Matlab Basics 1

Matrices 1: Creating and modifying matrices

MATLAB is built around the idea that every constant or variable is a matrix (or a vector). The most basic way to enter a matrix is to type in its entries between square brackets ([ ]), where rows are separated by semicolons (;). For example, the matrix

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 0 \\ 3 & 1 & 5 \end{bmatrix}$$

is entered by typing [1 2 3; 2 3 0; 3 1 5] at the prompt.† When you're done, you'll see something like this:

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

ans =

     1     2     3
     2     3     0
     3     1     5

>>
```

A vector is simply a matrix with one row or one column. Typing [1 2 4 0] at the prompt produces a *row* vector, and typing [1; 2; 4; 0] produces a *column* vector:

```
>> [1 2 4 0]

ans =

     1    2    4    0

>> [1; 2; 4; 0]

ans =

     1
     2
     4
     0

>>
```

†In the full version of MATLAB , the prompt is '>>', in the educational version the prompt is 'EDU>>'.

You can suppress the output by ending a line with a semicolon. If the matrix you entered wasn't given a name, then it is stored in the temporary variable '`ans`', but it will be lost the next time MATLAB computes anything. It is (almost) always a good idea to give the matrices and vectors that you enter names, so that you can recall them when you need them again. You can use any letter or combination of letters and numbers to name variables,[‡] as long as the word isn't 'protected' by MATLAB, which means that MATLAB uses that word for something. If you type the name of a variable at the prompt, MATLAB produces the value of that variable, as long as it has been defined.

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

>> A

A =

     1     2     3
     2     3     0
     3     1     5

>>
```

MATLAB also provides a variety of ways to gain access to parts of a matrix. If `A` is a matrix, then `A(i,j)` is the entry that appears in row i and column j of `A`. If you want to pick out a block of entries you can use colons to specify a range of rows and columns, e.g, `A(1:3, 1:2)` produces the *submatrix* of `A` consisting of the entries in rows 1 through 3 and columns 1 through 2:

```
>> A=[1 3 5 0; 4 5 1 -1; 6 9 0 0];

>> A(2,4)

ans =

    -1

>> A(1:3, 1:2)

ans =

     1     3
     4     5
     6     9

>>
```

Besides their use, as described above, for designating a range of column or row numbers, colons (':') are used more generally in MATLAB to designate a set of values which

[‡]Variable names must begin with a letter.

MATLAB treats as a vector. The command `m:n` produces the row vector

$$m \quad m+1 \quad m+2 \quad \cdots \quad \tilde{n},$$

where $\tilde{n}$ is the largest number less than or equal to n, such that $\tilde{n} - m$ is a positive integer. If $n - m$ is an integer, then $\tilde{n} = n$. If $n < m$, then `m:n` produces an error message. If you want to designate a different 'step size',[§] you can enter `m:s:n`, where `s` is the step size, which produces the row

$$m \quad m+s \quad m+2s \quad \cdots \quad \tilde{n}(s),$$

where $\tilde{n}(s)$ is the largest number less than or equal to n such that $\tilde{n}(s) - m$ is a positive integer *multiple* of s. E.g.,

```
>> vec = 2:5

vec =

    2  3  4  5

>> vec2 = 2.3:0.5:5.2

vec2 =

    2.3  2.8  3.3  3.8  4.3  4.8

>>
```

You can also 'concatenate' matrices, as long as the dimensions agree. If matrices `A` and `B` have the same number of rows, then typing `[A B]` 'tacks' the columns of `B` to the 'back' of `A` . If matrices `A` and `C` have the same number of columns, then typing `[C; A]` (note the semicolon!) stacks `C` on top of `A`:

```
>> A=[1 2 3; 4 5 6]; B=[1 0; 0 1]; C=[0 0 0];

>> [A B]

ans =

     1    2    3    1    0
     4    5    6    0    1

>> [C; A]

ans =

     0    0    0
     1    2    3
     4    5    6

>>
```

[§]The default step size is 1.

MATLAB also has commands for generating special matrices which can be very useful in a variety of contexts.

(i) The command `ones(n,m)` produces a matrix with n rows and m columns all of whose entries are 1's.

(ii) The command `zeros(n,m)` produces a matrix with n rows and m columns all of whose entries are 0's.

(iii) The command `eye(n)` produces the $n \times n$ *identity* matrix.

