

# Introduction to MATLAB

3/17/2009

1/134

## Contact Information

- Course website:
  - <http://www.ing.unife.it/simani/lessons23.html>
  - Information also will be contained on website

3/17/2009

2/134

## Course Structure

- Overview of MATLAB
  - History of MATLAB
  - Overview of MATLAB environment
  - Discussion of MATLAB tools
- Basic MATLAB
  - Simple MATLAB functionality
    - Syntax, Commands
  - Exercises involving basic MATLAB functionality and its *Toolboxes*

3/17/2009

3/134

## Course Structure

- Advanced MATLAB Functionality
  - Beyond MATLAB as a calculator
  - The MATLAB programming language
  - Project showcasing MATLABs advanced functionality
    - Other *Toolboxes*
    - Dynamic System Simulations
    - Digital Control Design

3/17/2009

4/134

## Coursework

- Collection of exercises:
  - Will occur during the laboratory sessions
  - Will involve MATLABs basic functionality
  - Will exploit its Toolboxes for Control System Design
- **Final Examination:**
  - Single practical project @ PCs;
  - Will cover **Digital Control System** theory.

3/17/2009

5/134

## MATLAB Overview

- What is MATLAB?
- History of MATLAB
  - Who developed MATLAB
  - Why MATLAB was developed
  - Who currently maintains MATLAB
- Strengths of MATLAB
- Weaknesses of MATLAB

3/17/2009

6/134

## What is MATLAB?

- MATLAB
  - **MATrix LABoratory**
  - Interactive & programming language
    - Will be covered during week 2
  - Control System Design & Programming tool
    - Will be covered during week 3

3/17/2009

7/134

## What is MATLAB con't: 2

- Considering MATLAB at home
  - Standard edition
    - Available for roughly 2 thousand dollars
  - Student edition
    - Available for roughly 1 hundred dollars.
    - Some limitations, such as the allowable size of a matrix

3/17/2009

8/134

## Strengths of MATLAB

- MATLAB is relatively easy to learn
- MATLAB code is optimized to be relatively quick when performing matrix operations
- MATLAB may behave like a calculator or as a programming language
- MATLAB is interpreted, errors are easier to fix
- Although primarily procedural, MATLAB does have some object-oriented elements

3/17/2009

9/134

## Weaknesses of MATLAB

- MATLAB is NOT a general purpose programming language
- MATLAB is an interpreted language (making it for the most part slower than a compiled language such as C++)
- MATLAB is designed for scientific computation and is not suitable for some things (such as parsing text)

3/17/2009

10/134

# Overview

- Review of main topics
- Review of the MATLAB environment
- Declaring and manipulating variables
- Useful functions

3/17/2009

11/134

# MATLAB GUI

- Launch Pad / Toolbox
- Workspace
- Current Directory
- Command History
- Command Window

3/17/2009

12/134

# Launch Pad / Toolbox

- Brief details
- Launch Pad allows you to start help/demos
- *Toolbox* is for use with specialized packages (e.g., Signal Processing)

3/17/2009

13/134

# Using MATLAB

- This is the window that appears when you start MATLAB

This is the menu bar

This window shows the current directory or the workspace

This is the command window - we use this as our calculator

This window shows the command history

3/17/2009

34

## Workspace

- Allows access to data
- Area of memory managed through the Command Window
- Shows Name, Size (in elements), Number of Bytes and Type of Variable

3/17/2009

15/134

## Current Directory

- MATLAB, like Windows or UNIX, has a current directory
- MATLAB functions can be called from any directory
- Your programs (to be discussed later) are only available if the current directory is the one that they exist in

3/17/2009

16/134



## MATLAB as a Calculator

- You can enter expressions at the command line and evaluate them right away.

previous command	>> 3 + 5*8
	ans =
	43
next command	>>

The '>>' symbols indicate where commands are typed.

## Mathematical Operators

Operator	MATLAB	Algebra
+	+	$5 + 4 = 9$
-	-	$5 - 4 = 1$
×	*	$5 * 4 = 20$
÷	/	$5 / 4 = 1.25$
$a^b$	$a^b$	$5^4 = 625$

## Command History

- Allows access to the commands used during this session, and possibly previous sessions
- Clicking and dragging to the Command window allows you to re-execute previous commands

3/17/2009

19/134

## Command Window

- Probably the most important part of the GUI
- Allows you to input the commands that will create variables, modify variables and even (later) execute scripts and functions you program yourself.

