

# BINF733 Introduction to MATLAB

Jeff Solka and Jennifer Weller

1

## INTRODUCTION TO MATLAB

### What is MATLAB?

- § Computing Platforms and Toolboxes
- § The Five Parts of MATLAB
- § Notes and Conventions

### Getting Started

- § The MATLAB Desktop
- § Desktop Features summary slide page numbers
- § Desktop Windows
- § Recording MATLAB Sessions
- § File Management

### The MATLAB Workspace

- § The Workspace Browser
- § Simple Arithmetic
- § Working with Variables
- § Saving and Retrieving Data

### Script Files

### MATLAB Objects

### Array and Matrices

- § Addressing
- § Construction
- § Cell Arrays & Structures
- § Operations

2

## WHAT IS MATLAB?

- § **MATLAB is an integrated computing environment for numeric computation and visualization.**
- § **It integrates numerical analysis, matrix computation, signal processing and graphics.**
- § **MATLAB is an interactive system whose basic data element is a matrix.**
- § **Besides common matrix operations, it offers programming features similar to those of other computer languages; e.g., functions, if statements.**
- § **MATLAB offers high-level graphics functions to visualize data.**
- § **The MATLAB Handle Graphics functions give users the ability to manipulate graphics in many ways.**
- § **MATLAB provides GUI tools so the user can develop applications.**
- § **There have been many versions, i.e., releases, of MATLAB over the years. The most current is release 13. Subsequent releases have increased MATLAB's capabilities and features, and have substantially changed the user's interface. However, the fundamentals have remained basically the same and the same technical philosophy has been maintained.**

3

## COMPUTING PLATFORMS & TOOL BOXES

The MATLAB development environment is available for the following

Operating systems

- § Microsoft - Windows
- § Macintosh
- § UNIX / Linux

There are many specialized MATLAB toolboxes by which MATLAB's capabilities can be extended and customized. Some of these are:

- § Signal Processing Toolbox
- § Communications Toolbox
- § Control System Toolbox
- § System Identification Toolbox
- § Optimization Toolbox
- § Neural Network Toolbox
- § Spline Toolbox
- § Robust-Control Toolbox
- § Statistics Toolbox
- § Symbolic Toolbox
- § Image Processing Toolbox
- § Fuzzy Logic
- § Financial
- § PDEs
- § Wavelet

4

Toolboxes are available from the Mathworks and from

# THE FIVE PARTS OF MATLAB

## 1. The Development Environment

The development environment is all the components that help you use MATLAB functions, see your results, and interact with data from other sources. Many of these components have graphical user interfaces that help make their use easier. Most of these are accessible through the MATLAB desktop.

## 2. The Mathematical Function Library

MATLAB contains a vast collection computational algorithms that encompass simple functions like sine, cosine, etc., as well as sophisticated functions for matrix computation and manipulation, statistical analysis, and signal processing.

## 3. The MATLAB Language

At the heart of MATLAB is a high-level programming language with the matrix as its fundamental entity. All the usual programming features such as flow-control, input-output, and object-oriented features are available. The language is so flexible that it can be used almost effortlessly for quick computations, or much more formally for complex programming tasks including special purpose GUI based applications.

5

# THE MATLAB DESKTOP

The MATLAB Desktop provides management for the following Desktop

## Tools:

Command Window	Where you can enter and run MATLAB functions.
Command History	Maintains a log of the functions you entered in the Command Window.
Launch Pad	Lets you run tools and access documentation for all your MathWorks products.
Current Directory Browser	Where you can view MATLAB files and perform high-level file operations such as open, and find content.
Help Browser	Lets you view and search MATLAB documentation.
Workspace Browser	Lets you view and alter the contents of the MATLAB Workspace.
Array Editor	Where you can view array contents in a table format and edit individual elements.
Editor / Debugger	Here you can create, edit, and debug files containing MATLAB functions.

6

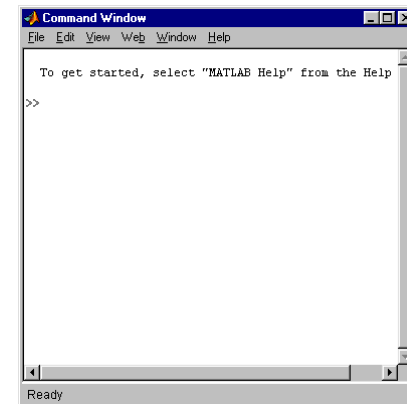
## MATLAB Websites

- § Accessing MathWorks' Web Sites
  - § The Desktop provides access to many MathWorks web pages.
  - § <http://www.mathworks.com> for the MathWorks home page.
  - § <http://www.mathworks.com/support> for MATLAB support.
  - § <http://www.mathworks.com/products> for product information.
  - § <http://www.mathworks.com/mla/index.shtml> Access page for Access members. Join to become a member and get frequent updates on the latest MATLAB developments.

7

## THE COMMAND WINDOW

The Command Window is the primary method by which you will communicate with MATLAB. Here you will run MATLAB functions and see their raw results.

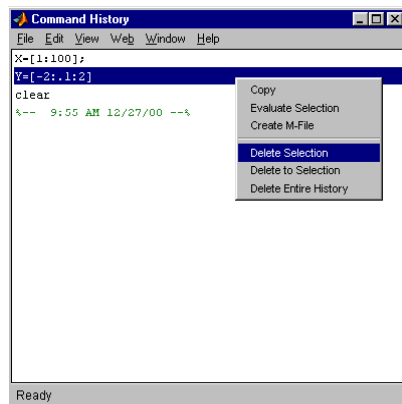


- § You will become very familiar with the Command window during your study of MATLAB!

8

## THE COMMAND HISTORY WINDOW

The Command History window displays a log of the functions most recently run in the Command Window.



- § The time and date for each session is logged.
- § The scroll bar lets you travel through the history.
- § All entries remain in the history until you delete them.
- § Double clicking on a command in the history will run that command.
- § You can copy and paste from the Command History to other tools.
- § Right-clicking a selection will open a list of operations.

9

## GETTING HELP IN MATLAB

There is a `help` command you can invoke in the command window:

- § `help 'function'` returns help information for a specific function.
- § `help general` returns a list of general topics to the command window.
- § Typing `help` with no argument returns a list of topics.
- § `helpwin` opens the Help Browser with a list of default help topics.
- § `lookfor topic` lets you look for key words in the help files.
- § Typing `helpdesk` in the command window in MATLAB 6 starts the **Help Browser** at the "Begin Here" page.
- § `info` Shows you how to contact the Mathworks.
- § `whatsnew` Starts the Help Browser at the "what's new" page.

10

## RECORDING YOUR MATLAB SESSIONS

Another method to record your sessions is by using the `diary` command.

The MATLAB `diary` command gives you a convenient way of recording all your keyboard entries and what they return, storing them to a file.

