MATLAB Reference Card

Note: This reference card gives only a short summary. For detailed syntax and behavior, see the MATLAB help.[1]

Getting help

All MATLAB functions have online documentation.
help <command> Help on <command>
doc <command> Detailed documentation on <command> (opens in help browser).

Workspace

who displays a list of variables in the workspace
whos displays a detailed list of variables in the workspace
format sets the default format how MATLAB displays numbers.

format short	5 digit fixed point
format long	15 digit fixed point
format short e	5 digit floating point
format long e	15 digit floating point

clear x clears the variable x
clear erases all variables in the workspace
clc clears the command window
close n closes figure window no. n
close all closes all figure windows
... Three or more periods at the end of a line continue the current command or function call onto the next line. Text on a line after ... is ignored. (Unlike C or Java, in MATLAB a command is normally terminated by a newline character.)

Data Creation

x=[1,2,4,...] define a row vector x
x=[1 2 4 ...] *same.*
x=[1; 2; 5; ...] define a column vector x
a:c the range a..c; equivalent to [a,a+1,...,c-1,c]
a:b:c the range a..c with step size b; equivalent to [a,a+b,a+2*b,..., c-b,c]
linspace(a,b,n) a row vector with n values linearly spaced from a to b (inclusive)
eye(n) the $n \times n$ identity matrix
zeros(n) a $n \times n$ zero matrix
zeros(m,n) a $m \times n$ zero matrix
ones(n) a $n \times n$ all-one matrix
ones(m,n) a $m \times n$ all-one matrix
diag(x) creates a diagonal matrix whose diagonal consists of the entries of the vector x
[X,Y]=meshgrid(x,y) transforms the domain specified by vectors x and y into matrices X and Y that can be used for the evaluation of functions of two variables.

[1] Based on the R Reference Card by Tom Short, tshort@epri-peac.com.
[2] For Matrices, these commands work columnwise.

Slicing and Extracting Data

Indexing vectors

x(1)	1st element
x(n)	nth element
x(end)	last element
x(1:n)	first n elements
x(end-n:end)	last n+1 elements
x([1 2 4])	specific elements (use any row or column vector as index)
x(x>3)	all elements greater than 3
x(x>3&x<5)	all elements between 3 and 5
x(:)	transformed to column vector

Indexing matrices

x(i,j)	element at row i, column j
x(i,:)	row i
x(:,j)	column j
x(1:m,:)	first n rows
x(:,1:n)	first n columns
x(end,end)	The last element in the last row
x(:)	transformed to column vector (column by column)

Variable Information

length(a) the length of the vector x. For matrices length returns the number of rows or columns, whichever is larger.
[x,y]=size(a) the number of rows (x) and columns (y) of the matrix a
size(a,1) the number of rows of a
size(a,2) the number of columns of a
numel(a) the number of elements in a
nnz(a) the number of non-zero elements in a

Data Selection and Manipulation

x' the complex conjugate transpose of x
x.' the non-conjugate transpose of x
[2]**max(x)** the greatest element of x
[2]**min(x)** the smallest element of x
[2]**fliplr(x)** reverses the elements of x from left to right
[2]**flipud(x)** reverses the elements of x from top to bottom
[2]**[a,i]=max(x)** returns in addition the position i of the greatest element
[2]**[a,i]=min(x)** returns in addition the position i of the smallest element
[2]**sort(x)** sorts the elements of x in ascending order
[2]**sortrows(x)** sorts the rows of x in ascending order as a group, according to the first column.
[2]**sortrows(x,c)** as above, but sorted according to column c. If c is negative, the rows are sorted by descending order. If c is a vector, the rows are sorted first by column c(1), then by column c(2), etc.
find(x) returns the indices corresponding to the nonzero entries of x
find(x==a) returns the indices of the positions j such that x[j]==a[j]
unique(x) returns the same values as in a but with no repetitions; the values will also be sorted.
reshape(x,m,n) returns the $m \times n$ matrix whose elements are taken columnwise from x.

