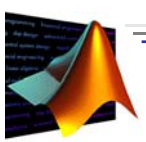
An Introductory Course in MATLAB with Financial Case Studies

Panayiotis C. Andreou, PhD
Cyprus University of Technology

What is Matlab?

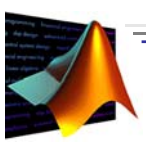
- ***Mat***rix ***Lab***oratories which is a registered trademark of the MathWorks, Inc.
- The first version of Matlab was written in 1970s by a numerical analyst named Cleve Moler.
- **Basic Feature**:

> *… is a high performance programming language and a computing environment that uses vectors and matrices as one of its basic data types and is a powerful tool for mathematical and technical calculations and for creating various types of plots.*



What is Matlab?

- It performs the basic functions of a programmable calculator whereas someone can write, run/execute and save a bundle of commands sentence by sentence.
- it integrates computation, visualization and programming in an easy to use environment via a subtle mathematical notation.
- Matlab has a broad spectrum of uses since it is keeping expanded by MathWorks, Inc. as well as by user defined programming codes. For example, the primary versions of Matlab were made for solving linear algebra type problems using matrices. Today, Matlab can be used in more fields.
- **Most Important**: ***You Learn By Doing!!!***



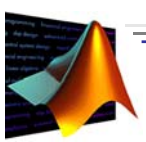
Two Important Features

- **Data Elements**:

  Its basic data element is an array that does not require dimensioning. That is, there is only one type of variable that is treated as a rectangular array. It can be either a ***scalar****, either a* ***row*** *or* ***column vector*** *or a* ***matrix****.*

- **Extensible and Powerful Language:**

  Matlab features a family of add-on application-specific solutions called toolboxes. A ***toolbox*** *is a comprehensive collection of Matlab functions (m-files) that extend the Matlab environment to solve particular classes of problems.* Additionally, the user is free to create its own classes of functions to deal with specialized problems.



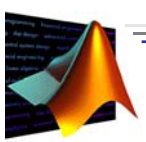
Starting Up Matlab

Step 1: Use the [**Ctrl**]+[**Alt**]+[**Del**] combination to bring up the logon screen (at this point you should enter the user name and your password and after to press [**Enter**])

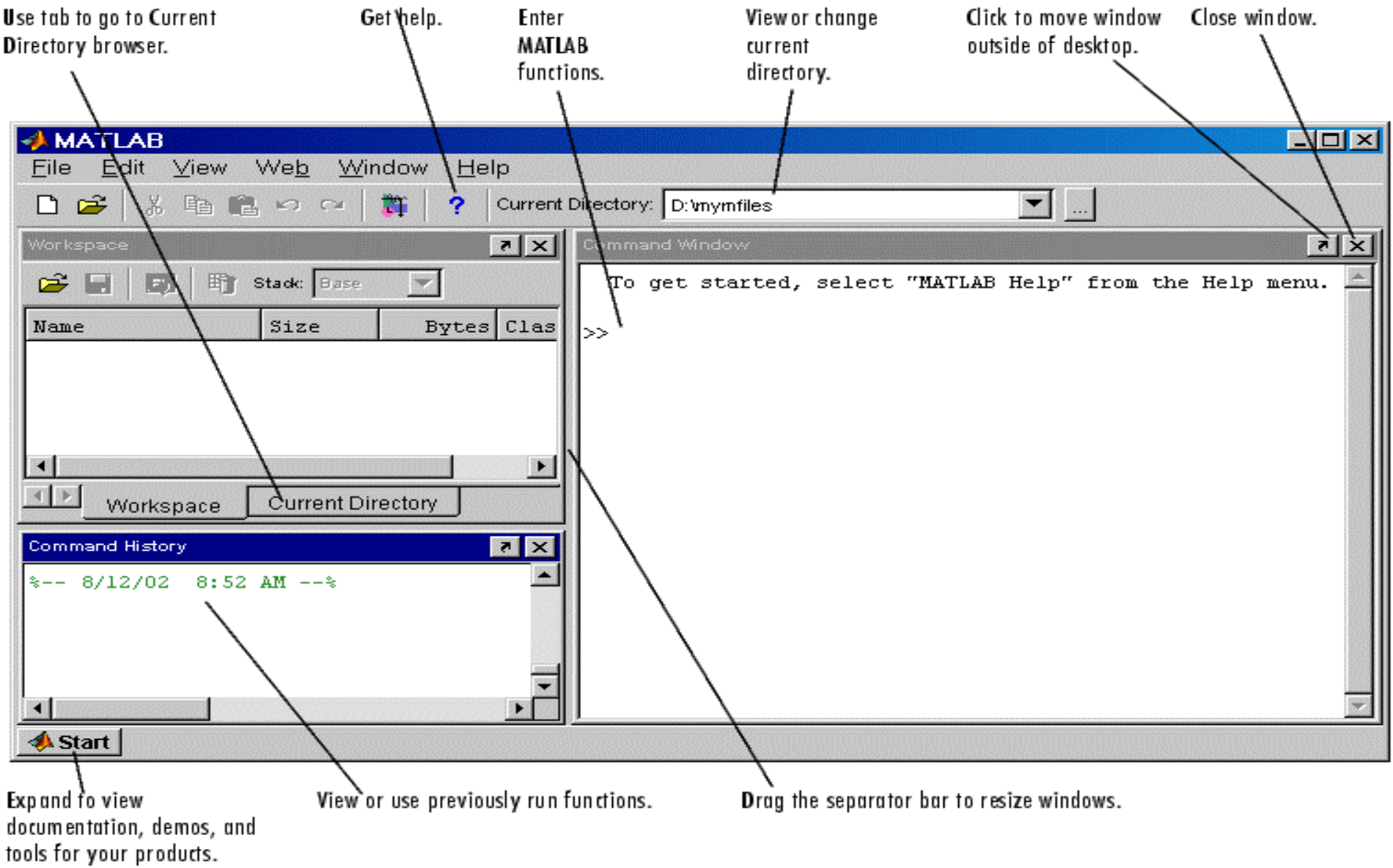
Step 2: After few seconds, you can view the PC's Desktop screen with all available icons. Find the Matlab's shortcut icon (labeled as "Matlab" and looks like: ) and double click on it. After few moments, the Matlab starts up and the following words appear in one of the screens:

To get started, select "MATLAB Help" from the Help menu

Step 3: The Matlab is now ready to be used! (if you want to quit Matlab, from the window named as *Command Window* either type *quit* or exit from the toolbar choose: *File > Exit Matlab*)

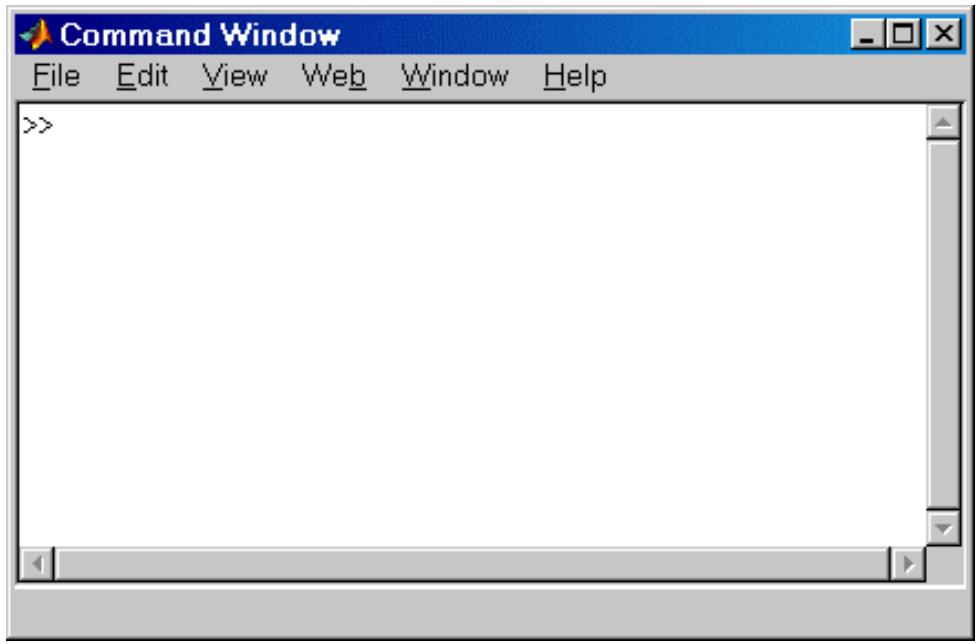


Matlab's Window: Desktop

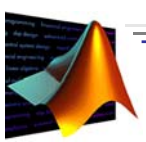


Matlab's Window: Command Window

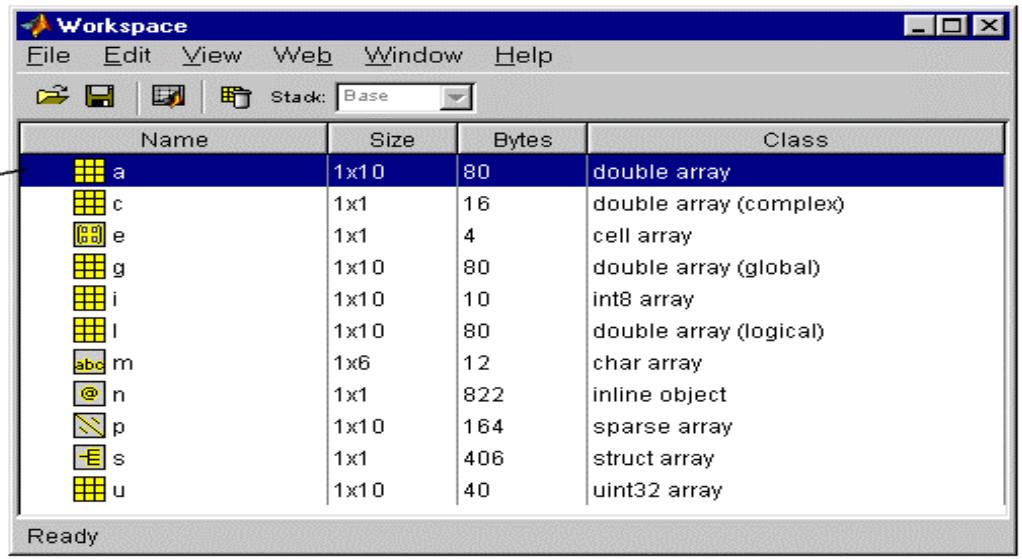
It is the main window in which the user communicates with the software. In the *command window*, the user can view the *prompt symbol* ">>" which indicates that Matlab is ready to accept various commands by the user.



Via this window, the user can employ the basic arithmetic operators like: "+" (addition), "-" (subtraction), "*" (multiplication), "/" (division), "^" (powers) and the "( )" (brackets), as well as many other build in elementary functions and commands

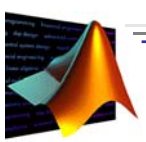


Matlab's Window: Workspace



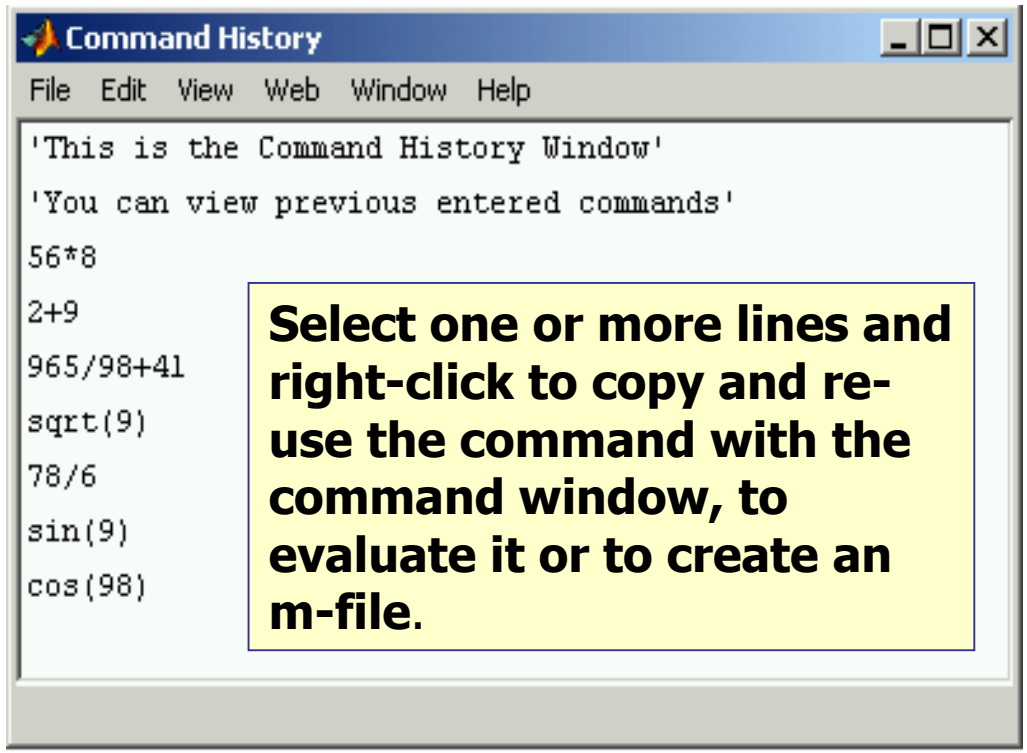
The Matlab *workspace* consists of the set of variables (named arrays) built up during a Matlab session and stored in memory.

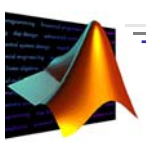
You add variables to the *workspace* by using *functions*, running *m-files*, and loading saved *workspaces*. The w*orkspace* is not maintained after you end the Matlab session. To save the *workspace* to a file that can be read during a later Matlab session, select *Save Workspace As* from the *File* menu.



Matlab's Window: Command History

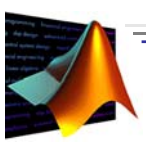
Statements that enter in the *Command Window* are logged in the *Command History*. In the *Command History*, you can view previously run statements, and copy and execute selected statements.





