

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/317313022>

MATLAB Cheat Sheet for Data Science – London School of Economics

Technical Report · May 2017

DOI: 10.13140/RG.2.2.16859.77607

CITATIONS

0

READS

12,012

2 authors:



[Ali Habibnia](#)

Virginia Polytechnic Institute and State University

13 PUBLICATIONS 10 CITATIONS

[SEE PROFILE](#)



[Eghbal Rahimikia](#)

The University of Manchester

3 PUBLICATIONS 6 CITATIONS

[SEE PROFILE](#)

Some of the authors of this publication are also working on these related projects:



Foreign Exchange Rate Risk Measurement and Managemen [View project](#)



Deep Learning in Finance [View project](#)



Ali Habibnia Eghbal Rahimikia

Last update: May, 2017
MATLAB r2017a

Preliminaries

func : This function or command requires a toolbox to execute.

Ways to get help

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

File extensions

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

Common data types

<code>single</code>	Single precision numerical data (32 bits).
<code>double</code>	Double precision numerical data (64 bits).
<code>char</code>	Character array.
<code>string</code>	String array.
<code>logical</code>	True (1) or false (0).
<code>struct</code>	Structure array.
<code>cell</code>	Cell array.

`map container` Map values to unique keys (dictionary).

Data import/export

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

Basic commands

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

Create basic variables

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

Basic math functions

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

Trigonometric functions

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

Linear algebra

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

Accessing/assignment elements

If `x` is a vector:

<code>x(i)</code>	Element i -th of <code>x</code> ($i, j \geq 1$).
<code>x(i:j)</code>	Elements from the i -th to the j -th of <code>x</code> .
<code>x(i:end)</code>	Elements from the i -th to the last one of <code>x</code> .
<code>x(:)</code>	All the elements of <code>x</code> .

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

If `A` is a cell:

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

Character and string

`st`: string.
`char`: character.

<code>x='text'</code>	Define variable <code>x</code> to be 'text' (char).
<code>x(i)</code>	i -th part of char.
<code>x(i:j)</code>	i -th to j -th part of char.
<code>x=string('text')</code>	Define variable <code>x</code> to be 'text' (str).
<code>x={'s1','s2'}</code>	Define more than one char.
<code>x=[string('st1')... ,string('st2')]</code>	Define more than one str.
<code>x{i,j}</code>	Element i,j of <code>x</code> (char).
<code>x(i,j)</code>	Element i,j of <code>x</code> (str).
<code>x{i}(j)</code>	i -th part of j -th char/str.
<code>x{i:m}(j)</code>	i to m -th part of j -th char/str.
<code>strcat(x1,x2)</code>	Concatenate characters/strings.

<code>char(x)</code>	Create character from numeric array.
<code>strfind(x1,ptrn)</code>	Search <code>x1</code> char/str for <code>ptrn</code> pattern.
<code>strjoin(x)</code>	Construct char array using <code>x</code> char/str elements.
<code>lower(x)</code>	Convert char/str to lowercase.
<code>upper(x)</code>	Convert char/str to uppercase.
<code>strcmp(x1,x2)</code>	Compare char/str of <code>x1</code> and <code>x2</code> .
<code>strcmpi(x1,x2)</code>	Compare char/str of <code>x1</code> and <code>x2</code> (case insensitive).
<code>split(x,d1)</code>	Split strings in string array (<code>x</code>) at <code>d1</code> delimiters.
<code>strsplit(x,d1)</code>	Split <code>x</code> char/str at <code>d1</code> delimiters.
<code>sprintf('fmt',x1,x2)</code>	Format data based on <code>fmt</code> structure.
<code>strvcat(x1,x2)</code>	Vertically concatenate <code>x1</code> and <code>x2</code> (ignore spaces).

Regular expression

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

'IS*' functions

Return true where:

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

Return true if:

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

'IS*' functions...

