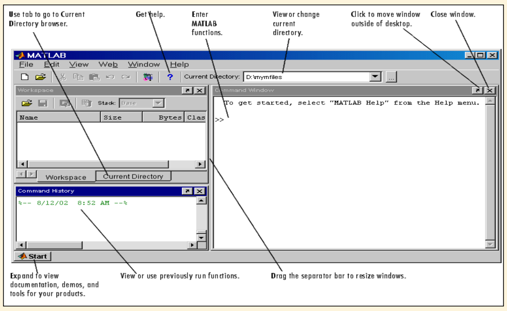
- **select Exit MATLAB from the File menu in the desktop, or type quit in the Command Window.**


MATLAB
File Edit View Web Window Help
New
Open... Ctrl+O
Close Command Window Ctrl+W
Import Data...
Save Workspace As... Ctrl+S
Set Path...
Preferences...
Print...
Print Selection...
1 C:\...-temp\animationmv.m
2 C:\...ab-temp\animation.m
3 C:\...g\matlab-temp\var.m
4 C:\...MAC\netsim\start1.m
Exit MATLAB Ctrl+Q



```
 To get started, select "MATLAB Help" fro

>> quit
```

MATLAB Desktop



```
%-- 8/12/02  8:52 AM --%
```

```
To get started, select "MATLAB Help" from the Help menu.
>>
```

Use tab to go to Current Directory browser.

Get help.

Enter MATLAB functions.

View or change current directory.

Click to move window outside of desktop.

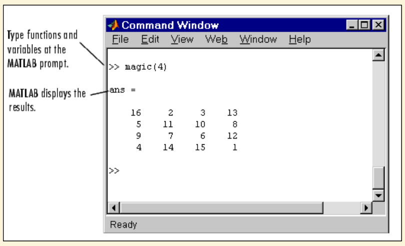
Close window.

Expand to view documentation, demos, and tools for your products.

View or use previously run functions.

Drag the separator bar to resize windows.

Command Window

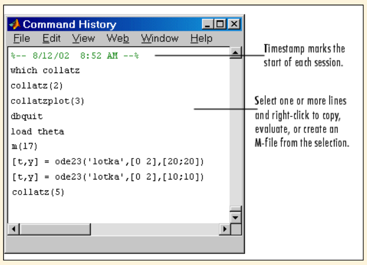


```
>> magic(4)

ans =

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

>>
```

Type functions and variables at the MATLAB prompt.

MATLAB displays the results.

Command History

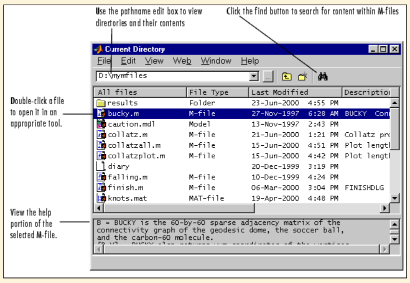


```
%-- 8/12/02  8:52 AM --%
which collatz
collatz(2)
collatzplot(3)
dbquit
load theta
m(17)
[t,y] = ode23('lotka',[0 2],[20;20])
[t,y] = ode23('lotka',[0 2],[10;10])
collatz(5)
```

Timestamp marks the start of each session.

Select one or more lines and right-click to copy, evaluate, or create an M-file from the selection.

Current Directory Browser



```
D:\mymfiles

All files       File Type   Last Modified          Descriptio
results         Folder      23-Jun-2000   4:55 PM
bucky.m         M-file      27-Nov-1997   6:28 AM  BUCKY  Con
caution.mdl     Model       13-Nov-1997   2:43 PM
collatz.m       M-file      21-Jun-2000   1:21 PM  Collatz pro
collatzall.m    M-file      15-Jun-2000   4:51 PM  Plot lengtl
collatzplot.m   M-file      15-Jun-2000   4:42 PM  Plot lengtl
diary                       20-Dec-1999   3:19 PM
falling.m       M-file      10-Dec-1999   4:24 PM
finish.m        M-file      06-Mar-2000   3:04 PM  FINISHDLG
knots.mat       MAT-file    19-Apr-2000   4:48 PM

B = BUCKY is the 60-by-60 sparse adjacency matrix of the
connectivity graph of the geodesic dome, the soccer ball,
and the carbon-60 molecule.
```

