Introduction to MATLAB

Simon O'Keefe
Non-Standard Computation Group
sok@cs.york.ac.uk

Content

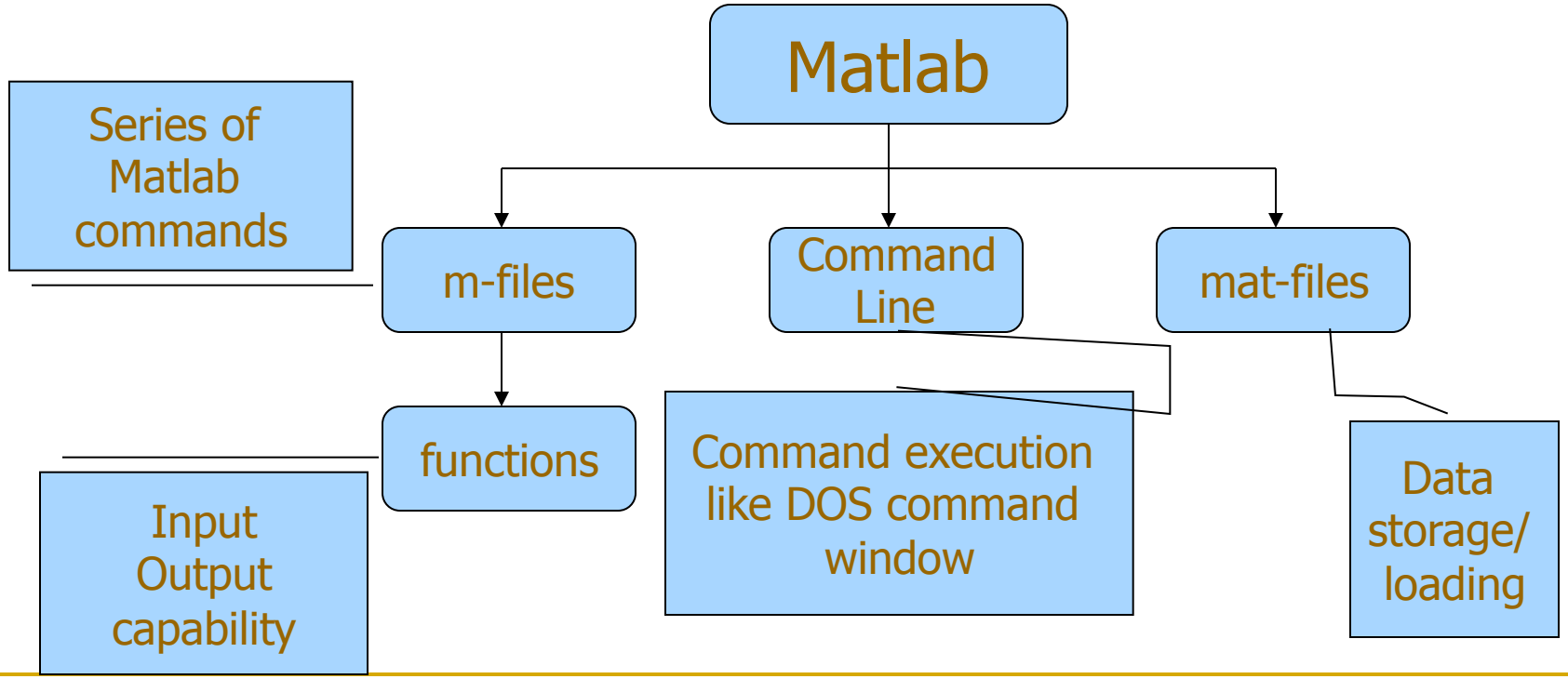
- An introduction to MATLAB
- The MATLAB interfaces
- Variables, vectors and matrices
- Using operators
- Using Functions
- Creating Plots

1 Introduction to MATLAB

- What is MATLAB?
  - MATLAB provides a language and environment for numerical computation, data analysis, visualisation and algorithm development
  - MATLAB provides functions that operate on
    - Integer, real and complex numbers
    - Vectors and matrices
    - Structures

What are we interested in?

- Matlab is too broad for our purposes in this course.
- The features we are going to require is



1 MATLAB Functionality

- Built-in Functionality includes
  - Matrix manipulation and linear algebra
  - Data analysis
  - Graphics and visualisation
  - …and hundreds of other functions ☺
- Add-on toolboxes provide*
  - Image processing
  - Signal Processing
  - Optimization
  - Genetic Algorithms

…* but we have to pay for these extras ☹

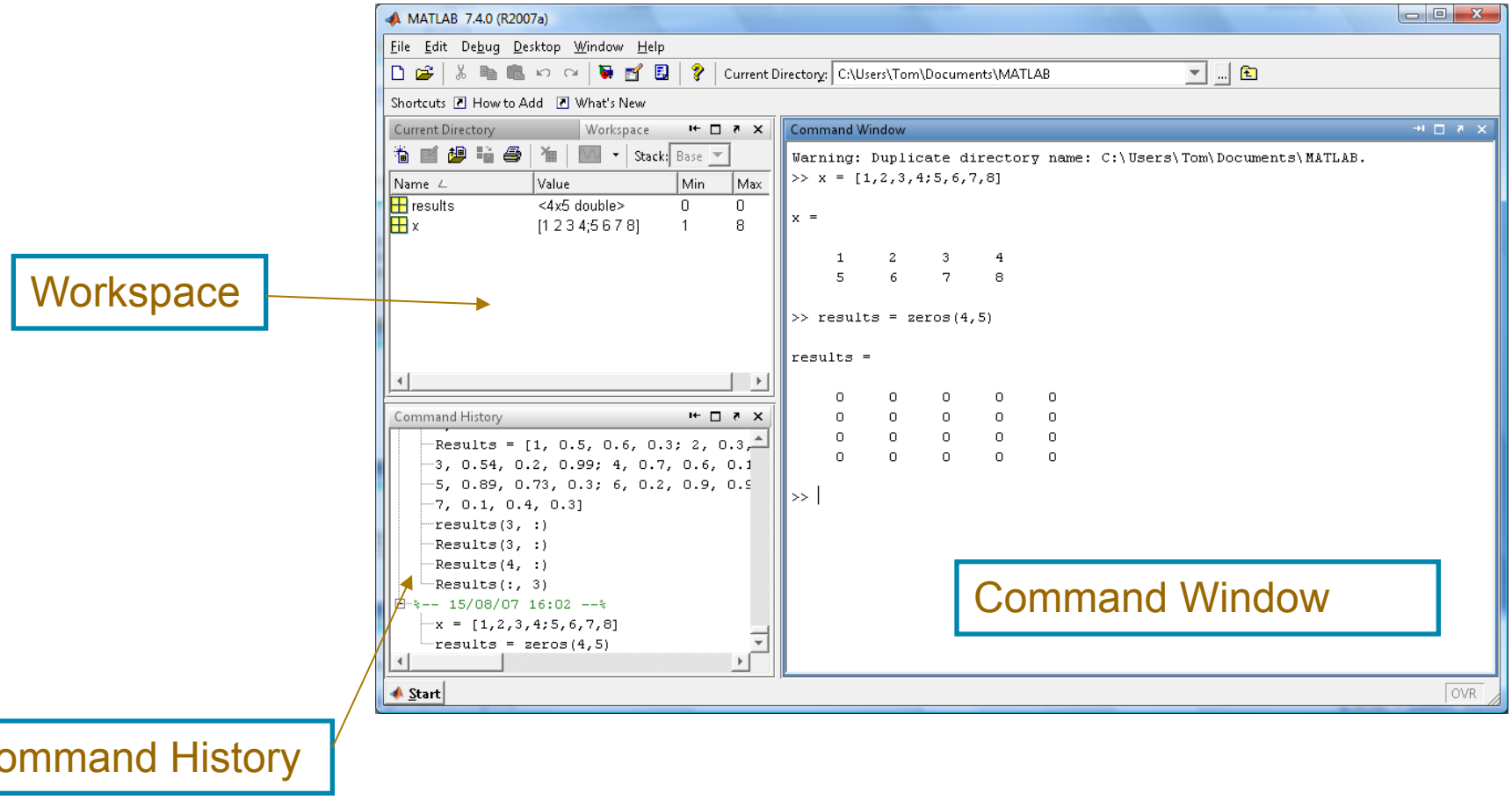
1 MATLAB paradigm

- MATLAB is an interactive environment
  - Commands are interpreted one line at a time
  - Commands may be scripted to create your own functions or procedures
- Variables are created when they are used
- Variables are typed, but variable *names* may be reused for different types
- Basic data structure is the matrix
  - Matrix dimensions are set dynamically
- Operations on matrices are applied to all elements of a matrix at once
  - Removes the need for looping over elements one by one!
  - Makes for fast & efficient programmes

