Introduction to MATLAB

MATLAB (MATrix LABoratory) is an interactive program for scientific and engineering numeric calculation.

**Navigation**

MATLAB uses an interactive environment for performing calculations.

There are three primary windows:

- **command window**: used to enter commands and data
- **graphics window**: used to display plots and graphs
- **edit window**: used to create and edit M-files (ie, programming)

When you initially open MATLAB, you will see the command window:

![Command Window Image]

The command prompt is “>>”. 
If you click “New Script” under the Home tab, you will open the editor window. This is where you will program m-files to execute MATLAB code:

And if you create a plot, MATLAB will automatically open a plot window:
Basic Arithmetic

MATLAB uses a straightforward notation for basic scalar arithmetic. The following table summarizes simple MATLAB notation:

+ addition
- subtraction
* multiplication
/ division
^ exponentiation

All of these work for two scalars, including complex scalars.

Typing a semicolon after your command suppresses output to the screen. Compare:

```
>> 2+2
```

with:

```
>> 2+2;
```

Matrix/Vector Operations

The most straightforward way to initialize a matrix is to type a command of the form:

```
variable = [value1-1 value1-2 value1-3 ... ; value2-1 value2-2 ...]
```

where each value may be rational or complex numbers. Within the square brackets that are used to form vectors and matrices, you can use a semicolon to end a row. For example:

```
>> x = [1 2 3 4; 0 9 3 8]
```

Vectors and scalars are initialized the same way as matrices. It is important to note that MATLAB indexes matrices in the following manner:

(1,1) (1,2) (1,3) (1,4) (2,1) (2,2) (2,3) (2,4)

This means that the first element always has index (1,1), not (0,0).
If \( x \) is already defined as a vector of the form \([\text{val}1 \ \text{val}2 \ \text{val}3 \ \text{val}4...]\) then you can define a new variable as a subset of \( x \) by using the index of the specific value in vector \( x \). For example, if

\[
\begin{align*}
\text{>> } x & = [2 \ 4 \ 1 \ 7]; \\
\text{then:} \\
\text{>> } z & = x(3) \\
z & = 1
\end{align*}
\]

You can specify a value in matrix \( y \) the same way:

\[
\begin{align*}
\text{>> } y & = [1 \ 2 \ 3 \ 4 \ 5; \ 3 \ 4 \ 5 \ 6 \ 7]; \\
\text{>> } z & = y(2,1) \\
z & = 3
\end{align*}
\]

Another useful operator is the colon (\( : \)), which you use to specify a range of numbers (as a vector). For example, try:

\[
\begin{align*}
\text{>> } x & = 1:4 \\
\text{>> } z & = (1:5) \\
z & = 1 \ 2 \ 3 \ 4 \ 5
\end{align*}
\]

You can also specify a range of numbers in a defined vector or matrix using the colon operator. The colon causes MATLAB to step in sequence through the numbers specified. For example,

\[
\begin{align*}
\text{>> } z & = y(1:2,2:3)
\end{align*}
\]
You can optionally give the range indicator a step size as the middle element in a series of colons. For example, try:

```
>> x = 8:-1:5
```

```
>> x = 0:0.25:1.25
```

The special operator ' (prime or apostrophe) denotes the transposition of a matrix. For example, try:

```
>> a = [1 2 3]
```

and then:

```
>> a'
```

**Graphics**

MATLAB supports several commands that allow you to display results of your computations graphically. Graphs are displayed in a graphics window that MATLAB creates when you give one of the plotting commands. The default graphics window starts up with a black background. To change this, before issuing any plotting commands, type the command `whitebg` at the MATLAB >> prompt.