3/17/2009

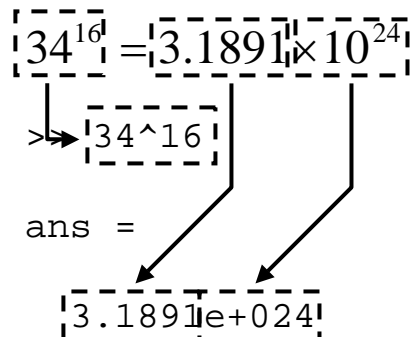
20/134

## Mathematical Operators

<pre>&gt;&gt; 5+4 ans =      9 &gt;&gt; 5-4 ans =      1 &gt;&gt; 5*4 ans =     20</pre>	<pre>&gt;&gt; 5/4 ans =     1.2500 &gt;&gt; 5^4 ans =     625 &gt;&gt; 34^16 ans =     3.1891e+024</pre>
--	--

## Number Representation

- MATLAB uses scientific notation for very large numbers and very small numbers.
- MATLAB has a special way of doing this:



## “BEDMAS”

B = Brackets  
E = Exponentials  
D = Division  
M = Multiplication  
A = Addition  
S = Subtraction

```
>> 3*4 + 2
```

```
ans =
```

```
14
```

```
>> 3*(4+2)
```

```
ans =
```

```
18
```

Be careful using brackets – check that opening and closing brackets are matched up correctly.

3/17/2009

23/134

## Simple Commands

- who
- whos
- save
- clear
- load

3/17/2009

24/134

## who

- **who** lists the variables currently in the workspace.
- As we learn more about the data structures available in MATLAB, we will see more uses of “**who**”

3/17/2009

25/134

## whos

- **whos** is similar to **who**, but also gives size and storage information
- **s = whos(...)** returns a structure with these fields: name, variable name, size, variable size, bytes, number of bytes allocated for the array, class, class of variable, and assigns it to the variable **s**. (We will discuss structures more).

3/17/2009

26/134

## Save

- **save** – saves workspace variables on disk
- **save filename** stores all workspace variables in the current directory in filename.mat
- **save filename var1 var2 ...** saves only the specified workspace variables in **filename.mat**. Use the \* wildcard to save only those variables that match the specified pattern.

3/17/2009

27/134

## clear

- **clear** removes items from workspace, freeing up system memory
- Examples of syntax:
  - **clear**
  - **clear name**
  - **clear name1 name2 name3 ...**

3/17/2009

28/134

## `clc`

- Not quite clear
- `clc` clears only the command window, and has no effect on variables in the workspace.

3/17/2009

29/134

## `load`

- `load` - loads workspace variables from disk
- Examples of Syntax:
  - `load`
  - `load filename`
  - `load filename X Y Z`

3/17/2009

30/134

## Declaring a Variable in MATLAB

- Not necessary to specify a type. (Such as integer or float)
- Several kinds of variables:
  - Vector
  - Matrix
  - Structure
  - Cell array

3/17/2009

31/134

## Declaring a variable, con't: 2

- For an integer or floating point number:  
simply set a variable name equal to some character
- Ex. `>> A = 5;`
- Or `>> A = 5`
  - Have you seen any difference?

3/17/2009

32/134



## Side Note 1

- The presence or lack of a **semi-colon** (*i.e.* ;) after a MATLAB command does not generate an error of any kind
- The presence of a **semi-colon** (;) tells MATLAB to suppress the screen output of the command

3/17/2009

33/134

## Side Note 1, con't: 2

- The lack of a **semi-colon** will make MATLAB output the result of the command you entered
- One of these options is not necessarily better than the other

3/17/2009

34/134

## Declaring a Variable, con't: 3

- You may now use the simple integer or float that you used like a normal number (though internally it is treated like a 1 by 1 matrix)
- Possible operations:
  - +, -, /, \*
  - Many functions (round(), ceil(), floor())

3/17/2009

35/134

## Declaring a variable, con't: 4

- You may also make a vector rather simply
- The syntax is to set a variable name equal to some numbers, which are surrounded by brackets and separated by either spaces or commas
- Ex. `>>A = [1 2 3 4 5];`
- Or `>>A = [1,2,3,4,5];`
  - Any difference?

3/17/2009

36/134

## Declaring a variable, con't: 5

- You may also declare a variable in a general fashion much more quickly
- Ex. `>> A = 1:1:10`
- The first 1 would indicate the number to begin counting at
- The second 1 would be the increase each time
- And the count would end at 10

3/17/2009

37/134

## Declaring a variable, con't: 6

- Matrices are the primary variable type for MATLAB
- Matrices are declared similar to the declaration of a vector
- Begin with a variable name, and set it equal to a set of numbers, surrounded by brackets. Each number should be separated by a comma or semi-colon

3/17/2009

38/134

## Declaring a variable, con't: 7

- The semi-colons in a matrix declaration indicate where the row would end
- Ex. `A = [ 1,2;3,4 ]` would create a matrix that looks like

```
[ 1 2
  3 4 ]
```

## Declaring a variable, con't: 7

- Matrices may be used as normal variables now. Multiplying is already defined for matrices, and additional code does not need to be written.