1 Starting and stopping

- To Start
  - On Windows XP platform select
  - `Start->Programs->Maths and Stats->`
    `MATLAB->MATLAB_local->R2007a->MATLAB R2007a`
  - For access to the Genetic Algorithms and Stats toolboxes, you must use R2007b on Windows
    - MATLAB runs on Linux quite happily but we do not have toolbox licences
- To stop (nicely)
  - Select `File -> Exit MATLAB`
  - Or type `quit` in the MATLAB command window

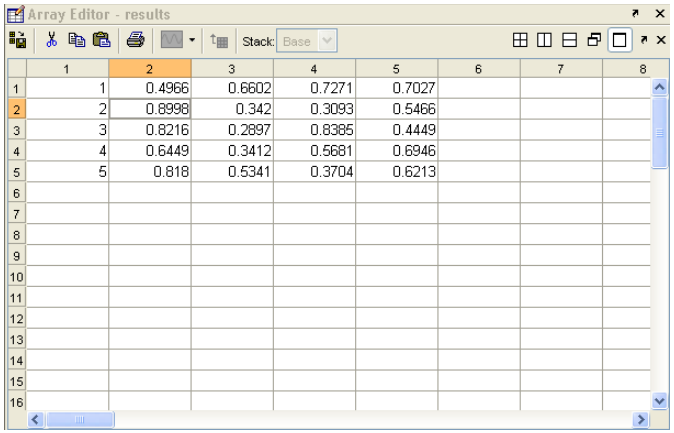
1 The MATLAB interfaces



1 Window Components

- Command Prompt – MATLAB commands are entered here.
- Workspace – Displays any variables created (Matrices, Vectors, Singles, etc.)
- Command History - Lists all commands previously entered.

- Double clicking on a variable in the Workspace will open an Array Editor. This will give you an Excel-like view of your data.



	1	2	3	4	5	6	7	8
1	1	0.4966	0.6602	0.7271	0.7027			
2	2	0.8998	0.342	0.3093	0.5466			
3	3	0.8216	0.2897	0.8385	0.4449			
4	4	0.6449	0.3412	0.5681	0.6946			
5	5	0.818	0.5341	0.3704	0.6213			
6								
7								
8								
9								
10								
11								
12								
13								
14								
15								
16								

1 The MATLAB Interface

- Pressing the up arrow in the command window will bring up the last command entered
  - This saves you time when things go wrong
- If you want to bring up a command from some time in the past type the first letter and press the up arrow.
- The current working directory should be set to a directory of your own

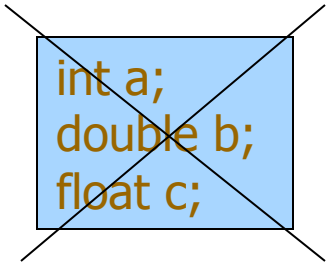
2 Variables, vectors and matrices

2.1 Creating Variables

- Variables
  - Names
    - Can be any string of upper and lower case letters along with numbers and underscores but it must begin with a letter
    - Reserved names are IF, WHILE, ELSE, END, SUM, etc.
    - Names are case sensitive
  - Value
    - This is the data the is associated to the variable; the data is accessed by using the name.
  - Variables have the type of the last thing assigned to them
    - Re-assignment is done silently – there are no warnings if you overwrite a variable with something of a different type.

Variables

- No need for types. i.e.,



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

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

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

2.1 Single Values

- Singletons
  - To assign a value to a variable use the equal symbol '='

    >> A = 32

```
>> A = 32

A =

     32

>>
```

  - To find out the value of a variable simply type the name in

```
>> A

A =

     32

>>
```

2.1 Single Values

- To make another variable equal to one already entered

  >> B = A

- The new variable is not updated as you change the original value

Note: using ; suppresses output

```
>> B = A

B =

    32

>> A = 15

A =

    15

>> B

B =

    32

>>
```

2.1 Single Values

- The value of two variables can be added t and the result displayed…

  >> A = 10

  >> A + A

- …or the result can be stored in another va

  >> A = 10

  >> B = A + A

```
>> A = 10

A =

     10

>> A + A

ans =

     20

>> B = A + A

B =

     20

>>
```

2.1 Vectors

- A vector is a list of numbers
  - Use square brackets [] to contain the numbers

	A	B	C	D	E	F	G	H	I
1									
2	Test Results:	0.5	0.3	0.54	0.7	0.89	0.2	1	
3									
4									

- To create a row vector use ‘,’ to separate the content

```
>> Test_Results = [0.5, 0.3, 0.54, 0.7, 0.89, 0.2, 1]

Test_Results =

    0.5000    0.3000    0.5400    0.7000    0.8900    0.2000    1.0000

>>
```

2.1 Vectors

- To create a column vector use ';' to separate the content

	A	B
1	Test Results	
2	0.5	
3	0.3	
4	0.54	
5	0.7	
6	0.89	
7	0.2	
8	1	
9		

```
>> Test_Results = [0.5; 0.3; 0.54; 0.7; 0.89; 0.2; 1]

Test_Results =

    0.5000
    0.3000
    0.5400
    0.7000
    0.8900
    0.2000
    1.0000

>>
```

2.1 Vectors

- A row vector can be converted into a column vector by using the transpose operator ‘

```
>> Test_Results = [0.5, 0.3, 0.54, 0.7, 0.89, 0.2, 1]

Test_Results =

    0.5000    0.3000    0.5400    0.7000    0.8900    0.2000    1.0000

>> Test_Results = Test_Results'

Test_Results =

    0.5000
    0.3000
    0.5400
    0.7000
    0.8900
    0.2000
    1.0000

>>
```

2.1 Matrices

- A MATLAB matrix is a rectangular array of numbers
  - Scalars and vectors are regarded as special cases of matrices
  - MATLAB allows you to work with a whole array at a time

2.1 Matrices

- You can create matrices (arrays) of any size using a combination of the methods for creating vectors
- List the numbers using ‘,’ to separate each column and then ‘;’ to define a new row

	A	B	C	D	E
1	Results				
2					
3	Subject	Test 1	Test 2	Test 3	
4	1	0.5	0.6	0.3	
5	2	0.3	0.33	0.75	
6	3	0.54	0.2	0.99	
7	4	0.7	0.6	0.1	
8	5	0.89	0.73	0.3	
9	6	0.2	0.9	0.94	
10	7	1	0.4	0.3	
11					

```
>> Results = [1, 0.5, 0.6, 0.3; 2, 0.3, 0.33, 0.75;
              3, 0.54, 0.2, 0.99; 4, 0.7, 0.6, 0.1;
              5, 0.89, 0.73, 0.3; 6, 0.2, 0.9, 0.94;
              7, 1, 0.4, 0.3]

Results =

    1.0000    0.5000    0.6000    0.3000
    2.0000    0.3000    0.3300    0.7500
    3.0000    0.5400    0.2000    0.9900
    4.0000    0.7000    0.6000    0.1000
    5.0000    0.8900    0.7300    0.3000
    6.0000    0.2000    0.9000    0.9400
    7.0000    1.0000    0.4000    0.3000

>>
```

2.1 Matrices

- You can also use built in functions to create a matrix
  >> A = zeros(2, 4)

  creates a matrix called A with 2 rows and 4 columns containing the value 0

  >> A = zeros(5) or >> A = zeros(5, 5)

  creates a matrix called A with 5 rows and 5 columns

- You can also use:
  >> ones(rows, columns)
  >> rand(rows, columns)

Note: MATLAB always refers to the first value as the number of Rows then the second as the number of Columns

2.1 Clearing Variables

- You can use the command "clear all" to delete all the variables present in the workspace

- You can also clear specific variables using:

  >> clear Variable_Name

2.2 Accessing Matrix Elements

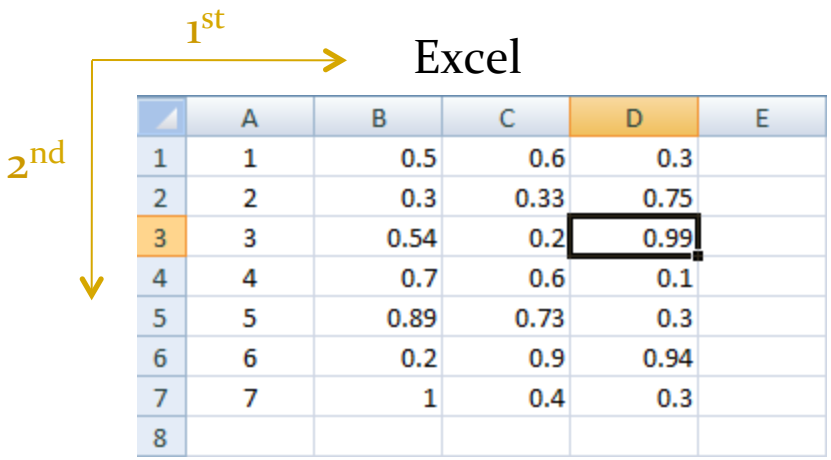
- An Element is a single number within a matrix or vector

- To access elements of a matrix type the matrices' name followed by round brackets containing a reference to the row and column number:

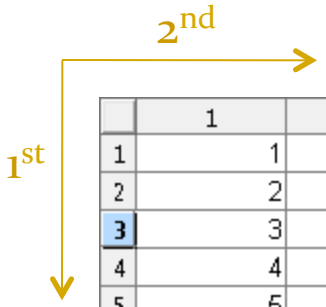
  >> Variable_Name(Row_Number, Column_Number)

NOTE: In Excel you reference a value by Column, Row. In MATLAB you reference a value by Row, Column

2.2 Accessing Matrix Elements



	A	B	C	D	E
1	1	0.5	0.6	0.3	
2	2	0.3	0.33	0.75	
3	3	0.54	0.2	0.99	
4	4	0.7	0.6	0.1	
5	5	0.89	0.73	0.3	
6	6	0.2	0.9	0.94	
7	7	1	0.4	0.3	
8					



	1	2	3	4	5
1	1	0.5	0.6	0.3	
2	2	0.3	0.33	0.75	
3	3	0.54	0.2	0.99	
4	4	0.7	0.6	0.1	
5	5	0.89	0.73	0.3	
6	6	0.2	0.9	0.94	
7	7	1	0.4	0.3	
8					

- To access Subject 3's result for Test 3
  - In Excel (Column, Row):

    D3
  - In MATLAB (Row, Column):

    >> results(3, 4)

2.2 Changing Matrix Elements

- The referenced element can also be changed

```
>> results(3, 4) = 10
```

or

```
>> results(3,4) = results(3,4) * 100
```

```
>> Results(3,4) = 10

Results =

    1.0000    0.5000    0.6000    0.3000
    2.0000    0.3000    0.3300    0.7500
    3.0000    0.5400    0.2000   10.0000
    4.0000    0.7000    0.6000    0.1000
    5.0000    0.8900    0.7300    0.3000
    6.0000    0.2000    0.9000    0.9400
    7.0000    1.0000    0.4000    0.3000

>>
```

```
>> Results(3,4) = Results(3,4) * 100

Results =

    1.0000    0.5000    0.6000    0.3000
    2.0000    0.3000    0.3300    0.7500
    3.0000    0.5400    0.2000  198.0000
    4.0000    0.7000    0.6000    0.1000
    5.0000    0.8900    0.7300    0.3000
    6.0000    0.2000    0.9000    0.9400
    7.0000    1.0000    0.4000    0.3000

>>
```

2.2 Accessing Matrix Rows

- You can also access multiple values from a Matrix using the : symbol
  - To access all columns of a row enter:

    >> Variable_Name(RowNumber, :)

	A	B	C	D	E
1	Subject	Test 1	Test 2	Test 3	
2	1	0.5	0.6	0.3	
3	2	0.3	0.33	0.75	
4	3	0.54	0.2	0.99	
5	4	0.7	0.6	0.1	
6	5	0.89	0.73	0.3	
7	6	0.2	0.9	0.94	
8	7	1	0.4	0.3	
9					

```
>> Results(4, :)

ans =

    4.0000    0.7000    0.6000    0.1000

>>
```

2.2 Accessing Matrix Columns

- To access all rows of a column
  - >> Variable_Name(:, ColumnNumber)

	A	B	C	D	E
1	Subject	Test 1	Test 2	Test 3	
2	1	0.5	0.6	0.3	
3	2	0.3	0.33	0.75	
4	3	0.54	0.2	0.99	
5	4	0.7	0.6	0.1	
6	5	0.89	0.73	0.3	
7	6	0.2	0.9	0.94	
8	7	1	0.4	0.3	
9					

```
>> Results(:, 3)

ans =

    0.6000
    0.3300
    0.2000
    0.6000
    0.7300
    0.9000
    0.4000

>>
```

2.2 Changing Matrix Rows or Columns

- These reference methods can be used to change the values of multiple matrix elements

- To change all of the values in a row or column to zero use

```
>> results(:, 3) = 0
```

```
>> Results(:,3) = 0

Results =

    1.0000    0.5000         0    0.3000
    2.0000    0.3000         0    0.7500
    3.0000    0.5400         0    0.9900
    4.0000    0.7000         0    0.1000
    5.0000    0.8900         0    0.3000
    6.0000    0.2000         0    0.9400
    7.0000    1.0000         0    0.3000

>>
```

```
>> results(:, 5) = results(:, 3) + results(:, 4)
```

```
>> Results(:, 5) = Results(:, 3) + Results(:, 4)

Results =

    1.0000    0.5000    0.6000    0.3000    0.9000
    2.0000    0.3000    0.3300    0.7500    1.0800
    3.0000    0.5400    0.2000    0.9900    1.1900
    4.0000    0.7000    0.6000    0.1000    0.7000
    5.0000    0.8900    0.7300    0.3000    1.0300
    6.0000    0.2000    0.9000    0.9400    1.8400
    7.0000    1.0000    0.4000    0.3000    0.7000

>>
```

2.2 Changing Matrix Rows or Columns

- To overwrite a row or column with new values
  >> results(3, :) = [10, 1, 1, 1]
  >> results(:, 3) = [1; 1; 1; 1; 1; 1; 1]

```
>> Results(3, :) = [10, 1, 1, 1]

Results =

    1.0000    0.5000    0.6000    0.3000
    2.0000    0.3000    0.3300    0.7500
   10.0000    1.0000    1.0000    1.0000
    4.0000    0.7000    0.6000    0.1000
    5.0000    0.8900    0.7300    0.3000
    6.0000    0.2000    0.9000    0.9400
    7.0000    1.0000    0.4000    0.3000

>>
```

```
>> Results(:, 3) = [1; 1; 1; 1; 1; 1; 1]

Results =

    1.0000    0.5000    1.0000    0.3000
    2.0000    0.3000    1.0000    0.7500
    3.0000    0.5400    1.0000    0.9900
    4.0000    0.7000    1.0000    0.1000
    5.0000    0.8900    1.0000    0.3000
    6.0000    0.2000    1.0000    0.9400
    7.0000    1.0000    1.0000    0.3000

>>
```

NOTE: Unless you are overwriting with a single value the data entered must be of the same size as the matrix part to be overwritten.

2.2 Accessing Multiple Rows, Columns

- To access consecutive Rows or Columns use : with start and end points:
- Multiple Rows:
  >> Variable_Name(start:end, :)
- Multiple Columns:
  >> Variable_Name(:, start:end)

```
>> Results(1:2, :)

ans =

    1.0000    0.5000    0.6000    0.3000
    2.0000    0.3000    0.3300    0.7500

>>

>> Results(:, 1:2)

ans =

    1.0000    0.5000
    2.0000    0.3000
    3.0000    0.5400
    4.0000    0.7000
    5.0000    0.8900
    6.0000    0.2000
    7.0000    1.0000

>>
```

2.2 Accessing Multiple Rows, Columns

- To access multiple non consecutive Rows or Columns use a vector of indexes (using square brackets [])
  - Multiple Rows:
    >>Variable_Name([index1, index2, etc.], :)
  - Multiple Columns:
    >>Variable_Name(:, [index1, index2, etc.])

```
>> Results([1,3], :)

ans =

    1.0000    0.5000    0.6000    0.3000
    3.0000    0.5400    0.2000    0.9900

>>
```

```
>> Results(:, [1,3])

ans =

    1.0000    0.6000
    2.0000    0.3300
    3.0000    0.2000
    4.0000    0.6000
    5.0000    0.7300
    6.0000    0.9000
    7.0000    0.4000

>>
```