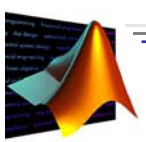
Matlab's Window: Current Directory

Matlab file operations use the *current directory* and the *search path* (*File> Set Path…*) as reference points. Any file you want to run must either be in the current directory or on the search path. A quick way to view or change the current directory is by using the *Current Directory* field in the desktop toolbar. To search for, view, open, and make changes to Matlab - related directories and files, use the Matlab *Current Directory Browser* which is called after clicking the icon:  .



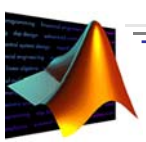
Matlab Variable Names

- **Variable** names are **case sensitive:**
  - Accepted *variables names* do not start with symbols (e,g: ~, _) or numbers, use lower and upper case letters, do not exceed 63 characters and do not resemble reserved works and build-in functions.
- Matlab recognizes only one type of variable:
  - ***scalar:*** *1-by-1 array*
  - ***vector:*** *1-by-c (row vector with c columns)*
    1-by-r (column vector with r rows)
  - ***array:*** *an r-by-c array (a matrix with r rows and c columns)*



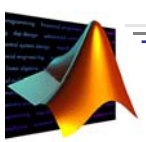
Matlab Data Types

- The **data type** is a classification of particular type information:
  - *integer* a whole number, a number without any fraction (e.g. 12);
  - *floating point* a number with a fractional part (e.g. 25.7)
  - *character* readable text character (e.g. 'Matlab').
- With Matlab, it is not needed to type or declare variables. Any operation that assigns a value to a variable creates the variable, if needed, or overwrites its current value, if it already exists.



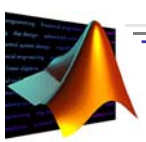
Command Window: Matlab as Calculator

- Matlab's Command Window is an active calculator in which mathematical statements are executed. At minimum, Matlab is a scientific calculator that can perform all operations that are carried out from pocket scientific calculators.
- User is allowed to assign a name to an expression. After assigning the name, this expression becomes a variable (scalar, vector or matrix) with a certain data type.
- All expressions entered, are saved in the Matlab's workspace and can be recalled in a later stage with their variable name.
- If no variable name is given to an expression, Matlab automatically assigns the name: **ans**



Math and Assignment Operators

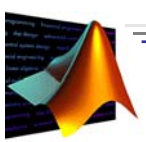
- **Basic Math Operators:**
  - Addition and unary addition: **+**
  - Subtraction and unary subtraction: **-**
  - Power: **^**
  - Division: **/**
  - Left division: ****
- **Assignment Operator:**
  - Assignment: **=**
- **Special Character:**
  - Bracket: **()**



Examples

- In the command window enter the following:

Input Command	Result
>>1+1	ans=2
>>8-2	ans=6
>>(5*2+1)	ans=11
>>52/47*8+1	ans=9.8511
>>(((2*(2+1)^2)/3)/9)*3	ans=2
>>10/5	ans=2
>>10\5	ans=0.5000

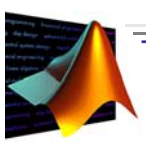


Help Facilities

- Matlab has additional operators except those exhibited previously. **How can you find them?**
- Matlab is a technical software that is enhanced with extensive online help, via various help facilities that follow:

1. **help command**
2. **lookfor command**
3. **Help Browser**

- **If the user knows the topic in which an informative help tip is needed it can use the help command.**
- **The problem with the help command is that the user must be familiar with the topic under consideration and the word following the help command must be exact and spell correctly.**

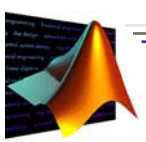


Help Facilities

- Matlab has additional operators except those exhibited previously. **How can you find them?**
- Matlab is a technical software that is enhanced with extensive online help, via various help facilities that follow:

1. **help command**
2. **lookfor command**
3. **Help Browser**

Example:
In the command window type:
>> help operators
>> help ops

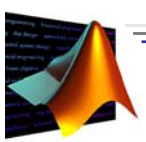


Help Facilities

- Matlab has additional operators except those exhibited previously. **How can you find them?**
- Matlab is a technical software that is enhanced with extensive online help, via various help facilities that follow:

1. **help command**
2. **lookfor command**
3. **Help Browser**

- **More flexible for pursuing help from Matlab when the user is not familiar with the exact key word. It looks for the given string in the first comment line of the help text in all *m-files* located in Matlab's toolboxes.**
- **It is time consuming and sometimes takes up to some minutes to come up with a result.**

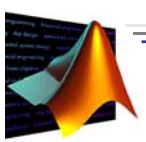


Help Facilities

- Matlab has additional operators except those exhibited previously. **How can you find them?**
- Matlab is a technical software that is enhanced with extensive online help, via various help facilities that follow:

1. **help command**
2. **lookfor command**
3. **Help Browser**

Example:
In the command window type:
>> lookfor operators
>> lookfor ops

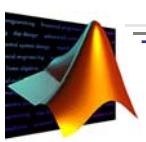


Help Facilities

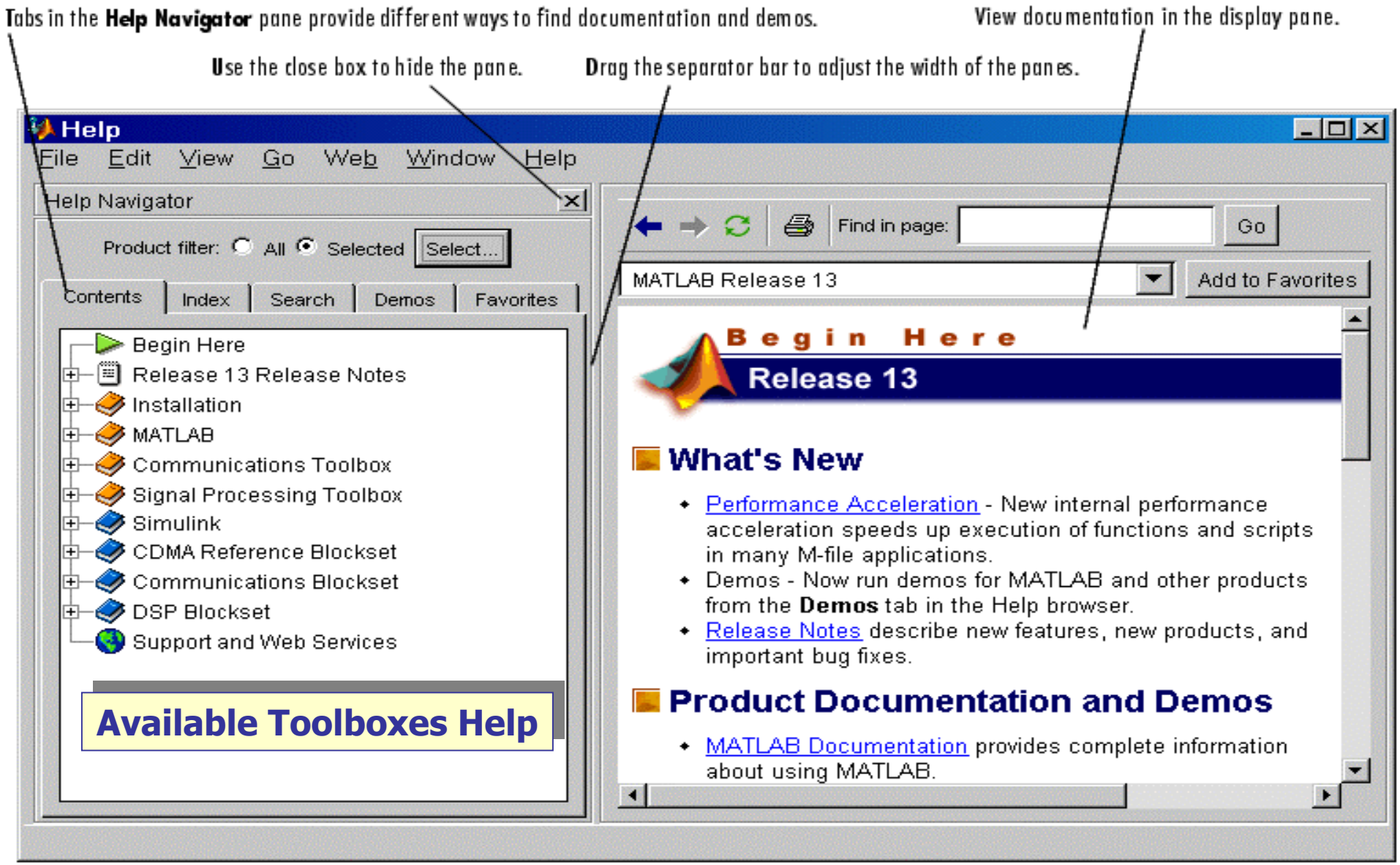
- Matlab has additional operators except those exhibited previously. **How can you find them?**
- Matlab is a technical software that is enhanced with extensive online help, via various help facilities that follow:

1. **help command**
2. **lookfor command**
3. **Help Browser**

Online help can also be obtained via the *Help* menu found in the Matlab's desktop. From the toolbar, select *Help>Matlab Help* to get the help browser with a list of help topics. Through this screen, the user can navigate around a variety of topics by double clicking on them (this browser displays html help pages and can be operate like the Internet Explorer).



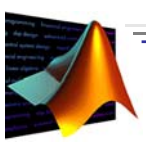
Help Browser



Examples with Elementary Functions

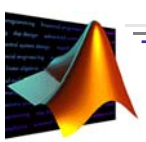
- In the command window type: "**help** elfun" and carry out the following examples:

Input Command	Result	Type
>> Tr1=sin(0.1)	Tr1 = 0.0998	Trigonometric
>> Tr2=cos(5)	Tr2 = 0.2837	Trigonometric
>>Tr3=tan(2.5)	Tr3=-0.7470	Trigonometric
>> Exp1=exp(1)	Exp1=2.7183	Exponential
>> Exp2=log(2)	Exp2=0.6931	Exponential
>> Exp3=sqrt(25)	Exp3=5	Exponential
>> RR1= fix(2.587)	RR1=2	Rounding and Remainder
>> RR2=round(1.252)	RR2= 1	Rounding and Remainder
>> RR3=rem(8,3)	RR3= 2	Rounding and Remainder



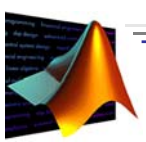
Controlling Command Window and Workspace

- The variables that you have assigned previously are saved in the workspace. Type their name at the prompt ">>" to recall them.
- **Useful Commands:**
  - **who**: Lists current variables located in the workspace.
  - **whos**: Is a long form of who. It lists all the variables in the current workspace, together with information about their size, bytes, class, etc.
  - **clear all**: Clear all variables and functions from memory.
  - **home**: Moves the cursor to the upper left corner of the command window and clears the visible portion of the window.
  - **clc**: Clears the command window and homes the cursor.
  - **quit**: Terminates Matlab.
  - **what**: Lists all Matlab files (*m-files*) in current directory
  - **dir, cd, cd ..**: Likewise DOS



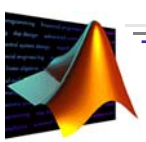
The format Command

- The **format** function controls the numeric format of the values displayed by Matlab. The function affects only how numbers are displayed, not how Matlab computes or saves them.
- In the command window, type first the **format** command and then recall a workspace variable (e.g. Exp1):
  - **format short e**: Floating point format with 5 digits.
  - **format short g**: Best of fixed or floating point format with 5 digits.
  - **format long e**: Floating point format with 15 digits.
  - **format long g**: Best of fixed or floating point format with 15 digits.
  - **format bank**: Fixed format for dollars and cents.
- Use the **help** command to see more formats.



Matlab Special Variables

pi	The Matlab value for π
eps	Machine precision. Floating point relative accuracy.
realmin	Smallest positive floating point number (=2.2251e-308).
reamax	Largest positive floating point number (= 1.7977e+308.
NaN	Not a Number (e.g. 0/0, Inf/Inf).
Inf	Positive infinitive (e.g. 1/0).
…	Continue a long statement to the next line.
,	Separates statements and vector elements.
;	Suppress output and rows in matrices.
%	Insert a comment line.
:	Specify range (e.g. starting_value**:**step**:**finishing_value*)*



Command Line Editing

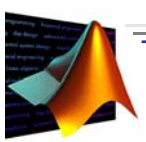
- Various arrow and control keys on your keyboard allow you to recall, edit, and reuse statements you have typed earlier. For example, suppose you mistakenly enter:

  >> G = (1 + sqt(5))*sin(pi)

- You have misspelled square root function: **sqrt**. Matlab responds with:

  "**Undefined function or variable 'sqt'**"

- Instead of retyping the entire line, simply press the ↑ key. The statement you typed is redisplayed. Use the ← key to move the cursor over and insert the missing "r". Repeated use of the ↑ key recalls earlier lines. Typing a few characters and then the ↑ key finds a previous line that begins with those characters. You can also copy previously executed statements from the Command History window.



The Black-Scholes-Merton (BSM) Formula

- Given a set of parameters, it gives the value of a European call or put option.

$$c^{BSM} = Se^{-\delta T} N(d_1) - Xe^{-rT} N(d_2)$$

$$d_1 = (ln(S/X) + (r - \delta + \sigma^2/2)T)/\sigma\sqrt{T} \qquad d_2 = d_1 - \sigma\sqrt{T}$$

S: *Current stock value (underlying asset)*

X: *Option's exercise price*

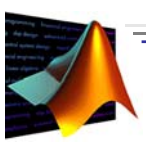
T: *Time to maturity in years fraction*

σ: *Stock's volatility/log-relative returns standard deviation (%)*

r: *Continuously compounded risk free rate with maturity **T** (%)*

δ: *Stock's dividend yield (%)*

N(.): *cumulative normal distribution function N(0,1)*

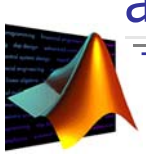


Practicing the BSM Formula

- Notice that the Matlab build in function for square root is **sqrt()**, for natural logarithm is **log()**, for exponential is **exp()**, and for the standard normal cumulative distribution is **normcdf()**.
- **In the command window write**:

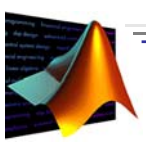
```
>> S=105, X=100, T=0.1, sig=0.25, r=0.05, div=0.02
>> d1=(log(S/X)+(r-div+sig^2/2)*T)/(sig*sqrt(T))
>> d2=d1-sig*sqrt(T); Nd1=normcdf(d1); Nd2=normcdf(d2);
>> Call=S*exp(-div*T)*Nd1-X*exp(-r*T)*Nd2
```

- After you type all commands, the correct price for the call option is: **Call = 6.5321**.
- Observe that multiple statements can be entered in one line if they are separated by “,” or “;” (what is their difference).