Use the pathname edit box to view directories and their contents

Click the find button to search for content within M-files

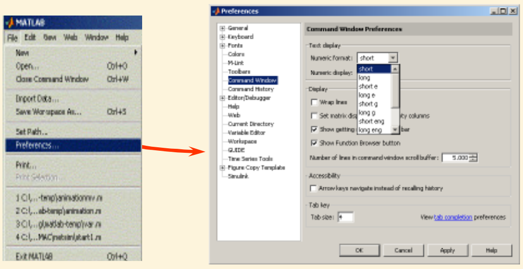
Double-click a file to open it in an appropriate tool.

View the help portion of the selected M-file.

Getting Started with MATLAB

```
» ver
---------------------------------------------------
MATLAB Version 6.0.0.88 (R12) on PCWIN
MATLAB License Number: Demo
---------------------------------------------------
MATLAB Toolbox      Version 6.0 (R12) 06-Oct-2000
Simulink            Version 4.0 (R12) 16-Jun-2000
MATLAB Compiler     Version 2.1 (R12) 26-Jul-2000
....
pwd    (present working directory)
ans =
c:\MatlabR11\work    (default)
```

Command Window Appearance



- **DIARY - saves text from MATLAB session**
- **MORE - controls scrolling of screen output**

Command Window Control

```
» x = 1.2345678901234567
x =
   1.2346
» format long
» x
x =
   1.23456789012346
» format bank
» x
x =
   1.23
» more on
» help stats
 Statistics Toolbox.
 Version 2.2  (R11)  24-Jul-1998
--more--
```

Getting help

- **The help command** `>> help`
- **The help window** `>> helpwin`
- **The lookfor command** `>> lookfor`
- **MATLAB Documentation**
- MATLAB Help Desk `>> helpdesk`
  - Online Reference (HTML / PDF) `>> doc`
  - Solution Search Engine
  - Link to The MathWorks (www.mathworks.com)
    - FTP site & latest documentation
    - Submit Questions, Bugs & Requests
- **MATLAB access -** MATLAB Digest / Download upgrades

Calculations at the Command Line

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

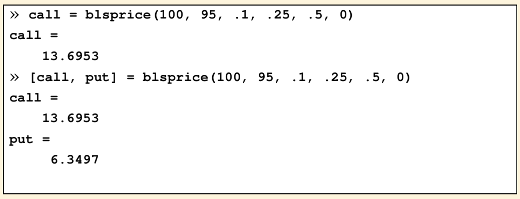
() parentheses for function inputs

Numbers stored in double-precision floating point format

Special Variables

- **ans : default variable name for the result**
- **pi: π = 3.1415926............**
- **eps: ε = 2.2204e-016, smallest amount by which 2 numbers can differ.**
- **Inf or inf : ∞, infinity**
- **NaN or nan: not-a-number**

Running an M-File

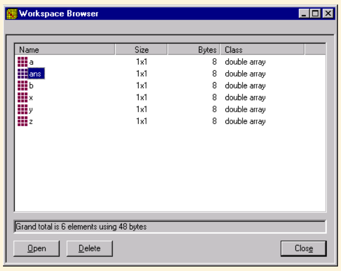


```
» call = blsprice(100, 95, .1, .25, .5, 0)
call =
    13.6953
» [call, put] = blsprice(100, 95, .1, .25, .5, 0)
call =
    13.6953
put =
     6.3497
```

Working with Files & Variables

- CD / PWD, LS / DIR - navigating directories
- WHAT - displays the files within a directory (grouped by type)
- ! - invoke operating system
- WHICH - identifies the object referenced by given name (function / variable)
- CLEAR - remove function / variable from memory
- WHOS - lists workspace variables and details (size, memory usage, data type)
- SIZE - returns the size of matrix

