Introduction to MATLAB

MATLAB is a technical computing environment for high-performance ***numeric*** (and not typically symbolic) computation and visualization. The name MATLAB stands for *matrix laboratory*. The following information is intended to familiarize the user with the basics of MATLAB: Performing Basic Calculations; Expressions and Parameters; Matrix (and Vector) Calculations; Functions; Plotting; and m-files.

1 Key Features

Before starting, it is important to point out a few *key* features of MATLAB:

- **To use user-defined functions or m-files your PATH must be set to look in the directory containing those files.**
  Before doing anything in MATLAB's command window, it is smart to make sure that the PATH (where MATLAB looks for files and functions besides its own libraries) contains the directory where your files are stored. To do this go to the File menu, select Set Path ..., and Add with Subfolders, then locate the directory you need. Hit OK, then SAVE these changes.
- **Commands in MATLAB do not need to end with any punctuation**
  To execute a command in MATLAB, and to display the results, simply hit Enter after the command line. If you wish to carry out but *suppress* the results, put a semi-colon (;) at the end of the command line.
- **MATLAB is sequential** ...
  In other words, you can not directly edit a line of previous work and re- evaluate it by hitting enter (as you can with Maple). Each command in MATLAB is performed sequentially, and each command has a new line, but ...
- **To retrieve/reuse previously executed lines, simply hit the up ↑ or down ↓ arrows** When working in MATLAB, it is often beneficial to repeat commands, or edit commands with mistakes in them. Rather than copying and pasting, you can use the up and down arrows to scroll through past commands. If you start typing a command, then hit the ↑ key, you can scroll through the history of commands with the same initial format.
- *Ans* **Represents the previous result if it wasn't assigned a parameter or variable name**
  This is similar to the `Ans` key on the TI calculators.
- **Multiplication must be indicated explicitly**
  Although you know that $2x + 4\cos(x)$ means multiply the x value by 2, and the $\cos(x)$ value by 4, then add the results together, you must tell MATLAB about the "understood" multiplication between the 2 and x and the 4 and the $\cos(x)$, by typing $2 * x + 4 * \cos(x)$.
- **MATLAB is for numeric calculations, rather than symbolic**
  The general MATLAB package (without the Symbolic Toolbox) is used for numerical approximations, rather than exact, algebraic calculations.
- **Changing how numbers are displayed**
  All computations in MATLAB are done in double precision. The FORMAT command/parameter may be used to switch between different output display formats by typing `format` *format_type*. Formats include

FORMAT SHORT	Scaled fixed point format with 5 digits.
FORMAT LONG	Scaled fixed point format with 15 digits.
FORMAT RAT	Approximation by ratio of small integers.

2 Performing Basic *Scalar* Calculations in MATLAB

MATLAB can be used as a simple calculator. Simply type in the expression that you want to evaluate after the MATLAB prompt (`>>`) and hit enter (end your statement with a semi-colon (;) if you want to suppress the output. Note, that there some special function calls you may need: $|-2|$ is expressed by `abs(2)`; e^3 is expressed by `exp(3)`; $\arcsin(4) = \sin^{-1}(4)$ is expressed by `asin(4)`; $\sqrt{5}$ is expressed by `sqrt(5)`; $\ln x$ is expressed by `log(x)`, and π is expressed by `pi`.

Example: Verifying $\sin^{-1}\left(\sin\left(\frac{5\pi}{4}\right)\right) = -\frac{\pi}{4}$:

```
>> asin(sin(5*pi/4))

ans =

   -0.7854

>> -pi/4

ans =

   -0.7854
```

3 *Scalar* Expressions and Parameters

As was mentioned above, MATLAB expresses all variables as matrices, essentially. In order to **define** an parameter value in MATLAB, you simply type the name of the parameter, followed by an equal sign, and its value. For example, to define the parameter, *tol*, to be 10^{-8}, you enter the following at the MATLAB prompt: `tol = 10^(-8);`. Thus, every time you refer to *tol* in the future, MATLAB will replace it's value with 10^{-8}.

