MATLAB Cheat Sheet for Data Science - London School of Economics

Technical Report · May 2017

DOI: 10.13140/RG.2.2.16859.77607

2 authors:

Ali Habibnia
Virginia Polytechnic Institute and State University
11 PUBLICATIONS **6** CITATIONS
SEE PROFILE

Eghbal Rahimikia
The University of Manchester
3 PUBLICATIONS **2** CITATIONS
SEE PROFILE

MATLAB Cheat Sheet for Data Science

THE LONDON SCHOOL OF ECONOMICS AND POLITICAL SCIENCE

Ali Habibnia Eghbal Rahimikia

Last update: May, 2017
MATLAB r2017a

Preliminaries

`fuc` : This function or command requires a toolbox to execute.

Ways to get help

`doc`	Display documentation.
`doc command`	Display documentation for function.
`help`	Display documentation in command window.
`help command`	Display help text in command window.
`lookfor (X)`	Search all M-files for `X`.
`docsearch (X)`	Search documentation for `X`.
`demo`	Access demonstration examples.
`which command`	Locate functions.

File extensions

`.m`	A MATLAB script, function, or class.
`.mat`	A MATLAB data, stores workspace.
`.fig`	MATLAB figure or GUI template.
`.p`	MATLAB protected function file.
`.mlx`	MATLAB live script.
`.mex`	MATLAB executable.
`.mlapp`	MATLAB App Designer template.
`.mdl .slx`	Simulink model.
`.mdlp .slxp`	Simulink protected model.
`.mlappinstall`	MATLAB app installer file.
`.mltbx`	MATLAB toolbox file.

Common data types

`single`	Single precision numerical data (32 bits).
`double`	Double precision numerical data (64 bits).
`char`	Character array.
`string`	String array.
`logical`	True (1) or false (0).
`struct`	Structure array.
`cell`	Cell array.
`map container`	Map values to unique keys (dictionary).

Data import/export

`xlsread/xlswrite`	Read/write Excel spreadsheet.
`load/save`	Load/save MATLAB variables.
`load/save -ascii`	Load/save text files (.txt, .csv).
`dlmread/dlmwrite`	Read/write ASCII-delimited file.
`readtable/writetable`	Create/write table from file.
`fscanf/fprintf`	Read/write data from/to text file.
`textscan`	Read formatted data from text file.
`fgetl`	Read line from file, removing newline characters.
`fgets`	Read line from file, keeping newline characters.
`fread/fwrite`	Read/write from/to binary file.
`fopen/fclose`	Open/close file.
`importdata`	Load data from file.
`readall`	Read data from data-store.
`imread/imwrite`	Read/write image file.
`save filename`	Save all variables to .mat file.
`save filename x,y`	Save `x`,`y` variables to .mat file.
`load filename`	Load all variables from .mat file.
`webread/webwrite (URL)`	Read/write content from/to `URL`.
`websave (URL)`	Save data from `URL` to file.

Basic commands

`clc`	Clear command window.
`clear`	Clear workspace.
`clear (X)`	Clear (`X`) from memory.
`close (X)`	Close figure (`X`).
`close all`	Close all figures.
`...`	Continue entering statement.
`clf`	Clear current figure.
`whos (X)`	Size, bytes, class, and attributes of (`X`).
`ver`	List MATLAB version and toolboxes.
`dir`	List current folder contents.
`tic/toc`	Start/stop stopwatch timer.
`beep`	Produce system beep sound.
`ans`	Last answer.
`pwd`	Current directory.
`path`	View/change search directory.
`pathtool`	Open set path window.
`mkdir`	Make new directory.
`cd`	Change current directory.
`what`	List of MATLAB files in folder.
`which`	Find directory of functions.
`lasterr`	Last error message.
`lastwarn`	Last warning message.
`rehash`	Refresh caches.
`home`	Send cursor home.
`exit`	Close MATLAB.

Create basic variables

`x=5`	Define variable `x` to be 5.
`x=nan`	Define variable `x` to be Not-a-Number.
`j:k`	Row vector from `j` to `k` (step size: 1).
`j:i:k`	Row vector from `j` to `k` (step size: `i`).
`linspace(a,b,n)`	`n` numbers linearly spaced between `a` and `b`.
`logspace(a,b,n)`	`n` numbers logarithmically spaced between `a` and `b`.
`NaN(a,b)`	`a` × `b` matrix of NaN values.
`ones(a,b)`	`a` × `b` matrix of 1 values.
`zeros(a,b)`	`a` × `b` matrix of 0 values.
`eye(a)`	Identity matrix of size `a`.
`sparse(a,b)`	`a` × `b` sparse matrix.
`rand(a,b)`	Uniform `a` × `b` random numbers in [0,1).
`randi(imax,a,b)`	Uniform `a` × `b` random integers in [1,imax].
`randn(a,b)`	Gaussian `a` × `b` random numbers.
`randperm(a)`	Integer random permutation in [1,a].
`diag(x)`	Square matrix (vector `x`: diagonal elements).

Basic math functions

`abs(x)`	Absolute value of `x`.
`sqrt(x)`	Square root of `x`.
`sign(x)`	Sign of `x`.
`round(x)`	Round of `x`.
`ceil(x)`	Round `x` toward positive infinity.
`fix(x)`	Round `x` toward zero.
`floor(x)`	Round `x` toward negative infinity.
`complex(a,b)`	Complex array ($z = a + bi$).
`real(x)`	Real part of complex number.
`image(x)`	Imaginary part of complex number.
`conj(x)`	Complex conjugate of `x`.
`log(x)`	Natural logarithm of `x`.
`log10(x)`	Common logarithm of `x`.
`exp(x)`	Exponential of `x` (e^x).
`rem(a,b)`	Remainder after division of `a` by `b`.
`mod(a,b)`	Remainder after division of `a` by `b` (modulo operation).
`lcm(a,b)`	Least common multiples of `a` and `b`.
`gcd(a,b)`	Greatest common multiples of `a` and `b`.
`nthroot(a,n)`	Real `n`-th root of `a`.

