

# A basic introduction to MATLAB

## Contents

- 1 What is Matlab ?
- 2 Getting started.
- 3 Built-in functions
- 4 Creating vectors and matrices in Matlab
- 5 Defining some common matrices and vectors (zeros, linspace etc)
- 6 Basic matrix operations
- 7 The dot symbol
- 8 Solving linear equations in Matlab
- 9 Plotting simple 2D graphs
- 10 Matlab works easily with vectors
- 11 Symbolic algebra and calculus within Matlab
- 12 Some easy graphics commands in 2D and 3D
- 13 Defining your own inline functions
- 14 Script files
- 15 Using some built-in functions

Appendix: Revising matrix multiplication

As you work through these notes there is space to write down your answers. I recommend that you do this. Also, think about your answers.

## 1. What is MATLAB ?

MATLAB started as an interactive program for doing matrix calculations and has now grown to a high level mathematical language that can solve integrals and differential equations numerically and plot a wide variety of two and three dimensional graphs.

In this subject you will mostly use it interactively like a super-calculator. You will also create MATLAB scripts that carry out a sequence of commands. MATLAB also contains a programming language that is rather like Pascal.

The first version of Matlab was produced in the mid 1970s as a teaching tool. The vastly expanded Matlab is now used for mathematical calculations and simulation in companies and government labs ranging from aerospace, car design, signal analysis through to instrument control & financial analysis. Other similar programs are Mathematica and Maple. Mathematica is somewhat better at symbolic calculations but is slower at large numerical calculations.

Recent versions of Matlab also include much of the Maple symbolic calculation program.

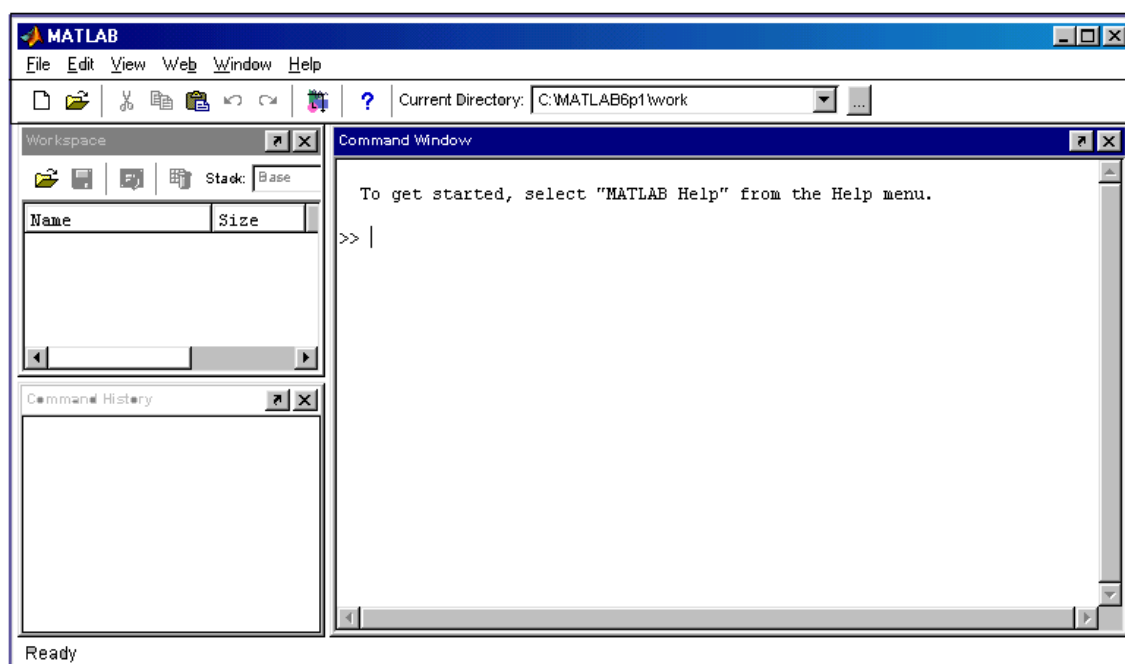
## 2 How to get started

When the computer has started go through the following steps in the different menus

- Look for the Network Applications folder and double click on that
- Within this you will see a little icon for Matlab61 – double click on that

Within about 30 seconds Matlab will open (you may get a little message box about ICA applications – just click OK on this)

You should now see a screen like the picture below



This is the default Matlab screen – it broken into 3 parts

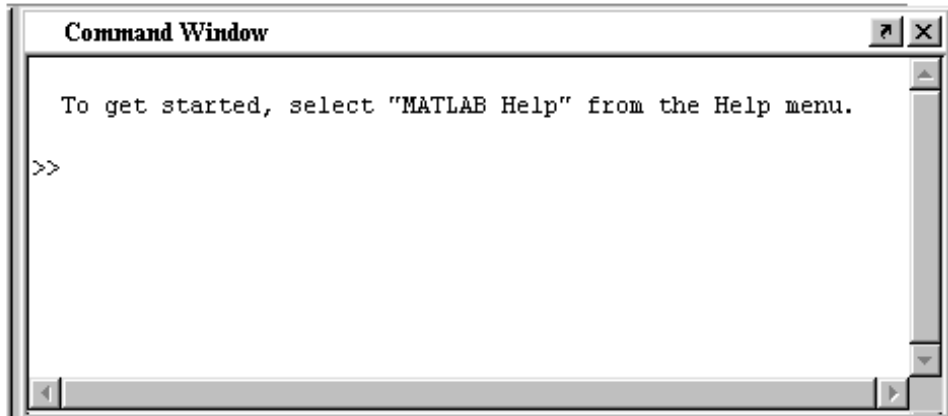
On the right you have the **Command Window** – this is where you type commands and usually the answers (or error messages) appear here too

On the top left you have the **Workspace** window – if you define new quantities (called variables) their names should be listed here.

On the bottom left you have **Command History** window – this is where past commands are listed. If you want to re-run a previous command or to edit it you can drag it from this window to the command window to re-run it.

## 2.1 The Matlab prompt is >> .

Look at the **Command Window** and you will see the cursor flickering after the >> prompt. This means that Matlab is waiting for further instructions.



## 2.2 Seeing the Matlab demo.

Just type the command

```
>> demo      (and then press Enter or Return key)
```

and run the simple slideshow under the **Matrix manipulation** option.

This briefly shows you some of the things that Matlab can do - don't worry if you do not know what everything means here.

### 2.3 Simple arithmetic with Matlab

The basic arithmetic operators are +, -, \* (for multiplication), / (for divide) & ^ for powers. Where necessary brackets should be used to make the order in which things are evaluated completely clear.

Enter the following commands to learn the basic way that formulas can be evaluated in Matlab (press enter after you have typed in the command. Record the answer beside the command.

>> **3 + 5 - 7** .....

The result of this calculation is stored in the temporary variable called **ans**. It is changed when you do another calculation.

If you want to save the answer from a calculation you can give it a name e.g.

>> **t = 3 + 5 - 7** .....

This line creates a variable *t* to save the answer in. If you want to find out what *t* is use

>> **t** .....

Equally you can use *t* in a formula

>> **2\*t + 2** .....

## 2.4 Correcting typing mistakes in Matlab

Sometimes you will make a mistake when you type a command into Matlab. Sometimes Matlab will detect the error and give a warning. Then you need to type in command correctly. Matlab provides a way to quickly edit the mistake using the up arrow key ↑

On most keyboards the ↑ key is in a group with the left and right arrows ← and →

### Example:

Suppose that you want to calculate  $\sin(\pi)$

The correct command is

```
>> sin(pi) [pi stands for  $\pi$  in Matlab]
```

Now make a deliberate error

```
>> sib(pi)
```

Matlab recognizes that sib is not a built-in function

```
>> sib(pi)
??? Undefined function or variable 'sib'.
```

Press the up-arrow key ↑ and the last command re-appears

```
>> sib(pi)
```

Now use the left and right arrow key or just click beside the b and change it to sin(3)

```
>> sin(pi) ← this is the correct command
```

## 2.4 The order of operations and putting brackets in can matter

In the following calculation does Matlab do the \* or the ^ first ?

>> 3^2\*4 .....

To find this out think about the two possible options

- One way to do the calculation is that powers are done before multiplication.  
This  $3^2 * 4 = \dots\dots\dots$
- The other option is to do the multiplication first is  $3^{2*4} = 3^8 = \dots\dots\dots$

In this case you should have obtained the answer 36.

Matlab interprets  $3^2*4$  as  $3^2 * 4$  - it does  $3^2$  first and then multiplies the answer by 4. Generally powers are done first, then multiplication and division and finally additions and subtractions.

One way to force Matlab (or any software) to do calculations in a defined order is to use brackets.

Compare the following commands and make sure that you understand what is happening and why you get the answer

>> 3 - 4/4 - 2 .....  $[3 - \frac{4}{4} - 2]$

>> (3-4)/(4-2) .....  $[\frac{3-4}{4-2}]$

>> (3-4)/4 - 2 .....  $[\frac{3-4}{4} - 2]$

The rules for evaluating expressions involving numbers are

- things inside brackets are done first
- within a bracket or expression generally powers (^) are done first, then multiplications and divisions and finally additions and subtractions.

## 2.5 Extended arithmetic (IEEE)

You need to understand the following section to interpret the output from some calculations – particularly when we make accidental errors.

What does Matlab produce if you ask it to find:

>> **1/0** .....

>> **-1/0** .....

>> **0/0** .....

>> **1/Inf** .....

The symbols **Inf** (meaning infinity) and **NaN** (**not a well defined number**) are part of a special standard for computer arithmetic. This is the IEEE international standard for extended arithmetic.

Typically you get **Inf** from 1/0 and **NaN** from 0/0 type calculations.

While Inf and NaN are usually signs that we have setup our problem incorrectly, Matlab will try to manipulate these results consistently

For example extended arithmetic will ensure that  $1/(1/x)$  always equals  $x$ .

Test this out by asking Matlab to calculate

>> **1/(1/0)** .....