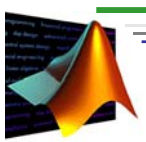
Vectors

- A vector is a list of numbers separated by either space or commas. Each different number/entry located in the vector is termed as either element or component. The number of the vector elements/components determines the **length** of the vector. In Matlab, square brackets "[ ]" are used to both define a vector and a matrix.
- Matlab can handle both row and column vectors. A row vector is produced by the transpose of a column vector and vise versa. Matlab returns the transpose of a vector when ' follows the definition of a vector.
- Build in functions of vectors are executed element-wise.



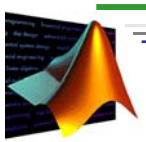
Examples with Vectors

Input	Output
>> y=[5 **exp**(2) **sign**(-5) **pi**]	y= 5 7.3891 -1 3.1416
>> L=**length**(y)	L= 4
>> y=2*y	y= 10.0000 14.7781 -2.0000 6.2832
>> X=1:3	X= 1 2 3
>> X=[2:3:11]	X= 2 5 8 11
>> X=[2:3:10]	X= 2 5 8
>> **exp**(**log**(X))	X= 2.0000 5.0000 8.0000
>> xx=X', xy=[1; 8; -9]	xx = 2 xy= 1 5 8 8 -9



Examples with Vectors

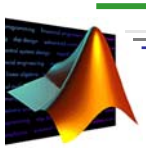
Input	Output
>> v1=1:8, v2=-4:2:2	v1= 1 2 3 4 5 6 7 8 v2= -4 -2 0 2
>> v3=v1(2:4)	v3= 2 3 4
>> v4=[v1(3:2:8) v2(2)]	v4= 3 5 7 -2
>> v5=[v1(1) v4(**end**) v2(**end**-1)]	v5= 1 -2 0
>> v6(1:2)=5; v6(3:4)=-1	v6=5 5 -1 -1
>> v7=[v4']; **length**(v7)	ans= 4
>> v2(2:**end**)	ans= -2 0 2
>> tanh(log(sqrt(exp(v2))))	ans=-0.9640 -0.7616 0 0.7616
>> v1(2:3:end)=[], v2=[]	v1=1 3 4 6 7, v2=[]



Vectors' Mathematical Operations

- When manipulating row and column vectors, pay attention to have similar lengths.

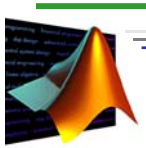
Input	Output
>> w=[1 0 2 -1]; u=[2; 4; -2; 1];	Nothing is displayed
>> prod1 = w * u	prod1=-3
>> prod2 = (2+w)*(u/2)	prod2= 3.5000
>> prod3 = w*w'	prod3= 6
>> dprod1=w.*u'	dprod1= 2 0 -4 -1
>> dprod2=u'.*u'.*w	dprod2= 4 0 8 -1
>> ddiv1=w./u'	ddiv1=0.5000 0 -1.0000 -1.0000
>> ddiv2=(u./w')'	ddiv2= 2 Inf -1 -1



Vectors' Mathematical Operations

- When manipulating row and column vectors, pay attention to have similar lengths.

Input	Output
>> who, clear all	Displays, cleans workspace
>> x=[0.5 1 20 50]; d=**sqrt**(0:3);	Nothing is displayed
>> (**exp**(x)-**exp**(-x))./(**exp**(x)+**exp**(-x))	ans= 0.4621 0.7616 1.0000 1.0000
>> d.^2	ans = 0 1.0000 2.0000 3.0000
>> d.^4	ans= 0 1.0000 4.0000 9.0000
>> d.*d	ans = 0 1.0000 2.0000 3.0000
>> ans.*ans	ans= 0 1.0000 4.0000 9.0000
>> ans./ans	ans= NaN 1.0000 1.0000 1.0000



The BSM Revisited

- Let's say that we want to price European call options for the following values of ***S***, ***X*** and ***T*** with all other data the same:

  (90, 100, 0.1), (80,150, 0.15), (100, 80, 2), (10,10, 1)

- **In the command window write**:

  >> S=[90 80 100 10], X=[100 150 80 10], T=[0.1 0.15 2 1]

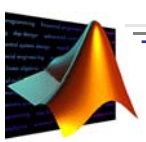
  >> sig=0.25, r=0.05, div=0.02

  >> d1=(**log**(S./X)+(r-div+(sig.^2)./2).*T)./(sig.***sqrt**(T))

  >> d2=d1-sig.***sqrt**(T); Nd1=**normcdf**(d1); Nd2=**normcdf**(d2);

  >> Call=S.***exp**(-div.*T).*Nd1-X.***exp**(-r.*T).*Nd2

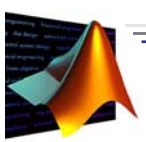
- After you type all commands, the correct price for the call option is:
  Call = 0.3439 0.0000 27.1985 1.1124



Two Dimensional Arrays: Matrices

- A two dimensional array is a composition of row and column vectors, created by using spaces (or commas) and semicolons.

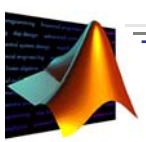
Input	Output
>> clear all, clc	Cleans workspace and homes cursor
>> B=**linspace**(-1,2,4); C=1:4;	Nothing is displayed
>> A1=[B;C]	A1= -1 0 1 2 1 2 3 4
>> A2=[A1;A1;[NaN Inf 1 -2]]	A2= -1 0 1 2 1 2 3 4 -1 0 1 2 1 2 3 4 NaN Inf 1 -2



Two Dimensional Arrays: Matrices

- A two dimensional array is a composition of row and column vectors, created by using spaces (or commas) and semicolons.

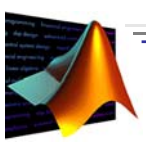
Input	Output
>> [M N]=**size**(A2)	M= 5 N=4
>> A2(:,1)'	ans = -1 1 -1 1 NaN
>> A2(1,:)	ans = -1 0 1 2
>> A3=[A2(1,1) A2(2,3) A2(10)]	A3= -1 3 Inf
>> A4=[A2(1:2,2:4); A2(5,2:4)]'	A4 = 0 2 Inf
	1 3 1
	2 4 -2



Matrices Mathematical Operations

- Matrices mathematical operations follow the same rationale as with vectors'.

Input	Output
>> G=[1 -3; 2 5]; P=[8; 9];	Displays, cleans workspace
>> (G*P)'	ans= -19 61
>> P'*G*G*P	ans= 787
>> G(1,:)*P	ans= -19
>> ([P+2,P./2]*G(:,2))'	ans = -10.0000 -10.5000
>> G.^3/2+[P,P]./2	ans= 4.5000 -9.5000 8.5000 67.0000
>> (G.*G./([P,P]+1)*P)'*(P-8.5)	ans= 7.9056



The BSM Revisited (Sensitivities)

- Let say that we want to price European call options for the following values of ***S***, ***T*** and ***r*** with all other data the same:

S

100	105	110	115
100	100	100	100
100	100	100	100

T

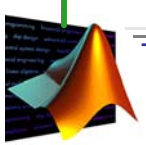
0.1	0.1	0.1	0.1
0.15	0.20	0.25	0.30
0.1	0.1	0.1	0.1

r

0.05	0.05	0.05	0.05
0.05	0.05	0.05	0.05
0.01	0.02	0.03	0.04

X = 100, **σ**= 0.25,

δ= 0.02



The BSM Revisited (Sensitivities)

- **In the command window write**:

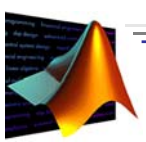
```
>> S=[100:5:115]; S(2:3,:)=100; X=100; sig=0.25; div=0.02;
>> T(1:3,1:4)=0.1; T(2,:)=0.15:0.05:0.30
>> r(1:2,1:4)=0.05; r=[r; 0.01 0.02 0.03 0.04]
>> d1=(log(S./X)+(r-div+sig.^2./2).*T)./(sig.*sqrt(T))
>> d2=d1-sig.*sqrt(T); Nd1=normcdf(d1); Nd2=normcdf(d2);
>> Call=S.*exp(-div.*T).*Nd1-X.*exp(-r.*T).*Nd2
```

- After you type all commands, the correct price for the call option is:

Call = 3.2938 6.5321 10.7024 15.3878

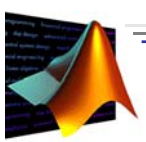
4.0690 4.7312 5.3208 5.8585

3.0987 3.1468 3.1953 3.2444



2D Plots

- There are powerful build in functions for creating 2D plots. Matlab can plot one vector Vs another. Always, the first vector is taken to be the abscissa vector (x-axis) and the second the ordinate (y-axis). Always, to create a 2D plot the length of the plotted vectors should be the same.
- There is the possibility to plot a vector Vs its index. That is, if only one vector is called with the plot command, then Matlab plots each element of the input vector in the ordinate (y-axis) Vs an index in the x-axis (Index=[1, 2, …, **length**(*input_vector*)]).
- Via a set of additional commands, a figure plot can be given a title, a label to its axis, add text anywhere in the plot, etc.



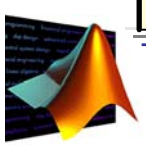
2D Plots

- A 2D line (or mark) plot is created via the **plot**() build in function. It general calling syntax is:

plot(X,Y, '#@$')

where: **#** represents a color Matlab symbol
@ represents a mark Matlab symbol
$ represents a line Matlab symbol

Symbol	Color (#)	Symbol	Line Style ($)
g	Green	**-, :, -., --**	Solid, dotted, dash dot and dashed
r	Red	**Symbol**	**Mark Style (@)**
c	Cyan	**., o, x**	Point, circle and x-mark
m	Magenta	**+, *, s**	Plus, star and square
y	Yellow	**d, v**	Diamond and triangle down
k	Black	**^, <, >**	Triangles: up, left and right
		p, h,	Pentagram, hexagram and solid

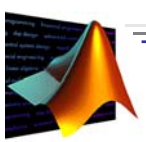


2D Examples

- Insert the following examples to the command window in order to plot the function ***f(x)*** in the area [0,5] and experiment with additional features related to plots:

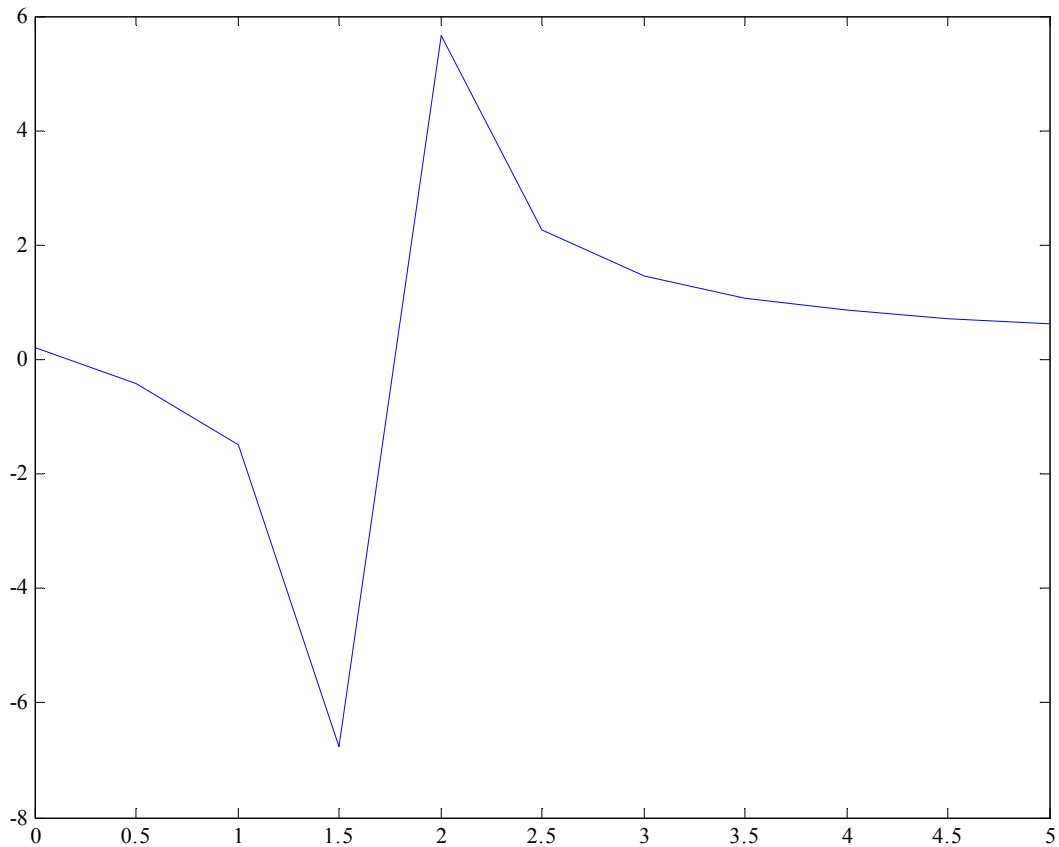
$$f(x) = (2x^2 + 5x - 1)/(x^3 - 5)$$

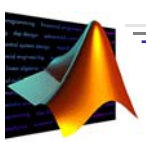
```
>> x=0:0.5:5; y=(2*x.^2+5*x-1)./(x.^3-5); plot(x,y);
>> x=0:0.25:5; y=(2*x.^2+5*x-1)./(x.^3-5); plot(x,y, 'rh-.');
>> xlabel('x-values'); ylabel('y-values'); title('y=(2x^2+5x-1)/(x^3-5)');
>> hold on; y1=rand(1,length(y)); plot(x,y1, 'g*-- ');
>> close all; plot(x,y, 'bp--',x,log(y1.^2), 'rh',x,x+zeros(1,length(x)), 'ms');
>> legend('y','log(y1^2)','x'); title('Various Plots');
>> figure; x=linspace(0,2*pi,25); subplot(3,1,1); plot(x, sin(x),'r*-.'); ylabel('sin(x)');
>> subplot(3,1,2); plot(x, cos(x),'gh-.'); ylabel('cos(x)');
>> subplot(3,1,3); plot(x, x.*sin(x), 'r*'); ylabel('x*sin(x)');
```