Trigonometric functions

`#`: sin, cos, tan, sec, or cot.	sine, cosine, tangent, secant, or cotangent.
`#/#d(x)`	`#` of `x` in radians/degrees.
`#h(x)`	Hyperbolic `#` of `x`.
`a#/a#d(x)`	Inverse `#` of `x` in radians/degrees.
`a#h(x)`	Inverse hyperbolic `#` of `x`.
`atan2/atan2d(x)`	Four-quadrant inverse tan of `x` in radians/degrees.
`hypot(x)`	Square root of sum of squares of `x`.
`deg2rad(x)`	Convert `x` from degrees to radians.
`rad2deg(x)`	Convert `x` from radians to degrees.

Linear algebra

`x=[1,2,3]`	1×3 vector (double array).
`x=[1;2;3]`	3×1 vector.
`x=[1,2;3,4]`	2×2 matrix (double array).
`x=[1,2;3,4;5,6]`	3×2 matrix.
`x={1,'st'}`	1×2 cell array.
`sx.x1=[1,2,3]`	1×3 vector stored in `sx` structure array.
`sx.x2={1,'st'}`	1×2 cell stored in `sx` structure array.
`x*y`	Matrix multiplication.
`x+y`	Element by element addition.
`x-y`	Element by element subtraction.
`x.*y`	Element by element multiplication.
`x./y`	Element by element division.
`A^n`	Normal matrix power of `A`.
`A.^n`	Element-wise power of `A`.
`A'`	Transpose of `A`.
`inv(A)`	Inverse.
`size(A)`	Size (rows and columns).
`numel(A)`	Number of elements.
`min(A)`	Smallest elements.
`cummin(A)`	Cumulative minimum of array elements.
`max(A)`	Largest elements.
`cummax(A)`	Cumulative maximum of array elements.
`sum(A)`	Sum of array elements.
`cumsum(A)`	Cumulative sum of array elements.
`mean(A)`	Average of elements.
`median(A)`	Median of elements.
`mode(A)`	Mode of elements.
`prod(A)`	Product of array elements.
`cumprod(A)`	Cumulative product of array elements.
`diff(x,k)`	Successive `k`-differences of `x`.
`std(A)`	Standard deviation.
`var(A)`	Variance.
`cov(A)`	Covariance.
`corrcoef(A)`	Correlation coefficients (columns: random variables, rows: observations).
`eig(A)`	Eigenvalues and eigenvectors.
`svd(A)`	Singular values.
`norm(A)`	Norms.
`sort(A)`	Sorts vector from smallest to largest.
`sortrows(A)`	Sorts rows of `A` in ascending order.
`rank(A)`	Rank.
`chol(A)`	Cholesky factorization of matrix.
`det(A)`	Determinant of square matrix.
`factor(A)`	Prime factors.
`perm(A)`	Permutations of the elements.

Accessing/assignment elements

If x is a vector:

`x(i)`	Element `i`-th of `x` ($i, j \geq 1$).
`x(i:j)`	Elements from the `i`-th to the `j`-th of `x`.
`x(i:end)`	Elements from the `i`-th to the last one of `x`.
`x(:)`	All the elements of `x`.
`x([i,j])`	`i`-th and `j`-th elements.

If A is a matrix:

`A(i,j)`	* Element `i`,`j` of `A` ($i, j, l, m \geq 1$).
`A(i)`	* Element `i`-th of `A`.
`A(i:j,l:m)`	* Elements in rows from `i` to `j` which are in columns from `l` to `m`.
`A([a,b],[c,d])`	* `a`-th and `b`-th elements in rows and `c`-th and `d`-th elements in columns.
`A(:,i)`	* `i`-th column elements.
`A(:,[a,b])`	`a`-th and `b`-th columns elements.
`A(i,:)`	* `i`-th row elements.
`A([i,j],:)`	* `i`-th and `j`-th rows elements.
`A>i`	Logical array of elements higher than `i`.
`find(A>i)`	Indices of elements higher than `i`.
`find(A==i)`	Indices of elements equal to `i`.
`diag(A)`	Elements in the principal diagonal of `A`.
`tril(A)`	Lower triangular part of `A`.
`triu(A)`	Upper triangular part of `A`.
`A(i,j)=a`	Replace element `i`,`j` of `A` with `a`.
`A(:,i)=[a;b]`	Replace `i`-th column of `A` with `[a;b]`.
`A(i,:)=[]`	* Delete `i`-th row of `A`.
`A([i,j],:)=[]`	* Delete `i`-th and `j`-th rows of `A`.
`A(A>m)=v`	Replace all elements over `m` with `v`.
`A(A==m)=v`	Replace all elements is equal to `m` with `v`.
`arrayfun(func,A)`	Apply a function to each element of `A`.
`bsxfun(func,A,B)`	Apply an element-wise binary operation specified by `func` to `A` and `B`.

If A is a cell:

	Above matrix operations which are marked with asterisk(*). Output is cell array of elements.
`A{i,j}`	Element `i`,`j` of `A` ($i, j, l, m \geq 1$).
`A(i,j)={a}`	Replace element `i`,`j` of `A` with cell `{a}`.
`A(:,i)={a;b}`	Replace `i`-th column of `A` with cell `{a;b}`.
`cellfun(func,A)`	Apply a function to each cell in `A`.

Character and string

st: string.
char: character.

`x='text'`	Define variable `x` to be `'text'` (char).
`x(i)`	i-th part of char.
`x(i:j)`	i-th to j-th part of char.
`x=string('text')`	Define variable `x` to be `'text'` (str).
`x={'s1','s2'}`	Define more than one char.
`x=[string('st1')...` `,string('st2')]`	Define more than one str.
`x{i,j}`	Element i,j of `x` (char).
`x(i,j)`	Element i,j of `x` (str).
`x{i}(j)`	i-th part of j-th char/str.
`x{i:m}(j)`	i to m-th part of j-th char/str.
`strcat(x1,x2)`	Concatenate characters/strings.
`char(x)`	Create character from numeric array.
`strfind(x1,ptrn)`	Search `x1` char/str for `ptrn` pattern.
`strjoin(x)`	Construct char array using `x` char/str elements.
`lower(x)`	Convert char/str to lowercase.
`upper(x)`	Convert char/str to uppercase.
`strcmp(x1,x2)`	Compare char/str of `x1` and `x2`.
`strcmpi(x1,x2)`	Compare char/str of `x1` and `x2` (case insensitive).
`split(x,dl)`	Split strings in string array (`x`) at `dl` delimiters.
`strsplit(x,dl)`	Split `x` char/str at `dl` delimiters.
`sprintf('fmt',x1,x2)`	Format data based on `fmt` structure.
`strvcat(x1,x2)`	Vertically concatenate `x1` and `x2` (ignore spaces).