Workspace Browser



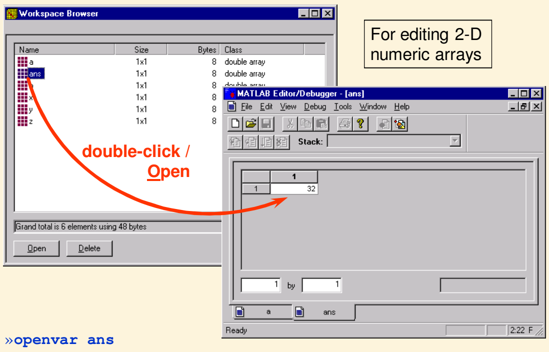
Name	Size	Bytes	Class
a	1x1	8	double array
ans	1x1	8	double array
b	1x1	8	double array
x	1x1	8	double array
y	1x1	8	double array
z	1x1	8	double array

Grand total is 6 elements using 48 bytes

Command line variables saved in MATLAB workspace

```
»workspace
```

Array Editor


Workspace Browser
Open
Delete
MATLAB Editor/Debugger - [ans]
File Edit View Debug Tools Window Help
Stack:
a
ans
Ready
2:22 F

Name	Size	Bytes	Class
a	1x1	8	double array
ans	1x1	8	double array
	1x1	8	double array
	1x1	8	
y	1x1	8	
z	1x1	8	

Grand total is 6 elements using 48 bytes

	1
1	32

1 by 1

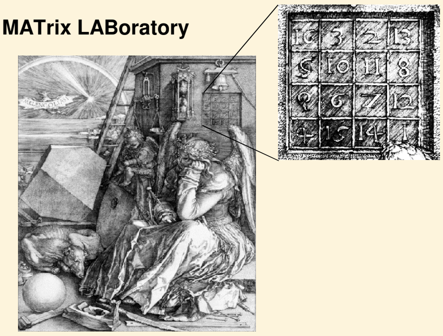
double-click / Open

For editing 2-D numeric arrays

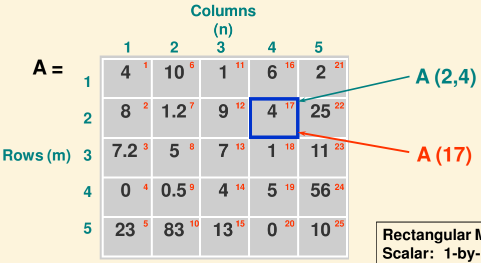
```
»openvar ans
```

Working with Matrices

MATLAB == MATrix LABoratory



The Matrix in MATLAB



Columns (n)

A =

Rows (m)	1	2	3	4	5
1	4 ^{1}	10 ^{6}	1 ^{11}	6 ^{16}	2 ^{21}
2	8 ^{2}	1.2 ^{7}	9 ^{12}	4 ^{17}	25 ^{22}
3	7.2 ^{3}	5 ^{8}	7 ^{13}	1 ^{18}	11 ^{23}
4	0 ^{4}	0.5 ^{9}	4 ^{14}	5 ^{19}	56 ^{24}
5	23 ^{5}	83 ^{10}	13 ^{15}	0 ^{20}	10 ^{25}

A (2,4)

A (17)

Rectangular Matrix:
Scalar: 1-by-1 array
Vector: m-by-1 array
1-by-n array
Matrix: m-by-n array

Entering Numeric Arrays

Row separator:
semicolon (;)

Column separator:
space / comma (,)

```
» a=[1 2;3 4]
a =
     1     2
     3     4
» b=[-2.8, sqrt(-7), (3+5+6)*3/4]
b =
   -2.8000    0 + 2.6458i   10.5000
» b(2,5) = 23
b =
   -2.8000   0 + 2.6458i  10.5000    0         0
         0             0        0    0   23.0000
```

Use square brackets [ ]

Matrices must be rectangular. (Set undefined elements to zero)

Any MATLAB expression can be entered as a matrix element

Running an M-File with Vector Input

```
» t = [0.1 0.25 0.5]
t =
    0.1000    0.2500    0.5000
» [tcall, tput] = blsprice(100, 95, t, .25, .5, 0)
tcall =
   13.6953   15.7429   19.3905
tput =
    6.3497    4.9871    3.2277
```

Entering Numeric Arrays - cont.