Finally, the *transpose* of the matrix `A` is the matrix `A'` whose columns are the rows of `A`. You can think of transposition as 'flipping' a matrix along the diagonal that descends from the upper left corner to the lower right corner. For example, the transpose of $A = \begin{bmatrix} 1 & 0 & 3 \\ 2 & 1 & 8 \end{bmatrix}$ is the matrix $A' = \begin{bmatrix} 1 & 2 \\ 0 & 1 \\ 3 & 8 \end{bmatrix}$. MATLAB produces the transpose of `A` if you type `A'`:

```
>> A = [1 0 3; 2 1 8]

A =

     1     0     3
     2     1     8

>> A'

ans =

     1     2
     0     1
     3     8

>>
```

Matrices 2: Algebra and other operations.

Once you have matrices and/or vectors to play with, MATLAB allows you to perform all the usual operations of matrix algebra. You can add to matrices (of the same dimension), you can multiply a matrix by a scalar and, if `A` and `B` have the correct dimensions,[¶] you can multiply `A` by `B`. To perform these operations, we use $+$ for addition, $-$ for subtraction and $*$ for multiplication. Furthermore, if the matrix `A` is invertible,[‖] then MATLAB interprets `B/A` as $\texttt{BA}^{-1}$, (provided the product $\texttt{BA}^{-1}$ is defined). And, while we're on the subject, if `A` is invertible, then $\texttt{A}^{-1}$ may be computed by typing `inv(A)` (or `A^-1`, see below).

[¶]The product `AB` is defined if and only if the number of columns in `A` is equal to the number of rows in `B`.

[‖]See chapter 2 in the textbook.

We can also multiply two matrices of precisely the same dimensions *entry by entry* in MATLAB . This is definitely not matrix multiplication, but it is convenient in certain settings. The command for this operation is `A.*B`, i.e., the standard multiplication symbol $*$ is preceded by a '.'

```
>> A=[1 2 1; 3 0 1; 0 1 4]; B=[0 1 3; 4 1 2; 1 1 1]; C=[1 ; 1 ; 2 ];

>> A+B

ans =

     1     3     4
     7     1     3
     1     2     5

>> 2*B

ans =

     0     2     6
     8     2     4
     2     2     2

>> A*C

ans =

     5
     5
     9

>> B/A

ans =

     0.1364    -0.0455    0.7273
     0.4545     1.1818    0.0909
     0.4545     0.1818    0.0909

>> A*B

ans =

     9     4     8
     1     4    10
     8     5     6

>> A.*B

ans =

      0     2     3
     12     0     2
      0     1     4
```

If `A` is a matrix and `f(x)` is a numerical function, then typing `f(A)` produces a matrix with the same dimension as `A` whose entries are the values of `f` evaluated at the entries of `A`.

The most common functions may be invoked in MATLAB by using their common names, and a complete list of functions available in MATLAB may be found easily using the MATLAB help window.

If `k` is a positive integer, then $\texttt{A}^{\texttt{k}}$ is simply `A` multiplied by itself `k` times. If `k` is a negative integer and `A` is invertible, then $\texttt{A}^{\texttt{k}}$ is the inverse of $\texttt{A}^{|\texttt{k}|}$. More generally, if `A` is a square matrix and `k` is a real number, then $\texttt{A}^{\texttt{k}}$ has a (complicated) definition within the context of matrix algebra.** Likewise, if `b` is a positive real number, then the expression $\texttt{b}^{\texttt{A}}$ has a matrix-algebraic interpretation which is beyond the scope of this course.

MATLAB distinguishes between the matrix-algebraic operations of exponentiation and raising to a power (which are only defined for square matrices) and the entry-by-entry versions of the same operations. The standard symbol for raising to a power is the *caret* `^` (above the 6 on the keyboard). Typing `A^k` or `k^A` invokes the corresponding matrix-algebraic operation, while typing `A.^k` or `k^A` invokes the entry-by-entry operation.