**Plotting Individual Graphs**

The `plot` command is the simplest way of graphing data. If `x` is a vector, `plot(x)` will plot the elements of `x` against their indices. The adjacent values of `x` will be connected by lines. For example, to plot the discrete-time sequence that is a sinusoid of frequency $\pi/6$, you would type:

```
>> n = 0:11;
```

```
>> y = sin((pi/6)*n);
```

```
>> plot(n,y)
```

When `plot` gets two vectors for arguments, it creates a graph with the first argument as the abscissa values, and the second vector as ordinate values. In the example above, `plot` will use the values of `y` for the y-axis, and the values of `n` for the x-axis. If you typed:

```
>> plot(y)
```
MATLAB would use the values of y for the y-axis and their indices for the x-axis. Notice that the first value graphed would have an abscissa value of one, and not zero. This is because MATLAB indexes vector elements beginning with one, not zero.

You can also change the type of line used to connect the points by including a third argument specifying line type. The format for this is plot(x,y,\text{line-type}). The line types available are:

- `'-'` solid line (default)
- `'-.'` line of alternating dots and dashes
- `':'` dotted line
- `'-.'` dashed line

Whichever character you chose to use must be enclosed by single quotes. For example, plot(n,y,':') would create the same graph as above, except that the points would be connected by a dotted line. The default line type is solid. In this case, it is misleading to connect the adjacent values by lines, since this is a graph of a discrete-time sequence. Instead, we should just put a mark to indicate each sample value. We can do this by using a different set of characters in place of the line-type argument. If we use a ',', each sample is marked by a point. Using a '+' marks each sample with a + sign, '*' uses stars, 'o' uses circles, and 'x' uses x's. For example, the following command plots the values of y against their indices, marking each sample with a circle:

```matlab
>> plot(n,y,'o')
```

You can also plot several graphs on the same axis. For example, the following command plots y1 versus x1 and y2 versus x2 on the same axis using different line types for each graph:

```matlab
plot(x1,y1,x2,y2)
```

You can also include a specific line or point type (from the list above) for each graph:

```matlab
plot(x1,y1,'line-type1',x2,y2,'line-type2')
```

You can also create plots with either or both axes changed to log-scale. All of these functions follow the same conventions for arguments and line or point types as plot:

You can use additional MATLAB commands to title your graphs or put text labels on your axes. For example, the following command labels the current graph at the top with the text enclosed in single quotes:

```matlab
>> title('MATLAB Graph #1')
```

Similarly, the following commands label the x- and y-axes:

```matlab
>> xlabel('This is the x-axis')
>> ylabel('This is the y-axis')
```
The axis command is used to control the limits and scaling of the current graph. Typing

\[ \text{[min-x max-x min-y max-y]} = \text{axis} \]

will assign a four-element vector to \( a \) that sets the "minimum ranges" for the axes. The first element is the minimum x-value, the second is the maximum x-value for the current graph. The third and fourth elements are the minimum and maximum y-values, respectively. You can set the values of the axes by calling the axis function with a four-element vector for an argument. You might want to do this, for example, if you were going to plot several sets of data on the same graph and you knew that the range of one set of data was significantly larger than the other.

The elements of the vector you use to set the values of the axes should be your choices for the x- and y-axis limits, in the same order as specified above \( ([x\text{-min } x\text{-max } y\text{-min } y\text{-max}] ) \). So, if you type

\[
\text{>> axis([-10 10 -5 5])}
\]

you will rescale the axis in the graphics window so the x-axis goes at least from -10 to 10, and the y-axis from -5 to 5. The axis command can be stubborn, and may expand your limits to larger limits it finds easier to draw. There is unfortunately little you can do about this.

The hold command will keep the current plot and axes even if you plot another graph. The new graph will just be put on the current axes (as much as fits). Typing hold a second time will toggle the hold off again, so the screen will clear and rescale for the next graph.