2D Examples

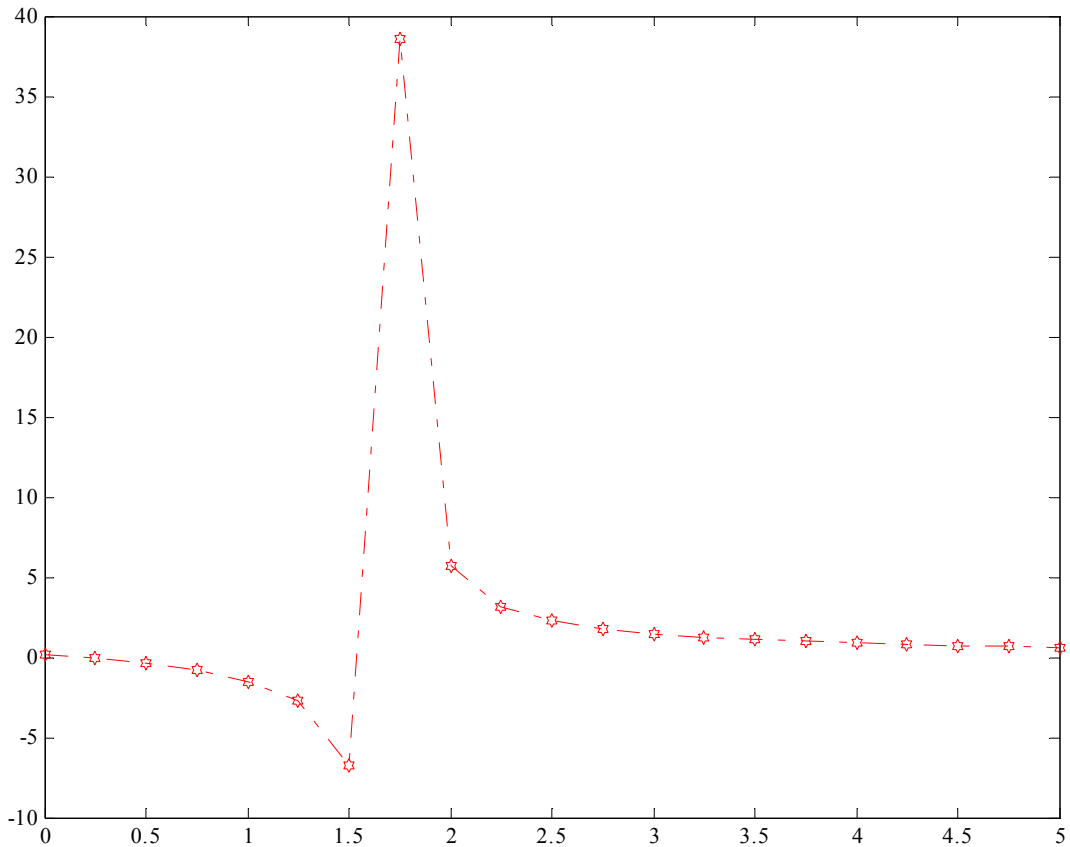
```
>> x=0:0.5:5; y=(2*x.^2+5*x-1)./(x.^3-5); plot(x,y);
```

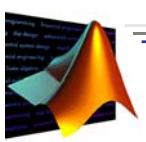




2D Examples

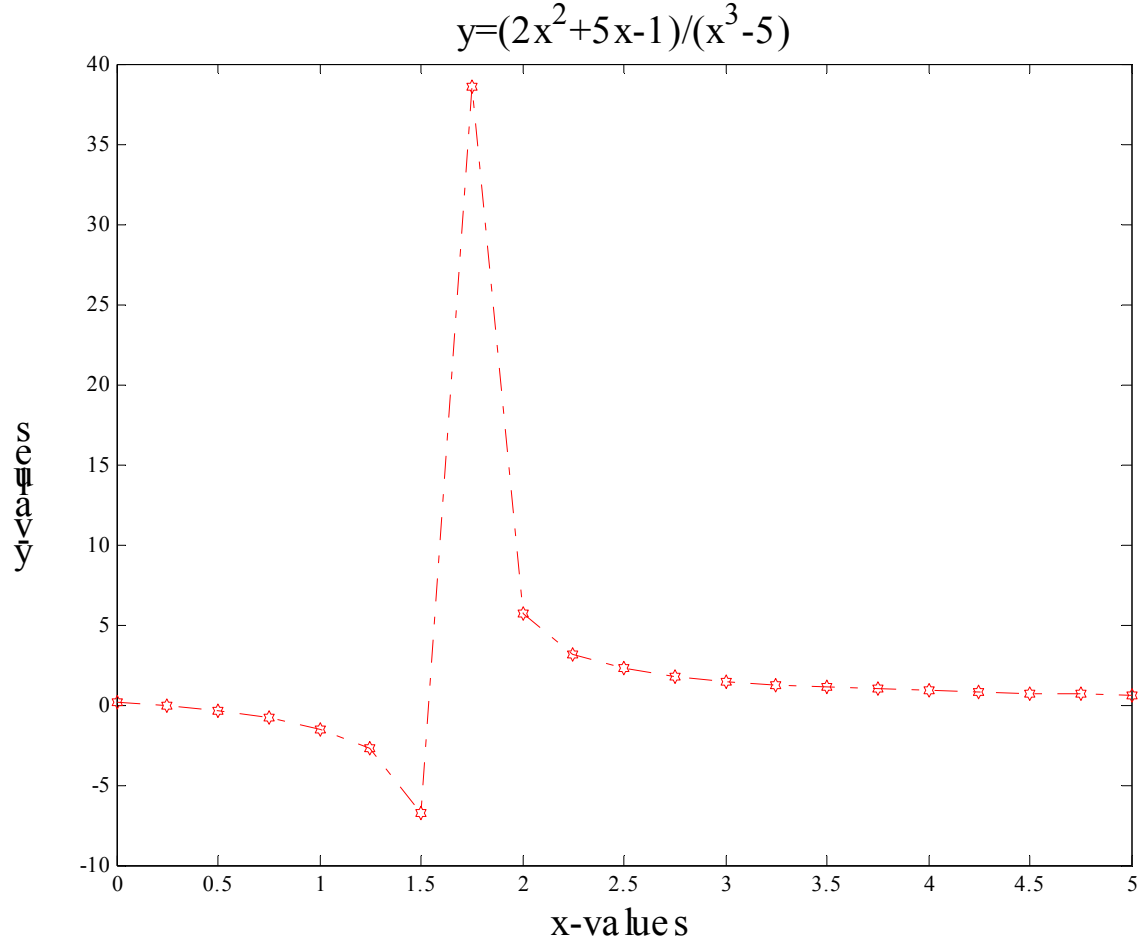
```
>> x=0:0.25:5; y=(2*x.^2+5*x-1)./(x.^3-5); plot(x,y, 'rh-.');
```

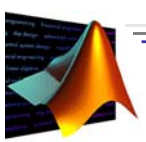




2D Examples

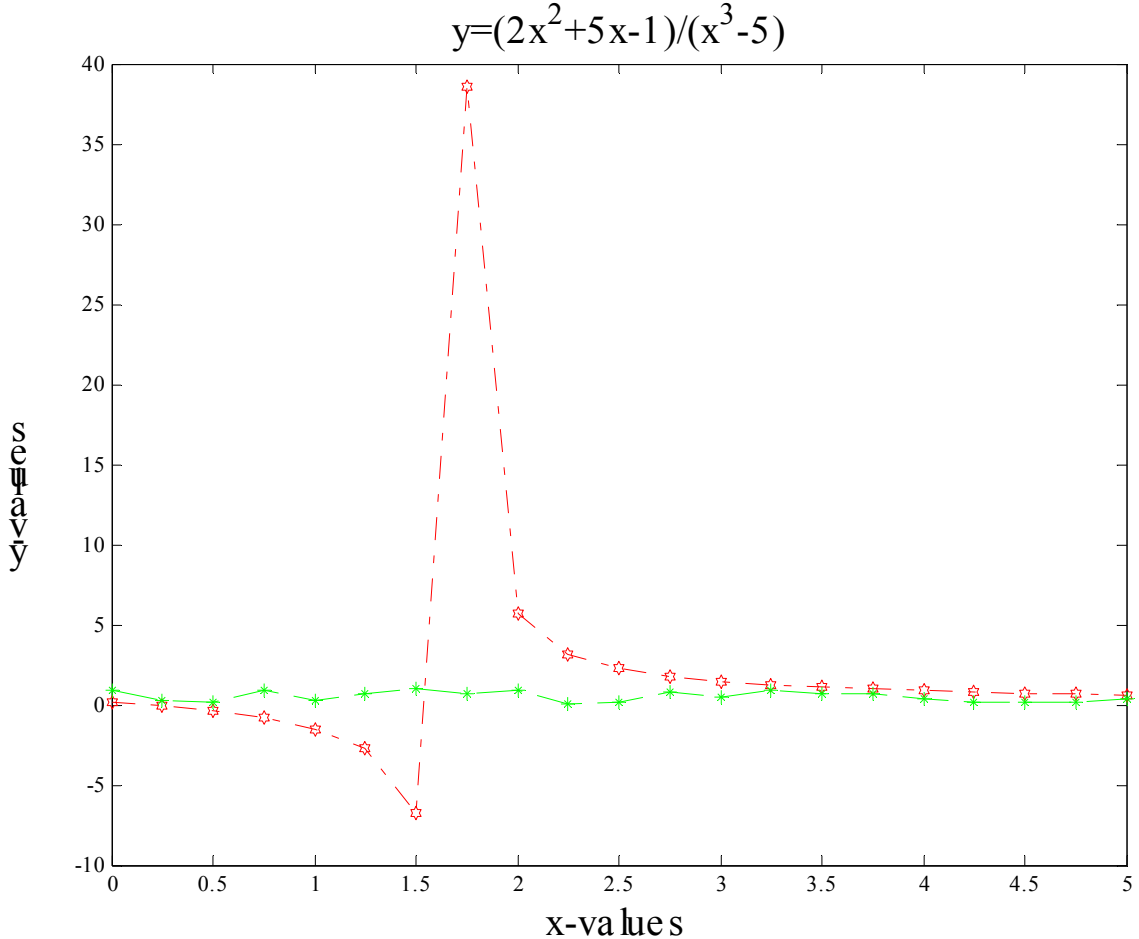
```
>> xlabel('x-values'); ylabel('y-values'); title('y=(2x^2+5x-1)/(x^3-5)');
```

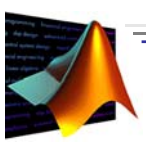




2D Examples

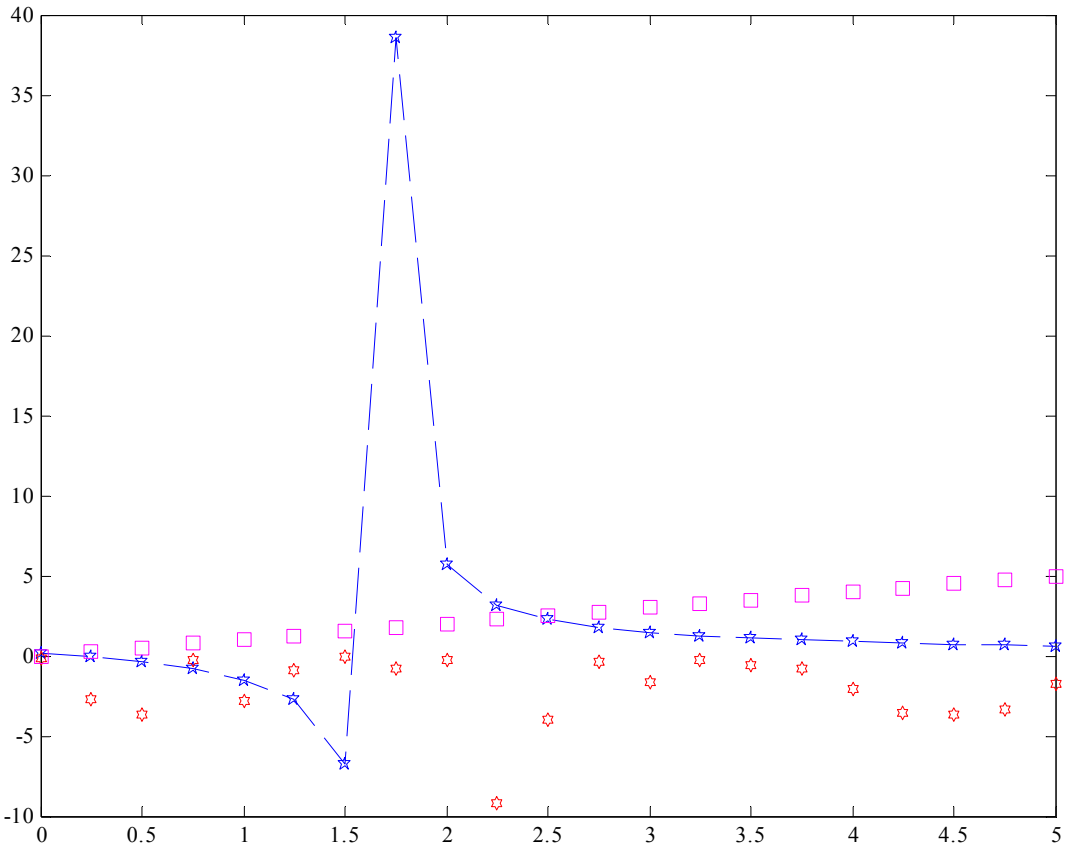
>> hold on; y1=rand(1,length(y)); plot(x,y1, 'g*-- ');

