MatLab Basics

MatLab was designed as a <u>Ma</u>trix <u>Lab</u>oratory, so all operations are assumed to be done on matrices unless you specifically state otherwise. (In this context, numbers (scalars) are simply regarded as matrices with one row and one column.) A matrix is an array of numbers arranged in rows and columns. You will study these in detail in a Linear Algebra course (Math 1114). Since we will be using MatLab here for Calculus of real numbers, not matrices, some of the arithmetic operators discussed below will be different than the standard ones that you would expect. Press "return" after each MatLab statement as you work through the examples below. Most of the time, the corresponding output from MatLab is also given so that you may check your typing.

An "equals" sign is used to make an assignment. For example, type the following after the ">>" prompt in your command window:

x=2

Now, press "return" to see what Matlab has stored as your variable x.
You should see:

»x = 2

x =

    2

If you just type 2, then MatLab will store it in the default variable, "ans".

»2

ans =

    2

Right now, you have assigned x = (a list consisting of just one number, 2)
You can also give a variable such as x a range of values (i.e., define it to be an array) by using a colon, "**:**". For example, to let "x" take on all of the positive integers from 2 to 4, type:

»x=2:4

x =   2   3   4

The semicolon, **;**, suppresses output. Type "x = 2:4;" and see what happens. You can also enter the array as x = [2, 3, 4] or x = [2 3 4]. Spaces separate the numbers in the last entry.

If you would like the step size to be .5 instead of one, insert the new step size between the first and last number:

»x=2:.5:4

x = 2.0000   2.5000   3.0000   3.5000   4.0000

You may find a negative step size useful:

»x=50:-10:0

x = 50   40   30   20   10   0

Notice that we can redefine the variable x as often as we like.

Arithmetic operations can be performed on each entry of an array x, but you must insert a "period" before certain operators, specifically "*", "/", and "^" for multiplication, division, and exponentiation respectively. These operations have a different definition as matrix operations and without the "period", they would be performed as such. To indicate that you want to perform an element-by-element multiplication, division, or exponentiation, you must precede the operation symbol by a period. For example, type the following. Press return after each line to see the output.

```
x=1:5
y=2:6
x*y
x.*y
x^2
x.^2
x/y        (Notice that you get a number, but not an array.  Matlab computes (xy')/(yy').)
x./y
```

Note that addition and subtraction are always defined as an element-by-element operation so that you do not need the period with them. You will learn in linear algebra that if x = 1:5 = [1 2 3 4 5], then x+1 is not defined. Nevertheless Matlab interprets the command x+1 to mean that the number 1 is to be added to each element of the array x.

»x=1:5

x = 1   2   3   4   5

»x+1

ans = 2   3   4   5   6

If you are just performing a numerical operation such as 5^4, you do not need to define anything, but just type the operation. You can evaluate functions with your defined variable x. For example, Matlab will return values for the sine function for each value of the array x.

```
»x=1:5, sin(x)

x = 1      2      3      4      5

ans = 0.8415    0.9093    0.1411    -0.7568    -0.9589
```

Sometimes it is useful to know how many rows and columns an array contains. The command **size(A)** determines the number of rows and columns in a matrix A.

```
»size(x)

ans =   1    5
```

For the array x defined previously, the response to the **size** command is that x has 1 row and 5 columns.

The **format** command controls the numeric format of the values displayed by Matlab. Some of the different formats are displayed below. For further information or examples of this command, type **help format** or go to the help window and type **format** in the upper left window.

```
»x=[4/3, 1.2345]

>>format short   (Output is to 4 places to the right of the
decimal.)
»x

x = 1.3333      1.2345

»format long  (Output is to 14 places to the right of the
decimal.)
»x

x = 1.33333333333333    1.23450000000000

»format rat   (Output is a fraction.)
»x

x =   4/3        2469/2000
```

Data Points

Since all inputs to Matlab are treated as matrices, you will automatically create lists of data when you use the colon operator described above. For example, suppose you want a table of values for time, using every half-hour over the interval from 1 to 4.

```
»format short
»time=1:.5:4

time =
    1.0000    1.5000    2.0000    2.5000    3.0000    3.5000
4.0000
```