Regular expression

`regexp/regexpi(str,exp)`	Search `exp` pattern in `str` char/ str (case sensitive/insensitive).
`regexprep(str,exp,rpc)`	Replace `str` which matches `exp` with the `rpc`.
`regexptranslate(type,str)`	Translate `str` to regular expression by `type` as type of translation.

'IS*' functions

Return true where:

`isnan(X)`	Elements of `X` which are NaN.
`isnumeric(X)`	Elements of `X` which are numeric.
`isinf(X)`	Elements of `X` which are infinite.
`isinteger(X)`	Elements of `X` which are integer.
`isfloat(X)`	Elements of `X` which are floating-point.
`isbetween(X,a,b)`	Elements of `X` which are between `a` and `b` (date and time).
`ismember(X,B)`	Elements of `X` which are found in `B`.
`ismissing(X,B)`	Elements of `X` which are missing.

Return true if:

`isvector(X)`	`X` is a vector.
`ismatrix(X)`	`X` is a logical array.
`isstring(X)`	`X` is a string (char).
`iscell(X)`	`X` is a cell array.
`iscellstr(X)`	`X` is a cell array of strings.
`isstruct(X)`	`X` is a structure.
`istable(X)`	`X` is a table.
`islogical(X)`	`X` is a logical array.
`isscalar(X)`	`X` is a scalar (size=[1,1]).
`isreal(X)`	There isn't imaginary value in `X`.
`isrow(X)`	`X` is a row vector.
`iscolumn(X)`	`X` is a column vector.
`isdiag(X)`	`X` is a lower diagonal matrix.
`istril(X)`	`X` is a lower triangular matrix.
`istriu(X)`	`X` is a upper triangular matrix.
`isdir(X)`	`X` is directory (folder).

'IS*' functions...

`isequal(X,B)`	`X` is equal to `B`.
`isequaln(X,B)`	`X` is equal to `B` (NaN values are equal).
`issorted(X)`	`X` elements are in ascending order.
`isvarname(X)`	`X` is a valid MATLAB variable name.

Convert functions

`num2str(x)`	Convert numeric array (`x`) to char array.
`num2cell(x)`	Convert numeric array (`x`) to cell array.
`num2int(x)`	Convert numeric array (`x`) to signed integer.
`num2hex(x)`	Convert numeric array (`x`) to IEEE hexadecimal string.
`str2num(x)`	Convert char array (`x`) to numeric array.
`str2mat(x)`	Convert char/str array (`x`) to matrix.
`str2double(x)`	Convert char/str array (`x`) to double precision.
`str2func(x)`	Convert char array (`x`) to function handle.
`cell2mat(x)`	Convert cell array (`x`) to matrix.
`cell2table(x)`	Convert cell array (`x`) to table.
`cell2struct(x)`	Convert cell array (`x`) to structure array.
`cellstr(x)`	Convert array `x` to cell array.
`mat2str(x)`	Convert matrix (`x`) to char array.
`mat2cell(x)`	Convert matrix (`x`) to cell array.
`table2cell(x)`	Convert table (`x`) to cell array.
`table2array(x)`	Convert table (`x`) to homogeneous array.
`table2struct(x)`	Convert table (`x`) to structure array.
`struct2cell(x)`	Convert structure array (`x`) to cell array.
`struct2table(x)`	Convert structure array (`x`) to table array.
`int2str(x)`	Convert integer (`x`) to char array.
`datenum(x)`	Convert date and time to a number.
`datestr(x)`	Convert date and time to string.

Programming

Script vs. Function vs. Live script

Script M-files: Contain a list of commands that MATLAB simply executes in order. They are useful to batch simple sequences of commonly used commands together.

Function M-files: Can be executed by specifying some inputs and return some desired outputs.

Live scripts: Contain MATLAB codes, embedded outputs, formated texts, equations, and images together in a single environment.

*** Add comment:** To put a comment within a line, type `%` followed by the comment in MATLAB command window, MATLAB script, or live script enviroment.

```
% This is a comment line.
x=2;  %This is a comment.
y=3;
```

User-defined functions

Function structure: `in1` and `in2` are function inputs and `out1` and `out2` are function outputs.

```
function [out1,out2] = fun_name(in1,in2)
...
end
```

Anonymous function structure: `@` operator creates a function handle.

```
f = @(x)(x.^2+exp(x))
% i.e. f(2) returns 11.3891.
```

Return: `return` forces MATLAB to return control to the invoking function before reaching to the end of that function.

Flow control

If statement: An `if` statement can be followed by an (or more) optional `elseif` and an `else` statement, which is useful to test various condition.

```
if (Condition_1)
    MATLAB Commands
elseif (Condition_2)
    MATLAB Commands
else
    MATLAB Commands
end
```

Switch statement: Evaluate a statement and selection one of the cases based on this evaluation.

```
switch (statement)
    case (value1)
        MATLAB Commands
    case (value2)
        MATLAB Commands
end
```

While loop statement: Repeat the commands while `condition` holds.

```
while (Condition)
    MATLAB Commands
end
```

For loop statement: Loop from `a` to `b` in steps of `s` in the variable `i`.

```
for i = a:s:b
    MATLAB Commands
end
```

Break: `break` terminates the execution of for or while loop. Code lines after the `break` do not execute. In nested loops, `break` exits from the loop in which it mentions.

Continue: `continue` passes control to the next iteration of for or while loop. In nested loops, `continue` passes the iteration in which it mentions.

Errors

Common errors

Error using *: inner matrix dimensions must agree.
The `*` operator performs matrix multiplication, where an `NxM` matrix is multiplied by an `MxP` matrix, resulting in an `NxP` matrix. Use `.*` instead of `*` to perform the element-wise multiplication.

Index exceeds matrix dimensions.
This error arises when you try to reference an element that doesn't exist. Some useful functions to check sizes and number of elements are `numel()`, `size()`, and `length()`.

The expression to the left of the equals sign is not a valid target for an assignment.
This error message arises because of misuse of the `=` and `==` operators. The `=` operator does an assignment, and the `==` operator does a logical test for equality.

