MATLAB TUTORIAL

The idea behind this tutorial is that you view it in one window or have a printout of it in front of you while running Matlab in another window. You should be able to re-do all of the plots and calculations in the tutorial by copying the text from the tutorial into Matlab or an m-file. `I will use this font to denote text that has to be typed in Matlab or the text that Matlab returns.`

Index

1. Getting Started

To start Matlab from Windows:

1) Click on Start
2) Select Programs and then Matlab
3) Click on the Matlab icon and this will bring up the window called the Command Window. This is the main space you will be using, and it will look as follows:

```
To get started, type one of these: helpwin, helpdesk, or demo. For
product information, type tour or visit www.mathworks.com.

»
```

This window allows a user to enter simple commands. To perform a simple computations type a command and next press the **Enter** or **Return** key. For instance

```
» s = 1 + 2

s =

     3
```

Note that the results of these computations are saved in variables whose names are chosen by the user. If they will be needed during your current MATLAB session, you can obtain their values typing their names and pressing the **Enter** or **Return** key. For instance, if you type again:

```
» s

s =

     3
```

Some useful format commands for the display are:

- `format short` 3.1416
- `format long` 3.14159265358979
- `format compact` suppresses extra line feeds

To close MATLAB type exit in the **Command Window** and next press **Enter** or **Return** key. A second way to close your current MATLAB session is to select **File** in the MATLAB's toolbar and next click on **Exit MATLAB** option. All unsaved information residing in the MATLAB **Workspace** will be lost. You can also exit by typing:

```
» quit
```

2. Matlab Help

One of the nice features of MATLAB is its help system.. A good place to start is with the command:

» `help help` that explains how the help systems works. In addition, the command

» `help` produces a list of topics for which help is available. The list is printed at the end of the tutorial. Looking at it we find, for example, the entry:

```
matlab\elfun          -  Elementary math functions.
```

» `help elfun` produces then a list of the math functions available. Alternatively, the `helpwin` command opens a new window on the screen. To find an information you need double click on the name of the subdirectory and next double click on a function to see the help text for that function. In general, to get more information on any given command, type:

```
» help command
```

3. Numbers, Arithmetic Operations and Special Characters

There are three kinds of numbers used in MATLAB:

- integers
- real numbers
- complex numbers

In addition to these, MATLAB has three variables representing non-numbers:

```
-Inf  Inf   NaN
```

The **–Inf** and **Inf** are the negative and positive infinity respectively. Infinity is generated by overflow or by the operation of dividing by zero. The **NaN** stands for Not-A-Number and it is obtained as a result of the mathematically undefined operations such as 0.0/0.0.

The list of basic arithmetic operations in MATLAB includes six operations:

+ : addition

- : substraction

* : multiplication

/ : right division

\ : left division

^ : power

Further, there is a menu of special characters that have specific uses. The main ones are:

- Equal	==
- Not equal	~=
- Less than	<
- Greater than	>
- Less than or equal	<=
- Greater than or equal	>=
- Logical AND	&
- Logical OR	|
- Logical NOT	~
- Colon	:
- Parentheses and subscripting	( )
- Brackets	[ ]
- Decimal point	.
- Continuation	...

- Separator ,

- Semicolon ; (suppresses the output of a calculation)

- Comment %

- Assignment =

- Quote ' *statement* '

- Transpose '

4. Vectors, Matrices and Three Dimensional Arrays

All inputs in in MATLAB are taken to be arrays. In this way, a scalar is a *1x1* array, a row vector is a *1xn* array, a column vector is an *nx1* array and a matrix is an *nxm* array. MATLAB can also handle three dimensional arrays (that you can think of as matrices stacked one on top of the other). To define arrays, we always use the brackets.

Vectors

To create the row vector a type:

```
» a = [1 2 3 4 5 6 9 8 7]

a =

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

Note that separating the elements by a space you create a row vector. If you want to create a column vector, separate the elements by semicolons:

```
» b = [1; 2; 3; 4; 5; 6]

b =

     1
     2
     3
     4
     5
     6
```

Another way to create column vectors is by using the transpose, which is denoted with a ' after the bracket:

```
» b = [1  2   3  4  5  6]'
```

The command length returns the number of components of a vector:

```
» length(a)


ans =

     9
```

If you ant to type a scalar just write:

```
» c = 2

c =
     2
```

Suppose you would like to add 2 to each of the elements in vector a. The equation for that looks like:

```
» e = a + 2

e =

          3      4      5      6      7      8     11     10      9
```

Now suppose, you would like to add two vectors together. If the two vectors are the same length, it is easy. Simply add the two:

```
» f = a + e

f =

     4      6      8     10     12     14     20     18     16
```

Subtraction of vectors of the same length works exactly the same way. A useful command is ``whos", which displays the names of all defined variables and their types:

```
>> whos

Name      Size          Bytes  Class

  a         1x9             72  double array
  c         1x1              8  double array
  e         1x9             72  double array
  f         1x9             72  double array
  t         1x11            88  double array

Grand total is 39 elements using 312 bytes
```

Note that each of these variables is an vector. The "shape" of the vector determines its exact type. The scalar c is a 1 x 1 array, the vector a is a 1 x 9 array.

Matrices

Entering matrices into Matlab is the same as entering a vector, except each row of elements is separated by a semicolon **(;)** or a return:

```
» B = [1 2 3 4; 5 6 7 8; 9 10 11 12]

     B =
          1     2     3      4
          5     6     7      8
          9    10    11     12
```

Alternatively, you can enter matrices as follows:

```
» B = [ 1   2   3   4
        5   6   7   8
        9  10  11  12]

    B =
          1     2     3      4
          5     6     7      8
          9    10    11     12
```

Like with vectors, we have an in-built function that gives the dimensions of a matrix:

```
 » size(B)

ans =

     3   4
```

Creating special matrices

- zeros(m,n) creates an **m x n** matrix of zeros
- ones(m,n) creates an **m x n** matrix of ones
- eye(n) creates the **n x n** identity matrix
- diag(v) (assuming v is an n-vector) creates an **n x n** diagonal matrix with v on the diagonal.