Matlab can also handle **complex numbers** – they are entered as  $1 + i$  or  $-1+3*i$  etc

The symbol **i** is best kept as a reserved symbol (Matlab also interprets **j** as a complex number to fit in with usage in electrical engineering)

## 2.6 Matlab works to 15 significant figures but usually only shows 5 figures

Type **pi** in Matlab and you will normally get 3.1416

```
>> pi .....
```

To see all 15 figures you can change to the **long format** in which all 15 figures are displayed  
To 20 figures  $\pi = 3.14159\ 26535\ 89793\ 23846$

```
>> format long
```

```
>> pi .....
```

An easier and more flexible format is scientific notation with 5 significant figures

```
>> format short e
```

```
>> pi .....
```

Mostly we work in the short format.

To go back to short format

```
>> format short .....
```

Exponential format is particularly useful for very large or very small numbers- e.g.  $6.00e+23 = 6 \cdot 10^{23}$ . Another format is **format long e**.

## 2.7 The consequences of finite accuracy on a computer

Matlab only has limited accuracy (although it is more than enough for most uses)

Computers are normally set up to store numbers in a fixed amount of memory  
Matlab uses 64 bits and this lets it store numbers as large as  $2 \times 10^{308}$  or as small as  $2 \times 10^{-308}$  But no matter which number you work with Matlab can only store it to 15 significant figures

### For example:

Matlab could handle 1.234 567 890 234 56 (fifteen figures)

But a 20 digit number like 1. 234 567 890 234 567 890 12 is truncated to 15 figures.

This is an example of a **round-off**. Other cases of round-off occur if you add or multiply two 15 figure numbers and then only store the first 15 significant figures of the answer. **Round-off cannot be avoided in computer calculations.**

### This type of unavoidable 'error' can affect the answers you get

What is the exact value of  $\sin(\pi)$  .....

What does Matlab gives for

```
>> sin(pi) .....
```

Matlab uses a high order polynomial to approximate  $\sin(x)$ . This has a very slight roundoff error that give a very small non-zero answer instead of exactly zero

### Interpreting very small answers

Sometimes a answer that really ought to be exactly zero might appear as a small number like  $10^{-15}$ .

## 2.8 Variable names:

Matlab distinguishes between lower case and capital letters - so **A and a are different objects for Matlab**. The normal convention to use lower case names except for global variables. Only the first 19 characters in a variable name are important.

### 3. Matlab has many built in functions

Among them are:

**Trig functions**                      sin, cos, tan, sec, cosec and cotan  
(sin( $x$ ) etc assumes that  $x$  is in radians)

Are the following what you expect ?

>>    **sin(pi/2)**                      .....

>>    **cos(pi/2)**                      .....

**Remember that pi is a built-in constant**

**Inverse trig functions**            asin, acos, atan

(these are abbreviations for arcsin, arccos and arctan - the answers are returned in radians so that, for example, asin(1) = pi/2)

**Exponential function**exp

To calculate  $e^5$  we must use exp(5). The command  $e^5$  will not work.

**Logarithm functions**              log is log to base e, while  
log10 and log2 are logs to bases 10 or 2

**Hyperbolic functions**cosh, sinh, and tanh

Matlab also has many other more sophisticated functions for solving linear equations, getting eigenvalues of matrices, solving differential equations or calculating integrals numerically.

To find out what a built-in function does you can use the **help** command

**Example:**                      Use help to find out about the sin function

>>    **help log**

LOG	Natural logarithm.	← Matlab name
LOG(X)	is the natural logarithm of the elements of X.	← Brief explanation
	Complex results are produced if X is not positive.	← More information
	See also LOG2, LOG10, EXP, LOGM.	← related functions
Overloaded methods:	help sym/log.m	← another use of log

#### 4. Vectors and matrices in Matlab

Matlab is often used to work with matrices and vectors. Vectors are either row vectors or column vectors and it is usually important to be clear as to what kind of vector you mean.

Row vectors have one row and several columns like  $[4 \ 7 \ 10]$

Column vectors have several rows in one column like  $\begin{bmatrix} 4 \\ 7 \\ 10 \end{bmatrix}$

**The main use of matrices and vectors is in solving sets of linear equations:**

$$\begin{array}{l} \text{The equations} \quad 2x + 6y + 3z = 5 \\ \quad \quad \quad 3x + 8y + z = 7 \\ \quad \quad \quad x + 5y + 3z = 2 \end{array}$$

can be re-written in matrix/vector form as

$$\begin{bmatrix} 2 & 6 & 3 \\ 3 & 8 & 1 \\ 1 & 5 & 3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 5 \\ 7 \\ 2 \end{bmatrix}$$

Here the vectors are column vectors . The matrix contains the coefficients from the equations.

## 4.1 Entering row vectors

To enter the row vector  $v = [1 \ 2 \ 3 \ 4]$  into Matlab use the command

```
>> v = [ 1, 2, 3, 4]
```

The square brackets are essential in Matlab when defining an array.  
To check this see what Matlab has stored as  $v$

```
>> v .....
```

You can leave out the commas and simply put in a few spaces between different entries.

```
>> w = [1.2 -6.7 8.91] .....
```

## 4.2 Entering column vectors

Column vectors are like the ones that appear on the RHS of a set - e.g.  $\begin{bmatrix} 3 \\ 4 \\ 1.2 \end{bmatrix}$

This is a matrix with just one column and 3 rows.

**To create this column vector in Matlab you must mark the end of a row with a semi-colon ;**

```
>> a = [ 3 ; 4 ; 1.2 ] .....
```

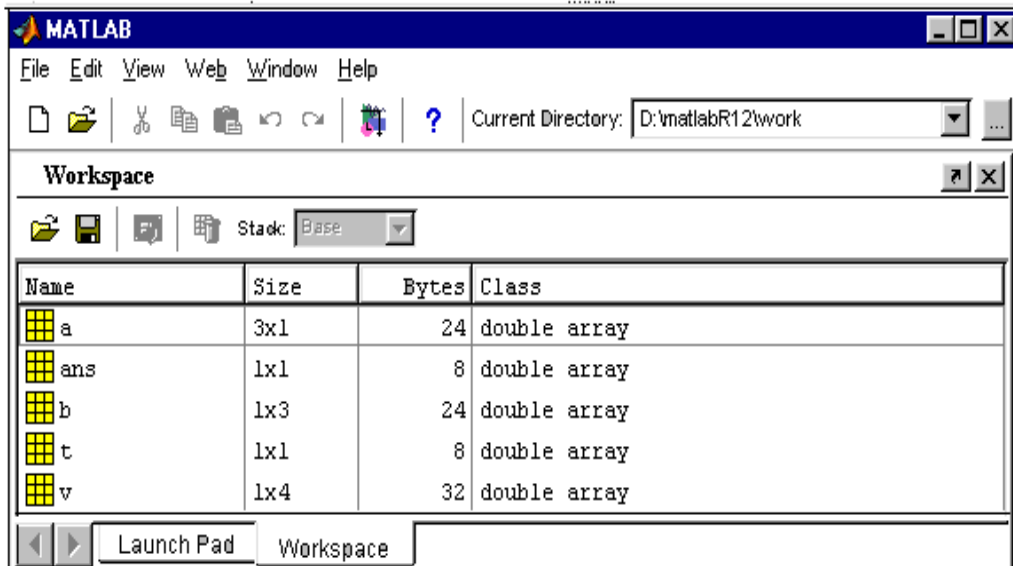
**The semi-colons are essential here.**

If you leave out the semicolons what will you get ?

```
>> b = [3 4 1.2] .....
```

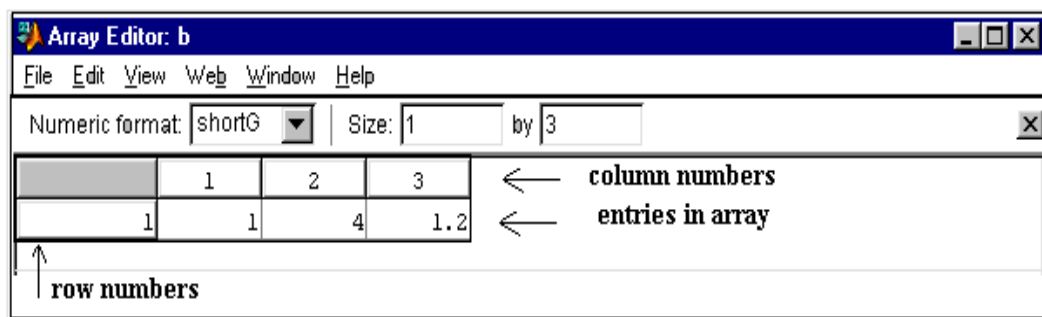
### 4.3 What variables have you defined already ?

Now look across at the **Workspace** window and you should see something like the picture below \_ you may need to expand this window a bit to see everything



This gives the name of the variable – its size as an array (1 x 1 is a single number). You can also see the content of an array by double clicking on the yellow box beside its name.

For example double clicking on b above gives



**When you open the array editor you can see the entries in the array - for large arrays you can scroll up/down and left/right**

If you want to keep in the command window you can just do **whos** and it prints a table like the window above.

#### 4.4 Creating matrices

To define a matrix we use the symbol ; to indicate the end of row.

Try the following to create the matrix  $p = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$

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

What output does this produce – how many rows & columns does it define:

Now repeat the definition of p but make the deliberate mistake

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

**Output:**

This is an example of a Matlab error message.  
Do you understand the error here ?

**Now check that this mistake has not altered the value of p:**

```
>> p
```

#### 4.5 Selecting numbers, rows and columns for a matrix

The matrix  $p$  that was just defined is  $\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$

Try the commands and comment on what they select:

```
>> p(1,2).....  
>> p(2,1) .....  
>> p(2,:) .....  
>> p(:,2) .....
```

Now use the information that you can get from these examples to write down the commands that will select. Check out your command to make sure that it gives the correct answer.

- the number in the 3rd row, 2nd column of  $p$  .....
- the number in the 1st row, 2nd column of  $p$  .....
- the first row from  $p$  .....
- the third row from  $p$  .....
- the first column from  $p$  .....
- the third column from  $p$  .....