Subscripted assignment dimension mismatch.
This error message arises because of an attempt to assign a vector or matrix into a compartment that it does not fit in.

Matrix dimensions must agree.
This error message arises when you try to do operations on matrices with different dimensions. For example `A+B` when `A` is 2×2 and `B` is 2×3.

Subscript indices must either be real positive integers or logicals.
This error message arises because of indexing problem. For example `A(1)` is the first element in a matrix not `A(0)` (like other programming languages).

Handling errors

Try, catch statement: `try` a statement, if it returns an error, `catch` it and do another statement.

```
try
    statements
catch expression
    statements
end
```

`error('msg')`	Display message and abort function.
`warning('msg')`	Display warning message.
`assert('msg')`	Throw error if condition is false.
`st=MException(ID...` `,txt)`	Capture information of a specific error and save it in the `st` object.

Parallel computing (CPU & GPU)

CPU:

`parpool(size)`	Create a new parallel pool (`size` is number of CPU workers).
`gcp`	Return the current parallel pool.
`ticBytes/...`	
`tocBytes(gcp)`	Start/stop calculation of the bytes transferred to the workers.
`batch('scr')`	Run a script or function on a worker.
`gather(A)`	Transfer distributed array to local workspace.

parfor: Replace `for` with `parfor` to execute code on CPU workers or cores without any guaranteed order.

```
parfor i = a:s:b
    MATLAB Commands
end
```

spmd: Execute code in parallel on workers of a pool.

```
spmd
    statements
spmd
```

parfeval: Directly execute a defined function on a specified worker.

```
p=gcp(); % Return current MATLAB pool.
f=parfeval(p,@sum,4,3); % Parallel execution of 4+3.
```

'distributed': Partition listed functions out among the workers in a pool: `zeros(5,5,'distributed')`, `ones`, `false`, `true`, `NaN`, `inf`, `eye`, `rand`, `randi`, and `randn`.
'codistributed': Access the arrays distributed among the workers in a pool: `zeros(5,5,'codistributed')`, etc.

GPU:

`gpuDevice(idx)`	Select GPU device specified by `idx`.
`gpuArray(x)`	Copy `x` array to GPU.
`arrayfun(func,A)`	Apply function to elements of `A` on GPU.
`bsxfun(func,A,B)`	Apply an element-wise binary operation specified by `func` to `A` and `B` on GPU.
`gather(A)`	Transfer gpuArray to local workspace.
`existsOnGPU(x)`	Determine if `x` is stored in GPU.

A basic calculation on GPU:

```
W=rand(5,'single'); % Basic random numbers.
GD=gpuArray(W); % Send W to GPU.
GO=GD.*GD; % Execute multiplication on GPU.
```

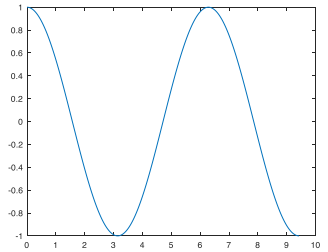
Plotting & Figures

`figure`	Open up a new figure window.
`axis normal`	Default axis limits and scaling behavior.
`axis tight`	Force axis to be equaled to data range.
`axis equal`	Force axis to be scaled equally.
`axis square`	Axis lines with equal lengths.
`axis fill`	Lengths of each axis line fill the rectangle.
`title('Title')`	Add a title at the top of the plot.
`xlabel('lbl')`	Label the x axis as `lbl`.
`ylabel('lbl')`	Label the y axis as `lbl`.
`zlabel('lbl')`	Label the z axis as `lbl`.
`legend('v','w')`	Add label to `v` and `w` curves.
`grid on/off`	Include/Omit a grid in the plot.
`box on/off`	Display/hide the box outline around axes.
`datetick('x',fm)`	Date formatted tick labels (`fm` is format).
`xtickformat(fm)`	X-axis label format.
`ytickformat(fm)`	Y-axis label format.
`xlim([min,max])`	X-axis limits from `min` to `max`.
`ylim([min,max])`	Y-axis limits from `min` to `max`.
`zlim([min,max])`	Z-axis limits from `min` to `max`.
`hold on/off`	Allow/prevent plotting on the same graph.
`text(x,y,text)`	Add `text` to a point (`x` and `y` are scalars in data units).

`fn` is a function:

`fplot(fn,rn)`	Plot a 2-D plot using `fn` over `rn` range.
`fmesh(fn,rn)`	Plot a 3-D mesh using `fn` over `rn` range.
`fsurf(fn,rn)`	Plot a 3-D surface using `fn` over `rn` range.
`fcontour(fn,rn)`	Plot contour using a function (`fn`) over `rn` range.

`plot(x,y)` Plot `y` versus `x` (have same length).



`plot(y)` Plot `y`, with 1,2,3,... as the `x` axis.
`plot(x,f(x))` If `f` is a function, plot the points.

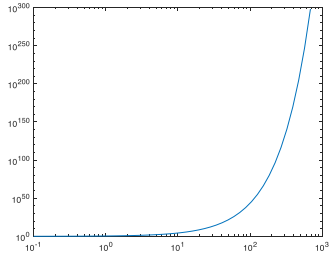
Input arguments:

Line styles:	{ `-` / `:` / `-.` / `--` }.
Markers:	`o` : Circle / `+` : Plus sign / `*` : Asterisk / `.` : Point / `x` : Cross / `s` : Square / `d` : Diamond / `p` : Pentagram / `h` : Hexagram / `^` : Upward triangle.
Colors:	`y` : Yellow / `m` : Magenta / `c` : Cyan / `r` : Red / `g` : Green / `b` : Blue / `w` : White / `k` : Black.

Name-value pair arguments:

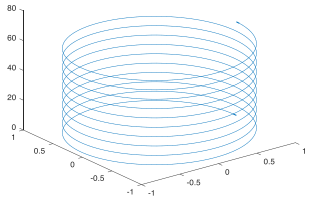
`Color`	Line color.
`LineStyle`	Line style.
`LineWidth`	Line width.
`Marker`	Marker symbol.
`MarkerIndices`	Indices of marker data points.
`MarkerEdgeColor`	Marker outline color.
`MarkerFaceColor`	Marker fill color.
`MarkerSize`	Size of marker.

`loglog(x,y)` Logarithmic x and y axes.