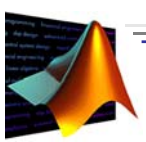




2D Examples

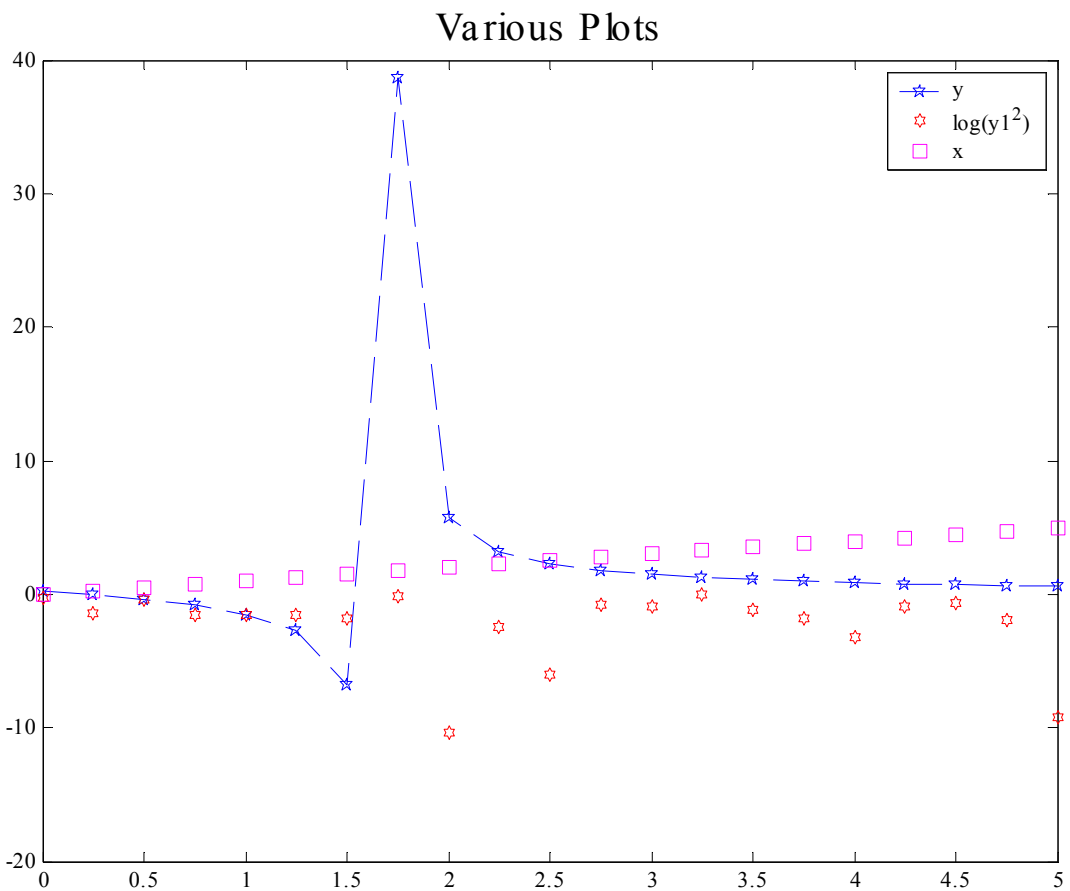
```
>> close all; plot(x, y, 'bp--', x, log(y1.^2), 'rh', x, x+zeros(1,length(x)), 'ms');
```

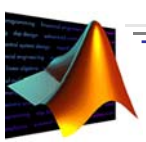




2D Examples

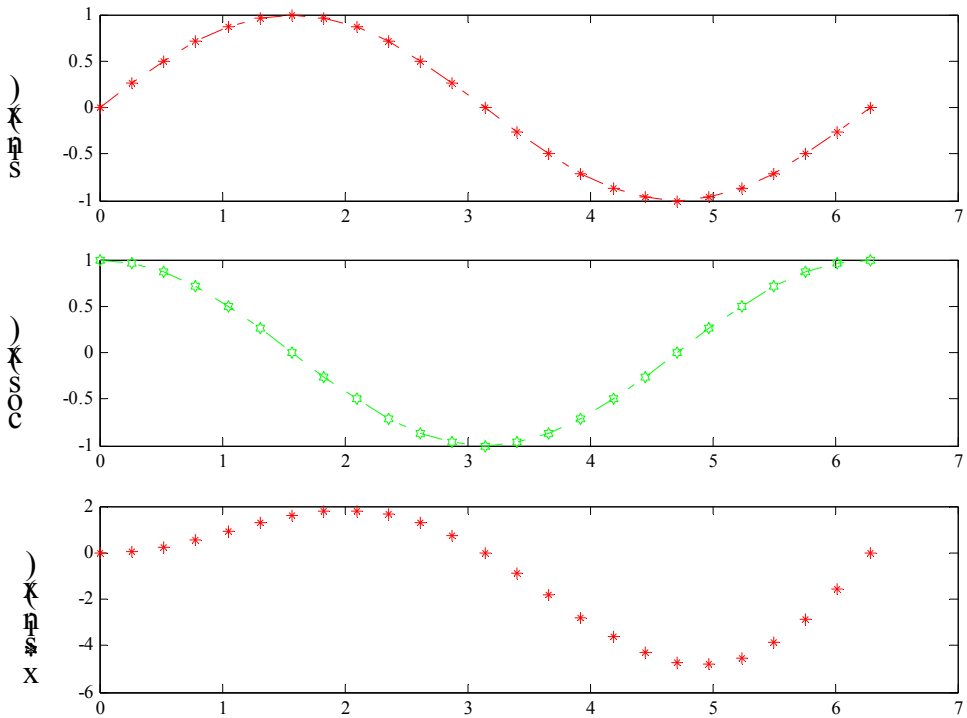
```
>> legend('y', 'log(y1^2)', 'x'); title('Various Plots');
```

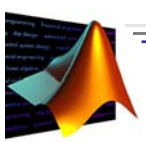




2D Examples

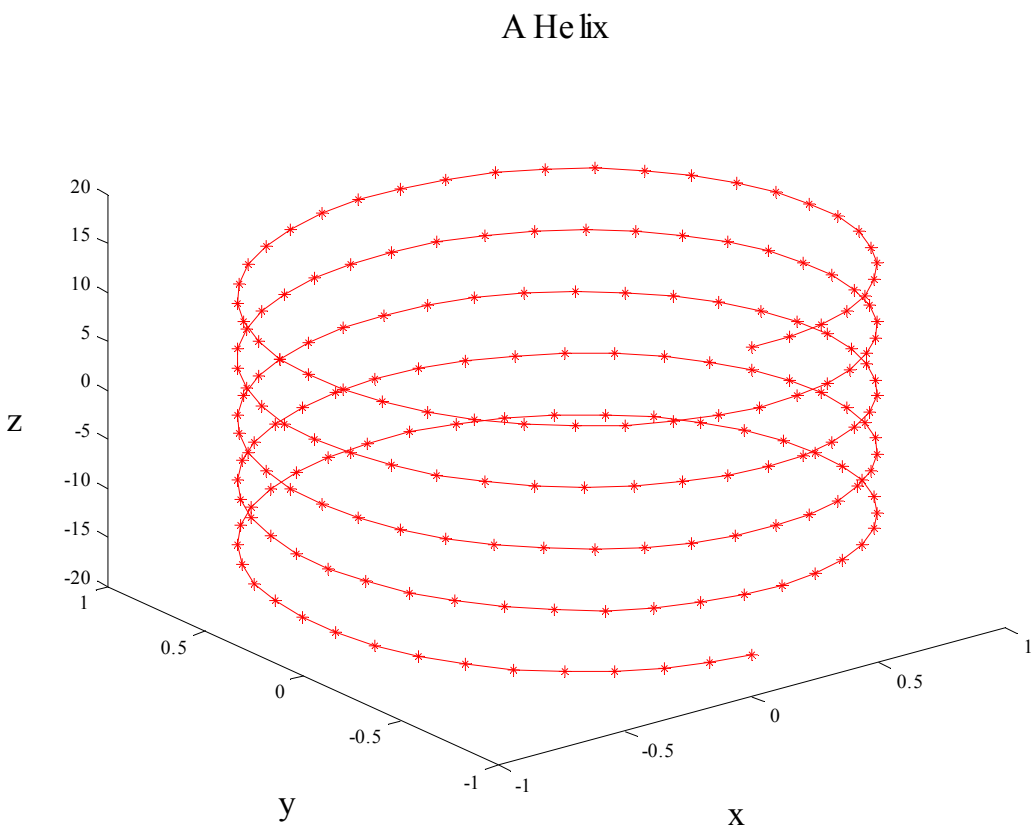
```
>> figure; x=linspace(0,2*pi,25);
>> subplot(3,1,1); plot(x, sin(x),'r*-.'); ylabel('sin(x)');
>> subplot(3,1,2); plot(x, cos(x),'gh-.'); ylabel('cos(x)');
>> subplot(3,1,3); plot(x, x.*sin(x), 'r*'); ylabel('x*sin(x)');
```

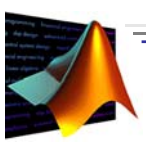




3D Line Plots

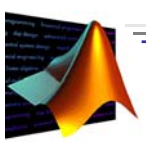
```
>> g=linspace(-5*pi,5*pi,200); plot3(sin(g), cos(g), g, 'r*-');
>> xlabel('x'); ylabel('y'); zlabel('z'); title('A helix');
```





3D Surface Graphs

- If it is needed to evaluate a bivariate function, ***f(x,y)***, at each (**x,y**) pair you should evaluate a value for ***f(.)***.
- To plot the surface, it is needed to create a grid of sample points (most preferable with high density) that cover the rectangular domain of the (**x, y**) plane in order to generate **X** and **Y** matrices consisting of repeated rows and columns, respectively, over the domain of the function. Then these matrices will be used to evaluate and graph the function.
- The **meshgrid** function transforms the domain specified by two vectors, **x** and **y**, into matrices, **X** and **Y**. You then use these matrices to evaluate functions of two variables. The rows of **X** are copies of the vector **x** and the columns of **Y** are copies of the vector **y**.



3D Surface Graphs

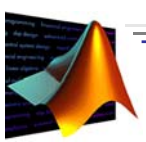
- Let's plot the peaks function with functional form:

$$Z = f(x,y) = 3(1-x)^2 e^{-x^2-(y+1)^2} - 10\left(\frac{x}{5} - x^3 - y^5\right)e^{-x^2-y^2} - \frac{1}{3}e^{-(x+1)^2-y^2}$$

- In the command window write:

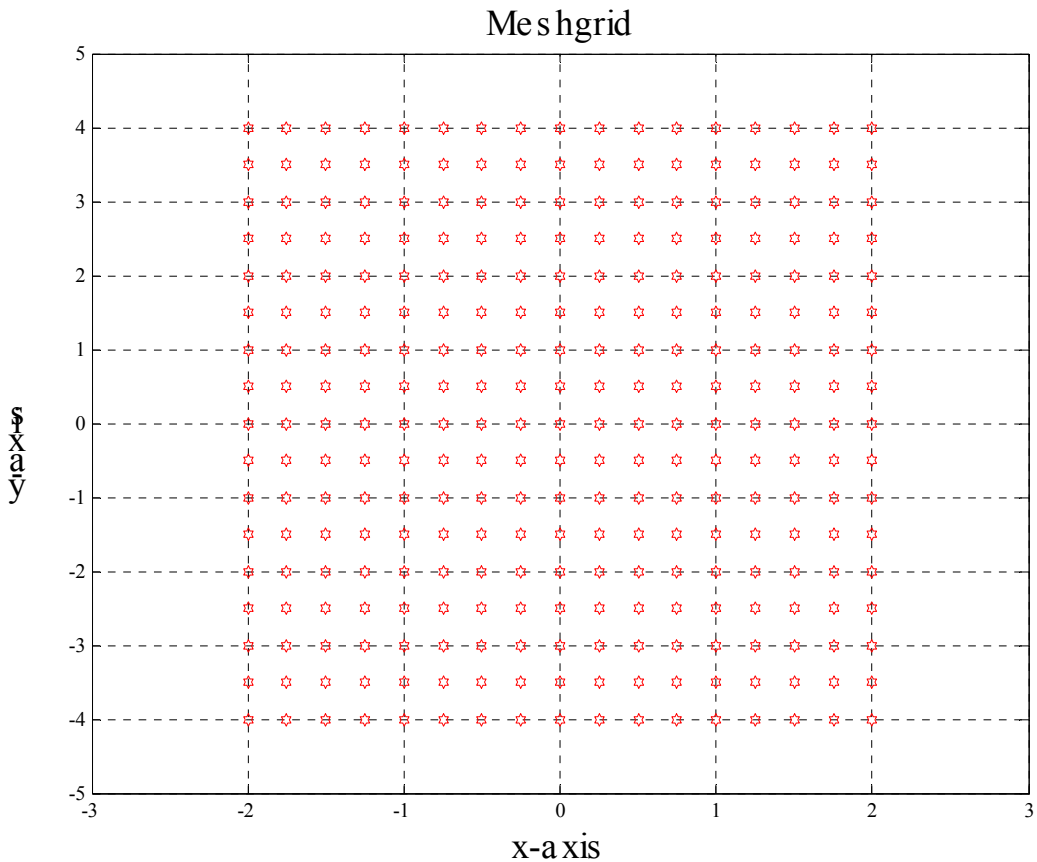
```
>> clear all; x=-2:0.25:2; y=-4:0.5:4; [X Y]=meshgrid(x,y);
>> plot(X,Y, 'rh'); axis([-3 3 -5 5]);
>> xlabel('x-axis'); ylabel('y-axis'); title('Meshgrid');
>> grid on;
```

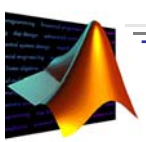
- View the plot in the next slide to understand the **meshgrid** functioning.