Scalar expansion

```
» w=[1 2;3 4] + 5
w =
     6     7
     8     9
```

Creating sequences:
colon operator (:)

```
» x = 1:5
x =
     1     2     3     4     5
» y = 2:-0.5:0
y =
   2.0000    1.5000    1.0000    0.5000    0
```

Utility functions for creating matrices.
(Ref: Utility Commands)

```
» z = rand(2,4)
z =
   0.9501    0.6068    0.8913    0.4565
   0.2311    0.4860    0.7621    0.0185
```

Numerical Array Concatenation - [ ]

Use [ ] to combine existing arrays as matrix “elements”

Row separator:
semicolon (;)

Column separator:
space / comma (,)

```
» a=[1 2;3 4]
a =
     1     2
     3     4
» cat_a=[a, 2*a; 3*a, 4*a; 5*a, 6*a]
cat_a =
     1     2     2     4
     3     4     6     8
     3     6     4     8
     9    12    12    16
     5    10     6    12
    15    20    18    24
```

Use square brackets [ ]

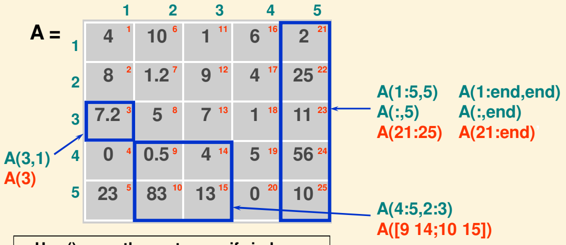
4*a

The resulting matrix must be rectangular.

Useful Commands

`x = start:end`	create row vector x starting with start, counting by one, ending at end
`x = start:increment:end`	create row vector x starting with start, counting by increment, ending at or before end
`linspace(start,end,number)`	create row vector x starting with start, ending at end, having number elements
`length(x)`	returns the length of vector x
`y = x'`	transpose of vector x
`dot (x, y)`	returns the scalar dot product of the vector x and y.

Array Subscripting / Indexing



A =

	1	2	3	4	5
1	4 [1]	10 [6]	1 [11]	6 [16]	2 [21]
2	8 [2]	1.2 [7]	9 [12]	4 [17]	25 [22]
3	7.2 [3]	5 [8]	7 [13]	1 [18]	11 [23]
4	0 [4]	0.5 [9]	4 [14]	5 [19]	56 [24]
5	23 [5]	83 [10]	13 [15]	0 [20]	10 [25]

- Use () parentheses to specify index
- colon operator (:) specifies range / ALL
- [ ] to create matrix of index subscripts
- ‘end’ specifies maximum index value

Array Subscripting

The colon notation may be used to address a block of elements:

(start : increment : end)

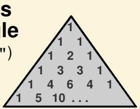
- start is the starting index
- increment is the amount to add to each successive index
- end is the ending index.

A shortened format (start : end) may be used if increment is 1.

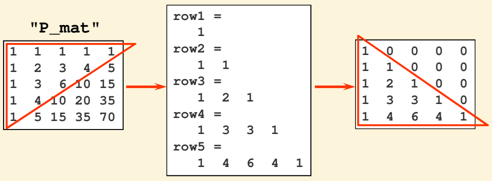
```
» >> x(1:3)
» ans =
»          0    0.7854    1.5708
```

Exercise: Creating Matrices

- Create a 5x5 Pascal matrix ("P_mat")
- Extract elements to create row vectors for the first 5 rows of Pascal's Triangle (Look for patterns in the element locations in "P_mat")
- Combine the vectors into a single matrix with zeros above the diagonal



Pascal's Triangle



"P_mat"

```
1   1   1   1   1
1   2   3   4   5
1   3   6  10  15
1   4  10  20  35
1   5  15  35  70
```

```
row1 =
   1
row2 =
   1   1
row3 =
   1   2   1
row4 =
   1   3   3   1
row5 =
   1   4   6   4   1
```

```
1   0   0   0   0
1   1   0   0   0
1   2   1   0   0
1   3   3   1   0
1   4   6   4   1
```

Solution appears on following page.

