COMSOL SCRIPT™

COMMAND REFERENCE

VERSION 1.2

COMSOL

How to contact COMSOL:

Benelux
COMSOL BV
Röntgenlaan 19
2719 DX Zoetermeer
The Netherlands
Phone: +31 (0) 79 363 4230
Fax: +31 (0) 79 361 4212
info@femlab.nl
www.femlab.nl

Denmark
COMSOL A/S
Diplomvej 376
2800 Kgs. Lyngby
Phone: +45 88 70 82 00
Fax: +45 88 70 80 90
info@comsol.dk
www.comsol.dk

Finland
COMSOL OY
Arabianranta 6
FIN-00560 Helsinki
Phone: +358 9 2510 400
Fax: +358 9 2510 4010
info@comsol.fi
www.comsol.fi

France
COMSOL France
WTC, 5 pl. Robert Schuman
F-38000 Grenoble
Phone: +33 (0)4 76 46 49 01
Fax: +33 (0)4 76 46 07 42
info@comsol.fr
www.comsol.fr

Germany
FEMLAB GmbH
Berliner Str. 4
D-37073 Göttingen
Phone: +49-551-99721-0
Fax: +49-551-99721-29
info@femlab.de
www.femlab.de

Italy
COMSOL S.r.l.
Via Vittorio Emanuele II, 22
25122 Brescia
Phone: +39-030-3793800
Fax: +39-030-3793899
info.it@comsol.com
www.it.comsol.com

Norway
COMSOL AS
Søndre gate 7
NO-7485 Trondheim
Phone: +47 73 84 24 00
Fax: +47 73 84 24 01
info@comsol.no
www.comsol.no

Sweden
COMSOL AB
Tegnérgatan 23
SE-111 40 Stockholm
Phone: +46 8 412 95 00
Fax: +46 8 412 95 10
info@comsol.se
www.comsol.se

Switzerland
FEMLAB GmbH
Technoparkstrasse 1
CH-8005 Zürich
Phone: +41 (0)44 445 2140
Fax: +41 (0)44 445 2141
info@femlab.ch
www.femlab.ch

United Kingdom
COMSOL Ltd.
UH Innovation Centre
College Lane
Hatfield
Hertfordshire AL10 9AB
Phone:+44-(0)-1707 284747
Fax: +44-(0)-1707 284746
info.uk@comsol.com
www.uk.comsol.com

United States
COMSOL, Inc.
1 New England Executive Park
Suite 350
Burlington, MA 01803
Phone: +1-781-273-3322
Fax: +1-781-273-6603

COMSOL, Inc.
10850 Wilshire Boulevard
Suite 800
Los Angeles, CA 90024
Phone: +1-310-441-4800
Fax: +1-310-441-0868

COMSOL, Inc.
744 Cowper Street
Palo Alto, CA 94301
Phone: +1-650-324-9935
Fax: +1-650-324-9936

info@comsol.com
www.comsol.com

For a complete list of international representatives, visit
www.comsol.com/contact

Company home page
www.comsol.com

COMSOL user forums
www.comsol.com/support/forums

COMSOL Script Reference Guide

Version: October 2007 COMSOL 3.4

CONTENTS

Chapter 1: COMSOL Script Command Reference

I

COMSOL Script Command Reference

Summary of Commands

Purpose Evaluate an elementary function.

Synopsis

```
abs(a)
```

and the same format for other elementary functions

Description `<elementary function>(a)` computes the elementary function of the matrix `a` pointwise. The following elementary functions are available:

FUNCTION	WHAT IT COMPUTES
`abs`	Absolute value
`acos`	Inverse cosine
`acosh`	Inverse hyperbolic cosine
`acot`	Inverse cotangent
`acoth`	Inverse hyperbolic cotangent
`acsc`	Inverse cosecant
`acsch`	Inverse hyperbolic cosecant
`angle`	Polar angle of complex number
`asec`	Inverse secant
`asech`	Inverse hyperbolic secant
`asin`	Inverse sine
`asinh`	Inverse hyperbolic sine
`atan`	Inverse tangent
`atanh`	Inverse hyperbolic tangent
`ceil`	Floating-point number rounded to the next integer towards infinity
`conj`	Complex conjugate
`cos`	Cosine
`cosh`	Hyperbolic cosine
`cot`	Cotangent
`coth`	Hyperbolic cotangent
`csc`	Cosecant
`csch`	Hyperbolic cosecant
`exp`	Exponential
`fix`	Floating-point number rounded to the next integer toward 0

FUNCTION	WHAT IT COMPUTES
`floor`	Floating-point number rounded to the next integer toward negative infinity
`imag`	Imaginary part of complex number
`log`	Natural logarithm
`log10`	Base-10 logarithm
`real`	Real part of complex number
`reallog`	Natural logarithm of nonnegative real number
`realsqrt`	Square root of nonnegative real number
`round`	Floating-point number rounded to nearest integer
`sec`	Secant
`sech`	Hyperbolic secant
`sign`	Sign of argument: +1 if positive, 0 if 0, −1 if negative
`sin`	Sine
`sinh`	Hyperbolic sine
`sqrt`	Square root
`tan`	Tangent
`tanh`	Hyperbolic tangent

Purpose Add one or more directories to the COMSOL Script search path.

Synopsis

```
addpath(dir1, ...)
addpath(dir1, ..., '-begin')
addpath(dir1, ..., '-end')
```

Description `addpath(dir1, ...)` and `addpath(dir1, ..., '-begin')` prepend directories to the COMSOL Script search path.

`addpath(dir1, ..., '-end')` appends directories to the COMSOL Script search path.

See also `path`

Purpose

Airy functions

Synopsis

```
w = airy(z)
w = airy(k,z)
```

Description

`airy` computes the Airy functions Ai(z), Bi(z) or their derivatives depending on the flag `k`, as indicated below. The default is zero.

TABLE 1-1: AIRY FUNCTION FLAG VALUES

K	FUNCTION
0	Ai(z)
1	Ai'(z)
2	Bi(z)
3	Bi'(z)

`airy` can give the following errors:

TABLE 1-2: AIRY FUNCTION ERROR CODES

ERROR CODE	DESCRIPTION
1	Illegal input
2	Overflow
3	Loss of significance by argument reduction
4	Complete loss of accuracy in argument reduction
5	No convergence

See also

`bessel, besselh, besseli, besselj, besselk, bessely`

Purpose

Determine if all the elements along a dimension are nonzero.

Synopsis

```
y = all(x)
y = all(x,dim)
```

Description

`y = all(x)` tests if all elements along a specific dimension are nonzero.

When `x` is a vector, `all(x)` returns true if all the elements of `x` are nonzero, and false otherwise. When `x` is a matrix, `y` is a row vector where each element is true or false depending on whether or not all the elements of corresponding column of x are nonzero. When `x` is an n-dimensional array, `all(x)` tests for nonzero elements along the first nonsingleton dimension of `x`.

`y = all(x,dim)` tests `x` for nonzero elements along the dimension `dim`.

Example

```
all(eye(10),2)
```

See also

`any`

Purpose

Compute the logical AND of two matrices pointwise.

Synopsis

```
d = and(a, b)
```

Description

`d = and(a, b)` computes the pointwise logical AND of the two matrices `a` and `b`. For each dimension, `a` and `b` must have the same size or either of them must have size 1. In the latter case, the unit dimension is expanded to the size of the nonunit dimension.

`and(a, b)` is equivalent to `a & b`.

Examples

```
[0 0 1 1] & [0 1 0 1]
```

```
[0 1] & 0
```

```
[0 1] & [1 ; 0]
```

See also

`not`, `or`, `xor`

Purpose Get the result of the last operation.

Synopsis

```
a = ans
```

Description `a = ans` returns the result of the last operation that produced a result that was not assigned to any variable.

Purpose

Determine if any element along a dimension is nonzero.

Synopsis

```
y = any(x)
y = any(x,dim)
```

Description

`y = any(x)` tests if any element along a specific dimension is nonzero.

When `x` is a vector, `any(x)` returns true if any element of `x` is nonzero, and false otherwise. When `x` is a matrix, `y` is a row vector where each element is true or false depending on whether or not any elements of corresponding column of x is nonzero. When `x` is an n-dimensional array, `any(x)` tests for nonzero elements along the first nonsingleton dimension of `x`.

`y = any(x,dim)` tests `x` for nonzero elements along the dimension `dim`.

Example

```
any(eye(10),2)
```

See also

`all`

Purpose Get the argument names for an inline function.

Synopsis

```
names = argnames(f)
```

Description `names = argnames(f)` returns the argument names for the inline function `f` in a cell array.

See also `inline`

Purpose Assign a value to a variable in another workspace.

Synopsis `assignin(ws, var, val)`

Description `assignin(ws, var, val)` assigns the value `val` to the variable `var` in the workspace `ws`. Possible values for `ws` are `'caller'` (the workspace owning the current workspace through a function call) and `'base'` (the root workspace).

See also `evalin`

Purpose Compute binary atan.

Synopsis

```
v = atan2(y, x)
```

Description `v = atan2(y, x)` computes the pointwise atan of the two matrices `x` and `y`. For scalars, `atan2(y, x)` is the angle `v` such that `tan(v) = y/x`.

The sizes of `x` and `y` must be identical unless one of them is a scalar; in that case, the scalar is expanded to a matrix of the correct size.

See also `Elementary functions`

Purpose

Create axes for use in a GUI.

Synopsis

```
ax = axes
```

Description

`ax = axes` creates an axes component that can be added to a `frame` or a `dialog` in a built graphical user interface.

The function returns an axes object that can then be manipulated further using the methods in the following table:

TABLE 1-3: METHODS FOR MANIPULATING A AXES OBJECT.

METHOD	DESCRIPTION
`getHandle`	Returns a handle to the axes. This handle can then be used as any other axes handle. It can for example be used as parent in plotting command or to set and get axes properties such as axis limits.
`addMouseListener(name)`	Specifies that the function with the given name should be run the mouse is moved or clicked over the axes.

See also

`dialog`, `frame`, `panel`

Purpose

Control axis limits and properties.

Synopsis

```
axis(limits)
axis('auto')
axis('equal')
axis('manual')
axis('normal')
axis('on')
axis('off')
axis('tight')
axis(ax,...)
```

Description

`axis(limits)` sets the limits of the current axis to the limits given by the vector `limits`. In 2D it has the values `[xmin xmax ymin ymax]` and in 3D it has the values `[xmin xmax ymin ymax zmin zmax]`.

`axis('auto')` dictates that axis limits should automatically be recomputed to fit graphics that are added to the axes.

`axis('equal')` sets the aspect ratio so that distances in different directions are equal in size on the screen.

`axis('manual')` sets axis limits in Manual mode, which means the axis limits are kept and not automatically updated when new graphics are plotted into the axes.

`axis('normal')` is the opposite of `axis('equal')`. It allows distances in different directions to have different lengths on the screen.

`axis('on')` displays the axis labeling, tick marks, and the box. This has an effect only in 3D.

`axis('off')` turns off the display of axis labeling, tick marks and the box. This has an effect only in 3D.

`axis('tight')` makes the axis limits tight around the plotted data.

`axis(ax,...)` can be used with all the different syntaxes just given to affect the axes `ax` instead of the current axes.

Purpose

Create a bar graph.

Synopsis

```
bar(x,y)
bar(y)
bar(x,y,width)
bar(y,width)
```

Description

`bar(x,y)` draws a bar graph. `x` is a vector and `y` is an `m`-by-`n` matrix or a vector. If `y` is a vector it has the same length as `x` and length(`x`) bars are produced. If `y` is a matrix `x` must have length(`m`) and `m` groups of `n` bars will be created.

`bar(y)` uses `x=1:m`.

`bar(...,width)` can be used to specify the relative width of the bars. The default value is 0.8 and a value of 1 means that the bars will touch each other.

`bar(...,'grouped')` draws multiple bars within each group.

`bar(...,'stacked')` stacks the bars vertically within each group.

The bars are normally colored using colors from the colormap in the figure plotted into.

`bar(...,'linecolor')` where `'linecolor'` is one of the color strings listed in PLOT can be used to color all bars using the same color.

`See also` `plot`

Purpose Convert strings in a specific base to decimal integers.

Synopsis

```
d = base2dec(str,b)
```

Description `d = base2dec(str,b)` converts a string `str` representing a number in base `b` to a decimal integer. `str` can also be a string matrix, in which case `base2dec` converts each row, or a cell array of strings, in which case `base2dec` converts each element. `b` must be an integer between 2 and 36, inclusive.

Example `base2dec('21',5)` converts `21` in base 5 to `11` in base 10.

See also `bin2dec`, `hex2dec`, `hex2num`, `dec2base`, `dec2bin`, `dec2hex`, `num2hex`

Purpose

Compute a Bessel function.

Synopsis

```
b = bessel(n, z)
b = besselh(n, z)
b = besselh(n, m, z)
b = besseli(n, z)
b = besselj(n, z)
b = besselk(n, z)
b = bessely(n, z)
```

Description

`b = bessel(n,z)`—see `b = besselj(n, z)` below.

`b = besselh(n,z)` computes the Bessel function of the third kind with `m` set to `1`.

`b = besselh(n,m,z)` computes $H_n^{(m)}$, the Bessel function of the third kind, also called the Hankel function, of order `n`, defined as

$$H_n^{(1)} = J_n(x) + iY_n(x)$$
$$H_n^{(2)} = J_n(x) - iY_n(x)$$

where $J_n(x)$ is the Bessel function of the first kind, and $Y_n(x)$ is the Bessel function of the second kind.

`b = besseli(n,z)` computes the modified Bessel function of the first kind of order n, defined as

$$I_n(z) = \frac{1}{2\pi i}\oint e^{(z/2)(t+1/t)}t^{-n-1}dt$$

`b = besselj(n,z)` and `b = bessel(n,z)` compute the Bessel function of the first kind of order `n`, defined as

$$J_n(z) = \frac{1}{2\pi i}\oint e^{(z/2)(t-1/t)}t^{-n-1}dt$$

`b = besselk(n,z)` computes the modified Bessel function of the second kind of order `n`, defined as

$$K_n(z) = \frac{\pi}{2}\frac{I_{-n}(x) - I_n(x)}{\sin(n\pi)}$$

`b = bessely(n,z)` computes the Bessel function of the second kind of order `n`, defined as:

$$Y_n(z) = \frac{J_n(z)\cos(n\pi) - J_{-n}(z)}{\sin(n\pi)}$$

The sizes of `z` and `n` must be identical unless one of them is a scalar; in that case, the scalar is expanded to a matrix of the correct size.

See also `airy`

Purpose

Beta function.

Synopsis

```
b = beta(x,y)
```

Description

`b = beta(x,y)` computes the beta function of `x` and `y`, defined as

$$B(x,y) = \int_0^1 t^{x-1}(1-t)^{y-1}dt = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$$

where $\Gamma(x)$ is the gamma function. `x` and `y` must be real arrays of the same size, or either can be a scalar.

See also

`betainc`, `betaln`, `gamma`

Purpose

Incomplete beta function.

Synopsis

```
IX = betainc(x,a,b)
```

Description

`IX = betainc(x,a,b)` computes the incomplete beta function (sometimes called the regularized incomplete beta function) defined as

$$I_x(a,b) = \frac{1}{B(a,b)} \cdot \int_0^x t^{a-1}(1-t)^{b-1}dt$$

where $B(a,b)$ is the beta function. `x`, `a` and `b` must be real arrays of the same size, or any can be a scalar. `x` must be in the interval `[0,1]`, inclusive. `a` and `b` must be nonnegative.

See also

`beta`, `betaln`, `gammainc`

Purpose

Logarithm of the beta function.

Synopsis

```
b = betaln(x,y)
```

Description

`b = betaln(x,y)` computes the natural logarithm of the beta function of `x` and `y` without computing the actual beta function. `x` and `y` must be real arrays of the same size, or either can be a scalar.

Example

`betaln(600,600)` computes the logarithm of the beta function where `log(beta(600,600))` would underflow.

See also

`beta`, `betainc`, `gamma`, `gammaln`

Purpose Convert binary strings to decimal integers.

Synopsis

```
d = bin2dec(str)
```

Description `d = bin2dec(str)` converts a string `str` representing a binary number to a decimal integer. `str` can also be a string matrix, in which case `bin2dec` converts each row, or a cell array of strings, in which case `bin2dec` converts each element.

Example `bin2dec('1100')` converts binary number `1100` to its decimal equivalent, `12`.

See also `base2dec`, `hex2dec`, `hex2num`, `dec2base`, `dec2bin`, `dec2hex`, `num2hex`

Purpose

Compute bitwise function of two matrices pointwise.

Synopsis

```
d = bitand(a, b)
d = bitor(a, b)
d = bitxor(a, b)
```

Description

`d = bitand(a, b)` computes the pointwise bitwise AND of the two matrices `a` and `b`.

`d = bitor(a, b)` computes the pointwise bitwise OR of the two matrices `a` and `b`.

`d = bitxor(a, b)` computes the pointwise bitwise XOR of the two matrices `a` and `b`.

For each dimension, `a` and `b` must have the same size or either of them must have size 1. In the latter case, the unit dimension is expanded to the size of the nonunit dimension.

Examples

```
bitand(20, 4)
bitand([16 32 64], [15 31 63])
bitor(15, 16)
bitor([8 16 32], 8)
bitxor(15, 31)
bitxor([0 1], [0 ; 1])
```

See also

`bitcmp`, `bitget`, `bitmax`, `bitset`, `bitshift`

Purpose

Create the bitwise complement.

Synopsis

```
d = bitcmp(a, ndig)
d = bitcmp(u)
```

Description

`d = bitcmp(a, ndig)` returns the bitwise complement of the matrix `a` when treated as a matrix of binary numbers with `ndig` digits.

The sizes of `a` and `ndig` must be identical unless one of them is a scalar; in that case, the scalar is expanded to a matrix of the correct size. All entries of `a` must be nonnegative integers less than `bitmax`, and all entries of `ndig` must be integers between 1 and 53.

`d = bitcmp(u)` returns the bitwise complement of the `uint8` matrix `u`.

See also

`bitand`, `bitor`, `bitxor`, `bitget`, `bitmax`, `bitset`, `bitshift`

Purpose

Extract bit values from a matrix.

Synopsis

```
d = bitget(a, pos)
```

Description

`d = bitget(a, pos)` returns the values of bits `pos` in the matrix `a`.The least significant bit has position 1, and the most significant bit has position 53.

The sizes of `a` and `pos` must be identical unless one of them is a scalar; in that case, the scalar is expanded to a matrix of the correct size. All entries of `a` must be nonnegative integers less than `bitmax`, and all entries of `pos` must be integers between 1 and 53.

See also

`bitand, bitor, bitxor`, `bitcmp`, `bitmax`, `bitset`, `bitshift`

Purpose The largest integer that can be used as an argument to bitwise functions.

Synopsis

```
d = bitmax
```

Description `d = bitmax` returns the largest integer that can be used as an argument to bitwise functions, specifically $2^{53} - 1$.

See also `bitand, bitor, bitxor`, `bitcmp`, `bitget`, `bitset`, `bitshift`

Purpose

Set bit values in matrix.

Synopsis

```
d = bitset(a, pos)
d = bitset(a, pos, val)
```

Description

`d = bitset(a, pos)` returns `a` with the bit(s) in position `pos` set to 1. The least significant bit has position 1, and the most significant bit has position 53.

`d = bitset(a, pos, val)` returns `a` with the bit(s) in position `pos` set to `val`, which must be 0 or 1.The least significant bit has position 1, the most significant bit has position 53.

The sizes of `a` and `pos` must be identical unless one of them is a scalar; in that case, the scalar is expanded to a matrix of the correct size. All elements of `a` must be nonnegative integers less than `bitmax`, and all elements of `pos` must be integers between 1 and 53.

See also

`bitand, bitor, bitxor`, `bitcmp`, `bitget`, `bitmax`, `bitshift`

Purpose

Shift bit values in a matrix.

Synopsis

```
d = bitshift(a, shift)
d = bitshift(a, shift, ndig)
```

Description

`d = bitshift(a, shift)` returns `a` with the bits shifted by `shift` steps. Positive values of `shift` are multiplications by powers of 2, and negative values of `shift` correspond to divisions by powers of 2.

`d = bitshift(a, shift, ndig)` first performs the shift as does `bitshift(a, shift)` but afterwards it zeroes out all bits with positions larger than `ndig`. It thus converts the returned matrix to binary numbers with `ndig` digits.

The sizes of `a`, `shift`, and `ndig` must be identical unless one of them is a scalar; in that case, the scalar is expanded to a matrix of the correct size. All entries of `a` must be nonnegative integers less than `bitmax`, all entries of `shift` must be integers, and all entries of `shift` and `ndig` must be integers between 1 and 53.

See also

`bitand, bitor, bitxor`, `bitcmp`, `bitget`, `bitmax`, `bitset`

Purpose Generate a string of blanks.

Synopsis

```
s = blanks(n)
```

Description `s = blanks(n)` generates a string `s` of `n` blanks.

See also `deblank`

Purpose Create a block-diagonal matrix or cell array.

Synopsis

```
b = blkdiag(a1, a2, ...)
```

Description `b = blkdiag(a1, a2, ...)` returns a block-diagonal matrix with `a1`, `a2` and so forth on the block diagonal. All other elements are assigned the default value for the output type (0 for matrices, [ ] for cell arrays).

All inputs must be 2D.

See also `diag`, `horzcat`, `vertcat`

Purpose Create a colormap with gray scales and a touch of blue.

Synopsis `bone(n)`

Description `bone(n)` returns a colormap with `n` colors. It is a matrix with `n` rows and 3 columns with RGB values for the colors in the colormap. The colors are gray scales and a touch of blue.

See also `colormap`, `cool`, `gray`, `grayprint`, `jet`, `hot`, `hsv`, `pink`, `wavemap`

Purpose

Add a box to 3D axes.

Synopsis

```
box('on')
box('off')
box
box(ax, ...)
```

Description

`box('on')` turns on a box in the current 3D axes.

`box('off')` turns off the box in the current 3D axes.

`box` toggles the box on or off.

`box(ax,...)` adds a box to the axes ax instead of to the current axes.

See also

`grid`

Purpose

Evaluate a built-in function.

Synopsis

```
builtin(name, arg1, ...)
```

Description

`builtin(name, arg1, ...)` evaluates the built-in function `name` with arguments `arg1`, ... and returns the result (if any). This overrides any other definition of name as a variable in the current workspace.

Purpose

Create a button.

Synopsis

```
b = button(text,...)
b = button(...)
```

Description

`b = button(text)` creates a button with the specified text.