## Declaring a variable, con't: 8

- The final type of variable we will discuss today will be a “struct”.
- The command `struct` is used to create a structure
- Syntax:
  - `s = struct('field1',{},'field2',{},...)`
  - `s = struct('field1',values1,'field2',values2,...)`

3/17/2009

41/134

## Declaring a variable, con't 9

- A simple declaration of a structure is as follows:

```
Student.name = 'Joe';  
Student.age = 23;  
Student.major = 'Computer Science';
```

3/17/2009

42/134

## Declaring a variable, con't: 10

- Arrays of structures are possible.
- Taking the previous example, if one were to write:

```
Student(2).name = 'Bill'  
...etc
```

Then the **array** would be created for you.

3/17/2009

43/134

## Declaring a variable, con't: 11

- Structures can group information, but methods are not written for them.

3/17/2009

44/134

## Built-In Functions

- Like a calculator, MATLAB has many built-in mathematical functions.

```
>> sqrt(4)
```

```
ans =
```

```
2
```

```
>> abs(-3)
```

```
ans =
```

```
3
```

- To find out more about MATLAB's functions use MATLAB's help (from command window).

## Variables

- We use variables so calculations are easily represented.

$$C = (F - 32) \times \frac{5}{9}$$

$$F = 100 \Rightarrow C = 37.8$$

$$F = 32 \Rightarrow C = 0$$

- You can think of variables as *named locations in the computer memory in which a number can be stored.*

# MATLAB Variables

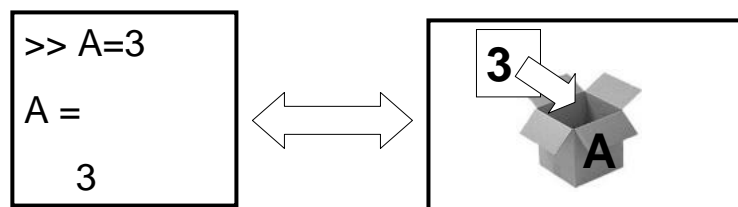
```
>> F=100
F =
    100
>> C=(F-32)*5/9
C =
    37.7778
>> F = 32
F = 32
>> C=(F-32)*5/9
C =
    0
```

3/17/2009

47/134

## Memory as a Filing System

- You can think of computer memory as a large set of “boxes” in which numbers can be stored. The values can be inspected and changed.
- Boxes can be labelled with a variable name.



3/17/2009

48/134



## Special Variables

- MATLAB has some special variables:
  - **ans** is the result of the last calculation.
  - **pi** represents  $\pi$ .
  - **Inf** represents infinity.
  - **NaN** stands for not-a-number and occurs when an expression is undefined e.g. division by zero.
  - **i**, **j** represent the square root of  $-1$  (necessary for complex numbers)

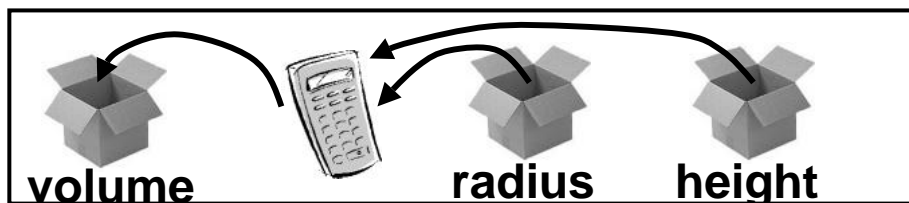
3/17/2009

49/134

## Calculations with Variables

- Suppose we want to calculate the volume of a cylinder.
- It's **radius** and **height** are stored as variables in memory.

```
>> volume = pi*radius^2*height
```



3/17/2009

50/134

## Simple Commands, con't: 2

- **who** and **whos** are similar, they allow you to see the variables in your workspace
- **save** saves the variables in your workspace to a binary file readable by MATLAB
- **clear** removes the variables from your workspace
- **load** loads the binary file created by the save command and restores the variables to your workspace

3/17/2009

51/134

## Simple Commands, con't: 3

- For any of these commands (and many others) you can get a more in depth explanation by typing `help` followed by the name of the command
- Ex. `>>help clear`
- Online documentation for all of these commands is also available on the Mathworks website

3/17/2009

52/134

## Declaring variables in MATLAB

- Learned how to declare several types of variables:
  - Normal floats and int(eger)s
  - Vectors
  - Matrices
  - Structures

3/17/2009

53/134

## Declaring variables in MATLAB, con't: 2

- Regular int/floats
- Variable name followed by an equals sign and the value you wish to assign
- Ex. **A = 5;**

3/17/2009

54/134

## Declaring variables in MATLAB, con't: 3

- Vectors
- Variable name followed by an equals sign and one or more numbers separated by either spaces or commas and surrounded by brackets
- Ex. `A = [ 1 2 3 4 5 ];`

3/17/2009

55/134

## Declaring variables in MATLAB, con't: 4

- Matrices
- Like vector – variable name followed by an equals sign and one or more numbers separated by either spaces or commas and surrounded by brackets. Use semi-colons to indicate a change in row.
- Ex. `A = [ 1 2; 3 4 ];`