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

>> A^3

ans =

     11    30
     15    41

>> A.^3

ans =

     1    8
     1   27

>> 2.^A

ans =

     2    4
     2    8

>> 2^A

ans =

     3.7577    6.9766
     3.4883   10.7344
```

**E.g., it is possible to define the *square root* of a square matrix `A`.

Basic plots.

The matlab command `plot` plots points in a two-dimensional figure and connects them with straight line segments. If `y` is a vector with n entries, then the command `plot(y)` plots the points

$$(1, y(1)),\ (2, y(2)),\ (3, y(3)),\ \ldots,\ (n, y(n)).$$

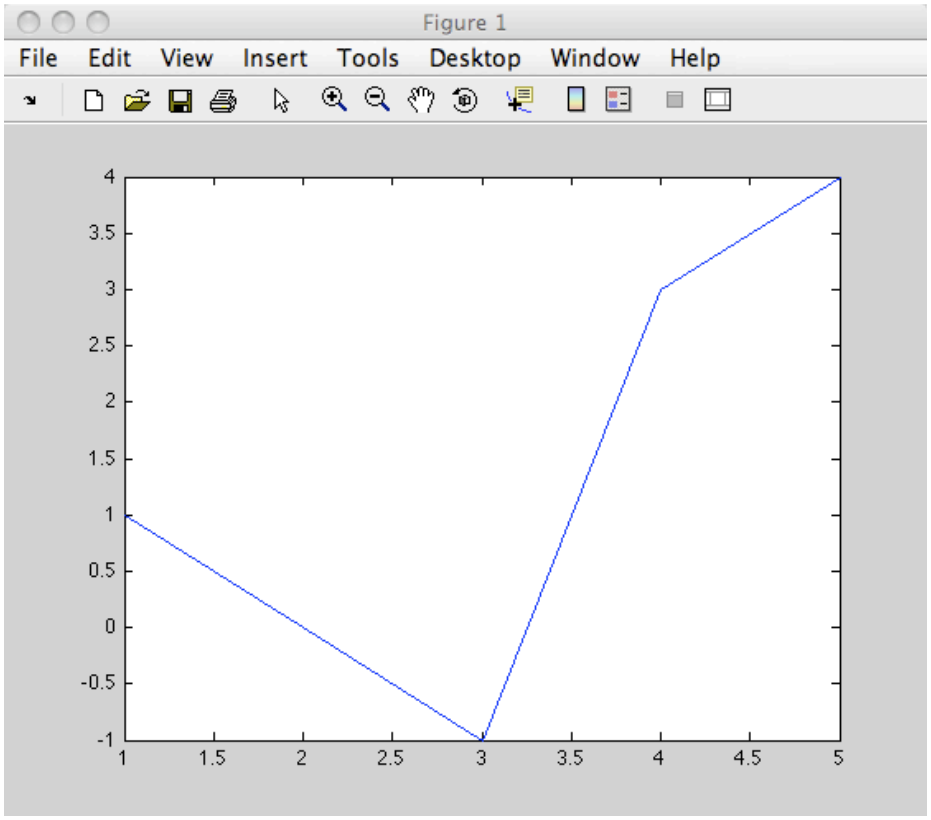
If `x` is another vector with the *same number of entries* as `y`, then the command `plot(x,y)` plots the points

$$(x(1), y(1)),\ (x(2), y(2)),\ (x(3), y(3)),\ \ldots,\ (x(n), y(n)).$$

The plot is displayed in a separate window. For example, the commands