Example: Setting the tolerance, *tol*, to be 10^{-8}:

```
>> tol = 10^(-8);
>> format long
>> tol

tol =

1.000000000000000e-008
```

4 Matrix and Vector Calculations

4.1 Entering Matrices and Vectors

In MATLAB, the values of a vector or matrix are enclosed by brackets, and rows of matrices are separated by semi-colons. For example to define

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} 2 \\ 4 \end{bmatrix}, \quad \text{and } \mathbf{y} = [5, 4, 3, 2, 1]$$

type in

```
A = [1 2 3; 4 5 6]
```

```
x = [2; 4]
y = [5 4 3 2 1]
```

There are several commonly used matrices and vectors, so they have special "shortcuts". A few of particular interest are given below.

`A = ones(3,1)`	creates a 3×1 vector of ones
`B = zeros(5)`	creates a 5×5 matrix of zeros
`C = diag([3,2,1])`	creates a 3×3 matrix whose main diagonal is $[3, 2, 1]$ (off-diagonals are 0)

There are also special commands to generate vectors with regularly spaced entries.

`x = linspace(a,b)`	creates a row vector of 100 values from a to b
`y = linspace(a,b,N)`	creates a row vector of N values from a to b
`z = [1:8]`	creates a row vector of values from 1 to 8 in increments of 1
`w = [1:0.1:8]`	creates a row vector of values from 1 to 8 incremented by 0.1

Example: Creating the matrix, D, with diagonal elements $-3, -1, 1, 3$:

```
>> d = -3:2:3

d =

    -3    -1     1     3

>> D = diag(d)

D =

    -3     0     0     0
     0    -1     0     0
     0     0     1     0
     0     0     0     3
```

4.2 Referencing an Entry in a Matrix or Vector

Matrices in MATLAB are indexed by integers starting with 1. To see the size of a matrix A, type `size(A)`, which will return the dimensions of the matrix (or you can use `length(x)` for a vector `x`). To see or use the certain entries of the matrix A or vector `x`, do the following:

`A(i,j)`	references the ij^{th} entry of A
`A(:,j)`	references the entire j^{th} column of A (think of the ":" as symbolizing ALL, hence A(:,j) = all rows for col j)
`A(i,:)`	references the entire i^{th} row of A
`x(k)`	references the k^{th} element of the vector `x`

Example: Imbedding one matrix into another:

```
>> A = zeros(4,5)

A =

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

```
     0     0     0     0     0

>> B = [-5,-4,-3; -3,-4,-5];

>> A(2:3,2:4) = B

A =

     0     0     0     0     0
     0    -5    -4    -3     0
     0    -3    -4    -5     0
     0     0     0     0     0
```

4.3 Matrix and Vector Arithmetic

`+, -, *,` $\wedge$ have the same meaning as one would expect with matrices. There are some special operations and functions as well:

'	transpose	e.g. `A'` is the transpose of A
\	left division	e.g. `A\ b` is $A^{-1} * b$
/	right division	e.g. `b/A` is $b * A^{-1}$
`inv()`	inverse	e.g. `inv(A)` is the inverse of A
`det()`	determinant	e.g. `det(A)` is the determinant of A

NOTE: There are special **component-wise** multiplication and power operations, indicated with a $\cdot$ in front of the symbol.

.*	array multiplication	e.g. `x.*y` is the vector resulting from the component-wise multiplication of **x** and **y** entries.
.$\wedge$	array powers	e.g. `x.`$\wedge$`3` is the vector resulting from cubing each entry of **x**.

Example: Cubing a vector's components:

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

Sean =

     1     2     3     4

>> Sean_Cubed = Sean.^3

Sean_Cubed =

     1     8    27    64
```

5 Functions

There are several ways to define functions in MATLAB.