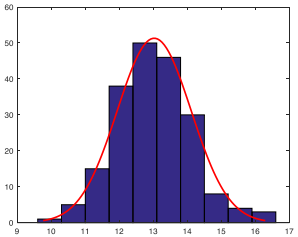


`semilogx(x,y)` Logarithmic x axis.
`semilogy(x,y)` Logarithmic y axis.

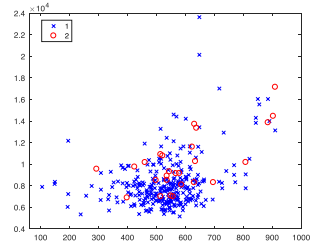
`plot3(x,y,z)` Three-dimensional analogue of plot.



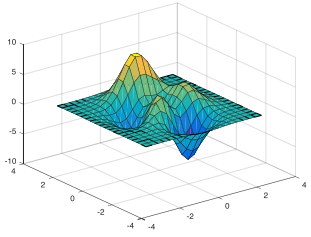
`histfit(y)` Histogram plot with distribution fit.



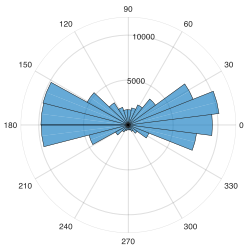
`gscatter(x,y,group)` 2-D scatter plot of `x` and `y` by `group`.



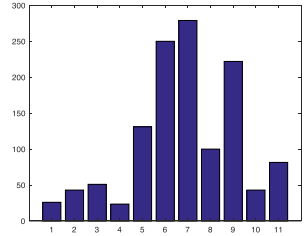
`surf(x,y,z)` 3-D shaded surface plot.



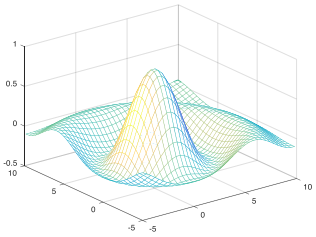
`polarhistogram(y)` Histogram plot (polar coordinates).



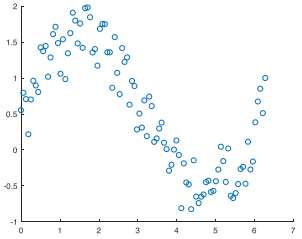
`bar(y)` Bar plot.



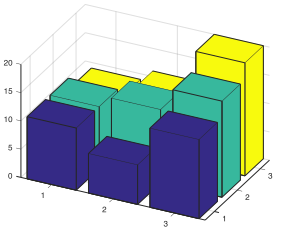
`mesh(x,y,z)` 3-D mesh surface plot.



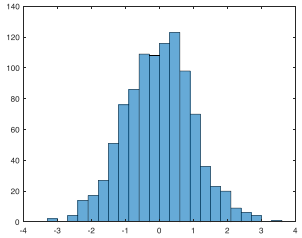
`scatter(x,y)` 2-D scatter plot by `x` and `y`.



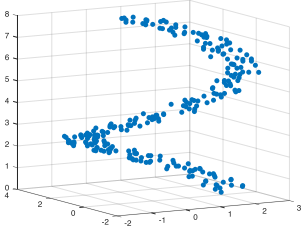
`bar3(y)` 3-D bar plot.



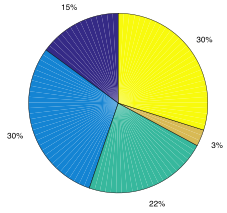
`histogram(y)` Histogram plot.



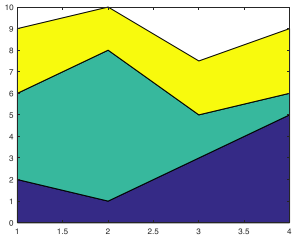
`scatter(x,y,z)` 3-D scatter plot by `x`, `y`, and `z`.



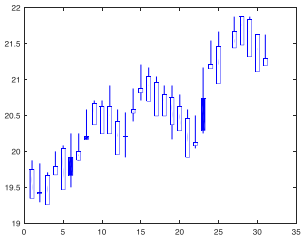
`pie(y)/pie3(y)` 2-D/3-D pie plot.



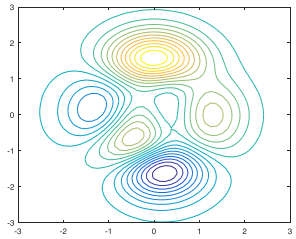
`area(y)` 2-D area plot.



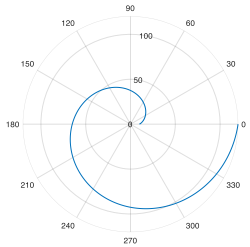
`candle(y)` Candlestick chart.



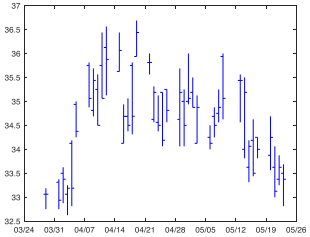
`contour(y)` Contour plot of matrix `y`.



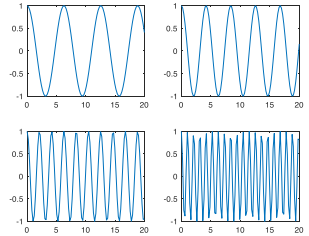
`polarplot(theta,rho)` Polar plot (`theta`: angle, `rho`: radius).



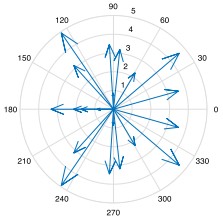
`highlow(h,l,o,c)` Plot `h` (high), `l` (low), `o` (open), and `c` (close) prices of an asset.



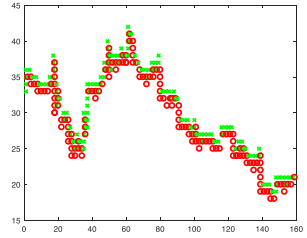
`subplot(a,b,c)` For multiple figures in a plot (`a`/`b`: number of rows/columns, `c`: selected plot).



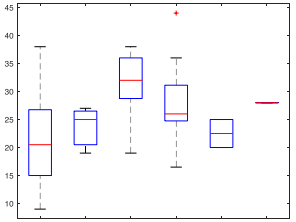
`compass(x)` Compass plot (arrows from center).