## 5 Defining some common matrices and vectors

This section explains some special commands that will quickly create some simple vectors and matrices.

### 5.1 Vectors and matrices of zeros

To create a row vector containing 4 zeros check that the following command works

```
>> r = zeros ( 1, 4)
```

What does the following command do

```
>> r = zeros( 4, 1)
```

Based on these two examples how would you create a matrix of zeros with 3 rows and 4 columns

```
>> .....
```

### 5.2 Constant vectors

What do the following commands give

```
>> ones(1, 3) .....
```

```
>> ones(1,5) .....
```

```
>> ones(2,1) .....
```

```
>> 3*ones(1,3)
```

Can you find the Matlab commands to produce the following vectors or matrices

(i)  $a = [1, 1, 1, 1]$

(ii)  $b = \begin{bmatrix} 3 & 3 \\ 3 & 3 \\ 3 & 3 \end{bmatrix}$

### 5.3 Evenly spaced vectors

Examples of evenly spaced vectors are  $[1, 2, 3]$ ,  $[3, 7, 11, 15, 19, 23]$  and  $[-3, 0, 3, 6, 9]$

The most common place to use this type of vector is when we want to plot a graph.

Matlab gives us two ways to produce evenly spaced vectors:

- (i) The linspace command - we must specify the first and last numbers and the number of entries
- (ii) The colon method - here we must specify the first number, the gap between the numbers and the last number.

Both ways are useful in different situations but probably the linspace method is most widely used.

Here are the ways to get the three examples

(i)  $a = [1, 2, 3]$

```
>> a = linspace(1, 3, 3)
```

[Explanation: 1 = the first entry, 3 = last entry, 3 = the number of entries]

```
>> a = 1:1 : 3
```

 ← be careful to use colon symbol :

[Explanation 1 = the first entry, 1 = the gap between entries, 3 = last entry]

(ii)  $b = [3, 7, 11, 15, 19, 23]$

```
>> b = linspace(3, 23, 6)
```

```
>> b = 3:4:23
```

What are the two ways to get

There are two ways to produce  $c = [-3, 0, 3, 6, 9]$

```
>> c = .....
```

```
>> c = .....
```



## 6 Basic matrix operations

The symbols for matrix addition, subtraction, multiplication and powers are +, -, \* and ^

If  $a$  is a square matrix then  $a^2$  means  $a*a$  - the matrix product of  $a$  with itself.

Define the following matrices and see what happens with the matrix commands (see the last page to revise the definition of matrix multiplication)

Write Matlab commands that will create the following matrices

$$a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \quad b = \begin{bmatrix} 3 & 4 \\ 0 & 1 \end{bmatrix} \quad c = \begin{bmatrix} 3 & 4 \\ 2 & 1 \end{bmatrix} \quad x = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

>> **a** = .....

>> **b** = .....

>> **c** = .....

>> **x** = .....

Now try to find (if the answer exists check that it is correct by hand)

>> **b + c** .....

>> **b - 2\*c** .....

>> **a + b** .....

**Matlab also does matrix multiplication (provided that the matrices can be multiplied consistently).**

- First calculate  $b*a$  by hand (see appendix at end of notes for details)

$$b*a = \begin{bmatrix} 3 & 4 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} =$$

Now check that Matlab gives the same answer

>> **b\*a** .....

- First calculate  $b*x$  by hand (see appendix at end of notes for details)

$$b*x = \begin{bmatrix} 3 & 4 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} 2 \\ 1 \end{bmatrix} =$$

Now check that Matlab gives the same answer

>> **b\*x** .....

Now check that Matlab gives the same results for  $b^3$  and  $b*b*b$  (you don't need to do these by hand)

>> **b\*b\*b** .....

>> **b^3** .....

Does the product  $a*b$  make sense ?

>> **a\*b**

More complicated algebraic expressions like  $b + c - (b^2)*c$  can also be found.

## 7 The dot symbol

The dot symbol is special to Matlab – it is not a standard mathematical notation and should not be confused with the dot product of two vectors.

It changes the meaning of  $*$ ,  $/$  and  $^$  in a very important way.

**a^2** If  $a$  is a square matrix then **a^2** means  $a*a$  –  
It is the matrix product of  $a$  with itself (provided that the product makes sense)

**a.^2** But **a.^2** squares each element in  $a$  separately.  
Even if  $a$  is not square (so that  $a*a$  is not defined) we can still define **a.^2**

**a^2** and **a.^2** are usually quite different

**Example 1** Suppose that  $c = \begin{bmatrix} 3 & 4 \\ 2 & 1 \end{bmatrix}$

Because  $c$  is a  $2 \times 2$  matrix we can calculate  $c^2 = c*c$

Do the calculation yourself and check that you get  $c^2 = \begin{bmatrix} 17 & 16 \\ 8 & 9 \end{bmatrix}$

Now see what Matlab gives for  $c^2$  and  $c.^2$

```
>> c^2 .....
```

```
>> c.^2 .....
```

**Example 2** We have defined  $a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$

Can we multiply  $a*a$  as a regular matrix product ?

Predict the values of  $a.^2$  and check that you are right using Matlab

```
>> a.^2 .....
```

Now see what Matlab gives if you try to calculate  $a*a$

```
>> a*a .....
```

**Example 3** Suppose that  $b = [1, 2, -1]$  and  $a = [1, 2, 3]$  and  $c = [1, 2]$

Try the following Matlab commands and make sure that you understand the answers

```
>> b = ..... [ define b ]
```

```
>> 1./b .....
```

What has this calculated ?

```
.....
```

Try the commands

```
>> a = ..... [define a]
```

```
>> a.*b .....
```

Do you understand how this was found ?

Try the commands

```
>> c = ..... [ define c ]
```

```
>> a.*c .....  
.....
```

What happened here ?

What does this message mean ?

**Example 4** Using the dot command to evaluate functions

- If  $x = [0, 1, 2, 3]$  what command would you use to square the entries in  $x$

>> **x** = .....

>> .....

- What Matlab command would you use to get the values of  $y = x^2 - 6x + 5$  at 0, 1, 2 and 3 ?

>> **y** = .....

**Example 5** This example is more involved as it uses several dots

First calculate the values of  $y = \frac{x}{(x^2 + 1)}$  for  $x = 0, 1, 2$  and  $3$

$x$	0	1	2	3
$\frac{x}{(x^2 + 1)}$				

Now check the following Matlab command and make sure that it gives the same answers

>> **y** = **x./(x.^2 + 1)**

>> .....

Notice that there are two dots here – one goes with the division sign and the one goes with the power symbol (^). You do not need to put a dot with the + sign

## 8 The real power of Matlab - solving linear equations in one step

Suppose that you want to solve the equations

$$\begin{aligned}y + 2z &= 1 \\x + 2y + z &= 3 \\3x + 5y + 2z &= 7\end{aligned}$$

In matrix form this is 
$$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 1 \\ 3 & 5 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 7 \end{bmatrix}$$

In Matlab you can solve these equations in three steps.

We write this in the symbolic form  $A\mathbf{x} = \mathbf{b}$  [here vectors are written in bold letters]  
A is the 3x3 coefficient matrix and  $\mathbf{b}$  is the 3 row  $\times$  1 column vector on the RHS.

**Step 1 Define the matrix A - fill in the definition**

```
>> A = [ 0 1 2; 1 2 1; 3 5 2]
```

**Step 2 Define column vector b**

```
>> b = [ 1; 3; 7 ]
```

**Step 3 Solve for x using the backslash command**

```
>> x = A\b
```

(notice that we do not have to define a symbolic vector containing the  $x, y, z$  symbols. the variable  $x$  is the column vector containing the numerical solutions to the equations)

Matlab returns the value 
$$\mathbf{x} = \begin{bmatrix} -2.0000 \\ 3.0000 \\ -1.0000 \end{bmatrix}$$

**Warning: Be careful to get the order and the direction of the slash correct.  
 $\mathbf{b}\backslash\mathbf{A}$  or  $\mathbf{A}/\mathbf{b}$  will be quite different**

**Step 4 Check that this is really a solution by calculating the difference between the two sides of the equation – this is  $A*x - b$**

>>  $A*x - b$

Matlab returns the value  $\text{ans} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$

**Practice the use of the backslash command**

Use the  $\backslash$  command to solve the following two equations and check that the Matlab solution is correct.

(i) 
$$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 1 \\ 3 & 5 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 8 \end{bmatrix}$$

>>  $A = \dots\dots\dots$

>>  $b = \dots\dots\dots$

>>  $x = A \backslash b \dots\dots\dots$

Check that  $x$  really is a solution by calculating  $A*x$

>>  $A*x \dots\dots\dots$

(ii) 
$$\begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$

First write these equations out in detail – do they have a solution.

Then set up A and b in Matlab and see what you get.

>>  $A = \dots\dots\dots$

>>  $b = \dots\dots\dots$

>>  $x = A \backslash b \dots\dots\dots$

This set of equations does not have a solution and Matlab gives you a warning.

## 9 Plotting 2D graphs

One very useful feature of Matlab is its ability to plot graphs of functions quickly. This often uses the `linspace` command and the dot notation. For example suppose that we want to plot the graphs of  $y = x^2$  between 0 and 5.

**Step 1: Create a vector of 100 evenly spaced points between 0 & 5**

```
>> x = linspace( 0, 5, 100);
```

The semicolon (;) is used to stop Matlab printing all 100 numbers out. If you really want to check that the command worked use Data Editor to see what is in `x` now. I choose 100 to give a large number of points. You could have chosen any large number.