In addition to the fixed arguments, additional property-value pairs can be given at the end of the list to further control how the button is created:

TABLE 1-4: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DESCRIPTION
`image`	`iconimage`	An image to display on the button.
`text`	`string`	A text to display on the button.

The function returns a button object that can then be manipulated further using the methods in the following table.

TABLE 1-5: METHODS FOR MANIPULATING A BUTTON OBJECT.

METHOD	DESCRIPTION
`addActionListener(name)`	Specifies that the function with the given name should be run when the button is clicked.
`addActionListenerThread(name)`	Specifies that the function with the given name should be run when the button is clicked. The function will be run in a separate thread. This can be used for operations that run for a long time and need to update graphics while running.
`getText`	Returns the text on the button.
`setText(text)`	Sets the text on the button.

See also the reference entry for `component` to get details on property-value pairs and methods that are valid for all components.

See also

`component`, `checkbox`, `radiobutton`, `togglebutton`

Purpose

Create a button group

Synopsis

```
bg = buttongroup
```

Description

`bg = buttongroup` creates a button group.

A button group is used to get the desired effect for multiple radio buttons or toggle buttons. When one of the buttons in the group is selected, the others are automatically be deselected.

Use `bg.add(button)` after the button group has been created to add radio buttons or toggle buttons whose selection state should be synchronized.

See also

`radiobutton`, `togglebutton`

Purpose Control the camera position.

Synopsis

```
pos = campos;
campos(pos)
campos(ax,...)
```

Description

`pos = campos` returns the camera position for the current axes.

`campos(pos)` where `pos` is a 3 element vector sets the camera position for the current axes.

`campos(ax,...)` uses the axes `ax` instead of the current axes.

See also `camtarget`, `camup`, `camva`

Purpose

Control the camera target.

Synopsis

```
target = camtarget;
camtarget(target)
camtarget(ax,...)
```

Description

`target = camtarget` returns the camera target for the current axes.

`camtarget(target)` where `target` is a 3 element vector sets the camera target for the current axes.

`camtarget(ax,...)` uses the axes `ax` instead of the current axes.

See also

`campos`, `camup`, `camva`

Purpose Control the camera up vector.

Synopsis

```
up = camup;
camup(up)
camup(ax,...)
```

Description

`up= camup` returns the camera up vector for the current axes.

`camup(up)` where `up` is a 3 element vector sets the camera up vector for the current axes.

`camup(ax,...)` uses the axes `ax` instead of the current axes.

See also `campos`, `camtarget`, `camva`

Purpose

Control the camera view angle.

Synopsis

```
angle = camva;
camva(angle)
camva(ax,...)
```

Description

`angle = camva` returns the camera view angle for the current axes.

`camva(angle)` where `angle` is an angle in degrees sets the camera view angle for the current axes.

`camva(ax,...)` uses the axes `ax` instead of the current axes.

See also

`campos`, `camtarget`, `camup`

Purpose

Transform from Cartesian to polar coordinates.

Synopsis

```
[theta,r] = cart2pol(x,y)
[theta,r,z] = cart2pol(x,y,z)
```

Description

`[theta,r] = cart2pol(x,y)` transforms Cartesian 2D coordinates in the arrays `x` and `y` into polar coordinates, where `theta` is the counterclockwise angle in radians from the *x*-axis, and `r` is the radius. `x` and `y` must be the same size or either one can be a scalar.

`[theta,r,z] = cart2pol(x,y,z)` transforms Cartesian 3D coordinates into cylindrical coordinates, where `theta` is the counterclockwise angle in radians from the *x*-axis, `r` is the radius and `z` the height. `x`, `y` and `z` must be the same size or a scalar.

Example

`[t,r,z] = cart2pol([0 1 0 0],[0 0 1 0],[0 0 0 1])` returns the cylindrical coordinates for the points (0,0,0), (1,0,0), (0,1,0) and (0,0,1) in the Cartesian 3D plane, that is points (0,0,0), (0,1,0), (pi/2,1,0) and (0,0,1), respectively.

See also

`pol2cart`, `cart2sph`, `sph2cart`

Purpose Transform from Cartesian to spherical coordinates.

Synopsis

```
[theta,phi,r] = cart2sph(x,y,z)
```

Description `[theta,phi,r] = cart2sph(x,y,z)` transforms Cartesian 3D coordinates into spherical coordinates, where `theta` is the azimuth, `phi` the elevation, and `r` the radius. `theta` and `phi` are in radians. `x`, `y`, and `z` must be the same size or a scalar.

Example `[t,p,r] = cart2sph([0 1 0 0],[0 0 1 0],[0 0 0 1])` returns the spherical coordinates for the points (0,0,0), (1,0,0), (0,1,0) and (0,0,1) in the Cartesian 3D plane, that is points (0,0,0), (0,0,1), (pi/2,0,1) and (0,pi/2,1), respectively.

See also `sph2cart`, `cart2pol`, `pol2cart`

Purpose

Concatenate matrices or cell arrays.

Synopsis

```
b = cat(dim, a1, ...)
```

Description

`b = cat(dim, a1, ...)` concatenates its input arguments along the dimension `dim`. The arguments need not be of the same type; if they differ, the result is the common base type of all the arguments.

`cat(1, a1, ...)` and `cat(2, a1, ...)` are equivalent to `[a1 ; ...]` and `[a1 , ...]` respectively.

See also

`horzcat`, `vertcat`

Purpose

Control mapping of data values to a colormap range.

Synopsis

```
lim = caxis
caxis(lim)
caxis('auto')
caxis('manual')
```

Description

`lim = caxis` returns the data values that map to the minimum and maximum color in the colormap.

`caxis(lim)` sets the data values that should map to the minimum and maximum colors in the colormap.

`caxis('auto')` sets that the color range should automatically be calculated to be the minimum and maximum of the plotted data.

`caxis('manual')` sets that the color range is manually given and should not be updated automatically.

`caxis(ax)` controls the axes `ax` instead of the current axes.

See also

`colormap`

Purpose Change or retrieve current directory.

Synopsis

```
dir = cd
cd(dir)
```

Description `dir = cd` returns the current directory.

`cd(dir)` changes the current directory to `dir`.

See also `pwd`

Purpose

Create empty cell array.

Synopsis

```
c = cell(n)
c = cell(sz)
c = cell(sz1, sz2, ...)
c = cell(javaobj)
```

Description

`c = cell(n)`, for an integer `n`, returns an empty `n` x `n` cell array.

`c = cell(sz)`, for a vector `sz`, returns an empty cell array of size `sz`.

`c = cell(sz1, sz2, ...)` returns an empty cell array of size `(sz1, sz2, ...)`.

`c = cell(javaobj)`, for a Java object `javaobj`, returns a cell array with the same size as `javaobj` where each cell contains one element of `javaobj`.

See also

`struct`, `deal`

Purpose Convert cell array to a matrix.

Synopsis

```
m = cell2mat(c)
```

Description `m = cell2mat(c)` returns a matrix formed by the concatenation of the elements of the cell array `c`. This is possible only if the cell-array elements are of compatible types and the sizes match; all elements with the same i^{th} index must have the same size along dimension i.

Example `cell2mat({[1 2 ; 3 4], [5 ; 6]})` is `[1 2 5 ; 3 4 6]`.

See also `mat2cell`

Purpose

Convert a cell array to a structure.

Synopsis

```
s = cell2struct(c, fields, dim)
```

Description

`s = cell2struct(c, fields, dim)` returns the structure where the dimension `dim` of `c` has been replaced by structure fields and all other dimensions of `c` are transferred to `s`: `size(s) == [csize(1:dim-1) csize(dim+1:end)]` where `csize = size(c)`.

`fields` must be a character array or cell array of character arrays containing `size(c, dim)` elements. These strings are used as fields names in `s`.

See also

`struct2cell`, `fieldnames`

Purpose

Apply a function to the elements of a cell array.

Synopsis

```
r = cellfun('prodofsize', c)
r = cellfun('isclass', c, cla)
r = cellfun(func, c, ...)
```

Description

`r` is a double matrix the same size as `c` where each element is the result of the application of a function to the corresponding element of `c`.

`r = cellfun('prodofsize', c)` results in `r(i) = prod(size(c{i}))` for all `i`.

`r = cellfun('isclass', c, cla)` results in `r(i) = 1` if `c{i}` is of the class `cla`, otherwise `r(i) = 0`.

`r = cellfun(func, c, ...)` results in `r(i) = eval(func, c{i}, ...)` where `func` must be a function that returns a scalar numerical value for any input.

Purpose Convert a character matrix to a cell array of strings.

Synopsis

```
c = cellstr(s)
```

Description `c = cellstr(s)` puts each row of the character matrix `s` in a separate cell in the cell array `c`.

See also `char`, `iscellstr`

Purpose

Convert a value to a character matrix.

Synopsis

```
s = char(c)
s = char(m)
s = char(jobj)
s = char(s1, s2, ...)
```

Description

`s = char(c)`, where `c` is a cell array of strings, returns a character matrix where the *i*th row is `c{i}`.

`s = char(m)`, where `m` is a full matrix, returns a character matrix of the same size as `m` where each element of `m` has been converted to a character.

`s = char(jobj)`, where `jobj` is a `java.lang.String` or an array of `java.lang.String`, returns a matrix where the rows equal the elements of `jobj`.

`s = char(jobj)`, where `jobj` is any other Java object, returns the result of invoking the `toChar()` method on the object. It generates an error if there is no such method.

`s = char(s1, s2, ...)` converts `s1`, `s2`, and so on to character matrices and returns a character matrix where the rows of `s1`, `s2`, and so on are concatenated vertically; the first `size(s1, 1)` rows of `s` are the rows of `s1`, the next `size(s2, 1)` rows of `s` are the rows of `s2`, and so on.

See also

`cellstr`, `ischar`

Purpose

Create a check box.

Synopsis

```
c = checkbox(text,...)
c = checkbox(...)
```

Description

`c = checkbox(text)` creates a check box with the specified text.

A `checkbox` behaves exactly like a `togglebutton` except that it is rendered as a check box. See the reference entry for `togglebutton` for available property values and methods.

See also

`togglebutton`

Purpose

Cholesky factorization.

Synopsis

```
c = chol(x)
[c,p] = chol(x)
```

Description

`c = chol(x)` returns the Cholesky factorization of `x` using LAPACK's DPOTRF and ZPOTRF functions. `c` is an upper triangular matrix such that `x = c'*c`.

`x` is assumed to be symmetric or Hermitian, hence the part below the main diagonal is not used. `x` must be positive definite.

`[c,p] = chol(x)` does not require `x` to be positive definite, but if that is the case, then `c` is the same as above and `p` is `0`. Otherwise, `p` is a positive integer and `c` is a matrix of order `p-1` such that `c'*c = x(1:p-1,1:p-1)`.

Purpose

Shift the indices of a matrix circularly.

Synopsis

```
b = circshift(a, shift)
```

Description

`b = circshift(a, shift)` where `a` is a matrix and `shift` is an integer vector returns a matrix with the same size and type as `a` where the i^{th} index has been shifted circularly with `shift(i)`.

See also

`shiftdim`

Purpose Clear all contents in the current axes.

Synopsis `cla`

Description `cla` removes all graphics objects from the current axes.

See also `clf`, `hold`

Purpose

Add labels to a contour plot.

Synopsis

```
clabel(c)
```

Description

```
clabel(c)
```

`clabel(c)` adds labels to the contour lines specified by the contour matrix `c`. See `contourc` for description of the contour matrix `c`. A marker and a text with the contour level value is placed on each line.

Additional property values pairs can be added at the end of the command to further control the label. Use the 'parent' property to specify what axes to add the labels to and the property values from `text` can to control color, size and font for the labels.

Example

```
[x,y]=meshgrid(linspace(-3,3,50));
z=(x.^2+y.^2).*exp(-x.^2-y.^2)+cos(y)+sin(x);
c=contour(x,y,z);
clabel(c);
```

See also

`contour`, `contour3`, `contourc`

Purpose Get the class of an object.

Synopsis

```
c = class(m)
```

Description `c = class(m)` returns a string containing the class name of `m`.

Purpose

Clear the contents in the command window.

Synopsis

```
clc
```

Description

`clc` clears the contents in the command window and moves the cursor to the upper left corner.

Purpose

Remove variables or functions from the workspace.

Synopsis

```
clear
clear('all')
clear('variables')
clear('functions')
clear(var1, ...)
clear('global', var1, ...)
clear('classes')
clear('classes', cl1, ...)
```

Description

`clear`, `clear('all')`, and `clear('variables')` remove all variables from the workspace.

`clear('functions')` removes all user-defined functions from memory.

`clear(var1, ...)` removes the variables with names `var1`, ... from the workspace. The variable names may contain the wildcard character `*`, which matches any character sequence.

`clear('global', var1, ...)` removes the variables with names `var1`, ... from the global workspace.

`clear('classes')` removes all variables from the workspace and also removes all class definitions for classes that there are no instances of in some other workspace.

`clear('classes', cl1, ...)` only removes the definitions of the classes `cl1`, ...

See also

`mlock`, `munlock`

Purpose Clear all the contents in the current figure.

Synopsis `clf`

Description `clf` removes all the graphics objects from the current figure.

See also `cla`, `hold`

Purpose Current time.

Synopsis

```
c = clock
```

Description `c = clock` returns the current time as a vector of six elements representing, in order: year, month, day, hour, minute, and seconds. All but the seconds field are integers.

See also `etime`, `date`

Purpose

Create a copy of an instance of a user-defined class.

Synopsis

```
copy = clone(obj)
copy = clone
```

Description

`copy = clone(obj)` returns a copy of obj, which must be an instance of a user-defined class.

`copy = clone`, when called from an instance method of a class, returns a copy of the instance object.

See also

`this`

Purpose Close a figure window.

Synopsis

```
close
close(h)
close('all')
```

Description

`close` closes the current figure window.

`close(h)` closes the figure window with handle h.

`close('all')` closes all open figure windows.

See also `delete`

Purpose

Compute range.

Synopsis

```
d = colon(a, b)
```

```
d = colon(a, b, c)
```

Description

`d = colon(a, b)` returns the vector `[a a+1 a+2 ... a+k]` where `k` is the largest integer for which `a <= a+k <= c`. `a` and `b` must be scalars.

`d = colon(a, b, c)` returns the vector `[a a+b a+2b ... a+kb]` where `k` is the largest integer for which `a <= a+kb <= b`. `a`, `b`, and `c` must be scalars.

`colon(a, b)` is equivalent to `a:b` and `colon(a, b, c)` is equivalent to `a:b:c`.

See also

`linspace`, `linspace`

Purpose Assign a colormap to plots and figure windows.

Synopsis

```
colormap(map)
colormap(h,map)
```

Description

`colormap(map)` sets the colormap of the current figure and all plots in the current figure to `map`.

`colormap(h,map)` sets the colormap of the graphics object `h` to `map`. `h` can be a handle to a figure window or to an individual plot.

See also `caxis`

Purpose

Create a combo box.

Synopsis

```
c = combobox(...)
```

Description

`c = combobox(...)` creates a combo box. The values and descriptions for the values in the combo box are specified using the properties in the following table

TABLE 1-6: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DESCRIPTION
`descr`	cell array of strings	The strings to display in the combobox. If not given the strings specified as `items` will be displayed in the combobox.
`items`	cell array of strings	String representing the value corresponding to each entry in the combobox. Can then be used to easily set and get the value of the combobox using strings instead of indices.

The function returns a combobox object that can then be further manipulated using the methods in the following table.

TABLE 1-7: METHODS FOR MANIPULATING A COMBOBOX OBJECT.

METHOD	DESCRIPTION
`addActionListener(name)`	Specifies that the function with the given name should be run when the selection in the combobox changes.
`getSelectedIndex`	Returns an index to the currently selected item in the combobox.
`getValue`	Returns a string corresponding to the currently selected item in the combobox.
`setItems(items)`	Sets the items to display in the combobox by passing a cell array of strings.
`setItems(items,descr)`	Sets the descriptions to display in the combobox and their corresponding values by passing two cell arrays of strings.
`setSelectedIndex(ind)`	Selects the item with the specified index in the combobox.
`setValue(value)`	Selects the item with the specified value in the combobox.

See also the reference entry for `component` to get details on property-value pairs and methods that are valid for all components.

See also `component`, `listbox`

Purpose

Compile C code into a shared library that can be called from COMSOL Script.

Synopsis

```
status = compile(options, ...)
```

Description

`status = compile(options, ...)` compiles one or more C source files and by default links them into a shared library. All options must be strings; any option that ends with `.c` is assumed to be a C source code file. The return value is 0 if compilation succeeded and nonzero if it failed.

The following options can be supplied:

TABLE 1-8: COMPILATION OPTIONS

OPTION	FUNCTION
`-c`	The source code files are compiled but not linked.
`-DSYMBOL`	Defines the preprocessor macro `SYMBOL` when compiling. Equivalent to inserting `#define SYMBOL` in the source code files.
`-DSYMBOL=VALUE`	Assigns the value `VALUE` to the preprocessor macro `SYMBOL` . Equivalent to inserting `#define SYMBOL VALUE` in the source code files.
`-fFILE`	Compilation options are read from `FILE`.
`-g`	Debug information is generated by the compiler.
`-h, -help`	Displays a help text.
`-IDIR`	Adds the directory `DIR` to the include file path.
`-LDIR`	Adds the directory `DIR` to the link directory path.
`-lLIB`	Adds the library `LIB` to the list of libraries to link against.
`-oOUTLIB`	Sets the name of the generated shared library to `OUTLIB`.
`-O`	`Enables optimization.`

Any unrecognized compiler options are passed as arguments to the linker if linking is done.

Example

To compile and link the source code file `myfftlib.c` with optimization enabled:

```
compile -O myfftlib.c
```

Purpose Create a complex matrix.

Synopsis

```
c = complex(a)
c = complex(re, im)
```

Description `c = complex(a)` returns the matrix `a` converted to a complex matrix.

`c = complex(re, im)` returns a complex matrix with real part `re` and imaginary part `im`. The sizes of `re` and im must be identical unless one of them is a scalar; in that case, the scalar is expanded to a matrix of the correct size.

See also `imag`, `real`

Purpose

The following property values can be used when creating all types of components:

TABLE 1-9: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DESCRIPTION
`size`	2 element vector	The preferred width and height for the component. Not needed for most components but can be useful for giving a size to frames, list boxes and scrollpanes.
`tag`	string	The tag to assign to the component. Can be used to access it later on.
`tooltip`	string	The tooltip to display with the component.

The following methods can be used to interact with all types of components after they have been created:

TABLE 1-10: METHODS FOR MANIPULATING ALL TYPES OF COMPONENTS.

METHOD	DESCRIPTION
`addFocusListener(name)`	Specifies that the function with the given name should be run when the component gains or looses focus.
`addMouseListener(name)`	Specifies that the function with the given name should be run when the mouse is moved or clicked over the component.
`getEnabled`	Returns a logical telling if the component is enabled or not.

`getMinimumSize`
`getMaximumSize`
`getPreferredSize`
	Returns the minimum, maximum or preferred size of the component as a 2 element vector with width and height.

`getTag`	Returns the tag of the component.
`getVisible`	Returns a logical telling if the component is visible or not.
`setEnabled(enab)`	Sets if the component is enabled using a logical.

`setMinimumSize(w,h)`
`setMaximumSize(w,h)`
`setPreferredSize(w,h)`
	Sets the minimum, maximum or preferred size of the component by specifying the width and height.

`setTag(tag)`	Sets the tag of the component to the specified string.
`setVisible(vis)`	Sets the visibility of the component using a logical.

Purpose

Get the machine type.

Synopsis

```
type = computer
[type, maxsize] = computer
[type, maxsize, endian] = computer
```

Description

`type = computer` returns the machine type. Possible values are

TABLE 1-11: TYPE CODES FOR MACHINE TYPES

TYPE	INTERPRETATION
GLNX86	Linux on x86
GLNXI64	Linux on Itanium
GLNXA64	Linux on AMD64
SOL2	32-bit Sun
SOL64	64-bit Sun
MAC	PowerPC Macintosh running Mac OS X
MACI	Intel Macintosh running Mac OS X
PCWIN	32-bit Windows
WIN64	64-bit Windows

`[type, maxsize] = computer` also returns the maximum number of bytes a matrix can occupy.

`[type, maxsize, endian] = computer` also returns the endianness: `'B'` for big-endian, and `'L'` for little-endian.

See also

`ispc`, `isunix`

Purpose

The condition number for inversion.

Synopsis

```
c = cond(x)
c = cond(x,p)
```

Description

`c = cond(x)` returns the 2-norm condition number of `x`.

`c = cond(x,p)` returns the `p`-norm condition number of `x` with respect to inversion. That is, the ratio of the largest singular value of `x` to the smallest. (For information about possible values for `p`, see norm)

See also

`condeig`, `svd`

Purpose

The condition number for eigenvalues.

Synopsis

```
c = condeig(A)
[X,LAMBDA,c] = condeig(A)
```

Description

`c = condeig(A)` returns a column vector containing the condition numbers for the eigenvalues of `A`.

`[X,LAMBDA,c] = condeig(A)` also returns `[X,LAMBDA] = eig(A)`. (See `eig` for further information.)

See also

`cond`, `eig`

Purpose

Create a contour plot.

Synopsis

```
contour(Z)
contour(X,Y,Z)
contour(X,Y,Z,lev)
```

Description

`contour(X,Y,Z)` creates a contour plot for a function defined on a grid. `X`, `Y`, and `Z` are matrices of the same size. The function has value `Z(i)` in the grid point `(X(i),Y(i))`. By default 7 contour lines between the minimum and maximum values of `Z` are created.

`contour(X,Y,Z,lev)` creates `lev` contour lines if `lev` is a scalar. If `lev` is a vector it creates contour lines at the values specified in `lev`.

`contour(x,y,Z,...)` when `x` and `y` are vectors can also be used. In that case `X` and `Y` will be created using `[X,Y] = meshgrid(x,y)`.

`contour(Z,...)` uses `x = 1:size(Z,2)` and `y = 1:size(Z,1)`.

`[c,h] = contour(...)` returns the contour matrix, `c`, and a handle, `h`, to the plotted lines. See `contourc` for details about the contour matrix.

Normally contours gets colors from the colormap of the figure plotted into or from a colormap passed to `contour` using the `'colormap'` property.

`contour(...,'lincolor')`, where `'lincolor'` is one of the color strings listed in `plot`, can be used to color all lines using the same color. You can give additional property values from `line` at the end of the command to control color, linewidth, and the axes into which to plot.