- § `diary` causes a copy of all subsequent terminal input and most of the resulting output to be written to the named file.
- § This will keep a record of your entire session on the A: disk drive in the file named "day1.txt".

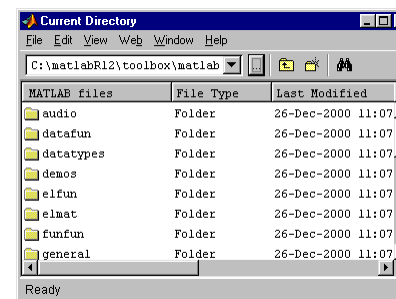
```
> diary c:/day1.txt
```

- § If no file is specified, the filename 'diary' is used.
- § `diary off` suspends the diary function.
- § `diary on` turns it back on.
- § `diary` by itself toggles the diary state.

11

## FILE MANAGEMENT

Earlier you briefly saw how the Current Directory Browser can be used to view and modify the search path and see all files.



You can also examine and modify the MATLAB search path from the command line.

- `path` Displays the search path.
- `path(pathstring)` Changes the search path to `pathstring`.
- `path(path,dirstring)` Appends `dirstring` to the current path.

12

## FILE MANAGEMENT

MATLAB uses a *search path* to find *script* and function *M-files*.

There are several file management commands that you can use to list, view and delete M-files.

- § `dir, ls` Show files in present directory
- § `delete filename` Delete file *filename*.
- § `cd, pwd` Show present directory.
- § `cd dir, chdir` Change directory.
- § `type filename` lists the contents of the file
- § `which filename` Displays the path to file *filename*.

When you enter a variable at the command line MATLAB does the following:

- § It checks to see if it is a variable in the MATLAB workspace.
- § It checks to see if it is a built-in function.
- § It checks to see if it is an *M-file* existing in the current directory.
- § It checks to see if it exists anywhere in the *search path*.

13

## THE MATLAB WORKSPACE

In the command window, MATLAB remembers the commands you enter and all the values of any variables you create. You can see the history of your commands in the Command History window. However, the variables you create and their values *live* in the *MATLAB Workspace*.

The Workspace Browser lets you view the names of the variables that are in the workspace, how much memory they use, what class of variable they are, and allows you to modify them.

You can also save the current workspace for later use, or load a previously saved workspace.

•Right-clicking on a variable will reveal a list of operations you can perform on that variable from the browser

14

## THE MATLAB WORKSPACE

Once you create a variable, it is saved in the MATLAB workspace and you can recall it at any time by typing in the variable name.

- § These shortcut keystrokes are useful to quickly recall and edit MATLAB commands and variables in the MATLAB workspace.
- § The ↑ key recalls the most recent command.
- § The ↓ key scrolls forward through commands
- § The ← key moves left through the present command.
- § The → key moves right through the present command.

The following commands will help you with finding out what is in your directory and your workspace:

- § who Lists all variables in the workspace
- § whos Same as above, but provides size and dimension of objects
- § what Lists the M-files and .mat files that are in the current directory.

The following commands are used to remove objects.

- § clear Removes objects from the workspace.

15

## VARIABLES IN THE MATLAB WORKSPACE

Variable names must start with a letter, followed by any number of letters, digits, or underscores.

- § MATLAB will only use the first 31 characters of a variable name.
- § MATLAB is case sensitive so each of these denote a different variable:

Temp, temp, TEMP

**MATLAB will echo the value of the variable in an assignment statement unless you end it with a semi-colon ";".**

**All values are long format, i.e., double precision IEEE floating point standard, internally to MATLAB regardless of the resolution shown in the Command Window.**

- § Floating-point numbers have a precision of approximately 16 significant decimal digits and a finite range of roughly  $10^{-308}$  to  $10^{308}$ .

16

## ARITHMETIC OPERATIONS IN MATLAB

You can do simple math with MATLAB, as you would with a calculator.

### Arithmetic Operations:

addition:	$a + b$
subtraction:	$a - b$
multiplication:	$a*b$
division:	$a/b = a \setminus b$
power:	$a^b$

Expressions can be assigned to a variable using the "=" sign,

```
temp = a + b * c / d
```

Expressions are evaluated left to right with the following order of precedence:

- § Power Operation
- § Multiplication and Division
- § Addition and Subtraction

Parentheses ( ) can be used to change this ordering.

17

## SAVING DATA

You can save your workspace or variables in several ways.

- § Use the Windows **F**ile menu command Save Workspace **A**s..
- § Or use save at the command line.

```
>> save %Save all variables in matlab.mat
```

```
>> save data %Save all variables in data.mat
```

```
>> save data temp %Saves the variable temp  
>> % in data.mat
```

```
>> save data temp -ascii %Saves as ascii 8bit
```

```
>> save data temp -ascii -double % ascii 16bit
```

- § The following command saves with tab delimiters:

```
>> save data temp -ascii -double -tabs
```

18

## RETRIEVING DATA

To retrieve your data at another session, use:

```
>> load          % loads matlab.mat

>> load data    % loads the file
data.mat

>> load data -ascii % loads the ascii
file data
```

19

## VARIABLE NAMING RULES

Variable names can have up to 31 characters and must start with a letter followed by letters, digits or underscores and contain no punctuation characters.

§ Special Variables in MATLAB:

ans	Default value for results
pi	$\pi$
eps	small number
flops	floating point operations
inf	infinity...1/0
NaN	Not-a-Number...0/0
i, j	$\sqrt{-1}$
nargout	number of output arguments
nargin	number of input arguments
realmin	smallest usable real number
realmax	largest usable real number

§ Don't name a variable after one of these special variables or a MATLAB command/function if at all possible. If you do its prior value is overwritten.

```
>> a = pi;
>> pi = 42;
>> b = pi;
>> a,b
a =
    3.1416
b =
    42
```

20

## COMMON FUNCTIONS IN MATLAB

**MATLAB provides many common functions that will operate on scalars, arrays or matrices.**

- § Examples:
  - § Trigonometry functions
  - § Square root
  - § Logarithms, Exponential
  - § Rounding, Truncating
  
- § These are used the same way you would mathematically.
  
- § Some of these functions are written as .m files. You can read them with any text editor.
  
- § Pages 16-20 of your *Mastering MATLAB 6* by Hanselman list many of the common mathematical functions.

21

## SCRIPT FILES

**You can do MATLAB sessions interactively by entering commands at the command line, or...**

**You can use an *M-file*, which is a text file ending with ".m" and is often called a script file to execute commands.**

- § When to do this:
  - § If you have a lot of MATLAB commands
  - § If you want to keep a record of commands
  - § If you need to repeat the same series of commands

**Write the commands in a text file and save it with the extension ".m".**

**You can use the File menu to open a new ".m" file.**

**To run your file, simply enter the name of the file - without the ".m". Or run your file by choosing Run Script from the File menu.**

22

## SCRIPT FILES

These are some commands to use in your script files to make them more interactive and easier to debug.