2.2 Changing Multiple Rows, Columns

- The same referencing can be used to change multiple Rows or Columns

>> results([3,6], :) = 0

```
>> Results([3,6], :) = 0

Results =

    1.0000    0.5000    0.6000    0.3000
    2.0000    0.3000    0.3300    0.7500
         0         0         0         0
    4.0000    0.7000    0.6000    0.1000
    5.0000    0.8900    0.7300    0.3000
         0         0         0         0
    7.0000    1.0000    0.4000    0.3000

>>
```

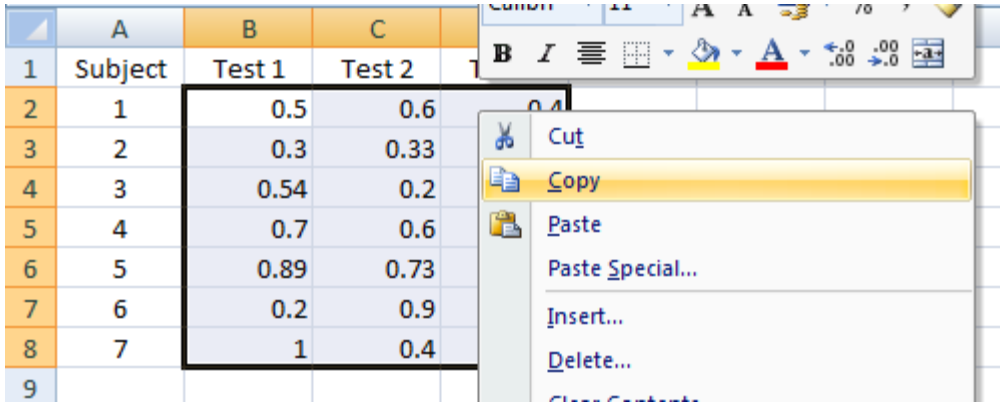
>> results(3:6, :) = 0

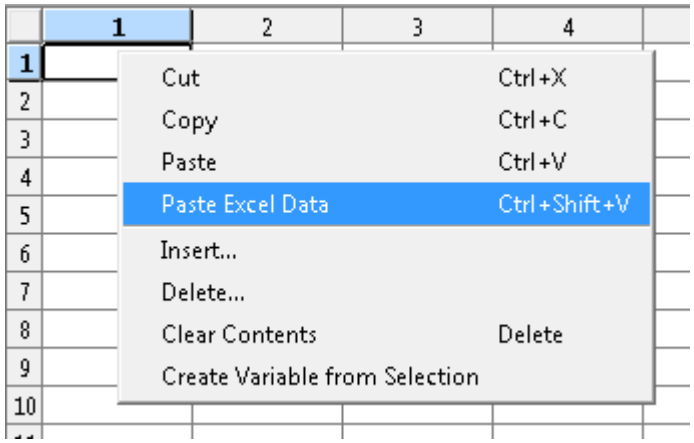
```
>> Results(3:6, :) = 0

Results =

    1.0000    0.5000    0.6000    0.3000
    2.0000    0.3000    0.3300    0.7500
         0         0         0         0
         0         0         0         0
         0         0         0         0
         0         0         0         0
    7.0000    1.0000    0.4000    0.3000

>>
```

2.3 Copying Data from Excel

- MATLAB's Array Editor allows you to copy data from an Excel spreadsheet in a very simple way
  - In Excel select the data and click on copy
  - Double click on the variable you would like to store the data in
    - This will open the array editor
  - In the Array Editor right click in the first element and select "Paste Excel Data"

2.3 Copying Data from Excel





	1	2	3
1	0.5	0.6	0.4
2	0.3	0.33	0.75
3	0.54	0.2	0.99
4	0.7	0.6	0.1
5	0.89	0.73	0.3
6	0.2	0.9	0.94
7	1	0.4	0.3

2.4 The colon operator

- The colon : is actually an operator, that generates a row vector
- This row vector may be treated as a set of indices when accessing a elements of a matrix
- The more general form is
  - [start:stepsize:end]

```
>> [11:2:21]
   11    13    15    17    19    21
>>
```

- Stepsize does not have to be integer (or positive)

```
>> [22:-2.07:11]
   22.00    19.93    17.86    15.79    13.72    11.65
>>
```

2.4 Concatenation

- The square brackets [] are the concatenation operator.
- So far, we have concatenated single elements to form a vector or matrix.
- The operator is more general than that – for example we can concatenate matrices (with the same dimension) to form a larger matrix

2.4 Saving and Loading Data

- Variables that are currently in the workspace can be saved and loaded using the save and load commands
- MATLAB will save the file in the Current Directory
- To save the variables use

  >> save File_Name [variable variable …]
- To load the variables use

  >> load File_Name [variable variable …]

3 More Operators

3.1 Mathematical Operators

- Mathematical Operators:
  - Add: +
  - Subtract: -
  - Divide: ./
  - Multiply: .*
  - Power: .^ (e.g. .^2 means squared)
- You can use round brackets to specify the order in which operations will be performed
- Note that preceding the symbol / or * or ^ by a '.' means that the operator is applied between pairs of corresponding elements of vectors of matrices

3.1 Mathematical Operators

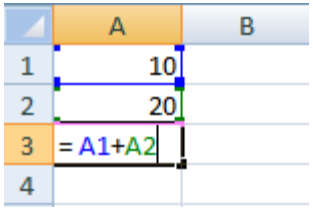
- Simple mathematical operations are easy in MATLAB
- The command structure is:

  >> Result_Variable =

  Variable_Name1 operator Variable_Name2

  - E.g. To add two numbers together:

    Excel:



    MATLAB:

    >> C = A + B

    >> C = (A + 10) ./ 2

3.1 Mathematical Operators

- You can apply single values to an entire matrix

  E.g.

  >> data = rand(5,1)

  >> A = 10

  >> results = data + A

```
>> data = rand(5, 1)

data =

    0.0967
    0.8181
    0.8175
    0.7224
    0.1499

>>
```

```
>> A = 10

A =

    10
```

```
>> data + A

ans =

   10.0967
   10.8181
   10.8175
   10.7224
   10.1499

>>
```

3.1 Mathematical Operators

- Or, if two matrices/vectors are the same size, you can perform these operations between them

  >> results = [1:5]'

  >> results2 = rand(5,1)

  >> results3 = results + results2

```
>> results = [1:5]'

results =

     1
     2
     3
     4
     5
```

```
>> results2 = rand(5, 1)

results2 =

    0.6596
    0.5186
    0.9730
    0.6490
    0.8003
```

```
>> results3 = results + results2

results3 =

    1.6596
    2.5186
    3.9730
    4.6490
    5.8003

>>
```

3.1 Mathematical Operators

- Combining this with methods from Accessing Matrix Elements gives way to more useful operations

```
>> results = zeros(3, 5)
>> results(:, 1:4) = rand(3, 4)
>> results(:, 5) = results(:, 1) + results(:, 2) + results(:, 3) + results(:, 4)
```

or

```
>> results(:, 5) = results(:, 1) .* results(:, 2) .* results(:, 3) .* results(:, 4)
```

NOTE: There is a simpler way to do this using the Sum and Prod functions, this will be shown later.

3.1 Mathematical Operators

```
>> results = zeros(3, 5)
>> results(:, 1:4) = rand(3, 4)
>> results(:, 5) = results(:, 1) + results(:, 2) + results(:, 3) + results(:, 4)
```

```
>> results = zeros(3, 5)

results =

     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
```

```
>> results(:, 1:4) = rand(3, 4)

results =

    0.4228    0.4177    0.7011    0.6981         0
    0.5479    0.9831    0.6663    0.6665         0
    0.9427    0.3015    0.5391    0.1781         0
```

```
>> results(:, 5) = results(:,1) + results(:,2) + results(:,3) + results(:,4)

results =

    0.4228    0.4177    0.7011    0.6981    2.2398
    0.5479    0.9831    0.6663    0.6665    2.8638
    0.9427    0.3015    0.5391    0.1781    1.9615

>>
```

3.1 Mathematical Operators

- You can perform operations on a matrix - you are very likely to use these
  - Matrix Operators:
    - Matrix Multiply: *
    - Matrix Right Division: /
- Example:

3.1 Operation on matrices

- Multiplication of matrices with * calculates inner products between rows and columns
- To transpose a matrix, use ‘
- det(A) calculates the determinant of a matrix A
- inv(A) calculates the inverse of a matrix A
- pinv(A) calculates the pseudo-inverse of A
- …and so on

3.2 Logical Operators

- You can use Logical Indexing to find data that conforms to some limitations

- Logical Operators:
  - Greater Than: >
  - Less Than: <
  - Greater Than or Equal To: >=
  - Less Than or Equal To: <=
  - Is Equal: ==
  - Not Equal To: ~=

3.2 Logical Indexing

- For example, you can find data that is above a certain limit:

```
>> r = results(:,1)
>> ind = r > 0.2
>> r(ind)
```

- ind is the same size as r and contains zeros (false) where the data does not fit the criteria and ones (true) where it does, this is called a Logical Vector.