```
>> y = [1 0 -1 3 4];
>> plot(y)
>>
```

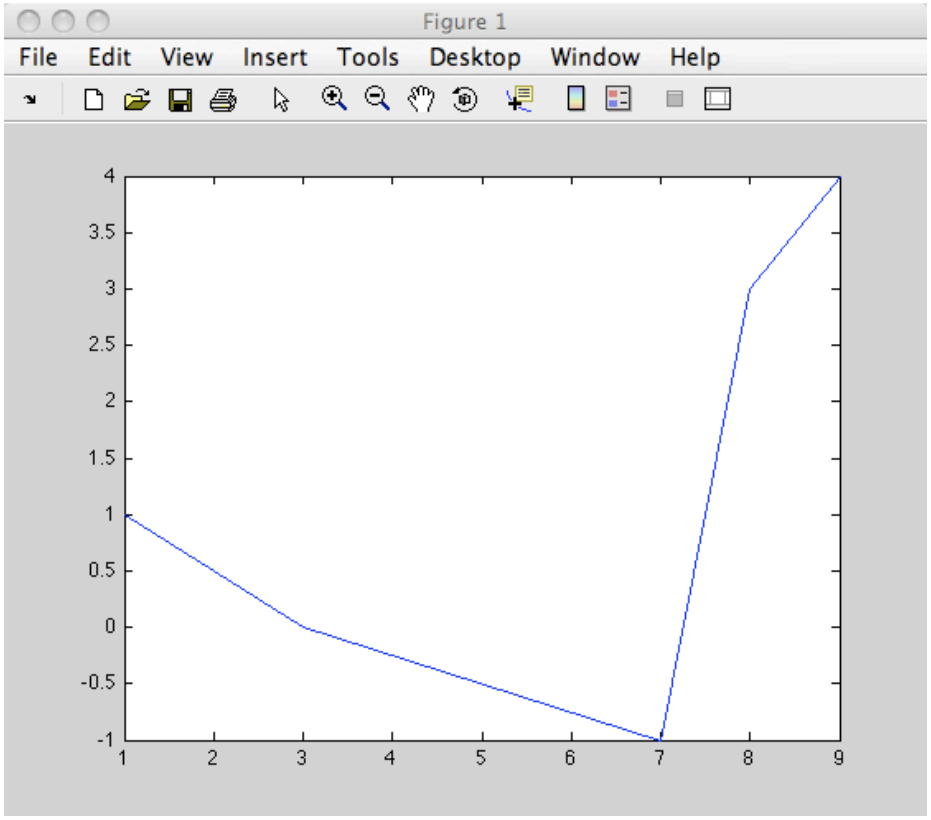
yield the plot



and the commands

```
>> y = [1 0 -1 3 4]; x=[1 3 5 7 10];
>> plot(x,y)
>>
```

produce



Plotting graphs of functions.

If $y = f(x)$ is a function, then the *graph* of this function is the collection of points $\{(x, f(x)) : x \in D\}$ in the plane, where D is the *domain* of the function, i.e., the set of points x where the function is defined. To display the graph of a function we typically choose some interval of values for x, and plot the points of the graph whose x-coordinates lie in the given interval.

In practice, we can't plot all of the points of a graph, even if we limit x to a bounded interval, because there are typically infinitely points to plot. What we do (and by we, I mean graphing calculators and other software that draws pictures for us), is plot a sufficiently dense, but ***finite***, set of points of the graph and connect them with (very short) line segments.[††]

[††]More sophisticated programs connect the dots with 'curved' segments.

To plot the graph of a function in MATLAB , you need to first define a vector that gives the set of x- coordinates of the points in the graph. In most cases this set has the form

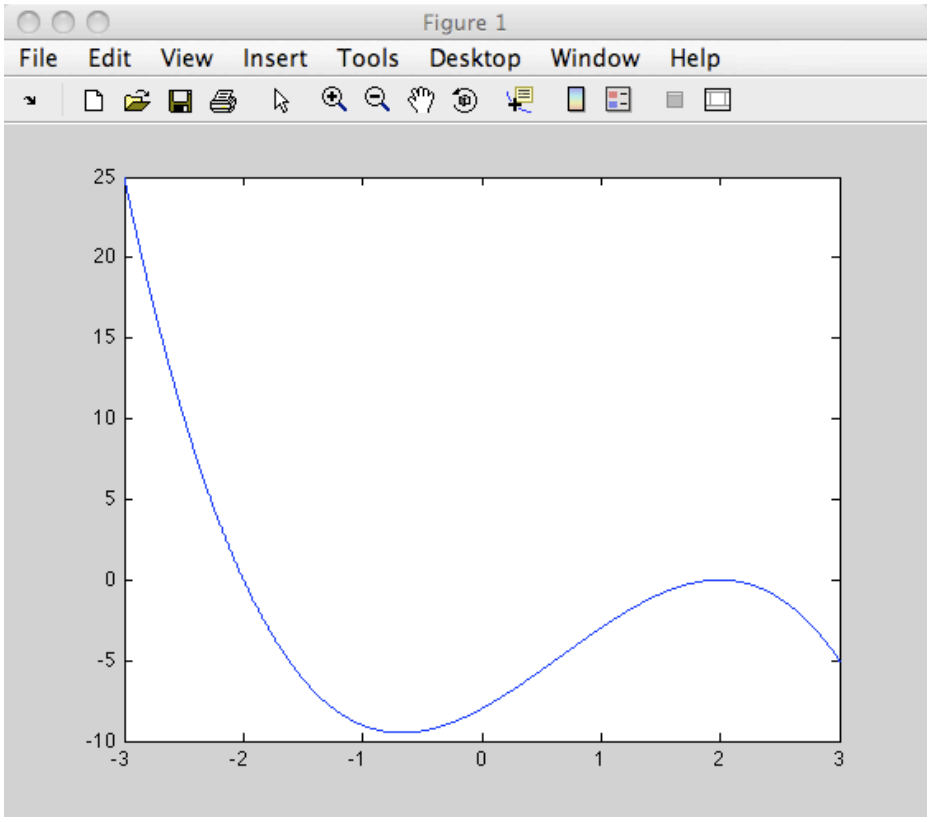
$$\{x = a + ks : 0 \leq k \leq (b-a)\cdot s\} = \{a, a+s, a+2s, \ldots, b-s, b\},$$

where a is the smallest x-value we use, b is the largest x-value and s is the *step size* between consecutive points in the set. Smaller step sizes produce a 'prettier' pictures, but require more computation.

To produce the vector of x values in MATLAB , we use the colon operator. The command `x=a:s:b` creates a vector `x` with approximately $(b-a)/s$ entries going from a to b with steps of size s. Next, the command `y=f(x)` produces the vector of corresponding y-values,[‡‡] and the command `plot(x,y)` produces the desired plot. For example, the commands