**Plotting Multiple Graphs**

You can use the subplot command to split the screen into multiple windows, and then select one of the sub-windows as active for the next graph. The subplot function can divide the graphics window into a maximum of four quadrants; first the window splits horizontally, then it splits vertically. The format of the command is:

\[
\text{>> subplot(xyn)}
\]

In this command, \( x \) is the number of vertical divisions, \( y \) is the number of horizontal divisions, and \( n \) is the window to select for the first plot. Both \( x \) and \( y \) must be less than or equal to two, and \( n \) must be less than or equal to \( x \) times \( y \). For example, subplot(121) will create two full-height, half-width windows for graphs, and select the first, e.g. left, window as active for the first graph. After that, unless you specifically indicate which window is active, MATLAB cycles through them with each successive plot. The order of that cycling is as follows:

```
1 2
-----
3 4
```
Typing subplot with no arguments returns the graphics window to its original, single-window state.

built-in functions

MATLAB has a variety of built-in functions to make it easier for you to construct matrices without having to enumerate all the elements. (The following examples show both vectors and matrices.)

**Ones**
The `ones` function creates a matrix whose elements are all ones. Typing `ones(m,n)` creates an m row by n column matrix of ones. To create a ones matrix that is the same size as an existing matrix, you can use `ones(size(X))`. This does not affect the input argument. For example (this definition of `x` applies to subsequent examples in this section):

```matlab
>> x = [1 2 3 4; 0 9 3 8]
x =
    1 2 3 4
    0 9 3 8

>> y = ones(size(x))
y =
    1 1 1 1
    1 1 1 1
```

**Zeros**
The `zeros` function is similar to the `ones` function. Typing `zeros(m,n)` creates an m-by-n matrix of zeros, and `zeros(size(x))` will create a two-by-four matrix of zeros, if `x` is defined the same way as above.

**Max and min**
The `max` and `min` functions return the largest and smallest values in a vector. For example (this definition of `z` applies to the following series of examples):

```matlab
>> z = [1 2 -9 3 -3 -5];

>> max(z)
ans =
```

If called with a matrix as its argument, max returns a row vector in which each element is the maximum value of each column of the input matrix. The max function can also return a second value: the index of the maximum value in the vector. To get this, assign the result of the call to max to a two element vector instead of just a single variable.

For example:

```matlab
>> [a b] = max(z)
```

```
a = 
3
b = 
4
```

where a is the maximum value of the vector and b is the index of that value. The MATLAB function min is exactly parallel to max:

```matlab
>> min(z)
```

```
ans =
-9
```

**Sum and prod**

*sum* and *prod* are two more useful functions for matrices. If z is a vector, sum(z) is the sum of all the elements of the vector z:

```matlab
>> sum(z)
```

```
ans =
-11
```
For matrices, sum sums the columns. For example:

```matlab
>> w = magic(3);

>> w

8 1 6
3 5 7
4 9 2

>> sum(w)

ans =
15 15 15

>> sum(sum(w))

ans =
45
```

Similarly, `prod(z)` is the product of all the elements of `z`.

```matlab
>> prod(z)

ans =
-810
```

Often, it is useful to define a vector as a subunit of a previously defined vector. To do this, you can use the colon operator. For example, using the `z` defined above,

```matlab
>> z

z =
1 2 -9 3 -3 -5

>> y = z(2:5)
```
\[ y = 2 -9 3 -3 \]

where (2:5) represents the sequence of index values to be taken from the larger vector.

**size**
The size function returns a two-element vector giving the dimensions of the matrix with which it was called. For example:

\[
\begin{align*}
\text{>> } & x = [1 2 3 4; 0 9 3 8] \\
& x = \\
& 1 2 3 4 \\
& 0 9 3 8 \\
\text{>> } & y = \text{size}(x) \\
& y = \\
& 2 4
\end{align*}
\]

You can also define the result to be two separate values (as shown in the max example):

\[
\begin{align*}
\text{>> } & [m n] = \text{size}(x) \\
& m = \\
& 2 \\
& n = \\
& 4
\end{align*}
\]

**length**
The length operator returns the length of a vector. If z is defined as in the above examples,

\[
\begin{align*}
\text{>> } & \text{length}(z) \\
& \text{ans} = \\
& 6
\end{align*}
\]

