An Introduction to MATLAB and M-Files

The MATLAB program efficiently manipulates matrices of different dimensions ($m \times n$). MATLAB can also handle scalars and vectors by treating them as 1×1 and $1 \times n$ (or $n \times 1$) matrices. The ability of MATLAB to organize data into matrix form allows the user to perform theoretical and numerical system analyses within a friendly environment.

General Information about MATLAB

- The double greater than sign (`>>`) is the MATLAB command prompt.
- MATLAB is case-sensitive: "A" is not "a". To deactivate case-sensitivity type `casesen` at the command prompt.
- Using the arrow keys can prevent retyping of commands. The "up" and "down" arrow keys will allow for scrolling through previously typed commands and the "left" and "right" arrow keys will allow for editing the current command at the prompt.
- MATLAB has 5 permanent variables:

  i = 1.00∗sqrt(-1)   NaN = 0.0/0.0   pi = 3.1415...

  j = 1.00∗sqrt(-1)   Inf = 1.0/0.0

- Semicolons after statements prevent listing of the variable(s).
- You can get help by typing `help` at the prompt or by clicking the mouse button on the command bar Help located across the top of the screen. Once in the help window you can search for a specific term. Also, typing in `help` *`command-name`* will display a help screen about that particular command.
- The `clear` command will clear the session by deleting all variables. To clear the screen type `clc` at the command prompt, and to clear the graphics window type `clg` at the command prompt.

Methods of Inputting Data

A single variable is the simplest input in MATLAB and only requires typing the variable's name, an equal's sign, and its value at the prompt (just like regular computer programming). The value can be real, complex, or a function of other variables.

```
>> x = 5 + 1i
x =
      5.00 + 1.00i
```

Remember even though x is a single element, MATLAB still treats it as a 1×1 matrix. To create larger matrices there are three important rules:

- Matrices are surrounded by brackets [ ].
- Matrix elements are separated by commas or blanks.
- Rows are separated by semicolons.

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

Parts of matrices, including single elements, rows, or columns may be extracted to perform further operations. Extracting an element from the 3rd row and 2nd column of the **A** matrix is done by:

```
>>A(3, 2)

ans =

      8
```

Extracting the entire 3rd row of the **A** matrix is done by:

```
>>A(3, :)

ans =

      7     8     9
```

Extracting the 2nd column of the **A** matrix is done by:

```
>>A(:, 2)

ans =

      2

      5

      8
```

Another useful method for inputting data into a 1 × *n* matrix (vector) is an array. Series of numbers (arrays) are use a colon to separate the starting number, the step size, and the last number of the array. The default step size is 1.

```
>> t = 1: 5

t =

      1     2     3     4     5

>> t = 2: -0.5: 0

t =

      2.0   1.5   1.0   0.5   0.0
```

Remember, placing a semicolon after the input prevents MATLAB from listing the variable. MATLAB stores the variable in memory but does not print it in the command window. The function is useful when dealing with large matrices.

Arithmetic Functions

Multiplication of the matrix **A** by a scalar:

```
>> A2 = A*3

A2 =

      3     6     9

      12    15    18

      21    24    27
```

Addition of **A** to **A2** (both matrices must be the same dimension):

```
>> A3 = A + A2
```

```
A3 =

      4     8     12

      16    20    24

      28    32    36
```

Multiplication of **A** by **A** (the dimensions must follow the multiplication rules for matrices):

```
>> A4 = A * A

A4 =

      30    36    42

      66    81    96

      102   126   150
```

Element-wise multiplication of the **A** by **A**:

```
>> A5 = A .* A

A5 =

      1     4     9

      16    25    36

      49    64    81
```

Element-wise division and powers can also be calculated (./ and .^).

Transposing matrix **A**:

```
>> B = A′

B =

      1     4     7

      2     5     8

      3     6     9
```

Other MATLAB functions

Absolute Value	**`abs(-5) = 5`**	
Square Root	**`sqrt(4) = 2`**	
Complex Conjugate	**`conj(5.0 + 2.00i) = 5.00 - 2.00i`**	
Exponential (e)	**`exp(1) = 2.7183`**	
Natural Log	**`log(2) = 0.6931`**	
Base-10 Log	**`log10(2) = 0.3010`**	
Finding Phase Angle (in radians)	**`angle(5.0 + 2.0i) = 0.3805`**	
Sine/Arcsine Function	**`sin(pi/2) = 1`**	**`asin(1) = 1.5708`**
Cosine/Arcosine Function	**`cos(pi) = -1`**	**`acos(1) = 0`**
Tangent/Arctangent	**`tan(pi/4) = 1`**	**`atan(1) = 0.7854`**
Finding roots of a polynomial:	$y = 1\, x^2 - 5\, x^2 + 6$	

At the MATLAB prompt, enter the constants in descending order of *x* power

```
>> y = [1, -5, 6]
>> roots(y)
ans =
      3
      2
```

Finding a polynomial from its roots: roots = 1, 4, 8

At the MATLAB prompt, enter the roots

```
>> y = [1, 4, 8]
>> poly(y)
ans =
      1     -13   44    -32
```