```
» zeros(2,1)

ans =

     0
     0
```

Matrix manipulation and operations

Matrices in Matlab can be manipulated in many ways. For one, you can find the transpose of a matrix using the apostrophe key:

```
» C = B'

      C =
                1    5     9
                2    6    10
                3    7    11
                4    8    12
```

Now you can multiply the two matrices B and C together. Remember that order matters when multiplying matrices.

```
» D = B * C
      D =       30     70    110
                70    174    278
               110    278    446

» D  = C * B

       D =
           107   122   137   152
           122   140   158   176
           137   158   179   200
           152   176   200   224
```

Another option for matrix manipulation is that you can multiply the corresponding elements of two matrices using the .* operator (the matrices must be the same size to do this).

```
» E = [1 2;3 4]

       E =

            1     2
            3     4

» F = [2 3;4 5]

       F =

            2     3
            4     5

» G = E .* F

      G =

           2     6
          12    20
```

If you have a square matrix, like E, you can also multiply it by itself as many times as you like by raising it to a given power.

```
» E^3
      ans =
            37    54
            81   118
```

If wanted to cube each element in the matrix, just use the element-by-element cubing.

```
» E.^3
           ans =
                1      8
               27     64
```

In general:

`a .* b`	multiplies each element of a by the respective element of b
`a ./ b`	divides each element of a by the respective element of b
`a .\ b`	divides each element of b by the respective element of a
`a .^ b`	raise each element of a by the respective b element

You can also find the inverse of a matrix:

```
» X = inv(E)

           X =
              -2.0000    1.0000
               1.5000  -0.5000
```

or its eigenvalues and eigenvectors:

```
» eig(E)

      ans =

         -0.3723
          5.3723

» [vec,va]=eig(E)

      vec =

         -0.8246   -0.4160
          0.5658   -0.9094


      va =

         -0.3723         0
               0    5.3723
```

If A and B are matrices, then Matlab can compute A+B and A-B *when these operations are defined*. For example, consider the following commands:

```
>> A = [1 2 3;4 5 6;7 8 9]
A =

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

```
>> B = [1 1 1;2 2 2;3 3 3]
B =

     1     1     1
     2     2     2
     3     3     3

>> C = [1 2;3 4;5 6]
C =

     1     2
     3     4
     5     6

>> A+B

ans =
     2     3     4
     6     7     8
    10    11    12

>> A+C

??? Error using ==> +
Matrix dimensions must agree.
```

The Colon Operator

The colon operator **:** can be used to perform special and useful operations. In particular, you will be able to use it to extract or manipulate elements of matrices. The following command creates a row vector whose components increase arithmetically:

```
>> 1:5

ans =
     1     2     3     4     5
```

And, if three numbers, integer or non-integer, are separated by two colons, the middle number is interpreted to be a "range" and the first and third are interpreted to be "limits". Thus

```
» b = 0.0 : 0.2 : 1.0

b =
         0    0.2000    0.4000    0.6000    0.8000    1.0000
```

Suppose you want to create a vector with elements between 0 and 20 evenly spaced in increments of 2. Then you have to type:

```
» t = 0:2:20

t =

     0     2     4     6     8    10    12    14    16    18    20
```

A negative step is also allowed. The command linspace has similar results; it creates a vector with linearly spaced entries.

```
» help linspace

 LINSPACE Linearly spaced vector.
    LINSPACE(x1, x2) generates a row vector of 100 linearly
    equally spaced points between x1 and x2.

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

The colon operator can also be used to create a vector from a matrix. Define:

```
» x = [ 2  6  8
        0  1  7
       -2  5 -6 ]

» y = x(:,1)

y =
     2
     0
    -2
```

Note that the expressions before the comma refer to the matrix rows and after the comma to the matrix columns. The same applies to 3D arrays, where the third index indicates the third dimension.

```
» yy = x(:,2)

     yy =

     6
     1
     5

» z = x(1,:)

z =
     2     6     8
```

The colon operator is also useful in extracting smaller matrices from larger matrices. If the 4 x 3 matrix c is defined by

```
     c = [ -1  0  0
            1  1  0
            1 -1  0
            0  0  2 ]

» d1 = c(:,2:3)
```

creates the following 4 x 2 matrix:

```
d1 =

     0     0
     1     0
    -1     0
     0     2

» d2 = c(3:4,1:2)
```

creates a 2 x 2 matrix in which the rows are defined by the 3rd and 4th row of c and the columns are defined by the 1st and 2nd columns of the matrix, c.

```
d2 =

     1    -1
     0     0
```

Solving linear systems of equations with matrix inversion and division

If *A* is a square, nonsingular matrix, then the solution of the equation *Ax=b* is *x=inv(A)b*. Matlab implements this operation with the backslash operator.

Suppose we have a matrix A and a vector b of random numbers. We can generate these arrays with the following commands:

`>> rand`	returns a random number between 0 and 1.
`>> randn`	returns a random number selected from a normal distribution with a mean of 0 and variance of 1.
`>> rand(A)`	returns a matrix of size A of random numbers from a uniform distribution with mean of 0 and variance of 1.

```
>> A = rand(3,3)

A =

    0.9501    0.4860    0.4565
    0.2311    0.8913    0.0185
    0.6068    0.7621    0.8214

>> b = rand(3,1)

b =

    0.4447
    0.6154
    0.7919
```

The can then find the solution to this system with:

```
>> x = A\b
x =
   -0.0638
    0.6995
    0.3622
```

Thus `A\b` is (mathematically) equivalent to multiplying *b* on the left by `inv(A)`. If you try to do that, you will obtzain the same solution. Try it yourself typing `inv(A)*b`. Another example of how to solve a system of equations using the matrix inverse and the matrix division is given below. Consider the following system of three equations.

```
x1 - 4x2 + 3x3 = -7

3x1 +  x2 - 2x3 = 14

2x1 +  x2 +  x3 =  5
```

Note that you can write this system as Ax = b, where:

```
A = [  1  -4  3
       2  1  -2
       2  1   1]

x = [   x1
        x2
        x3]

b = [   -7
        14
        5]
```

Using 'matlab', define the two matrices for [a] and [b].

```
>>  a = [ 1 -4  3; 3  1 -2; 2  1  1];
>>  b = [ -7; 14; 5];
```

and find the solution vector, x, using the inverse:

```
>>  x = inv(a)*b

x =

    3.0000
    1.0000
   -2.0000
```

Or find the solution using the \ operator:

```
>>  x = a\b

x =

     3
     1
    -2
```

It is often useful, when entering a matrix, to suppress the display. this is done by ending the line with a semicolon.