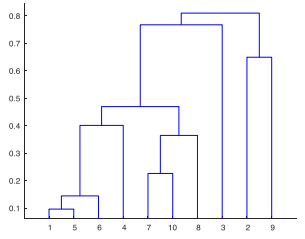
`pointfig(y)` Point and figure chart.



`boxplot(y)` Box plot.



`dendrogram(tree)` Dendrogram plot by `tree`.



Data science

Neural network

`nnstart`	Neural network app (GUI).
`patternnet(s,t,p)`	A pattern recognition (classification) network with `s`, `t`, and `p` as number of hidden layers, train, and performance function.
`feedforwardnet(s,t)`	An approximation (regression) network with `s` and `t` as number of hidden layers and train function.
`fitnet(s,t)`	Function fitting network with `s` and `t` as number of hidden layers and train function.
`cascade...` `forwardnet(s,t)`	An approximation (regression) network with `s` and `t` as number of hidden layers and train function.
`selforgmap`	Design a self-organizing map.
`competlayer(nc)`	Design a competitive layer network with `nc` as number of classes.
`network`	Design a custom neural network with different properties.
`view(net)`	View a designed neural network.
`train(net,i,o)`	Train a network using `i` and `o` as input and output.

predict(net,i)	Predictions for i as input by net.
perform(net,i,o)	Calculate network performance using i and o as input and output.

learning functions:

trainlm	Levenberg-Marquardt backpropagation.
trainbr	Bayesian regularization backpropagation.
trainrp	Resilient backpropagation.
trainscg	Scaled conjugate gradient backpropagation.
trainbfg	BFGS quasi-Newton backpropagation.
traincgb	Conjugate gradient backpropagation with Powell-Beale restarts.
traincgp	Conjugate gradient backpropagation with Polak-Ribire updates.
traincgf	Conjugate gradient backpropagation with Fletcher-Reeves updates.
traingda	Gradient descent with adaptive learning rate backpropagation.
traingdx	Gradient descent with momentum and adaptive learning rate backpropagation.
traingdm	Gradient descent with momentum backpropagation.
trainru	Unsupervised random order weight/bias training.
trainr	Random order incremental training.
trains	Sequential order incremental training.
learncon	Conscience bias learning function.
learnk	Kohonen weight learning function.
learnis	Instar weight learning function.
learnos	Outstar weight learning function.

Transfer functions:

tansig	Hyperbolic tangent sigmoid transfer function.
radbas	Radial basis transfer function.
radbasn	Normalized radial basis transfer function.
logsig	Log-sigmoid transfer function.
tribas	Triangular basis transfer function.
purelin	Linear transfer function.
satlin	Saturating linear transfer function.
poslin	Positive linear transfer function.
satlins	Symmetric saturating linear transfer function.
hardlim	Hard-limit transfer function.
hardlims	Symmetric hard-limit transfer function.
elliotsig	Elliot symmetric sigmoid transfer function.
elliot2sig	Elliot 2 symmetric sigmoid transfer function.
softmax	Soft max transfer function.
compet	Competitive transfer function.

Performance functions:

mse	Mean squared normalized error performance function.
mae	Mean absolute error performance function.
sae	Sum absolute error performance function.
sse	Sum squared error performance function.
crossentropy	Cross entropy performance function.

Input/Output process functions:

mapminmax	Normalize inputs/outputs between -1 and 1.
mapstd	Zero mean and unity variance normalization.
processpca	Principal component analysis for input.
removecons...	
tantrows	Remove constant inputs/outputs.
fixunknowns	Process unknown inputs.

Plots:

ploterrhist	Plot error histogram.
plotregression	Plot linear regression.
plotfit	Plot function fit.
plotperform	Plot network performance.
plottrainstate	Plot training state values.
plotconfusion	Plot classification confusion matrix.
plotroc	Plot receiver operating characteristic.
plotsomtop	Plot self-organizing map topology.
plotsomhits	Plot self-organizing map sample hits.
plotsomnc	Plot self-organizing map neighbor connections.
plotsomnd	Plot self-organizing map neighbor distances.
plotsomplanes	Plot self-organizing map weight planes.
plotsompos	Plot self-organizing map weight positions.

Basic Neural Network implementations (classification & regression):

```
%% Classification:
[i,o]=iris_dataset; % Import iris dataset.
nt=patternnet(5); % Design a netwrok.
i_w=nt.IW; % Store initial input weights.

%% Regression:
[i,o]=simplefit_dataset; % Import a sample dataset.
nt=feedforwardnet(10); % Design a network.
nt.performFcn='mae'; % Change performance func.
nt.inputs{1}.processFcns={'processpca'}; % PCA for
                                          input 1.

[nt,tx]=train(nt,i,o); % Train the network.
view(nt) % Show the network.
y=nt(i); % Insert input into the network.
perf=perform(nt,o,y); % Calculate performance.
plotconfusion(c,y); % Plot confusion matrix.
plotperform(tx); % Plot network performance.
```

Support vector machines/regression

Support vector machines:

classi...	
ficationLearner	Open classification learner app (GUI).
fitcsvm(i,o)	Train SVM with i as input and o as binary output for low or moderate dimensional data.
fitclinear(i,o)	Train linear SVM with i as input and o as binary output for high dimensional data.
fitcecoc(i,o)	Train SVM with i as input and o as multi-class output by error-correcting output codes model.
fitSVM...	
Posterior(svm_m)	Return trained SVM that contains the estimated score transformation function (svm_m is trained model).
templateSVM /Linear/ECOC	SVM/Linear SVM/Error-correcting output templates.
predict(svm_c,i)	Predict class labels using svm_c trained model and i as predictor data.

Support vector regression:

regres...	
sionLearner	Open regression learner app (GUI).
fitrsvm(i,o)	Train SVR with i as input and o as output for low or moderate dimensional data.
fitrlinear(i,o)	Train linear SVR with i as input and o as output for high dimensional data.
predict(svm_r,i)	Predict response using svm_r trained model and i as predictor data.