Example

```
[x,y] = meshgrid(linspace(-3,3,50));
z = (x.^2+y.^2).*exp(-x.^2-y.^2)+cos(y)+sin(x);
contour(x,y,z);
```

See also

`clabel`, `contour3`, `contourc`, `contourf`

Purpose Create a 3D contour plot.

Synopsis

```
contour3(Z)
contour3(X,Y,Z)
contour3(X,Y,Z,lev)
```

Description `contour3` supports the same syntaxes as `contour`. The difference is that `contour3` draws the contour lines at a Z-coordinate corresponding to the value of the contour level.

See also `clabel`, `contour`, `contourc`

Purpose

Calculate a contour data matrix.

Synopsis

```
c = contourc(Z)
c = contourc(X,Y,Z)
c = contourc(X,Y,Z,lev)
```

Description

`c=contourc(...)` calculates the contour data matrix. The same syntaxes as for `contour` is supported. The contour data matrix, `c`, has two rows with blocks of data for the contour lines. Each block starts with a column with information about that block. The first row in the information column is value for that contour level and the second row is the number of following columns that contains *x*- and *y*-coordinates for that contour line. The *x*- and *y*-coordinates are ordered within each segment so that lines can be drawn directly between them to form the contour lines.

This means that the contour matrix `c` looks as follows:

```
c = [level1 x11 x12 ... x1n1 level2 x21 x22 ... x2n2 ;
     n1     y11 y12 ... y1n1 n2     y21 y22 ... y2n2 ];
```

See also

`clabel`, `contour`, `contour3`

Purpose

Create a filled contour plot.

Synopsis

```
contourf(Z)
contourf(X,Y,Z)
contourf(X,Y,Z,lev)
```

Description

`contourf(X,Y,Z)` creates a filled contour plot for a function defined on a grid. `X`, `Y`, and `Z` are matrices of the same size. The function has value `Z(i)` in the grid point `(X(i),Y(i))`. By default 7 contour lines between the minimum and maximum values of `Z` are created.

`contourf(X,Y,Z,lev)` creates `lev` contour lines and `lev+1` bands if `lev` is a scalar. If `lev` is a vector it creates contour lines at the values specified in `lev`.

`contourf(x,y,Z,...)` where `x` and `y` are vectors can also be used. In that case `X` and `Y` are created using `[X,Y] = meshgrid(x,y)`.

`contourf(Z,...)` uses `x = 1:size(Z,2)` and `y = 1:size(Z,1)`.

`[c,h] = contourf(...)` returns the contour matrix, `c`, and a handle to the plotted lines. See `contourc` for details about the contour matrix.

Filled contours get their colors from the colormap of the figure plotted into or from a colormap passed to `contourf` using the `'colormap'` property.

You can give the additional property `parent` at the end of the command to control the axes into which to plot.

Example

```
[x,y] = meshgrid(linspace(-3,3,50));
z = (x.^2+y.^2).*exp(-x.^2-y.^2)+cos(y)+sin(x);
contourf(x,y,z);
```

See also

`contour`, `contour3`, `contourc`

Purpose

Compute the convolution of two vectors.

Synopsis

```
c = conv(a, b)
```

Description

`c = conv(a, b)` returns the convolution of `a` and `b`, which must be real or complex vectors. This can be used to multiply polynomials in this way: If `a` and `b` contain the coefficients of two polynomials, then `c` contains the coefficients of their product.

See also

`conv2`, `convn`, `deconv`

Purpose

Compute the 2D convolution of two matrices.

Synopsis

```
out = conv2(a, b)
out = conv2(a, b, c)
out = conv2(..., 'full')
out = conv2(..., 'same')
out = conv2(..., 'valid')
```

Description

`c = conv2(a, b)` returns the 2D convolution of `a` and `b`, which must be real or complex matrices.

`c = conv2(a, b, c)` first convolutes the rows of `c` with `a`, then it convolutes the rows of the result with `b`.

See `convn` for an interpretation of the optional shape argument `'full'`, `'same'`, and `'valid'`.

See also

`conv2`, `convn`

Purpose

Compute the nD convolution of two matrices.

Synopsis

```
out = convn(a, b)
out = convn(..., 'full')
out = convn(..., 'same')
out = convn(..., 'valid')
```

Description

`c = convn(a, b)` returns the nD convolution of `a` and `b`, which must be real or complex matrices.

The optional *shape* argument dictates the size of the output matrix. It has the following effect:

TABLE 1-12: SHAPE ARGUMENT INTERPRETATION

SHAPE	INTERPRETATION
`'full'` (default)	The whole output matrix is returned.
`'same'`	The output matrix has the same size as `a`. It is a centered submatrix of the result returned for `'full'`.
`'valid'`	The output matrix only contains the entries that can be computed without assuming that `b` is extended with zeros when indexed out of bounds. The size of the output is `max(size(a)-size(b)+1, 0)`.

See also

`conv`, `conv2`

Purpose Create a colormap with different shades of cyan and magenta.

Synopsis `cool(n)`

Description `cool(n)` returns a colormap with `n` colors. It is a matrix with `n` rows and 3 columns with RGB values for the colors in the colormap. The colors are different shades of cyan and magenta.

See also `colormap`, `bone`, `gray`, `grayprint`, `jet`, `hot`, `hsv`, `pink`, `wavemap`

Purpose

Correlation coefficients.

Synopsis

```
R = corrcoef(x,...)
R = corrcoef(x,y,...)
[R,P] = corrcoef(...)
[R,P,L,U] = corrcoef(...)
```

Description

`R = corrcoef(x)` returns the of correlation coefficients of `x`. `x` is a matrix where each row is an observation and each column a variable. `R` is a matrix such that each element $R(i,j) = \frac{C(i,j)}{\sqrt{C(i,i) \cdot C(j,j)}}$, where C is the covariance matrix of `x` (see `cov`).

`corrcoef(x,y)` is equivalent to `corrcoef([x(:),y(:)])`.

`[R,P] = corrcoef(...)` also returns the matrix `P` where each element is the p-value representing the probability of getting a correlation as large as the observed value, given that the null hypothesis is true. Hence, a small p-value means that the corresponding correlation is significant. `corrcoef` computes `P` using Student's t-test on the statistic $t = R \cdot \sqrt{\frac{n-2}{1-R^2}}$, where `n` is the number of samples.

`[R,P,L,U] = corrcoef(...)` also returns lower (`L`) and upper (`U`) bounds for a confidence interval specified by the `alpha` property (see below). Default is 95%.

In addition to the fixed arguments, property-value pairs can be given at the end of the argument list:

TABLE 1-13: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
`alpha`	A real value between 0 and 1	`0.05`	Specifies the confidence level of 100*(1-alpha)%. Hence default gives 95% confidence intervals.
`rows`	string	`'all'`	A string with the value `'all'` (all rows are used), `'complete'`, (rows containing NaN are ignored) or `'pairwise'` (rows with no NaN values in column *i* or *j* are used to compute `R(`*i*`,`*j*`)`)

Examples

```
a = randn(5);
a(1,2) = NaN; a(3,5)=NaN;
r_all=corrcoef(a)
r_comp=corrcoef(a,'row','complete')
r_row=corrcoef(a,'row','pairwise')
```

See also

`cov`

Purpose

Covariance matrix.

Synopsis

```
c = cov(x)
c = cov(x,y)
c = cov(...,n)
```

Description

`c = cov(x)` and `cov(x,0)` return the covariance matrix of `x` using normalization by m-1, where m is the number of observations. `x` is a matrix where each row is an observation and each column a variable. The diagonal of `c` contains the variance of each column of x. If `x` is a vector, `cov(x)` is the variance of `x`.

`c = cov(x,y)` and `c = cov(x,y,0)` return the covariance matrix of `x` and `y` using normalization by m-1. This is equivalent to `cov([x(:),y(:)])`.

`c = cov(...,1)` returns the covariance matrix using normalization by `m`.

`cov` removes the mean from each column before calculation.

Example

```
a = [0 1 1;2 3 4;1 2 3];
c = cov(a);
v = diag(c)';
v1 = var(a); %Identical to v
```

See also

`var`, `corrcoef`

Purpose CPU time used.

Synopsis

```
c = cputime
```

Description `c = cputime` returns the amount of CPU time used in seconds.

See also `tic`, `toc`

Purpose Cross product.

Synopsis

```
c = cross(u,v)
c = cross(u,v,dim)
```

Description `c = cross(u,v)` computes the cross product of the arrays `u` and `v`, both of which must be either vectors with three elements or n-dimensional arrays of equal size with at least one dimension of length three. The cross product is computed along the first such dimension.

`c = cross(u,v,dim)` returns the cross product along the dimension `dim`.

Example `x = [1 -1 3];y=[4 3 2];cross(x,y)` gives the cross product of `x` and `y`, that is `[-11 10 7]`

See also `dot`

Purpose Compute the complex conjugate transpose of a matrix.

Synopsis

```
d = ctranspose(a)
```

Description `d = ctranspose(a)` computes the complex conjugate transpose of the matrix `a`.

`ctranspose(a)` is equivalent to `a'`.

See also `transpose`

Purpose

Computes the cumulative product of array elements.

Synopsis

```
y = cumprod(x)
y = cumprod(x,dim)
```

Description

`y = cumprod(x)` computes the cumulative product of `x`. `y` is the same size as `x` and contains the cumulative product of the elements along the first nonsingleton dimension of `x`.

`y = cumprod(x,dim)` computes the cumulative product of the elements along the dimension `dim` of `x`.

Examples

```
x = [0 2 3;-3 1 3;2 4 5];
cumprod(x) returns [0, 2, 3 ; 0, 2, 9 ; 0, 8, 45]
cumprod(x,2) returns [0, 0, 0 ; -3, -3, -9 ; 2, 8, 40]
```

See also

`prod`, `sum`, `cumsum`

Purpose

Computes the cumulative sum of an array.

Synopsis

```
y = cumsum(x)
y = cumsum(x,dim)
```

Description

`y = cumsum(x)` computes the cumulative sum of `x`. `y` is the same size as `x` and contains the cumulative sum along the first nonsingleton dimension of `x`.

`y = cumsum(x,dim)` computes the cumulative sum of the elements along the dimension `dim` of `x`.

Examples

```
x = [0 2 3;-3 1 3;2 4 0];
```

`cumsum(x)` returns `[0, 2, 3 ; -3, 3, 6 ; -1, 7, 6]`.

`cumsum(x,2)` returns `[0, 2, 5 ; -3, -2, 1 ; 2, 6, 6]`.

See also

`sum`, `prod`, `cumprod`

Purpose

Cumulative trapezoidal numerical integration.

Synopsis

```
z = cumtrapz(y)
z = cumtrapz(x,y)
z = cumtrapz(y,dim)
z = cumtrapz(x,y,dim)
```

Description

`z = cumtrapz(y)` computes the cumulative integral of `y` using the trapezoidal method with unit spacing. (To compute the integral for different spacing, multiply `z` by the spacing increment.) `z` is the same size as `y` and contains the cumulative integral along the first nonsingleton dimension of `y`.

`z = cumtrapz(x,y)` computes the cumulative integral of `y` with respect to `x`. `x` must be a vector with the same length as the first nonsingleton dimension of `y`. Alternatively, both `x` and `y` must be vectors of equal length.

`z = cumtrapz(y,dim)` or `z = cumtrapz(x,y,dim)` integrates across dimension `dim` of `y`. If given, `x` must be a vector with the same length as `y` along the dimension `dim`.

Examples

```
y = reshape(0:11,3,4);
```

`cumtrapz(y)` returns `[0 0 0 0;0.5 3.5 6.5 9.5;2 8 14 20]`

`cumtrapz(y,2)` returns `[0 1.5 6 13.5;0 2.5 8 16.5;0 3.5 10 19.5]`

See also

`trapz`

Purpose

Solve a stiff ordinary differential equation.

Synopsis

```
[t, y] = daspk(f, tlist, y0)
[t, y] = daspk(f, tlist, y0, options, ...)
```

Description

`[t, y] = daspk(f, tlist, y0)` solves ODEs and DAEs of the form $M(t,y)y' = f(t,y)$, in both cases with the initial value $y(t(1)) = y_0$. `f` is the name of a function such that `f(t, y)` returns a vector when `t` is a scalar and `y` is a column vector. If `tlist` has a length two, then it is the interval over which the ODE is to be solved, otherwise it gives the times at which the solution is requested. `tlist` must be strictly increasing or decreasing. On return, `t` is a column vector containing the times, and `y` is a matrix where the rows contain the corresponding solutions.

`[t, y] = daspk(f, tlist, y0, options)` allows for supplying options to the ODE solver. `options` is a structure returned by `odeset`. If it is empty, default options are used.

`[t, y] = daspk(f, tlist, y0, options, farg1, ...)` results in `f` being invoked with `f(t, y, farg1, ...)`.

Examples

To solve $\frac{dy}{dt} = y + \sin t$ with $y(0) = 5$:

```
f = inline('y+sin(t)', 't', 'y');
[t y] = daspk(f, [0 2], 5);
```

To solve the Lotka-Volterra equation, first create a function `lv.m` that defines the equation:

```
function ydot = lv(t, y)
ydot = [y(1)-y(1).*y(2) ; -y(2)+y(1).*y(2)];
```

and then solve the ODE with

```
[t y] = daspk('lv', [0 10], [2 ; 1]);
```

See also

`odeget`, `odeset`

Purpose	Current date.
Synopsis	`d = date`
Description	`d = date` returns the current date as a string in the format dd-mmm-yyyy.
See also	`clock`, `etime`

Purpose

Remove breakpoints.

Synopsis

```
dbclear
dbclear('all')
dbclear(line)
dbclear(func)
dbclear(func, line)
dbclear('if', 'error')
dbclear('if', 'caught', 'error')
```

Description

`dbclear` or `dbclear('all')` removes all breakpoints.

`dbclear(line)` removes the breakpoint set on line `line` of the function currently being debugged. This syntax can be used only in Debug mode.

`dbclear(func)` removes all breakpoints from the function called `func`.

`dbclear(func, line)` removes the breakpoint on line `line` in the function called `func`.

`dbclear('if', 'error')` removes the breakpoint set on any uncaught error that occurs.

`dbclear('if', 'caught', 'error')` removes the breakpoint set on any error that occurs.

See also

`dbcont`, `dbdown`, `dbquit`, `dbstack`, `dbstatus`, `dbstep`, `dbstop`, `dbtype`,`dbup`

dbcont

Purpose Resume execution when in debug mode.

Synopsis

```
dbcont
```

Description `dbcont` resumes execution from the point where a breakpoint triggered and Debug mode was entered. This command has no effect outside Debug mode.

See also `dbclear`, `dbdown`, `dbquit`, `dbstack`, `dbstatus`, `dbstep`, `dbstop`, `dbtype`, `dbup`

Purpose

Move down in debug call stack.

Synopsis

```
dbdown
dbdown(steps)
```

Description

`dbdown` changes the debug workspace to the child of the current debug workspace, i.e., the workspace created from the current workspace.

`dbdown(steps)` is equivalent to `steps` calls to `dbdown` without arguments.

This function can only be used in Debug mode.

See also

`dbclear`, `dbcont`, `dbquit`, `dbstack`, `dbstatus`, `dbstep`, `dbstop`, `dbtype`, `dbup`

Purpose	Terminate the script being executed and leave Debug mode.
Synopsis	`dbquit`
Description	`dbquit` terminates the script being executed and leaves Debug mode.
See also	`dbclear`, `dbdown`, `dbcont`, `dbstack`, `dbstatus`, `dbstep`, `dbstop`, `dbtype`,`dbup`

Purpose Display the function-call stack.

Synopsis

```
dbstack
```

Description `dbstack` displays the function-call stack with the most recently entered function displayed first. `dbstack` can be used only in Debug mode.

See also `dbclear`, `dbdown`, `dbcont`, `dbquit`, `dbstatus`, `dbstep`, `dbstop`, `dbtype`, `dbup`

dbstatus

Purpose Display all breakpoints that are set.

Synopsis

```
dbstatus
```

Description `dbstatus` displays all breakpoints and other conditions where execution of a function should be stopped and Debug mode entered.

See also `dbclear`, `dbdown`, `dbcont`, `dbquit`, `dbstack`, `dbstep`, `dbstop`, `dbtype`, `dbup`

Purpose

Step to the next line of source code.

Synopsis

```
dbstep
dbstep('in')
dbstep('out')
```

Description

`dbstep` resumes execution at the current breakpoint and steps to the line of source code in the current function.

`dbstep('in')` resumes execution at the current breakpoint and steps to the next line of source code in the current function or a function being called.

`dbstep('out')` resumes execution at the current breakpoint and steps until the function currently being executed has returned.

This function can be used only in Debug mode.

See also

`dbclear`, `dbcont`, `dbdown`, `dbquit`, `dbstack`, `dbstatus`, `dbstop`, `dbtype`, `dbup`

Purpose

Set a breakpoint.

Synopsis

```
dbstop(func)
dbstop(func, line)
dbstop(line)
dbstop('if', 'error')
dbstop('if', 'caught', 'error')
```

Description

`dbstop(func)` sets a breakpoint at the entry of the function called `func`.

`dbstop(func, line)` sets a breakpoint on line `line` of the function called `func`.

`dbstop(line)` sets a breakpoint on line `line` of the function currently being debugged.

`dbstop('if', 'error')` sets a breakpoint on any uncaught error that occurs.

`dbstop('if', 'caught', 'error')` sets a breakpoint on any caught error that occurs, i.e., any error that occurs within one or more try-catch blocks.

Debug mode is entered when the conditions for a breakpoint are triggered.

See also

`dbclear`, `dbcont`, `dbdown`, `dbquit`, `dbstack`, `dbstatus`, `dbstep`, `dbtype`,`dbup`

Purpose

Display source code of a function.

Synopsis

```
dbtype
dbtype(range)
dbtype(func)
dbtype(func, range)
```

Description

`dbtype` displays the source code around the breakpoint of the function currently being debugged. This can only be done in Debug mode.

`dbtype(range)`, where `range` is a string containing a line number or a range of line numbers such as `'11:47'`, displays a line number range of the source code of the function currently being debugged. This can only be done in debug mode.

`dbtype(func)` displays the source code of the function called `func`.

`dbtype(func, range)` displays a range of line numbers for the source code of the function called `func`.

See also

`dbclear`, `dbcont`, `dbdown`, `dbquit`, `dbstack`, `dbstatus`, `dbstep`, `dbstop`, `dbup`, `type`

Purpose Move up in debug call stack.

Synopsis

```
dbup
dbup(steps)
```

Description `dbup` changes the debug workspace to the parent of the current debug workspace, i.e., the workspace from which the current workspace was created.

`dbup(steps)` is equivalent to steps calls to dbup without arguments.

This function can only be used in Debug mode.

See also `dbclear`, `dbcont`, `dbdown`, `dbquit`, `dbstack`, `dbstatus`, `dbstep`, `dbstop`, `dbtype`

Purpose

Distribute function inputs to several output variables.

Synopsis

```
[out1, out2, ...] = deal(in)
[out1, out2, ...] = deal(in1, in2, ...)
```

Description

`[out1, out2, ...] = deal(in)` assigns `in` to `out1`, `out2`, and so on.

`[out1, out2, ...] = deal(in1, in2, ...)` assigns `in1` to `out1`, `in2` to `out2`, and so on. The number of inputs and outputs must be identical.

Example

`deal` is most commonly used together with cell arrays such as in this example: `d = {2 3 5}; [a b c] = deal(d{:});`, which assigns 2 to `a`, 3 to `b`, and 5 to `d`.

Purpose Remove trailing blanks.

Synopsis

```
s = deblank(s)
```

Description `s = deblank(s)` removes all trailing blanks from `s`, which can be a either a string (in which case `deblank` removes all trailing blanks from `s`) or a cell array of strings (in which case `deblank` removes all trailing blanks from each element of `s`).

Example

```
c = {'blue   ','green';'red      ','  yellow'};
deblank(c) returns {'blue', 'green' ; 'red', '  yellow'}
```

See also `blanks`

Purpose

Convert decimal integers to strings in a specific base.

Synopsis

```
s = dec2base(d,b)
s = dec2base(d,b,n)
```

Description

`s = dec2base(d,b)` converts an array of nonnegative integers to string representations in base `b`, where `b` must be an integer between `2` and `36`, inclusive. `s` is a character matrix where each row represents one number.

`s = dec2base(d,b,n)` converts `d` into strings with at least `n` characters by padding with zeros.

Example

`dec2base(210,5,9)` converts `210` in base 10 to `1320` (represented by the string `'000001320'`) in base 5.

See also

`base2dec`, `bin2dec`, `hex2dec`, `hex2num`, `dec2bin`, `dec2hex`, `num2hex`

Purpose Convert decimal integers to binary strings.

Synopsis

```
s = dec2bin(d)
s = dec2bin(d,n)
```

Description `s = dec2bin(d)` converts an array of nonnegative integers to binary string representations. `s` is a character matrix where each row represents one number.

`s = dec2bin(d,n)` converts `d` into strings with at least `n` characters by padding with zeros.

Example `dec2bin(210,9)` converts `210` to its binary equivalent `11010010` (represented by the string `'011010010'`).

See also `base2dec`, `bin2dec`, `hex2dec`, `hex2num`, `dec2base`, `dec2hex`, `num2hex`

Purpose

Convert decimal integers to hexadecimal strings.

Synopsis

```
s = dec2hex(d)
s = dec2hex(d,n)
```

Description

`s = dec2hex(d)` converts an array of nonnegative integers to hexadecimal string representations. `s` is a character matrix where each row represents one number.

`s = dec2hex(d,n)` converts `d` into strings with at least `n` characters by padding with zeros.

Example

`dec2hex(44562)` converts `44562` to its hexadecimal equivalent `AE12` (represented by the string `'AE12'`).

See also

`base2dec`, `bin2dec`, `hex2dec`, `hex2num`, `dec2base`, `dec2bin`, `num2hex`

Purpose

Compute the deconvolution of two vectors.

Synopsis

```
q = deconv(a, b)
[q, r] = deconv(a, b)
```

Description

`q = deconv(a, b)` returns the deconvolution of `a` and `b`, which must be real or complex vectors. This can be used to divide polynomials in this way: If `a` and `b` contain the coefficients of two polynomials, then `q` contains the coefficients of their product. This interpretation holds if the coefficients are listed in decreasing degree, that is., `a(end)` is the constant term and `a(1)` is the highest coefficient.

`[q, r] = deconv(a, b)` also returns the remainder in the polynomial division.

See also

`deconv`

Purpose

Discrete Laplacian.