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

Convert functions

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

```
% 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
```

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

Parallel computing (CPU & GPU)

CPU:

<code>parpool(size)</code>	Create a new parallel pool (<code>size</code> is number of CPU workers).
<code>gcp</code>	Return the current parallel pool.
<code>ticBytes/...</code> <code>tocBytes(gcp)</code>	Start/stop calculation of the bytes transferred to the workers.
<code>batch('scr')</code>	Run a script or function on a worker.
<code>gather(A)</code>	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:

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

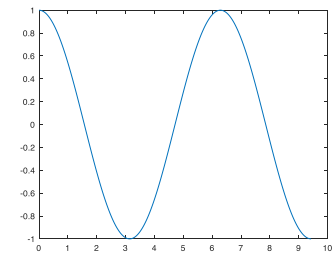
Plotting & Figures

<code>figure</code>	Open up a new figure window.
<code>axis normal</code>	Default axis limits and scaling behavior.
<code>axis tight</code>	Force axis to be equaled to data range.
<code>axis equal</code>	Force axis to be scaled equally.
<code>axis square</code>	Axis lines with equal lengths.
<code>axis fill</code>	Lengths of each axis line fill the rectangle.
<code>title('Title')</code>	Add a title at the top of the plot.
<code>xlabel('lb1')</code>	Label the x axis as <code>lb1</code> .
<code>ylabel('lb1')</code>	Label the y axis as <code>lb1</code> .
<code>zlabel('lb1')</code>	Label the z axis as <code>lb1</code> .
<code>legend('v','w')</code>	Add label to <code>v</code> and <code>w</code> curves.
<code>grid on/off</code>	Include/Omit a grid in the plot.
<code>box on/off</code>	Display/hide the box outline around axes.
<code>datetick('x',fm)</code>	Date formatted tick labels (<code>fm</code> is format).
<code>xtickformat(fm)</code>	X-axis label format.
<code>ytickformat(fm)</code>	Y-axis label format.
<code>xlim([min,max])</code>	X-axis limits from <code>min</code> to <code>max</code> .
<code>ylim([min,max])</code>	Y-axis limits from <code>min</code> to <code>max</code> .
<code>zlim([min,max])</code>	Z-axis limits from <code>min</code> to <code>max</code> .
<code>hold on/off</code>	Allow/prevent plotting on the same graph.
<code>text(x,y,text)</code>	Add <code>text</code> to a point (<code>x</code> and <code>y</code> are scalars in data units).

`fn` is a function:

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

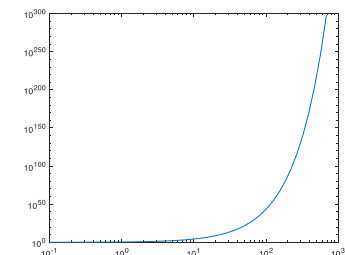
Input arguments:

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:

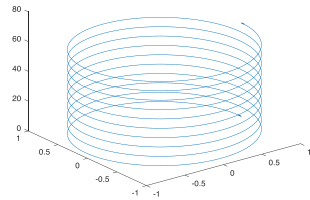
Color	Line color.
LineStyle	Line style.
LineWidth	Line width.
Marker	Marker symbol.
MarkerIndices	Indices of marker data points.
MarkerEdgeColor	Marker outline color.
MarkerFaceColor	Marker fill color.
MarkerSize	Size of marker.

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

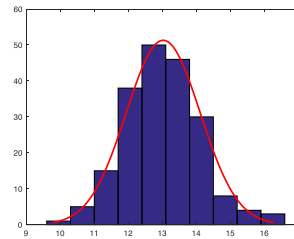


`semilogx(x,y)` Logarithmic `x` axis.
`semilogy(x,y)` Logarithmic `y` axis.

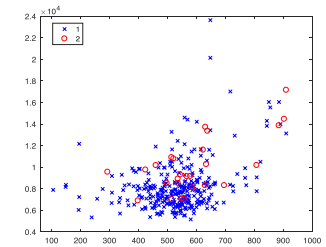
`plot3(x,y,z)` Three-dimensional analogue of plot.