**disp(variable) :** Display result without identifying variable names.

In all other ways it's the same as leaving the semicolon off an expression except that empty arrays don't display.

**echo:** Turns on or off echoing of commands inside Script files.

§ `echo on` turns on echoing

§ `echo off` turns off echoing.

§ `echo file on` where *file* is a function name causes the named function-file to be echoed when it is used.

§ `echo file off` turns it off.

§ `echo file` toggles it.

§ `echo on all` turns on the echoing of commands inside any function-files that are currently in memory (i.e., the functions returned by `inmem`). 23

§ `echo off all` turns them all off.

## SCRIPT FILES

**input:** Prompts the user to provide input from the keyboard.

§ `X = input('text string')` displays to the user the prompt in the text string and then waits for a response from the keyboard. The response can be any MATLAB expression, which is evaluated, using the variables in the current workspace, and the result is returned in X. If the user presses the return key without entering anything, `input` returns an empty matrix.

§ `X = INPUT('text string','s')` gives the prompt in the text string and waits for character string input. The typed input is not evaluated; the characters are simply returned as a MATLAB string.

§ The text string for the prompt may contain one or more '\n'. The '\n' means skip to the beginning of the next line. This allows the prompt string to span several lines. To output just a '\ ' use '\\ '.

24

## SCRIPT FILES

**keyboard:** Stops execution of the file and gives control to the user's keyboard. Variables may be examined or changed - all MATLAB commands are valid.

§ indicated by a `K` appearing before the prompt.

§ The keyboard mode is terminated by executing the command `return` by typing the six letters `r-e-t-u-r-n` and pressing the return key. Control then returns to the invoking M-file.

§ `dbquit` can also be used to get out of keyboard mode but in this case the invoking M-file is terminated.

§ The keyboard mode is useful for debugging your M-files.

**pause:** Causes a procedure to wait until the user presses any keyboard key before continuing.

**pause:**

§ `pause (n)` pauses for `n` seconds before continuing, where `n` can also be a fraction. Although the resolution of the clock is platform specific, fractional pauses of 0.01 seconds should be supported on most platforms.

§ `pause off` indicates that any subsequent `pause` or `pause(n)` commands should not actually pause. This allows normally interactive scripts to run unattended.

§ `pause on` indicates that subsequent `pause` commands should pause.<sup>25</sup>

## SCRIPT FILES

**waitforbuttonpress:** Stops program execution (pauses) until a key or mouse button is pressed over a figure window.

§ Returns 0 when terminated by a mouse buttonpress

§ Returns 1 when terminated by a keypress.

26

## TYPES OF DATA OBJECTS IN MATLAB

### Matrices

- § A matrix is a rectangular array of numbers. Matrices are the basic elements of MATLAB, so there is no need to handle these as you would in C-programming.
- § You can operate on these objects as you would in mathematics or on paper. MATLAB adheres strictly to the conventions and rules of linear algebra:
- § For example:
  - §  $a$  is an  $n \times 1$  vector
  - §  $b$  is a  $1 \times n$  vector
  - §  $C = a*b$  is an  $n \times n$  matrix, the outer product
  - §  $c=b*a$  is a scalar, the inner product
- § There is NO need to loop through all of the elements in the matrix to perform operations. MATLAB works with matrices as entities.
- § Scalars are matrices with only a single element.
- § Vectors are matrices with only one row or one column.

### Arrays:

- § Column vector
- § Row vector

27

## OTHER DATA CONSTRUCTS

### Beginning in MATLAB 5 new data constructs were introduced

- § These include multidimensional arrays:
  - § MATLAB is not restricted to 2 dimensions.
  - § They can be Numeric, Character, Cell, or Structure arrays.
- § Cell arrays have elements that contain any type of MATLAB data, including other cells.
  - § So cell arrays can be a mixture of different data types all in one package.
- § Structures are similar to the ones in the C language.
  - § They are constructs that contain named fields that correspond to any kind of data.

28

## BUILDING ARRAYS

If you are entering your arrays or matrices on the command line, then you enclose the data in square brackets.

```
>> x = [1 2 3 4 5];
```

(Note that we can separate by spaces or commas.)

§ Examples:

This vector will have the values 1, 2, ..., 10:

```
>> x = 1:10
```

```
x =
```

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

You can use increments other than one:

```
>> x = .1:.1:1
```

29

## BUILDING ARRAYS

§ You can decrement also

```
>> x = 10:-1:1
```

```
x =
```

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

§ `linspace(x1, x2)` generates a row vector of 100 linearly equally spaced points between `x1` and `x2`.

§ `linspace(x1, x2, N)` generates `N` points between `x1` and `x2`.

```
>> linspace(0,90,10)
```

30

## BUILDING ARRAYS

Note the use of the semi-colon to tell MATLAB that these are separate rows. Also note that either spaces or commas can separate elements in the arrays

```
>>X = [ 1 2 3 ; 4 5 6]
```

§ is a matrix where the first row is '1 2 3' and the second row is '4 5 6'.

```
>>X = [ 1:10 ; 11: 20]
```

§ is a matrix where the first row is 1 to 10 and the second row is 11 to 20.

You can also enter the above as

```
>>X = [ 1:10  
11:20]
```

You can combine array construction notation.

```
>>X = [ (1:10)' , (11: 20)']
```

§ is a matrix where the first column is 1 to 10 and the second column is 11 to 20.

```
>>X=[ A , B ]
```

§ is a matrix made up of the matrices A and B side by side.

31

## BUILDING ARRAYS

The semi-colon ";" concatenates the elements as rows.

The comma "," concatenates them as columns.

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

```
A =  
1 2 3
```

```
>> B=[4 5 6]
```

```
B =  
4 5 6
```

```
>> X=[A,B]
```

```
X =  
1 2 3 4 5 6
```

```
>> Y=[A;B]
```

```
Y =  
1 2 3  
4 5 6
```

```
>> Z=[[A;B],[B;A]]
```

```
Z =  
1 2 3 4 5 6  
4 5 6 1 2 3
```

32

## ARRAY ADDRESSING

Individual array elements are accessed using subscripts enclosed in parentheses.

§ Examples:

§ `X(1)` returns the first element of a row or column vector `X`.

§ `X(length(X))` returns the last element of a row or column vector `X`.

§ `X(end)` will also return the last element.

If you want to access several elements, then use the colon ":" notation.

§ Examples:

§ `x(1:5)` returns the first five elements of vector `x`

§ `x(3:-1:1)` returns the third, second and first elements of `x`

§ `x([ 2 4 6 ])` returns the second, fourth and sixth elements of `x`.

33

## ARRAY ADDRESSING

This is very similar to vector addressing, but not the use of the comma.

§ `X(1,4)` returns the element in the 1<sup>st</sup> row and the 4<sup>th</sup> column.

§

§ `X(1:5,3)` returns the values in the first 5 rows in the third column.