Matrix Computations

a+b If a and b are $m \times n$ matrices, this is the standard matrix addition. If a is a matrix and b is a scalar, or vice-versa, the scalar is added to every entry of the matrix.
a-b If a and b are $m \times n$ matrices, this is the standard matrix subtraction. If a is a matrix and b is a scalar, or vice-versa, the scalar is subtracted from every entry of the matrix.
a*b If a is an $k \times m$ matrix and b is an $m \times n$ matrix, this is the standard matrix multiplication, *i.e.*, yielding an $k \times n$ matrix. If a is a matrix and b is a scalar, or vice-versa, every element of the matrix is multiplied by the scalar.
a.*b If a and b are $m \times n$ matrices, this is their *pointwise* multiplication. If either element is a scalar, this is the same as a*b.
a/b If a and b are matrices of appropriate dimensions, this is roughly a*inv(b). If b is a scalar, this divides every entry of a by b.
a./b If a and b are $m \times n$ matrices, this is their *pointwise* division. If a is a scalar, then this divides a by every entry of b. If b is a scalar, then this divides every entry of a by b.
a\b If a is an $n \times n$ matrix and b is an $n \times 1$ column vector, or a matrix with several such columns, then x=a\b is the solution to the equation a*x=b. If a is a scalar, then this divides every entry of b by a.
a.\b If a and b are $m \times n$ matrices, this is their *left pointwise* division. If a is a scalar, then this divides every entry of b by a. If b is a scalar, then this divides b by every entry of a.
a'*b If a and b are $n \times 1$ column vectors, this is their inner product (or scalar product or dot product). (This is not another operator, just a combination of ' (conjugate transpose) and *).
inv(a) The inverse of the $n \times n$ matrix a.
eig(a) is a vector containing the eigenvalues of the $n \times n$ matrix a. [v,d]=eig(a) produces a diagonal matrix d of eigenvalues and a full matrix v whose columns are the corresponding eigenvectors such that a*v=v*d.
rank(a) is the rank, or number of linearly independent rows or columns of the matrix a.

Math

sin,cos,tan,asin,acos,atan,atan2,log,log10,exp,...
These are the standard mathematical functions; they always operate pointwise on their arguments.
sum(x) sum of the elements of x
prod(x) product of the elements of x
diff(x) difference (and sample-wise derivative) of the vector x
cumsum(x) cumulative sum of the elements of x (and sample-wise integral)
cumprod(x) same, for the product
mean(x) mean of the elements of x
median(x) median of the elements of x
log(x, base) computes the logarithm of x with base base
real(x) real part of a complex number
imag(x) imaginary part of a complex number
abs(x) absolute value of x, or complex magnitude if x is a complex number
angle(x) angle in radians of the complex number
conj(x) the complex conjugate of x

Constants

i Imaginary unit `sqrt(1)`
j *same.*
pi $\pi = 3.1415926535897\ldots$
Inf Infinity; results *e.g.* when dividing a non-zero value by zero.
NaN Not a number; results *e.g.* when computing `0/0`.
realmax Largest positive floating point number in MATLAB.
realmin Smallest positive floating point number in MATLAB.
intmax Largest positive integer value in MATLAB.
intmin Smallest integer value in MATLAB.
eps Spacing of floating point numbers. Use it to prevent unwanted behavior due to rounding errors. (See help for details.)
exp(1) The base of the natural logarithm.
Attention: It is possible to assign a value to a predefined constant and thus to override its original value (MATLAB will not warn you if you do so.)

Signal Processing

c=conv(a,b) Convolution; *e.g.*, `c(1)=a(1)*b(1)`
c=xcorr(a,b) Cross-correlation estimates.
fft(x) Fast Fourier Transform of the vector `x`
ifft(x) Inverse Fast Fourier Transform
fftshift(x) swaps the left and right halves of `x` to shift the zero-frequency component to the center of the spectrum.
filter(b,a,x) filters the data in vector `x` with the filter described by vectors `a` and `b`.
[b,a]=butter(n,Wn) designs an n^{th} order lowpass digital Butterworth filter and returns the filter coefficients in the vectors `b` (numerator) and `a` (denominator). The cutoff frequency must be $0.0 < W_n < 1.0$, with 1.0 corresponding to half the sample rate.
downsample(x,n) downsamples the signal `x` by keeping every n^{th} sample starting with the first.
upsample(x,n) upsamples the signal `x` by inserting $n-1$ zeros between input samples.
resample(x,p,q) resamples the signal `x` at `p/q` times the original sample rate.