3/17/2009

56/134

## Declaring variables in MATLAB, con't: 5

- Structures
- Like a struct in C or C++, similar to a class in C++ or Java, but lacking class specific functions or methods
- Declared using a point operator

3/17/2009

57/134

## Declaring variables in MATLAB, con't: 6

- Structures, con't
- Ex. 

```
A.name = 'Joe';  
A.age = 23;  
A.occupation = 'student';
```

3/17/2009

58/134

## Declaring variables in MATLAB, con't: 7

- Structures, con't: 2
- Can have an array of structures
- Ex. `A(2).name = 'Bob';`  
...

3/17/2009

59/134

## Sample MATLAB functions

- `Min`
- `Max`
- `Median`
- `Mean`
- `Sum`
- `Diff`

3/17/2009

60/134

## MATLAB Functions: `min`

- `min`
- Will find the minimum element of the array
- Works slightly different on vectors and matrices

3/17/2009

61/134

## MATLAB Functions: `max`

- `max`
- Will find the maximum element of the array
- Also works slightly different on vectors and matrices

3/17/2009

62/134

## MATLAB Functions: `median`

- `median`
- Will find the median value of the array
- Also works slightly different on vectors and matrices

3/17/2009

63/134

## MATLAB Functions: `mean`

- `mean`
- Returns the average value of the array
- Works slightly different on vectors and matrices

3/17/2009

64/134



## MATLAB Functions: `sum`

- `sum`
- Will return a sum of the array elements
- Also works slightly different on vectors and matrices

3/17/2009

65/134

## `diff`

- `diff`
- Will return the difference between adjacent elements in an array
- This is an approximate derivative

3/17/2009

66/134

## Overview

- Matlab further functions
- Plotting – in depth
- File I/O – few details

3/17/2009

67/134

## New MATLAB Function

- **rand( )** - Uniformly distributed random numbers and arrays
- Example of syntax:
  - **A = rand(n)**
  - **A = rand(m,n)**
- Where **m** and **n** are dimensions of the matrix

3/17/2009

68/134

## rand( ) con't: 2

- Scalars may be generated
  - Ex. `A = rand(1,1);`
- Vectors may be generated
  - Ex. `A = rand(10,1);`

3/17/2009

69/134

## rand( ) con't: 3

- Generated random numbers will be between 0 and 1.
- Scaling can be done by multiplying the resulting matrix or vector by the number you wish to scale with

3/17/2009

70/134

# Plotting

- Several types of plots available
  
- `plot`
- `polar`
- `bar`
- `hist`

3/17/2009

71/134

## `plot()` (from *MATLAB help*)

- Linear 2-D plot
  
- Syntax:
  - `plot(Y)`
  - `plot(X1,Y1,...)`
  - `plot(X1,Y1,LineStyle,...)`
  - `plot(...,'PropertyName',PropertyValue,...)`
  - `h = plot(...)`

3/17/2009

72/134

## plot ( ) con't: 2

- MATLAB defaults to plotting a blue line between points
- Other options exist:
  - Different color lines
  - Different types of lines
  - No line at all!

3/17/2009

73/134

## plot ( ) con't: 3 – Color options

- Color options:
  - Yellow - 'y'
  - Magenta - 'm'
  - Cyan - 'c'
  - Red - 'r'
  - Green - 'g'
  - Blue - 'b'
  - White - 'w'
  - Black - 'k'
- Example:
  - >> temp=1:1:10;
  - >> plot(temp, 'y');

3/17/2009

74/134

## plot ( ) con't: 4 – Line options

- Line styles:
  - `'-'`: solid line (default)
  - `'--'`: dashed line
  - `'.'`: dotted line
  - `'-.'`: dash-dot line

3/17/2009

75/134

## plot ( ) con't: 5 – Line Markings

- `+` - plus sign
- `o` - circle
- `*` - asterisk
- `.` - Point
- `x` - cross
- `s` - square
- `d` - diamond
- `^` - upward pointing triangle
- `v` - downward pointing triangle
- `>` - right pointing triangle
- `<` - left pointing triangle
- `p` - five-pointed star (pentagram)
- `h` - six-pointed star (hexagram)

3/17/2009

76/134

## `polar()`

- Plot polar coordinates
- Syntax:
  - `polar(theta,rho)`
  - `polar(theta,rho,LineStyle)`
- **theta** – Angle counterclockwise from the 3 o'clock position
- **rho** – Distance from the origin

3/17/2009

77/134

## `polar()` con't: 2

- Line color, style and markings apply as they did in the example with `plot()`.