Solution: Creating Matrices (1)

```
» P_mat = pascal(5)              % Create Pascal's Matrix

» row1 = P_mat(1)                % Extract Rows
» row2 = P_mat(2:4:6)  .....
» row5 = P_mat(5:4:21)

» new_mat = row1                 % Create New Matrix
» new_mat(2,1:2) = row2 .....
» new_mat(5,1:5) = row5

% NOTE: GENERALIZED FORM:
% Nmax = length(P_mat);
% rowN = P_mat(N:Nmax-1:(N-1)*Nmax+1)
% new_mat(N,1:N) = rowN
```

»create_mat

Solution: Creating Matrices (2)

```
» P_mat = pascal(5)              % Create Pascal's Matrix

% Rearrange Matrix - Pascal Triangle in upper triangle
%  Flip Matrix Horizontally (LEFT-RIGHT):
%  - Could also use FLIPLR():
» temp = P_mat(1:5, 5:-1:1)

% Extract diagonals (rows of Pascal's Triangle)
» row1 = diag(temp,4)'
» row2 = diag(temp,3)'
» row3 = diag(temp,2)'
» row4 = diag(temp,1)'
» row5 = diag(temp)'

% Create New Matrix as before
```

»create_mat

Solution: Creating Matrices (3)

```
» P_mat = pascal(5)              % Create Pascal's Matrix

% Rearrange Matrix - Pascal Triangle in lower triangle
%  Flip Matrix Vertically (UP-DOWN):
%  - Could also use FLIPUD():
» temp1 = P_mat(5:-1:1, 1:5)

% Extract lower triangular matrix:
» temp2 = tril(temp1)

% Create New Matrix by sorting columns of "temp2":
» new_mat = sort(temp2)
```

»create_mat

Matrix Operations

- given: matrices of same size
- Element-by-Element Mathematics

Operation	Algebraic Form	MATLAB
Addition	a + b	a + b
Subtraction	a − b	a − b
Multiplication	a x b	a .* b
Division	a ÷ b	a ./ b
Exponentiation	a^b	a .^ b

»mat_ops

Matrix Multiplication

- Inner dimensions must be equal
- Dimension of resulting matrix = outermost dimensions of multiplied matrices
- Resulting elements = dot product of the rows of the 1st matrix with the columns of the 2nd matrix

```
» a = [1 2 3 4; 5 6 7 8];                     [2x4]
» b = ones(4,3);                              [4x3]
» c = a*b                  [2x4]*[4x3] ────→  [2x3]
c =
     10     10     10
     26     26     26 ←──── a(2nd row).b(3rd column)
```

»mat_mult

Array Multiplication

- Matrices must have the same dimensions
- Dimensions of resulting matrix = dimensions of multiplied matrices
- Resulting elements = product of corresponding elements from the original matrices

```
» a = [1 2 3 4; 5 6 7 8];
» b = [1:4; 1:4];
» c = a.*b
c =
      1      4      9     16
      5     12     21     32 ←──── c(2,4) = a(2,4)*b(2,4)
```

Same rules apply for other array operations

»array_mult

Example: Array Operations

- In most languages - use loops:

```
» tic; for I = 1:1000
Density(I) = Mass(I)/(Length(I)*Width(I)*Height(I));
end; toc
elapsed_time =                         Use TIC and TOC to
   0.0500                              measure elapsed time
```

- In MATLAB - use Array Operations:

```
» tic; Density = Mass./(Length.*Width.*Height); toc
elapsed_time =
                                       Vectorized code is
     0                                 much faster than loops
```

»array_examp

Boolean Operations

Boolean Operators
= = equal to
>   greater than
<   less than
~   not
& and
| or
isempty()
isfinite(), etc. . . .
any()
all()

```
» Mass = [-2 10 NaN 30 -11 Inf 31];
» all_pos = all(Mass>=0)
all_pos =
    0
» each_pos = Mass>=0
each_pos =
    0     1     0     1     0     1     1
» pos_fin = (Mass>=0)&(isfinite(Mass))
pos_fin =
    0     1     0     1     0     0     1
```

1 = TRUE
0 = FALSE

»bool_ops

More Operations

`zeros(n)`	returns a n x n matrix of zeros
`zeros(m,n)`	returns a m x n matrix of zeros
`ones(n)`	returns a n x n matrix of ones
`ones(m,n)`	returns a m x n matrix of ones
`size (A)`	for a m x n matrix A, returns the row vector [m,n] containing the number of rows and columns in matrix.
`length(A)`	returns the larger of the number of rows or columns in A.