3D Surface Graphs - meshgrid

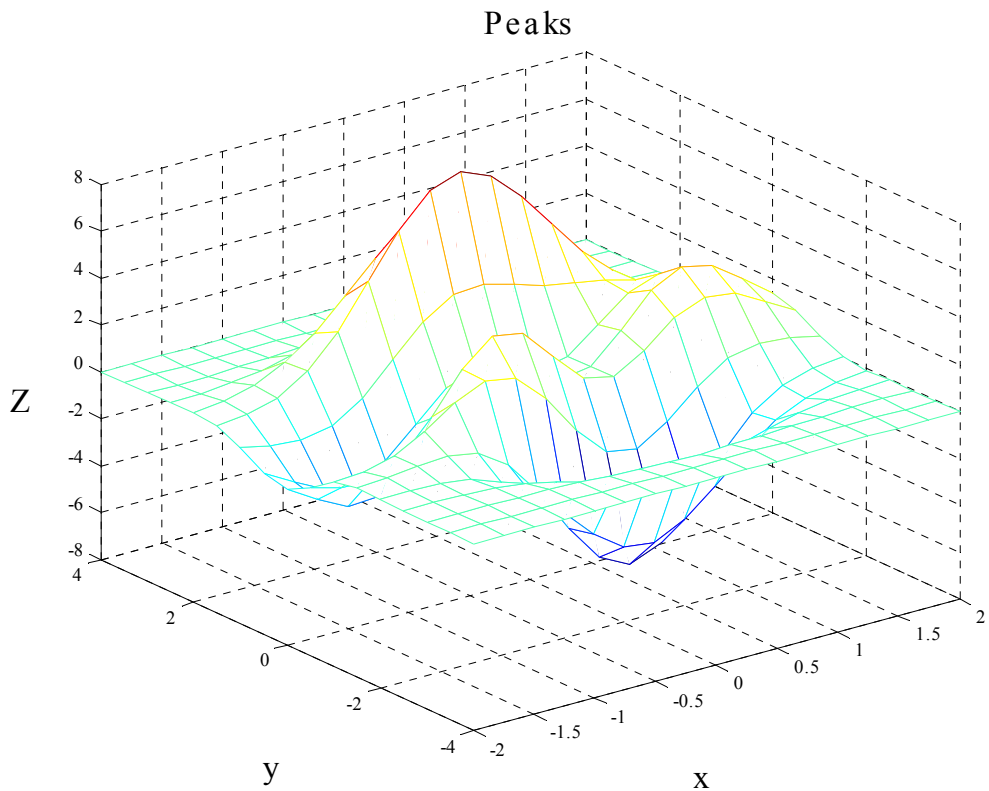
- Observe that **meshgrid** has sampled all possible interior points of the (**x, y**) plane.

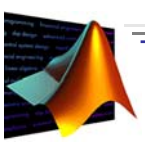




3D Surface Graphs - mesh

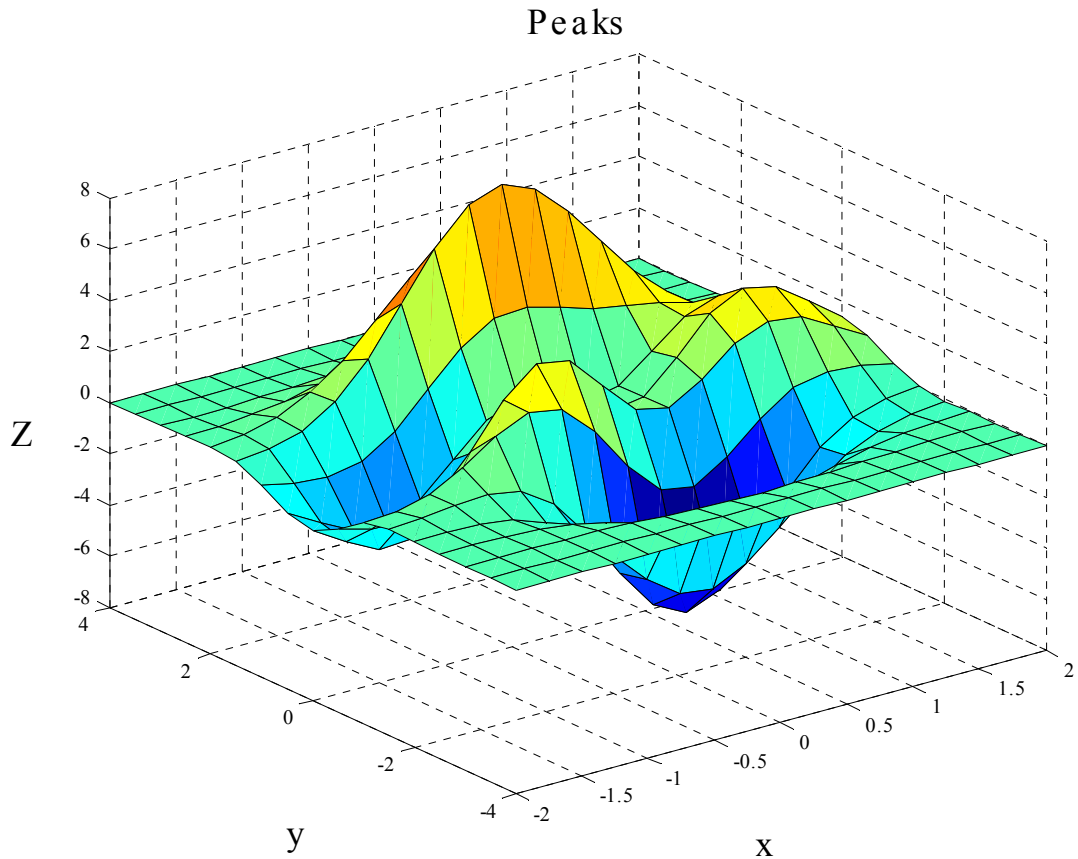
```
>> Z =3*(1-X).^2.*exp(-(X.^2) - (Y+1).^2)- 10*(X/5-X.^3- ...
        Y.^5).*exp(-X.^2-Y.^2)-1/3*exp(-(X+1).^2-Y.^2);
>> mesh(X, Y, Z); xlabel('x'); ylabel('y'); zlabel('Z'); title('Peaks');
```

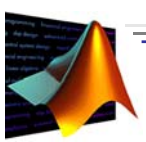




3D Surface Graphs - surf

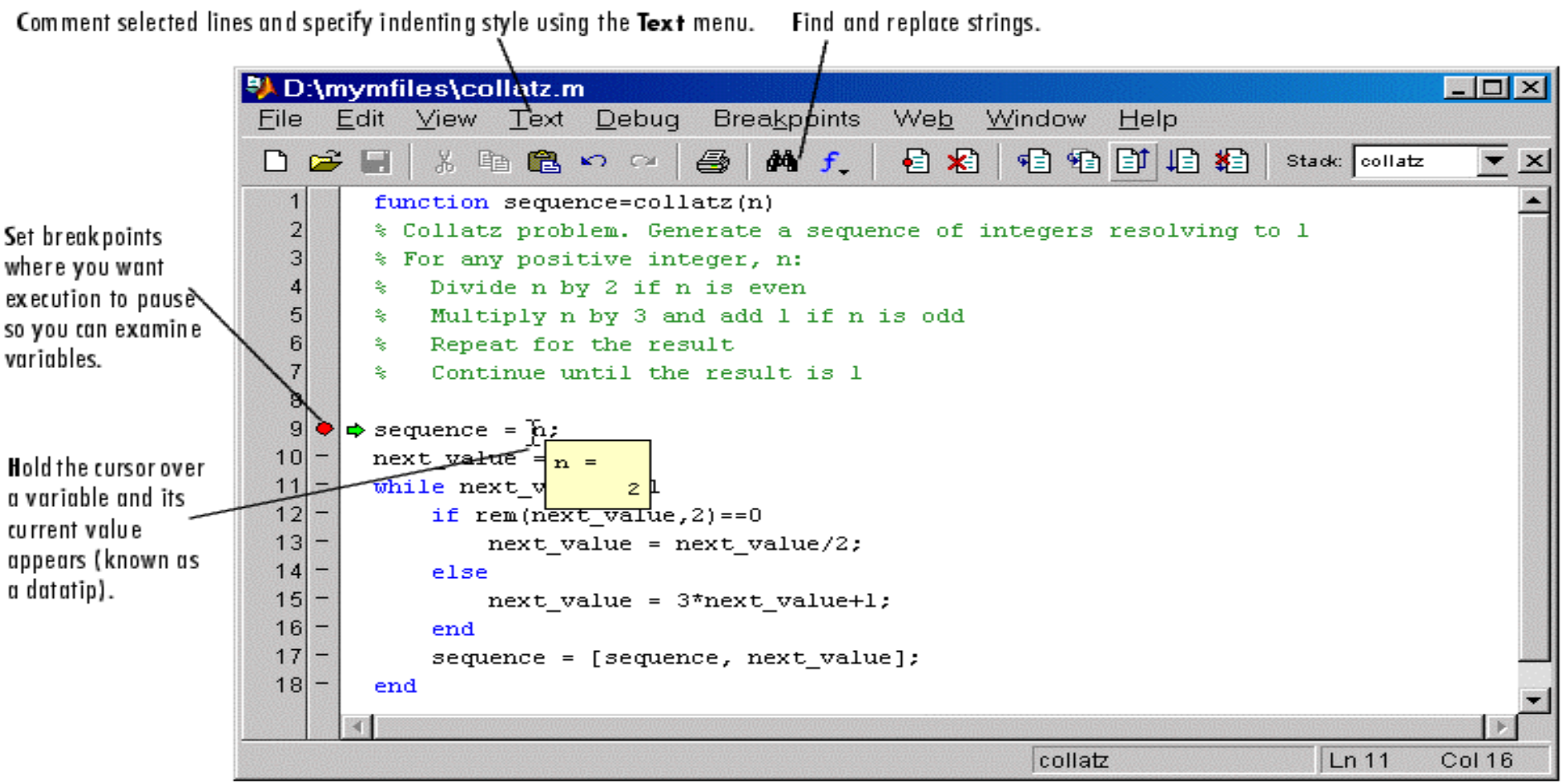
>> figure; mesh(X, Y, Z); xlabel('x'); ylabel('y'); zlabel('Z'); title('Peaks');



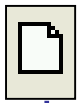


Matlab Editor/Debugger

- Use the *Editor/Debugger* to create and debug *m-files*, which are programs you write to run Matlab functions. To open the Editor/Debugger window go: ***File>New>M-file***



BSM with Editor/Debugger

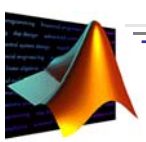
- Open the Editor/Debugger either via the ***File>New>M-File*** from the menu or by clicking the  icon on Matlab's desktop.
- Write the set of commands needed to:
  - Price call options for the following parameter values:

    S= 60:5:130, ***X***= 100, ***T***= 0.1, ***σ***= 0.25, ***r***= 0.05, ***δ***= 0.02
  - Plot the **call** values Vs ***S***
  - Make the 3D surface of **call** Vs ***S*** Vs ***T*** for the following ranges:

    S=80:2:120, ***T***= 0.1:0.02:0.3
- The resulting file is a ***script*** and it is saved a an *m-file* with a name (e.g. *pres_BSM.m*). To run the script either select: ***Debug>Save and Run*** or save it in a directory, go to the command window and at ">>" write its name without the ***.m*** extension (notice that the current directory should be the one that you save the file).

What is a Script?

- *Scripts* can operate on existing data in the workspace, or they can create new data on which to operate. Although scripts do not return output arguments, any variables that they create remain in the workspace, to be used in subsequent computations. *Scripts* are useful for automating a series of steps that are needed to be performed many times.
- A *script* has no a specific structure. It includes a number of commands that are serially executed. As long as the command series has a logical interpretation, the *script* will result to the desire output. Remember that a *script* does not take and does not return input and output arguments respectively. The vectors and matrices (variables or scalars) are stored in the Matlab's *workspace*.



C:\Documents and Settings\Administrator\My Documents\Matlab Examples\pres_BSM.m*

File Edit View Text Debug Breakpoints Web Window Help

Stack: Base

```
% This is a script that executes some commands related with Black-Scholes-Merton Model
clear all; clc; % Clearing the workspace and cleaning the screen

% Defining the BSM parameters
S=60:5:130; X=100; T=0.1; sig=0.25; r=0.05; div=0.02;

% Defining what is necessary to find the call value
dl=(log(S./X)+(r-div+(sig.^2)/2).*T)./(sig.*sqrt(T)); d2=dl-sig.*sqrt(T);
Ndl=normcdf(dl); Nd2=normcdf(d2);

% Calculating the European call value
Call=S.*exp(-div.*T).*Ndl-X.*exp(-r.*T).*Nd2

% Plotting the Call vs S
plot(S,Call,'rh-.');
% Giving labels and title to 2D plot
xlabel('S'); ylabel('Call'); title('Plot of a call option for values of S');

S=80:2:120; T=0.1:0.02:0.3; % Redifining S and T
[Ss, Tt]=meshgrid(S,T); % Creating the meshgrid matrix

% Defining what is necessary to find the call value
dl=(log(Ss./X)+(r-div+(sig.^2)/2).*Tt)./(sig.*sqrt(Tt)); d2=dl-sig.*sqrt(Tt);
Ndl=normcdf(dl); Nd2=normcdf(d2);

% Calculating the European call value
Call=Ss.*exp(-div.*Tt).*Ndl-X.*exp(-r.*Tt).*Nd2

% Creating the 3D surface in a new figure window
figure; surf(Ss,Tt,Call);

% Giving labels and title to the 3D graph
xlabel('S'); ylabel('T'); zlabel('Call'); title('Behaviour of a European call option for S and T values');

% To run this script go: Debug>Save and Run. Otherwise, save it to the desired directory, go to command window
% and type its name at ">>" (make sure that yours and Matlab's directory are the same)
```

- Script location and name
- * indicates that script is not saved

- Indicates that is a script

script Ln 1 Col 1

Relational and Logical Operators

- ***To compare various "quantities" (e.g. A=B) or to define a logical condition (X>2) Matlab offers the following alternatives (in order of precedence):***

Relational Operators:

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

Logical Operators:

not: ~

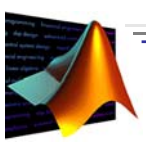
and: &

or: |

Logical Functions:

any(x), all(x)

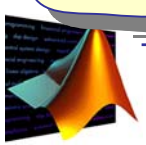
Logical Functions: Given that **x** is a vector, **any**(x) returns **1** if any element of **x** is nonzero and **all**(x) returns **1** if all elements of **x** are nonzero.