Synopsis

```
l = del2(u)
l = del2(u,h)
l = del2(u,hx,hy)
l = del2(u,hx,hy,hz,...)
```

Description

`l = del2(u)` computes the discrete Laplacian of `u`. When `u` is a matrix, `l` is a discrete approximation of

$$\frac{\nabla^2 u}{4} = \frac{1}{4}\left(\frac{d^2 u}{dx^2} + \frac{d^2 u}{dy^2}\right)$$

`l` is the same size as `u`, with each element equal to the difference between corresponding element of `u` and the average of its four neighbors. When `u` is an nD array, `l` is an approximation of

$$\frac{\nabla^2 u}{2n}$$

where n is the number of dimensions of `u`.

`l = del2(u,h)` computes the discrete Laplacian of `u` using spacing `h`, where `h` is a scalar.

`l = del2(u,hx,hy)` computes the discrete Laplacian of `u` using the spacing given by `hx` and `hy`. `u` must be 2D, while `hx` and `hy` must be either scalars (in which case they specify spacing between points in the x and y direction, respectively) or vectors (in which case they specify the coordinates of the points in their respective directions). If either `hx` or `hy` is a vector, its length must match the corresponding dimension of `u`.

`l = del2(u,hx,hy,hz,...)` computes the discrete Laplacian of `u` when `u` is an n-dimensional array, and uses the spacing given by `hx`, `hy`, `hz`, and so on.

See also

`diff`, `gradient`

Purpose

Delaunay triangulation.

Synopsis

```
t = delaunay(x,y)
t = delaunay(x,y,bnd)
```

Description

`t = delaunay(x,y)` returns a Delaunay triangulation of the points in the vectors `x` and `y`, that is, a set of triangles such that no points are contained in any triangle's circumcircle. `t` is a matrix where each row contains the indices in `x` and `y` that define one triangle.

`t = delaunay(x,y,bnd)` also uses boundary element information contained in `bnd`, a `4xn` matrix, where `n` is the number of elements. The first two rows contain the indices of boundary element corners and rows three and four contain up and down subdomains, respectively.

Example

```
x = [0 0 3 3];
y = [0 1 0 1];
t = delaunay(x,y);
trimesh(t,x,y)
```

Coordinates that define two intersecting ellipses and a rectangle (in no particular order):

```
p1 = [-10   0   0   0   6   8   8  10  10  10  16  16  26;
        0 -10   0  10   0  -6   6   0   8  10 -10  10   0];

p2 = [  7  10  -7  -7   6   7   9  12  23  23   0  -4  -4;
       10   5  -7   7   8   4   4   9  -7   7  -4  -2   3];

p3 = [  4   4   2   8   8  19  14  18  17  14  13;
        6   3   5   8   2   0  -3  -5   5   2   7];

p = [p1,p2,p3];
```

Boundary information:

```
b1 = [ 3   3   4  14   5   8  15   9   1  16;
       4   5  14  10   8  15   9  10  16   2;
       2   3   1   1   5   7   7   4   2   2;
       3   2   4   4   6   8   8   1   1   1];

b2 = [ 1  17   2   4  18   5   5  19   6   6;
      17   4   6  18   7   6  19   7   8  11;
       1   1   2   4   4   6   3   3   6   8;
       2   2   1   3   3   2   5   5   8   1];

b3 = [ 7  20   7   9  21  11  22  12  23;
      20   8   9  21  12  22  13  23  13;
       7   7   4   1   1   8   8   1   1;
       5   5   7   8   8   1   1   8   8];
```

```
bnd = [b1,b2,b3];

t = delaunay(p(1,:),p(2,:));
tbnd = delaunay(p(1,:),p(2,:), bnd);

h = gca;
set(h,'xlim',[-30 59])
set(h,'ylim',[-20 20])
trimesh(t,p(1,:),p(2,:), 'parent',h);

figure
h = gca;
set(h,'xlim',[-30 59])
set(h,'ylim',[-20 20])
trimesh(tbnd,p(1,:),p(2,:), 'parent',h);
```

See also delaunay3, trimesh

Purpose

3D Delaunay triangulation.

Synopsis

```
t = delaunay3(x,y,z)
t = delaunay3(x,y,z,bnd)
```

Description

`t = delaunay3(x,y,z)` returns a 3D Delaunay triangulation of the points in the vectors `x`, `y`, and `z`, that is, a set of tetrahedrons such that no points are contained in any tetrahedron's circumsphere. `t` is a matrix where each row contains the indices in `x`, `y`, and `z` that define one tetrahedron.

`t = delaunay3(x,y,z,bnd)` also uses boundary element information contained in `bnd`, a `5`-by-`n` matrix, where `n` is the number of elements. The first three rows contain the indices of boundary element corners and rows three and four contain up and down subdomains, respectively.

Example

```
x = [0, 1, 0, 1, 0, 1, 0, 1];
y = [0, 0, 2, 2, 0, 0, 2, 2];
z = [0, 0, 0, 0, 3, 3, 3, 3 ];
t = delaunay3(x,y,z);
```

For an example using boundary information, see the 2D example under `delaunay`.

See also

`delaunay`, `trimesh`

Purpose

Delete files or graphics objects.

Synopsis

```
delete(h)
delete(filename)
```

Description

`delete(h)` deletes all graphics objects in the array of handles `h`. For entries that are handles to a figure window, the corresponding window is closed.

`delete(filename)` deletes the file `filename`.

Purpose	Determinant of a square matrix.
Synopsis	`det(A)`
Description	`det(A)` returns the determinant of a square matrix `A`. To test for singular matrices, use `cond` instead of `det`.
Example	`det([2 -3 1; 4 -2 2;1 1 3])` returns 20.
See also	`cond`

Purpose

Extract diagonal from a matrix or create a diagonal matrix.

Synopsis

```
d = diag(v)
d = diag(v, k)
v = diag(m)
v = diag(m, k)
```

Description

`d = diag(v)`, for a vector `v`, returns a matrix with `v` on the diagonal

`d = diag(v, k)`, for a vector `v`, returns a matrix with `v` on the *k*th diagonal. `k=0` is the diagonal, `k = 1` is the superdiagonal, and so on.

`v = diag(m)`, for a matrix `m`, returns a vector containing the elements on the diagonal of `m`.

`v = diag(m, k)`, for a matrix `m`, returns a vector containing the elements on the *k*th diagonal of `m`.

See also

`tril, triu`

Purpose

Create a dialog box.

Synopsis

```
d = dialog(title,...)
```

Description

`d = dialog(title)` creates a dialog box with the specified title.

In addition to the fixed arguments, additional property-value pairs can be given at the end of the function to further control how the dialog box is created:

TABLE 1-14: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DESCRIPTION
`modal`	string	A string with the value `'on'` or `'off'` telling if a modal dialog box should be created. The default is to create a nonmodal dialog box.
`parent`	frame	The frame that is the parent to this dialog box.
`position`	2-element vector	The position on the screen for the upper left corner of the dialog box.
`size`	2-element vector	The size of the dialog box. If not given the dialog box will be packed to fit the size of the components that have been added to it.

The function returns a `dialog` object that can be manipulated further using the methods in the following table:

TABLE 1-15: METHODS FOR MANIPULATING A DIALOG BOX.

METHOD	DESCRIPTION
`addMenu(menu)`	Adds the specified menu at the end of the main menu bar of the dialog box.
`close`	Closes the dialog box.
`getParent`	Returns the `frame` that is the parent of this dialog box.
`getSize`	Returns the size of the dialog box as a 2-element vector with width and height.
`setSize(width,height)`	Sets the size of the dialog box.
`show`	While the dialog box is being created it is invisible. Call the show method after adding all components to it to show it on screen.

The methods for `panel` are also available for `dialog`, thereby allowing you to add panels and components to a dialog box.

See also the reference entry for `component` for property-value pairs and methods that are valid for all components.

See also `component`, `frame`, `panel`

Purpose

Save activity on the command line to a text file.

Synopsis

```
diary(filename)
diary('on')
diary('off')
```

Description

`diary(filename)` starts saving all input and output on the command line to the file `filename`.

`diary('off')` temporarily turns of logging and flushes the file.

`diary('on')` turns logging back on again.

Purpose

Compute the difference of an array.

Synopsis

```
y = diff(x)
y = diff(x,n)
y = diff(x,n,dim)
```

Description

`y = diff(x)` computes the difference between adjacent elements of `x` along the first nonsingleton dimension. When `x` is a vector, `y` is the difference between adjacent elements. When `x` is a matrix, `y` is a matrix containing the differences between adjacent rows of `x`.

`y = diff(x,n)` computes the n^{th} order difference of `x`.

`y = diff(x,n,dim)` computes the n^{th} order difference of `x` along the dimension `dim`.

Examples

```
a = [1 4 10 12;0 3 -10 20;2 -1 3 4];
diff(a) returns [-1, -1, -20, 8 ; 2, -4, 13, -16].
diff(a,2,1) returns [3, -3, 33, -24]
```

See also

`del2`, `gradient`

Purpose

Get a list of the files in a directory.

Synopsis

```
dir
dir(path)
f = dir
f = dir(d)
```

Description

`dir` displays the files in the current directory.

`dir(path)` displays the files in the path `path`. The path can contain the wildcard character `*`, which matches any character sequence.

`f = dir` returns a structure array with one element for each file in the current directory. It has the following fields:

FIELD	CONTENTS
`name`	Name.
`date`	Creation date.
`bytes`	Number of bytes occupied.
`isdir`	`true` if directory, `false` otherwise.

`f = dir(path)` returns a struct array with one element for each file in the path `path`.

See also

`cd`, `pwd`

Purpose Display a value.

Synopsis `disp(v)`

Description `disp(v)` displays the value of the variable or expression `v`.

See also `display`

display

Purpose Display a value.

Synopsis `display(v)`

Description `display(v)` displays the value of the variable or expression `v`.

See also `disp`

Purpose

Read a delimited file.

Synopsis

```
out = dlmread(filename)
out = dlmread(filename, delimiter)
out = dlmread(filename, delimiter, range)
out = dlmread(filename, delimiter, row, col)
```

Description

`out = dlmread(filename)` reads the file `filename` and returns a matrix where each row contains a row of the file. The delimiter used, if any, is guessed from the contents of the file; the default is to use whitespace as delimiter. The elements of the matrix must be real or complex numbers.

`out = dlmread(filename, delimiter)` uses the character `delimiter` as delimiter between elements.

`out = dlmread(filename, delimiter, range)` reads a part of the file. If `range` is a vector of the form `[R1 C1 R2 C2]`, then rows `R1..R2` and columns `C1..C2` are read; row and column numbers are 0-based. `range` can also be a string in spreadsheet notation; for example, `'C4..H7'` selects rows 4–7 and columns 3–8.

`out = dlmread(filename, delimiter, row, col)` ignores rows and columns with numbers less than `row` and `col`, respectively; row and column numbers are 0-based.

See also

`dlmwrite, strread, textread`

Purpose

Write a delimited file.

Synopsis

```
dlmwrite(filename, data)
dlmwrite(filename, data, delim)
dlmwrite(filename, data, delim, row, col)
dlmwrite(filename, data, ...)
dlmwrite(filename, data, '-append', ...)
```

Description

`dlmwrite(filename, data)` writes the matrix `data` as a comma-separated text file to `filename`.

`dlmwrite(filename, data, delim)` uses `delim` as delimiter.

`dlmwrite(filename, data, delim, row, col)` uses `delim` as delimiter. The data is preceded by `row` empty rows; each row begins with `col` spaces.

`dlmwrite(filename, data, ...)` accepts the following property/value pairs:

TABLE 1-16: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DESCRIPTION
`coffset`	Integer	Number of spaces preceding the data on each row.
`delimiter`	Character	Element separator.
`newline`	`'pc'` or `'unix'`	Newline convention.
`precision`	Integer or string	Number or significant digits or format string of the form used by `fprintf` and `sprintf`.
`roffset`	Integer	Number of empty lines preceding data.

`dlmwrite(filename, data, '-append', ...)` appends the data to the file instead of over-writing it.

Example

```
data = reshape(sin(1:9), 3, 3);
dlmwrite('TABLE', data, 'delimiter', ':', 'precision', 2)
```

creates a file called `TABLE` with the following contents:

```
0.84:-0.76:0.66
0.91:-0.96:0.99
0.14:-0.28:0.41
```

See also

`dlmread`, `fprintf`, `sprintf`

Purpose

Simulate a discrete-time state space model.

Synopsis

```
y = dlsim(A, B, C, D, U)
y = dlsim(A, B, C, D, U, x0)
[y x] = dlsim(A, B, C, D, U)
[y x] = dlsim(A, B, C, D, U, x0)
```

Description

`[y x] = dlsim(A, B, C, D, U)` simulates the state space model

$$x_{n+1} = Ax_n + Bu_n$$
$$y_n = Cx_n + Du_n$$

using the input `u`. The input and output arguments have the following dimensions:

INPUT/OUTPUT ARGUMENT	DIMENSIONS
`A`	`nx-by-nx`
`B`	`nx-by-nu`
`C`	`ny-by-nx`
`D`	`ny-by-nu`
`U`	`N-by-nu`
`x0`	`nx-by-1` (default `zeros(nx,1)`)
`y`	`N-by-ny`
`x`	`N-by-nx`

`dlsim(..., x0)` uses `x0` as initial state.

See also

`filter`

Purpose Run a DOS command.

Synopsis

```
status = dos(cmd)
[status output] = dos(cmd)
```

Description `status = dos(cmd)` runs the DOS command `cmd` in the operating system and returns the exit code, which is 0 if the execution was successful and nonzero otherwise.

`[status output] = dos(cmd)` runs the DOS command `cmd` and returns any output to the standard output stream in `output`.

`dos(cmd)` is equivalent to `system(['cmd.exe /C ' cmd])`.

See also `system`, `unix`

Purpose

Dot product.

Synopsis

```
c = dot(u,v)
c = dot(u,v,dim)
```

Description

`c = dot(u,v)` returns the scalar product of the arrays `u` and `v`. Both arrays must be vectors with the same length or n-dimensional arrays of equal size, in which case the scalar product is computed along the first nonsingleton dimension of `u` and `v`.

`c = dot(u,v,dim)` returns the scalar product along the dimension `dim`.

Example

`x = [1 -1 3];y=[4 3 2];dot(x,y)` gives the dot product of `x` and `y`, that is `7`.

See also

`cross`

Purpose Convert a value to a real or complex matrix.

Synopsis

```
d = double(v)
```

Description `d = double(v)` returns a matrix the same size as `v` where each element has been converted to a real or complex number.

If `v` is a Java object but not a subclass of `java.lang.Number`, then `v.toDouble` is invoked to do the conversion. It generates an error if no such method exists.

Purpose

Flush drawing to the screen.

Synopsis

```
drawnow
```

Description

When you draw several plots in a row in a script, the screen is not automatically repainted after each of them.

A flush happens when control returns to the prompt after running a script or when calling getFrame on a movie-generation object.

You can also force a flush in a script to achieve repaints while the script is running by calling the `drawnow` function.

Purpose

Enable/disable echoing of lines executed in functions and scripts.

Synopsis

```
echo('on')
echo('off')

echo(func)
echo(func, 'on')
echo(func, 'off')

echo('on', 'all')
echo('off', 'all')
```

Description

`echo('on')` and `echo('off')` enable and disable, respectively, echoing of all lines executed in user-defined scripts.

`echo(func, 'on')` and `echo(func, 'off')` enable and disable, respectively, echoing of all lines executed in the function called `func`.

`echo(func)` toggles echoing of all lines executed in the function called `func`.

`echo('on', 'all')` and `echo('off', 'all')` enable and disable, respectively, echoing of all lines executed in scripts and functions.

Purpose

Compute eigenvalues and eigenvectors.

Synopsis

```
eig(A)
[X,LAMBDA]=eig(A)
eig(A,B)
[X,LAMBDA]=eig(A,B)
```

Description

`eig(A)` computes the eigenvalues of the square matrix `A`.

`[X,LAMBDA]=eig(A)` computes the right eigenvectors `X` and eigenvalues of the square matrix `A`, so that `A*X=X*LAMBDA`. `LAMBDA` is a diagonal matrix with the eigenvalues on the diagonal.

`eig(A,B)` computes the generalized eigenvalues of `A` and `B`.

`[X,LAMBDA]=eig(A,B)` computes the right eigenvectors `X` and eigenvalues of the generalized eigenvalues of `A` and `B` so that `A*X=B*X*LAMBDA`. `LAMBDA` is a diagonal matrix with the eigenvalues on the diagonal.

See also

`condeig`, `eigs`

Purpose

Compute a few eigenvalues and eigenvectors for a sparse matrix.

Synopsis

```
D = eigs(A)
[V D] = eigs(A)

D = eigs(A, k)
[V D] = eigs(A, k)

D = eigs(A, k, sigma)
[V D] = eigs(A, k, sigma)

D = eigs(A, B)
[V D] = eigs(A, B)

D = eigs(A, B, k)
[V D] = eigs(A, B, k)

D = eigs(A, B, k, sigma)
[V D] = eigs(A, B, k, sigma)

D = eigs(..., opts)
[V D] = eigs(..., opts)
```

Description

`D = eigs(A)` computes the six eigenvalues of largest magnitude of the square sparse matrix `A`.

`[V D] = eigs(A)` computes the six eigenvalues of largest magnitude and the corresponding eigenvalues for the sparse matrix `A`. `V` is a `size(A,1)`-by-6-matrix where the columns are the eigenvectors, and `D` is a 6-by-6-matrix with the eigenvalues on the diagonal. The matrices satisfy the relation `AV = VD`.

`D = eigs(A, k)` computes `k` eigenvalues.

`[V D] = eigs(A, k)` computes `k` eigenvectors and eigenvalues.

`D = eigs(A, k, sigma)` computes `k` eigenvalues in the vicinity of `sigma`, which can be a real or scalar constant or a string. String arguments decide what eigenvalues to search for. The following values are allowed:

SIGMA	INTERPRETATION
`'lm'`	Largest magnitude.
`'sm'`	Smallest magnitude.
`'lr'`	Largest real part.
`'sr'`	Smallest real part.

SIGMA	INTERPRETATION
'li'	Largest imaginary part.
'si'	Smallest imaginary part.

`D = eigs(A, B)` returns six eigenvalues for the generalized eigenvalue problem `AV = BVD`.

`[V D] = eigs(A, B)` returns six eigenvectors and corresponding eigenvalues for the generalized eigenvalue problem `AV = BVD`.

`D = eigs(A, B, k)` and `[V D] = eigs(A, B, k)` return `k` eigenvalues and eigenvectors for the generalized eigenvalue problem `AV = BVD`.

`D = eigs(A, B, k, sigma)` and `[V D] = eigs(A, B, k, sigma)` return `k` eigenvalues and eigenvectors close to `sigma` for the generalized eigenvalue problem `AV = BVD`. Possible values for sigma are listed above.

`D = eigs(..., opts)` and `[V D] = eigs(..., opts)` solve eigenvalue problems with options taken from the structure `opts`. The following fields of `opts` are used:

FIELD	INTERPRETATION
'tol'	Convergence tolerance.
'maxit'	Maximum number of Arnoldi iterations.
'p'	Dimension of Krylov subspace.

Algorithm

The function uses the ARPACK package. For generalized problems and problems where you specify a numerical value for `sigma`, `eigs` uses the shift-invert mode (ARPACK mode 3), otherwise it uses the standard mode (ARPACK mode 1). The shift-invert mode can be numerically more stable also for standard problems; setting `sigma=0` forces `eigs` to use it.

See also

`condeig`, `eig`

Purpose

Encrypt .M-files and .CSL-files.

Synopsis

```
encrypt(file1, ...)
encrypt(..., '-inplace')
```

Description

`encrypt(file1, ...)` creates encrypted versions of `file1`, ... The input file(s) must exist and be valid .M- or .CSL-files. For each input file, an .MC- or .CSLC-file is created in the current directory. When executed, it is equivalent to the original file, but its contents have been scrambled to make it unreadable.

`encrypt(..., '-inplace')` creates each encrypted file in the directory where the corresponding file was found.

Purpose

Retrieve the difference between a number and the next larger number.

Synopsis

```
e = eps
e = eps(v)
e = eps('double')
e = eps('single')
```

Description

`e = eps` returns the smallest `e` such that `1` and `1+e` are different floating-point numbers.

`e = eps(v)` returns the difference between `v` and the next-larger floating-point number.

`e = eps('double')` is equivalent to `e = eps`.

`e = eps('single')` returns what `eps` would be if 32-bit floating-point numbers were used instead of 64-bit floating-point numbers.

Purpose Compare matrices pointwise.

Synopsis

```
d = eq(a, b)
```

Description `d = eq(a, b)` tests if the elements of the two matrices `a` and `b` are equal pointwise. For each dimension, `a` and `b` must have the same size or either of them must have size 1. In the latter case, the unit dimension is expanded to the size of the nonunit dimension.

`eq(a, b)` is equivalent to `a == b`.

Examples

```
[2 3 5] == [0 3 6]
[10 20 30] == 30
[0 1] == [0 ; 1]
```

See also `ge`, `gt`, `le`, `lt`, `ne`

Purpose

Error function

Synopsis

```
y = erf(x)
```

Description

`y = erf(x)` computes the error function of the elements of `x`, where `x` must be a real array.

The error function is defined as:

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

See also

`erfc`, `erfcx`, `erfinv`

Purpose

Complementary error function

Synopsis

```
y = erfc(x)
```

Description

`y = erfc(x)` computes the complementary error function of the elements of `x`, where `x` must be a real array.

The complementary error function is defined as:

$$\mathrm{erfc}(x) = 1 + \mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_{x}^{\infty} e^{-t^2} dt$$

See also

`erf` `erfcx`, `erfinv`

Purpose

Scaled complementary error function.

Synopsis

```
y = erfcx(x)
```

Description

`y = erfcx(x)` computes the scaled complementary error function of the elements of `x`, where `x` must be a real array.

The scaled complementary error function is defined as:

$$\mathrm{erfcx}(x) = e^{x^2}\mathrm{erfc}(x)$$

See also

`erf`, `erfc`, `erfinv`

Purpose Inverse error function.

Synopsis

```
y = erfinv(x)
```

Description `y = erfinv(x)` computes the inverse error function of `x` such that `erf(erfinv(x)) = x`, where each element of `x` must satisfy `-1 <= x <= 1`. For any other values, corresponding elements of `y` are `NaN`.

Example `erf(erfinv([-0.5 0 0.3]))` returns `[-0.5 0 0.3]`.

See also `erf`, `erfc`, `erfcx`

Purpose

Throw an error exception.