**Step 2 Compute the y values for the curve  $y = x^2$**

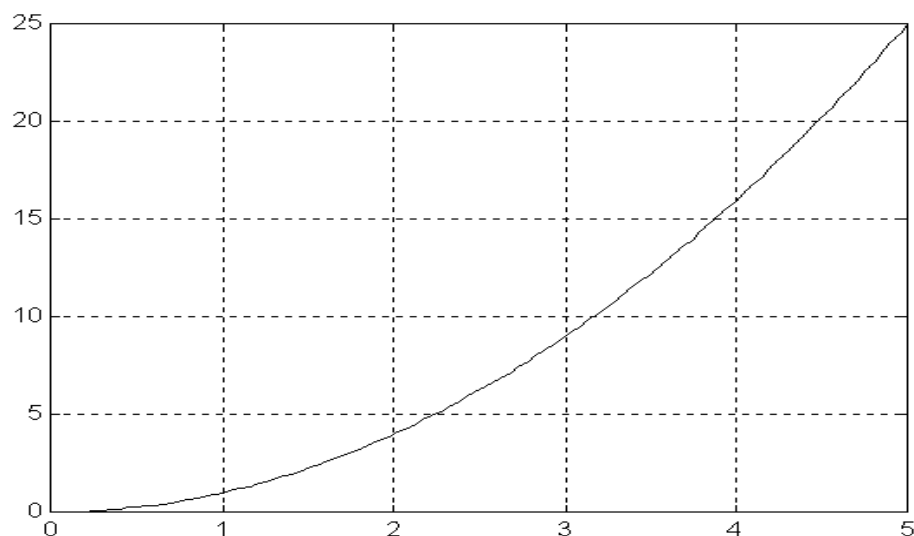
```
>> y = x.^2;
```

[the dot here is very important - it squares the elements of `x` one at a time]

**Step 3 Now plot just graph of  $y = x^2$**

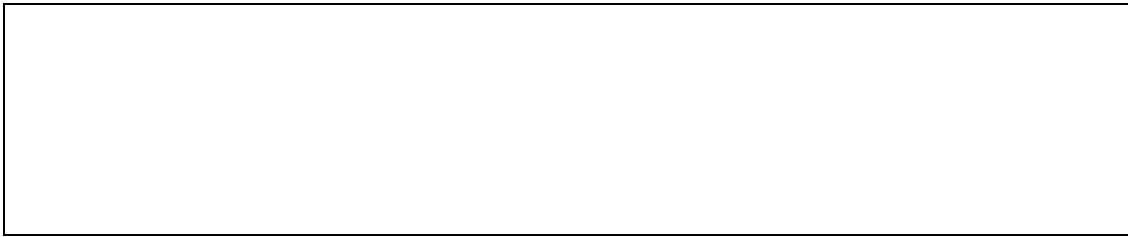
```
>> plot( x, y ); grid; shg
```

**Here I put three commands on the one line – `grid` puts dotted lines on the graph to make it easier to interpret. The `shg` command means ‘show graph’.**



You should see a graph drawn in blue [the default colour].

**Question:** What happens to the graph if you use 10 points instead of 100 ?

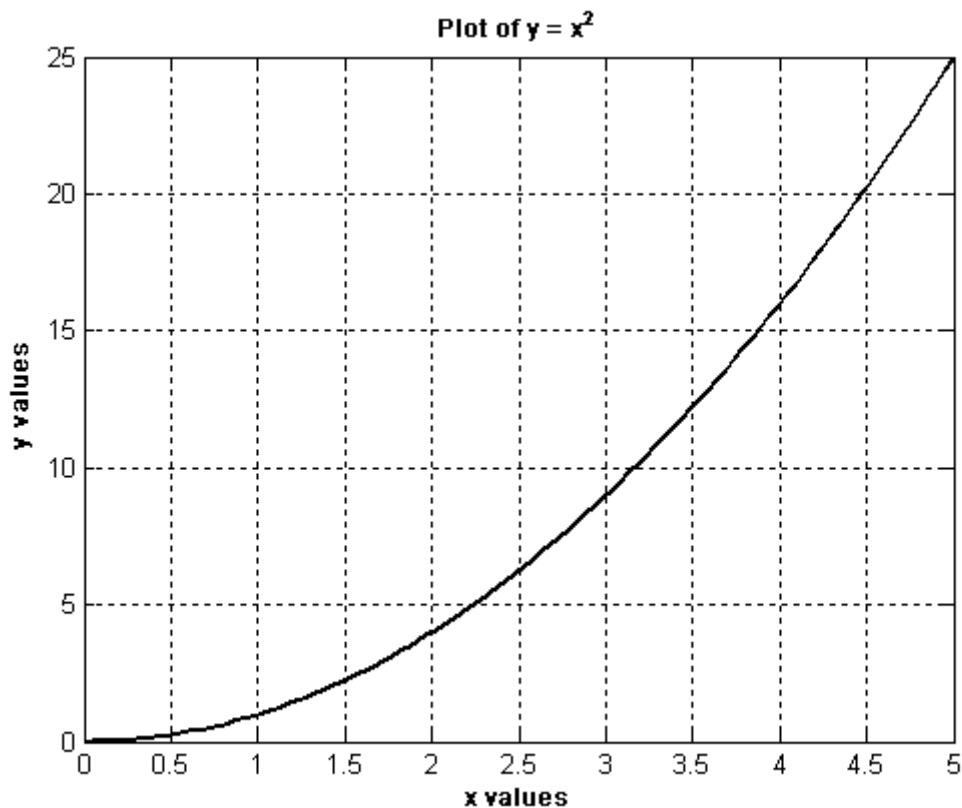


- **Adding titles and labels to a graph**

You can add labels to the x and y axes and also put a title on the graph as shown below. You do not have to add labels or titles to graphs but if you are generating a graph to paste in a Word document then labels and/or title are very useful.

The following graph was generated by the commands

```
>> plot(x, y); grid
>> xlabel(' x values' )
>> ylabel(' y values' )
>> title( ' Plot of y = x^2 ' )
```



## 9.1 Putting several graphs of the one plot

Suppose that I want to compare the graphs of  $y = x$ ,  $y = x^2$  and  $y = x^3$  over the interval from -2 to +2.

We need to set up the x array (fill this in yourself)

```
>> .....
```

Next we need to calculate the coordinates on these three graphs

```
>> y1 = x;
```

```
>> y2 = x.^2;
```

```
>> y3 = .....
```

An extended version of plot can take more pairs of x, y arrays.  
Each x, y pair is added to the graph

```
>> plot(x, y1, x, y2, x, y3); grid; shg [try this yourself]
```

Add in the grid and the shg command as well

Matlab uses a default colour scheme that usually plots them using colours in the order blue first, then green and then red.

You can control the colour scheme yourself – e.g. the following command makes the graphs blue, black and green

```
>> plot(x, y1, 'b', x, y2, 'k', x, y3, 'g'); grid; shg [try this yourself]
```

You can change the colours or the style of the three plots yourself by adding a third 'style parameter to any (or all) of the x, y pairs.

This is typical of Matlab – to make plots you have to give the x & y data and then there are optional things that you do the plot.

## 9.2 You can change the colour and style of the different lines in the graph

Style refers to the way that the curve appears – a solid line, a dotted line, a line made up of circles etc and/or to the colour of the curve.

The complete list of style options is

Various line types, plot symbols and colors may be obtained with `PLOT(X,Y,S)` where `S` is a 1, 2 or 3 character string made from the following characters:

The left column gives the colour options, the right column the options for the way the line looks.

y	yellow	.	point
m	magenta	o	circle
c	cyan	x	x-mark
r	red	+	plus
g	green	-	solid
b	blue	*	star
w	white	:	dotted
k	black	-.	dashdot
		--	dashed

### Examples

- >> `plot(x, y, 'b')` plots the graph in blue as a solid line (default)
- >> `plot(x, y, 'bo')` plots graph as blue circles
- >> `plot(x, y, 'g:')` graph appears as a dotted green line.
- >> `plot(x, y1, 'k', x, y2, 'b')` first curve is a black line; second is blue dots

### Warning

When you come to plot a graph on a black & white printer the lighter colours are very faint. A good rule with graphs to use different styles sparingly and just enough to clearly separate different curves.

## Exercise

Go back to your plot of  $y = x$ ,  $y = x^2$  and  $y = x^3$ .

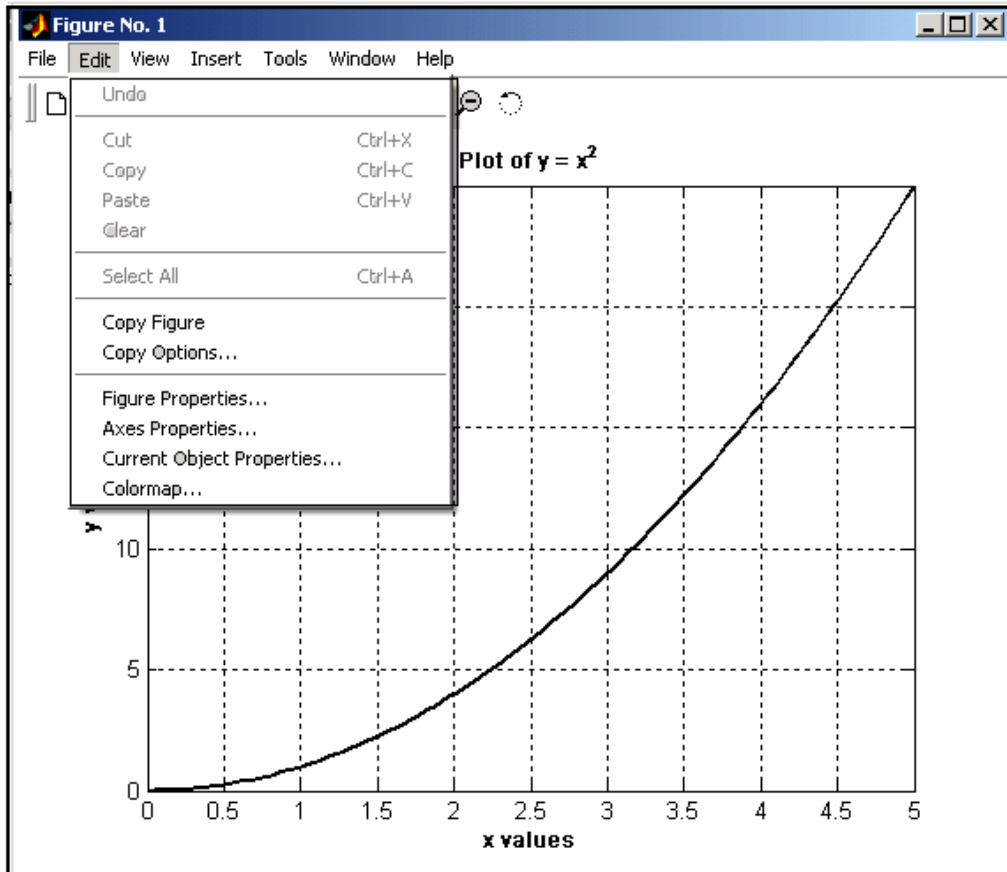
```
>> plot(x, y1,x, y2, x, y3)
```

Modify this command and add others so that you get a graph of these three curves with