§ `X(:,4)` returns the fourth column, all rows

§ `X(4,:)` returns the fourth row, all columns

§ `X(temp,:)` returns the rows that are indexed in the variable `temp` and all columns

Some special arrays and matrices:

§ All 0's:

§ `zeros(n)` creates an `n x n` matrix of zeros

§ `zeros(1,n)` creates a row vector of `n` zeros

§ `zeros(m,n)` creates an `m x n` matrix of zeros

§ All 1's:

34

§ `ones(n)` creates an `n x n` matrix of ones

## ARRAY ADDRESSING

§ Identity

```
» eye(3) % a 3x3 identity matrix
```

```
ans =
```

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

§ Random:

```
» rand(2) % 2x2 uniform (0,1) matrix
```

```
ans =
```

```
0.9501 0.6068
0.2311 0.4860
```

```
» randn(3) % 3x3 standard normal (0,1) matrix
```

```
ans =
```

```
-0.4326 0.2877 1.1892
-1.6656 -1.1465 -0.0376
0.1253 1.1909 0.3273
```

35

## GENERATING MULTIDIMENSIONAL ARRAYS

You can think of the dimensions 3 or higher in an array as corresponding to pages.

Say we have  $A = [5\ 7; 0\ 1]$  for a 2-D array.

We can add a third dimension as:

```
» A=[5 7;0 1] % create the 2d array A
```

```
A =
```

```
5 7
0 1
```

```
» A(:,:,2)=[1 0;0 1] % add the 3rd dimension
```

```
» A
```

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

```
5 7
0 1
```

```
A(:,:,2) =
```

```
1 0
0 1
```

36

## OPERATING ON ARRAYS AND MATRICES

When two arrays or matrices have the same dimensions, addition and subtraction are applied element-by-element.

```
>> a = 1:5
>> b = 2:6
>> a+b
     3  5  7  9 11
```

§ Scalar-array mathematics (addition, subtraction, multiplication, division) work element-wise.

```
>> a*2
     2  4  6  8 10
```

§ If you want element-by-element array-array multiplication, powers, or division, then you have to use different notation:

- §  $x .* y$  means multiply element-by-element
- §  $x ./ y$  means divide element-by-element
- §  $x.^2$  means square each element of  $x$

§ reshape let's us change the shape of an array

```
>> a=1:6
a =
     1     2     3     4     5     6
>> reshape(a,2,3)
```

37

## OTHER ARRAY OPERATIONS

§ The *inner product* is calculated using:

```
>> a*b'
ans =
    70
```

§ The *outer product* is calculated using:

```
>> a'*b
ans =
     2     3     4     5     6
     4     6     8    10    12
     6     9    12    15    18
     8    12    16    20    24
    10    15    20    25    30
```

38

## ARRAY MANIPULATION

§ You can use the `:` notation to stretch out a matrix into one column:

```
>> c=[2 3;4 6]
```

```
c =
```

```
     2     3  
     4     6
```

```
>> c(:)
```

```
ans =
```

```
     2  
     4  
     3  
     6
```

39

## USING TEXT IN MATLAB

You can have strings in MATLAB by enclosing characters in single quotes.

```
>> t = 'This is a character string.'
```

§ The function `disp` can be used to display the variable without repeating the variable name.

```
>> disp(t)  
This is a character string.
```

§ The function `eval` evaluates a string as a MATLAB command.

```
>> eval('load data') % this loads the  
file                % data.mat
```

§ The function `feval('fun',x)` evaluates the function 'fun' at x. We will examine this more closely when we study M-files.

40

## USEFUL TEXT FUNCTIONS

It is useful to use a text string with the input function, which allows the user to input data:

```
> s='Enter the number of time steps ';  
> num_steps=input(s)  
  
Enter the number of time steps 5  
  
num_steps =  
  
5
```

Some useful conversion functions are:

- § int2str            Integer to string.
- § num2str           Number to string.
- § char                Numeric values into character string.
- § This function was previously setstr but has been renamed to char. Although setstr still works it may be removed in the future. Use char instead.
- § str2num            String to number

41

## MATRIX FUNCTIONS

MATLAB has many functions that will handle linear algebra problems. A complete table is given on pg. 235-238 of Hanselman.

- § chol                Cholesky factorization
- § eig                 Eigenvalues and eigenvectors
- § logm                Matrix logarithm
- § lu                  Factors from Gaussian elimination
- § norm                Matrix and vector norms
- § pinv                Pseudoinverse
- § rank                Rank of a matrix
- § sqrtm               Matrix square root
- § svd                 Singular value decomposition
- § trace                Sum of the diagonal elements

42

## SOLVING SETS OF EQUATIONS

MATLAB got its name from Matrix Laboratory. It was originally created to provide a convenient interface to numerical linear algebra subroutines.

A common problem in linear algebra is the solution of a set of equations: Consider 3 equations with 3 unknowns. We can consider the following,

$$\begin{aligned}x_1 + 2x_2 + 3x_3 &= 366 \\4x_1 + 5x_2 + 6x_3 &= 804 \\7x_1 + 8x_2 &= 351\end{aligned}$$

This is of the form  $Ax = b$

§ Here multiplication is defined as matrix multiplication.

§ In MATLAB we can solve this in either of two ways:

```
>> A=[1,2,3;4,5,6;7,8,0]
>> b=[366;804;351]
```

43

## SOLVING SETS OF EQUATIONS

Continuing our solution we could use the `inv` function.

```
>> x=inv(A)*b %take the inverse then multiply
x =
    25.0000
    22.0000
    99.0000
```

§ The function `inv()` gives the inverse of a square matrix.

§ A better way, from both an execution time and numerical accuracy standpoint, is to use the matrix division operator

§  $x = A \backslash b$

§ Also known as matrix left division, this produces the solution using Gaussian elimination, without forming the inverse.

```
>> x=A\b %the preferred method
using the
%left division of A
into b
```

44

## RELATIONAL OPERATORS

Relational operators can be used to compare two arrays (of the same size).

They can be used to compare a scalar with all elements of an array.

```
>> A = 1:9;

>> tf = A > 4           % elements greater than
4
0 0 0 0 1 1 1 1 1
```

§ Compare two vectors:

```
>> b=[5:-1:1, 6:9]
5 4 3 2 1 6 7 8 9
>> a=1:9
1 2 3 4 5 6 7 8 9
>> tf=a==b
0 0 1 0 0 1 1 1 1
```

45

## LOGICAL OPERATORS

You can use logical operators to combine or negate relational expressions.

These are similar to other programming languages.

<u>Logical Operator</u>	<u>Description</u>
&	And
	Or
~	Not

46

## RELATIONAL & LOGICAL FUNCTIONS

One function you might find useful is `find`.

- § This can be used to find elements of a matrix that are non-zero.
- § It returns a vector containing the indices that correspond to the non-zero elements.

One useful way to employ this function is to check which elements in a vector satisfy a certain criterion.