More Operations

Operation	Matlab
Transpose	B = A'
Identity Matrix	eye(n) ➔ returns an n x n identity matrix eye(m,n) ➔ returns an m x n matrix with ones on the main diagonal and zeros elsewhere.
Scalar Multiplication	B = α*A, where α is a scalar.
Matrix Multiplication	C = A*B
Matrix Inverse	B = inv(A), A must be a square matrix in this case. rank (A) ➔ returns the rank of the matrix A.
Matrix Powers	B = A.^2 ➔ squares each element in the matrix C = A * A ➔ computes A*A, and A must be a square matrix.
Determinant	det (A), and A must be a square matrix.

Systems of Linear Equations

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

Systems of Linear Equations

- **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

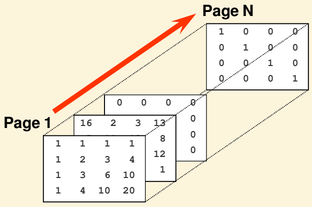
  can be computed using left division.

- **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

Multidimensional Arrays



```
» A = pascal(4);
» A(:,:,2) = magic(4)
A(:,:,1) =
     1     1     1     1
     1     2     3     4
     1     3     6    10
     1     4    10    20
A(:,:,2) =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
» A(:,:,9) = diag(ones(1,4));
```

```
»mult_dim
```

Polynomials

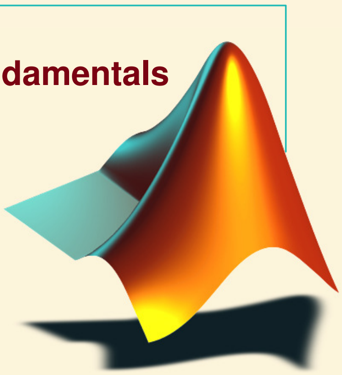
- The polynomials are represented by their coefficients in MATLAB.
- Consider the following polynomial:
  $A(s) = s^3 + 3s^2 + 3s + 1$
- For s is scalar: use scalar operations
  A = s^3 + 3*s^2 + 3*s + 1;
- For s is a vector or a matrix: use array or element by element operation
  A = s.^3 + 3*s.^2 + 3.*s + 1;
- function **polyval(a,s)**: evaluates a polynomial with coefficients in vector a for the values in s.

Polynomials

Operation	MATLAB command	Description
Addition	`c = a + b`	sum of polynomial A and B, the coefficient vectors must be the same length.
Scalar Multiple	`b = 3*a`	multiply the polynomial A by 3
Polynomial Multiplication	`c = conv(a,b)`	returns the coefficient vector for the polynomial resulting from the product of polynomial A and B.
Polynomial Division	`[q,r] = deconv(a,b)`	returns the long division of A and B. q is the quotient polynomial coefficient, and r is the remainder polynomial coefficient.
Derivatives	`polyder(a)`	returns the coefficients of the derivative of the polynomial A
	`polyder(a, b)`	returns the coefficients of the derivative of the product of polynomials A and B.
	`[n,d]=polyder(b,a)`	returns the derivative of the polynomial ratio B/A, represented as N/D
Find Roots	`roots(a)`	returns the roots of the polynomial A in column vector form.
Find Polynomials	`poly(r)`	returns the coefficient vector of the polynomial having roots r.

Matlab Basics

Plotting Fundamentals



2-D Plotting

- Specify x-data and/or y-data
- Specify color, line style and marker symbol

  (Default values used if ‘clmnot specified)

- Syntax:
  - Plotting single line:

    ```
    plot(xdata, ydata, 'color_linestyle_marker')
    ```

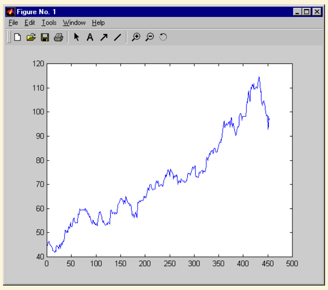
  - Plotting multiple lines:

    ```
    plot(x1, y1, 'clm1', x2, y2, 'clm2', ...)
    ```

2-D Plotting - example

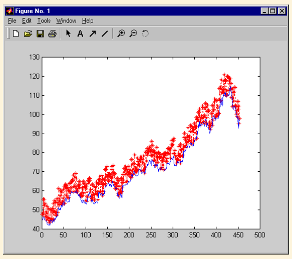
Load and plot a stock price time-series

```
» load ibm.mat
» plot(1:length(ibm), ibm)
```