- r(ind) then extracts the data where ones exist in ind

3.2 Logical Indexing

```
>> r = results(:,1)
>> ind = r > 0.2
>> r(ind)
```

```
>> r = results(:,1)

r =

    0.5000
    0.3000
    0.5400
    0.7000
    0.8900
    0.2000
    1.0000
```

```
>> ind = r > 0.2

ind =

     1
     1
     1
     1
     1
     0
     1
```

```
>> r(ind)

ans =

    0.5000
    0.3000
    0.5400
    0.7000
    0.8900
    1.0000
```

3.3 Boolean Operators

- Boolean Operators:
  - AND: &
  - OR: |
  - NOT: ~

- Connects two logical expressions together

3.3 Boolean Operators

- Using a combination of Logical and Boolean operators we can select values that fall within a lower and upper limit

```
>> r = results(:,1)
>> ind = r > 0.2 & r <= 0.9
>> r(ind)
```

- More later...

4 Functions

4 Functions

- A function performs an operation on the input variable you pass to it
- Passing variables is easy, you just list them within round brackets when you call the function
  - function_Name(input)
- You can also pass the function parts of a matrix

  >> function_Name(matrix(:, 1))

  or

  >> function_Name(matrix(:, 2:4))

```
>> mean(Results(:, 2:end))

ans =

    0.5900    0.5371    0.5257

>>
```

4 Functions

- The result of the function can be stored in a variable

  >> output_Variable = function_Name(input)

  e.g.

  >> mresult = mean(results)

- You can also tell the function to store the result in parts of a matrix

  >> matrix(:, 5) = function_Name(matrix(:, 1:4))

4 Functions

- To get help with using a function enter

  >> help function_Name

- This will display information on how to use the function and what it does

4 Functions

- MATLAB has many built in functions which make it easy to perform a variety of statistical operations
  - sum – Sums the content of the variable passed
  - prod – Multiplies the content of the variable passed
  - mean – Calculates the mean of the variable passed
  - median – Calculates the median of the variable passed
  - mode – Calculates the Mode of the variable passed
  - std – Calculates the standard deviation of the variable passed
  - sqrt – Calculates the square root of the variable passed
  - max – Finds the maximum of the data
  - min – Finds the minimum of the data
  - size – Gives the size of the variable passed

4 Special functions

- There are a number of special functions that provide useful constants
  - `pi` = 3.14159265….
  - `i` or `j` = square root of -1
  - `Inf` = infinity
  - `NaN` = not a number

4 Functions

- Passing a vector to a function like sum, mean, std will calculate the property within the vector

```
>> sum([1,2,3,4,5])
= 15

>> mean([1,2,3,4,5])
= 3
```

4 Functions

- When passing matrices the property, by default, will be calculated over the columns

```
>> results = [1,2,3,4,5;6,7,8,9,0]

results =

     1     2     3     4     5
     6     7     8     9     0

>> sum(results)

ans =

     7     9    11    13     5
```

```
>> mean(results)

ans =

    3.5000    4.5000    5.5000    6.5000    2.5000

>> std(results)

ans =

    3.5355    3.5355    3.5355    3.5355    3.5355
```

4 Functions

- To change the direction of the calculation to the other dimension (columns) use:

  >> function_Name(input, 2)

- When using std, max and min you need to write:

  >> function_Name(input, [], 2)

4 Functions

- From Earlier

  >> results(:, 5) = results(:, 1) + results(:, 2) + results(:, 3) + results(:, 4)

  or

  >> results(:, 5) = results(:, 1) .* results(:, 2) .* results(:, 3) .* results(:, 4)

- Can now be written

  >> results(:, 5) = sum(results(:, 1:4), 2)

  or

  >> results(:, 5) = prod(results(:, 1:4), 2)

4 Functions

- More usefully you can now take the mean and standard deviation of the data, and add them to the array

```
Results =

    1.0000    0.5000    0.6000    0.3000
    2.0000    0.3000    0.3300    0.7500
    3.0000    0.5400    0.2000    0.9900
    4.0000    0.7000    0.6000    0.1000
    5.0000    0.8900    0.7300    0.3000
    6.0000    0.2000    0.9000    0.9400
    7.0000    1.0000    0.4000    0.3000
```

```
>> Results(:, 5) = mean(Results(:, 2:4), 2)

Results =

    1.0000    0.5000    0.6000    0.3000    0.4667
    2.0000    0.3000    0.3300    0.7500    0.4600
    3.0000    0.5400    0.2000    0.9900    0.5767
    4.0000    0.7000    0.6000    0.1000    0.4667
    5.0000    0.8900    0.7300    0.3000    0.6400
    6.0000    0.2000    0.9000    0.9400    0.6800
    7.0000    1.0000    0.4000    0.3000    0.5667
```

```
>> Results(:, 6) = 2 * std(Results(:, 2:4), [], 2)

Results =

    1.0000    0.5000    0.6000    0.3000    0.4667    0.3055
    2.0000    0.3000    0.3300    0.7500    0.4600    0.5032
    3.0000    0.5400    0.2000    0.9900    0.5767    0.7925
    4.0000    0.7000    0.6000    0.1000    0.4667    0.6429
    5.0000    0.8900    0.7300    0.3000    0.6400    0.6102
    6.0000    0.2000    0.9000    0.9400    0.6800    0.8323
    7.0000    1.0000    0.4000    0.3000    0.5667    0.7572
```

4 Functions

- You can find the maximum and minimum of some data using the max and min functions

  >> max(results)

  >> min(results)

```
>> min(Results)

ans =

    0.2000    0.2000    0.1000

>> max(Results)

ans =

    1.0000    0.9000    0.9900

>>
```

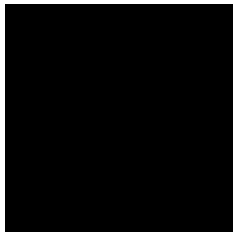
4 Functions

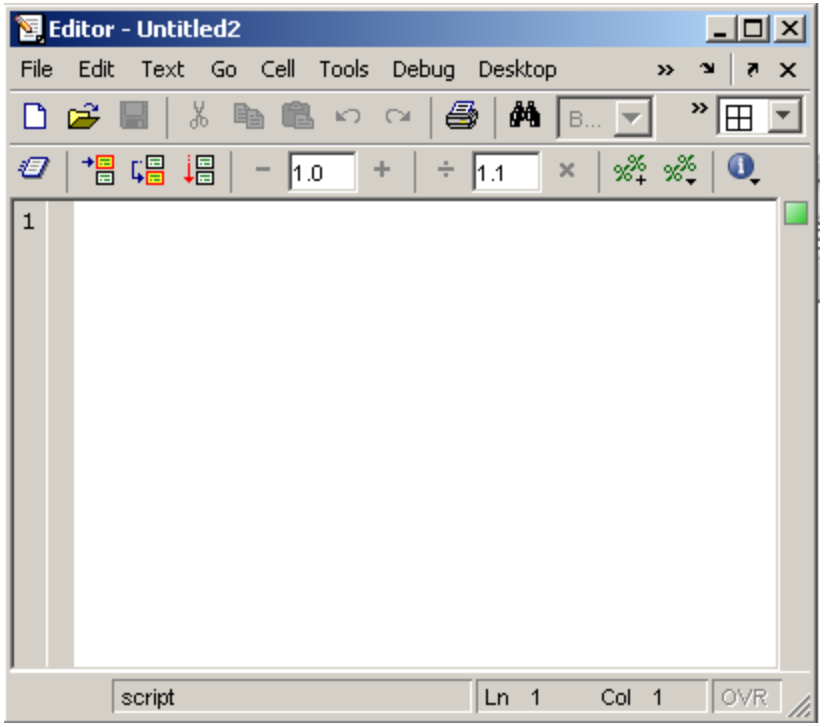
- We can use functions and logical indexing to extract all the results for a subject that fall between 2 standard deviations of the mean

```
>> r = results(:,1)
>> ind = (r > mean(r) – 2*std(r)) & (r < mean(r) + 2*std(r))
>> r(ind)
```