Then, suppose you want corresponding values for temperature but the values are not equally spaced. Simply list the values within square brackets separated by spaces or commas.

```
»temp=[51 53 52 49 49 46 41]

temp =
    51    53    52    49    49    46    41
```

You can access any of these values using parentheses. For example, to find the third value in the temperature list:

```
»temp(3)

ans = 52
```

To combine both time and temperature into a table, use a single quotation mark (the "transpose operator") after each variable. This will convert a row matrix into a column matrix and vice versa. You will study more about the transpose of a matrix in Linear Algebra.

```
»[time',temp']

ans =
    1.0000    51.0000
    1.5000    53.0000
    2.0000    52.0000
    2.5000    49.0000
    3.0000    49.0000
    3.5000    46.0000
    4.0000    41.0000
```

Graphing

Matlab can be used to create different types of graphs. Pay careful attention to the coordinate axes and ranges that you see for each command used.

Use **plot(x,y)** to graph a curve in Matlab. The sizes of the x and y arrays must be the same. In the following example, define the domain from –3 to 3 and create the x array. Unless you tell Matlab how many points to use in the graph, it will use the integers only. The .01 between the colons lets Matlab know that you want to use the numbers .01 units apart from –3 to 3 (–3, -2.99, -2.98, -2.97,…, 2.97, 2.98, 2.99, 3). Then graph $y = x^2$. Remember, the semicolon "**;**" suppresses output.

```
»x=-3:.01:3;
»y=x.^2;  (y = x.^2 creates a y-array of the same size.  Each
            entry in the y-array will be the square of the
            corresponding entry in the x-array.)
»plot(x,y)
```

Note that you can also plot the graph without defining y by typing the following.

```
»x=-3:.01:3;
»plot(x,x.^2)
```

To see what the graph looks like using the default, type x = -3:3 and then plot the graph.

If you want to plot more than one graph on the same set of axes, you must list the domain defined by the variable x for each graph. Below is the graph of $y = 2x+1$ and $y = x^2$ on the same set of axes.

```
»x=-3:.01:3;
plot(x,x.^2,x,2.*x+1)
```
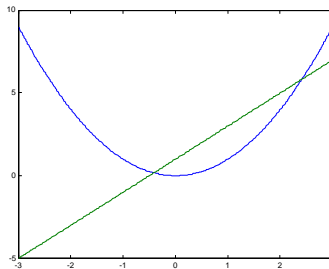
You can also use the **hold** command to plot more than one curve on the same set of axes. Note that you can put more than one command on the same line. They must be separated by commas or semicolons (, or ;).
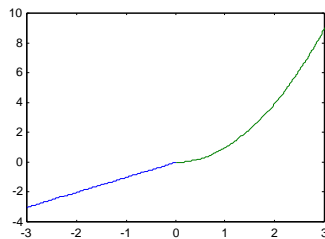
```
x=-3:.01:3;plot(x,x.^2)
»hold on
»plot(x,2.*x+1);hold off
```
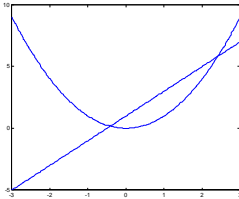

To graph the piecewise function $f(x) = \begin{cases} x, & x < 0 \\ x^2 & x \geq 0 \end{cases}$, you must define two intervals x1 and x2.

Each function must be defined in terms of its interval. For example on interval x2, the function is y = x2.^2 .

```
»x1=-3:.01:0;x2=0:.01:3;plot(x1,x1, x2, x2.^2)
```


To graph horizontal lines (y = b), type **plot([xmin, xmax],[b, b])** where x goes from xmin to xmax. To graph vertical lines (x = a), type **plot([a, a],[ymin, ymax])** where y goes from ymin to ymax. Recall that square brackets surround lists of data. These lines can be used to put axes on graphs.

```
plot([-3,3],[2,2])
```