```
» E = [1 2;3 4];
```

5. Conditionals and loops

Matlab has a standard if-elseif-else conditional. For example:

```
>> t = rand(1);
>> if t > 0.75
      s = 0;
   elseif t < 0.25
      s = 1;
   else
      s = 1-2*(t-0.25);
   end

>> t
t =
    0.7622

>> s
s =
     0
```

Thus the general form of the if statement is

```
if expr1
   statements
 elseif expr2
   statements
 .
 .
 else
   statements
end
```

Matlab provides two types of loops, a for-loop and a while-loop. A for-loop repeats the statements in the loop as the loop index takes on the values in a given row vector:

```
>> for i=[1,2,3,4]
      disp(i^2)      (the  built-in  function  disp,  displays  its
argument.)
   end

     1
     4
     9
    16
```

The loop must also be terminated by end. This loop would more commonly be written as:

```
>> for i=1:4         (recall that 1:4 is the same as [1,2,3,4])
      disp(i^2)
   end
```

```
      1
      4
      9
     16
```

The while-loop repeats as long as the given expr is true:

```
>> x=1;
>> while 1+x > 1
      x = x/2;
   end

>> x
x =
   1.1102e-16
```

Thus, the general form of a while loop is

```
       while   relation
           statements
       end
```

The statements will be repeatedly executed as long as the relation remains true. Another example: for a given number a, the following will compute and display the smallest nonnegative integer n such that 2^n >=a. Let, for example a=4.

```
>> n = 0;
>> while  2^n < a
>>  n = n + 1;
>> end
>> n
n =

     2
```

6. Scripts and Functions

Matlab includes many standard functions. Indeed, all of the standard functions such as sin, cos, log, exp, sqrt, as well as many others. Commonly used constants such as pi are also incorporated into Matlab. Some examples are (a complete list is provided at the end of the tutorial):

```
» sin(pi/4)
ans =

    0.7071
```

Note that, as long as you don't assign a variable a specific operation or result, Matlab will store it in a temporary variable called "ans".

```
» cos(.5)^2+sin(.5)^2
 ans =
       1
```

```
» exp(1)
  ans =
      2.7183

» log(ans)+1
  ans =
       2
```

To determine the usage of any function, type help [function name] at the Matlab command window. For example:

```
» help sqrt

 SQRT   Square root.
    SQRT(X) is the square root of the elements of X. Complex
    results are produced if X is not positive.
    See also SQRTM.
```

Note that, when entering a command such as sqrt into matlab what you are really doing is running an m-file or a script.

What is an m-file or a script?

An m-file is a simple text file where you can place Matlab commands, so, it is simply a collection of Matlab commands in an m-file (a text file whose name ends in the extension ".m", e.g. sqrt.m). When the file is run, Matlab reads the commands and executes them exactly as it would if you had typed each command sequentially at the Matlab prompt. To make life easier, choose a name for your m-file that doesn't already exist. To see if a filename.m exists, type help filename at the Matlab prompt.

```
» help eva

eva.m not found.
```

The m-file must be located in one of the directories in which Matlab automatically looks for m-files. One of the directories in which Matlab always looks is the **current working directory**; the command **cd** identifies the current working directory, and **cd newdir** changes from the working directory to **newdir.**

For simple problems, entering your requests at the Matlab prompt is fast and efficient. However, as the number of commands increases typing the commands over and over at the Matlab prompt becomes tedious. M-files will be helpful and almost necessary in these cases. You actually use m-files to create your own programs.

How to create, save, open and run an m-file?

To create an m-file, choose **New** from the **File** menu and select **m-file**. This procedure brings up a text editor window in which you can enter Matlab commands. To save the m-file when you have typed your commands, simply go to the **File** menu and choose **Save as** (remember to save it with the '.m' extension). To open an existing m-file, go to the **File** menu and choose **Open** .

After the m-file is saved with the name filename.m in the Matlab folder or directory, you can execute the commands in the m-file by simply typing filename at the Matlab prompt. You can also use m-files to create your own functions.

You can either type commands directly into matlab, or put all of the commands that you will need together in an m-file, and just run the file. If you put all of your m-files in the same directory that you run matlab from, then matlab will always find them.

How to create your own function?

The new function must be given a filename with a '.m' extension. For example create a function that is called addition.m, which will add two numbers. The first line of the file should contain the syntax for this function in the form:

```
function [output1,output2,…outputn] = filename(input1,input2,…,inputn)
```

The inputs are what you have to give to the function, in this case the two variables you want to add, and the output will be the sum of the two variables. So, you open a new m-file, as explained above, and type the following:

```
function [var3] = addition(var1,var2)
```

The next few lines contain the text that will appear when the help addition command is evoked. For example, you can write: `%addition is a function that adds two numbers`

These lines are optional, but must be entered using % in front of each line in the same way that you include comments in an ordinary m-file. Finally, below the help text, the actual function with all of the commands is included. In this case, we would then have:

```
function [var3] = addition(var1,var2)
%addition is a function that adds two numbers
var3 = var1+var2;
```

If you save these three lines in a file called "addition.m" in the Matlab directory, then you can use it always by typing at the command line:

```
» y = addition(3,8)

y =
    11
```

Obviously, most functions will be more complex than the one demonstrated here. This example just shows what the basic form looks like. Try help function for more information.

7. Solving nonlinear problems

In addition to functions for numerical linear algebra, Matlab provides functions for the solution of a number of common problems, such as numerical integration, initial value problems in ordinary differential equations, root-finding, and optimization.

Polynomials

In Matlab, a polynomial is represented by a vector. To create a polynomial in Matlab, simply enter each coefficient of the polynomial into the vector in descending order. For instance, let's say you have the following polynomial:

$$s^4 + 3s^3 - 15s^2 - 2s + 9$$

To enter this into Matlab, just enter it as a vector in the following manner

```
» x = [1 3 -15 -2 9]

x =
      1 3  -15  -2  9
```

Matlab can interpret a vector of length n+1 as an nth order polynomial. Thus, if your polynomial is missing any coefficients, you must enter zeros in the appropriate place in the vector. For example,

$$s^4 + 1$$

would be represented in Matlab as:

```
» y = [1  0  0  0  1]
```

You can find the value of a polynomial using the polyval function. For example, to find the value of the above polynomial at s=2,

```
» z = polyval([1  0  0  0  1],2)

z =
      17
```

You can also extract the roots of a polynomial. This is useful when you have a high-order polynomial such as

$$s^4 + 3s^3 - 15s^2 - 2s + 9$$

Finding the roots would be as easy as entering the following command;

```
» roots([1  3  -15  -2  9])

ans =
               -5.5745
                2.5836
               -0.7951
                0.7860
```

Optimization commands

- `fzero root-finding (single variable)`

Let now **f** be a function. MATLAB function **fzero** computes a zero of the function **f** using user supplied initial guess of a zero sought. In the following example let **f(x) = cos(x) –** x. First we define a function **y = f1(x)** in a separate m-file which we call function1.m:

```
 function y = f1(x)
 y = cos(x) - x;
```

Then, we can type:

```
» r = fzero('function1', 0.5)
r =
0.73908513321516
```

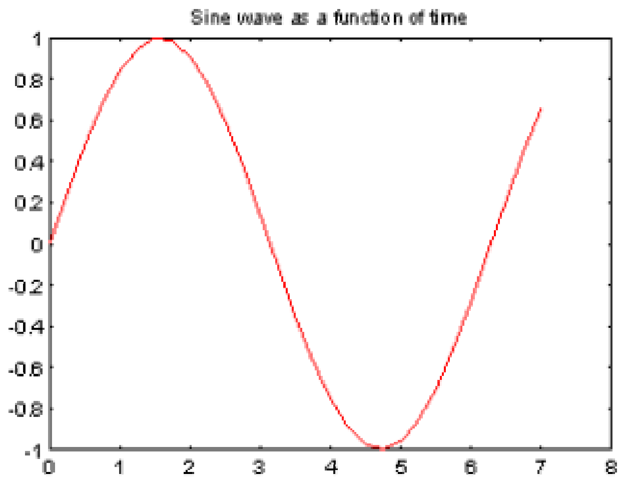
In the first line we tell matlab that f1 is a function and the second line defines the function. To compute its zero we use MATLAB function **fzero** and we have to give an initial guess (type help fzero for more information). Other useful built-in functions are:

- `fsolve` root-finding (several variables)
- `fmin` nonlinear minimization (single variable)
- `fmins` nonlinear minimization (several variables)

8. Plotting

It is easy to create plots in Matlab. Suppose you wanted to plot a sine wave as a function of time. First make a time vector (the semicolon after each statement tells Matlab we don't want to see all the values) and then compute the sin value at each time.

```
» t=0:0.25:7;
» y = sin(t);
» plot(t,y)
```



As you see. it is easy to create the needed vectors to graph a built-in function, since Matlab functions are *vectorized*. This means that if a built-in function such as sine is applied to a array, the effect is to create a new array of the same size whose entries are the function values of the entries of the original array.

The Plot function

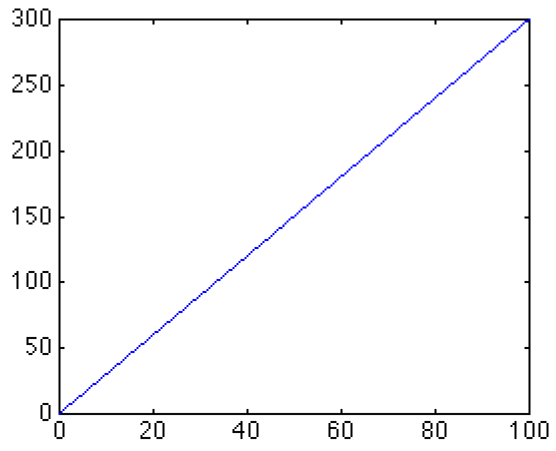
One of the most important functions in Matlab is the plot function. Plot also happens to be one of the easiest functions to learn how to use. The basic format of the function is to enter the following command in the Matlab command window or into a m-file.

```
plot(x,y)
```

This command will plot the elements of vector x on the horizontal axis of a figure, and the elements of the vector y on the vertical axis of the figure. The default is that each time the plot command is issued, the current figure will be erased; we will discuss how to override this below. If we wanted to plot the simple, linear formula: y=3x, we could type the following (or create a m-file with the following lines of code):

```
» x = 0:0.1:100;
» y = 3*x;
» plot(x,y)
```

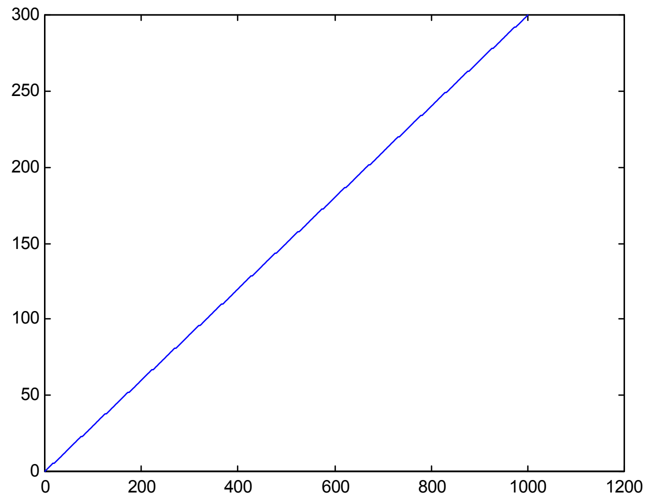
which will generate the following plot,



One thing to keep in mind when using the plot command is that the vectors x and y must be the same length. The plot command can also be used with just one input vector. In this case, the vector 1:1:n will be used for the horizontal axis, where n is the length of y. So, if you type:

```
» plot(y)
```

you will get the following picture:

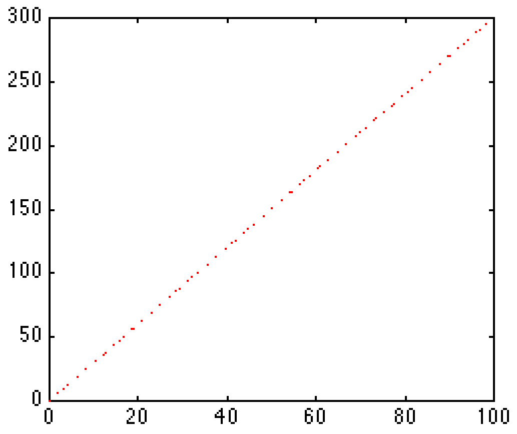


Plot aesthetics

The color and point marker can be changed on a plot by adding a third parameter (in single quotes) to the plot command. For example, to plot the above function as a red, dotted line, the m-file should be changed to:

```
» x = 0:0.1:100;
» y = 3*x;
» plot(x,y,'r:')
```

The plot now looks like:



The third input consists of one to three characters which specify a color and/or a point marker type. The list of colors and point markers is as follows:

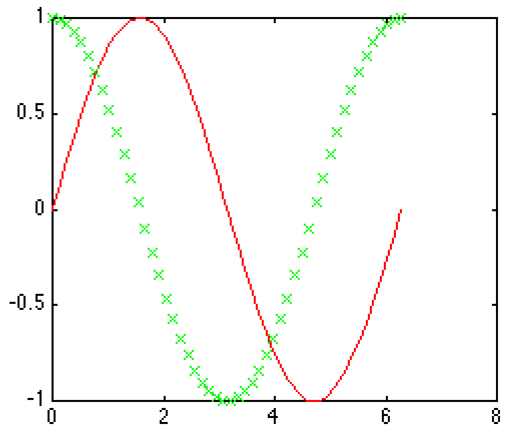
y	yellow	.	point
m	magenta	o	circle
c	cyan	x	x-mark
r	red	+	plus

g	green	-	solid
b	blue	*	star
w	white	:	dotted
k	black	-.	dashdot
		--	dashed

You can plot more than one function on the same figure. Let's say you want to plot a sine wave and cosine wave on the same set of axes, using a different color and point marker for each. The following m-file could be used to do this:

```
» x = linspace(0,2*pi,50);
» y = sin(x);
» z = cos(x);
» plot(x,y,'r', x,z,'gx')
```

You will get the following plot of a sine wave and cosine wave, with the sine wave in a solid red line and the cosine wave in a green line made up of x's:



When plotting many things on the same graph it is useful to use the hold on and hold off commands. The same plot shown above could be generated using the following commands:

```
» x = linspace(0,2*pi,50);
» y = sin(x);
» plot(x,y,'r')
» z = cos(x);
» hold on
» plot(x,z,'gx')
» hold off
```

Always remember that if you use the hold on command, all plots from then on will be generated on one set of axes, without erasing the previous plot, until the hold off command is issued.

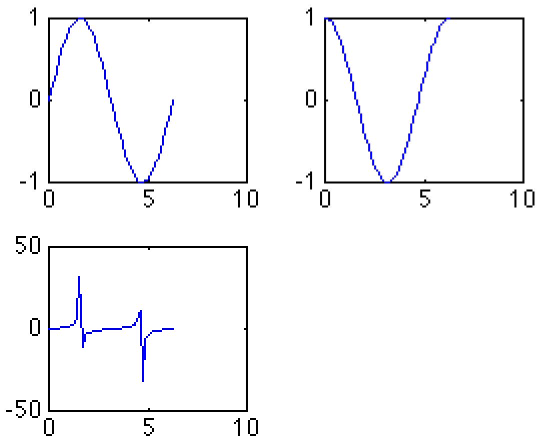
Subplotting

More than one plot can be put on the same figure using the subplot command. The subplot command allows you to separate the figure into as many plots as desired, and put them all in one figure. To use this command, the following line of code is entered into the Matlab command window or an m-file:

```
» subplot(m,n,p)
```

This command splits the figure into a matrix of m rows and n columns, thereby creating m*n plots on one figure. The p'th plot is selected as the currently active plot. For instance, suppose you want to see a sine wave, cosine wave, and tangent wave plotted on the same figure, but not on the same axis. The following m-file will accomplish this:

```
» x = linspace(0,2*pi,50);
» y = sin(x);
» z = cos(x);
» w = tan(x);
» subplot(2,2,1)
» plot(x,y)
» subplot(2,2,2)
» plot(x,z)
» subplot(2,2,3)
» plot(x,w)
```



There are only three plots, even though I created a 2 x 2 matrix of 4 subplots. Thus, you do not have to fill all of the subplots you have created, but Matlab will leave a spot for every position in the matrix. The subplots are arranged in the same manner as you would read a book. The first subplot is in the top left corner, the next is to its right.

One thing to note about the subplot command is that every plot command issued later will place the plot in whichever subplot position was last used, erasing the plot that was previously in it. For example, in the m-file above, if a plot command was issued later in the m-file, it would be plotted in the third position in the subplot, erasing the tangent plot. To solve this problem, a new figure should be specified (using figure). For example, two plots will be opened if you type:

```
» figure(1)
» plot(x,y)
» figure(2)
» plot(x,z)
```

Changing the axis

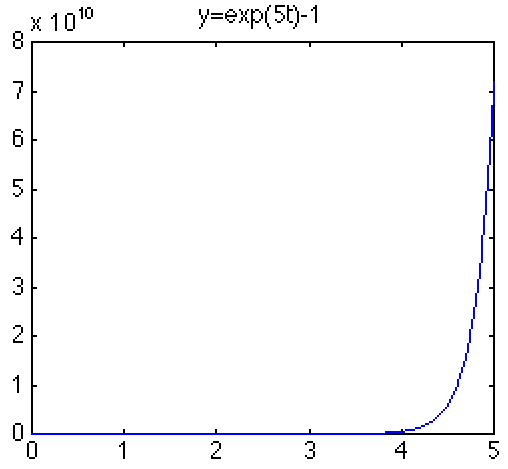
The axis command changes the axis of the plot shown, so only the part of the axis that is desirable is displayed. The axis command is used by entering the following command right after the plot command (or any command that has a plot as an output):

```
» axis([xmin, xmax, ymin, ymax])
```

For instance, suppose want to look at a plot of the function y=exp(5t)-1. If you enter the following into Matlab

```
» t=0:0.01:5;
» y=exp(5*t)-1;
» plot(t,y)
```

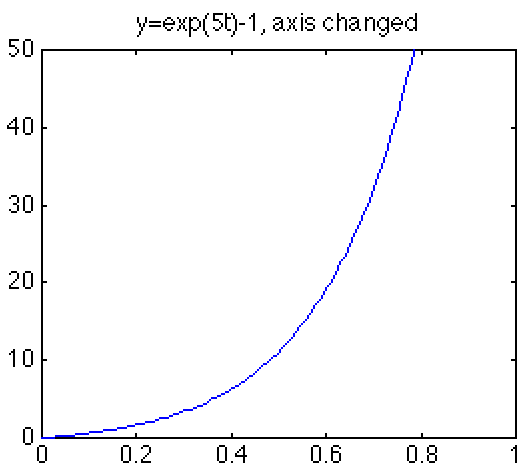
you should have the following plot:



To get a better idea of what is going on in this plot, let's look at the first second of this function. Enter the following command into the Matlab command window.