Use of M-File

Click to create
a new M-File

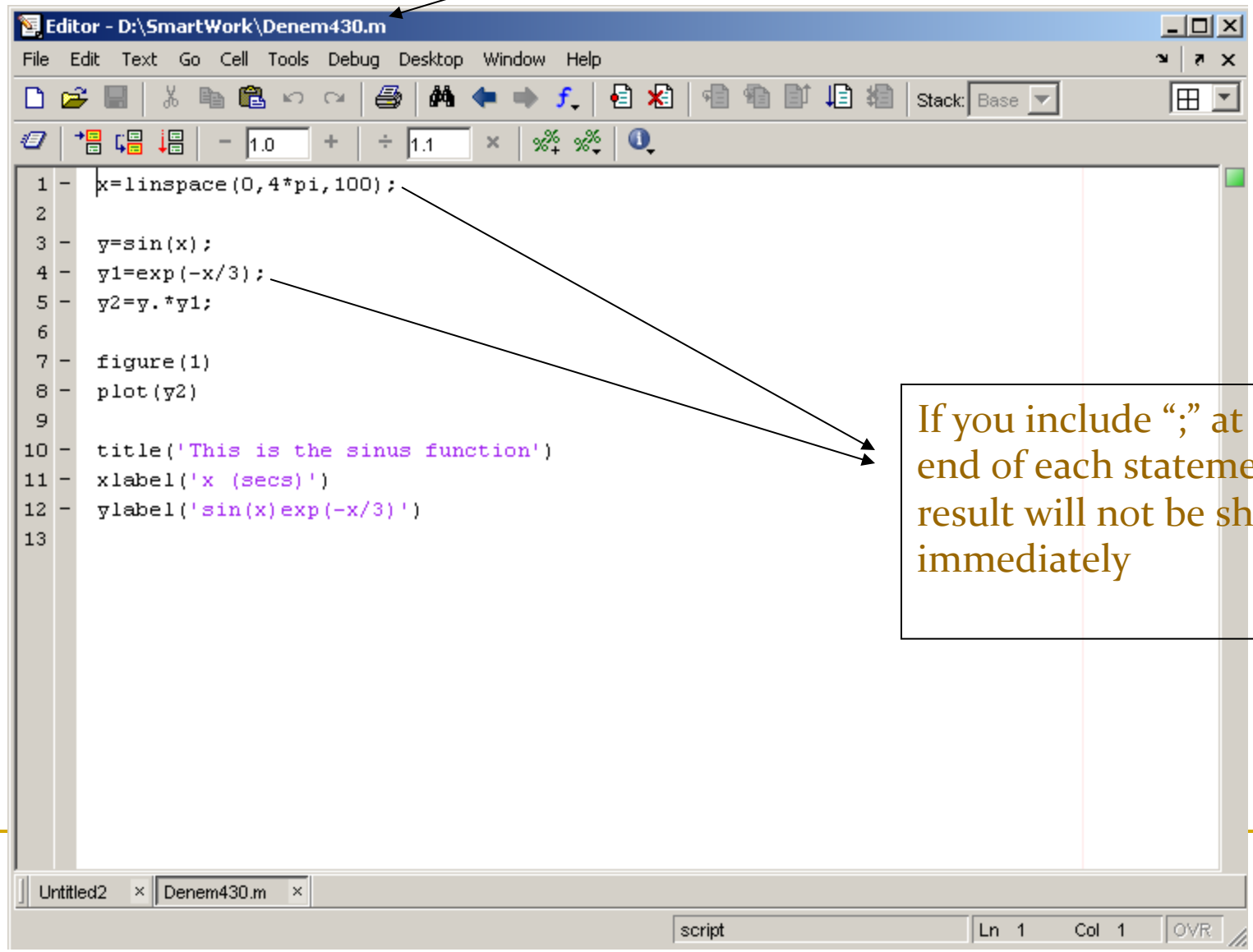




- Extension ".m"
- A text file containing script or function or program to run

Use of M-File

Save file as *Denem430*.m



If you include ";" at the end of each statement, result will not be shown immediately

Writing User Defined Functions

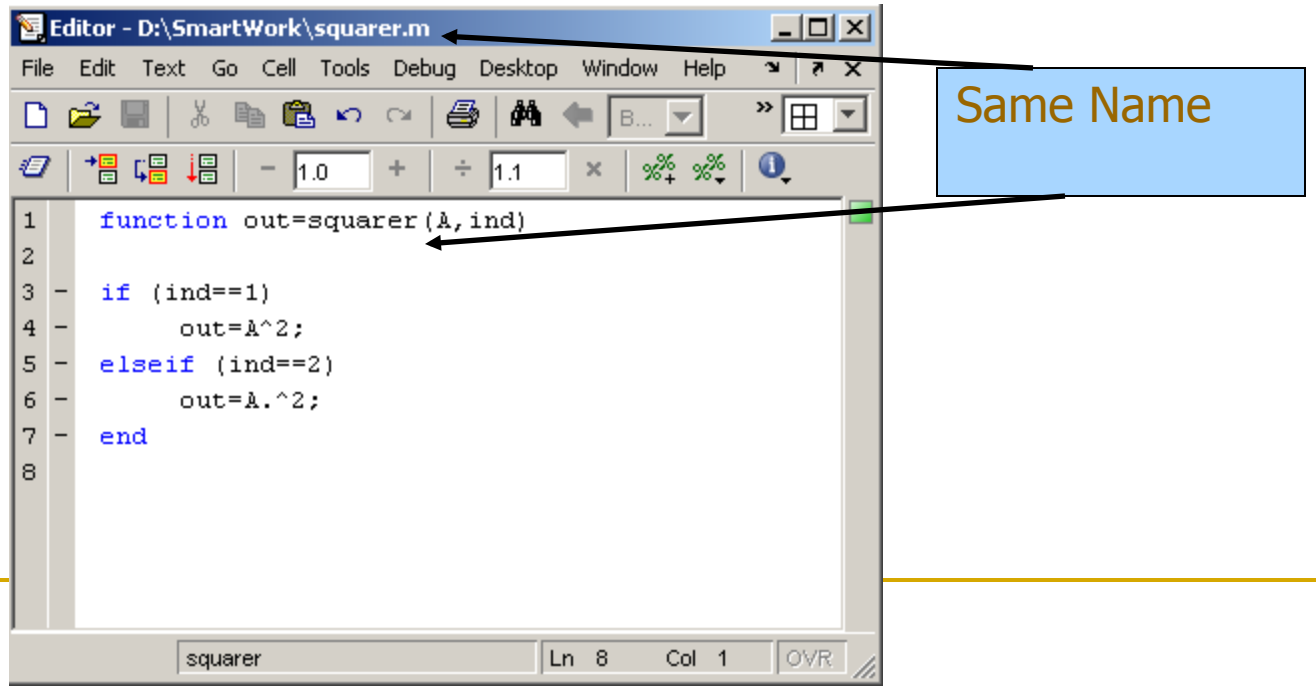
- Functions are m-files which can be executed by specifying some inputs and supply some desired outputs.
- The code telling the Matlab that an m-file is actually a function is