6
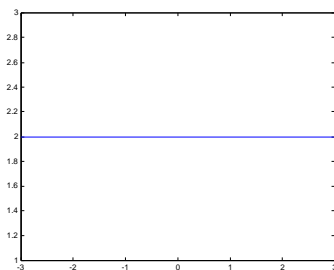
```
plot([2,2],[-3,3])
```
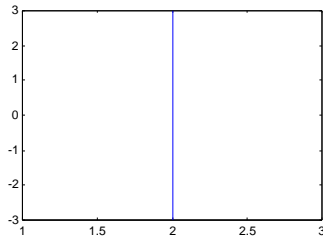

Options

It is possible to specify color, linestyle, and markers, such as plus signs or circles. Use **plot(x,y,'color_style_marker')**. Colors are denoted by 'c', 'm', 'y', 'r', 'g', 'b', 'w', and 'k'. These correspond to cyan, magenta, yellow, red, green, blue, white and black. Linestyles are denoted by '-' for solid, '--' for dashed, ':' for dotted, '-.' for dash-dot, and 'none' for no line. The most common marker types are '+' , 'o', '*', and 'x'.
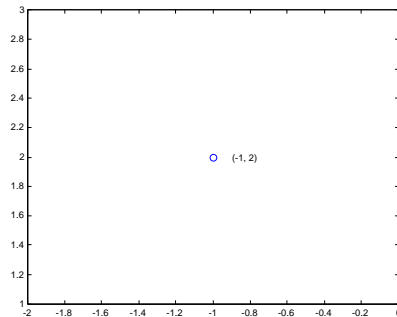
```
>>x=-3:.3:3;plot(x,x.^2,'r--o')
```
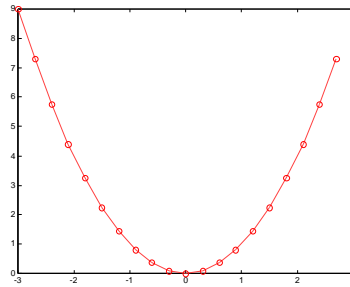

Both axes and graphs may be labeled.

```
»plot(x,x.^2)
»xlabel('x');ylabel('y');title('Graph of x^2')
```
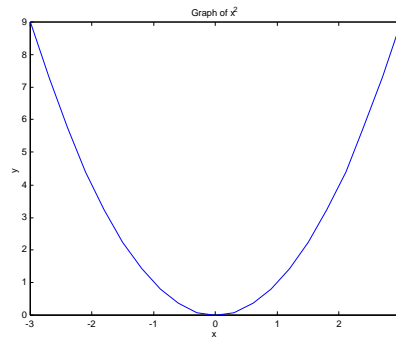
To add text to a graph use **text(x,y, 'text')** where x and y are your x and y coordinates to place the beginning of the text and the words in quotes are the text that you want on your graph.

```
»x=-1;y=2;plot(x,y,'o');text(-.9,2,'(-1, 2)')
```
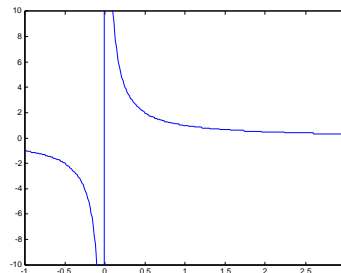
Sometimes it is necessary to adjust the range of the graph. If you want to plot y = 1/x on the domain –1 £ x £ 3 and do not specify the range, some machines will show this incorrect graph.

```
x=-1:.01:3;plot(x,1./x)
```
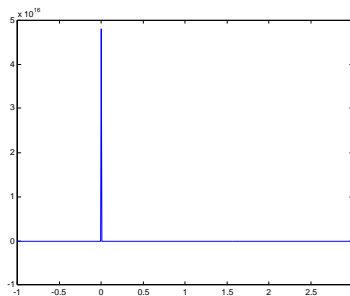
To adjust the range use the command **axis([x1 x2 y1 y2])** where x1 £ x £ x2 and y1 £ y £ y2.

```
x=-1:.01:3;plot(x,1./x);axis([-1 3 -10 10])
```
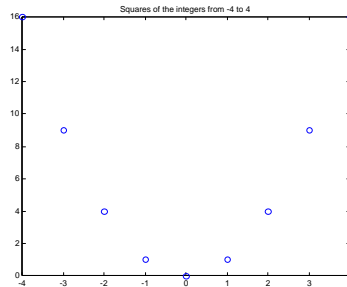
Note that on some machines the vertical line may not be graphed.