Adding additional plots to a figure

- HOLD ON holds the current plot
- HOLD OFF releases hold on current plot
- HOLD toggles the hold state

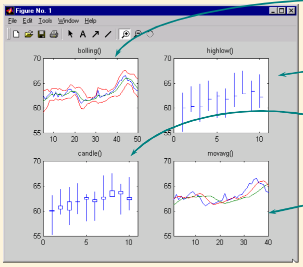


```
» new = ibm + rand(length(ibm),1)*10;
» hold on
» plot(1:length(ibm), new, 'r*:')
```

Subplots

SUBPLOT- display multiple axes in the same figure window

```
subplot(#rows, #cols, index)
```

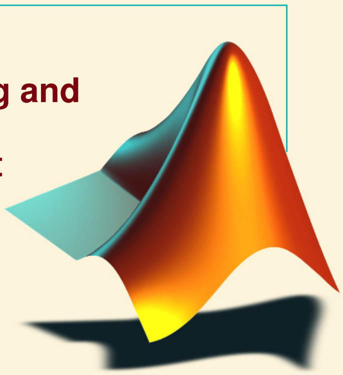


```
»subplot(2,2,1);
» …
»subplot(2,2,2)
» ...
»subplot(2,2,3)
» ...
»subplot(2,2,4)
» ...
```

```
»subplot_ex
```

Matlab Basics

Programming and Application Development



Script and Function Files

- **Script Files**
  - Work as though you typed commands into MATLAB prompt
  - Variable are stored in **global** work space
- **Function Files**
  - Let you make your own MATLAB commands
  - All variables within a function are **local**
  - All information must be passed to functions as parameters

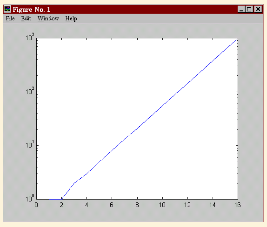
Programming Features

- **Subfunctions are supported**
  - multiple functions per file
- **Varying number of input/output arguments**
  - varargin, varargout
- **Many high-level language features**
  - if, while, for, switch, etc
- **Pcode compiler**
  - propreitary code hiding
  - first time code speedups

Script M-files

- **Standard ASCII Text file**
- **Executes a series of MATLAB commands on the base workspace.**

```
% An M-file to calculate
% Fibonacci numbers
  f = [1 1]; i = 1;
  while f(i) + f(i+1) < 1000
        f(i+2) = f(i) + f(i+1);
        i = i + 1;
  end
  semilogy(f)
```



Exercise: Script M-files

Given a current stock price, an expected rate of return and price volatility, write a script to generate series of possible stock prices for the next 12 months. Calculate data on a daily basis and plot the results.

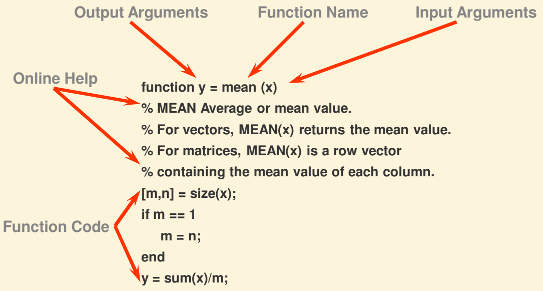
Hint: Use the function STOCKRND to generate the series.

Solution:

```
% time vector
t= 0:1/365:1;

% stock price
stockPrice = stockrnd(60, 0.1, t, 0.2, 10);

% plot prices
plot(stockPrice)
```

```
» stockscript
```

Function M-file



```
function y = mean (x)
% MEAN Average or mean value.
% For vectors, MEAN(x) returns the mean value.
% For matrices, MEAN(x) is a row vector
% containing the mean value of each column.
[m,n] = size(x);
if m == 1
    m = n;
end
y = sum(x)/m;
```

Function M-file

```
function r = ourrank(X,tol)
% rank of a matrix
s = svd(X);
if (nargin == 1)
  tol = max(size(X)) * s(1)* eps;
end
r = sum(s > tol);
```