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

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

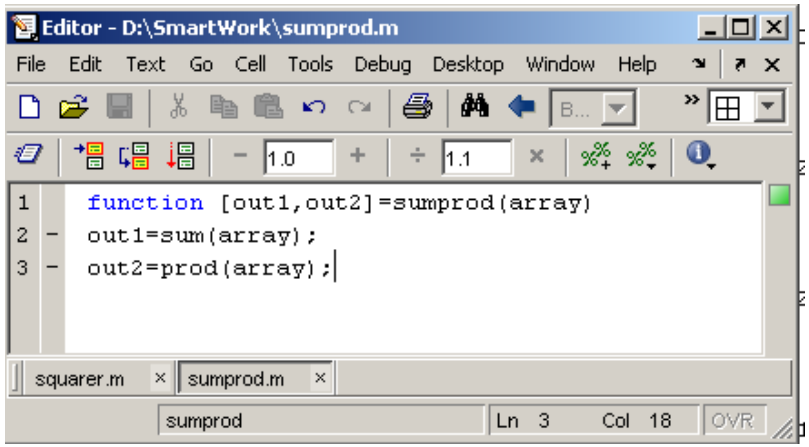
Writing User Defined Functions

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

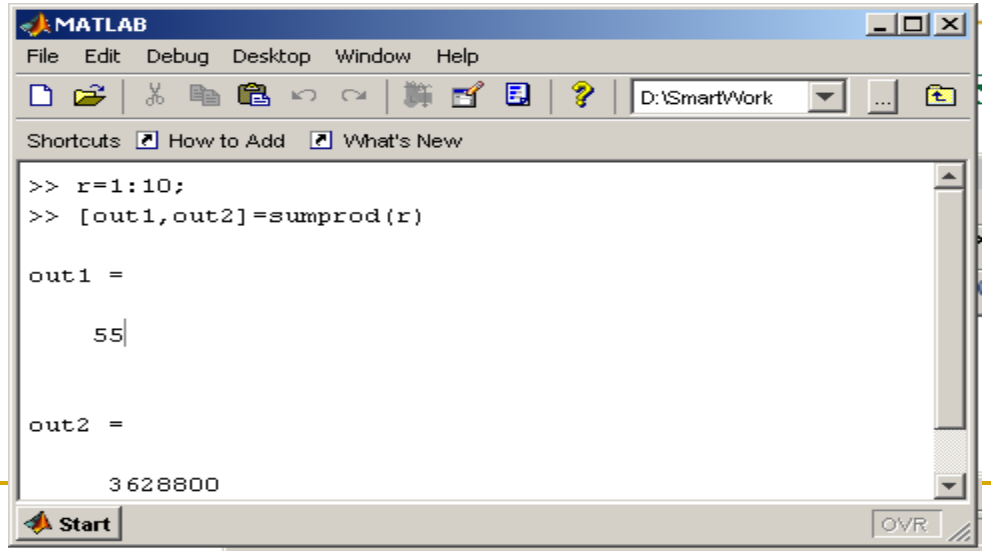


Writing User Defined Functions

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



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



5 Plotting

5 Plotting

- The plot function can be used in different ways:

  >> plot(data)

  >> plot(x, y)

  >> plot(data, 'r.-')

- In the last example the line style is defined

  Colour: r, b, g, c, k, y etc.

  Point style: . + * x o > etc.

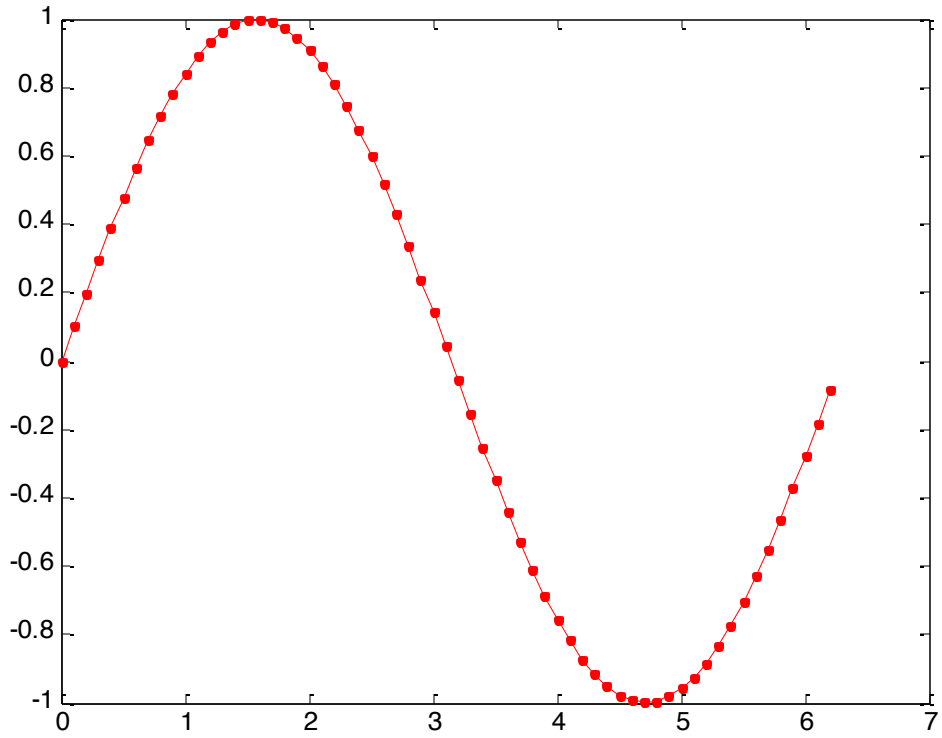
  Line style: - -- : .-

  - Type 'help plot' for a full list of the options

5 Plotting

- A basic plot

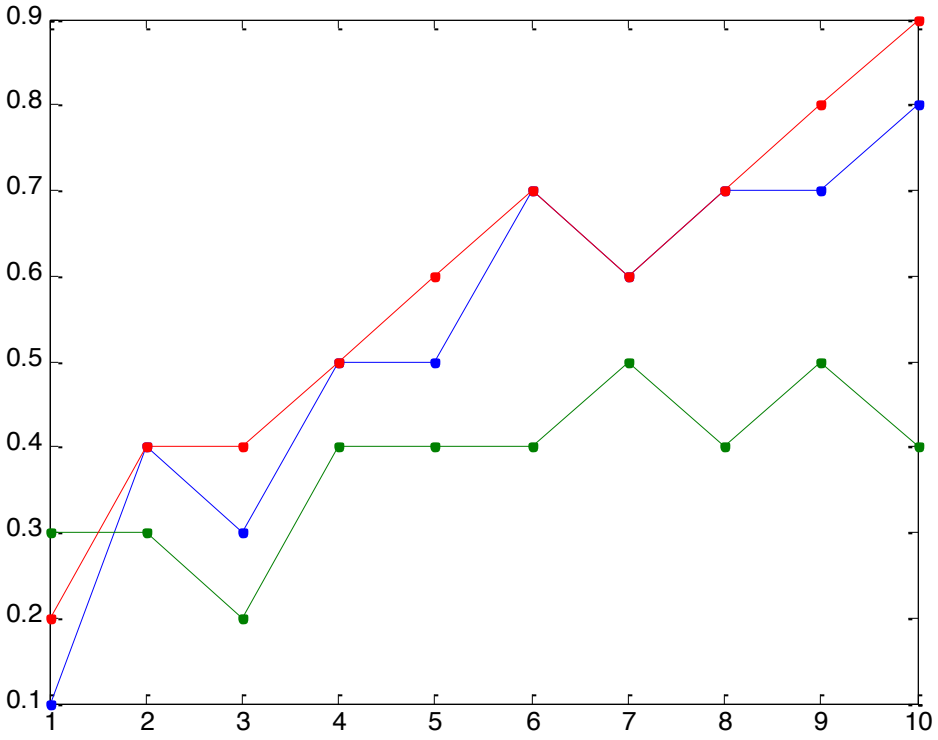
```
>> x = [0:0.1:2*pi]
>> y = sin(x)
>> plot(x, y, 'r.-')
```



5 Plotting

- Plotting a matrix
  - MATLAB will treat each column as a different set of data

```
results =

    0.1000    0.3000    0.2000
    0.4000    0.3000    0.4000
    0.3000    0.2000    0.4000
    0.5000    0.4000    0.5000
    0.5000    0.4000    0.6000
    0.7000    0.4000    0.7000
    0.6000    0.5000    0.6000
    0.7000    0.4000    0.7000
    0.7000    0.5000    0.8000
    0.8000    0.4000    0.9000

>> plot(results, '.-')
```