8

Graphing data

Let's graph the points (x, x^2) where x takes on integer values between -4 and 4. To graph these points we will type **plot(x,y,'o').** If you do not use a marker, Matlab will connect the points.

```
»x=-4:4;y=x.^2; plot(x,y,'o');title('Squares of the integers
from -4 to 4')
```

You can also graph lists of data. Suppose you want to plot the temperature recorded for 7 days in a particular month. First make tables of the data and then you can plot them.

```
»days=[1 2 3 4 5 6 7];temp=[51 53 52 49 49 46 41];
»plot(days,temp,'o')
»xlabel('days'),ylabel('temperature')
```
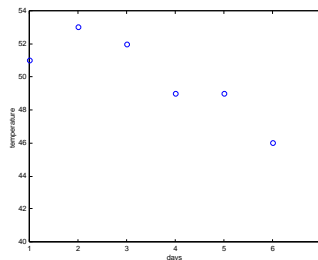
Other commands

Sometimes you might want to zoom in on a graph. The **zoom** command allows you to zoom in on an area of interest in a graph. First type **zoom on**, then click on your graph where you would like a closer look. The command **zoom off** turns it off.

The **abs** command, **abs(x)**, computes the absolute value of x. (x may be a number or an array.)

```
»x=[-4,1/2, 0, -15.178, 32] ,abs(x)

x =   -4.0000     0.5000          0   -15.1780     32.0000
ans = 4.0000     0.5000          0    15.1780     32.0000
```

The **max** command, **max(x)** returns the largest value in an array x. The **min** command, **min(x)** returns the smallest value in an array x. For example, suppose you want the range for $y = \sin x$ over the interval $-\quad x\quad$.

```
»x=-pi:.01:pi; y=sin(x);
»max(y), min(y), abs(min(y))

ans = 1.0000
ans =   -1.0000
ans = 1.0000
```

The **subplot** command allows you to display multiple plots on the same window. The command **subplot(m,n,p)** breaks the figure into an m-by-n array of small subplots and selects the pth subplot for the current plot. The plots are numbered first along the top row of the figure window, then the second row and so on.

```
»x=-3:.01:3;y1=x.^2;y2=2*x+5;
»subplot(2,1,1)      %first plot in a 2 x1 array of plots
»plot(x,y1)
»title('x^2')
»subplot(2,1,2)      %second plot in a 2x1 array of plots
»plot(x,y2)
»title('2x+5')
```
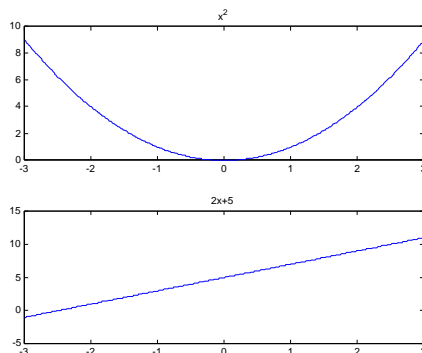
The **ginput** function allows you to get coordinate points from a graph. The form **[x,y]=ginput(n)** finds n points from the current plot or subplot based on mouse click positions within the plot or subplot. If you press the Return or Enter key before all n points are selected, **ginput** stops with fewer points. We will find 5 points on $y = x^2$. Note that the mouse clicks record cursor coordinates and therefore may not be exact points on the curve. For instance in the example below, the point (-0.0227, 0) is chosen, but this point does not lie directly on the curve (x, x.^2).

```
»x=-3:.01:3;
»plot(x,x.^2)
»[x,y]=ginput(5)
x =
   -2.2292
   -0.9295
   -0.0227
    1.5340
    2.3350
y =
    5.0032
    0.8051
         0
    2.3866
    5.5208
```