Multiple Input Arguments
use ( )

```
»r = ourrank(rand(5),.1);
```

```
function [mean,stdev] = ourstat(x)
[m,n] = size(x);
if m == 1
  m = n;
end
mean = sum(x)/m;
stdev = sqrt(sum(x.^2)/m – mean.^2);
```

Multiple Output Arguments
use [ ]

```
»[m,std]=ourstat(1:99);
```

Matlab Commands for Functions

`function`	define function
`global`	define global variables
`nargchk`	validate number of input arguments
`nargin`	number of input parameters
`nargout`	number of output parameters
`return`	return to invoking function
`error`	display message and abort function
`eval`	execute string with Matlab expression
`feval`	evaluate function specified by string
`input`	prompt for user input
`keyboard`	invoke keyboard as if it were a script
`menu`	generate menu of choices for user input
`pause`	wait for user response

Programming Features

- **Loops and Conditional Statements**
- **For loops**
  - for i= 1:10
    y(i)= 3*i
    end
- **while loops**
  - i=1;
  - while i <= 10
    y(i)=3*i
    i=i+1;
    end

Programming Features (cont)

- **if statement**

  -
    ```
    for i = 1:41
       if i <= 10
          y(i)= 2*i;
       elseif  i <=20
          y(i)= 3*i;
       else
          y(i)= i-1;
       end
    end
    ```
- **Use efficient vectorised commands if possible**
  - y= [2:2:20 33:3:60 20:40]

Programming Features (cont)

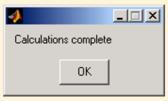
- **switch statement**

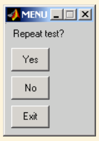
  -
    ```
    switch inputval
      case 1,          y = 2*i;
      case 2,          y = 4*i;
      otherwise,  disp(‘unknown option‘);
    end
    ```
- **Does not “fall through”   (not like C)**

```
  case  TrueValue,  y=1;
  case  NextValue,  y=2;   %%Not tested
```

Program Flow

- **Input from command line**
  - a = input('Enter value:')
- **Pause until key pressed**
  - pause
  - pause(2)   %wait 2 secs
- **Message box**
  - msgbox('Calculations complete')
- **Menu Window**
  - menu('Repeat test?','Yes','No',’Exit’)


Calculations complete



Exercise: Function M-Files

Write MATLAB function to price stock call option. The historical stock price is stored in IBM.MAT

The inputs should be: data file name (string), strike price and expected rate of return. The output is option price. The option expires in 6 months.

Use Monte-Carlo simulation (STOCKRND.M file) to calculate future stock prices. Create a scatter plot and a histogram of possible future stock values.

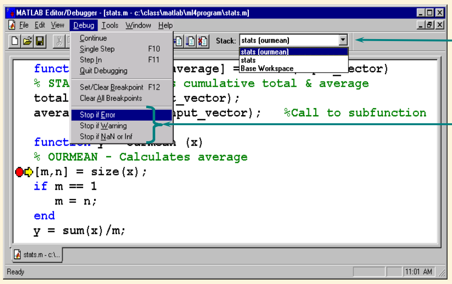
Assume that the option price will be equal to expected payoff, discounted by the rate of return.

Solution:

```
function optPrice = stockoption(fileName, Strike, Rate)

load(fileName);
currPrice = ibm(end);
Volat = std(ibm)/100;

stockPrice = stockrnd(currPrice, Rate, 0.5, Volat, 10000);

subplot(1,2,1)
plot(stockPrice,(1:10000),'.')
subplot(1,2,2)
hist(stockPrice)

payoff = max(0, stockPrice - Strike);
optPrice = mean(payoff) * exp(-0.1*0.5);
```

» **optprice = stockoption('ibm', 100, 0.1);**

Visual Debugging



```
funct                    average] =                  ector)
% STA                    s cumulative total & average
total                    t_vector);
avera                    nput_vector);   %Call to subfunction

function y                   (x)
% OURMEAN - Calculates average
[m,n] = size(x);
if m == 1
   m = n;
end
y = sum(x)/m;
```

Select Workspace

Set Auto-Breakpoints

- Set Breakpoint
- Clear Breaks
- Step In
- Single Step
- Continue
- Quit Debugging