MATLAB Cheat Sheet

Basic Commands

`%`	Indicates rest of line is commented out.
`;`	If used at end of command it suppresses output. If used within matrix definitions it indicates the end of a row.
`save filename`	Saves all variables currently in workspace to file `filename.mat`.
`save filename x y z`	Saves x, y, and z to file `filename.mat`.
`save -append filename x`	Appends file `filename.mat` by adding x.
`load filename`	Loads variables from file `filename.mat` to workspace.
`!`	Indicates that following command is meant for the operating system.
`...`	Indicates that command continues on next line.
`help function/command`	Displays information about the function/command.
`clear`	Deletes all variables from current workspace.
`clear all`	Basically same as clear.
`clear x y`	Deletes x and y from current workspace.
`home`	Moves cursor to top of command window.
`clc`	Homes cursor and clears command window.
`close`	Closes current figure window.
`close all`	Closes all open figure windows.
`close(H)`	Closes figure with handle H.
`global x y`	Defines x and y as having global scope.
`keyboard`	When placed in an M-file, stops execution of the file and gives control to the user's keyboard. Type `return` to return control to the M-file or `dbquit` to terminate program.

`A=xlsread(‘data’,...`
`‘sheet1’,‘a3:b7’)`
	Sets A to be a 5-by-2 matrix of the data contained in cells A3 through B7 of sheet `sheet1` of excel file `data.xls`

`Succes=xlswrite(...`
`‘results’,A,‘sheet1’,‘c7’)`
	Writes contents of A to sheet `sheet1` of excel file `results.xls` starting at cell C7. If successful `success`$= 1$.

`path`	Display the current search path for `.m` files
`addpath c:\my_functions`	Adds directory `c:\my_functions` to top of current search path.
`rmpath c:\my_functions`	Removes directory `c:\my_functions` from current search path.
`disp(’random statement’)`	Prints `random statement` in the command window.
`disp(x)`	Prints only the value of x on command window.
`disp([’x=’,num2str(x,5)])`	Displays `x=` and first 5 digits of x on command window. Only works when x is scalar or row vector.

`fprintf(...`
`’The %g is %4.2f.\n’, x,sqrt(x))`
	Displays `The 3 is 1.73.` on command window.

`format short`	Displays numeric values in floating point format with 4 digits after the decimal point.
`format long`	Displays numeric values in floating point format with 15 digits after the decimal point.

Plotting Commands

`figure(H)`	Makes H the current figure. If H does not exist is creates H.

	Note that H must be a positive integer.
`plot(x,y)`	Cartesian plot of x versus y.
`plot(y)`	Plots columns of y versus their index.
`plot(x,y,‘s’)`	Plots x versus y according to rules outlined by s.
`semilogx(x,y)`	Plots $\log(x)$ versus y.
`semilogy(x,y)`	Plots x versus $\log(y)$.
`loglog(x,y)`	Plots $\log(x)$ versus $\log(y)$.
`grid`	Adds grid to current figure.
`title(‘text’)`	Adds title `text` to current figure.
`xlabel(‘text’)`	Adds x-axis label `text` to current figure.
`ylabel(‘text’)`	Adds y-axis label `text` to current figure.
`hold on`	Holds current figure as is so subsequent plotting commands add to existing graph.
`hold off`	Restores hold to default where plots are overwritten by new plots.

Creating Matrices/Special Matrices

`A=[1 2;3 4]`	Defines A as a 2-by-2 matrix where the first row contains the numbers 1, 2 and the second row contains the number 3, 4.
`B=[1:1:10]`	Defines B as a vector of length 10 that contains the numbers 1 through 10.
`A=zeros(n)`	Defines A as an n-by-n matrix of zeros.
`A=zeros(m,n)`	Defines A as an m-by-n matrix of zeros.
`A=ones(n)`	Defines A as an n-by-n matrix of ones.
`A=ones(n,m)`	Defines A as an m-by-n matrix of ones.
`A=eye(n)`	Defines A as an n-by-n identity matrix.
`A=repmat(x,m,n)`	Defines A as an m-by-n matrix in which each element is x.

`linspace(x1, x2, n)`	Generates n points between $x1$ and $x2$.

Matrix Operations

`A*B`	Matrix multiplication. Number of columns of A must equal number of rows of B.
`A^n`	A must be a square matrix. If n is an integer and $n > 1$ than `A^n` is A multiplied with itself n times. Otherwise, `A^n` is the solution to $A^n v_i = l_i v_i$ where l_i is an eigenvalue of A and v_i is the corresponding eigenvector.
`A/B`	This is equivalent to `A*inv(B)` but computed more efficiently.
`A\B`	This is equivalent to `inv(A)*B` but computed more efficiently.

`A.*B,A./B,`
`A.\B,A.^n`
	Element-by-element operations.

`A'`	Returns the transpose of A.
`inv(A)`	Returns the inverse of A.
`length(A)`	Returns the larger of the number of rows and columns of A.
`size(A)`	Returns of vector that contains the dimensions of A.
`size(A,1)`	Returns the number of rows in A.
`reshape(A,m,n)`	Reshapes A into an m-by-n matrix.

`kron(A,B)`	Computes the Kronecker tensor product of A with B.
`A = [A X]`	Concatenates the m-by-n matrix A by adding the m-by-k matrix X as additional columns.
`A = [A; Y]`	Concatenates the m-by-n matrix A by adding the k-by-n vector Y as additional rows.

Data Analysis Commands

`rand(m,n)`	Generates an m-by-n matrix of uniformly distributed random numbers.
`randn(m,n)`	Generates an m-by-n matrix of normally distributed random numbers.
`max(x)`	If x is a vector it returns the largest element of x. If x is a matrix it returns a row vector of the largest element in each column of x.
`min(x)`	Same as `max` but returns the smallest element of x.
`mean(x)`	If x is a vector it returns the mean of the elements of x. If x is a matrix it returns a row vector of the means for each column of x.
`sum(x)`	If x is a vector it returns the sum of the elements of x. If x is a matrix it returns a row vector of the sums for each column of x.
`prod(x)`	Same as `sum` but returns the product of the elements of x.
`std(x)`	If x is a vector it returns the standard deviation of the elements of x. If x is a matrix it returns a row vector of the standard deviations for each column of x.
`var(x)`	Same as `std` but returns the variance of the elements of x.

Conditionals and Loops

```
for i=1:10
  procedure
end
```

Iterates over `procedure` incrementing i from 1 to 10 by 1.

```
while(criteria)
  procedure
end
```

Iterates over `procedure` as long as `criteria` is true.

```
if(criteria 1)
  procedure 1
elseif(criteria 2)
  procedure 2
else
  procedure 3
end
```

If `criteria 1` is true do `procedure 1`, else if `criteria 2` is true do `procedure 2`, else do `procedure 3`.