3/17/2009

78/134

## **bar( )**

- Creates a bar graph
- Syntax
  - `bar(Y)`
  - `bar(x,Y)`
  - `bar(...,width)`
  - `bar(...,'style')`
  - `bar(...,LineStyle)`

3/17/2009

79/134

## **hist( )**

- Creates a histogram plot
- Syntax:
  - `n = hist(Y)`
  - `n = hist(Y,x)`
  - `n = hist(Y,nbins)`

3/17/2009

80/134



## File I/O

- Both high-level and low-level file I/O
- High-level covered here

3/17/2009

81/134

## High-Level File I/O

- I/O = input/output; 3 important commands for input:
  - **csvread**: `M = CSVREAD('FILENAME')`  
reads a comma separated value formatted file **FILENAME**. The result is returned in **M**. The file can only contain numeric values.
  - **dlmread**: `RESULT= dlmread(FILENAME,DELIMITER)` reads numeric data from the ASCII delimited file **FILENAME** using the delimiter **DELIMITER**. The result is returned in **RESULT**. Use `'\t'` to specify a tab.
  - **textread**: `A = textread('FILENAME')` read formatted data from text file. It reads also numeric data from the file **FILENAME** into a single variable. If the file contains any text data, an error is produced.

3/17/2009

82/134

## csvread

- Read a comma-separated value file
- Syntax:
  - `a = csvread('filename')`
  - `a = csvread('filename',row,col)`
  - `a = csvread('filename',row,col,range)`
- Note – `csvread` does not like to read in text!

3/17/2009

83/134

## dlmread

- Like `csvread`, only instead of a comma, you specify the delimiter
- Syntax:
  - `a = dlmread(filename,delimiter)`
  - `a = dlmread(filename,delimiter,R,C)`
  - `a = dlmread(filename,delimiter,range)`
- Treat this like a generalized form of `csvread`.

3/17/2009

84/134

## textread

- Reads formatted data from a text file
- Syntax:
  - `[A,B,C,...] = textread('filename','format')`
  - `[A,B,C,...] = textread('filename','format',N)`
  - `[...] = textread(...,'param','value',...)`
- Useful, but try to do without it, MATLAB is somewhat slower when dealing with text data

3/17/2009

85/134

## Delimiters

- Delimiter: A character or sequence of characters marking the beginning or end of a unit of data.
- Ex. `1,2,3` (the delimiter would be `,`)
- Also `1:2:3` (the delimiter would be `:`)

3/17/2009

86/134

## Delimiters, con't: 2

- The most common delimiter is a comma: hence the term csv (CSV, *i.e.* Comma Separated Value) or comma separated values.
- Microsoft Excel can read csv formatted files

3/17/2009

87/134

## High Level File Output

- Some of the input commands have corresponding high-level output commands
- `csvwrite`
- `dlmwrite`

3/17/2009

88/134

## csvwrite

- Write a matrix to a comma-separated value file
- Syntax:
  - `csvwrite('filename',M)`
  - `csvwrite('filename',M,row,col)`writes matrix `M` starting at offset `row`, and column `col` in the file. `row` and `col` are zero-based, that is `row=col=0` specifies first number in the file.
- Ex. `csvwrite('blah.csv',a);`

3/17/2009

89/134

## dlmwrite

- Writes a matrix `M` to a delimited file (using the delimiter you specify)
- Syntax:
  - `dmlwrite(filename,M,delimiter)`
  - `dmlwrite(filename,M,delimiter,row,col)`
- Ex. `dmlwrite('blah.txt',a,':');`

3/17/2009

90/134

## Low-Level file I/O

- `fopen`
- `fclose`
- `fprintf`
- `fgetl / fgets`

3/17/2009

91/134

## `fopen`

- Opens a file and returns the handle to the file object
- `File_ID = fopen('blah.txt')`
- Capturing the file handle is necessary to write or read to/from the file

3/17/2009

92/134

## **fclose**

- Closes a file associated with a specific file identification handle
- Ex. `fclose(File_ID);`
- Ex. `fclose('all');`

3/17/2009

93/134

## **fprintf**

- Multi-use: can output to a file or a screen
- Ex. `fprintf(fid,'%6.2f %12.8f\n',y);`
- `%6.2f` means a floating point with 6 leading decimals and 2 trailing
- Specifying 1 instead of `fid` will output to the screen

3/17/2009

94/134

## **fgetl / fgets**

- Get line and get string, respectively. **fgetl** will get you a line without the newline character at the end, while **fgets** will preserve the newline character (**\n**).
- Syntax:
  - `Line = fgetl(File_ID);`
  - `Line = fgets(File_ID);`