```
>> x=-3:0.01:3;

>> y=-x.^3+2*x.^2+4*x-8;

>> plot(x,y)
```

produce the graph of the function $y = -x^3 + 2x^2 + 4x - 8$, for x between -3 and 3:



[‡‡]You need to enter an explicit function at this point, like `y= sin(x)` or `y=x.^2-3*x+1`, not the generic expression `y=f(x)`.

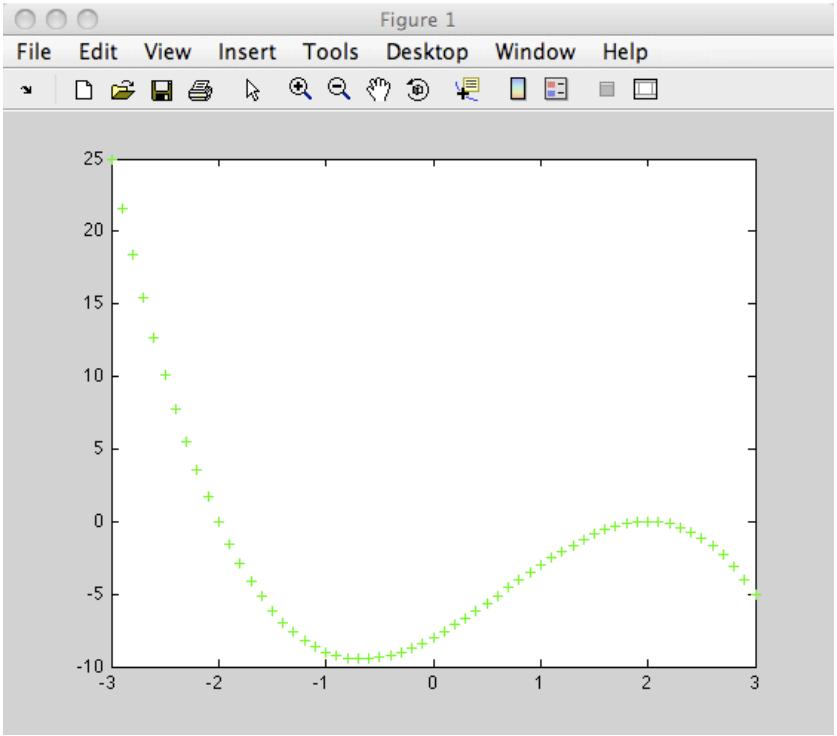
Optional arguments and plotting commands.

There are a variety of optional arguments for the `plot` command in MATLAB that allow users to change colors, change the symbols MATLAB uses as markers for the points being plotted, etc., as well as additional commands to add title to figures, add grids to the plot, and so on.

The default style of a `plot` in MATLAB consists of single dots for the markers of the points being plotted, a solid line for the segments between the points and the color blue. To change these parameters, we add a *string* of letters and symbols following the vector(s) to be plotted in the `plot(...)` command. For example, the string `'go'` specifies green circles as the markers of the points (with no connecting line segments) and the string `':r+'` specifies red plus signs as the markers and (red) dotted lines for the connecting segments. Strings must appear between single quotes ' (on the key with the double quotes, next to the return key). The commands