Basic Support Vector Machines & Support Vector Regression implementations:

```
%% Classification:
load fisheriris; % Load iris dataset.
i=meas; % Input data.
o=species; % Output data.
sz=numel(o); % Sample size.
trn_r=1:1:sz-40; % Train range.
tst_r=sz-39:1:sz; % Test range.
svm_c=fitcecoc(i(trn_r,:),o(trn_r,:)); % Train SVM.
svm_cl=crossval(svm_c); % Cross-validation.
svm_c_loss=kfoldLoss(svm_cl); % Error estimation.
pr_out=predict(svm_c,i(tst_r,:)); % Prediction.

%% Regression:
[i,o]=simplefit_dataset; % Import a sample dataset.
svm_r=fitrsvm(i',o'); % Train SVR.
svm_r_loss_1=resubLoss(svm_r); % Resubstitution loss.
conv=Mdl.ConvergenceInfo.Converged; % Convergance info.
nml=Mdl.NumObservations; % Number of observations.
```

Deep learning

Autoencoder:

`dp1=train...`	
`Autoencoder(i,hs)`	Train an Autoencoder.
`encode(dp1,x)`	Encode input data (`x`) using `dp1`.
`decode(dp1,v)`	Decode encoded data (`v`) using `dp1`.
`network(dp1)`	Convert Autoencoder to network object.
`plotWeights(dp1)`	Plot encoder weights of trained Autoencoder.
`view(autoenc)`	View Autoencoder structure.
`stack(dp1,dp2)`	Stack encoders together.
`predict(dp1,in)`	Predict response using `dp1` trained model and `in` as predictor data.

Convolutional neural network:

`op=training...`	
`Options(sv,ops)`	Training options (`ops`) of a network (`sv`: solver name).
`dp2=trainNet...`	
`work(i,o,lyr,op)`	Train a convolutional network using `i` and `o` as predictor and response variables (`op`: options).
`activations(...`	
`dp2,i,lyr)`	Network activations for `lyr` layer using `i` and `dp2` as data and trained network.
`predict(dp2,i)`	Predict response using `dp2` model and `i` as model and predictor data.
`classify(dp2,i)`	Classify data using `dp2` model and `i` as predictor data.
`importCaffe...`	
`Network`	Import pretrained networks models (Caffe).
`importCaffe...`	
`Layers`	Import network layers (Caffe).
`dp3=alexnet`	Pretrained AlexNet network.
`dp4=vgg16`	Pretrained VGG-16 network.
`dp5=vgg19`	Pretrained VGG-19 network.

Layers:

`imageInputLayer`	Image input layer.
`reluLayer`	Rectified linear unit layer.
`convolution2dLayer`	Create 2-D convolutional layer.
`maxPooling2dLayer`	Max pooling layer.
`fullyConnectedLayer`	Fully connected layer.
`averagePooling 2dLayer`	Average pooling layer.
`crossChannel... NormalizationLayer`	Channel-wise local response normalization layer.
`softmaxLayer`	Softmax layer.
`dropoutLayer`	Dropout layer.
`classificationLayer`	Classification output layer.
`regressionLayer`	Regression output layer.

Basic Autoencoder & Convolutional Neural Network implementations:

```
%% Autoencoder:
dt=abalone_dataset; % Load Abalone dataset.
dp1=trainAutoencoder(dt); % Train network.
dt_p=predict(dp1,dt); % Reconstruction.
mse_er=mse(dt-dt_p) % Calculate MSE.

%% Convolutional neural network:
load lettersTrainSet; % Load train data.
load lettersTestSet;; % Load test data.
lyrs=[imageInputLayer([21 21 1]);
    convolution2dLayer(8,15);
    reluLayer();
    fullyConnectedLayer(2);
    softmaxLayer();
    classificationLayer()];
ops=trainingOptions('sgdm'); % Settings.
rng('default') % For reproducibility.
nt=trainNetwork(XTrain,TTrain,lyrs,ops)
                                % Train network.
o_t=classify(nt,XTest); % Classify test data.
r_t=testDigitData.Labels; % Real test labels.
acc=sum(o_t==r_t)/numel(r_t); % Test accuracy.
```

Decision tree

`dt1=fitctree(i,o)`	Fit a binary classification decision tree to `i` and `o` as predictor and response.
`dt2=fitrtree(i,o)`	Fit a binary regression decision tree to `i` and `o` as predictor and response.
`templateTree`	Design a decision tree template.
`dt3=fitc... ensemble(i,o)`	Fit ensemble of classification decision trees to `i` and `o` as predictor and response.
`dt4=fitr... ensemble(i,o)`	Fit ensemble of regression decision trees to `o` and `i` as predictor and response.
`templateEnsemble`	Design an ensemble model template.
`dt5=Tree... Bagger(nt,i,o)`	Fit bag of decision trees to `i` and `o` as predictor and response (`nt`: number of trees.).
`md:`	`dt1`, `dt2`, `dt3`, `dt4`, or `dt5`.
`predict(md,in)`	Predict response using `mt` fitted model and `in` as predictor data.
`mdo:`	`dt3`, `dt4`, or `dt5`.
`oobPredict(mdo)`	Predict out-of-bag response of `mdo`.

Two basic Decision Trees implementations:

```
%% A simple classification decision tree.
load fisheriris; % Load iris dataset.
i=meas; % Input data.
o=species; % Output data.
dt1=fitctree(i,o,'CrossVal','on'); % Fit decision tree
                        to data using cross validation.
view(dt.Trained{1},'Mode','graph'); % Graph.
er=kfoldLoss(dt1) % Cross validation error rate.

%% An ensemble learner.
dt2=fitcensemble(i,o,'Method','Subspace'); Fit an ensemble
                                    using Subspace method.
er=resubLoss(dt2); % Resubstitution loss.
```

Linear/Nonlinear regression

Linear:

`fitlm(x,y)`	Fit linear regression to `x` and `y` as input and response.
`fitglm(x,y)`	Fit generalized linear regression to `x` and `y` as input and response.
`stepwiselm(x,y)`	Fit stepwise linear regression to `x` and `y` as input and response.
`stepwiseglm(x,y)`	Fit generalized stepwise linear regression to `x` and `y` as input and response.
`regress(x,y)`	Fit multiple linear regression to `x` and `y` as input and response.
`fitrlinear(x,y)`	Fit linear regression to `x` and `y` as input and response for high dimensional data.
`robustfit(x,y)`	Fit robust linear regression to `x` and `y` as input and response for high dimensional data.
`mvregress(x,y)`	Fit multivariate linear regression to `x` and `y` as input and response.
`fitlme(tb,fml)`	Fit linear mixed-effects model for `tb` and `fml` as data table and specified formula.
`fitglme(tb,fml)`	Fit generalized linear mixed-effects model for `tb` and `fml` as data table and specified formula.
`ls=lasso(x,y)`	Fit regularized least-squares regression to `x` and `y` as input and response using lasso or elastic net algorithms.
`ls=lassoglm(x,y)`	Fit regularized least-squares regression to `x` and `y` as input and response using lasso or elastic net. algorithm (generalized linear model regression).
`lasso(ls)`	Trace plot of lasso fitted model.
`ridge(y,x,k)`	Fit multilinear ridge regression to `x`, `y`, and `k` as input, response, and ridge parameters.
`plsreg... ress(x,y,nc)`	Fit Partial Least-Squares (PLS) regression to `x` and `y` as input and response(`nc`: PLS components).