- § This function can also be used with matrices:

```
[i,j]=find(X) % finds nonzero entries on  
X
```

47

## FOR LOOPS

These allow several commands to be repeated a fixed number of times.

- § The general form of a 'for loop' is:

```
for x = array  
    commands  
end
```

```
x = zeros(1,10);           % set up storage  
    space first  
for n = 1 : 10  
x(n) = sqrt(n);  
end
```

- § NOTE: You could do the above without using a for loop.

```
» x=sqrt(1:10)
```

- § This will achieve the same result much faster.

48

## WHILE LOOPS

A while loop will execute an indefinite number of times.

- The general syntax is:

```
while expression
    commands
end
```

- The commands between the while and end are executed as long as 'expression' is True.
- Example:

```
>> EPS = 1;
>> while ( 1 + EPS ) > 1
    EPS = EPS/2;
end
```

- Usually a scalar entry is used in the expression, but you can use arrays also.
- In the case of arrays, all elements of the resulting array must be true for the commands to execute.

49

## IF-ELSE-END STRUCTURES

Sometimes commands must be executed based on a relational test.

You can use an if-else-end structure for these cases.

§ The syntax for the simplest form is:

```
if expression
    commands
end
```

50

## IF-ELSE-END STRUCTURES

### ALTERNATIVES

§ You can use the if-else-end structure if there are two alternatives.

§ The general syntax for this is:

```
if expression
  commands
else
  commands
end
```

51

## SWITCH STATEMENT

```
switch expression

case value1
  commands execute if expression is
  value1

case value2
  commands execute if expression is
  value2

.
.
.

otherwise
  commands

end
```

§ Expression must be either scalar or a character string.

§ If the one statement executes, the others do not.

§ If all comparisons are false, commands following otherwise are executed

52

## FPRINTF AND SPRINTF

- § fprintf is used to convert numerical results into ASCII format and append it to a file.
- § sprintf does the same thing except that it appends it to a string.

```
» s=sprintf('A rectangle of width %.5g and length %.5g has an area of %.5g\n',w,l,a)
```

```
s =
```

```
A rectangle of width 5 and length 2 has an area of 10
```

- § This form is more prone to errors

```
» s=['A rectangle of width ' num2str(w) ' and length ' num2str(l) ' has an area of ' num2str(a)]
```

```
s =
```

```
A rectangle of width 5 and length 2 has an area of 10
```

- § The table on page 137-138 of *Mastering MATLAB 6* lists conversion specifications for use with sprintf.

53

## SOME PROPERTIES OF SCRIPT FILES

- § **Once you type the name of your script file at the command line, MATLAB does the following:**

- § It searches the current variables and built-in MATLAB commands.
- § If your M-file name is not there, it searches the current directory.  
If the file is there, it opens it and runs the commands.
- § If it is not in the above places, MATLAB searches the path.

- § **Variables created by the commands in your M-file stay in the workspace.**

- § Commands in your M-file have access to all of the variables referred to in the M-file.
- § Usually, the commands in the M-file are not displayed as they are executed.
- § The command `echo on` displays the commands to the command window as they are read and evaluated.
- § The command `echo off` stops this and the command `echo` just toggles the state.

54

## FUNCTION M-FILES

- § There is a different type of M-file called an M-file Function.
- § You can speed up your code significantly by writing things as M-file functions rather than just script files.
- § Functions can be thought of as black boxes: all you see is what goes in and what comes out.
  - § Any commands that are evaluated or variables that are created are hidden from the user.
  - § These are very useful for evaluating mathematical sequences of commands that you might want to use many times.
  - § It is similar to the script file, in that it also is a text file with the .m extension, however intermediate variables within the function do not appear in or interact with the MATLAB workspace.
  - § You create this file in the same manner, but with slightly different syntax.
  - § Pages 176-178 of the text lists the rules and criteria governing Function M-Files.

55

## FUNCTIONS

- § A Function M-file is different from a script file; the key differences are:
  - § Functions communicates with the workspace through the variables passed into it and that are produced from it.
  - § Any intermediate variables created by the function are hidden and do not interact with the workspace.
  - § There is a specific syntax that tells MATLAB that an M-file is a function and not a script file.
  - § The first line of an M-file function must be either:

```
function arg_out = function_name(arg_in)
or
function [arg1,arg2] = func_name(in1,in2)
```

- § It is a good idea to put several comment lines in the beginning of your function file.
  - § These will be returned by the `help` command.
  - § The first comment line, also called the H1 line, is searched by the `lookfor` command.

56

## FUNCTIONS

### § Rules about function M-Files:

- § You should always name the function and its file name the same.
- § The first time MATLAB executes a Function M-file, it opens the text file and compiles the commands.
- § The function is now represented in memory, which speeds execution time.
  
- § Other functions that are called by a function M-File are also compiled.
  
- § In contrast, script M-files are interpreted and are not compiled even if they are called by another function.

Here is a simple function that returns a square of the value passed.

```
function x = squareit(y)
%SQAREIT returns the square of an array or
  scalar.
%This is to show a very simple example of a
  function.

x=y^2;
```

```
>> squareit(5)
ans =
25
```

57

## MATLAB DEBUGGING

- § When developing MATLAB M-Files eventually errors will occur. MATLAB will tell you when you have errors. There are two types of errors, *syntax* and *run-time*.
  
- § *Syntax* errors can be generated when an expression or function is compiled into memory.
  - § These can be things like
    - § misspelled variables and function names, misplaced quotes or parenthesis, etc..
  
- § MATLAB flags these errors immediately and provides feedback describing the error and where it occurred.
  - § Syntax errors are usually easy to identify.
  
- § *Run-time* errors are generated when an operation leads to unnatural results.
  - § These can be caused by operations the result in things like
    - § Empty arrays
    - § NaNs
  
- § MATLAB flags these errors then returns control to the

58

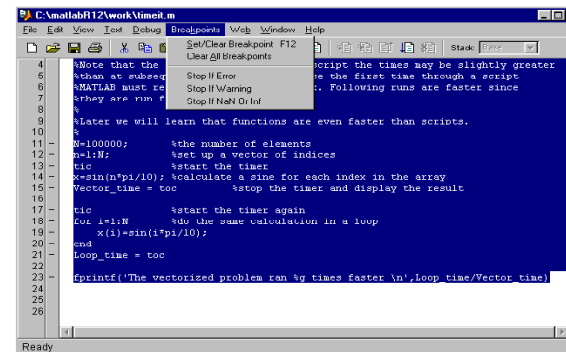
## DEBUGGING BY HAND

- § For simple problems *manual* debugging techniques can be quite useful.
  
- § Remove semicolons from selected lines within your function so that intermediate results are dumped to the screen.
  
- § Add statements that display variables of interest within the function.
  
- § Place the keyboard command at places in the function where you want to examine a variable.
  - § Remember to use `return` to exit the keyboard state at the `K>>` prompt.
  