Communication Toolbox

randint(m,n) generates an $m \times n$ matrix of random binary numbers.
randint(m,n,p) generates an $m \times n$ matrix of random integers between 0 and `p-1`.
pskmod,pskdemod phase shift keying modulation/demodulation
qammod,qamdemod quadrature amplitude modulation/demodulation
rcosine designs a raised or root raised cosine filter
rcosflt filters a signal using raised or root raised cosine filter
awgn add white Gaussian noise to a signal
bi2de converts a binary vector to a decimal value
de2bi converts a nonnegative integer decimal vector to a binary matrix
biterr computes the bit error rate
symerr computes the symbol error rate

Sparse Matrices

Using sparse matrices can result in a significant computational gain if you work with large matrices that have relatively few non-zero entries.

sparse(x) converts a sparse or full matrix to sparse
sparse(m,n) creates an $m \times n$ all-zero sparse matrix
speye(n) creates an $n \times n$ sparse identity matrix
spones(x) creates a matrix with the same sparsity structure as `x`, but with ones in the nonzero positions.

Plotting

plot(x) plot of the values of `x` (on the *y*-axis) versus `0:length(x)-1`
plot(x,y) bivariate plot of `x` (on the *x*-axis) and `y` (on the *y*-axis)
plot(x,y,...) allows you to specify formatting options (cf. `help plot`)
hist(x) histogram of the frequencies of `x`
stem(...) is the same as `plot(...)`, but the data sequence is plotted as discrete "stems" from the x-axis with circles for the data values.
semilogy(...) is the same as `plot(...)`, except a logarithmic (base 10) scale is used for the y-axis.
scatterplot(x) generates a scatter plot of `x`. `x` can be a real or complex vector, or a two-column matrix with real signal in the first column and imaginary signal in the second column.

Figures

Plots are drawn on figure windows. The following commands control the appearance of figures and plots.
h=figure creates a new figure and returns its handle.
figure(h) makes h the current figure, forces it to become visible, and raises it above all other figures on the screen.
figure('name', '...') creates a new figure window with the specified window title
subplot(m,n,k) divides the current figure window into $m \times n$ subfigures and selects the k^{th} for the current plot.
xlabel('...') sets the text for the x-axis. `xlabel`, as well as `ylabel`, `title` etc. accept basic LaTeX-like strings such as `a^2` for a^2 or `\alpha` for α.
ylabel('...') sets the text for the y-axis.
title('...') sets a title for the current plot.
print -depsc2 fig.eps saves the current figure into the file `fig.eps`.

Input and Output

disp(x) displays the contents of variable `x`
fprintf(fmt, vars, ...) Like the C function `printf`
sprintf(fmt, vars, ...) Like `printf`, but returns the string instead of printing it to the screen.
error('...') displays an error message and halts execution. The message can also be a formatting string as for `fprintf`, followed by the corresponding variables, e.g. `error('Warning %d\n', val)`.
warning('...') Like `error`, but execution of the function/script is continued.
waitbar displays progress information.
load foo loads the variables saved in the file `foo.mat` into the current workspace.
load('foo') returns the variables saved in the file `foo.mat` as a structure; this structure will have a field for each variabe in the file. For example, if `foo.mat` contains variables `x` and `y`, and you load the file with `a = load('foo')`, then `x` and `y` will be accessible as `a.x` and `a.y`.
save foo a b ... saves the variables `a`, `b`, etc. in the file `foo.mat`.
save('foo', 'a', 'b') *same.*

Programming

Function definition:

```
[a,b,...]=function(x,y,...)
...
a = ...;
b = ...;
end
```

To call a function in another file, the file must have the same name as the function.
There are two ways of having more than one function in the same file:

```
function foo(...)
  ...
  function bar(...)
    ...
  end
end
```

```
function foo(...)
  ...
end

function bar(...)
  ...
end
```

In the left case, the nested function `bar` inherits all variables that are accessible in the outer function `foo`. In the right case, the function `bar` cannot access variables local to `foo`. In both cases, `bar` can only be called from `foo` and not from a function in another file.

Control structures:

```
if <condition>
  ...
end

for k = x
  ...
end

(x is any vector, can be e.g., 1:n)

while <condition>
  ...
end
```

```
switch <expression>
  case <condition1>
    ...
  case <condition2>
    ...
  ...
  otherwise
    ...
end
```