3/17/2009

95/134

## **Programming in MATLAB**

- Two types of files:
  - **Scripts**
  - **Functions**

3/17/2009

96/134



## MATLAB Scripts

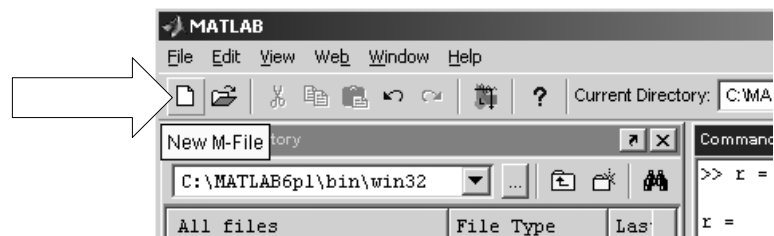
- Scripts are MATLAB commands stored in text files. When you type the name of the script file at the MATLAB prompt the commands in the script file are executed as if you had typed them in from the keyboard. Scripts end with the extension **.m**
- Referred to as **M-Files**

3/17/2009

97/134

## Script Files

- You can save a sequence of commands for reuse later
- Create a new M-File (script file)

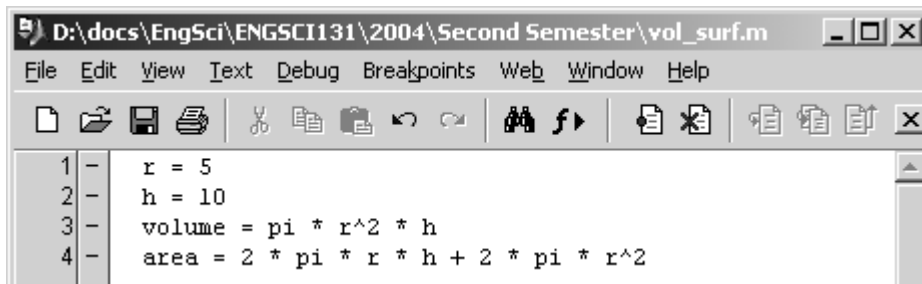


3/17/2009

98/134

## Script Files

- Each line is the same as typing a command in the command window
- Save the file as *filename.m*



```
D:\docs\EngSci\ENGSCI131\2004\Second Semester\vol_surf.m
File Edit View Text Debug Breakpoints Web Window Help
[Icons]
1 - r = 5
2 - h = 10
3 - volume = pi * r^2 * h
4 - area = 2 * pi * r * h + 2 * pi * r^2
```

3/17/2009

99/134

## Script Files

- Run the sequence of commands by typing the **filename** in the command window

```
>> vol_surf
r =
    5
h =
   10
volume =
  785.3982
area =
  471.2389
>>
```

3/17/2009

00/134

## MATLAB Functions

- Have input and output parameters
- MATLAB can return more than one variable at the end of a function
- Variables in scope in the MATLAB function go out of scope and are eliminated when the MATLAB function ceases to exist.

3/17/2009

101/134

## Programming in MATLAB

- Two types of files:
  - **Scripts**
  - **Functions**

3/17/2009

102/134

## MATLAB Scripts

- Scripts are MATLAB commands stored in text files. When you type the name of the script file at the MATLAB prompt the commands in the script file are executed as if you had typed them in from the keyboard. Scripts end with the extension **.m**
- Referred to as **M-Files**

3/17/2009

103/134

## MATLAB Functions

- Functions are similar to scripts
- Functions may take arguments
- Functions may return one or more values

3/17/2009

104/134

## MATLAB Functions, con't: 2

- `function [output] = function_name(input_arguments)`
- The above is a **function header** and should be the first non-comment line in the function file
- Comments may be placed **below** the function header

3/17/2009

105/134

## MATLAB Functions, con't: 3

- Example function

```
function [output] = square(input)
%
% The function [output] = square(input)
% computes the square of its input
%
    output = input*input;
    return
```

- Body of functions can contain code just like scripts could
- Comment line will be the output of the command
  - `>> help square`

3/17/2009

106/134

## MATLAB Functions, con't: 4

- Another example function

```
function r = rank(A,tol)
%RANK Matrix rank.
% RANK(A) provides an estimate of the number of linearly
% independent rows or columns of a matrix A.
% RANK(A,tol) is the number of singular values of A
% that are larger than tol.
% RANK(A) uses the default tol = max(size(A)) * norm(A) * eps.

% Copyright 1984-2001 The MathWorks, Inc.
% $Revision: 5.10 $ $Date: 2001/04/15 12:01:33 $

s = svd(A);
if nargin==1
    tol = max(size(A)') * max(s) * eps;
end
r = sum(s > tol);
```

3/17/2009

107/134

## MATLAB Functions, con't: 5

- Help of the main functions...

- SVD Singular value decomposition.  
[U,S,V] = SVD(X) produces a diagonal matrix S, of the same dimension as X and with nonnegative diagonal elements in decreasing order, and unitary matrices U and V so that  $X = U*S*V'$ .  
  
S = SVD(X) returns a vector containing the singular values.
- NARGIN Number of function input arguments.  
Inside the body of a user-defined function, NARGIN returns the number of input arguments that were used to call the function.