Using the Symbolic Toolbox

You can define variables to be symbolic so that you can perform some basic algebraic and calculus operations. For example, to define the function, $f(x) = 4x^2 - 3x - 10$, first tell MatLab to treat the letter x as a symbolic variable. Then define the function. When working with symbolic variables, you do not need to use the period before the operation symbol.

```
»syms x
»f=4*x^2-3*x-10

f =

4*x^2-3*x-10
```

To evaluate f(2), you must request a "substitution". The command **subs(f,x,x0)** replaces the variable x with the value x0 in the function f.

```
»subs(f,x,2)
ans = 0
```

To solve equations, use the **solve** command. Notice that the **solve** command assumes that its argument is set equal to zero.
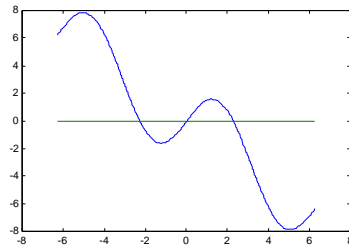
```
»solve(f)

ans =[ -5/4]
     [    2]
```

Suppose we want the solutions to $3\sin(x) - x = 0$ in the interval $-2 \quad x \quad 2$.

```
»solve(3*sin(x)-x)
ans = 0
```

Matlab returns one answer , x= 0. Now look at the graph of $3\sin(x) - x$ over the interval. (The x-axis has been included to help identify points where the function is zero.)

»x=-2*pi:.01:2*pi;plot(x,3*sin(x)-x,[-2*pi,2*pi],[0,0])

There are three values of x where the function equals zero. The **solve** command only finds one of them. If you want the other answers, you must use the command **fzero**. This command identifies points where the function crosses the x-axis. (Note that this command is not symbolic and it does not identify points where the function becomes zero without crossing the x-axis.) The form is **fzero('function', x0)** where x0 is a point near the spot where the function crosses. You will need to look at the graph to find the values for x0. Use the value 2 in **fzero** to find the positive solution and the value -2 to find the negative solution. (You must use the variable x with the **fzero** command; it doesn't work with other variables.)

```
»fzero('3*sin(x)-x',-2),fzero('3*sin(x)-x',2)

ans = -2.2789
ans = 2.2789
```

You can make adjustments to the **solve** command using arithmetic operations.
For example, to solve the equation $4x^2 - 3x - 10 = x$ where $f(x) = 4x^2 - 3x - 10$ was defined previously:

```
»solve(f-x)
ans =
[ 1/2+1/2*11^(1/2)]
[ 1/2-1/2*11^(1/2)]
```

If you want a decimal representation for your answer, use the command **double**. Remember that MatLab stores results in a default variable called **ans** unless you state otherwise. Type:
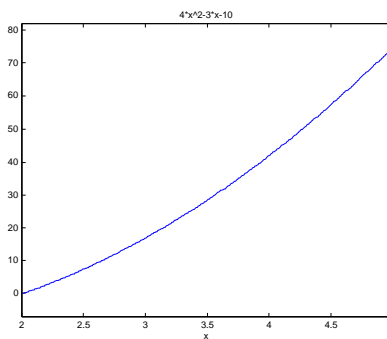
```
»double(ans)
ans = 2.15831239517770
      -1.15831239517770
```

To plot symbolic functions, use "**ezplot**". The default domain is approximately –2pi£x£2pi unless you specify your own. For example type:

```
»ezplot(f)
```
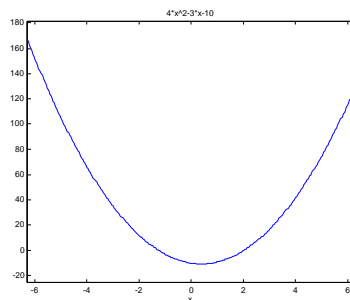


```
»ezplot(f,2,5)
```


You must use the symbolic toolbox when you want to perform some calculus techniques such as differentiation. To differentiate a function we use the command **diff(f).**

```
»syms x
»f=4*x^2-3*x-10

f =
4*x^2-3*x-10

»diff(f)
ans =
8*x-3
```

Index