For matrices, length is the length or the width, whichever is greater, i.e., length(z) is equivalent to max(size(z)).
Element-Wise Operations

You often may want to perform an operation on each element of a vector while doing a computation. For example, you may want to add two vectors by adding all of the corresponding elements. The addition (+) and subtraction (-) operators are defined to work on matrices as well as scalars. For example, if \( x = [1 \ 2 \ 3] \) and \( y = [5 \ 6 \ 2] \), then

\[
>> w = x+y
\]

\[
w = \begin{bmatrix} 6 & 8 & 5 \end{bmatrix}
\]

Multiplying two matrices element by element is a little different. The * symbol is defined as matrix multiplication when used on two matrices. Use .* to specify element-wise multiplication. So, using the \( x \) and \( y \) from above,

\[
>> w = x \cdot y
\]

\[
w = \begin{bmatrix} 5 & 12 & 6 \end{bmatrix}
\]

You can perform exponentiation on a vector similarly. Typing \( x .^2 \) squares each element of \( x \).

\[
>> w = x .^2
\]

\[
w = \begin{bmatrix} 1 & 4 & 9 \end{bmatrix}
\]

Finally, you cannot use / to divide two matrices element-wise, since / and \( \backslash \) are reserved for left and right matrix division." Instead, you must use the ./ function. For example:

\[
>> w = y ./ x
\]

\[
w = \begin{bmatrix} 5.0000 & 3.0000 & 0.6667 \end{bmatrix}
\]

All of these operations work for complex numbers as well (the * operator is no longer required in complex numbers).

The \textit{abs} operator returns the magnitude (absolute value) of its argument. If applied to a vector, it returns a vector of the magnitudes of the elements. For example, if \( x = [2 -4 3i -3i] \):

\[
>> y = \text{abs}(x)
\]
y =
 2 4 5 3

The \texttt{sqrt} function computes the square root of its argument. If its argument is a matrix or vector, it computes the square root of each element. For example:

\begin{verbatim}
>> x
x =
 4 -9 i 2 -2i
>> y = sqrt(x)
y =
 2.0000 0 + 3.0000i 0.7071 + 0.7071i 1.5538 - 0.6436i
\end{verbatim}

MATLAB also includes functions for exponentials and logarithms. The \texttt{exp} operator computes $e$ to the power of its argument. This works element-wise, and on complex numbers. The \texttt{pi} function returns the floating point number nearest the value of pi. So, to generate the complex exponential with a frequency of pi/4, we could type:

\begin{verbatim}
>> n = 0:7;
>> s = exp(i*(pi/4)*n)
s =
Columns 1 through 4
 1.00 0.7071 + 0.7071i 0.00 + 1.0000i -0.7071 + 0.7071i
Columns 5 through 8
-1.0000 + 0.0000i -0.7071 - 0.7071i -0.0000 - 1.0000i 0.7071 - 0.7071i
\end{verbatim}

MATLAB also has natural and base-10 logarithms. The \texttt{log} function calculates natural logs, and \texttt{log10} calculates base-10 logs. Both operate element-wise for vectors. Both are defined for complex values.
MATLAB Help

MATLAB has a good built-in help facility. Typing help with no arguments at the MATLAB >> prompt displays a list of all primary help topics, by directory in the MATLABPATH, along with a short description for each one. Typing help directoryname displays a list of .m files in the directory, along with a brief description of each one.

Typing help function displays the syntax for the function (i.e. what arguments it expects) and a short description of what that function does. If you think you are doing everything right, but MATLAB claims you are in error, try looking at the help for the functions you are using.

A new command, lookfor xyz, searches for the string xyz in the first comment line of help text in all M-files in the MATLABPATH. Because lookfor must look at a lot of files, be ready for the search to take a minute or so.

When you write your own functions, you can also include help information for them. This can be very useful for other people using your function, or for your own use if you haven't used the function for a while. For more information on how to include help information for your own functions, see the section Functions.