- $y = x$  a dotted black line
- $y = x^2$  a blue dot dash line
- $y = x^3$  is a green dotted line
- add a grid to this graph
- add a title just below the x axis that says

**‘Comparing  $y = x$  (solid),  $y = x^2$  (-.) and  $y = x^3$  (..) graphs’**

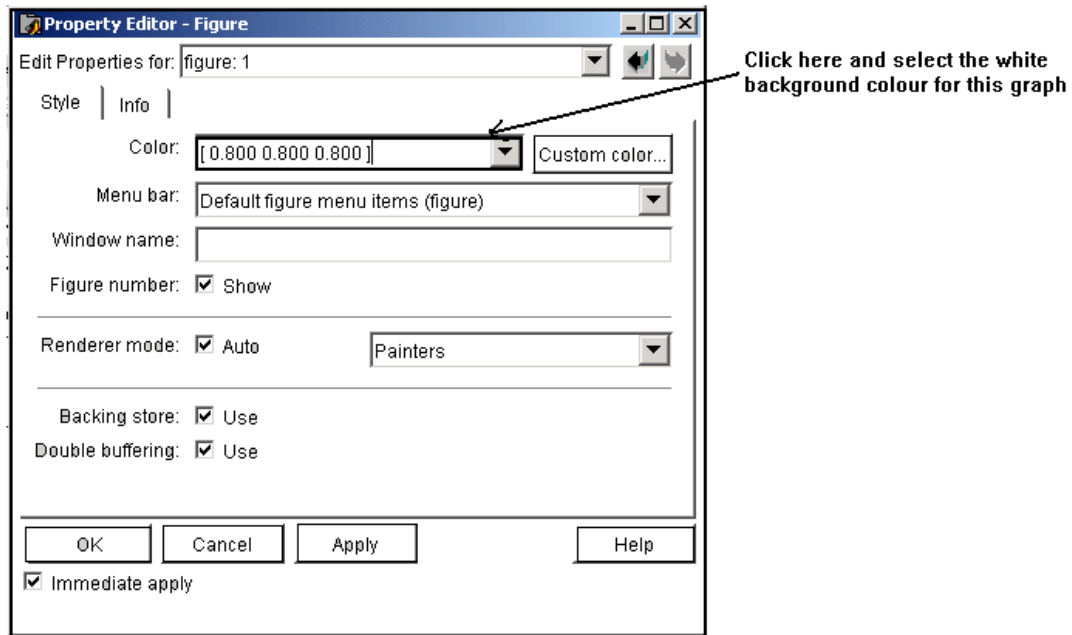
We can also adjust the size of the graph, the axes and put labels on the axes as well. You can also control various features of a graph by using the Edit menu and selecting the Figure Properties option - you can try experimenting yourself.



### 9.3 Printing Matlab graphs:

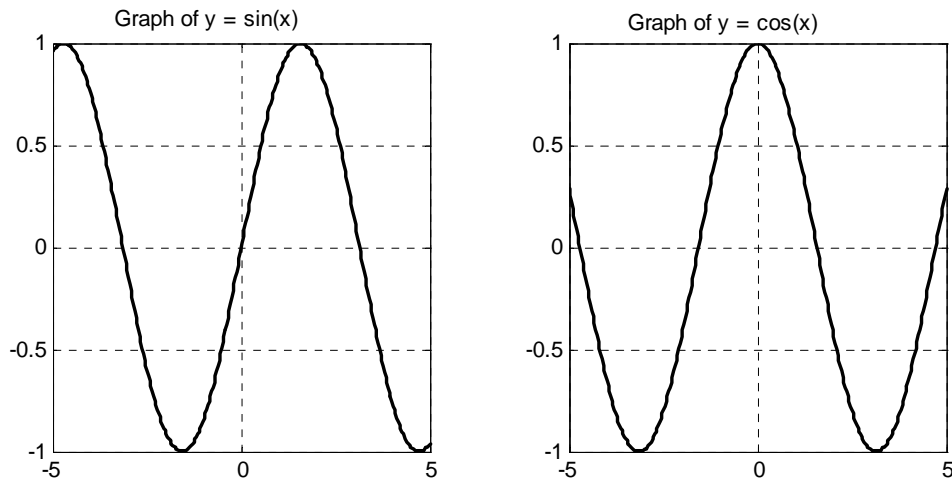
Matlab displays graphs with a graph background but if you want to put the graph in a report it is better to change the background to white. The easiest way to do this is use the Figure Editor.

Go to the last figure that you plotted and open the Figure Editor.



## 9.4 Putting several graphs side by side.

Sometimes we need to put two graphs close together – like the ones below



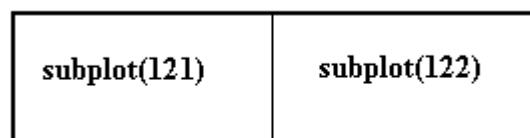
The Matlab code to do this is

```
>> x = linspace(-5, 5, 500); y1 = sin(x); y2 = cos(x);  
>> subplot(1, 2, 1); plot(x, y1); grid; shg  
>> title(' The graph of y = sin(x) ');  
>> subplot(1, 2, 2); plot(x, y2); grid; shg  
>> title(' The graph of y = cos(x) ');
```

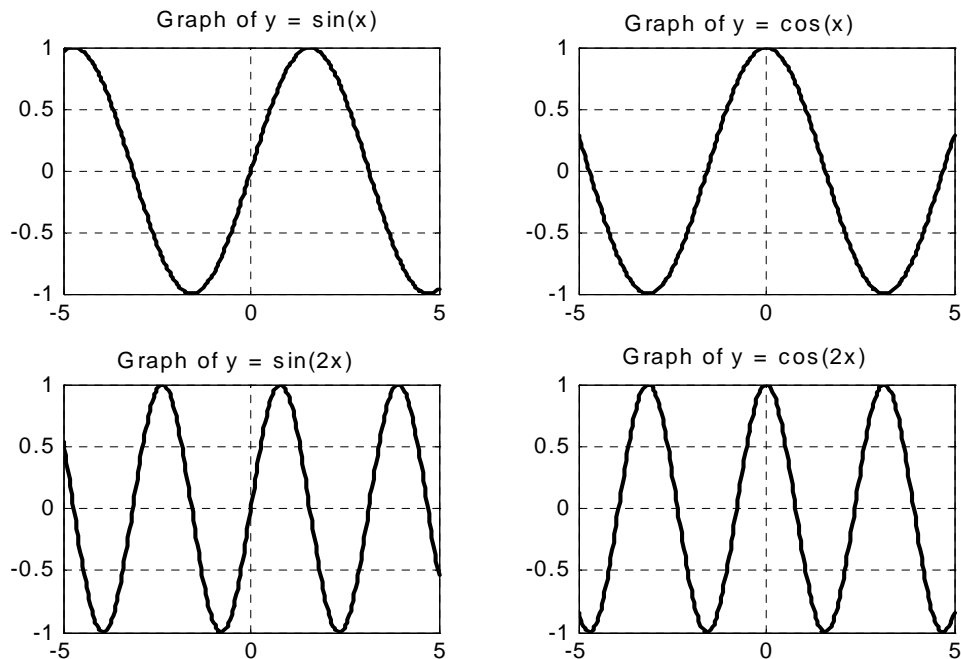
The subplot command is used to put the graph in the correct place – the number 1, 2, 1 is interpreted as:

1 = number of rows in plot  
2 = number of columns in plot (there are 2 graphs across the page)

The last number tells you where the plot goes as shown below.



- Another way that is sometimes used is to put 4 graphs in a 2 x 2 pattern.



The code for this is

```
>> x = linspace(-5, 5, 500);
>> y1 = sin(x); y2 = cos(x); y3 = sin(2*x); y4 = cos(2*x);
>> subplot( 2, 2, 1 ); plot( x, y1 ); grid ; shg
>> subplot( 2, 2, 2 ); plot( x, y2 ); grid ; shg
>> subplot( 2, 2, 3 ); plot( x, y3 ); grid ; shg
>> subplot( 2, 2, 4 ); plot( x, y4 ); grid ; shg
```