3/17/2009

108/134

## Looping!

- Scripts and functions also allow the ability to loop using conventional **for** and **while** loops.
- Note that the interpreter also lets you do it, it is simply less easy to grasp

3/17/2009

109/134

## for Loops

- Common to other programming languages

```
for variable = expression  
statement  
...  
statement  
end
```

3/17/2009

110/134

## For Loops, con't: 2

- Example: (taken from MATLAB help)
- ```
a = zeros(k,k) % Pre-allocate matrix
for m = 1:k
    for n = 1:k
        a(m,n) = 1/(m+n -1);
    end
end
end
```

3/17/2009

111/134

## For Loops, con't: 3

- The looping variable is defined in much the same way that we defined arrays/vectors.
- Ex. `m = 1:k`
- Or, `m = 1:10`

3/17/2009

112/134



## For Loops, con't: 4

- Loops are shown to end by the keyword "end"
- Curly braces are not present to subdivide packets of code
- Make use of adequate white-space and tabbing to improve code readability

3/17/2009

113/134

## while Loops

- Similar to `while` loops in other languages

```
while expression  
    statement  
    ...  
end
```

3/17/2009

114/134

## **while Loops, con't: 2**

- Ex. (taken from help while)

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

3/17/2009

115/134

## **while Loops, con't: 3**

- Same notes apply to while loops.
- Code is separated by the keyword “**end**”

3/17/2009

116/134

## Looping conclusion

- Some other aspects of looping exist
- Use
  - >> `help while`
  - and
  - >> `help for`
  - to see them

3/17/2009

117/134

## MATLAB Code Optimization

- Two ways to optimize MATLAB code
- Vectorise code
- Pre-allocate matrices

3/17/2009

118/134

## Look Ahead

- Review of topics (when requested) or use Matlab **help** and its **helpdesk**
- Code generation for **Digital Control System CAD**
- Each laboratory class will introduce more information about **Matlab** and its **Toolboxes**

3/17/2009

119/134

## Review

- MATLAB commands
- Looping!
- Optimization

3/17/2009

120/134

## Case statements

- **Syntax**

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

3/17/2009

121/134

## Case statements, con't: 2

- **Ex. (taken from help case)**

```
method = 'Bilinear';
switch lower(method)
    case {'linear','bilinear'}
        disp('Method is linear')
    case 'cubic'
        disp('Method is cubic')
    case 'nearest'
        disp('Method is nearest')
    otherwise disp('Unknown method.')
end
```

Method is linear

NOTE – when case matches it will not execute all following cases.  
(Break not necessary).

3/17/2009

122/134

## if statements

- Ex. (taken from Matlab help)

```
if expression
    statements
elseif expression
    statements
else
    statements
end
```

3/17/2009

123/134

## if statements, con't: 2

- Ex. (taken again from Matlab help)

```
if I == J
    A(I,J) = 2;
elseif abs(I-J) == 1
    A(I,J) = -1;
else
    A(I,J) = 0;
end
```

3/17/2009

124/134

# MATLAB Code Optimization

- Two ways to optimize MATLAB code
- Vectorise code
- Pre-allocate matrices

3/17/2009

125/134

# More plotting

- **plotyy: example**

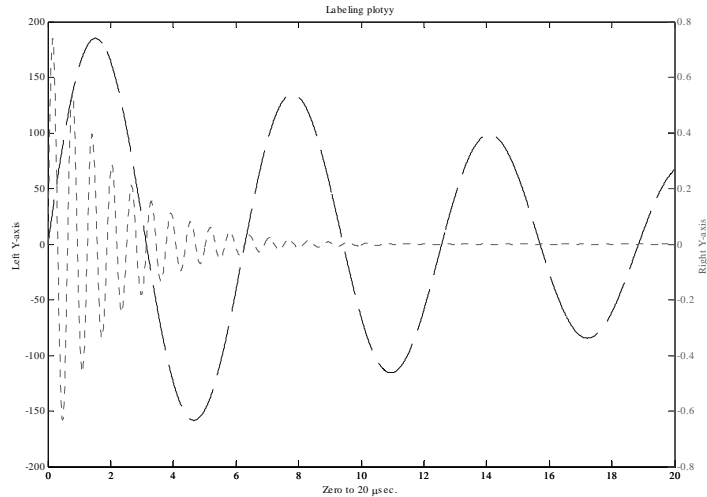
```
x = 0:0.01:20;
y1 = 200*exp(-0.05*x).*sin(x);
y2 = 0.8*exp(-0.5*x).*sin(10*x);
[AX,H1,H2] = plotyy(x,y1,x,y2,'plot');
set(get(AX(1),'Ylabel'),'String','Left Y-axis')
set(get(AX(2),'Ylabel'),'String','Right Y-axis')
xlabel('Zero to 20 \musec.')
title('Labeling plotyy')
set(H1,'LineStyle','--')
set(H2,'LineStyle',':')
```