5.1 Using m-files

New functions may be added to MATLAB's vocabulary if they are expressed in terms of other existing functions. The commands and functions that comprise the new function must be put in a text file whose name defines the name of the new function, and saved with a filename extension of '.m'. (Do this with MATLAB's text editor or any text editor such as Notepad.) At the top of the file must be a line that contains the syntax definition for the new function:

`function` $[list\ of\ outputs] = function\_name(list\ of\ input\ variables)$

5.1.1 Defining the Function

Example: The following is in the file titled `my_stats.m`:

```
function [mean,stdev] = my_stats(x)
% STAT Interesting statistics.
% The vector x contains the observations
n = length(x);
mean = sum(x) / n;
stdev = sqrt(sum((x - mean).^2)/n);
```

5.1.2 Evaluating the Function

Your path MUST contain the directory this m-file is saved in, then you can simply type the name of the function, with the inputs in parenthesis:

```
>> [mu,s]=my_stats([3, 22, 18, 9])

mu = 13

s = 7.4498
```

or you can use the `feval` command, where it takes as arguments, the name of the function in single quotes, followed by the input to the function:

```
>> [m,std] = feval('my_stats',[3,22,18,9])

m = 13

std = 7.4498
```

Note the single quotes are needed to call this m-file function, or you could use an @ symbol in front of the file name.

5.2 As an inline function

For simple functions that have only one or two variables for input, it may be easiest to define them as inline functions. Inline functions are functions created on the command line with the syntax:

$function\_name =$ `inline('`$function\ \ formula$`')`

MATLAB will generally be able to identify the variables in the function (or you can specify them – see `help inline`).

5.2.1 Defining the Function

Example: Defining $f(w, v) = sin(w) * cos(v)$ as an inline function

```
>> f=inline('sin(w)*cos(v)')

f =

     Inline function:
     f(v,w) = sin(w)*cos(v)
```

5.2.2 Evaluating the Function

You can evaluate an inline function directly:

Example: Evaluating $f(w, v) = sin(w) * cos(v)$ at $w = \frac{\pi}{4}$ and $v = \frac{\pi}{6}$:

```
>> f(pi/4,pi/6)

ans =

    0.3536
```

Or with the `feval()` command:

Example: Evaluating $f(w, v) = sin(w) * cos(v)$ at $w = \frac{\pi}{4}$ and $v = \frac{\pi}{6}$:

```
>> feval(f, pi/4, pi/6)

ans =

    0.3536
```

Note that the single quotes aren't needed here.

6 Plotting

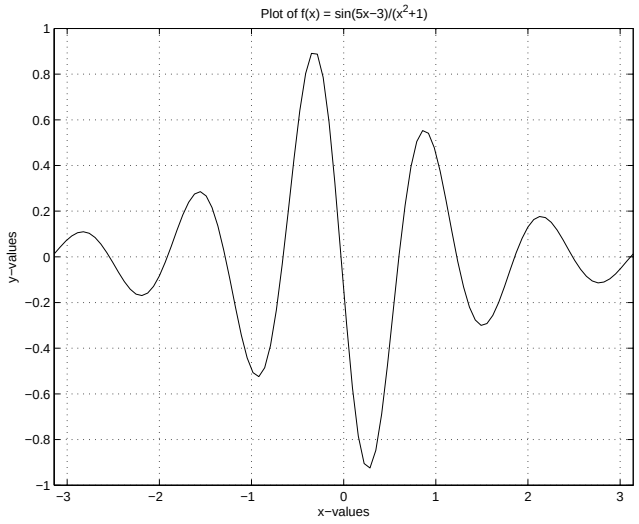
MATLAB is capable of plotting in 2D and 3D, but in most cases, the user must supply vectors of input and output values, rather than the function expression. The best way to learn about the plotting abilities of MATLAB is to do a `help` command for the plotting functions. To access MATLAB's help files, simply type `help` *function_name* at the prompt. Below are a few examples of plotting.

6.1 Plotting with `plot`:

Format: `plot(`*input_vector*`,`*output_vector*`,` *options*`);`

Example: Plotting $f(x) = \dfrac{\sin(5x - 3)}{x^2 + 1}$ over $-\pi \leq x \leq \pi$

```
>> x = linspace(-pi,pi);
>> y = 1./(x.^2+1).*sin(5.*x-3);
>> plot(x,y);
>> hold on;
>> grid on;
>> axis([-pi,pi,-1,1])
>> title('Plot of f(x) = sin(5x-3)/(x^2+1)')
>> xlabel('x-values')
>> ylabel('y-values')
```