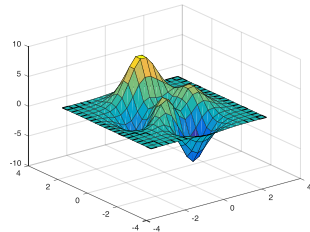
`histfit(y)` Histogram plot with distribution fit.



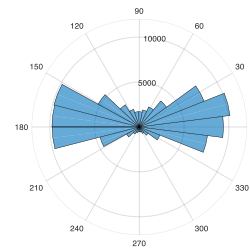
`gscatter(x,y,group)` 2-D scatter plot of x and y by group.



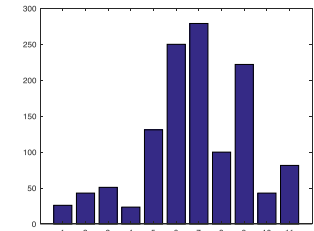
`surf(x,y,z)` 3-D shaded surface plot.



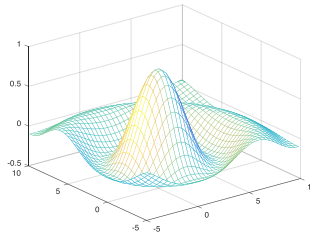
`polarhistogram(y)` Histogram plot (polar coordinates).



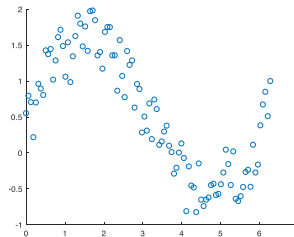
`bar(y)` Bar plot.



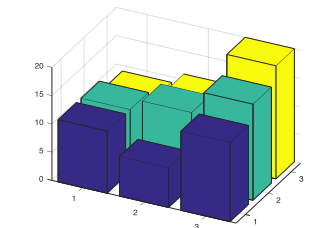
`mesh(x,y,z)` 3-D mesh surface plot.



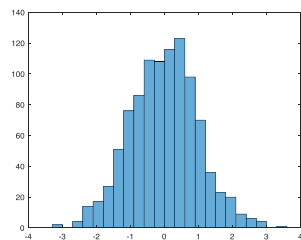
`scatter(x,y)` 2-D scatter plot by x and y.



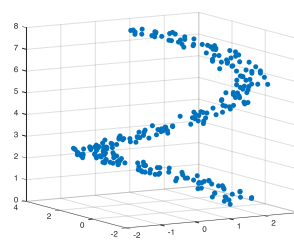
`bar3(y)` 3-D bar plot.



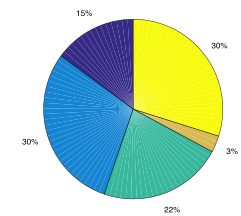
`histogram(y)` Histogram plot.



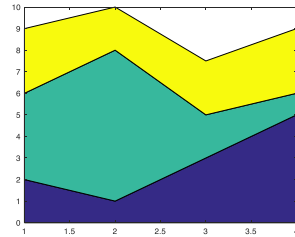
`scatter(x,y,z)` 3-D scatter plot by x, y, and z.



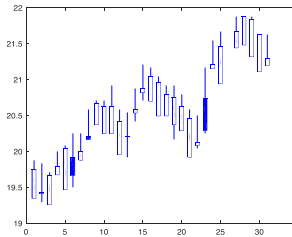
`pie(y)/pie3(y)` 2-D/3-D pie plot.



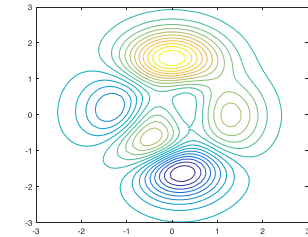
`area(y)` 2-D area plot.



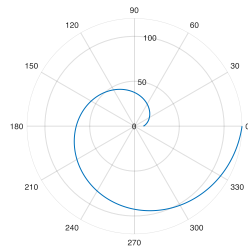
`candle(y)` Candlestick chart.



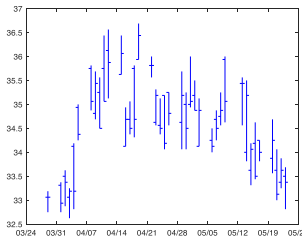
`contour(y)` Contour plot of matrix y .