- § Change the function M-file into a script M-file by placing a `%` before the function definition statement at the beginning of the M-file.
  - § This will let you examine the workspace when the termination occurs.

59

## THE GRAPHICAL DEBUGGER

- § MATLAB on the PC features an integrated M-file editor / debugger.
  
- § It can be launched right from your M-file editing session by typing `edit` at the command prompt.
  
- § Or launch it by choosing `File` ▶ `New` or `Open`.



```
4 %Note that the
5 %loop is subseq
6 %MATLAB must re
7 %they are run f
8 %
9 %Later we will learn that functions are even faster than scripts.
10 %
11 N=100000; %the number of elements
12 n=1:N; %set up a vector of indices
13 tic %start the timer
14 x=sin(n*pi/10); %calculate a sine for each index in the array
15 Vector_time = toc %stop the timer and display the result
16 %
17 tic %start the timer again
18 for i=1:N %do the same calculation in a loop
19     X(i)=sin(i*pi/10);
20 end
21 Loop_time = toc
22 %
23 fprintf('The vectorized problem ran %g times faster \n', Loop_time/Vector_time)
24
25
26
```

60

## THE GRAPHICAL DEBUGGER

- § Buttons exist on the toolbar to do rudimentary procedures such as single step, continue, and quit debugging.
- § You can set and clear breakpoints in an easy manner.
- § You can view the value of a variable or expression by highlighting it in the editor then using Text ▶ Evaluate Selection

61

## LOW LEVEL FILE I/O

- § **MATLAB is not a world unto itself**
  - § Many times we wish to work with data from various sources
  - § Or we want to output data in a specific format
  - § MATLAB's low level I/O functions let you read data collected in other formats and write data for other applications.

**Here is a summary of the MATLAB low level file I/O functions. These functions can be found in the directory `iofun` in the MATLAB toolbox.**

- § File opening and closing
  - § `fopen` opens files
  - § `fclose` closes files

- § Binary I/O
  - § `fread` reads binary data from a file
  - § `fwrite` writes binary data to a file

Formatted I/O

<code>fscanf</code>	reads formatted data from a file
<code>fprintf</code>	writes formatted data to a file
<code>fgetl</code>	reads a line from a file but discards the
	newline character
<code>fgets</code>	reads a line from a file, keeping the newline

62

## LOW LEVEL FILE I/O

Low level I/O functions continued.

### § String Conversion

§ `sprintf` writes formatted data to a string  
§ `scanf` reads a string under format control

### § File Positioning

§ `ferror` returns the file I/O error status  
§ `feof` tests for the end of file  
§ `fseek` sets the file position pointer  
§ `ftell` gets the file position pointer  
§ `frewind` resets the file position pointer to the beginning

### § Temporary Files

§ `tempdir` gets the temporary directory's name  
§ `tempname` creates a temporary file name

**Most of MATLAB's file I/O functions are based on the I/O functions of the ANSI Standard C Library.**

**If you know C, you are probably familiar with these routines.**

63

**Be sure to check the MATLAB function - not everything is exactly the**

## Low Level File I/O - Opening and Closing Files

§ **Before reading or writing an ASCII or binary file, you must open it with `fopen`.**

```
§ fid = fopen('filename','permission')
```

'permission' can be;

§ 'r' reading only  
§ 'w' writing only (creates if necessary)  
§ 'a' appending only (creates if necessary)  
§ 'r+' read and write (do not create)  
§ 'w+' truncate or create for read and write  
§ 'a+' read and append (create if necessary)

§ **After opening a file you must close it when you are finished with it. Do this with `fclose(fid)`.**

§ `fclose` returns 0 if the file was successfully closed, or -1 if it was not.

§ If 'filename' does not exist when you use `fopen`, it will be created (where indicated). You must include the desired extension in the name; none is assumed.

```
» fopen('iotest','w') %will create if not there
ans =
     3
» fclose(3) %fid is 3
ans =
     0
```

64

## LOW LEVEL FILE I/O - BINARY DATA

You can read / write binary files using `fread` and `fwrite`.

```
[A,count] = fread(fid,size,'precision')
```

§ You can control the number of values read by `fread` with `size`.

`A = fread ( fid, 100)` will read the first 100 data values of the file specified by `fid` into a column vector `A`.

Replacing 100 with `[10 10]` reads the same first 100 values, but stores them in `A` as a 10 by 10 array.

```
count = fwrite(fid,A,'precision')
```

65

## LOW LEVEL FILE I/O - BINARY DATA

'precision' is a string representing the numeric precision of the values read, controlling the number of bits read for each value. Any of the following strings, either the MATLAB versions, or their C or Fortran equivalents, may be used. By default, numeric values are returned in double precision arrays.

<u>MATLAB</u>	<u>C or Fortran</u>	<u>Description</u>
'uchar'	'unsigned char'	unsigned character, 8 bits.
'schar'	'signed char'	signed character, 8 bits.
'int8'	'integer*1'	integer, 8 bits.
'int16'	'integer*2'	integer, 16 bits.
'int32'	'integer*4'	integer, 32 bits.
'int64'	'integer*8'	integer, 64 bits.
'uint8'	'integer*1'	unsigned integer, 8 bits.
'uint16'	'integer*2'	unsigned integer, 16 bits.
'uint32'	'integer*4'	unsigned integer, 32 bits.
'uint64'	'integer*8'	unsigned integer, 64 bits.
'single'	'real*4'	floating point, 32 bits.
'float32'	'real*4'	floating point, 32 bits.
'double'	'real*8'	floating point, 64 bits.
'float64'	'real*8'	floating point, 64 bits.

66

## LOW LEVEL FILE I/O FORMATTED DATA

Reading and Writing from/to formatted ASCII text files.

```
[A,count] = fscanf(fid,'format',size)
```

§ reads data from the file specified by file identifier `fid`, converts it according to the specified `'format'` string, and returns it in matrix `A`.

§ `count` is an optional output argument that returns the number of elements successfully read.

§ `size` is optional; it puts a limit on the number of elements that can be read from the file; if not specified, the entire file is considered; if specified, valid entries are:

`N` read at most `N` elements into a column vector.

`inf` read at most to the end of the file.

`[M,N]` read at most `M * N` elements filling at least an `MxN` matrix, in column order. `N` can be `inf`, but not `M`.

§ If the matrix `A` results from using character conversions only and `size` is not of the form `[M,N]` then a row vector is returned.

§ `fscanf` differs from its *C* language namesake in one major respect, it is *vectorized* in order to return a matrix argument. The format string is recycled <sup>67</sup>

## LOW LEVEL FILE I/O - FORMATTED DATA

Reading and Writing from/to formatted ASCII text files.

`count = fprintf(fid,format,A,...)` formats the data in the real part of matrix `A` under control of the specified format string, and writes it to the file associated with file identifier `fid`. `fprintf` returns a count of the number of bytes written.