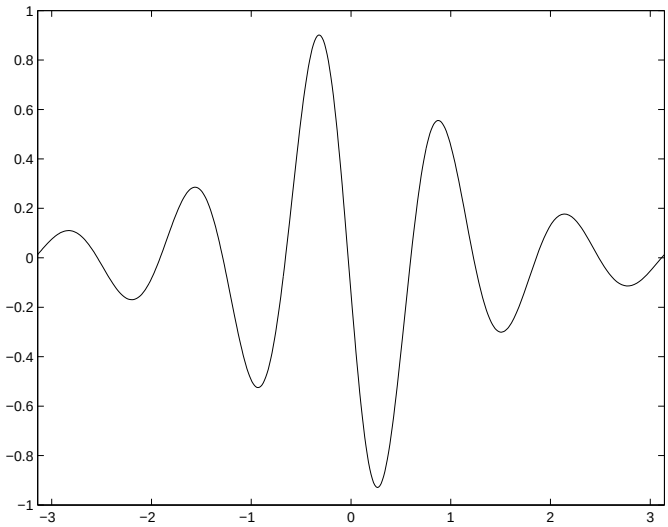


6.2 Plotting with `fplot`:

`fplot` can only be used to plot a single-variable function, with the variable x:
Format: `fplot('`$function\_formula$`',[`a`,`b`])`

Example: Plotting $f(x) = \dfrac{\sin(5x-3)}{x^2+1}$ over $-\pi \leq x \leq \pi$

```
>> fplot('sin(5*x-3)/(x^2+1)',[-pi,pi])
```



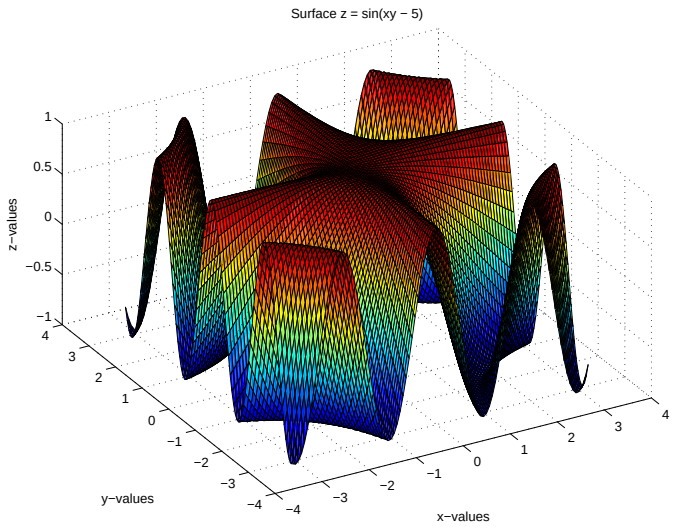
6.3 Plotting with in 3D with `surf` and `mesh`:

To plot in a function of the form $z = f(x, y)$ for $a \leq x \leq b$ and $c \leq y \leq d$, you must generate matrices, rather than vectors of values, in order to represent all possible (x, y) combinations. The `meshgrid()` function generates these easily. These matrices serve as input to the function z, then you can plot the resulting surface supplying the input matrices and resulting matrix of z values.
Format:

```
xrange = linspace(a,b);
yrange = linspace(c,d);
[X,Y] = meshgrid(xrange,yrange);
Z = <desired function of X and Y goes here>;
surf(X,Y,Z); or mesh(X,Y,Z)
```

Example: Plotting the surface $z = sin(xy - 5)$ over $-\pi \leq x, y \leq \pi$

```
>> xrange = linspace(-pi,pi);
>> yrange = linspace(-pi,pi);
>> [X,Y] = meshgrid(xrange,yrange);
>> Z = sin(X.*Y - 5);
>> surf(X,Y,Z)
>> title('Surface z = sin(xy - 5)')
>> xlabel('x-values')
>> ylabel('y-values')
>> zlabel('z-values')
```