3/17/2009

126/134

# More plotting

- `plotyy`: example



3/17/2009

127/134

# More plotting

- `plot3`: example

```
t = 0:pi/50:10*pi;  
plot3(sin(t),cos(t),t)  
grid on  
axis square
```

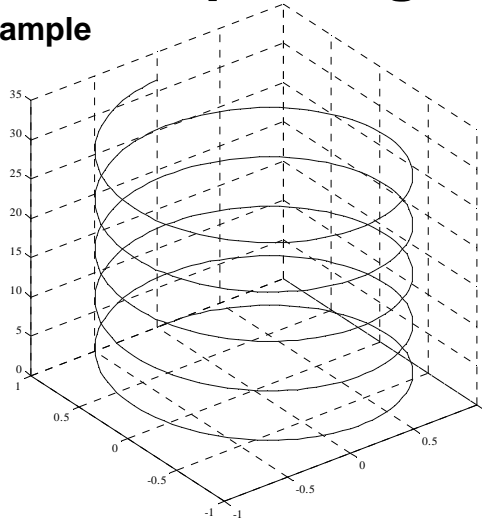
3/17/2009

128/134



## More plotting

- `plot3`: example



3/17/2009

129/134

## More plotting

- `bar3` example

```
Y = cool(7);

subplot(3,2,1)
bar3(Y,'detached')
title('Detached')

subplot(3,2,2)
bar3(Y,0.25,'detached')
title('Width = 0.25')

subplot(3,2,3)
bar3(Y,'grouped')
title('Grouped')
```

```
subplot(3,2,4)
bar3(Y,0.5,'grouped')
title('Width = 0.5')

subplot(3,2,5)
bar3(Y,'stacked')
title('Stacked')

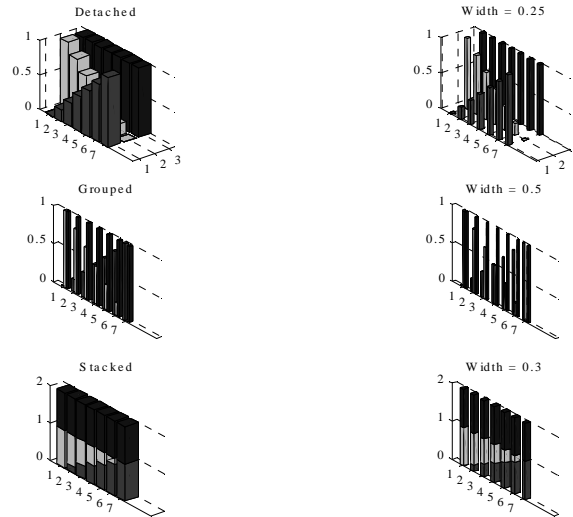
subplot(3,2,6)
bar3(Y,0.3,'stacked')
title('Width = 0.3')

colormap([1 0 0;0 1 0;0 0 1])
```

3/17/2009

130/134

• **bar3 example** **More plotting**



3/17/2009

131/134

**More plotting**

• **surf: 2 examples**

```
% Example 1
[X,Y,Z] = peaks(30);
surfc(X,Y,Z)
colormap hsv
axis([-3 3 -3 3 -10 5])
```

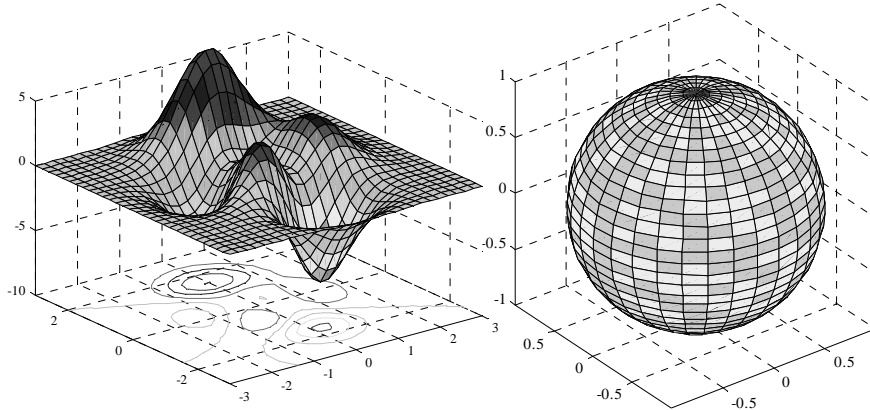
```
%Example 2
k = 5;
n = 2^k-1;
[x,y,z] = sphere(n);
c = hadamard(2^k);
surf(x,y,z,c);
colormap([1 1 0; 0 1 1])
axis equal
```

3/17/2009

132/134

## More plotting

- `surf`: 2 examples



**2 dimensional Gaussian**

3/17/2009

**Sphere**

133/134

## Matlab and its Toolboxes

**Direct application examples in  
laboratory room**

3/17/2009

134/134