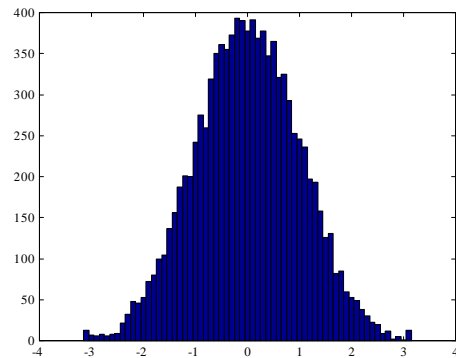
```
» x = 0:.1:1;  
» y = [x; exp(x)];  
» fid = fopen('a:/exp.txt','w');  
» fprintf(fid,'%6.2f %12.8f\n',y);  
» fclose(fid)
```

§ data is written into the matrix in column order

68

## STATISTICAL ANALYSIS

```
» x = -3.1:0.1:3.1;  
» y = randn(10000,1);  
» hist(y,x)
```



69

## Getting Graphics Hardcopy

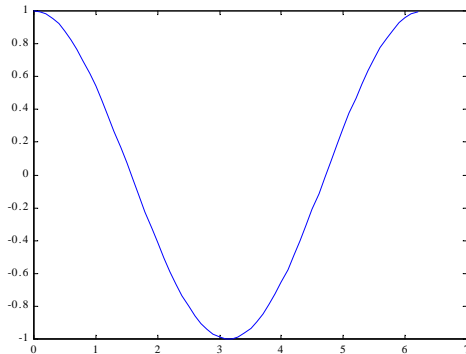
- § You can use the normal Windows 'Print' command from the 'File' menu.
- § Use the menu from the Figure window.
- § You can Copy/Paste into your document.
  - § NOTE: in previous versions of MATLAB the default background color is black. Be sure to invert the background or else you will have a black figure.
- § You can also use the 'print' command at the command line.
  - § Arguments to the print function call different devices.

70

## PLOT FUNCTION

- § The most common command for plotting data in 2-D is the `plot` function. This function plots sets of data (vectors) using appropriate axes and connects the points with straight lines.

```
» x=0:.1:2*pi; % create horiz vector
» y=cos(x);    % find cos of each one
» plot(x,y)    % plot
```



.1

## PLOT FUNCTION

- § `plot` opens a Figure window, scales the axes to fit the data and plots the points.
- § It adds a scale and tic marks to both axes.
  - § If a Figure window already exists, then it clears the current window and draws a new plot.

You can plot several lines on the same plot by putting a series of them as arguments to the `plot` function.

```
plot(x1,y1,x2,y2)
```

- § If one of the arguments is a matrix and the other a vector, then it plots each column of the matrix versus the vector.

If you provide just one argument, then the following can happen:

- § If it is complex, then it plots the real part versus the imaginary.
- § If it is real-valued, then it plots the vector (or matrix) versus the index of its values.

72

## LINESTYLES, SYMBOLS AND COLORS

§ The default linestyle is a solid line...MATLAB allows you to choose from several.

- solid line
- : dotted line
- . dash-dot line
- dashed line

§ If you plot several lines on one plot, then MATLAB starts with blue and cycles through the colors green, red, cyan, magenta, yellow, black, and white.

§ You can use a symbol for each point and they will not be connected by lines.

§ The symbols that are available are:

- . point
- o circle
- x x-mark
- + plus
- \* star
- s square
- d diamond
- v down triangle
- ^ up triangle
- < left triangle
- > right triangle
- p pentagram
- h hexagram

73

## LINESTYLES, SYMBOLS AND COLORS

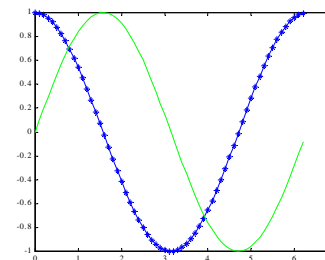
§ You can select the color of a line or symbol.

- b blue
- g green
- r red
- c cyan
- m magenta
- y yellow
- k black
- w white

§ You can combine both lines and symbols.

§ For example, plot the sine and cosine on the same plot, with the cosine plotted as a line and with symbols.

```
>> y2=sin(x);  
>> plot(x,y,x,y,'b*',x,y2,'g-')
```



74

## GRIDS AND LABELS

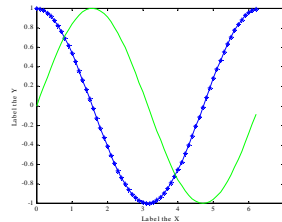
- § The command `grid` adds grid lines to the plot at the tic marks.
- § Repeated use of the command toggles the grid lines on and off.

- § You can easily add a title to your plot using:

```
title('My Plot Title')
```

- § You can add labels to the horizontal and vertical axes by using:

```
xlabel('Label the X')  
ylabel('Label the Y')
```



75

## GRIDS AND LABELS

- § Text can be added to any location on the plot with the `text` command:

```
text(x,y,'label')
```

- § where the `x` and `y` represent the coordinates of the center left edge of the string in units corresponding to the axes.
- § You can use the function `gtext('label')` to place the label with the mouse.
- § The `gtext` function activates the current Figure window, gives you a cross-hair that follows the mouse and waits for the mouse click.
- § The text is placed in the lower left corner of the first character at that location.

76

## ZOOM

`zoom on` turns zooming on.

- § Click left mouse button in the Figure window to expand by a factor of two.
- § Click right mouse button to zoom out by a factor of two.
- § Click and drag rectangle to zoom into a particular area.
- § `zoom(n)` zooms by a factor of `n`.
- § `zoom out` returns the plot to its initial state.
- § `zoom off` turns off zoom mode.
- § `zoom` toggles the zoom state.
- § For zoom to be used the legend must be turned off.
- § Since both `zoom` and `legend` respond to mouse clicks they can interfere with each other.

77

## AXES

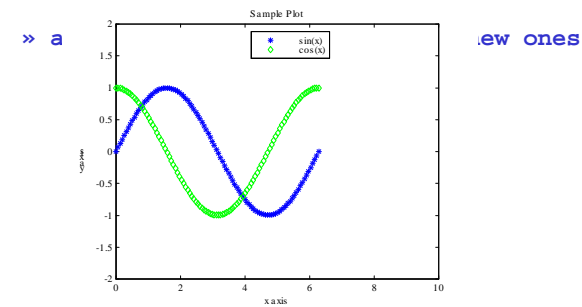
You can use the command `axis` to change the axes of your plot.

- § The argument to this is a four element vector containing the following information:

```
[xmin xmax ymin ymax]
```

- § You can use the function without any arguments to get the current axes values.

```
>> axis % get current axes values  
ans =  
0 7 -1 1
```



78

## MULTIPLE PLOTS

§ You have seen that you can plot multiple data against a single axis using the `plot` command.

```
§  
plot(x,sinx,'b*',x,cosx,'gd')
```

§ You can also add new plots to an existing plot by using the `hold` command.