The layout of the plots is

<b>subplot(221)</b>	<b>subplot(222)</b>
<b>subplot(223)</b>	<b>subplot(224)</b>

## 9.5 Using Matlab graphics to investigate a function

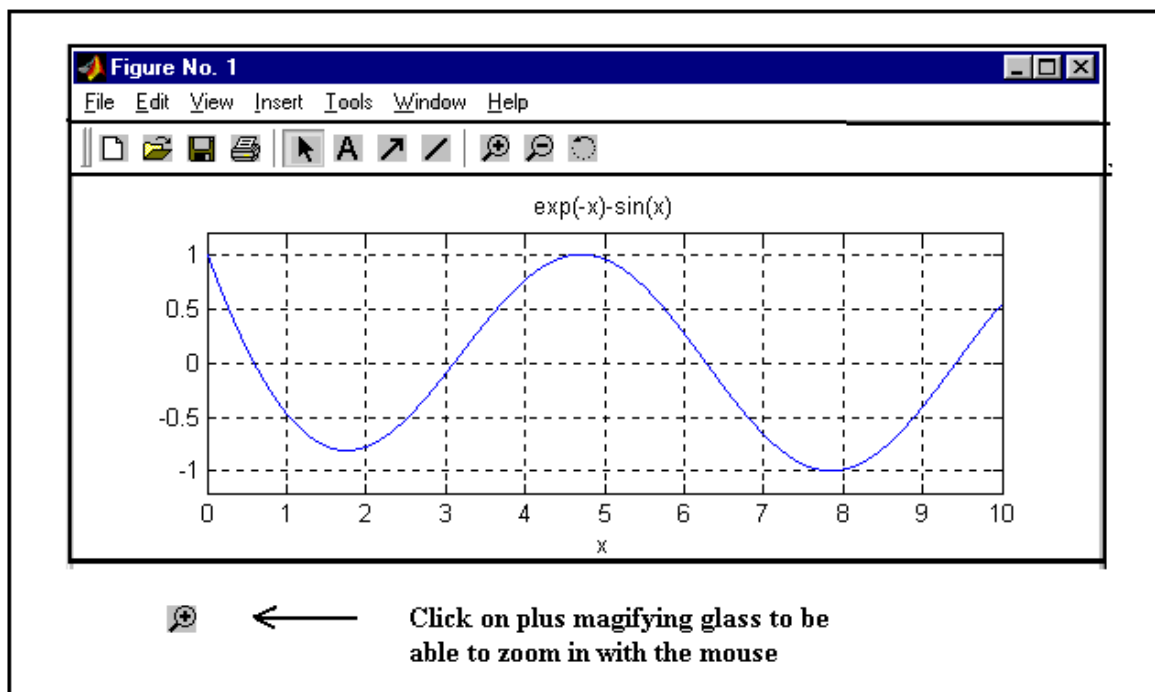
This section shows how Matlab graphs can be used to investigate an example of solving a nonlinear equation. This is not the only way to do this – Matlab in fact has a fast numerical method that will do this without needing to graph the problem.

**Problem** Investigate the solutions of the equation  $y = e^{-x} - \sin(x) = 0$

You can plot the graph of this function between 0 and 10 using

```
>> x = linspace( 0, 10, 1000 );
```

```
>> y = exp(-x) - sin(x); plot( x, y ); grid; shg
```



### HOW TO MAGNIFY PARTS OF THE GRAPH

- Click on the magnifying glass symbol at the top of the graph to let you zoom in to parts of the graph.
- Choose the part of the graph that you want to magnify by holding the mouse button down and draw a small box around the 'zoom region'. Release the mouse button to see the zoomed region.
- Double click zooms back out to the original graph.

- **Approximately solving the nonlinear equation  $e^{-x} - \sin(x) = 0$  using this graph.**

The places where  $e^{-x} - \sin(x) = 0$  are where the graph of  $y = e^{-x} - \sin(x)$  crosses the x axis.

You can zoom in on the points where y is small and get quite accurate estimates of the solutions. It is possible to zoom in so that you can find a solution to 1 or 2 decimal places - try this method out to

- find the solution closest to zero .....
- find the 2<sup>nd</sup> solution ? .....
- find the 3<sup>rd</sup> solution ? .....

## 10 Matlab is designed to work easily with vectors

With ordinary calculators you can usually only calculate the values of functions like sin, cos etc at individual points. In Matlab you can get the values of sin at a whole list of values in one step.

For example

```
>> sin(1) ..... [gets one value]
>> x = [ 3.0, 1.0, 7.5 ]; y = sin(x) [gets all 3 values]
```

This gives the array [ 0.1411 , 0.8415 , 0.9380]

Here  $0.8415 = \sin(1)$ ,  $0.1411 = \sin(3)$  and  $0.9380 = \sin(7.5)$  - with all values taken in radians.

### How does Matlab do this ?

In most other languages (e.g. Pascal, C, Java) if you wanted to do this calculation you would need to set up a loop to process the three entries individually. In Matlab the loop is done for us because Matlab has been programmed so that its sine function can process individual numbers and arrays.

When you type the Matlab command `sin(x)`, Matlab first finds the size of the  $x$  array. If  $x$  is a single number it calculates the sine of that number. If  $x$  is an array or a matrix, Matlab will process each number in the array.

Because Matlab has this sophisticated 'interpreter' Matlab programs are typically much shorter than programs in other languages.

### Functions that can process a whole array

- Mathematical functions like sin, cos, log, exp, sqrt etc all accept vector input and give vector output.
- There are some other functions that can process a whole array (or vector)
  - sum – this adds together all the entries in an array
  - max - finds the largest (maximum) number in an array
  - min – finds the smallest (minimum) number in an array
  - length – gives us the number of entries in the array

**Example 1** Experiment with `sum(x)`, `max(x)`, `min(x)` and `length(x)` when `x = [ 3, 1, 7.5]`.

```
>> sum(x) .....
>> max(x) .....
>> min(x) .....
>> length(x) .....
```

**Exercise 2** Finding  $1^3 + 2^3 + 3^3 + \dots + 11^3$

- Construct  $b = [1^3 \ 2^3 \ 3^3 \ \dots \ 10^3 \ 11^3]$ 

```
>> a = 1:11;
>> b = a.^3;
```
- Now add up the entries in `b` - this uses the **sum** function

```
>> sum(b) .....
```

**Exercise 3** - Calculate  $s = \sin(0) + \sin(\frac{\pi}{6}) + \sin(\frac{2\pi}{6}) + \dots + \sin(\frac{11\pi}{6})$

This can also be written as  $s = \sum_{k=0}^{11} \sin(\frac{\pi}{6}k)$

- Find the shortest way to set up the vector  $k = [0, 1, 2, \dots, 11]$ 

```
>> .....
```
- get  $b = [0, \frac{\pi}{6}, \frac{2\pi}{6}, \frac{3\pi}{6}, \frac{4\pi}{6}, \dots, \frac{11\pi}{6}]$  from `k`

```
>> b = .....
```
- Find  $c = [\sin(0), \sin(\frac{\pi}{6}), \sin(\frac{2\pi}{6}), \dots, \sin(\frac{11\pi}{6})]$ 

```
>> c = .....
```
- Finally get the answer for the sum

```
>> s = .....
```

- The exact answer for this sum is zero but you might not get this because of ‘round off’.

Write the answer that you get in scientific form ( like  $2.45 \times 10^{-8}$  )

.....

**Example 4** Use the same idea to calculate the following sum

$$c = \cos(0) + \cos\left(\frac{\pi}{4}\right) + \cos\left(\frac{2\pi}{4}\right) + \cos\left(\frac{3\pi}{4}\right) + \cos\left(\frac{4\pi}{4}\right) + \cos\left(\frac{5\pi}{4}\right)$$

## 11 Symbolic algebra and calculus exercises within Matlab

Matlab was originally only a numerical program but it has linked up with the symbolic algebra package Maple to include some parts of Maple.

Symbolic algebra packages do algebra and calculus symbolically.

They can handle mathematical formulas like  $(x + y)^2$  and expand or simplify them because the rules of algebra are programmed into the package. They can also do basic calculus because the rules of differentiation and some techniques for integration are also programmed into them.

Symbolic algebra packages are helpful for many purposes but they do not always give us the simplest answer. Maple and Mathematica are examples of more powerful symbolic algebra packages.

### 11.1 Basic symbolic algebra

- **Declaring symbolic variables**

To do symbolic algebra in Matlab we need to tell Matlab that we are dealing with symbolic variables

**Try out the following commands**

>> **syms x y**      ← **x and y are defined as symbolic variables**

>> **whos**            ← check to see what variables are defined

>> **f = x^2 - 5\*x - 3**      ← define a new expression  $f = x^2 - 5x - 3$

>> **pretty(f)**        ← tries to print it in a more conventional form

**Note:** Symbolic variables like x and f are quite different from arrays. The dot notation is never used with symbolic variables.

- **Expand the expression**  $g = (x^2 - 5x - 3)^3$

```
>> g = expand(f ^3)
```

.....

**Will the Matlab simplify function convert g into its original form ?**

```
>> simplify(g)
```

.....

The simplify function sometimes works and sometimes doesn't –

- **Simplifying trig expressions**

```
>> f = cos(2*x) + 1
```

```
>> fe = expand(f)
```

.....

```
>> fs = simplify(fe)
```

.....

### Exercise

Can you get  $\sin(x + y) \cdot \cos(x - y)$  into a simpler form ?

## 11.2 Basic symbolic calculus with Matlab

This shows you how to get the derivative and integral of a function  $f(x)$

- Getting the first derivative of  $f = x^2 - 5*x - 3$

```
>> df = diff( f, x)
```

You should be able to check that this is correct and to find the type of df

- **Integration**

Matlab can also do symbolic integration for many functions. Let's do a simple example that you know how to do without Matlab

What is the indefinite integral of  $f = x^2 + \frac{1}{x}$  ?

.....

Check your answer with Matlab

```
>> intf = int( f, x )
```

If need be use **pretty(intf)** to put in a nicer form

You should find that your answers agree.

- **Definite integrals**

Mostly we need to work out definite integrals like  $\int_2^5 (x^2 + \frac{1}{x}).dx$

We use an extended version of **int** that allows us to include the limits on the integral.

```
>> int( f, x, 2, 5)
```

- In this case use **whos** to see what type of object this calculation produced – was it a real number or a symbolic object.

.....

### 11.3 Solving simple differential equations symbolically

Matlab can also solve some differential equations symbolically. The command is called `dsolve`. It will not work for every differential equation but is worth trying.

**Example** Find the general solution of  $\frac{dy}{dt} = y$

```
>> soln = dsolve('Dy=y', 't') ← 'Dy = y' sets up the DE  
                                't' is the independent variable
```

```
soln = C1*exp(t) ← soln = C1.et
```

**Example** Find the general solution of  $\frac{dy}{dt} = y$  which starts with  $y = 2$  at  $t = 0$

```
>> soln = dsolve('Dy=y', 'y(0)=2', 't') ← 'y(0)=2' sets up the  
                                           initial condition
```

```
soln = 2*exp(t) ← soln = 2.et
```

You can use the command `>> help dsolve` to learn more about `dsolve`.

#### Warning:

Symbolic packages are limited. Sometimes they will not be able to find any solution and other times they will give you an answer that will involve unusual or unfamiliar functions.

## 12 Some easy graphics commands in 2D and 3D

The easy plot commands all start with ez.