```
» axis([0, 1, 0, 50])
```

and you should get the following plot:



Now you can see more clearly what is going on as the function moves toward infinity. You can customize the axis to your needs. Type help axis for more information.

Adding text

Another thing that may be important for your plots is labeling. You can give your plot a title (with the title command), x-axis label (with the xlabel command), y-axis label (with the ylabel command), and put text on the actual plot. All of the above commands are issued after the actual plot command has been issued.

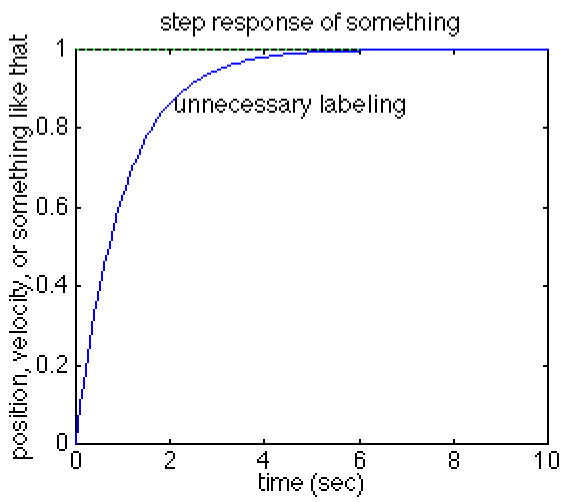
A title will be placed, centered, above the plot with the command: title('title string'). The x-axis label is issued with the following command: xlabel('x-axis string'). The y-axis label is issued with the following command: ylabel('y-axis string').

Furthermore, text can be put on the plot itself with the command gtext('textstring'), and then you just move the cross-hair to the desired location with the mouse, and click on the position you want the text placed.

Suppose you have a plot and you type:

```
» title('step response of something')
» xlabel('time (sec)')
» ylabel('position, velocity, or something like that')
» gtext('unnecessary labeling')
```

The text "unnecessary labeling" was placed right above the position, I clicked on. The plot should look like the following:



You may wish to save in a graphics file, a plot you have created. To do so, simply append to the 'print' command the name of the file. The command

```
>> print filename
```

will store the contents of the graphics window in the file titled 'filename.ps' in a format called postscript.

9. Basic Commands and Functions

- `max(x)` returns the largest entry of x, if x is a vector; see help max for the result when x is a *k*-dimensional array
- `min(x)` analogous to max
- `abs(x)` returns an array of the same size as x whose entries are the magnitudes of the entries of x
- `mean(x)` returns the mean value of the elements of a vector or if x is a matrix, returns a row vector whose elements are the mean value of the elements from each column of the matrix.
- `median(x)` same as mean(x), only returns the median value.
- `sum(x)` returns the sum of the elements of a vector or if x is a matrix, returns the sum of the elements from each respective column of the matrix
- `prod(x)` same as sum(x), only returns the product of elements.
- `std(x)` returns the standard deviation of the elements of a vector or if x is a matrix, a row vector whose elements are the standard deviations of each column of the matrix.
- `sort(x)` sorts the values in the vector x or the columns of a matrix and places them in ascending order. Note that this command will destroy any association that may exist between the elements in a row of matrix x.
- `size(A)` returns a **1x k** vector with the number of rows, columns, etc. of the *k*-dimensional array A
- `save fname` saves the current variables to the file named fname.mat
- `load fname` load the variables from the file named fname.mat
- `clear x` erases the matrix 'x' from your workspace
- `clear or clear all` erases ALL matrices from your workspace

Line continuation

Occasionally, a line is so long that it can not be expressed in the 80 spaces available on a line, in which case a line continuation is needed. In matlab, the ellipsis defining a line continuation is three successive periods, as in "...". For example:

```
>>  4 +  5  +  3 ...
      +  1  +  10  +  2 ...
      + 5
```

10. Suggestions

These are a few pointers about programming and programming in MATLAB in particular.

- Use the indented style. It makes the programs easier to read, the program syntax is easier to check, and it forces you to think in terms of building your programs in blocks.
- In MATLAB, try to avoid loops in your programs. MATLAB is optimized to run the built-in functions. The following two command sequences have the same effect:

  ```
  >> t = (0:0.001:1)';
  >> y=sin(t);
  ```

  and

  ```
  >> t = (0:0.001:1)';
  >> for i=1:length(t)
          y(i) = sin(t(i));
     end
  ```

  However, the explicit for-loop takes much longer as the vectorized sine function. Of course there will be occasions where you cannot avoid using the **for** loop especially when you are working with recursive problems. In this case, you should always assign an initial array of e.g. zeros that will be filled up as the loop progresses.
- Always supress any unecessary outputs with the semicolon (;). If you want to see the outputs as the programme runs, beware that the speed of execution will be significantly higher. When you write your m-file start it with a command **clear all;** this will clear the memory and improve the performance of the processor.
- Last, although it's good to write comments (using %) in your m-file, these increase the execution time, as the computer actually "reads" the line but does not execute it. So, here you face the trade-off of computing time versus readability of your code.

11. Built-in Functions and Help Topics

Built-in functions

This is a list of functions available in Matlab as of 1984, which should be taken as a quick reminder of the most basic tools available.

