MATLAB QUICK REFERENCE CARD

Frequently used MATLAB commands – Version v0.4 May 2015

Getting help

All MATLAB functions have online documentation.

- `help command` Help on command
- `helpwin` ...... invokes windowed help utility
- `doc command` . Detailed documentation on command (opens in help browser).

Commands and Functions

Workspace

- `who` .... lists variables in memory
- `whos` .... lists variable names, sizes, and types in memory
- `format` .... invoke output style ..
  - `sets` the default format how MATLAB displays numbers.
  - `short` 5 digit fixed point
  - `long` 15 digit fixed point
- `clear, clear v` clears workspace, variable v
- `close all, close n` closes all figure windows, window n
- `Clc` .... clears command window
- `Diary` .... creates a copy of all commands and most results
- `clock, date` . returns the time, date
- `exit` .... terminates MATLAB
- `quit` .... terminates MATLAB

File & Folder Operations

- `cd` .... change direction
- `copyfile` ..... copy from *pathA* to *pathB*
- `dir` .... output content of a folder
- `exist` .... determines whether variable, function or folder exists
- `open('workspace.mat'), load('workspace.mat')` opens file to command line, additionaly load it into worspace window
- `csvwrite()` .. write to CSV format in current folder

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

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

Data Selection and Manipulation

- `x'` .... the complex conjugate transpose of x
- `x.'` .... the non-conjugate transpose of x
- `max(x)` .... the greatest element of x
- `min(x)` .... the smallest element of x
- `fliplr(x)` ... reverses the elements of x from left to right
- `flipud(x)` ... reverses the elements of x from top to bottom
  - `[a,i]=max(x)` returns in addition the position i of the greatest element
  - `[a,i]=min(x)` returns in addition the position i of the smallest element
- `sort(x)` sorts the elements of x in ascending order
- `sortrows(x)` . sorts the rows of x in ascending order as a group, according to the first column.
- `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 $m \times nmatrix$ whose elements are taken columnwise from x.

Math

Basic Math Functions

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

⇒ `other functions: sin, cos, tan, asin, acos, atan, atan2, log, log10, exp, ..`

Basic Math Operations

`+`	addition
`-`	subtraction
`*`	multiplication
`.*`	array multiplication
`/`	division
`./`	array division
`^`	exponential
`.^`	array exponential

Special Characters

`[]`	forms matrices
`()`	used in statements to group operations
`.`	decimal point
`,!`	separates subscripts or matrix elements
`;`	separates rows in a matrix definition or suppresses output
`:`	indicates all rows or all columns
`=`	assignment operator (not equality)
`%`	indicates a comment
`%%`	cell divider

Special Variables & Constants

`Inf`	Infinity; results e.g. when dividing a non-zero value by zero.
`NaN`	Not a number; results e.g. when computing 0/0.
`ans`	most recent temporary answer
`eps`	Spacing of floating point numbers. Use it to prevent unwanted behavior due to rounding errors.

⇒ `default:` $2.2204e^{-16}$

`exp(1)`	The base of the natural logarithm.
`flops`	count of floating point operations
`i`	Imaginary unit $sqrt(1)$
`j`	same.
`pi`	the math pi (3.1415e)
`realmin, realmax`	smallest, largest real number MATLAB can represent
`intmin, intmax`	returns smallest, largest possible integer used in MATLAB

Relational and Logical Operators

`<`	less than
`<==`	less than or equal to
`>`	greater than
`>==`	greater than or equal to
`==`	equal to
`~=`	not equal to
`&`	and
`!`	or
`~`	not

Vectors & Matrices

Creating Vectors

`linspace(a,b,n)`	a row vector with n values linearly spaced from a to b (inclusive)
`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 $[a, a+b, a+2*b, ..., c-b, c]$

Creating Matrices

`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.

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)

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 $k \times m$ 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 $x = a\backslash b$ is the solution to the equation $a * x = b$. If a is a scalar, then this divides every entry of b by a.
$a.\backslash b$	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.

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.

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 zerofrequency 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 nth order lowpass digital Butterworth filter and returns the filter coefficients in the vectors b (numerator) and a (denominator). The cutoff frequency must be $0.0 < Wn < 1.0$, with 1.0 corresponding to half the sample rate.
- `downsample(x,n)` downsamples the signal x by keeping every nth 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
- `biterr`....... computes the bit error rate

`symerr` .... computes the symbol error rate

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

`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 *kth* 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 $a\hat{}2$ for a^2 or $\backslash alpha$ 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

`input('prompt', fmt)`
shows prompt for user input

⇒ `assigns 'string' to variable` x `(quotation marks matters): x=input('foo bar: ')`

⇒ `option 's' interprets all input as character string, eg. numbers.: x=input('foo bar: ', s)`

`disp(x)` ...... displays the contents of variable x

`fprintf(fmt, vars, ..)`
Like the C function *printf*

`isnumeric(), ischar()`
tests whether content of x is numeric or a character textstring (boolean logic).

`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;

⇒ `a = load('foo'): if` *foo.mat* `contains variables` x `and` y`, and you load the file like this, then` x `and` y `will be accessible as` $a.x$ `and` $a.y$.

`save foo a b` *or* `save('foo', 'a', 'b')`
saves the variables a, b, etc. in the file *foo.mat*.

String Conversions

`func2str`	Constructs a function name string from a function handle
`str2func`	Constructs a function handle from a function name string
`int2str`	Integer to string conversion
`mat2str`	Convert a matrix into a string
`num2str`	Number to string conversion
`sprintf`	Write formatted data to a string
`sscanf`	Read string under format control
`str2double`	Convert string to double-precision value
`str2mat`	String to matrix conversion
`str2num`	String to number conversion
`bin2dec`	Binary to decimal number conversion
`dec2bin`	Nonnegative integer decimal to binary number conversion
`dec2hex`	Decimal to hexadecimal number conversion
`hex2dec`	Hexadecimal to decimal number conversion
`hex2num`	Hexadecimal to double number conversion

Conditional Statements

```
if expression
    statements
elsif expression
    statements
else expression
    statements
end

switch switch_expression
    statements
case case_expression
    statements
case case_expression
    statements
otherwise
    statements
end

for k = vectorOrColumnList
    statements
end

while logicalExpression
    statements
end
```

github.com/emzap79/QRCs

emzap79@gmail.com