### 12.1 The 2D ezplot command

To plot the function  $y = \sin(x)$  between  $x = 0$  and  $x = 3$ .

```
>> ezplot('sin(x)', [0, 3])      (define function and the x range of the plot)
```

**Advantage:** Easy to use

#### Disadvantages

- (i) it does not give you any control over how many  $x, y$  points are plotted in the graph – it always uses 300  $x$  points. Usually this is enough but sometimes we definitely need more points
- (ii) ezplot only handles one function at a time.

- **ezplot can also graph 'implicitly defined functions'**

These are graphs that are defined by a formula  $f(x, y) = 0$ .

A simple example is the circle with radius 1.

It has the implicit formula  $x^2 + y^2 = 1$

This is better written as  $x^2 + y^2 - 1 = 0$

The command that plots this is

```
>> ezplot('x^2 + y^2 - 1', [-3, 3])
```

Implicit functions can give very interesting shapes.

Here are other examples

- $x^4 - x^2 * y^2 + y^4 - 1 = 0$
- $x^4 - 6 * x^2 * y^2 + y^4 - 1 = 0$

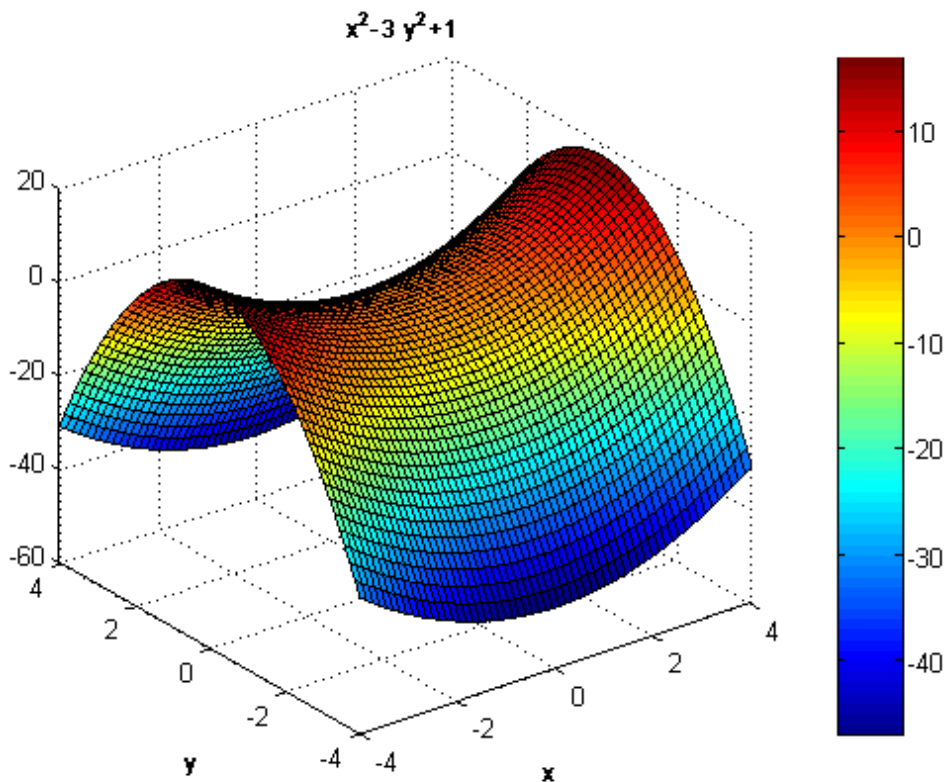
## 12.2 3D graphics using ezsurf

The most common type of 3D graphics is to plot a surface.

Surfaces are described by an equation  $z = f(x, y)$ . At each point in the  $xy$  plane we calculate  $z$  from the formula  $z = f(x, y)$  and this gives the height of the surface. Here are some examples of surfaces

**Example 1:**  $z = x^2 - 3y^2 + 1$

**Plot the shape of the surface over the range of x values from -4 to +4 and y values from -4 to +4**



Here are commands that will plot this surface

```
>> ezsurf('x^2-3*y^2+1', [-4, 4, -4, 4]); shg
>> colorbar
```

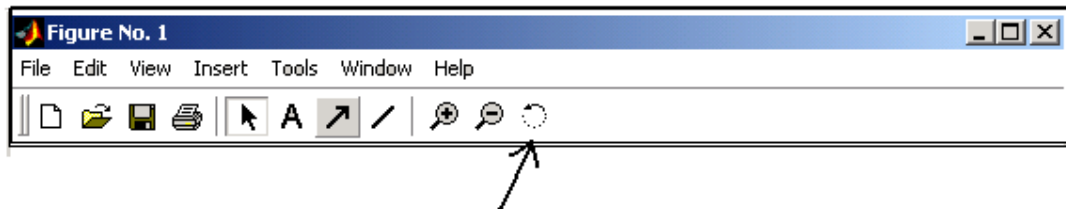
The four numbers in the array are

[minimum x value, maximum x value, minimum y value, maximum y value]

On the screen you will see a colour coded graph – the colour is related to the height and it is very useful to add a **colorbar** on the side.

## 12.3 Rotating a 3D plot

It is very helpful to be able to twist a 3D surface to see what it looks like from different directions.



**This is button to click to allow 3D rotations**

Once you have clicked the rotation button you can 'grab' hold of the plot with the mouse and rotate the picture to get the most informative view – best to rotate slowly.

## 12.4 Other types of plots in 2D and 3D

There are many other types of 2D and 3D plots in Matlab. To see some of them you can use the demonstration programs

```
>> graf2d           [shows you 2D plots]
>> graf2d2        [shows 3D plots]
>> demos           [the full list of Matlab demos]
```

### 13 Defining your own inline functions.

So we have used built-in functions like sin, cos etc.

However very often we need to define our own functions. There are several ways to this. Inline functions are useful for functions that have a simple one line formula

Three functions that occur in science and engineering are

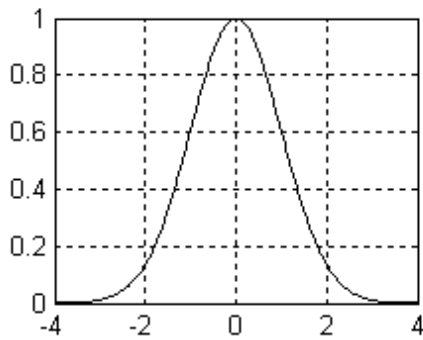
The normal distribution  $y = e^{-x^2/2}$

The chirp pulse  $y = \sin(t^2)$

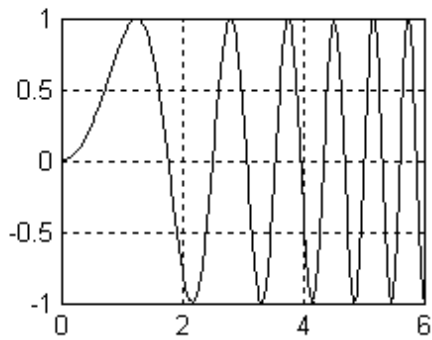
The sine function  $y = \sin(t)$

The sine function oscillates at a steady rate and so it has a constant frequency. The chirp pulse oscillates more and more quickly and its frequency increases with time.

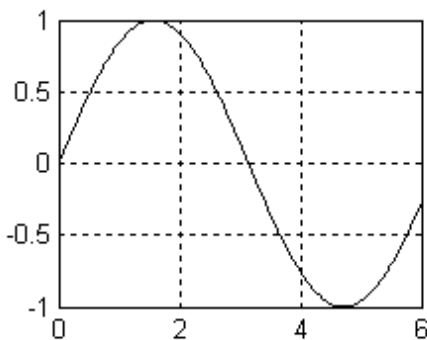
The normal curve is widely used in statistics and probability.



**Plot of normal curve**



**Plot of chirp signal**



**Plot of sin curve**

- **Defining these functions using inline definitions**

Defining the formula for the normal curve

```
>> f = inline( ' exp(-x.^2/2) ', 'x');
```

Defining the formula for the chirp signal

```
>> g = inline('sin(t.^2)', 't');
```

An alternative definition of sin function

```
>> h = inline( 'sin(x)', 'x'); [not really needed]
```

- **Calculating the value of an inline function**

To get the value of the normal curve at  $x = 2$

```
>> f(2) [should get 0.1353]
```

- **Using an inline definition to plot a graph**

To plot a graph of the normal curve between -5 and +5

```
>> x = linspace(-5,5,400);  
>> y = f(x); [this works because f uses dot notation]  
>> plot(x,y); grid; shg
```

Inline definitions have lots of other uses but once you leave Matlab your inline definitions vanish. If you want a more permanent definition you need to create an 'm file'

## 14 Script files functions

So far we have typed individual commands into Matlab. This can get tiring and if we are going to be using the same set of commands several times it is useful to store the commands in a single file.

Matlab will let you store commands in a script files. Then you can easily change the commands without having to retype them all.