Synopsis

```
error(msg)
error(s)
```

Description

`error(msg)`, where `msg` is a nonempty string, throws an error exception containing `msg`. Note that `error('')` does nothing.

`error(s)`, where `s` is a structure, is equivalent to `error(s.message)`.

See also

`warning`

Purpose

Throw an error exception.

Synopsis

```
errorbar(x,y,l,u)
errorbar(x,y,e)
h=errorbar(...)
```

Description

`errorbar(x,y,l,u)` plots `y` versus `x` and adds error bars according to `l` and `u`. `l` and `u` are the lower and upper error ranges for each point in `y`. If the inputs are matrices one line with error bars is drawn for each column.

`errorbar(x,y,e)` uses `e` as both `l` and `u`.

`errorbar('linespec')` can be used to control line color and line style. See `plot` for allowed values.

`h=errorbar(...)` returns handles to the drawn lines.

The property values for `line` can be passed at the end of the command to further control the plot.

See also

`hist`

Purpose

Determine elapsed time.

Synopsis

```
t = etime(t2,t1)
```

Description

`t = etime(t2,t1)` computes the time difference in seconds between `t2` and `t1`.

`t2` and `t1` must be vectors of the form returned by `clock`, that is, vectors with six elements representing, in order, year, month, day, hour, minute, and seconds. (See `clock` for further information.)

Example

```
A = rand(500);
t1 = clock;
B=svd(A);
t = etime(clock,t1)
```

See also

`clock`, `date`

Purpose Evaluate an expression or a sequence of statements.

Synopsis

```
a = eval(expr)
a = eval(expr1, expr2)
eval(stmts)
eval(stmts1, stmts2)
```

Description

`a = eval(expr)` evaluates the expression string `expr` and returns the result(s). It is possible for the evaluation to return more than one value.

`a = eval(expr1, expr2)` behaves like `a = eval(expr1)` except when this evaluation results in an error; in that case, `a = eval(expr2)` is performed.

`eval(stmts)` evaluates the statement string `stmts`.

`eval(stmts1, stmts2)` behaves like `eval(stmts1)` except when this evaluation results in an error; in that case, `eval(stmts2)` is performed.

See also `evalc`, `evalin`

Purpose

Evaluate an expression or a sequence of statements and retrieve any output made in the process.

Synopsis

```
[out, a] = evalc(expr)
[out, a] = evalc(expr1, expr2)
out = evalc(stmts)
out = evalc(stmts1, stmts2)
```

Description

The string `out` that is returned contains any text output to the prompt during the evaluation of the expression or statements. This is the only difference between `evalc` and `eval`.

See also

`eval`, `evalin`

Purpose Evaluate an expression or a sequence of statements in a specific workspace.

Synopsis

```
a = evalin(ws, expr)
a = evalin(ws, expr1, expr2)
evalin(ws, stmts)
evalin(ws, stmts1, stmts2)
```

Description The string `ws` specifies in which workspace the evaluation is performed: `'base'` is the root workspace, and `'caller'` is the parent workspace in the function-call stack. This is the only difference between `evalin` and `eval`.

See also `eval`, `evalc`

Purpose

Test whether or not a named object exists.

Synopsis

```
e = exist(name, 'var')
e = exist(name, 'file')
e = exist(name, 'builtin')
e = exist(name, 'dir')
e = exist(name, 'class')
e = exist(name)
```

Description

`exist(name, 'var')` returns 1 if there is a variable called `name`, otherwise 0.

`exist(name, 'file')` returns 2 if there is a file called `name`, 7 if there is a directory called `name`, otherwise 0.

`exist(name, 'builtin')` returns 5 if there is a built-in function called `name`, otherwise 0.

`exist(name, 'dir')` returns 7 if there is a directory called `name`, otherwise 0.

`exist(name, 'class')` returns 8 if there is a Java class called `name`, otherwise 0.

`exist(name)` tests `name` against all the above criteria and uses the same return-value conventions.

See also

`which`

Purpose	Close the command window.
Synopsis	`exit`
Description	`exit` closes the command window.
See also	`quit`

Purpose

Matrix exponential.

Synopsis

```
b = expm(a)
```

Description

`b = expm(a)` computes the matrix exponential of the square matrix `a`.

Example

```
a = [1 2;3 4];
em = expm(a);
e = exp(a);
```

returns `em` approximately `[51.97 74.74;112.1 164.1]` and `e` approximately `[2.718 7.389; 20.09 54.6]`.

See also

`exp`, `mpower`

Purpose Create a matrix with ones on the diagonal.

Synopsis

```
e = eye(n)
e = eye(sz)
e = eye(m, n)
```

Description In all cases it returns a matrix with ones on the diagonal and zeros elsewhere.The matrix size is determined as follows:

`eye(n)`, where `n` is a nonnegative integer, returns an `n` x `n`-matrix.

`eye(sz)`, where `sz` is a vector of length two, returns a matrix of size `sz`.

`eye(m, n)`, where `m` and `n` are nonnegative integers, returns an `m` x `n`-matrix.

See also `ones`, `repmat`, `zeros`

Purpose Prime factors.

Synopsis `f = factor(n)`

Description `f = factor(n)` computes the prime factors of `n` as a row vector `f`.

Example `factor(1275)` returns `[3, 5, 5, 17]`.

See also `isprime`, `primes`

Purpose	Factorial function.
Synopsis	`b= factorial(a)`
Description	`b = factorial(a)` computes the factorial of all elements of `a`, where `a` is an array of nonnegative integers.
Example	`factorial(5)` returns `120`, that is, `1*2*3*4*5`.

Purpose

Create all-false logical matrix.

Synopsis

```
f = false
f = false(n)
f = false(m, n, ...)
f = false(sz)
```

Description

In all cases, it returns all-false logical matrix whose size is determined as follows:

`f = false` returns a scalar.

`f = false(n)`, where `n` is a nonnegative integer, returns an `n` x `n matrix.`

`f = false(m, n, ...)`, where `m`, `n`, ... are nonnegative integers, returns an `m` x `n` x...-matrix.

`f = false(sz)`, where `sz` is a vector, returns a matrix of size `sz`.

See also

`true`

Purpose Close an open file or all open files.

Synopsis

```
fclose(h)
fclose('all')
```

Description `fclose(h)`, for an integer `h`, closes the file associated with the handle `h`, which must be one returned by `fopen`.

`fclose('all')` closes all open files.

See also `fopen`

Purpose Test whether end-of-file has been reached for an open file.

Synopsis

```
e = feof(h)
```

Description `feof(h)`, for an integer `h`, returns `true` if the end-of-file has been reached for the file associated with the handle `h`, otherwise it returns `false`. `h` must be a handle returned by `fopen`.

See also `fopen`

Purpose

Return or reset the error message for an open file.

Synopsis

```
e = ferror(h)
ferror(h, 'clear')
```

Description

`ferror(h)` returns the error message, if any, set by a previous failed file operation on `h`.

`ferror(h, 'clear')` clear the error message for `h`.

`h` must be a handle returned by `fopen`.

See also

`fopen`, `fread`

Purpose

Evaluate a function.

Synopsis

```
[a ...] = feval(func, arg1, ...)
```

Description

`feval(func, arg1, ...)` evaluates the function `func` for the arguments `arg1`, ... and returns the result(s). The number of inputs to and outputs from `func` can both be zero.

See also

`builtin`

Purpose

Compute the fast Fourier transform of a vector or matrix.

Synopsis

```
f = fft(v)
f = fft(v, n)
f = fft(v, n, dim)
f = fft(v, [], dim)
```

Description

`f = fft(v)` computes the FFT along the first nonunit dimension of `v`.

`f = fft(v, n)` computes the `n`-point FFT. `v` is padded with zeros if it is shorter than `n` and truncated if it is longer.

`f = fft(v, n, dim)` computes the `n`-point FFT along the dimension `dim`.

`f = fft(v, [], dim)` computes the FFT along the dimension `dim`.

See also

`ifft`, `fft2`, `ifft2`, `fftn`, `ifftn`

Purpose

Compute the 2D fast Fourier transform of a matrix.

Synopsis

```
f = fft2(m)
f = fft2(m, rows, cols)
```

Description

`f = fft2(m)` computes the 2D FFT of `m`.

`f = fft2(m, rows, cols)` computes the 2D FFT of size `(rows, cols)`. The input matrix is truncated or padded with zeros if necessary.

See also

`fft`, `ifft`, `ifft2`, `fftn`, `ifftn`

Purpose Compute the n-dimensional fast Fourier transform of an array.

Synopsis

```
f = fftn(m)
f = fftn(m, size)
```

Description `f = fftn(m)` computes the n-dimensional FFT of `m`.

`f = fftn(m, size)` computes the n-dimensional FFT of `m` of size `size`. The input array is truncated or padded with zeros if necessary.

See also `fft, ifft`, `fft2`, `ifft2`, `ifftn`

Purpose Shift a frequency spectrum computed with an FFT.

Synopsis

```
f = fftshift(m)
```

Description `f = fftshift(m)` shifts the indices in each dimension circularly so that index `1` in `m` corresponds to the middle index in `f`.

See also `circshift`, `fft, ifft`, `fft2`, `ifft2`, `fftn`, `ifftn`, `ifftshift`

Purpose Read a line from a file and discard the linefeed character(s).

Synopsis

```
s = fgetl(h)
```

Description `s = fgetl(h)` reads a line from the file pointed to by `h` and returns the line with the linefeed character(s) removed.

`h` must be a handle returned by `fopen`.

See also `fgets`, `fopen`

Purpose

Read a line from a file.

Synopsis

```
s = fgets(h)
s = fgets(h, n)
```

Description

`s = fgets(h)` reads and returns a line from the file pointed to by `h`.

`s = fgets(h, n)` reads from the file pointed to by `h` until it has read `n` characters or reached a linefeed character. Unlike `fgetl`, this functions returns linefeed characters.

`h` must be a handle returned by `fopen`.

See also

`fgetl`, `fopen`

Purpose

Get fields in structure or Java object.

Synopsis

```
f = fieldnames(s)
f = fieldnames(jo)
f = fieldnames(obj)
f = fieldnames(obj, attr)
f = fieldnames(obj, attr, noattr)
```

Description

`f = fieldnames(s)`, where `s` is a structure, returns a cell array containing the field names of `s`.

`f = fieldnames(jo)`, where `jo` is a Java object, returns a cell array containing the public fields in the class to which `jo` belongs.

`f = fieldnames(obj)`, where `obj` is an instance of a user-defined class, returns a cell array containing the public nonstatic fields of the class of `obj`.

`f = fieldnames(obj, attr)`, where `obj` is an instance of a user-defined class and `attr` is a string or cell array of strings, returns a cell array containing the fields that have at least one of the attributes listed in `attr`. Possible attributes are `'public'`, `'protected'`, `'private'`, `'static'`, and `'transient'`.

`f = fieldnames(obj, attr, noattr)` is like `f = fieldnames(obj, attr)` but excludes any field having an attribute listed in `noattr`, which must be a string or cell array of strings.

See also

`methods`

Purpose

Create a new figure window.

Synopsis

```
figure
figure(h)
```

Description

`figure` creates and opens a new figure window. You can retrieve the handle to the created figure with the syntax `h=figure`.

`figure(h)` makes the figure with handle h the current figure and shows it on top of all other windows.

See also

`clf`, `close`, `gcf`, `subplot`

Purpose

Split a file name into its path, name, and extension.

Synopsis

```
p = fileparts(name)
[p, n] = fileparts(name)
[p, n, e] = fileparts(name)
```

Description

`[p, n, e] = fileparts(name)`, where `name` is a string, returns the path of `name` in `p`, the name in `n`, and the extension in `e`. It is possible to omit the last or the last two output parameters.

Example

`[p, n, e] = fileparts('C:/COMSOL/license.txt')` results in `p = 'C:/COMSOL'`, `n = 'license'`, and `e = '.txt'`.

Purpose Get the system file separator.

Synopsis

```
sep = filesep
```

Description `sep = filesep` returns the directory separator in file names. For Windows this is `'\'`, and on all other platforms it is `'/'`.

See also `pathsep`

Purpose

1D digital filtering.

Synopsis

```
y = filter(b,a,x)
y = filter(b,a,x,zi)
y = filter(b,a,x,zi,dim)
y = filter(b,a,x,[],dim)
[y,zf] = filter(...)
```

Description

`y = filter(b,a,x)` uses a filter that is a Direct Form II Transposed implementation of the standard difference equation:

a(1)y(n) = b(1)x(n) + b(2)x(n-1) + ... + b(nb+1)x(n-nb)

- a(2)y(n-1) - ... - a(na+1)y(n-na)

where `x` is a data array, `y` is the filtered data, while `a` and `b` describe the filter. `n-1` is the filter order. Filter coefficients are normalized by *a(1)*.

When `x` is a matrix, `filter` works along the columns of `x`. When `x` is an array, `filter` works along the first nonsingleton dimension of `x`.

`y = filter(b,a,x,zi)` also includes `zi`, the initial conditions of the filter delays. `zi` must be either a vector of length `max(length(a),length(b))-1`, or an array of the same size as `x` except for the leading dimension, which must be `max(length(a),length(b))-1`.

`[y,zf] = filter(...)` also returns the final conditions of the filter delays.

`filter(b,a,x,zi,dim)` and `filter(b,a,x,[],dim)` work along the dimension `dim`.

See also

`dlsim`

Purpose

Find nonzero elements.

Synopsis

```
i = find(x)
[i,j] = find(x)
[i,j,y] = find(x)
```

Description

`i = find(x)` returns the linear indices of the nonzero elements of `x`.

`[i,j] = find(x)` returns row and column indices of nonzero elements of `x`, where `x` must be a matrix.

`[i,j,v] = find(x)` also returns the value of each nonzero element of `x`. If `x` is a row vector, `v` will be a row vector. Otherwise, `v` will be a column vector.

Examples

```
a = [1 4 0 12; 0 3 -10 0; 2 -1 3 4];
```

`find(a>0)` returns `[1; 3; 4; 5; 9; 10; 12]`.

`[j,i,b] = find(a);` returns
`i = [1; 3; 1; 2; 3; 2; 3; 1; 3]`,
`j = [1; 1; 2; 2; 2; 3; 3; 4; 4]` and
`b = [1; 2; 4; 3; -1; -10; 3; 12; 4];`

See also

`sparse`

Purpose Find graphics objects.

Synopsis

```
h = findobj(...)
h = findobj(parents,...)
```

Description `h = findobj(...)` finds graphics objects. The properties `'tag'` and `'type'` with a following value can be used to find graphics objects of a certain type or with a certain tag.

`h = findobj(parents,...)` searches only in the figure windows listed in `parents`.

Example `h = findobj('type','line')` finds all graphics objects of the type `'line'`.

See also `gca`, `gcf`, `get`, `set`

Purpose

Find a shorter string within a longer one.

Synopsis

```
ind = findstr(str1,str2)
```

Description

`ind = findstr(str1,str2)` finds occurrences of the shorter of the two strings `str1` and `str2` within the other and returns the first index of each such occurrence.

To find one string within another in a set order, use `strfind`.

Examples

`findstr('blue yellow green red','e')` and `findstr('e','blue yellow green red')` both return [4, 7, 15, 16, 20].

See also

`strfind`

Purpose Flip a dimension of a matrix.

Synopsis
```
f = flipdim(m, dim)
```

Description `f = flipdim(m, dim)` returns a matrix with the same contents as `m` but where the matrix indices in dimension `dim` have been flipped.

Example `flipdim([2 3 ; 5 7], 1)` returns `[5 7 ; 2 3]`.

See also `fliplr`, `flipud`, `permute`, `ipermute`

Purpose Flip a matrix horizontally.

Synopsis

```
f = fliplr(m)
```

Description `f = fliplr(m)` returns a matrix with the same contents as `m` but where each row has been flipped.

Example `fliplr([2 3 ; 5 7])` returns `[3 2 ; 7 5]`.

See also `flipdim`, `flipud`, `permute, ipermute`

Purpose Flip a matrix vertically.

Synopsis `f = flipud(m)`

Description `f = flipud(m)` returns a matrix with the same contents as `m` but where each column has been flipped.

Example `flipud([2 3 ; 5 7])` returns `[5 7 ; 2 3]`.

See also `flipdim`, `fliplr`, `permute, ipermute`

Purpose

Solve an unconstrained nonlinear optimization problem using the Nelder-Mead simplex algorithm.

Syntax

```
x = fminsearch(f,x0,...)
[x,f] = fminsearch(f,x0,...)
[x,f,exitflag] = fminsearch(f,x0,...)
[x,f,exitflag,infostruct] = fminsearch(f,x0,...)
fminsearch(f,x0,options)
```

Description

`x = fminsearch(f,x0)` solves the unconstrained nonlinear optimization problem min $f(x)$, where `f` is a function and `x0` the initial guess. (Functions can be either strings, denoting the function name, or inline functions.)

Aside from `x`, `fminsearch` can return `f`, the value of the objective function at `x`, `exitflag`, indicating the exit condition (`0`, if `fminsearch` reached the maximum number of iterations or function evaluations, `1` for successful completion) and a struct `infostruct` containing information about number of iterations and function evaluations.

`fminsearch(f,x0,options)` also includes an `options` structure, which can have the following fields (`N` is the number of variables):

TABLE 1-17: VALID PROPERTIES FOR THE FMINSEARCH FUNCTION

PROPERTY	VALUE	DEFAULT	DESCRIPTION
`Display`	`'off'| 'iter'`	`'off'`	`'iter'` displays the result at each iteration and whether `fminsearch` performs a reflection, expansion, inner or outer contraction, or a shrinking step.
`MaxFunEvals`	`integer`	`200*N`	Limit on number of function evaluations.
`MaxIter`	`integer`	`200*N`	Iterations limit.
`LengthScale`	`numeric`	1	Length scale used when creating the initial simplex, which is defined by the starting guess `x0` and `n` more points `x0 + LengthScale*eye(n)`.

TABLE 1-17: VALID PROPERTIES FOR THE FMINSEARCH FUNCTION

PROPERTY	VALUE	DEFAULT	DESCRIPTION
`Param`	any	`empty`	Allows additional arguments to be passed along to the callback function. Use a cell array to pass along more than one argument. Note that the cell array will be unpacked in the function call, hence setting `Param` to `{a1,a2}` will result in function being called with `userfun(x,a1,a2)`.
`TolFun`	`numeric`	`1e-4`	Absolute termination tolerance on the function precision
`TolX`	`numeric`	`1e-4`	Absolute termination tolerance on the largest diameter of the simplex, using infinity norm

Examples

```
function f = exnm_obj(x)
f = x(1)^4 + x(2)^4 - x(1)*x(2) + 1;

[x,f] = fminsearch('exnm_obj',[1 1]);

% With inline function
[x,f] = fminsearch(inline('x(1)^4 + x(2)^4 - x(1)*x(2) + 1'),
                   [1 1]);
```

Algorithm

`fminsearch` uses the Nelder-Mead simplex algorithm, as defined in "Convergence Properties of the Nelder-Mead Simplex Method in Low Dimensions" (Jeffrey C. Lagarias, James A. Reeds, Margaret H. Wright, Paul E. Wright, SIAM Journal of Optimization, 9(1) p.112–147, 1998).

Purpose

Open a file or get information about an open file.

Synopsis

```
h = fopen(filename)
h = fopen(filename, mode)
h = fopen(filename, mode, endian)

[name, mode, endian] = fopen(h)
```

Description

`h = fopen(filename)` opens the file `filename` for reading and returns a handle to the open file.

`h = fopen(filename, mode)` opens the file `filename` in the mode `mode`. The following modes can be used:

TABLE 1-18: FOPEN MODES

MODE	INTERPRETATION
`'r'`	Open for reading.
`'w'`	Open for writing.
`'a'`	Open for writing, position the file pointer at the end of the file.
`'r+'`	Open for reading and writing.
`'w+'`	Open for reading and writing, remove the current contents of the file.
`'a+'`	Open for reading and writing, position the file pointer at the end of the file.

If running Windows, you can append a `'t'` to the mode string. This results in the file being opened in text mode.

`h = fopen(filename, mode, endian)` opens the file `filename` in the mode `mode` and the endianness `endian`. The following endiannesses can be used:

TABLE 1-19: ENDIANNESSES

MODE	INTERPRETATION
`'n'` or `'native'`	Open in the native system endianness.
`'b'`, `'s'`, `'ieee-be'`, or `'ieee-be.l64'`	Open as big-endian.
`'l'`, `'a'`, `ieee-le'`, or `'ieee-le.l64'`	Open as little-endian.

The file is open in the native endianness if no endianness is specified.

`[name, mode, endian] = fopen(h)` returns the filename, mode, and endianness used when the file handle `h` was created using `fopen`. The file must be open.

See also `fclose`, `fread`, `fwrite`

Purpose

Set the output format.

Synopsis

```
format('compact')
format('loose')

format('short')
format('long')
format('hex')
format('+')
```

Description

`format('compact')` results in output being generated with no extra vertical space.

`format('loose')` results in output being generated with empty lines inserted to improve readability. This is the default.

`format('short')` results in floating-point numbers being displayed with approximately 8 significant digits. This is the default.

`format('long')` result in floating-point numbers being displayed with approximately 16 significant digits.

`format('hex')` results in floating-point numbers being displayed as the hexadecimal form of their IEEE-754 representation.

`format('+')` results in floating-point numbers being displayed as `'+'` if they are positive; `'-'` if they are negative; and `' '` if they are zero.

Purpose Get the formula computed by an inline function.

Synopsis

```
form = formula(func)
```

Description `form = formula(func)`, where `func` is an inline function, returns the string that defines the function computed by `func`. This is the first argument that was given to `inline` when `func` was created.

See also `inline`

Purpose

Write formatted output to a file.

Synopsis

```
n = fprintf(h, format, ...)
n = fprintf(format, ...)
```

Description

For the syntax and interpretation of the format string, see the manual entry for `sprintf`.

`n = fprintf(h, format, ...)` writes formatted output to the file pointed to by the handle `h`. The return value is the number of bytes written.

`n = fprintf(format, ...)` writes formatted output to the terminal.

See also

`fopen`, `sprintf`

Purpose

Create a window for use when creating a custom user interface.

Synopsis

```
f = frame(title,...)
```

Description

`f = frame(title)` creates a frame with the specified title.

In addition to the fixed arguments, additional property-value pairs can be given at the end of the function to further control how the frame is created:

TABLE 1-20: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DESCRIPTION
`position`	2-element vector	The position on the screen for the upper left corner of the frame.
`size`	2-element vector	The size of the frame. If not given the frame will be packed to fit the size of the components that have been added to it.