```
intro    <        chol     end      function lu       quit     sprintf
help     >        clc      eps      global   macro    qz       sqrt
demo     =        clear    error    grid     magic    rand     startup
[        &        clg      eval     hess     max      rcond    string
]        |        clock    exist    hold     memory   real     subplot
(        ~        conj     exit     home     mesh     relop    sum
)        abs      contour  exp      ident    meta     rem      svd
.        all      cos      expm     if       min      return   tan
,        ans      cumprod  eye      imag     nan      round    text
;        any      cumsum   feval    inf      nargin   save     title
%        acos     delete   fft      input    norm     schur    type
!        asin     det      filter   inv      ones     script   what
:        atan     diag     find     isnan    pack     semilogx while
'        atan2    diary    finite   keyboard pause    semilogy who
+        axis     dir      fix      load     pi       setstr   xlabel
-        balance  disp     floor    log      plot     shg      ylabel
*        break    echo     flops    loglog   polar    sign     zeros
\        casesen  eig      for      logop    prod     sin
/        ceil     else     format   ltifr    prtsc    size
^        chdir    elseif   fprintf  ltitr    qr       sort
```

```
acosh        demo         hankel       membrane     print        table1
angle        demolist     hds          menu         quad         table2
asinh        dft          hilb         meshdemo     quaddemo     tanh
atanh        diff         hist         meshdom      quadstep     tek
bar          eigmovie     histogram    mkpp         rank         tek4100
bench        ergo         hp2647       movies       rat          terminal
bessel       etime        humps        nademo       ratmovie     toeplitz
bessela      expm1        idft         nelder       readme       trace
besselh      expm2        ieee         neldstep     residue      translate
besseln      expm3        ifft         nnls         retro        tril
blanks       feval        ifft2        null         roots        triu
cdf2rdf      fft2         info         num2str      rot90        unmkpp
census       fftshift     inquire      ode23        rratref      vdpol
citoh        fitdemo      int2str      ode45        rratrefmovie versa
cla          fitfun       invhilb      odedemo      rref         vt100
compan       flipx        isempty      orth         rsf2csf      vt240
computer     flipy        kron         pinv         sc2dc        why
cond         funm         length       plotdemo     sg100        wow
conv         gallery      log10        poly         sg200        xterm
conv2        gamma        logm         polyfit      sinh         zerodemo
corr         getenv       logspace     polyline     spline       zeroin
cosh         ginput       matdemo      polymark     sqrtm
ctheorem     gpp          matlab       polyval      square
dc2sc        graphon      mean         polyvalm     std
deconv       hadamard     median       ppval        sun
```

```
addtwopi buttap   cov      fftdemo  freqz    kaiser   specplot
bartlett butter   decimate filtdemo fstab    numf     spectrum
bilinear chebap   denf     fir1     hamming  readme2  triang
blackman chebwin  detrend  fir2     hanning  remez    xcorr
boxcar   cheby    eqnerr2  freqs    interp   remezdd  xcorr2
yulewalk
```

```
>> help

HELP topics:

matlab/general        -  General purpose commands.
matlab/ops            -  Operators and special characters.
matlab/lang           -  Language constructs and debugging.
matlab/elmat          -  Elementary matrices and matrix manipulation.
matlab/specmat        -  Specialized matrices.
matlab/elfun          -  Elementary math functions.
matlab/specfun        -  Specialized math functions.
matlab/matfun         -  Matrix functions - numerical linear algebra.
matlab/datafun        -  Data analysis and Fourier transform functions.
matlab/polyfun        -  Polynomial and interpolation functions.
matlab/funfun         -  Function functions - nonlinear numerical methods.
matlab/sparfun        -  Sparse matrix functions.
matlab/plotxy         -  Two dimensional graphics.
matlab/plotxyz        -  Three dimensional graphics.
matlab/graphics       -  General purpose graphics functions.
matlab/color          -  Color control and lighting model functions.
matlab/sounds         -  Sound processing functions.
matlab/strfun         -  Character string functions.
matlab/iofun          -  Low-level file I/O functions.
matlab/demos          -  The MATLAB Expo and other demonstrations.
toolbox/chem          -  Chemometrics Toolbox
toolbox/control       -  Control System Toolbox.
fdident/fdident       -  Frequency Domain System Identification Toolbox
fdident/fddemos       -  Demonstrations for the FDIDENT Toolbox
toolbox/hispec        -  Hi-Spec Toolbox
toolbox/ident         -  System Identification Toolbox.
toolbox/images        -  Image Processing Toolbox.
toolbox/local         -  Local function library.
toolbox/mmle3         -  MMLE3 Identification Toolbox.
mpc/mpccmds           -  Model Predictive Control Toolbox
mpc/mpcdemos          -  Model Predictive Control Toolbox
mutools/commands      -  Mu-Analysis and Synthesis Toolbox.: Commands directory
mutools/subs          -  Mu-Analysis and Synthesis Toolbox -- Supplement
toolbox/ncd           -  Nonlinear Control Design Toolbox.
nnet/nnet             -  Neural Network Toolbox.
nnet/nndemos          -  Neural Network Demonstrations and Applications.
toolbox/optim         -  Optimization Toolbox.
toolbox/robust        -  Robust Control Toolbox.
toolbox/signal        -  Signal Processing Toolbox.
toolbox/splines       -  Spline Toolbox.
toolbox/stats         -  Statistics Toolbox.
toolbox/symbolic      -  Symbolic Math Toolbox.
toolbox/wavbox        - (No table of contents file)
simulink/simulink     -  SIMULINK model analysis and construction functions.
simulink/blocks       -  SIMULINK block library.
simulink/simdemos     -  SIMULINK demonstrations and samples.
toolbox/codegen       -  Real-Time Workshop
```

12. Useful Sites on the Web

-The MathWorks Web site: http://www.mathworks.com/

-Matlab Educational Sites: http://www.eece.maine.edu/mm/matweb.html

-Some Matlab Links: http://math.uc.edu/~kingjt/matlab_lnk.html

-Matlab resources on the web:

http://www.eeng.brad.ac.uk/help/.packlangtool/.maths/.matlab/.resource.html

-Online Matlab Tutorials

http://mechanical.poly.edu/faculty/vkapila/matlabtutor.htm

-One of the best tutorials I found on the web:

http://www.math.siu.edu/matlab/tutorials.html

13. Practice Exercises

1. Determine the size for the following vectors and matrices. Enter them in Matlab and check your results using the 'whos' statement.