The corresponding polynomial is $y = 1\,x^3 - 13\,x^2 + 44\,x - 32$

Plotting Results in MATLAB

MATLAB can plot data against its own indices (location within the matrix) or against data set B. Listed below is an example of the two different types of plots and how to change the line type and color.

```
>> t = 10: -1: 6;
>> y = 2: 2: 10;
```

Plotting a data set *t* vs. its indices

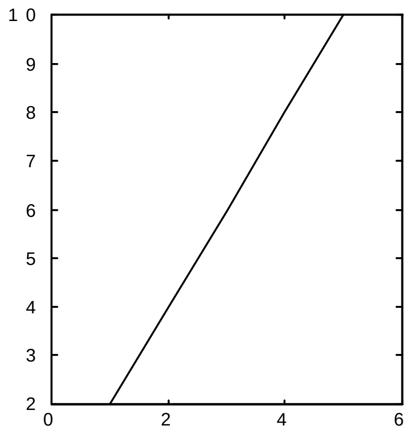
```
>> plot(y)
>> xlabel('Index Number')
>> ylabel('y')
```

MATLAB will return:

Plotting data set *t* vs. data set *y*

```
>> plot(t, y)
>> xlabel('Time')
>> ylabel('y')
```

MATLAB will return:



The labels were add by the `xlabel` and `ylabel` commands shown above. A title may be added:

```
>> title('MATLAB Demo')
```

MATLAB also enables its user to overlay plots.

```
>> plot(t1, y1, t2, y2)
```

This will plot *t*1 vs. *y*1 and *t*2 vs. *y*2 on the same graph. It is also possible to plot the two graphs on one screen without overlaying them. The `subplot(xyn)` command will divide the graphics screen into a grid of separate plots (`x` is the number of rows, `y` is the number columns, and `n` is the desired location of the graph being plotted). An example of the `subplot` function is as follows:

```
>> subplot(121), plot(t1, y1), subplot(122), plot(t2, y2)
```

The `subplot` statement would plot two separate graphs of *t*1 vs. *y*1 and *t*2 vs. *y*2 on the screen.

The line-type, color, or data points may be changed when plotting data.

Commands to change the line-type:

To get Solid Line (default)	`plot(t, y, '-')`
To get Dashed Line	`plot(t, y, '--')`
To get Dotted Line	`plot(t, y, ':')`
To get Dot-Dash Line	`plot(t, y, '-.')`

Commands to change the line to individual data points:

To get Dots	`plot(t, y, '.')`
To get Stars	`plot(t, y, '*')`
To get X's	`plot(t, y, 'x')`
To get Circles	`plot(t, y, 'o')`
To get + Symbols	`plot(t, y, '+')`

Commands to change the color of the lines or data points:

To get Yellow (default)	`plot(t, y, 'y')`
To get Red	`plot(t, y, 'r')`
To get Green	`plot(t, y, 'g')`
To get White	`plot(t, y, 'w')`
To get Blue	`plot(t, y, 'b')`

Command to change the color and line-type/data point type:

To get Blue Dashed Line	`plot(t, y, '--b')`

Any combination of color and line-type or color and data point type can be obtain by the replacing the desired color and type in the above plot statement.

MATLAB also has the ability to do many other types of plots including: bode, logspace, semilogx, semilogy, margin, nichols, and nyquist diagrams.

Creating M-Files

It is often easier to create a m-file containing a sequence of commands instead of typing commands into MATLAB one at a time (unless you enjoy typing so much). That way, if you make a mistake, you can simply reload the command file instead of typing all the commands in again. To do this, create a text file *filename.m*. Type all the commands you would type normally at the MATLAB prompt (>>), and save it. If you wish to include comments in your file, start those lines with a `%`. In MATLAB, you can execute the script (a list of commands) by typing in its name.

For example, if you have a m-file named `sample.m` that contains the MATLAB command `sqrt(4)`, you will get the following results from MATLAB:

```
» sample

ans =

     2
```

Note that you may have to change the working directory of your MATLAB session using the `chdir` command (especially if you are saving your m-files on a floppy disk).

Example: Simulation of a Second Order System

To solve a second order differential equation, we need to make a special kind of m-file called a function. In this example, the function is saved on the computer under the filename `sdof.m`. Then we used MATLAB's integration command `ode23` to call the function repeatedly to solve the differential equation numerically.

Here are the contents of `sdof.m`:

```
function yp = sdof(t, y)

%  Comments:
%  t      - Time (scalar)
%  y      - Solution column-vector
%
%  we want to solve: m x'' + c x' + k x = f
%  let:  y1 = x
%        y2 = x'
%
%  then: m y2' + c y2 + k y1 = f
%
%  in matrix form, this is written:
%        y1' = y2
%        y2' = -k/m y1 - c/m y2 + 1/m sin(w t)

m = 5;
c = 5;
k = 100;

yp = [y(2); -k/m*y(1)-c/m*y(2)] + [0; 1/m*sin(20*t)];
```

We can solve the differential equation using the `sdof.m` file with the following input at the MATLAB prompt:

```
[t, y] = ode23('sdof', 0, 10, [0;.01])

plot(t, y(:,1))
```

You can get more information about the `ode23` command by typing `help ode23` at the MATLAB command prompt.