The function returns a `frame` object that can then be further manipulated using the methods in the following table.

TABLE 1-21: METHODS FOR MANIPULATING A FRAME OBJECT.

METHOD	DESCRIPTION
`addMenu(menu)`	Adds the specified menu at the end of the main menu bar of the frame.
`close`	Closes the frame.
`getSize`	Returns the size of the frame as a 2-element vector with width and height.
`setSize(width,height)`	Sets the size of the frame.
`show`	While the frame is being created it is invisible. Call the show method after adding all components to it to show it on screen.

The methods for `panel` are also available for `frame`, thereby allowing you to add panels and components to a frame.

See also the reference entry for `component` for property-value pairs and methods that are valid for all components.

See also

`component`, `dialog`, `panel`

Purpose

Read binary data from a file.

Synopsis

```
d = fread(h)
d = fread(h, sz)
d = fread(h, type)
d = fread(h, sz, type)
```

Description

`d = fread(h)` reads one character at a time from the file with the handle `h` until it reaches the end of the file. The characters read are returned in a real column vector.

`d = fread(h, sz)`, where `sz` is a numerical vector, reads one character at a time from the file with the handle `h` until it has filled a matrix of size `sz`. The last element of `sz` can be `Inf`; in that case, that dimension grows until the end of the file is reached.

`d = fread(h, type)`, where `type` is a string, reads data of type `type` until it reaches the end of the file. The following type syntaxes are supported:

TABLE 1-22:

TYPE SYNTAX	INTERPRETATION

`'char'`
`'schar'`
`'signed char'`
`'char*1'`
	Read signed 8-bit characters into double matrix.

`'uchar'` `'unsigned char'`	Read unsigned 8-bit characters into double matrix.
`'int8'` `'integer*1'`	Read signed 8-bit integers into double matrix
`'uint8'`	Read unsigned 8-bit integers into double matrix.

`'int16'`
`'short'`
`'integer*2'`
	Read signed 16-bit integers into double matrix.

`'uint16'`
`‘ushort'`
`'unsigned short'`
	Read unsigned 16-bit integers into double matrix.

`'int32'`
`'int'`
`'integer*4'`
	Read signed 32-bit integers into double matrix.

`'uint32'`
`'uint'`
`'unsigned int'`
	Read unsigned 32-bit integers into double matrix.

TABLE 1-22:

TYPE SYNTAX	INTERPRETATION

`'int64'`
`'long'`
`'integer*8'`
	Read signed 64-bit integers into double matrix.

`'uint64'`
`'unsigned long'`
`'ulong'`
	Read unsigned 64-bit integers into double matrix.

`'float'`
`'float32'`
`'real*4'`
`'single'`
	Read 32-bit IEEE floating-point numbers into double matrix.

`'double'`
`'float64'`
`'real*8'`
	Read 64-bit IEEE floating-point numbers into double matrix.

`'*`*type*`'`	Read data of type *type* into a matrix of the closest type. Equivalent to only using `'`*type*`'`.
`'`*type1*`=>`*type2*`'`	Read data of type *type1* into a matrix of the type closest to *type2*.
`'`*block*`*`*type*`'`	Read *block* values of type *type* into a double matrix. Must be used together with a skip parameter, see below.
`'`*block*`*`*type1*`=>`*type2*`'`	Read *block* values of type *type1* into a matrix of the type closest to *type2*. Must be used together with a skip parameter, see below.

`d = fread(h, sz, type)` reads data of type `type` until it has filled a matrix of size `sz`.

`d = fread(h, sz, type, skip)` reads data of type `type` until it has filled a matrix of size `sz`. After reading one or more values (depending on the `type` string), it reads `skip` values of the same type but ignores them. This only has meaning if the `type` string contains a `'=>'` (see the table), and it can be used for reading entries from records of a fixed size.

Examples

`d = fread(h, [4 inf])` returns a 4 x *n*-matrix of doubles where *n* is the largest integer such that it is possible to read 4*n* characters without reaching the end of the file.

`d = fread(h, 'int32=>double')` reads signed 32-bit integers until it reaches the end of the file is reached and returns them in a column vector of doubles.

`d = fread(h, 20, '5*double', 3)` reads five doubles from file, skips the next three, and so on until it has read 20 doubles.

See also `fopen`, `fwrite`

Purpose

Create a frequency range.

Synopsis

```
freq = freqspace(sz)
freq = freqspace(sz, 'whole')

[freq1, freq2] = freqspace(sz)
[freq1, freq2] = freqspace(sz, 'meshgrid')
```

Description

`freq = freqspace(sz)`, for a scalar `sz`, returns a vector with `(sz+1)/2` uniformly spaced values between 0 and 1.

`freq = freqspace(sz, 'whole')`, for a scalar `sz`, returns a vector with `sz` uniformly spaced values between 0 and `2(1-1/sz)`.

`[freq1, freq2] = freqspace(sz)`, for a vector `sz`, of length 2 returns `length(sz(2))` uniformly spaced values between `-1+1/sz(2))` and `1-1/sz(2)` in `freq1`, and it returns `length(sz(1))` uniformly spaced values between `-1+1/sz(1))` and `1-1/sz(1)` in `freq2`.

`[freq1, freq2] = freqspace(sz, 'meshgrid')`, for a vector of length 2, computes `[a, b] = freqspace(sz)` and returns `meshgrid(a, b)`.

See also

`meshgrid`

Purpose Rewind a file.

Synopsis `frewind(h)`

Description `frewind(h)`, for an integer `h`, rewinds the file associated with the handle.

`h` must be a handle returned by `fopen`.

See also `fopen`

Purpose Read formatted data from file.

Synopsis

```
d = fscanf(h, format)
d = fscanf(h, format, sz)
```

Description `d = fscanf(h, format, ...)` reads formatted data from the file handle `h`. For the interpretation of the `format` and `sz` argument, see `sscanf`. For valid format strings, see `sprintf`.

`h` must be a handle returned by `fopen`.

See also `sscanf`, `fprintf`, `sprintf`

Purpose

Move a file pointer.

Synopsis

```
fseek(h, offset, dir)
```

Description

`fseek(h, offset, dir)`, for integers `h` and `offset`, moves the file pointer by a distance `offset` bytes in a way defined by `dir`, which can have the following values:

DIR	INTERPRETATION
`'bof'` or `-1`	Move `offset` bytes from the beginning of the file; `offset` must be nonnegative.
`'cof'` or `0`	Move `offset` bytes from the current position in the file.
`'eof'` or `+1`	Move `offset` bytes from the end of the file; `offset` must be nonpositive.

`h` must be a handle returned by `fopen`.

See also

`fopen`

Purpose Get the position of the file pointer.

Synopsis

```
pos = ftell(h)
```

Description `pos = ftell(h)`, for an integer `h`, returns the offset (in bytes) of the file pointer relative to the beginning of the file. If the handle is invalid, it returns `-1`.

`h` must be a handle returned by `fopen`.

See also `fopen`, `fseek`

Purpose Convert a matrix from sparse to full.

Synopsis

```
f = full(sp)
```

Description `f = full(sp)`, where `sp` is a sparse matrix, returns a full matrix with the same contents. If `sp` is a full matrix, `full` returns `sp`.

See also `sparse`

Purpose Create a file name.

Synopsis

```
name = fullfile(dir1, ..., file)
```

Description `name = fullfile(dir1, ..., file)` creates a file name from one or more directory names `dir1`, ...and a file name `file`.

See also `fileparts`

Purpose

Evaluate matrix function.

Synopsis

```
F = funm(A,fun)
F = funm(A,fun,options)
[F,taylorflag] = funm(...)
[F,taylorflag,stat] = funm(...)
funm(A,fun,[],x1,x2,...)
funm(A,fun,options,x1,x2,...)
```

Description

`F = funm(A,fun)` computes the matrix function `fun` of a square matrix `A`. `fun` must have a Taylor series with an infinite radius of convergence and `fun(x,k)` should return the `k`'th derivative of `fun` evaluated at `x`. `fun = 'log'` is a special case and returns the matrix logarithm as described in `logm`.

`F = funm(A,fun,options`) computes the matrix function with one or more parameters given in the structure options:

FIELDNAME	VALUE/{DEFAULT}	DESCRIPTION
TolBlk	positive scalar {0.1}	Tolerance for blocking Schur form.
TolTay	positive scalar {eps}	Termination tolerance for Taylor series.
MaxTerms	positive integer {250}	Maximum number of Taylor series terms.
MaxSqrt	positive integer {100}	Maximum number of square roots in inverse scaling and squaring method. Only applicable when computing logarithm.)
Ord	integer vector {[]}	Specific ordering of the Schur form, T. (See `ordschur` for more information.)

`[F,taylorflag] = funm(...)` returns `taylorflag`, which is `1` if one or more Taylor series evaluations did not converge and `0` otherwise.

`[F,taylorflag,stat] = funm(...)` also returns a structure stat with the following fields:

FIELDNAME	DESCRIPTION
terms	Vector containing the number of Taylor series terms used when evaluating each block. In the case of the logarithm, it contains instead the number of square roots evaluations.
ind	Cell array that specifies the blocking, that is, the block (i,j) of the reordered Schur matrix T is T(stat.ind{i},stat.ind{j}).

FIELDNAME	DESCRIPTION
ord	The ordering passed to `ordschur`
T	The reordered Schur matrix.

When the Schur form is diagonal,

```
stat = struct('terms',ones(n,1),'ind',{1:n})
```

`funm(A,fun,[],x1,x2,...)` and `funm(A,fun,options,x1,x2,...)` allows additional input arguments `x1`, `x2`, … to be passed to `fun`.

Example

```
function c = coshm(a,k)
if mod(k,2);
  c = sinh(a);
else
  c = cosh(a);
end
```

```
F = funm(X,'coshm');
```

See also `expm`, `logm`, `sqrtm`

Purpose

Write data to a binary file.

Synopsis

```
n = fwrite(h, mat)
n = fwrite(h, mat, type)
n = fwrite(h, mat, type, skip)
```

Description

`n = fwrite(h, mat)` writes the matrix `mat` to the file handle `h`.

`n = fwrite(h, mat, type)` writes each element as type `type`. For a listing of the available types, see `fread`.

`n = fwrite(h, mat, type, skip)` moves the file pointer forward a distance of `skip` bytes before writing each element.

The number of elements successfully written is returned.

Examples

`fwrite(h, pi)` writes the double `pi` to the file.

`fwrite(h, 1:100, 'int16')` writes the 100 16-bit integers 1, 2, … to file.

`fwrite(h, 1:100, '5*int32', 3)` skips three bytes, then writes the five 32-bit integers 1, 2, …, 5, then skips three more bytes, and so on, until it has written the integers `1:100`. The file pointer moves 460 bytes forwards in the process; 400 bytes are written and 60 bytes are skipped (3 bytes each for 20 blocks).

See also

`fopen`, `fread`

Purpose

Find a zero of a function.

Synopsis

```
x = fzero(f, x0, ...)
```

Description

`x = fzero(f, x0, ...)` finds the `x` argument where the function `f` is equal to zero. `f` can be an M-file, inline function, or expression. If it is an expression, the argument has to be `'x'`. All other variables in the expression have to be passed to `fzero` in the same order as they appear in the expression. `fzero` uses the secant method to find a zero.

Valid property-value pairs:

TABLE 1-23:

PROPERTY	DESCRIPTION
`'maxiter'`	Maximum number of iterations before giving up.
`'tol'`	Absolute tolerance for x.
`'x1'`	Second point for the secant method.

All other arguments are passed to the function call.

Examples

Find a zero of cosine near 1:

```
x0 = fzero('cos(x)',[1]);
```

`fzero` using an inline function:

```
myfun=inline('exp(y).*cos(x)');
x=-5:0.05:5;
plot(x,myfun(x,2));
```

Find a zero of `exp(2).*cos(x)` using start guess 5:

```
x1 = fzero(myfun,-5,2)
```

Find a zero of `exp(2).*cos(x)` using start guess -2

```
x2 = fzero(myfun,-2,2);
```

See also

`inline`

Purpose

Gamma function.

Synopsis

```
g = gamma(z)
```

Description

`g = gamma(z)` computes the gamma function, as defined in the following equation, of the elements of `z`:

$$\Gamma(z) = \int_0^\infty e^{-t} t^{z-1} dt$$

`z` must be a real array.

See also

`gammainc`, `gammaln`, `beta`

Purpose

Incomplete gamma function.

Synopsis

```
g = gammainc(x,a)
g = gammainc(x,a,tail)
```

Description

`g = gammainc(x,a)` computes the incomplete gamma function (sometimes called the regularized incomplete gamma function), as defined in the following equation:

$$P(x,a) = \frac{1}{\Gamma(a)} \cdot \int_0^x t^{a-1} e^{-t} dt$$

where $\Gamma(x)$ is the gamma function.

`a` and `x` must be nonnegative real arrays of the same size or either one can be a scalar.

`g = gammainc(x,a,tail)` computes the incomplete gamma function using either the upper or lower tail, denoted by the strings `'upper'` and `'lower'`, respectively. The default is `'lower'`.

`gammainc(x,a,'upper') = 1 - gammainc(x,a,'lower')`.

See also

`gamma`, `gammaln`, `betainc`

Purpose Logarithm of the gamma function

Synopsis

```
g = gammaln(z)
```

Description `g = gammaln(z)` computes the natural logarithm of the gamma function of `z` without computing the actual gamma function. `z` must be a real array.

Example `gammaln(200)` computes the logarithm of the gamma function where `log(gamma(200))` would overflow.

See also `gamma`, `gammainc`, `beta`, `betaln`

Purpose Get the handle to the current axes in the current figure.

Synopsis

```
h = gca
```

Description `h = gca` returns the handle to the current axes in the current figure. This is the axes object into which the plotting commands plot if the `'parent'` property is not used to explicitly specify one.

You can change the current axes with the `subplot` command or by clicking in the axes. Giving a figure focus makes the current axes in that figure the current axes.

See also `cla`, `gcf`, `subplot`

Purpose

Greatest common divisor

Synopsis

```
[g,x,y] = gcd(a,b)
```

Description

`g = gcd(a,b)` computes the greatest common divisors of the corresponding elements of arrays `a` and `b`, which must be the same size or either one can be a scalar.

`[g,x,y] = gcd(a,b)` also returns integers `x` and `y` such that `ax + by = g`.

Examples

`gcd([1209 678 211 136],342) returns [3, 6, 1, 2]`.

`[g,x,y] = gcd([120 78 111 136], [142,20,12,98])` returns `g = [2, 2, 3, 2]`, `x = [-13, -1, 1, -18]` and `y = [11, 4, -9, 25]`.

See also

`lcm`

Purpose Get the handle to the current figure.

Synopsis

```
h = gcf
```

Description `h = gcf` returns the handle to the current figure, which you can change by giving a certain figure window the focus.

The plotting commands plot into the current axes of the current figure if the `'parent'` property is not used to explicitly specify an axes object.

See also `clf`, `figure`, `gca`

Purpose

Compare matrices pointwise.

Synopsis

```
d = ge(a, b)
```

Description

`d = ge(a, b)` tests if the elements of the matrix `a` are pointwise greater than or equal to those of the matrix `b`. For each dimension, `a` and `b` must have the same size or either of them must have size 1. In the latter case, the unit dimension is expanded to the size of the nonunit dimension.

`ge(a, b)` is equivalent to `a >= b`.

Examples

```
[2 3 5] >= [1 3 7]
[5 -10 20] >= 0
[1 2 3] >= [1 ; 2]
```

See also

`eq`, `gt`, `le`, `lt`, `ne`

genpath

Purpose Return path string for a directory tree.

Synopsis

```
p = genpath(dir)
```

Description `p = genpath(dir)`, where `dir` is a string containing a directory name, returns a path string containing `dir`, its subdirectories, the subdirectories of the subdirectories, and so on.

See also `path`

Purpose

Get data from a graphics object.

Synopsis

```
get(h,name)
get(h)
```

Description

`get(h,name)` returns the value of the property `name` for the object to which the graphics handle h refers.

`get(h)` returns the values of all properties in a structure where the property names are the field names of the structure.

See also

`set`

Purpose Return application data from a frame or a dialog box.

Synopsis

```
data = getdata(f)
```

Description `data = getdata(f)` returns data that has been stored in `f` using the `storedata` function. `f` can be a frame or a dialog box.

See also `storedata`

Purpose

Get the value of a structure field.

Synopsis

```
f = getfield(s, field)
f = getfield(s, index1, field, index2)
```

Description

`f = getfield(s, field)`, for a structure `s`, returns the value of `s.(field)`.

`f = getfield(s, index1, field, index2)` returns `s(index1{:}).(field)(index2{:})` where `index1` and `index2` are cell arrays containing array indices.

See also

`setfield`

Purpose

Compute approximate gradient.

Synopsis

```
df = gradient(f)
df = gradient(f,h)
[fx,fy] = gradient(f)
[fx,fy] = gradient(f,h)
[fx,fy] = gradient(f,hx,hy)
[fx,fy,fz,...] = gradient(f)
[fx,fy,fz,...] = gradient(f,h)
[fx,fy,fz,...] = gradient(f,hx,hy,hz,...)
```

Description

`df = gradient(f)` computes the 1D gradient of a vector `f` using unit spacing.

`df = gradient(f,h)` computes the gradient using spacing `h` between points. `h` must be a scalar.

`[fx,fy] = gradient(f)` computes the gradient of a matrix `f` using unit spacing. `fx` corresponds to $\frac{df}{dx}$, the differences in the column direction, and `fy` corresponds to $\frac{df}{dy}$, the differences in the row direction.

`[fx,fy] = gradient(f,h)` computes the gradient using spacing `h` between points. `h` must be a scalar.

`[fx,fy] = gradient(f,hx,hy)` computes the gradient using pacing specified by `hx` and `hy`. `f` must be 2D and `hx` and `hy` must be either scalars (in which case they specify spacing between points in the x and y directions, respectively) or vectors, in which case they specify the coordinates of the points in their respective directions. If either `hx` or `hy` is a vector, its length must match the corresponding dimension of `f`.

`[fx,fy,fz,...] = gradient(f)` computes the gradient of the 3D array `f`. `fz` corresponds to $\frac{df}{dz}$, the differences in the z direction.

`[fx,fy,fz] = gradient(f,h)` computes the gradient using the spacing `h` between points. `h` must be a scalar.

`[fx,fy,fz]=gradient(f,hx,hy,hz)` uses the spacing given by `hx`, `hy`, `hz`.

Similarly, when `f` is an n-dimensional array, `gradient` must have n outputs, and the input must be in the form `gradient(f,h)` or `gradient(f,h1,h2...hn)`.

Examples

`gradient([1 3 5 10])` computes the 1D gradient with unit spacing and returns `[2, 2, 3.5, 5]`.

```
a = [1 4 0 12;0 3 -10 0;2 -1 3 4];
```

`[fx,fy] = gradient(a)` computes `fx` and `fy` with unit spacing.
`fx = [3, -0.5, 4, 12 ; 3, -5, -1.5, 10 ; -3, 0.5, 2.5, 1]` and
`fy = [-1, -1, -10, -12 ; 0.5, -2.5, 1.5, -4 ; 2, -4, 13, 4]`.

`[fx,fy] = gradient(a,0.2,0.4)` computes `fx` and `fy` with spacing `0.2` in the *x* direction and `0.4` in the *y* direction.
`fx = [15 -2.5 20 60;15 -25 -7.5 50;-15 2.5 12.5 5]` and
`fy = [-2.5 -2.5 -25 -30;1.25 -6.25 3.75 -10;5 -10 32.5 10]`.

See also `del2`, `diff`

Purpose Create a colormap with gray scales.

Synopsis `gray(n)`

Description `gray(n)` returns a colormap with `n` colors. It is a matrix with `n` rows and 3 columns with RGB values for the colors in the colormap. The colors are gray scales.

See also `colormap`, `bone`, `cool`, `grayprint`, `jet`, `hot`, `hsv`, `pink`, `wavemap`

Purpose

Create a colormap with printer-friendly gray scales.

Synopsis

```
grayprint(n)
```

Description

`grayprint(n)` returns a colormap with `n` colors. It is a matrix with `n` rows and 3 columns with RGB values for the colors in the colormap. The colors are printer-friendly gray scales.

See also

`colormap`, `bone`, `cool`, `gray`, `jet`, `hot`, `hsv`, `pink`, `wavemap`

Purpose

Display grid lines.

Synopsis

```
grid('on')
grid('off')
grid
grid(ax, ...)
```

Description

`grid('on')` turns the display of grid lines on.

`grid('off')` turns the display of grid lines off.

`grid` toggles the display of grid lines on or off.

`grid(ax,...)` controls the display of grid lines in the axes `ax` instead of in the current axes.

See also

`box`

Purpose

2D data gridding.

Synopsis

```
zi = griddata(x,y,z,xi,yi)
[xi,yi,zi] = griddata(x,y,z,xi,yi)
s = griddata(x,y,xi,yi)
griddata(...,method)
griddata(...,method,bnd)
griddata(...,method,bnd,strategy)
```

Description

`zi = griddata(x,y,z,xi,yi)` performs a delaunay triangulation on `x` and `y`, where `z = f(x,y)`, and interpolates `xi` and `yi` linearly to determine `zi = f(xi,yi)`. The points do not need to be uniformly spaced.

`x` and `y` must either be of the same size or vectors of different orientation, in which case `griddata` uses `[x,y] = meshgrid(x,y)`. `z` must either be the same size as `x` and y or, when they are vectors of different orientation, a matrix with `length(x)` rows and `length(y)` columns. Similarly, when `xi` and `yi` are vectors of different orientation, `griddata` uses `[xi,yi] = meshgrid(xi,yi)`. Otherwise, `xi` and `yi` must have the same size.

`[xi,yi,zi] = griddata(x,y,z,xi,yi)` returns `xi` and `yi` used by griddata.

`s = griddata(x,y,xi,yi)` returns a struct `s` that contains the triangulation of `x` and `y` and information about which delaunay element the points in `xi` and `yi` belong to, including local coordinates. This can be used together with `tinterp` to interpolate different data values using the same points and triangulation. (See `tinterp` for more details.)

`griddata(...,method)` specifies the interpolation method, which can be either `'linear'` (denoting linear interpolation) or `'nearest'` (denoting nearest neighbor interpolation). Nearest neighbor in this case signifies the closest vertex in the nearest delaunay triangle. Default method is linear.

`griddata(...,method,bnd)` also includes boundary information `bnd`, which `griddata` sends on to the internal call to delaunay (see `delaunay` for further information).