```
>> x=-3:0.1:3;

>> y=-x.^3+2*x.^2+4*x-8;

>> plot(x,y,'g+')
```

plots the designated points of the graph of the function $y = -x^3+2x^2+4x-8$, for x between -3 and 3, with green plus signs:

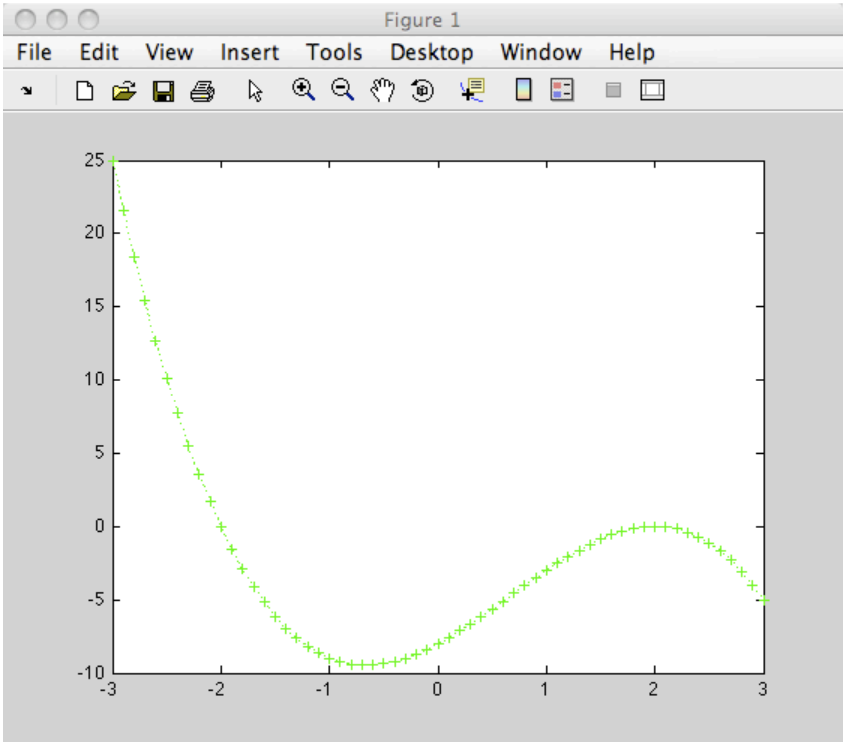


Since no line style was specified, MATLAB assumes that no line segments should be used.

With `x` and `y` as before, the command

```
>> plot(x,y,':g+')
```

connects the points of the previous plot with dotted line segments. Note that when the markers are very close together, MATLAB doesn't put line segments between them.



The color choices, line styles and markers that MATLAB offers are listed in the table below, together with the symbols used to invoke them.

Marker	**Symbol**	**Color**	**Symbol**	**Line style**	**Symbol**
dot (default)	.	blue (default)	b	solid (default)	-
diamond	d	cyan	c	dotted	:
circle	o	green	g	dashed	--
hexagram	h	yellow	y	dash-dot	-.
pentagram	p	red	r		
plus sign	+	magenta	m		
square	s	black	k		
star	*				
triangle down	v				
triangle up	^				
triangle right	>				
triangle left	<				
x-mark	x				

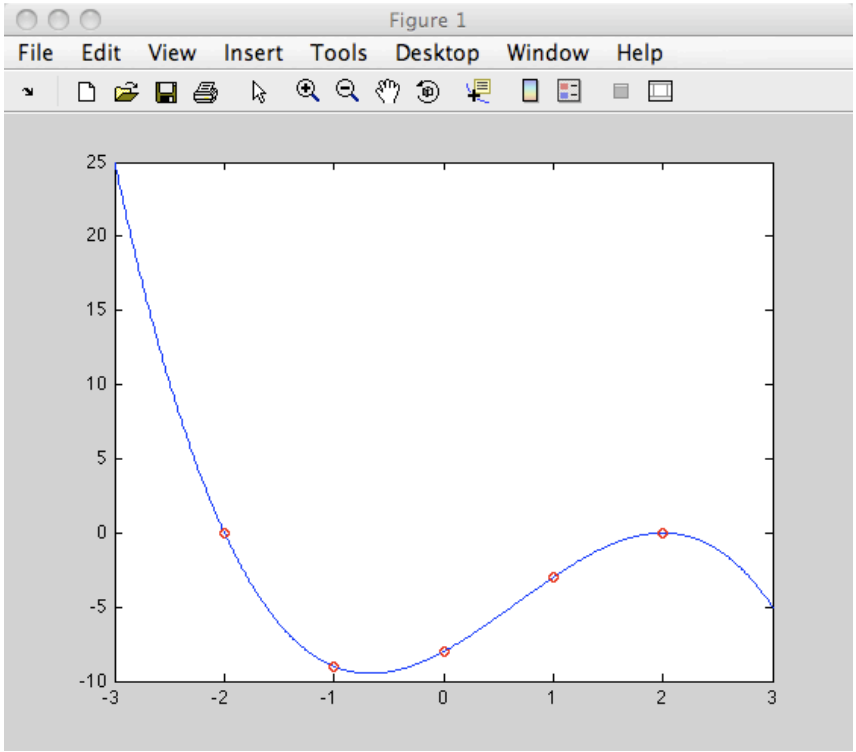
Additional plotting commands.

There are additional basic plotting commands in MATLAB that allow users to plot more than one graph in the same figure, insert a basic grid in the figure, name a figure, etc.

If you type `plot(x,y,...)`, then change something and type `plot(x,y,...)` again, the first plot is simply replaced by the second one. If you want to plot two graphs in the same figure window, you can use the command `hold on` which freezes the current graph in the figure window, so that the next `plot` command adds a graph to the current graph, rather than replacing it. For example, the commands

