Earth 468 Matlab Reference Card

by Trever Hines

This document is intended to be a quick reference for frequently used Matlab commands in Earth 468. This is by no means exhaustive and should be considered a work in progress. Students are encouraged to check mathworks.com for complete documentation and stackoverflow.com for solutions to common Matlab questions.

Getting help

`help('func')`: prints help on a function named 'func' in the command window
`doc('func')`: opens online help documentation

Environment

`run('myscript')`: evaluates the script called 'myscript.m'
`who()`: list variables in the environment
`whos()`: detailed list of variables
`clear()`: clear all variables in environment
`clear('a','b', ...)`: clear indicated variables

Input and output

`load('input_file.mat')`: load variables saved in a Matlab file into the environment
`save('output_file.mat')`: saves variables in the environment into a Matlab file
`A=csvread('input_file.csv')`: read comma separated value (csv) file and store as A
`A=csvread('input_file.csv',N)`: read csv file starting at row N
`csvwrite('output_file.csv',x)`: writes array x to a csv file

Data creation

`x=[1,2,4,...]`: define a row vector named x
`x=[1;2;4;...]`: define a column vector named x
`A=[1,2,...;3,4,...;...]`: define a matrix
`x=[a:b]`: create a list counting by one from a to b
`x=[a:c:b]`: counts from a to b by c
`x=linspace(a,b,N)`: list with length N counting from a to b
`s='foo bar'`: define a string (note that single quotes must be used)
`I=eye(N)`: creates an NxN identity matrix
`O=zeros(M,N)`: creates a matrix of zeros
`D=diag(x)`: turns a vector into a diagonal matrix
`d=diag(A)`: returns the diagonal components of a matrix
`A=unifrnd(a,b,M,N)`: creates an M by N array of uniformly distributed random numbers from a to b
`A=normrnd(mu,sigma,M,N)`: creates a array of normally distributed random numbers with mean mu and standard deviation sigma
`A=reshape(x,M,N)`: reshape x into an M by N array. x must contain M*N elements

Variable Information

`length(x)`: returns the length of x
`size(A)`: returns the number of rows and columns of A
`size(A,1)`: returns the number of rows in A
`size(A,2)`: returns the number of column in A
`max(x)` , `min(x)`: returns the maximum / minimum value in x
`mean(x)`, `median(x)`, `mode(x)`: returns the mean / median / mode of x
`std(x)`: returns the standard deviation of x
`var(x)`: returns the variance of x
`cov(A)`: returns the covariance matrix for the columns of A

Data comparisons

a and b can be scalars or arrays. If they are arrays then element-wise comparisons are performed and a boolean array is returned
`a==b`: returns true (1) if a is equal to b, otherwise returns false (0)
`a~=b`: returns true if a is not equal to b
`a>b`: returns true if a is greater than b
`a>=b`: returns true if a is greater than or equal to b

Boolean operations

a and b can be boolean values or arrays of boolean values
`~a`: returns true if a is false
`a&b`: returns true if a and b are both true
`a|b`: returns true if either a or b are true
`all(a)`: returns true if all values in a are true
`any(a)`: returns true if any value in a is true
`find(a)`: returns the indices of a which are true

a and b are often replaced with data comparisons expressions, for example:
`find(x>=2)`: returns the indices of x which are greater than 2

Slicing and extracting data

`x(N)`: returns the N'th element in a vector
`x(M:N)`: returns the N'th to M'th element
`x(N:end)`: returns the N'th to last element in a vector
`A(M,N)`: returns the element from the M'th row and N'th column of A
`A(N,:)`: returns row N
`A(:,N)`: returns column N
`A(M:N,P)`: returns elements in rows M through N of column P

Boolean arrays can also be used to extract elements from arrays, for example:
`x(x>c)`: returns values of x which are greater than c

Lists of integers may be used as well:
`x([2,3,4])`: returns specified elements of x

Mathematical operations

`a+b` , `a-b`: addition / subtraction, if a or b are arrays then the operations are element-wise
`a*b` , `a/b`: scalar multiplication / division
`a^n`: raise the scalar a to the n'th power
`x.*y` , `x./y`: element-wise multiplication / division of x and y
`x.^n`: raise each element of x to the n'th power
`sum(x)`: returns the sum of each element of x
`prod(x)`: returns the product of each element of x
`sqrt(a)`: returns the square root of a, if a is an array then returns element-wise square root of a
`sin(a)`,`cos(a)`,`tan(a)`: trigonometric operations
`log(a)`: base e log of a
`log10(a)`: base 10 log of a
`abs(a)`: returns the absolute value of a

Linear algebra

`det(A)`: determinant of A
`inv(A)`: inverse
`eig(A)`: returns eigenvalues of A
`[U,S,V] = svd(A)`: singular value decomposition
`pinv(A)`: pseudo-inverse
`A'`: transpose
`A*B`: matrix multiplication (number of columns in A must equal number of rows in B)

Plotting

`plot(x,y)`: plots data with x and y coordinates. See help documentation for additional arguments
`hist(x,bins)`: plots a histogram of the data in x using the specified number of bins
`errorbar(x,y,sigma)`: plots data with x and y coordinates and adds error bars with width sigma
`text(x,y,'foo')`: adds text at the specified x and y location
`xlabel('mylabel')`: adds label to the x axis
`ylabel('mylabel')`: adds label to the y axis
`title('mytitle')`: adds title to the plot
`hold on`: continue plotting on current figure
`hold off`: redraw figure on the next plot command
`figure()`: create a new figure
`legend('label1','label2', ... )`: adds a legend to the plot where the lines have the specified label

Programming

function definition: functions must be defined in a separate file where the filename is the same as the function name except with a .m extension. This example demonstrates how to write a function called 'myfunc' which takes 'arg1', 'arg2', ... as arguments and returns 'out'. 'out' must be defined somewhere in the body of the function

```
function out = myfunc(arg1,arg2,...)
  <function body>
end
```

if statements: runs the block of code within the 'if' statement if 'mycondition' is true

```
if mycondition:
  <block of code>
end
```

for loops: runs a block of code for each element in 'mylist'. Each time the block of code is run the iterator, 'i', is replaced with the next element in 'mylist'.

```
for i = mylist
  <block of code operating on i>
end
```

while loops: runs a block of code until 'mycondition' is false

```
while mycondition
  <block of code>
end
```

Additional tips

use Tab to auto-complete commands
use the up/down arrow to scroll between previous commands
use % to indicate comments in scripts
use ; at the end of commands to suppress the output