Examples - Relational and Logical Operators

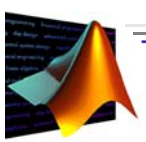
Input	Output
>> A=5; B=0; C=-5; D=[-1 0 1]; E=[-2 8 1];	Nothing is displayed
>>A>1, A>=10, A<5, C<=B, D(2)==B, D(3)~=8	ans=1, 0, 0, 1, 1, 1
>> D>=E	ans= 1 0 1
>> D==E	ans= 0 0 1
>>A1= ~A, A2= ~B, A3= ~D	A1= 0, A2= 1, A3= 0 1 0
>> A==B & C~=B, A4= -2 & 5	ans= 0, A4=1
>> ~D | (D~=E)	ans= 1 1 0
>> A5=any(D), A6=any(E)	A5=1, A6=1
>> A7=all(D), A8=all(E)	A7=0, A8=1

Finally all result to a Boolean expression that takes two values: TRUE (**1**) and FALSE (**0**). Always the comparisons are done element by element and the result is a scalar/vector/matrix of the same size with elements set to **1** where the relation is true and elements set to **0** where it is not.



Conditional Statements and Loops

- The relational and logical operators are very useful when it is needed to either execute a conditional statement or when a segment of commands are needed to be executed a number of times.
- A conditional statement is a segment of programming code that evaluates a statement; if the statement is TRUE then it executes some commands, otherwise if it is FALSE it runs a bulk of different programming code:
  - **Two most important**: ***if … end*** and ***switch … end***
- Loops can execute a bulk of commands as long as an expression is TRUE or for a specific number of time:
  - **Most important**: ***for .. end***, ***while … end***



The *if* Conditional Expression

Syntax	Example
if *logical expression* *programming code executed if TRUE* ***end***	flag=0; ***if*** ~flag disp('Hello') ***end***
if *logical expression* *programming code executed if TRUE* ***else*** *programming code executed if FALSE* ***end***	sales=5000; ***if*** sales<1000 Profit=sales*0.1; ***else*** Profit=(sales-1000)*0.2+1000*0.1 ***end***

if *logical expression #1*
programming code executed if TRUE #1
elseif *logical expression #2*
programming code executed if TRUE #2
else
programming code executed if FALSE
end
	if (sales>1000 & sales<=2000)
	disp('Low Sales ');
	Profit=sale*0.1
	elseif (sales>2000 & sales<=10000)
	disp('Medium Sales ');
	Profit=sales*0.15
	else
	disp('Satisfactory Sales ');
	Profit=sales*0.17
	end

The *switch* Conditional Statement

Syntax	Example
switch *expression* ***case*** *choice #1* *segment of executable programming code* ***case*** *choice #2* *segment of executable programming code* ***otherwise*** *segment of executable programming code* ***end***	dice=3; ***switch*** (dice) ***case*** 1 disp('One') ***case*** 2 disp('Two') ***case*** 3 disp('Three ') ***case*** 4 disp('Four') ***case*** 5 disp('Five') ***otherwise*** disp('Six') ***end***

It executes groups of statements based on the value of a variable or expression. Only the first matching case is executed. The expression following the case should be either a scalar or a string.

The *for* Loop

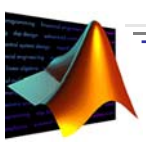
Syntax	Example
for *index=first_value**:**step**:**last_value* *segment of executable programming code* ***end***	***for*** *i=1:10* *disp(i)* ***end*** *sumj=0;* ***for*** *j=25:-2:-12* *sumj=sumj+j* ***end***

The colon notation is similar as in the case of the vectors. Actually, index in the for syntax is a vector with n elements with first element being the *first_value* and last the *last_value*. The difference between the index elements is *step*. If step is not displayed, then by default is set to 1.

The *while* Loop

Syntax	Example
while *expression* *segment of executable programming code* ***end***	*X=-3;* ***while*** *X<=10* *disp(X)* *X=X+1;* ***end***

The while loop repeats a group of statements an indefinite number of times under control of a loical condition. That is, as long as an expression is TRUE, then the segment of executable programming code that is included in the while statement is executed.



C:\Documents and Settings\Administrator\My Documents\Matlab Examples\pres_Conditionals.m

File Edit View Text Debug Breakpoints Web Window Help

Stack: Base

```
% This is a script that illustrates conditional statements

clear all; % Clearing the workspace
clc; % Cleaning the screen and send home the cursor

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% USER DEFINED VALUES
Sales=9125;
MarginAmount=[5000 10000]; % Amounts that change the commission rate
MarginComm=[0.125 0.175 0.225]; % Commissions related to MarginAmount
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Evaluating Sales to define Profit
if Sales <= MarginAmount(1)
    pnt=1; Profit= Sales * MarginComm(pnt);
elseif Sales > MarginAmount(1) & Sales <= MarginAmount(2)
    pnt=2; Profit= Sales * MarginComm(pnt);
else
    pnt=3; Profit= Sales * MarginComm(pnt);
end

% Printing on screen the related information
switch (pnt)
  case 1
      fprintf('Sales are %g, and profit with %g%% commission is $%g.', Sales, MarginComm(pnt)*100, Profit);
  case 2
      fprintf('Sales are %g, and profit with %g%% commission is $%g.', Sales, MarginComm(pnt)*100, Profit);
  otherwise
      fprintf('Sales are %g, and profit with %g%% commission is $%g.', Sales, MarginComm(pnt)*100, Profit);
end

% To run this script go: Debug>Save and Run. Otherwise, save it to the desired directory, go to command window
% and type its name at ">>" (make sure that yours and Matlab's directory are the same)
```

script Ln 1 Col 1

C:\Documents and Settings\Administrator\My Documents\Matlab Examples\pres_loops.m

File Edit View Text Debug Breakpoints Web Window Help

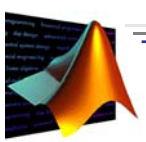
Stack: Base

```
% This is a script that illustrates nested loops and conditional statement use

clear all; clc % Clearing the workspace, cleaning the screen and send home the cursor

x=2*pi; temp=pi; % Defining two reference variables

while x<=4*pi % For the first time this is TRUE since 2*pi is LESS THAN 4*pi
    figure; hold on; % A new figure is created and it is hold also for each loop
    X=temp:pi/10:x; % Creation of a vector

    for j=1:4 % j takes the values 1, 2, 3 and 4 for each while loop
        subplot(2,2,j); plot(X,sin(j*X)./X, 'bp:'); % plotting figures

        switch (j) % switch is used to give titles to subplots
            case 1
                title('Plot of y=sin(x)/x');
            case 2
                title(' Plot of y=sin(2x)/x');
            case 3
                title(' Plot of y=sin(3x)/x');
            otherwise
                title(' Plot of y=sin(4x)/x');
         end % end of switch statement

         xlabel('x'); ylabel('y'); % Labeling the axes

     end % end of for statement

     temp=x; x=x+pi; % Assigning new values to reference variables

 end % end of while statement

% To run this script go: Debug>Save and Run. Otherwise, save it to the desired directory, go to command window
% and type its name at ">>" (make sure that yours and Matlab's directory are the same)
```

script Ln 1 Col 1

Functions

- They are *m-files* that can accept input arguments and return output arguments. The name of the *m-file* and of the function should be the same.
- They operate on variables within their own *workspace*, separate from the *workspace* you access at the Matlab command prompt.
- They are useful for extending the existing Matlab language for personal applications.
- Functions are included in *scripts* and have their own *calling syntax*.



Function's Structure

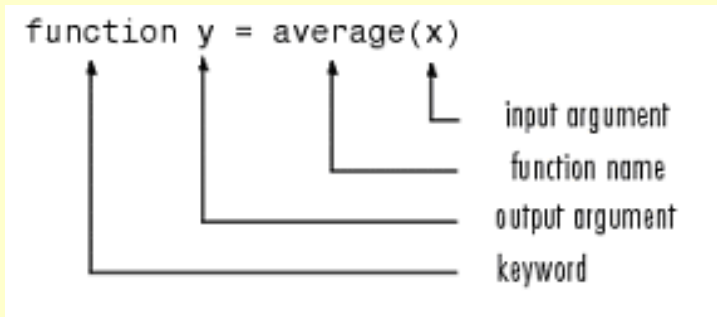
C:\Documents and Settings\Administrator\My Documents\Matlab Examples\pres_DiagExtract.m

File Edit View Text Debug Breakpoints Web Window Help

Stack: Base

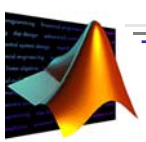
```
1  function [V]=pres_DiagExtract(X)
2
3
4
5
6
7
8
9
10
11
12 -
13
14 -
15 -
16 -
17 -
18 -
19 -
20 -
21 -
22 -
23 -
24 -
```

❖ **A functions has a very specific *calling structure (for example)*:**

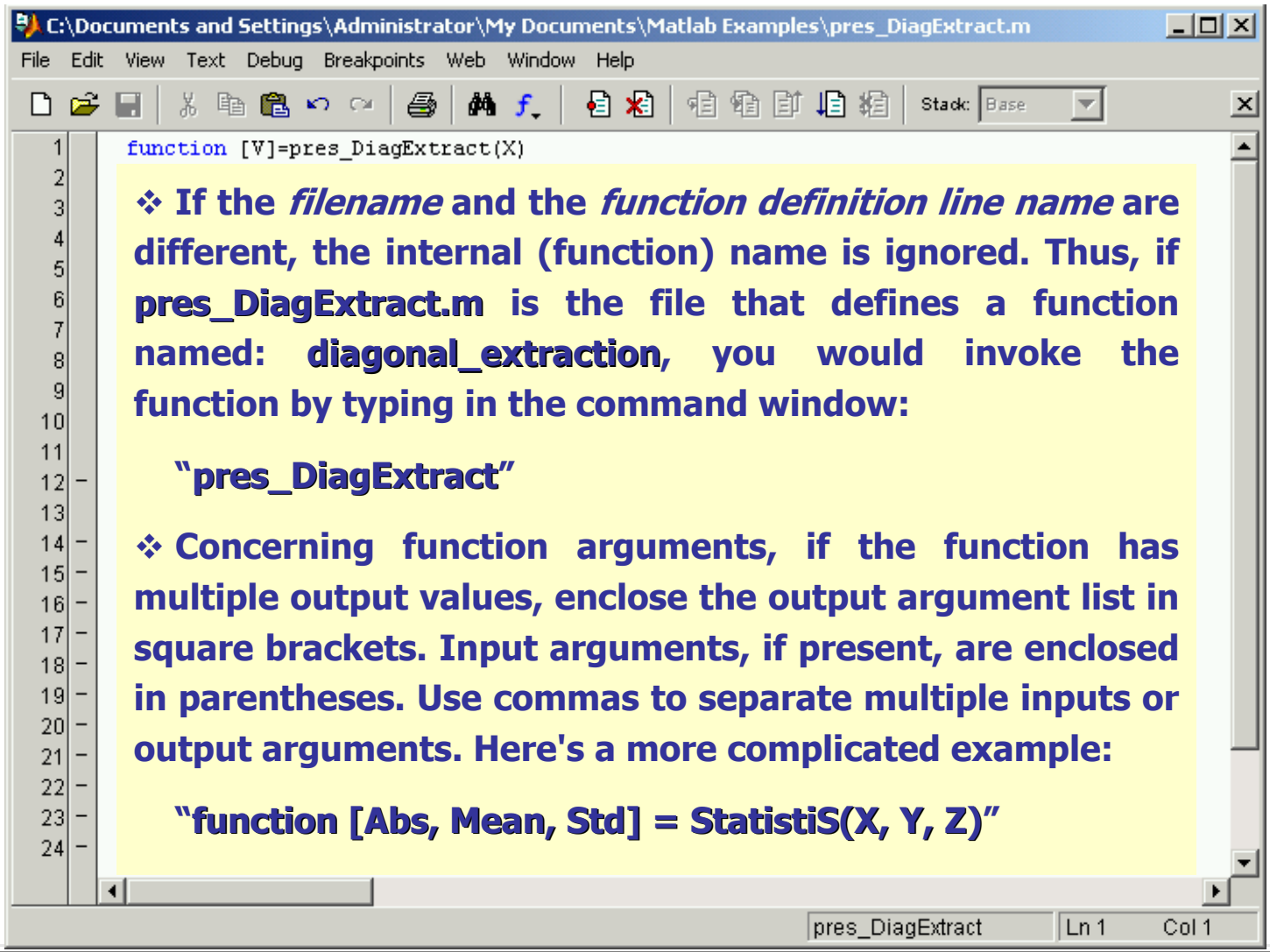


❖ **Matlab *function names* have the same constraints as *variable names*. The name must begin with a letter, which may be followed by any combination of letters, digits, and underscores.**

pres_DiagExtract Ln 1 Col 1



Function's Structure



Function's Structure

C:\Documents and Settings\Administrator\My Documents\Matlab Examples\pres_DiagExtract.m

File Edit View Text Debug Breakpoints Web Window Help

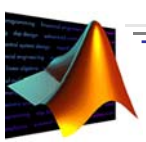
Stack: Base

```
1  function [V]=pres_DiagExtract(X)
2  %pres_DiagExtract: Extracts the diagonal elements of a square matrix
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
```

The H1 Line:

- **The H1 line, so named because it is the first help text line, is a comment line immediately following the function definition line. Because it consists of comment text, the H1 line begins with a percent sign, '%'.**
- **This is the first line of text that appears when a user types help "*function_name*" at the Matlab prompt. Further, the lookfor searches and displays only the H1 line of the functions. Because this line provides important summary information about the m-file, it is important to make it as descriptive as possible.**

pres_DiagExtract Ln 1 Col 1



Function's Structure

C:\Documents and Settings\Administrator\My Documents\Matlab Examples\pres_DiagExtract.m

File Edit View Text Debug Breakpoints Web Window Help

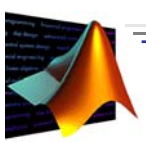
Stack: Base

```
1  function [V]=pres_DiagExtract(X)
2  %pres_DiagExtract: Extracts the diagonal elements of a square matrix
3  %   This function takes as input a square matrix named X and does the following:
4  %      1. If by mistake the user does not enter a square matrix, it returns
5  %      an error message
6  %      2. If a square matrix is correclty entered, then it returns the
7  %      elements of its main diagonal in the vector V
8
9  %   This function uses the error command that displays a message and aborts
10 %   the function's execution
11
12
13
14
15
16
17
18
19
20
21
22
23
24
```

pres_DiagExtract Ln 1 Col 1

The Help Text:

- **You can create online help for your *m-files* by entering text on one or more comment lines, beginning with the line immediately following the H1 line.**
- **When you type help *function_name*, Matlab displays the comment lines that appear between the function definition line and the first non-comment line. The help system ignores any comment lines that appear after this help block.**



Function's Structure

C:\Documents and Settings\Administrator\My Documents\Matlab Examples\pres_DiagExtract.m

File Edit View Text Debug Breakpoints Web Window Help

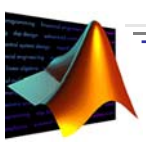
Stack: Base

```
1  function [V]=pres_DiagExtract(X)
2  %pres_DiagExtract: Extracts the diagonal elements of a square matri
3  %   This function takes as input a square matrix named X and does
4  %      1. If by mistake the user does not enter a square matrix,
5  %      an error message
6  %      2. If a square matrix is correclty entered, then it retu
7  %      elements of its main diagonal in the vector V
8
9  %   This function uses the error command that displays a message
10 %   the function's execution
11
12 [m n]=size(X); %Saving the size of the matrix
13
14 if isempty(X) %Checking if it is an empty matrix
15     error('You have not passed a square matrix');
16 elseif (m~=n) %Checking if matrix is square
17     error('You have enter a non square matrix');
18 elseif (m==n & m==1) %Checking if it is a single value
19     error('You have enter a single value');
20 else % Continue to extract the diagonal elements
21     for K=1:m
22         V(K)=X(K,K); %Extracting to V the diagonal elements
23     end
24 end
```

pres_DiagExtract

The Body Text:

- **The *function body* contains all the Matlab code that performs computations and assigns values to output arguments. The statements in the *function body* can consist of function calls, programming constructs like flow control and interactive input and output, calculations, assignments, comments, and blank lines.**



Function's Structure

C:\Documents and Settings\Administrator\My Documents\Matlab Examples\pres_DiagExtract.m

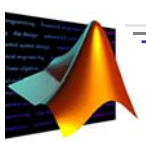
File Edit View Text Debug Breakpoints Web Window Help

Stack: Base

```
1   function [V]=pres_DiagExtract(X)
2   %pres_DiagExtract: Extracts the diagonal elements of a square matrix
3   %   This function takes as input a square matrix named X and does the following:
4   %      1. If by mistake the user does not enter a square matrix, it returns
5   %      an error message
6   %      2. If a square matrix is correclty entered, then it returns the
7   %      elements of its main diagonal in the vector V
8
9   %   This function uses the error command that displays a message and aborts
10  %   the function's execution
11
12  [m n]=size(X); %Saving the size of the matrix
13
14  if isempty(X) %Checking if it is an empty matrix
15      error('You have not passed a square matrix');
16  elseif (m~=n) %Checking if matrix is square
17      error('You have enter a non square matrix');
18  elseif (m==n & m==1) %Checking if it is a single value
19      error('You have enter a single value');
20  else % Continue to extract the diagonal elements
21      for K=1:m
22          V(K)=X(K,K); %Extracting to V the diagonal elements
23      end
24  end
```

pres_DiagExtract Ln 1 Col 1

- **Indicates that is a function**



The BSM Revisited: A *function*

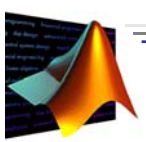
C:\Documents and Settings\Administrator\My Documents\Matlab Examples\pres_BSMprice.m

File Edit View Text Debug Breakpoints Web Window Help

Stack: Base

```
1   function [Call, Put]=pres_BSMprice(s,x,t,sig,r,dv)
2
3   %BSMprice Black-Scholes-Merton put and call pricing.
4   %   [Call, Put] = BSMprice(s,x,t,r,sig,dv) returns the value of call and a put
5   %   option(s) using the Black-Scholes-Merton pricing formula.
6   %   s is the current asset price, x is the exercise price, t is the time to maturity
7   %   of the option in years, r is the risk-free interest rate, sig is the standard
8   %   deviation of the annualized continuously compounded rate of return of the asset
9   %   (also known as volatility), and dv is the dividend rate of the asset.
10
11
12  % Calculating dl definition with dot operations
13  dl = (log(s./x)+(r-dv+(sig.^2)/2).*t)./(sig.*sqrt(t));
14
15  % Calculating d2 definition with dot operations
16  d2 = dl - (sig.*sqrt(t));
17
18  % Calculating the BSM European call option value
19  Call = s.*exp(-dv.*t).*normcdf(dl)-x.*(exp(-r.*t).*normcdf(d2));
20
21  % Calculating the BSM European put option value
22  Put=x.*exp(-r.*t).*normcdf(-d2)-s.*exp(-dv.*t).*normcdf(-dl);
```

pres_BSMprice Ln 1 Col 1



The BSM Revisited: The *script*

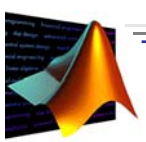
C:\Documents and Settings\Administrator\My Documents\Matlab Examples\pres_BSM_fun.m

File Edit View Text Debug Breakpoints Web Window Help

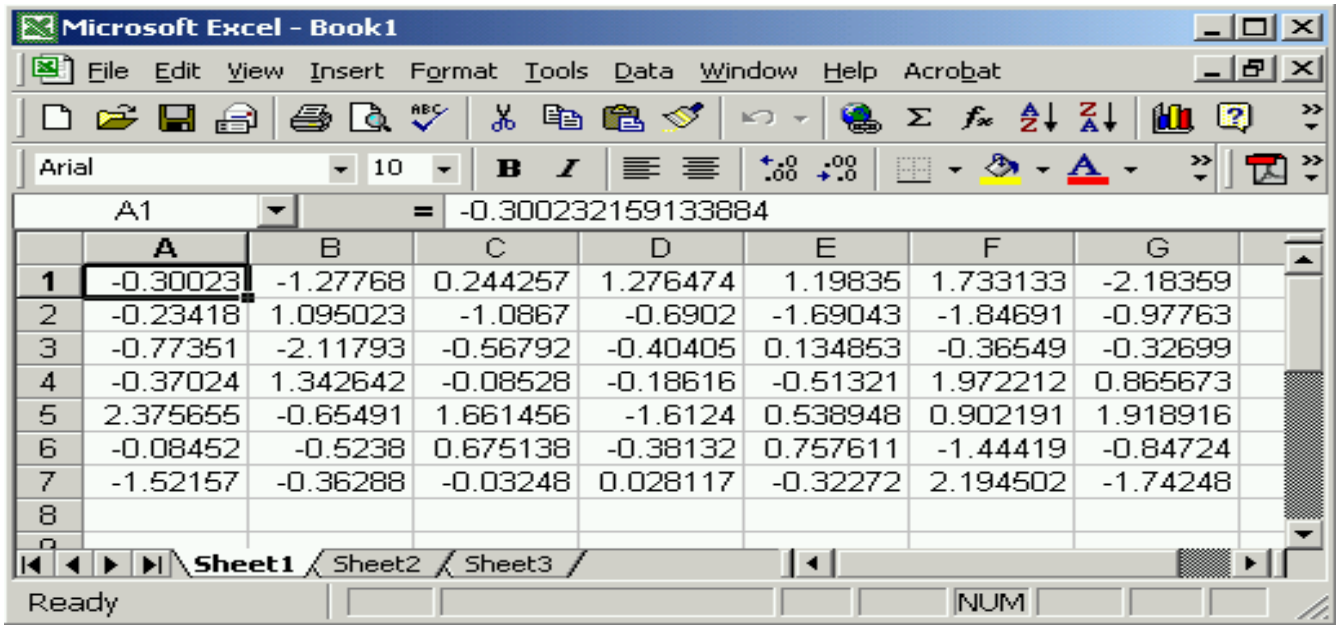
Stack: Base

```
1  % This is a script that executes some commands related with Black-Scholes-Merton Model
2  clear all; clc; % Clearing the workspace and cleaning the screen
3
4  % Defining the BSM parameters
5  S=60:5:130; X=100; T=0.1; sig=0.25; r=0.05; div=0.02;
6  % Calling a function to find call and put values based on BSM
7  [Call Put]=pres_BSMprice(S,X,T,sig,r,div);
8  % Plotting the Call vs S
9  plot(S,Call,'rh-.');
10 % Plotting the Put vs S on the same plot
11 hold on; plot(S,Put,'b*--');
12 % Giving labels, title and legend to 2D plot
13 xlabel('S'); ylabel('Call and Put Values');
14 title('Plot of a call and put option for values of S'); legend('Call', 'Put')
15
16 S=80:2:120; T=0.1:0.02:0.3; % Redifining S and T
17
18 [Ss, Tt]=meshgrid(S,T); % Creating the meshgrid matrix
19
20 [CallNew]=pres_BSMprice(Ss,X,Tt,sig,r,div); % Requesting call values only
21 % Creating the 3D surface in a new figure window
22 figure; surf(Ss,Tt,CallNew);
23 % Giving labels and title to the 3D graph
24 xlabel('S'); ylabel('T'); zlabel('Call');
25 title('Behaviour of a European call option for S and T values');
```

script Ln 17 Col 1



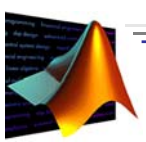
Loading ASCII Files of Data

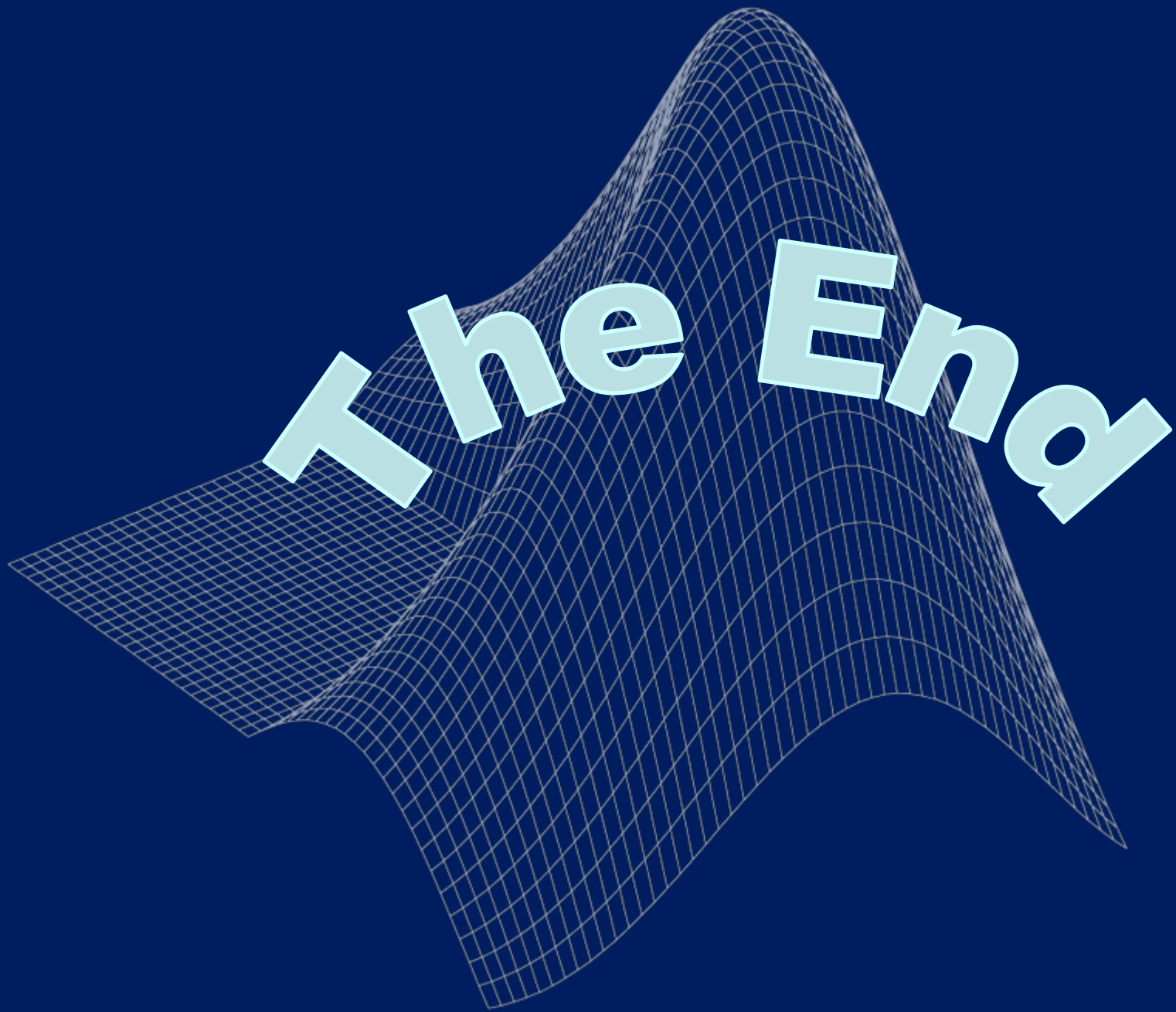


	A	B	C	D	E	F	G
1	-0.30023	-1.27768	0.244257	1.276474	1.19835	1.733133	-2.18359
2	-0.23418	1.095023	-1.0867	-0.6902	-1.69043	-1.84691	-0.97763
3	-0.77351	-2.11793	-0.56792	-0.40405	0.134853	-0.36549	-0.32699
4	-0.37024	1.342642	-0.08528	-0.18616	-0.51321	1.972212	0.865673
5	2.375655	-0.65491	1.661456	-1.6124	0.538948	0.902191	1.918916
6	-0.08452	-0.5238	0.675138	-0.38132	0.757611	-1.44419	-0.84724
7	-1.52157	-0.36288	-0.03248	0.028117	-0.32272	2.194502	-1.74248
8							

Data that is saved in a text format can be loaded in the Matlab's workspace with the load command. Readable text data do not contain any text (only numerical data) and all columns and rows are completely filled. Internet data saved in an ASCII form is similar with the ones shown on the above spreadsheet. The general calling syntax is:

DataMatrix =load ('*filename*')




The End