```
>> x=-3:0.01:3;

>> y=-x.^3+2*x.^2+4*x-8;

>> plot(x,y)

>> hold on

>> z = [−2 −1 0 1 2];

>> u=-z.^3+2*z.^2+4*z-8;

>> plot(z,u,'ro')
```

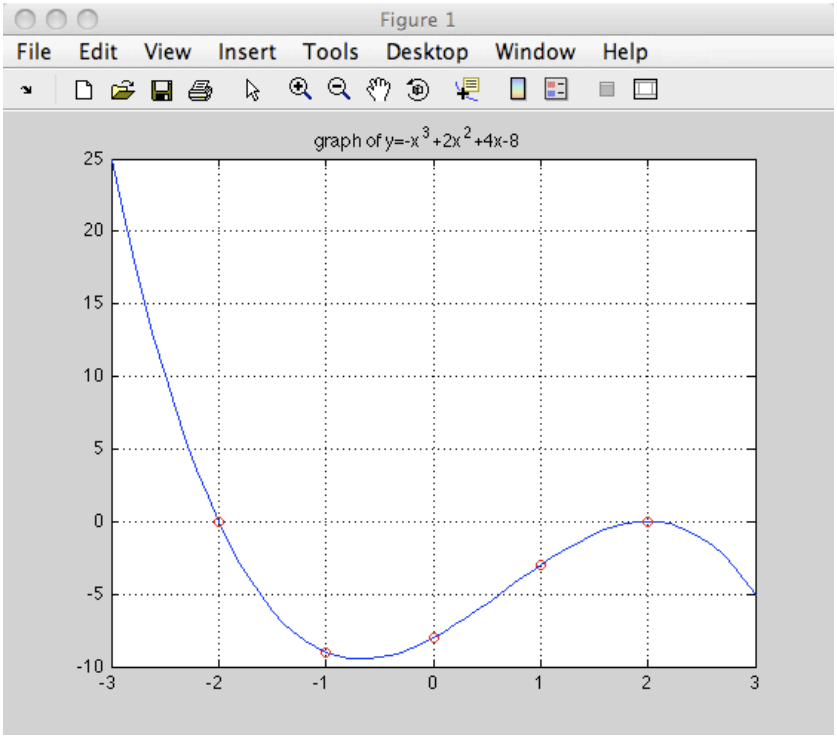
draws the graph of the function $y = -z^3 + 2x^2 + 4x - 8$ and puts red circles at the points $(-2, 0)$, $(-1, -9)$, $(0, -8)$, $(1, -3)$ and $(2, 0)$



Finally (for now), following the previous commands with the commands

```
>> grid on
>> title('graph of y=-x^3+2x^2+4x-8')
```

adds a grid to the figure, as well as the chosen title.



Notice how MATLAB interprets the ^ symbol in the displayed title. Needless to say, you can invoke `grid on` without `title('...')` or vice versa.