Linear: ...

`mnrfit(x,y)`	Fit multinomial logistic regression to `x` and `y` as input and response.
`glmfit(x,y)`	Fit generalized linear model regression to `x` and `y` as input and response.
`predict(lr,xn)`	Predict response using `lr` trained model and `xn` as predictor data.
`display(mdl)`	Display fitted model.

Nonlinear:

`fitnlm(x,y,...` `mdf,beta)`	Fit specified model of `mdf` for `x`, `y` and `beta` as input, response, and coefficients.
`nlinfit(x,y,...` `mdf,beta)`	Fit specified model of `mdf` for `x`, `y` and `beta` as input, response, and coefficients.
`mnrfit(x,y,...` `gr,v,mdf,beta)`	Fit nonlinear mixed-effects regression for `x`, `y`, `gr`, `v`, `mdf`, and `beta` as input, response, groups, predictor (take the same value in a group), function, and initial estimates for fixed effects.
`nlmefitsa(--)`	Fit nonlinear mixed-effects model with stochastic EM algorithm using above inputs.
`fitrgp(x,y)`	Fit a Gaussian Process Regression (GPR) model to `x` and `y` as input and response.

Basic linear & nonlinear regression implementations:

```
%% Linear regression:
load carsmall; % Load carsmall dataset.
x=[Weight,Acceleration]; % Input data.
y=MPG; % Response data.
lm=fitlm(x,y); % Fit linear regression.
display(lm); % Display reports.
plot(lm); % Scatter plot.
plotAdjustedResponse(lm,1); % Adjusted response plot
                                    of variable 1.
predict(lm,x); % Reconstruct response.

%% Nonlinear regression:
mdf=@(b,x)b(1)+b(2)*x(:,1);
                            % Model function.
bt=[-50 500]; % Beta values.
nl=fitnlm(x(:,1),y,mdf,bt); % Fit nonlinear
                                regression.
display(nl); % Display reports.
predict(nl,x(:,1)); % Reconstruct response.
```

Clustering

`linkage(x)`	Agglomerative hierarchical cluster tree for `x` as data.
`clusterdata(x,cf)`	Agglomerative clusters for `x` and `cf` as data and cutoff.
`kmeans(x,k)`	K-means clustering using `x` and `k` as data and number of clusters.
`findcluster`	Fuzzy clustering tool (GUI).
`subclust(x,rng)`	Fuzzy subtractive clustering using `x` and `rng` as data and cluster influence range.
`fcm(x,k)`	Fuzzy c-means clustering using `x` and `k` as data and number of clusters.
`kmedoids(x,k)`	Kmedoids clustering using `x` and `k` as data and number of clusters.
`ts=KDTree...` `Searcher(x)`	Grow kd-tree using `x` as data.
`createns(x)`	Create object for growing kd-tree using `x` as data.
`ts=Exhaustive...` `Searcher(x)`	Prepare exhaustive nearest neighbors searcher using `x` as data.
`knnsearch(ts,y)`	Search for the nearest neighbor in `ts` to each point in `y`.
`range...` `search(ts,y,r)`	Find all neighbors within specified distance in `ts` to each point in `y` (`r`: radius around each `ts` to each point).

Two basic clustering models implementations:

```
% k-means clustering.
load fisheriris; % Load iris dataset.
data=meas(:,1:2); % Select data.
[inx,c]=kmeans(data,3); % k-means clustering.
% inx: cluster indices, c: cluster centers.

% Fuzzy c-means clustering.
[cnt,u]=fcn(data,3); % Fuzzy c-means clustering.
% cnt: cluster centers, u: fuzzy partition matrix.
```

Dimension reduction/feature selection

`c=pca(x)`	Principal component analysis of `x`.
`c=ppca(x,m)`	Probabilistic principal component analysis of `x` and `m` as data and number of components.
`biplot(c)`	Biplot of the PCA coefficients.
`pcacov(w)`	Principal component analysis on `w` as covariance matrix.
`pcares(x,d)`	Residuals from principal component analysis for `x` and `d` as data and number of dimensions.
`f=factoran(x,m)`	Factor analysis of `x` and `m` as data and number of factors.
`rotatefactors(f)`	Rotate factor loadings.
`sequentialfs...` `(fun,i,o)`	Sequential feature selection using `i`, `o`, and `fun` as input, predictor, and function handle that defines criterion.
`relieff(i,o,k)`	ReliefF algorithm attributes importance extraction using `i`, `o`, and `k` as input, predictor, and number of neighbors.

Cross-validation

`c=cvpartition...` `(o,'KFold',k)`	K-fold cross-validation (o: predictor).
`repartition(c)`	Data repartition for cross-validation.
`crossval(fun,x)`	Loss estimate of cross-validation for the function `fun` and `x` as data.
`training(c,ix)`	Training indices of cross-validation for repetition `ix`.
`test(c,ix)`	Test indices of cross-validation for repetition `ix`.
`testcholdout...` `(y1,y2,yr)`	Compare predictive accuracies of two classification models (McNemar test) using `y1`, `y2`, and `yr` as first model output, second model output, and true labels.
`testckfold...` `(c1,c2,x1,x2)`	Compare predictive accuracies of two classification models by paired F cross-validation test using `c1`, `c2`, `x1`, and `x2` as first model, second model, first data table and second data table.

Ali Habibnia a.habibnia@lse.ac.uk
Eghbal Rahimikia erahimi@ut.ac.ir
Download page: http://personal.lse.ac.uk/habibnia/
* Feel free to send your feedback.