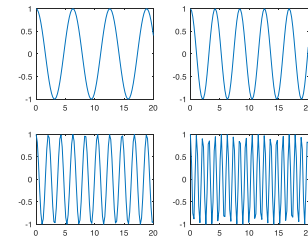
`polarplot(theta, rho)` Polar plot (θ : angle, ρ : radius).



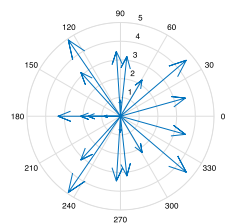
`highlow(h, l, o, c)` Plot h (high), l (low), o (open), and c (close) prices of an asset.



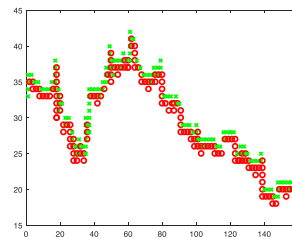
`subplot(a, b, c)` For multiple figures in a plot (a/b : number of rows/columns, c : selected plot).



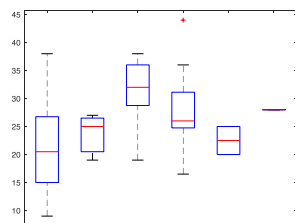
`compass(x)` Compass plot (arrows from center).



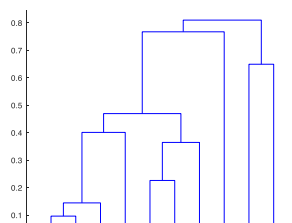
`pointfig(y)` Point and figure chart.



`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.

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

learning functions:

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

Transfer functions:

<code>tansig</code>	Hyperbolic tangent sigmoid transfer function.
<code>radbas</code>	Radial basis transfer function.
<code>radbasn</code>	Normalized radial basis transfer function.
<code>logsig</code>	Log-sigmoid transfer function.
<code>tribas</code>	Triangular basis transfer function.
<code>purelin</code>	Linear transfer function.
<code>satlin</code>	Saturating linear transfer function.
<code>poslin</code>	Positive linear transfer function.
<code>satlins</code>	Symmetric saturating linear transfer function.
<code>hardlim</code>	Hard-limit transfer function.
<code>hardlims</code>	Symmetric hard-limit transfer function.
<code>elliotsig</code>	Elliot symmetric sigmoid transfer function.
<code>elliott2sig</code>	Elliot 2 symmetric sigmoid transfer function.
<code>softmax</code>	Soft max transfer function.
<code>compet</code>	Competitive transfer function.

Performance functions:

<code>mse</code>	Mean squared normalized error performance function.
------------------	---

<code>mae</code>	Mean absolute error performance function.
<code>sae</code>	Sum absolute error performance function.
<code>sse</code>	Sum squared error performance function.
<code>crossentropy</code>	Cross entropy performance function.

Input/Output process functions:

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

Plots:

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

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

Support vector regression:

<code>regres...</code>	
<code>sionLearner</code>	Open regression learner app (GUI).
<code>fitrsvm(i,o)</code>	Train SVR with <code>i</code> as input and <code>o</code> as output for low or moderate dimensional data.
<code>fitrlinear(i,o)</code>	Train linear SVR with <code>i</code> as input and <code>o</code> as output for high dimensional data.
<code>predict(svm_r,i)</code>	Predict response using <code>svm_r</code> trained model and <code>i</code> 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; % Convergence info.
nml=Mdl.NumObservations; % Number of observations.

```


Deep learning

Autoencoder:

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

Convolutional neural network:

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

Layers:

<code>imageInputLayer</code>	Image input layer.
<code>reluLayer</code>	Rectified linear unit layer.
<code>convolution2dLayer</code>	Create 2-D convolutional layer.
<code>maxPooling2dLayer</code>	Max pooling layer.
<code>fullyConnectedLayer</code>	Fully connected layer.
<code>averagePooling2dLayer</code>	Average pooling layer.
<code>crossChannel...</code>	
<code>NormalizationLayer</code>	Channel-wise local response normalization layer.
<code>softmaxLayer</code>	Softmax layer.
<code>dropoutLayer</code>	Dropout layer.

<code>classificationLayer</code>	Classification output layer.
<code>regressionLayer</code>	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

<code>dt1=fittctree(i,o)</code>	Fit a binary classification decision tree to i and o as predictor and response.
<code>dt2=fitttree(i,o)</code>	Fit a binary regression decision tree to i and o as predictor and response.
<code>templateTree</code>	Design a decision tree template.
<code>dt3=fittc...ensemble(i,o)</code>	Fit ensemble of classification decision trees to i and o as predictor and response.
<code>dt4=fitr...ensemble(i,o)</code>	Fit ensemble of regression decision trees to o and i as predictor and response.
<code>templateEnsemble</code>	Design an ensemble model template.
<code>dt5=Tree...Bagger(nt,i,o)</code>	Fit bag of decision trees to i and o as predictor and response (nt: number of trees.).
<code>md:</code>	dt1, dt2, dt3, dt4, or dt5.
<code>predict(md,in)</code>	Predict response using mt fitted model and in as predictor data.
<code>mdo:</code>	dt3, dt4, or dt5.
<code>oobPredict(mdo)</code>	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:

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

Linear: ...

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

Nonlinear:

<code>fitnlm(x,y,... mdf,beta)</code>	Fit specified model of mdf for x , y and beta as input, response, and coefficients.
<code>nlinfit(x,y,... mdf,beta)</code>	Fit specified model of mdf for x , y and beta as input, response, and coefficients.
<code>mnrfit(x,y,... gr,v,mdf,beta)</code>	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.
<code>nlmefitsa(--)</code>	Fit nonlinear mixed-effects model with stochastic EM algorithm using above inputs.
<code>fitrgp(x,y)</code>	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

<code>linkage(x)</code>	Agglomerative hierarchical cluster tree for x as data.
<code>cclusterdata(x,cf)</code>	Agglomerative clusters for x and cf as data and cutoff.
<code>kmeans(x,k)</code>	K-means clustering using x and k as data and number of clusters.
<code>findcluster subclust(x,rng)</code>	Fuzzy clustering tool (GUI). Fuzzy subtractive clustering using x and rng as data and cluster influence range.
<code>fcm(x,k)</code>	Fuzzy c-means clustering using x and k as data and number of clusters.
<code>kmedoids(x,k)</code>	Kmedoids clustering using x and k as data and number of clusters.
<code>ts=KDTree... Searcher(x) createns(x)</code>	Grow kd-tree using x as data. Create object for growing kd-tree using x as data.
<code>ts=Exhaustive... Searcher(x)</code>	Prepare exhaustive nearest neighbors searcher using x as data.
<code>knnsearch(ts,y)</code>	Search for the nearest neighbor in ts to each point in y .
<code>range... search(ts,y,r)</code>	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]=fcm(data,3); % Fuzzy c-means clustering.
% cnt: cluster centers, u: fuzzy partition matrix.
```

Dimension reduction/feature selection

<code>c=pca(x)</code>	Principal component analysis of x .
<code>c=ppca(x,m)</code>	Probabilistic principal component analysis of x and m as data and number of components.
<code>biplot(c)</code>	Biplot of the PCA coefficients.
<code>pcacov(w)</code>	Principal component analysis on w as covariance matrix.

<code>pcares(x,d)</code>	Residuals from principal component analysis for x and d as data and number of dimensions.
<code>f=factoran(x,m)</code>	Factor analysis of x and m as data and number of factors.
<code>rotatefactors(f) sequentialfs... (fun,i,o)</code>	Rotate factor loadings. Sequential feature selection using i , o , and fun as input, predictor, and function handle that defines criterion.
<code>relieff(i,o,k)</code>	Relieff algorithm attributes importance extraction using i , o , and k as input, predictor, and number of neighbors.

Cross-validation

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

Copyright ©2017 / MATLAB R2017a
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.