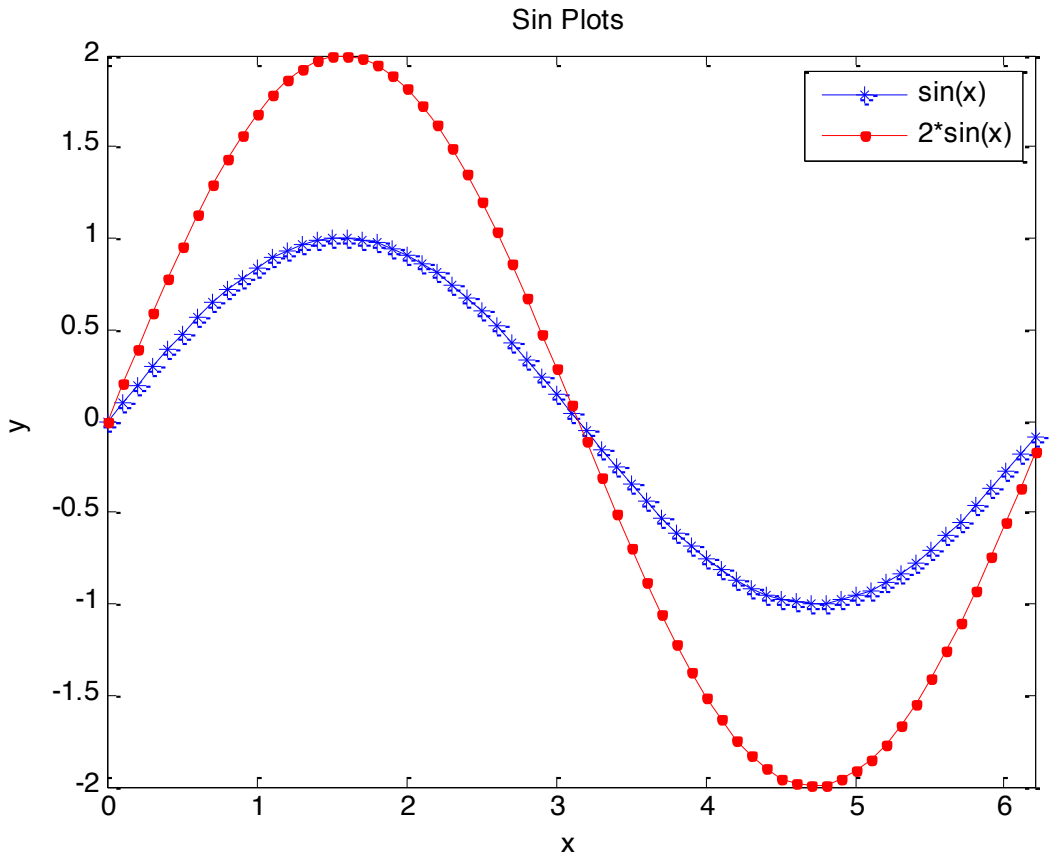


5 Plotting

- Some other functions that are helpful to create plots:
  - hold on and hold off
  - title
  - legend
  - axis
  - xlabel
  - ylabel

5 Plotting

```
>> x = [0:0.1:2*pi];
>> y = sin(x);

>> plot(x, y, 'b*-')

>> hold on

>> plot(x, y*2, 'r.-')

>> title('Sin Plots');

>> legend('sin(x)', '2*sin(x)');

>> axis([0 6.2 -2 2])

>> xlabel( 'x' );

>> ylabel( 'y' );

>> hold off
```

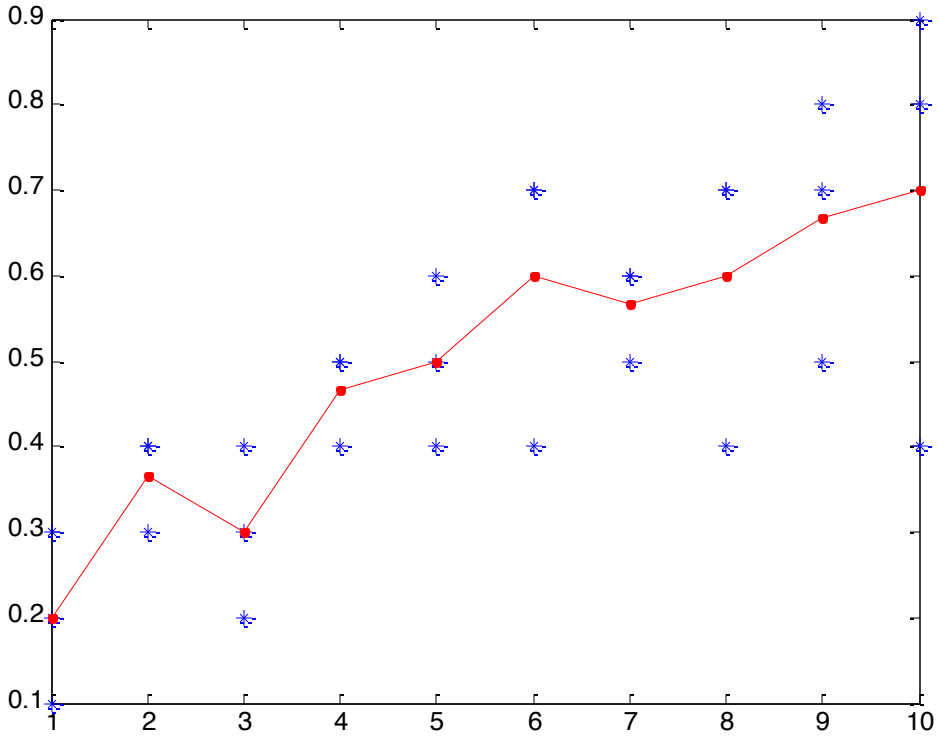


5 Plotting

- Plotting data

```
results =

    0.1000    0.3000    0.2000
    0.4000    0.3000    0.4000
    0.3000    0.2000    0.4000
    0.5000    0.4000    0.5000
    0.5000    0.4000    0.6000
    0.7000    0.4000    0.7000
    0.6000    0.5000    0.6000
    0.7000    0.4000    0.7000
    0.7000    0.5000    0.8000
    0.8000    0.4000    0.9000

>>
```

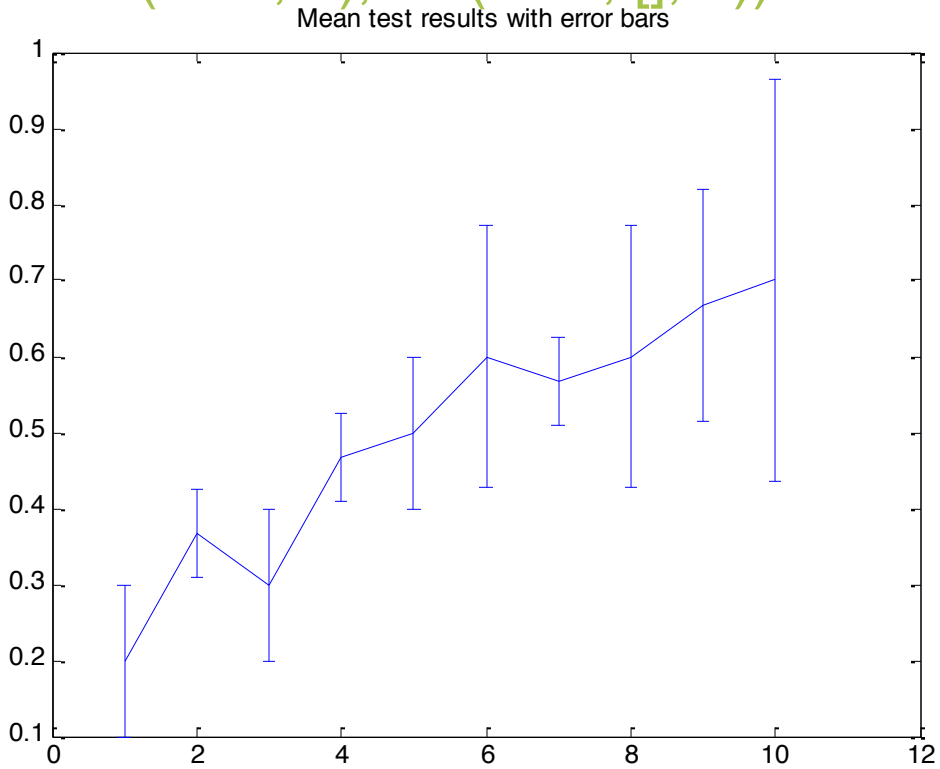
```
>> results = rand(10, 3)
>> plot(results, 'b*')
>> hold on
>> plot(mean(results, 2), 'r.-')
```



5 Plotting

- Error bar plot

```
>> errorbar(mean(data, 2), std(data, [], 2))
```



5 Plotting

- You can close all the current plots using ‘close all’

Notes:

- “%” is the neglect sign for Matlab (equaivalent of “//” in C). Anything after it on the same line is neglected by Matlab compiler.
- Sometimes slowing down the execution is done deliberately for observation purposes. You can use the command “pause” for this purpose

```
pause %wait until any key
pause(3) %wait 3 seconds
```

Useful Commands

- The two commands used most by Matlab users are

```
>>help functionname
```

```
>>lookfor keyword
```