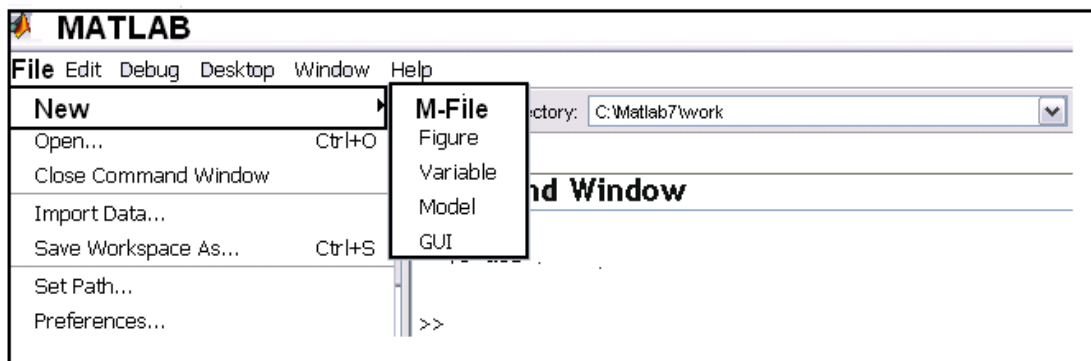
### 14.1 Creating and storing a script file to plot a function

- First let's open up the editor

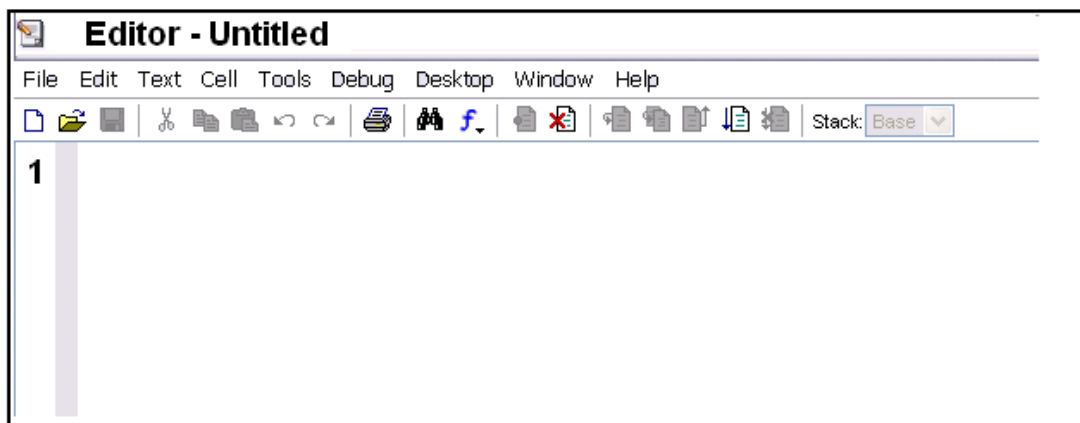
Go to the **File** menu (top left)

Click on this and choose **New** and then **M- file**.

Then release the mouse button



- The editor window now opens up - it looks like



Now we just type Matlab commands in this window.

The line numbers on the left are used by error messages to tell you where errors occur.

- **Creating a script file**

Type the following commands into the Editor window

```
% first define the function to plot  
f = inline( ' sin(x.^2)', ' x' );  
  
% next set up x and y arrays  
x = linspace( -3, 3, 500);  
y = f(x);  
  
% plot the graph and add label  
plot(x, y, 'b'); grid; shg  
xlabel('Plot of sin(x^2) between x = - 3 and + 3')
```

The lines that start with % are comment lines.  
They are added to explain what the code is doing  
These lines do not start with >>

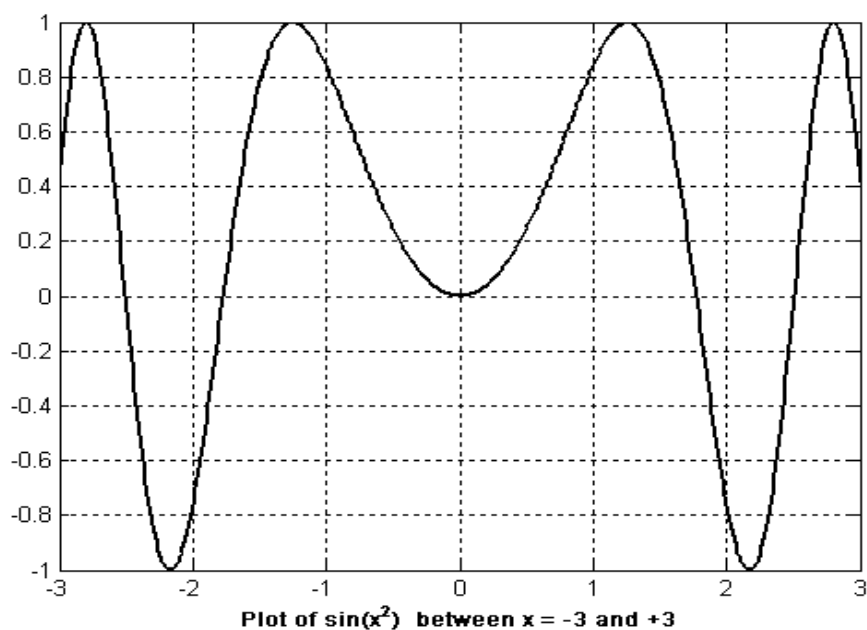
- **Saving this file**

Go to the **Edit** menu and use the **Save As** option  
Save this file under the name **test.m**

- **Go back to the Command Window**

Then run the script file get a graph like the

```
>> test
```



- **Seeing what happens if the script file contains an error**

Go back to the editor and change the 2<sup>nd</sup> line to

```
x = linspace(-3, 3, 500);
```

Save the changed file and then run it from the command window.

You ought to get the following error messages

```
??? Undefined function or variable 'linspace'.
```

```
Error in ==> C:\MATLAB6p5p2\work\test.m
```

```
On line 5 ==> x = linspace( -3, 3, 500);
```

The first line tells you that Matlab does know a command called **linspace**

The second line tells you the file where the error occurs.

The last line tells you that the error occurs on line 5.

## 14.2 Changing this script file

One advantage of this script file is that we can easily edit it to change any of the following

- the definition of the function being plotted
- the limits on the x axis
- the title on the graph

Switch back to the editor window and change the script file so that it will plot the

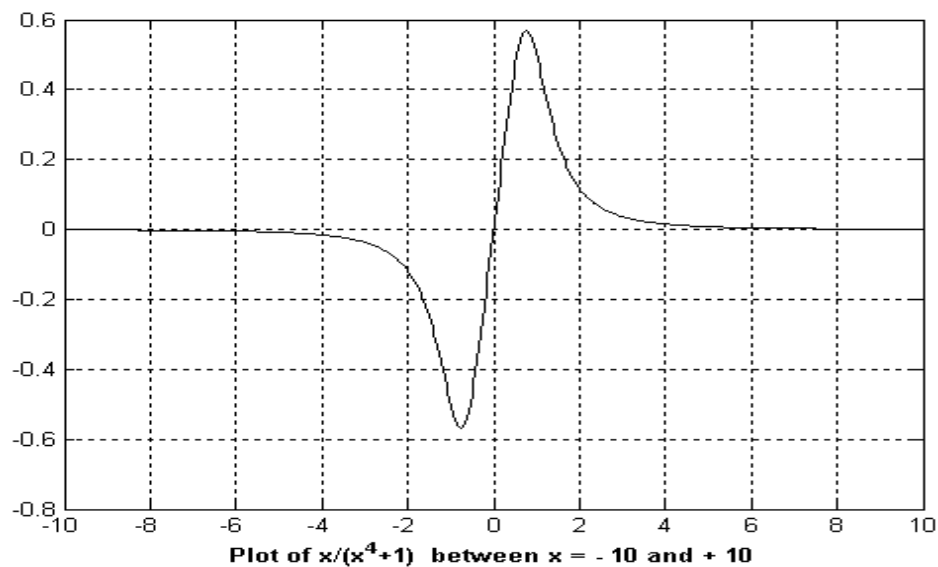
graph of  $y = \frac{x}{x^4 + 1}$  between -10 and 10

You will need to change the definition of f to **f = (x)./(x.^4 + 1)**

Also change the x array

Be careful to save your new script file before returning to the command window  
You can use the **Save** option in the **Edit** menu.

If you run test again you should get a graph



## 15 Using some built-in functions

You have seen some of the simpler built-in functions

- Basic Maths functions like sin, cos, exp, ..
- Functions that work on arrays like sum, max, min, ....
- Functions for plotting graphs like plot, ezplot, ...

There are many hundreds of other built-in functions that do more sophisticated things.

One example of a useful built-in function is **quad**.

This is used to estimate the value of an integral numerically.

**Example:** Estimate the value of the integral  $\int_1^5 \frac{x^2}{e^x + 1} dx$

It is not obvious how to do this by calculus so we estimate the integral numerically

There are two steps

- 1 Define the function  $f(x) = \frac{x^2}{e^x + 1}$  as an inline function using the dot Notation

```
>> f = inline( '(x.^2)./(exp(x)+1)', 'x' )
```

2. Now use the quad function to work out the integral from  $x = 1$  to  $x = 5$

```
>> res = quad( f, 1, 5 )
```

This gives the answer as 1.4471

### Learning more about what quad does.

The quad function has optional extras – for example, you can set the accuracy that you want the answer calculated to (the default accuracy is  $10^{-6}$ ). To learn more about the quad function you can use

```
>> help quad
```

## Appendix 1: Revising matrix multiplication.

If  $A = \begin{bmatrix} 2 & 3 \\ 1 & 4 \end{bmatrix}$  and  $B = \begin{bmatrix} 4 & 3 \\ 2 & 1 \end{bmatrix}$  then the product

$$A * B = \begin{bmatrix} 2 & 3 \\ 1 & 4 \end{bmatrix} * \begin{bmatrix} 4 & 3 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 2*4+3*2 & 2*3+3*1 \\ 1*4+4*2 & 1*3+4*1 \end{bmatrix}$$

Here each entry in AB is made by combining one row from A with one column from B.

### For example:

The (1,1) entry of A comes from row 1 of A = [2 3] - and column 1 of B =  $\begin{bmatrix} 4 \\ 2 \end{bmatrix}$

This is combined as  $2 * 4 + 3*2 = 8 + 6 = 14$ .

### Carry through the rest of the calculation to check that

$$A * B = \begin{bmatrix} 2 & 3 \\ 1 & 4 \end{bmatrix} * \begin{bmatrix} 4 & 3 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 14 & 9 \\ 12 & 7 \end{bmatrix}$$

If you still need some practice try to calculate

$$A^2 = A * A = \begin{bmatrix} 2 & 3 \\ 1 & 4 \end{bmatrix} * \begin{bmatrix} 2 & 3 \\ 1 & 4 \end{bmatrix}$$

### Note:

Matrices A and B can only be multiplied if the number of columns of A = the number of rows of B

If A is a 2 x 3 matrix and B is a 3 x 3 matrix which of the following products are defined

- (i)  $A * B$
- (ii)  $A^2 = A * A$
- (iii)  $B^2$
- (iv)  $B * A$