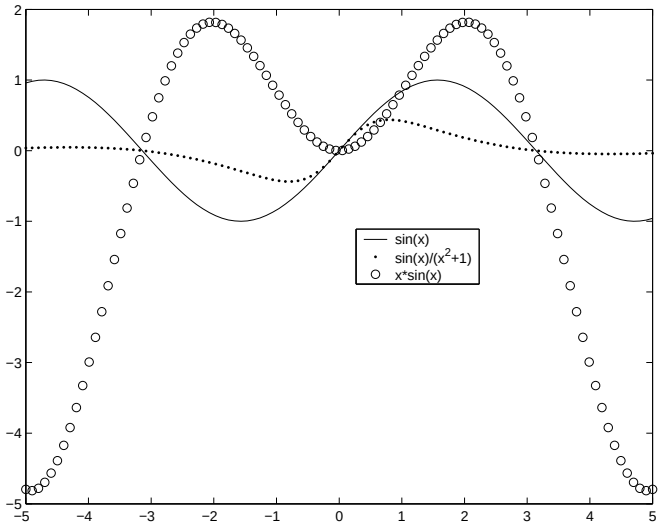
6.4 Plotting multiple functions:

There are a couple of ways to plot multiple functions:

1. Plot your first function. Then type `hold on`. Continue plotting functions.
2. Most of MATLAB's plotting functions allow you to plot a set of functions in one command. `PLOT(X1,Y1,S1,X2,Y2,S2,X3,Y3,S3,...)` combines the plots defined by the (X,Y,S) triples, where the X's and Y's are vectors or matrices and the S's are strings specifying drawing options.

Example: Plotting graphs of $y = \sin(x)$, $y = \frac{\sin(x)}{x^2+1}$ and $y = x\sin(x)$ over $-5 \leq x \leq 5$

```
>> x=linspace(-5,5);
>> plot(x,sin(x),'-b',x,sin(x)./(x.^2+1),'.g',x,x.*sin(x),'or')
>> legend('sin(x)','sin(x)/(x^2+1)','x*sin(x)',0)
```



6.5 Helpful Commands

As the examples above have shown, additional commands are needed to add title and axes labels, grids, etc. The following are just a few of the typical commands used when plotting in MATLAB:

`axis`, `grid`, `title`, `xlabel`, `hold`, `legend`

7 m-files

You can create your own more complicated functions or programs in MATLAB using m-files. You've seen procedures before as one way to define a function. M-files are simply text files (you can create these with the MATLAB editor, represented by the blank page icon on the menu bar) or with any text editor (such as Notepad). You must simply save the text file with a '.m' extension. In order to get started writing your own m-files in MATLAB, there are a few things you should be familiar with:

7.1 Local Variables:

MATLAB does not require you to define a local variable before using it.

7.2 Function Files versus Script Files

You can use m-files to define a function, which takes an input and returns an output (as discussed earlier). These require the header

`function` $[list\ of\ outputs] = function\_name(list\ of\ input\ variables)$.

You can also use m-files as script files. These simply contain the commands as you would type them at the MATLAB prompt. By typing the name of the script file at the MATLAB prompt, MATLAB will execute

the commands listed in the script file. NOTE: all variables within the script file are considered local and are not accessible from the command window, unless they are declared to be global (in the script file and in the command window).

7.3 Looping:

The example above illustrates one looping structure in MATLAB. There are several key things to notice:

- Indexed loops don't contain the words "from (start) to (end)", rather you give the index (and possibly step size) in the same way you generate an array of values with the colon (:) notation, such as `for i=1:100`
- All loops must have an "end" statement.
- There are three types of loops we'll make use of:

1. for loops:
   Example:

```
>> y = linspace(-10,10,5)

y =

   -10    -5     0     5    10

>> for i=1:length(y) x(i) = y(i)^2; end;
>> x

x =

  100.0000   25.0000   0   25.0000  100.0000    0.2225   -0.6235   -1.0000
```

2. if / else loops:
   Example:

```
if (a < b)
   k = 0;
elseif (a == b)
   k = 1;
else
   k = 2;
end;
```

3. while loops:
   Example:

```
while (err>tol)
    x(i+1)= feval('fofx',x(i));
   err = abs(x(i+1)-x(i));
   i = i+1;
end;
```