```
 a = [1,0,0,0,0,1]

 b = [2;4;6;10]

 c = [5  3  5; 6  2  -3]


 e = [3  5  10  0;  0  0 ...
      0  3;  3  9  9  8  ]

 t = [4  24  9]

 q = [t 0 t]
```

2. Define the 5 x 4 matrix, g.

```
     g = [ 0.6  1.5  2.3 -0.5
           8.2  0.5 -0.1 -2.0
           5.7  8.2  9.0  1.5
           0.5  0.5  2.4  0.5
           1.2 -2.3 -4.5  0.5 ]
```

Determine the content and size of the following matrices and check your results for content and size using matlab.

```
 a = g(:,2)

 b = g(4,:)

 c = [10:15]

 d = [4:9;1:6]

 e = [-5,5]

 f= [1.0:-.2:0.0]
 t1 = g(4:5,1:3)
```

3. Find the solution to

```
          2x1 +  x2 - 4x3 + 6x4 + 3x5 - 2x6 = 16

          -x1 + 2x2 + 3x3 + 5x4 - 2x5       = -7

           x1 - 2x2 - 5x3 + 3x4 + 2x5 +  x6 =  1

          4x1 + 3x2 - 2x3 + 2x4       +  x6 = -1

          3x1 +  x2 -  x3 + 4x4 + 3x5 + 6x6 = -11

          5x1 + 2x2 - 2x3 + 3x4 +  x5 +  x6 =  5
```

using the matrix inverse and matrix division.

4.Create a ten-dimensional row vector whose all components are equal 2.

5. Let x = [2 5 1 6].

  a. Add 16 to each element
  b. Add 3 to just the odd elements
  c. Compute the square root of each element
  d. Compute the square of each element

6. Let x = [3 2 6 8]' and y = [4 1 3 5]'

  a. Add the sum of the elements in x to y
  b. Raise each element of x to the power specified by the corresponding element in y.
  c. Multiply each element in x by the corresponding element in y, calling the result "z".
  d. Add up the elements in z

7. Evaluate the following MATLAB expressions by hand and use MATLAB to check the answers

```
  a. 2 / 2 * 3
  b. 6 - 2 / 5 + 7 ^ 2 - 1
  c. 10 / 2 \ 5 - 3 + 2 * 4
  d. 3 ^ 2 / 4
  e. 3 ^ 2 ^ 2
```

8. Create a vector x with the elements ...

  a. 2, 4, 6, 8, ...
  b. 10, 8, 6, 4, 2, 0, -2, -4
  c. 1, 1/2, 1/3, 1/4, 1/5, ...
  d. 0, 1/2, 2/3, 3/4, 4/5, ...

9. Plot the functions x, x^3, e^x and e^{x2} over the interval 0 < x < 4. Put a title to each function and add some text inside the graphs.

10. Make a good plot (i.e., a non-choppy plot) of the function

```
    f(x) = sin(1/x)   for 0.01 < x < 0.1.

11. Given x = [3 1 5 7 9 2 6], explain what the following commands
    "mean" by summarizing the net result of the command.

  a. x(3)
  b. x(1:7)
  c. x(1:end)
  d. x(1:end-1)
  e. x(6:-2:1)
  g. sum(x)

12. Given the array A = [ 2 4 1 ; 6 7 2 ; 3 5 9], provide the commands
    needed to

  a. assign the first row of A to a vector called x1
  b. assign the last 2 rows of A to an array called y
  c. compute the sum over the columns of A
  d. compute the sum over the rows of A

13. Given the arrays x = [1 4 8], y = [2 1 5] and A = [3 1 6 ; 5 2 7],
    determine which of the following statements will correctly execute
    and provide the result. If the command will not correctly execute,
    state why it will not. Using the command whos may be helpful here.

  a. x + y
  b. x + A
  c. x' + y
  d. A - [x' y']
  e. [x ; y']
  f. [x ; y]
  g. A - 3

14. Given the array A = [2 7 9 7 ; 3 1 5 6 ; 8 1 2 5], explain the
results of the following commands:

  a. A'
  b. A(:,[1 4])
  e. A(:)
  h. [A A(end,:)]
  i. A(1:3,:)
  j. [A ; A(1:2,:)]
  m. sum(A,2)

15.Given the array A from problem 4, above, provide the command that
   will

  a. assign the even-numbered columns of A to an array called B
  b. assign the odd-numbered rows to an array called C
  c. convert A into a 4-by-3 array
  e. compute the square-root of each element of A

16. Provide the right answers and use MATLAB to check them.

1.   if n > 1            a. n = 7   m = ?
```

```
        m = n+1          b. n = 0   m = ?
     else                c. n = -10 m = ?
        m = n - 1
     end

2.   if z < 5            a. z = 1    w = ?
        w = 2*z          b. z = 9    w = ?
     elseif z < 10       c. z = 60   w = ?
        w = 9 - z        d. z = 200  w = ?
     elseif z < 100
        w = sqrt(z)
     else
        w = z
     end

3.   if T < 30           a. T = 50    h = ?
        h = 2*T + 1      b. T = 15    h = ?
     elseif T < 10       c. T = 0     h = ?
        h = T - 2
     else
        h = 0
     end

4.   if 0 < x < 10           a. x = -1   y = ?
        y = 4*x              b. x = 5    y = ?
     elseif 10 < x < 40      c. x = 30   y = ?
        y = 10*x             d. x = 100  y = ?
     else
        y = 500
     end
```

17. Given the vector x = [1 8 3 9 0 1], create a short set of commands that will (use loops for this)

   a. Add up the values of the elements (Check the result with **sum.**)
   b. computes the sine of the given x-values

18. Create an M-by-N array of random numbers (use **rand**). Move through the array, element by element, and set any value that is less than 0.2 to 0 and any value that is greater than (or equal to) 0.2 to 1. (use loops for this)

19. Given x = [4 1 6] and y = [6 2 7], compute the following arrays

   a. $a_{ij} = x_i y_j$ (i.e, the element a(1,1) will be x1*y1)
   b. $b_{ij} = x_i / y_j$
   c. $c_i = x_i y_i$, then add up the elements of c.