§ `hold on` tells MATLAB not to remove the existing axes when new plot functions are issued.

§ If the new data do not fit within the current axes limits, the axes are rescaled.

§ `hold off` releases the current figure window for new plots.

§ `hold` with no arguments toggles the setting.

§ The color cycle starts anew with each call to `plot`. You might want to specify plot colors when using `hold` so that lines aren't plotted in the same color.

```
§ ishold returns 1 if hold is on.  
» hold  
Current plot held  
» ishold
```

```
ans =
```

```
1
```

79

## SUBPLOTS

Sometimes you might want to plot more than one data set on multiple axes, rather than several plots on one axis. You can do this with the `subplot(m,n,p)` command.

§ This gives a matrix of  $m \times n$  plots in a single Figure window.

§ The  $p$  stands for the  $p$ -th area to be active.

§ The subplots are numbered left to right from the top row to the bottom.

§ Be careful about putting too many plots in one Figure window.

§ The active subplot is the one responsive to the previous commands to a Figure (e.g., `axis`, `xlabel`, `ylabel`, `title`)

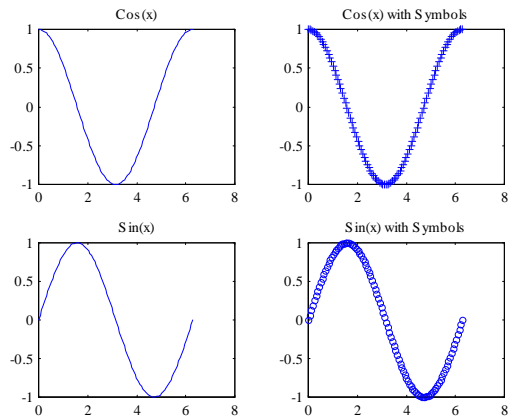
§ When you want to go back to one axis in a Figure window, you must use:

```
subplot(1,1,1)
```

80

## SUBPLOTS

```
» subplot(2,2,1),plot(x,cosx)
» title('Cos(x)')
» subplot(2,2,2),plot(x,cosx,'+')
» title('Cos(x) with Symbols')
» subplot(2,2,3),plot(x,sinx)
» title('Sin(x)')
» subplot(2,2,4),plot(x,sinx,'o')
» title('Sin(x) with Symbols')
```



81

## Multiple figure windows

- § You can create multiple figure windows and plot different data sets in different ways in each one.
- § Select **New Figure** from the **File** menu or,
- § Use `figure(n)` in the command window

82

## RETRIEVING DATA FROM PLOTS

- § The function `ginput` allows you to select points from a plot based on the position of a mouse click.
  - § The returned data are not necessarily points from the data set used to create the plot, but rather the explicit x and y coordinate values where the mouse was clicked.
  - § If points are selected outside the plot axes limits, the points returned are extrapolated values.
  - § Returned data are with respect to the current or active subplot.
  - § `[x,y]=ginput(n)` will retrieve n points. Not specifying n will allow retrieves until the Return key is pressed.
  - § Before using `ginput`, `zoom` and `legend` should be turned off, since all respond to mouse clicks and can interfere with each other.

```
> [u,v]=ginput(3)
```

```
u =
```

```
2.5091
```

```
2.7273
```

```
2.8000
```

83

## OTHER 2-D PLOTS

MATLAB provides a host of specialized 2-D plots.

<code>polar</code>	plot of polar coordinates as a function of angle and radius
<code>Bar</code>	bar graph
<code>stairs</code>	stairstep graph...no spacing or lines
<code>stem</code>	stem plot
<code>Errorbar</code>	graph with errorbars
<code>feather</code>	displays angle and mag as arrows
<code>compass</code>	same as above, except it emanates from origin.
<code>bar3</code>	vertical 3-D bar chart
<code>bar3h</code>	horizontal 3-D bar chart
<code>barh</code>	horizontal bar chart
<code>pie</code>	pie chart
<code>pie3</code>	3-D pie chart
<code>rose</code>	draws polar histogram

84

## OTHER PLOTTING COMMANDS

`plotmatrix(x,y)` scatter plots columns of  $x$  against the columns of  $y$ .

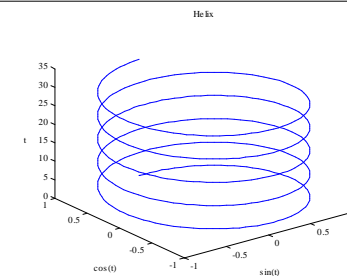
§ If  $X$  is  $P$ -by- $M$  and  $Y$  is  $P$ -by- $N$ , `PLOTMATRIX` will produce an  $N$ -by- $M$  matrix of axes.

85

## 3-D PLOTTING

The `plot3` function is similar to the 2-D plot function, except we are now in three dimensions. The syntax is similar to `plot` except that you need to provide 3 data sets or vectors.

```
» t=0:pi/50:10*pi;  
» plot3(sin(t),cos(t),t)  
» xlabel('sin(t)'),ylabel('cos(t)'),zlabel('t')  
» title('Helix')
```



86

## 3-D PLOTTING

- § As you just saw, there is a `zlabel` function that you can use with 3-D plots.
- § You can use the `hold` command or several arguments to the `plot3` function just like in the 2-D case.

You have a certain viewpoint with 3-D graphics, specified by azimuth and elevation.

- § The azimuth is the angle with respect to the  $x=0$  plane.
- § The elevation is the angle with respect to the  $z=0$  plane.

- § You can change this view with the command:

```
view([az,el])
```

- § See the page 398 - 400, of your text for a complete description of the view command.

You can interactively rotate the view of a 3-D plot with the command `rotate3d`.

- § Try the following:

```
> [x,y,z]=peaks;  
> mesh(x,y,z);  
> rotate3d
```

87

## 3-D PLOTTING

- § Sometimes we need to view a scalar function of two variables:

$$z=f(x,y)$$

- § A plot of this is a surface in 3 dimensions.
- § To plot this in MATLAB, the values of  $z$  are stored in a matrix.
- § One way to get these values is to first create a matrix of  $x$  and  $y$  values:

```
[X,Y]=meshgrid(-3:3,1:5)
```

- § `meshgrid` will duplicate  $x$  for each of the rows in  $y$  and will duplicate  $y$  as a column for each of the columns in  $x$ .
- § This allows all the values of  $z$  to be computed in a single statement.

- § If  $f(x,y) = (x + y)^2$  then,

```
Z = (X + Y).^2 % element by element
```

88

## 3-D PLOTTING

§ You can plot a mesh surface defined by the z-coordinates of points above a rectangular grid in the x-y plane.

§ A mesh is formed by joining adjacent points with straight lines.

```
» [X,Y,Z]=peaks(30);  
» mesh(X,Y,Z)
```

§ `peaks` is a function of two variables, obtained by translating and scaling Gaussian distributions.