`griddata(...,method,bnd,strategy)` allows the search strategy to be set explicitly. `strategy` can be either `'boxonly'` (default for linear interpolation), in which case `griddata` returns `NaN` for all points outside the mesh, or `'closest'` (default for nearest neighbor interpolation), in which case `griddata` locates the nearest element for all points.

Examples

```
rand('state',0);
```

```
x = 4*rand(1,100)-2;y = 4*rand(1,100)-2;
z=sin(x).*sin(y).*exp(-x.^2-y.^2);
ti = -2:.1:2;
[xi,yi] = meshgrid(ti,ti);
zi = griddata(x,y,z,xi,yi,'linear',[],'closest');

plot3(x,y,z,'*');
hold on;
mesh(xi,yi,zi);
hold off;

g = griddata(x,y,xi,yi, 'linear',[],'closest');
zi1 = tinterp(g,z);
z2 = sin(x).*sin(y);
zi2 = tinterp(g,z2);

plot3(x,y,z,'*');
hold on;
mesh(xi,yi,zi1);
hold off;
figure;
plot3(x,y,z2,'*');
hold on;
mesh(xi,yi,zi2);
hold off;
```

g is the struct:

```
g =
    method:  'linear'
  strategy:  'closest'
         t:  [182x3 double]
       ind:  [1681x1 double]
     coord:  [1681x3 double]
      size:  [41 41]
```

See also griddata3, griddatan, tinterp, tsearch, tsearchn, delaunay, delaunay3

Purpose

3D data gridding.

Synopsis

```
vi = griddata3(x,y,z,v,xi,yi,zi)
s = griddata3(x,y,xi,zi)
griddatan3(...,method)
griddatan3(...,method,bnd)
griddatan3(...,method,bnd,stratgy)
```

Description

`vi = griddata3(x,y,z,v,xi,yi,zi)` performs a 3D delaunay triangulation on the points defined by `x`, `y` and `z`, where `v = f(x,y,z)`, and interpolates the points defined by `xi`, `yi` and `zi` linearly to determine `vi = f(xi,yi,zi)`. The points do not need to be uniformly spaced.

`s = griddata3(x,y,z,xi,yi,zi)` returns a struct `s` containing the triangulation of `x`, `y` and `z` and information about which delaunay element each point `xi`, `yi` and `zi` belongs to, including local coordinates. This can be used together with `tinterp` to interpolate different data values using the same points and triangulation. (See `tinterp` for more details.)

`griddata3(...,method)` specifies the interpolation method, which can be either `'linear'` (denoting linear interpolation) or `'nearest'` (denoting nearest neighbor interpolation). Nearest neighbor in this case signifies the closest vertex in the nearest delaunay tetrahedron. Default method is linear.

`griddata3(...,method,bnd)` also includes boundary information `bnd`, which `griddata3` sends on to the internal call to `delaunay3` (see `delaunay3` for further information).

`griddata3(...,method,bnd,strategy)` allows the search strategy to be set explicitly. `strategy` can be either `'boxonly'` (default for linear interpolation), in which case `griddata3` returns `NaN` for all points outside the mesh, or `'closest'` (default for nearest neighbor interpolation), in which case `griddata3` locates the nearest element for all points.

Examples

```
x = rand(1,13);y = rand(1,13);z = rand(1,13);
v = sin(x).*sin(x).*sin(z);
[xi,yi,zi] = meshgrid(0:.24:1);
vi = griddata3(x,y,z,v,xi,yi,zi,'linear',[],'closest');
```

```
g = griddata3(x,y,z,xi,yi,zi,'nearest');
vi1 = tinterp(g,rand(1,13));
vi2 = tinterp(g,rand(1,13));
```

`g` is the struct:

```
g =
```

```
        method:  'nearest'
      strategy:  'closest'
             t:  [28x4 double]
           ind:  [125x1 double]
         coord:  [125x4 double]
          size:  [5 5 5]
```

See also griddata, griddatan, tinterp, tsearch, tsearchn, delaunay, delaunay3

Purpose

nD data gridding.

Synopsis

```
yi = griddatan(pts,y,ptsi)
s = griddatan(pts,ptsi)
griddatan(...,method)
griddatan(...,method,bnd)
griddatan(...,method,bnd,strategy)
```

Description

`yi = griddatan(pts,y,ptsi)` performs a delaunay triangulation on the points in `pts`, where `y = f(x1,x2,...)`, `xj = pts(:,j)`, and interpolates the points in `ptsi` linearly to determine `yi = f(xi1,xi2,...)`, where `xij = ptsi(:,j)`. The points do not need to be uniformly spaced. `pts` and `ptsi` must be of size `nx2` (for 2D) or `nx3` (for 3D).

`s = griddatan(pts,ptsi)` returns a struct `s` containing the triangulation of `pts` and information about which delaunay element each point belongs to, including local coordinates. This can be used together with `tinterp` to interpolate different data values using the same points and triangulation. (See `tinterp` for more details.)

`griddatan(...,method)` specifies the interpolation method, which can be either `'linear'` (denoting linear interpolation) or `'nearest'` (denoting nearest neighbor interpolation). Nearest neighbor in this case signifies the closest vertex in the nearest delaunay element. Default method is linear.

`griddatan(...,method,bnd)` also includes boundary information `bnd`, which `griddatan` sends on to the internal delaunay call (see `delaunay` or `delaunay3` for further information).

`griddatan(...,method,bnd,strategy)` allows the search strategy to be set explicitly. `strategy` can be either `'boxonly'` (default for linear interpolation), in which case `griddatan` returns `NaN` for all points outside the mesh, or `'closest'` (default for nearest neighbor interpolation), in which case `griddatan` locates the nearest element for all points.

Examples

```
rand('state',0);
p = 4*rand(100,2)-2;
z=sin(p(:,1)).*sin(p(:,2)).*exp(-p(:,1).^2-p(:,2).^2);
ti = -2:.1:2;
[xi,yi] = meshgrid(ti,ti);
ptsi = [xi(:),yi(:)];
zi = griddatan(p,z,ptsi,'linear',[],'closest');
```

```
g = griddatan(p,ptsi,'linear',[],'closest');
zi1 = tinterp(g,z);
z2 = sin(p(:,1)).*sin(p(:,2));
```

```
zi2 = tinterp(g,z2);
```

g is the struct:

```
g =
    method:  'linear'
  strategy:  'closest'
         t:  [182x3 double]
       ind:  [1681x1 double]
     coord:  [1681x3 double]
      size:  [1681 1]
```

See also griddata, griddata3, tinterp, tsearch, tsearchn, delaunay, delaunay3

Purpose

Compare matrices pointwise.

Synopsis

```
d = gt(a, b)
```

Description

`d = gt(a, b)` tests if the elements of the matrix `a` are pointwise greater than those of the matrix `b`. For each dimension, `a` and `b` must have the same size or either of them must have size 1. In the latter case, the unit dimension is expanded to the size of the nonunit dimension.

`gt(a, b)` is equivalent to `a > b`.

Examples

```
[2 3 5] > [1 3 7]
[5 -10 20] > 0
[1 2 3] > [1 ; 2]
```

See also

`eq`, `ge`, `le`, `lt`, `ne`

Purpose

Display help text.

Synopsis

```
help
help(topic)
help(obj)
```

Description

`help` displays a brief list of available help topics.

`help`(`topic`) displays the help text for a `topic`, which can be the name of an M-file or class on the path; in that case, the help text is the first contiguous block of comment lines in the file. If no matching file is found and there is a class instance variable `topic` in the current workspace, then the help for the class to which `topic` belong is displayed.

`help(obj)` where `obj` is an instance of a user-defined class displays the help text for the class to which `obj` belongs.

Purpose

Hessenberg form.

Synopsis

```
H = hess(A)
[Q,H] = hess(A)
[H,T,Q,Z] = hess(A,B)
```

Description

`H = hess(A)` returns a Hessenberg form of a square matrix `A`.

`[Q,H] = hess(A)` also returns a unitary matrix `Q` such that `A = Q*H*Q'` and `Q'*Q = I`.

`[H,T,Q,Z] = hess(A,B)` returns a Hessenberg matrix `H`, an upper triangular matrix `T`, and unitary matrices `Q` and `Z` such that `Q*A*Z = H` and `Q*B*Z = T`. `A` and `B` must be square matrices, and `B` must be upper triangular.

See also

`schur`

Purpose Convert hexadecimal strings to decimal integers.

Synopsis

```
d = hex2dec(str)
```

Description `d = hex2dec(str)` converts a string `str` representing a hexadecimal number to a decimal integer. `str` can also be a string matrix, in which case `hex2dec` converts each row, or a cell array of strings, in which case `hex2dec` converts each element.

Example `hex2dec('AE12')` converts the hexadecimal string `'AE12'` to its decimal equivalent `44562`.

See also `base2dec`, `bin2dec`, `hex2num`, `dec2base`, `dec2bin`, `dec2hex`, `num2hex`

Purpose Convert IEEE-754 hexadecimal strings to decimal numbers.

Synopsis

```
d = hex2num(str)
```

Description `d = hex2num(str)` converts a string str representing an IEEE-754 hexadecimal number to a decimal floating point number `d`. `str` can also be a string matrix, in which case `hex2num` converts each row, or a cell array of strings, in which case `hex2num` converts each element. If an input string is shorter than 16 characters, `hex2num` automatically pads it with zeros. For strings longer than 16 characters, `hex2num` ignores any beyond the first 16.

Example `hex2num({'3ff','4034','7ff'})` returns `[1 ; 20 ; Inf]`.

See also `base2dec`, `bin2dec`, `hex2dec`, `dec2base`, `dec2bin`, `dec2hex`, `num2hex`

Purpose

Remove or show hidden lines for a mesh plot.

Synopsis

```
hidden('on')
hidden('off')
hidden
hidden(ax, ...)
```

Description

`hidden('on')` turns the removal of hidden lines in the current axes on.

`hidden('off')` turns the removal of hidden lines in the current axes off.

`hidden` toggles hidden-line removal on or off.

`hidden(ax,...)` controls hidden-line removal in the axes `ax` instead of in the current axes.

See also

`mesh`

Purpose

Calculate histogram data or plot histogram.

Synopsis

```
n = hist(x)
n = hist(x,ni)
n = hist(x,c)
[n,cent] = hist(...)
```

Description

`n = hist(x)` divides the interval between the minimum and maximum value of `x` into 10 intervals of equal size. It returns the number of elements of `x` that fall into each of these bins. If `x` is a matrix, one count is done for each of the columns.

`n = hist(x,ni)` divides the interval into `ni` intervals of equal size.

`n = hist(x,c)` divides the interval between minimum and maximum value of `x` into intervals centered at the positions given by the vector `c`. It then returns the number of elements of `x` that falls into each of these bins.

`[n,cent] = hist(...)` also returns the centers for each of the subintervals in cent.

When `hist` is called without output arguments a histogram plot is produced from the generated data.

See also

`histc`

Purpose

Histogram count.

Synopsis

```
n = histc(x,edges)
n = histc(x,edges,dim)
[n,bin] = histc(...)
```

Description

`n = histc(x,edges)` returns the number of elements of `x` that fall in the bins specified by `edges`, which is a vector containing monotonically nondecreasing values. Thus `x(i)` falls in bin `k` if `edges(k) <= x(i) < edges(k+1)`. The last bin contains the number of elements that exactly match the last element of `edges`. To include all values (except `NaN`), put `-Inf` and `Inf` at the extremities of `edges`.

When `x` is a vector, `n` is the histogram count of `x`. When `x` is a matrix, `n` is matrix containing the histogram count of each column of `x`. When `x` is an n-dimensional array, `n` is the histogram count along the first nonsingleton dimension of `x`.

`n = histc(x,edges,dim)` returns the histogram count along the dimension `dim`.

`[n,bin] = histc(...)` also returns the index vector `bin`. For each element of `x`, `bin` contains the index into which it falls, or `0` if it does not fit into any bin.

Example

```
a=1:20;
```

`histc(a,[0 5 7 12 20])` returns `[4, 2, 5, 8, 1]`.

```
b=[-10.1 2 2.12 3 pi 1 1 0];
[res,ri] = histc(b,[-10,1,2,4])
```

returns `res = [1, 2, 4, 0]` and `ri = [0, 3, 3, 3, 3, 2, 2, 1]`.

See also

`hist`

Purpose

Concatenate matrices or cell arrays horizontally.

Synopsis

```
c = horzcat(arg1, ...)
```

Description

`c = horzcat(arg1, ...)` returns the horizontal concatenation of its input arguments, which need not be of the same type; if they differ, the result is the common base type of them all.

`horzcat(arg1, ...)` is equivalent to `[arg1 , ...]` or `cat(2, arg1, ...)`.

See also

`cat`, `vertcat`

Purpose Retain contents in an axes when adding new plots.

Syntax

```
hold('on')
hold('off')
```

Description

`hold('on')` specifies that the contents in the current axes should be kept when new plots are added.

`hold('off')` specifies that the current axes should be cleared automatically before adding new plots.

See also `ishold`

Purpose

Create a colormap with colors from red and yellow to white.

Synopsis

```
hot(n)
```

Description

`hot(n)` returns a colormap with `n` colors. It is a matrix with `n` rows and 3 columns with RGB values for the colors in the colormap. The colors are from red and yellow to white.

See also

`colormap`, `bone`, `cool`, `gray`, `grayprint`, `jet`, `hsv`, `pink`, `wavemap`

Purpose Create a colormap containing a HSV colormap.

Synopsis

```
hsv(n)
```

Description `hsv(n)` returns a colormap with `n` colors. It is a matrix with `n` rows and 3 columns with RGB values for the colors in the colormap. The HSV colormap varies the hue component of the hue-saturation-value color model. The colors begin with red, pass through yellow, green, cyan, blue, magenta, and return to red. The map is particularly useful for displaying periodic functions.

See also `colormap`, `bone`, `cool`, `gray`, `grayprint`, `jet`, `hot`, `pink`, `wavemap`

Purpose Get the imaginary unit.

Syntax `i`

Description `i` is the imaginary unit.

See also `imag`, `j`

Purpose

Compute the inverse fast Fourier transform of a vector or matrix.

Synopsis

```
f = ifft(v)
f = ifft(v, n)
f = ifft(v, n, dim)
f = ifft(v, [], dim)
f = ifft(..., 'symmetric')
```

Description

`f = ifft(v)` computes the inverse FFT along the first nonunit dimension of `v`.

`f = ifft(v, n)` computes the `n`-point inverse FFT. `v` is padded with zeros if it is shorter than `n` and truncated if it is longer.

`f = ifft(v, n, dim)` computes the `n`-point inverse FFT along the dimension `dim`.

`f = ifft(v, [], dim)` computes the inverse FFT along the dimension `dim`.

`f = ifft(..., 'symmetric')` computes the inverse FFT under the assumption that the input has Hermitian symmetry. As a result, the output `f` is always real.

See also

`fft`, `fft2`, `ifft2`, `fftn`, `ifftn`

Purpose

Compute the inverse 2D fast Fourier transform of a matrix.

Synopsis

```
f = ifft2(m)
f = ifft2(m, rows, cols)
f = ifft2(..., 'symmetric')
```

Description

`f = ifft2(m)` computes the inverse 2D FFT of the matrix `m`.

`f = ifft2(m, rows, cols)` computes the 2D inverse FFT of size `(rows, cols)`. The input matrix is truncated or padded with zeros if necessary.

`f = ifft2(..., 'symmetric')` computes the inverse 2D FFT under the assumption that the input has Hermitian symmetry. As a result, the output `f` is always real.

See also

`fft`, `ifft`, `fft2`, `fftn`, `ifftn`

Purpose

Compute the inverse n-dimensional fast Fourier transform of an array.

Synopsis

```
f = ifftn(m)
f = ifftn(m, size)
f = ifftn(..., 'symmetric')
```

Description

`f = ifftn(m)` computes the inverse n-dimensional FFT of the n-dimensional array `m`.

`f = ifftn(m, size)` computes the inverse n-dimensional FFT of size `size`. The input array is truncated or padded with zeros if necessary.

`f = ifftn(..., 'symmetric')` computes the inverse n-dimensional FFT under the assumption that the input has Hermitian symmetry. As a result, the output `f` is always real.

See also

`fft`, `ifft`, `fft2`, `ifft2`, `fftn`, `ifftn`

Purpose Undo the frequency-spectrum shift performed by `fftshift`.

Synopsis

```
f = ifftshift(m)
```

Description `f = ifftshift(m)` shifts the indices in each dimension circularly so that index `1` in `f` corresponds to the middle index in `m`. `ifftshift` is the inverse of `fftshift`.

See also `circshift`, `fft`, `ifft`, `fft2`, `fftshift`, `ifft2`, `fftn`, `ifftn`

Purpose Return imaginary part.

Synopsis

```
b = imag(a)
```

Description `b = imag(a)` returns the imaginary part of the complex matrix `a`.

See also `i`, `j`, `real`

Purpose

Show an image.

Synopsis

```
image(im)
```

Description

`image(im)` displays the matrix `im` as and image. `im` is either an `m`-by-`n` matrix or a `m`-by-`n`-by-3 matrix. If `im` is an `m`-by-`n` matrix the values in `im` are used as direct indices into the colormap. If `im` is an `m`-by-`n`-by-3 matrix it is treated as direct specification of colors and the last index corresponds to red, green and blue color component values respectively. The lower left corner in the image will be centered over (1,1) in the axes and the upper left corner over (`n`,`m`).

`image(x,y,im)` where `x` and `y` are two element vector places the corner of the image at (`x`(1),`y`(1)) and (`x`(2),`y`(2)) in the axes.

The property values for `patch` can also be given at the end of the command to control how the image is created.

`image` is suitable for displaying small images. Use `imshow` to display larger images.

See also

`imagesc`, `imread`, `imshow`, `imwrite`

Purpose Create an image icon that can be added to buttons and labels.

Synopsis

```
im = imageicon(name)
```

Description `im = imageicon(name)` creates an image icon using the image in the file `name`. That file can be an image of the types JPEG, GIF or PNG.

You can then add the image icon to buttons or labels when creating those objects.

See also `button`, `label`

Purpose

Show an image.

Synopsis

```
imagesc(im)
```

Description

`imagesc` has the same functionality as `image` except that a scaled mapping is used when mapping the data values to the color map.

`imagesc` is suitable for displaying small images. Use `imshow` to display larger images.

See also

`image`, `imread`, `imshow`, `imwrite`

Purpose Read an image from file.

Syntax

```
im=imread(filename)
```

Description `im=imread(filename)` reads the image from the file `filename` into the matrix `im`. `im` will be a height-by-width-by-3 matrix with RGB values for each pixel in the image. The RGB values will be between 0 and 255 and of the type uint8.

The extension of `filename` is used to determine the type of the image. On 32-bit Windows, Linux, Solaris, and Macintosh, the `imread` function supports bmp, jpeg, png, and tiff images. On other platforms jpeg and png images are supported.

See also `image`, `imagesc`, `imshow`, `imwrite`

Purpose

Show an image.

Synopsis

```
imshow(im)
imshow(im,colormap)
```

Description

`imshow(im)` show the image matrix `im` in a window. The matrix has dimensions height-by-width-3 where the last index corresponds to the RGB values for the color at each position. If `im` is a uint8 matrix the RGB values ranges from 0 to 255. If it is a double matrix it ranges from 0 to 1.

`im` can also be a width-by-height matrix. In that case the values of im are mapped to a colormap to create an image. By default the jet(1024) colormap is used. If you want to use another colormap you can pass that as the second argument to imshow.

See also

`image`, `imagesc`, `imread`, `imwrite`

Purpose

Write an image to file.

Synopsis

```
imwrite(im,filename)
```

Description

`imwrite(im,filename)` writes the image `im` to the file `filename`. IM is a height-by-width-by-3 matrix with RGB values for each pixel in the image. It can either be a uint8 matrix with RGB values between 0 and 255 or a double matrix with RGB values between 0 and 1.

The extension of `filename` is used to determine the type of the image. On 32-bit Windows, Linux, Solaris and Macintosh bmp, jpeg, png and tiff images are supported. On other platforms jpeg and png images are supported.

See also

`image`, `imagesc`, `imread`, `imshow`

Purpose Convert a 1D matrix index into an equivalent multidimensional index vector.

Synopsis

```
[ix1, ...] = ind2sub(sz, n)
```

Description `[ix1, ...] = ind2sub(sz, n)` returns the multidimensional index vector (`ix1, ...`) that is equivalent to the matrix index `n` for a matrix of size `sz`.

Example `[row, col] = ind2sub([4 5], 7)` results in `row = 3` and `col = 2` as `M(7)` and `M(3, 2)` refer to the same element for a 4 x 5 matrix `M`.

See also `sub2ind`

Purpose Get an infinite value.

Synopsis

```
inf
m = inf(n)
m = inf(sz)
m = inf(n1,n2,...)
```

Description `inf` returns an infinite floating-point value.

`m = inf(n)`, where `n` is an integer, returns an `n`x`n` all-inf matrix.

`m = inf(sz)`, where `sz` is a vector of integers, returns an all-inf matrix of size `sz`.

`m = inf(n1,n2,...)`, where `n`*i* are integers, returns an `n1`x`n2`x `...` all-inf matrix.

See also `nan`

Purpose

Create an in-line function.

Synopsis

```
f = inline(expr)
f = inline(expr, n)
f = inline(expr, in1, ...)
```

Description

`f = inline(expr)` creates an in-line function that computes the expression `expr`. The inputs to the function are the identifiers returned by `symvar`. If `symvar` finds no identifiers, the inline function takes a single input, `x`.

`f = inline(expr, n)`, where `n` is a nonnegative integer, creates an in-line function that computes the expression `expr`. Inputs to the function are `x`, `P1`, ..., `P`*n*.

`f = inline(expr, in1, ...)` creates an inline function that computes the expression `expr` using inputs `in1`, ...

Examples

`r = inline('sqrt(x.^2+y.^2)')` defines an in-line function with two inputs: `x` and `y`.

`r = inline('c*a+b', 'a', 'b', 'c')` defines an in-line function with three inputs in a specified order. Compare this with `r = inline('c*a+b')` which would assume that the function arguments are given in the order 'c', 'a', 'b'.

See also

`argnames`, `symvar`

Purpose

Ask for user input.

Synopsis

```
a = input(quest)
a = input(quest,'s')
```

Description

`a = input(quest)` displays the text in the string `quest` and waits for user input at the prompt. When the user presses the return key, it evaluates the entered text using the variables in the current context and returns the result in `a`.

`a = input(quest,'s')` returns the entered text as a string in `a` without trying to evaluate it.

Purpose

Get the name of an input to a user-defined function.

Synopsis

```
name = inputname(n)
```

Description

`name = inputname(n)` returns the name of the variable used as the n^{th} input to the user-defined function currently being executed. `n` must be an integer between 1 and `nargin`. If the *n*th input does not map to a variable in the calling workspace, the command returns `''`.

See also

`nargin`, `varargin`

Purpose Integer to string conversion.

Synopsis

```
str = int2str(n)
```

Description `int2str(n)` converts an integer or a 2D array of integers `n` into a string after rounding all noninteger values.

Example `int2str([1,-10,-1.4,-1.5,1.49,1.5,Inf,NaN])` returns the string `'1 -10 -1 -2 1 2 Inf NaN'`.

See also `num2str`, `sprintf`

Purpose

Convert a matrix into an integer matrix.

Synopsis

```
m = int8(a)
m = int16(a)
m = int32(a)
m = int64(a)
```

Description

`m = int8(a)` converts the real matrix `a` to an integer matrix by rounding each element to the closest 8-bit integer. Elements too large or too small to be represented using 8-bit integers are rounded to the largest and smallest 8-bit integers, respectively.

`int16`, `int32`, and `int64` instead round to 16-, 32-, and 64-bit integers, respectively.

The maximum and minimum values of *n*-bit integers are:.

TABLE 1-24:

FUNCTION	MIN	MAX
int8	-128	127
int16	-32768	32767
int32	-2147483648	2147483647
int64	-9223372036854775808	9223372036854775807

See also

`uint8, uint16, uint32, uint64`

Purpose

1D interpolation.

Synopsis

```
yi = interp1(x,y,xi)
yi = interp1(y,xi)
yi = interp1(...,method)
yi = interp1(...,method,extrap)
```

Description

`yi = interp1(x,y,xi)`, where *y=f(x)*, performs linear interpolation to determine *yi=f(xi)*. `x` must be a vector, and `y` must be an array whose first dimension equals the length of `x`.

`yi = interp1(y,xi)` performs linear interpolation using the default values `x = 1...n`, where `n` is the length of the first dimension of `y`.

`yi = interp1(...,method)` performs interpolation using a specific method: `'nearest'` (nearest neighbor interpolation), `'linear'`(linear interpolation), `'spline'` (cubic spline interpolation) or `'cubic'`(piecewise cubic Hermite interpolation, same as`'pchip'`).

`yi = interp1(...,method,extrap)` performs interpolation using a specific method for out-of-range values. `extrap` can be either the string `'const'` or `'extrap'` (denoting a constant extension or extrapolation, respectively) or a scalar, which is then returned for any out-of-range values. The default method for linear and nearest neighbor interpolation is to set all out of range values to `NaN`. The other interpolation methods use extrapolation.

Example

This example interpolates points from the sine curve.

```
x = linspace(0,2*pi,10); y = sin(x);
xi = linspace(0,2*pi,20);
yin = interp1(x,y,xi,'nearest');
yil = interp1(x,y,xi,'linear');
yis = interp1(x,y,xi,'spline');
```

See also

`interp2`, `interp3`, `spline`, `pchip`

Purpose

2D interpolation.

Synopsis

```
zi = interp2(x,y,z,xi,yi)
zi = interp2(z,xi,yi)
zi = interp2(...,method)
zi = interp2(...,method,extrap)
```

Description

`zi = interp2(x,y,z,xi,yi)`, where *z = f(x,y)*, performs linear interpolation to determine *zi = f(xi,yi)*. If `x` and `y` are vectors of length `n` and `m` respectively, then `z` must be a matrix of size `mxn`. `x` and `y` can also be grid matrices as described in meshgrid. `xi` and `yi` can be matrices or vectors of different orientations.

`zi = interp2(z,xi,yi)` performs linear interpolation using the default values `x = 1...n` and `y = 1...m`.

`zi = interp2(...,method)` performs interpolation using a specific method: `'nearest'` (nearest neighbor interpolation) or `'linear'`(linear interpolation).

`zi = interp2(...,method,extrap)` performs interpolation using a specific method for out-of-range values. `extrap` can be either the string `'const'` or `'extrap'` (denoting a constant extension or extrapolation, respectively) or a scalar, which is then returned for any out-of-range values. The default method is to set all out-of-range values to `NaN`.

See also

`interp1`, `interp3`

Purpose

3D interpolation.

Synopsis

```
vi = interp3(x,y,z,v,xi,yi,zi)
vi = interp3(v,xi,yi,zi)
vi = interp3(...,method)
vi = interp3(...,method,extrap)
```

Description

`vi = interp3(x,y,z,v,xi,yi,zi)`, where *v = f(x,y,z)*, performs linear interpolation to determine *vi = f(xi,yi,zi)*. If `x`, `y`, and `z` are vectors of length `n`, `m`, and `p` respectively, then `v` must be a matrix of size `mxnxp`. `x`, `y`, and `z` can also be grid matrices as described in meshgrid. `xi`, `yi`, and `zi` can be matrices or vectors of different orientation.

`vi = interp3(v,xi,yi,zi)` performs linear interpolation using the default values `x = 1...n`, `y = 1...m`, and `z = 1...p`.

`vi = interp3(...,method)` performs interpolation using a specific method: `'nearest'` (nearest neighbor interpolation) or `'linear'`(linear interpolation).

`vi = interp3(...,method,extrap)` performs interpolation using a specific method for out-of-range values. `extrap` can be either the string `'const'` or `'extrap'` (denoting a constant extension or extrapolation, respectively) or a scalar, which is then returned for any out-of-range values. The default method is to set all out-of-range values to `NaN`.

See also

`interp1`, `interp2`

Purpose

Set intersection.

Synopsis

```
c = intersect(a,b)
c = intersect(a,b,'rows')
[c,ai,bi] = intersect(...)
```

Description

`c = intersect(a,b)` returns the intersection of `a` and `b`, that is, the elements contained in both `a` and `b`, both of which can be either arrays or cell arrays of strings.

`c = intersect(a,b,'rows')`, where `a` and `b` must be 2D matrices, returns the row intersection, that is the rows common to both `a` and `b`, both of which must have the same number of columns.

`[c,ai,bi] = intersect(...)` also returns index vectors `ai` and `bi`, which contain the linear indices of the elements of `c` in `a` and `b`, respectively.

In addition to the fixed arguments, additional property-value pairs can be given at the end of the command.

TABLE 1-25: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
`sort`	'on' | 'off'	`'on'`	Controls whether or not output should be sorted.

Examples

```
a = [1 2 0 1 2 3];
b = [2 4 5 7 0 8];
intersect(a,b) returns [0, 2]
```

```
a = [1 2 3; 2 3 1; 3 4 5; 5 4 3; 4 3 5;1 3 3];
b = [3 4 5; 3 4 5; 1 2 2; 4 3 5];
intersect(a,b,'rows') returns [3, 4, 5 ; 4, 3, 5]
```

```
a = {'green','yellow','blue','green','red'};
b = {'red','green'};
[c,ai,bi] = intersect(a,b) returns c = {'green', 'red'}
```

`[c1,ai1,bi1] = intersect(a,b,'sort','off')` returns the same result unsorted.

See also

`ismember`, `setdiff`, `setxor`, `union`, `unique`

Purpose

Get the largest and smallest values that can be represented as *k*-bit integers.

Synopsis

```
m = intmax(type)
m = intmin(type)

m = intmax
m = intmin
```

Description

`m = intmax(type)` and `m = intmin(type)` return the smallest and largest values, respectively, that can be returned by the integer conversion function `type`. Possible values for `type` are `'int8'`, `'int16'`, `'int32'`, `'int64'`, `'uint8'`, `'uint16'`, `'uint32'`, and `'uint64'`.

`m = intmax` and `m = intmin` are equivalent to `m = intmax('int32')` and `m = intmin('int32')`, respectively.

See also

`int8`, `int16`, `int32`, `int64`, `uint8`, `uint16`, `uint32`, `uint64`

Purpose Matrix inverse.

Synopsis `inv(A)`

Description `inv(A)` computes the inverse of the matrix A.

Purpose Test if a value belongs to a class.

Synopsis

```
d = isa(val, classname)
```

Description `d = isa(val, classname)` returns `true` if `val` belongs to the class `classname`, otherwise `false`.

See also `class`

Purpose Test if a value is a cell array.

Synopsis

```
d = iscell(c)
```

Description `d = iscell(c)` returns `true` if `c` is a cell array, otherwise `false`.

Purpose Test if a value is a cell array of strings.

Synopsis

```
d = iscellstr(c)
```

Description `d = iscellstr(c)` returns `true` if `c` is a cell array of strings, otherwise `false`.

Purpose Test if a value is a character matrix.

Synopsis

```
d = ischar(c)
```

Description `d = ischar(c)` returns `true` if `c` is a character matrix, otherwise `false`.

Purpose Test if a directory exists.

Synopsis

```
d = isdir(name)
```

Description `d = isdir(name)` returns `true` there is a readable directory called `name`.

Purpose Test if a value is empty.

Synopsis

```
d = isempty(m)
```

Description `d = isempty(m)` returns `true` if any dimension of `m` has size 0, otherwise `false`.

Purpose Test if values are equal.

Synopsis

```
d = isequal(a, b, ...)
```

Description `d = isequal(a, b, ...)` returns `true` if all input arguments are equal, otherwise `false`. For matrices and cell arrays, equality means that the sizes and all elements are equal. For structures to be equal, they must have the same fields and the values of the fields must be equal. In all these cases, the equality tests for elements and fields are performed by recursively invoking `isequal`.

Note `isequal(NaN, NaN)` returns `false`. If you want to consider NaNs as being equal, use `isequalwithequalnans`.

See also `isequalwithequalnans`

Purpose

Test if two values are equal without special semantics for NaN.

Synopsis

```
d = isequalwithequalnans(a, b, ...)
```

Description

`d = isequalwithequalnans(a, b, ...)` returns `true` if all input arguments are equal, otherwise `false`. For matrices and cell arrays, equality means that the sizes and all elements are equal. For structures to be equal, they must have the same fields and the values of the fields must be equal. In all these cases, the equality tests for elements and fields are performed by recursively invoking `isequalwithequalnans`.

See also

`isequal`

Purpose Test if a structure has a certain field.

Synopsis `d = isfield(s, name)`

Description `d = isfield(s, name)` returns `true` if `s` is a structure that contains a field called `name`, otherwise `false`.

See also `getfield`, `setfield`

Purpose Test if elements of a matrix are finite.

Synopsis

```
d = isfinite(a)
```

Description `d = isfinite(a)`, for a matrix `a`, returns a logical array of the same size as `a`. The elements in `d` are false if the corresponding position in `a` is Inf, -Inf, or NaN, otherwise true. For complex matrices, this criterion is applied to the real and imaginary parts.

See also `isinf`, `isnan`

Purpose Test if a variable is global.

Synopsis

```
d = isglobal(name)
```

Description `d = isglobal(name)` returns `true` if the workspace contains a global variable `name`, otherwise `false`.

Purpose Test if a variable is a graphics handle.

Synopsis

```
is = ishandle(h)
```

Description `is = ishandle(h)` returns a logical array of the same length as `h` with true for the entries in `h` that are graphics handles.

Purpose

Check if hold is on.

Synopsis

```
h = ishold
h = ishold(ax)
```

Description

`h = ishold` returns 1 if hold is on in the current axes and 0 otherwise. When hold is on, graphics commands that plot into the axes add data to the existing plot instead of replacing it.

`h = ishold(ax)` returns the hold state of the axes `ax`.

See also

`hold`

Purpose Test if elements of a matrix are infinite.

Synopsis

```
d = isinf(a)
```

Description `d = isinf(a)`, for a matrix `a`, returns a logical array of the same size as `a`. The elements in `d` are true if the corresponding position in `a` is Inf or -Inf, otherwise false. For complex matrices, this criterion is applied to the real and imaginary parts.

See also `isfinite`, `isnan`

Purpose Test if a value is a Java object.

Synopsis

```
d = isjava(jo)
```

Description `d = isjava(jo)` returns `true` if `jo` is a Java object, otherwise `false`.

Purpose

Test if a string is a reserved word.

Synopsis

```
iskeyword
d = iskeyword(str)
```

Description

`iskeyword` returns a cell array containing all reserved words.

`d = iskeyword(str)` returns `true` if `str` is a reserved word, otherwise false.

Purpose Test for letters.

Synopsis

```
x = isletter(str)
```

Description `x = isletter(str)`, where `str` is a character array, returns a logical array `x` of the same size as `str`, containing `true` for each character that is a letter of the alphabet and `false` otherwise.

Example `isletter('ab89*%')` returns `[true, true, false, false, false, false]`.

See also `ischar`, `isspace`

Purpose Test if a value is logical.

Synopsis

```
d = islogical(a)
```

Description `d = islogical(a)` returns `true` if `a` is a logical matrix, otherwise `false`.

Purpose

Determine set members.

Synopsis

```
c = ismember(a, b)
c = ismember(a, b, 'rows')
[c, ai] = ismember(a, b, ...)
```

Description

`c = ismember(a,b)` determines which elements of a belong to set `b`. `a` and `b` can be either arrays or cell arrays of strings. `c` is an array of the same size as `a`, containing logical `true` and `false` depending on whether or not corresponding element of `a` belongs to `b`.

`c = ismember(a,b,'rows')`, where `a` and `b` must be two-dimensional matrices, determines which rows of `a` belong `b`. `a` and `b` must have the same number of columns.

`[c,ai] = ismember(...)` also returns index vector `ai`, containing the linear indices of the last occurrences of elements in `a` that are in `b`, or `zero` otherwise.

Examples

```
a = [1 2 0 1 2 3];
b = [2 4 5 7 0 8];
```

`[c,ai] = ismember(a,b)` returns `c = [false, true, rue, false, true, false]` and `ai = [0, 1, 5, 0, 1, 0]`.

```
a = [1 2 3; 2 3 1; 3 4 5; 5 4 3; 4 3 5;1 3 3];
b = [3 4 5; 3 4 5; 1 2 2; 4 3 5];
```

`ismember(a,b,'rows')` returns `[0 ; 0 ; 1 ; 0 ; 1 ; 0]`.

```
a = {'green','yellow','blue','green'};
b = {'red','purple','yellow'};
```

`ismember(a,b)` returns `[0, 1, 0, 0]`.

See also

`intersect`, `setdiff`, `setxor`, `union`, `unique`

Purpose Test if elements of a matrix are NaN.

Synopsis

```
d = isnan(a)
```

Description `d = isnan(a)`, for a matrix `a`, returns a logical array of the same size as `a`. The elements in `d` are true if the corresponding position in `a` is NaN, otherwise false. For complex matrices, this criterion is applied to the real and imaginary parts.

See also `isfinite`, `isinf`

Purpose Test if a value is numeric.

Synopsis

```
d = isnumeric(a)
```

Description `d = isnumeric(a)` returns `true` if `a` is a real or complex matrix, otherwise `false`.

Purpose Test if a value is an object.

Synopsis `d = isobject(obj)`

Description `d = isobject(obj)` returns `true` if `obj` is a COMSOL Script object, otherwise `false`.

Purpose Test if COMSOL Script is running on a PC.

Synopsis

```
d = ispc
```

Description `d = ispc` returns `true` if COMSOL Script is running on a PC, otherwise `false`.

See also `isunix`

Purpose Test for prime numbers

Synopsis

```
y = isprime(x)
```

Description `y = isprime(x)` tests each element of the array `x` for prime numbers. `y` is an array of the same size as `x` that contains `true` for each element of `x` that is prime and `false` otherwise.

Example `isprime([3 17 19 231 86421 99823])` returns `[true, true, true, false, false, true]`.

See also `factor`, `primes`

Purpose Test if a value is a real matrix.

Synopsis

```
d = isreal(a)
```

Description `d = isreal(a)` returns `true` if `a` is a double matrix, a character matrix or a logical matrix, or a Java object, otherwise it returns `false`.

Purpose

Test if a value is a scalar.

Synopsis

```
d = isscalar(a)
```

Description

`d = isscalar(a)` returns `true` if all dimensions of `a` have length 1, otherwise `false`.

Purpose Test for white space.

Synopsis

```
x = isspace(str)
```

Description `x = isspace(str)`, where `str` is a character array, returns a logical array `x` of the same size as `str`, containing `true` for each character that is a white-space character and `false` otherwise.

White-space characters are defined as the following ASCII values: 9 (horizontal tabulation), 10 (new line), 11 (vertical tab), 12 (form feed), 13 (carriage return), and 32 (space).

Example `isspace(['a b',char([9 10 11 12 13 32]),'/*'])` returns `[false,true, false,true,true,true,true,true,true,false,false]`

See also `ischar`, `isletter`

Purpose Test if a value is a sparse matrix.

Synopsis

```
d = issparse(a)
```

Description `d = issparse(a)` returns `true` if `a` is a sparse matrix, otherwise `false`.

Purpose Test if a value is a character matrix.

Synopsis

```
d = isstr(c)
```

Description `d = isstr(c)` returns `true` if `c` is a character matrix, otherwise `false`.

Remark `isstr` is equivalent to `ischar`.

Purpose Test if a value is a structure.

Synopsis

```
d = isstruct(s)
```

Description `d = isstruct(s)` returns `true` if `s` is a structure, otherwise `false`.

Purpose Test if COMSOL Script is running under Unix.

Synopsis

```
d = isunix
```

Description `d = isunix` returns `true` if COMSOL Script is running under Unix, otherwise `false`.

See also `ispc`

Purpose

Test if a string can be used as a variable name.

Synopsis

```
d = isvarname(str)
```

Description

`d = isvarname(str)` returns `true` if `str` is a string that contains a valid variable name, otherwise `false`. A variable name can contain only letters, digits, and underscores, and it must start with a letter.

Purpose Test if a value is a vector.

Synopsis

```
d = isvector(a)
```

Description `d = isvector(a)` returns `true` if `a` has the size $(1, n)$ or $(n, 1)$, otherwise `false`.

Purpose Get the imaginary unit.

Syntax `j`

Description `j` is the imaginary unit.

See also `i`, `imag`

Purpose Create an array of Java objects.

Syntax `j = javaArray(cls, dim1, ...)`

Description `j = javaArray(cls, dim1, ...)`, where `cls` is a Java class name, creates an array of size `(dim1, ...)` of Java objects of class `cls`.

See also `javaMethod`, `javaDeclare`,`javaObject`

Purpose

Load declarations of Java methods.

Syntax

```
javaDeclare(file)
javaDeclare(file, replace)
javaDeclare(methods, file)
```

Description

`javaDeclare(file)`, where `file` is a file name, loads declarations of Java methods from `file`. The declaration file should contain method declarations written in Java.

`javaDeclare(file, replace)`, replaces the existing set of Java declarations with those in `file` if `replace` is `true` and appends to the existing set if replace is false.

`javaDeclare(methods, file)`, where `methods` is a cell array of Java class names and `file` is a file name, writes the declarations of all public methods found in the class name list to `file`.

Only public methods can be accessed through the Java interface. Methods with no visibility specified are assumed to be public.

For overloaded methods and constructors, the number of arguments must differ: It is not possible to declare two methods in a class that have the same name and number of arguments.

Example

The file my.decls has the following contents:

```
// The file can contain comments.
/* Both types of comments can be used. */
java.lang.String(String);
static java.lang.String.valueOf(double);
int java.lang.String.indexOf(java.lang.String);
```

`javaDeclare('my.decls')` adds the declarations of three member methods of `java.lang.String` to the declaration database.

See also

`javaArray`, `javaMethod`,`javaObject`

Purpose

Invoke a Java method.

Syntax

```
d = javaMethod(method, cls)
d = javaMethod(method, cls, arg1, ...)
d = javaMethod(method, obj)
d = javaMethod(method, obj, arg1, ...)
```

Description

`d = javaMethod(method, cls, arg1, ...)`, where `method` is a method name and `cls` a class name, invokes the static method called `method` in the class `cls` with the arguments `(arg1, ...)` and returns the result.

`d = javaMethod(method, obj, arg1, ...)`, where `method` is a method name and `obj` a Java object, invokes the member function called `method` in the Java object `obj` with the arguments `(arg1, ...)` and returns the result.

See also

`javaArray`, `javaDeclare`, `javaObject`

Purpose Create Java object.

Syntax

```
j = javaObject(cls, ...)
```

Description `j = javaObject(cls, ...)`, where `cls` is a Java class name, creates a Java object of class `cls`. The arguments, if any, after `cls` are passed on to the constructor.

See also `javaArray`, `javaDeclare`, `javaMethod`

Purpose Create a colormap with all colors from blue to red.

Synopsis `jet(n)`

Description `jet(n)` returns a colormap with `n` colors. It is a matrix with `n` rows and 3 columns with RGB values for the colors in the colormap. The colors are all colors from blue to red.

See also `colormap`, `bone`, `cool`, `gray`, `grayprint`, `hot`, `hsv`, `pink`, `wavemap`

Purpose

Manually place a breakpoint in the code.

Synopsis

```
keyboard
```

Description

When you place `keyboard` somewhere in a script or function, execution stops on that line just as if you had placed a break point there. Ordinary debugging commands can then be used.

Purpose Kronecker tensor product.

Synopsis

```
C = kron(A,B)
```

Description `C = kron(A,B)` computes the Kronecker tensor product of matrices `A` and `B`. If `A` is an *mxn* matrix and `B` is a *pxq* matrix, then the Kronecker product of `A` and `B` is the *mpxnq* block matrix.

Example `kron([1 2;0 2],[2,3,4;1,1,1])` returns a 4x6 matrix `[2 3 4 4 6 8;1 1 1 2 2 2;0 0 0 4 6 8;0 0 0 2 2 2]`.

Purpose

Create a label.

Synopsis

```
l = label(text,...)
l = label(...)
```

Description

`l = label(text)` creates a label with the specified text.

In addition to the fixed arguments, additional property-value pairs can be given at the end of the command to further control how the label is created.

PROPERTY	VALUE	DESCRIPTION
`image`	iconimage	An image to display on the label.
`text`	string	A text to display on the label.

See also the reference entry for `component` for property-value pairs and methods that are valid for all components.

See also

`component`

Purpose

Get or set the current error message.

Syntax

```
lasterr(msg)
lasterr(msg, id)
msg = lasterr
[msg, id] = lasterr
```

Description

`lasterr(msg)`, where `msg` is a string, sets the current error message to `msg`.

`lasterr(msg, id)`, where `msg` and `id` are strings, sets the current error message to `msg` and the current error ID to `id`.

`msg = lasterr` returns the current error message.

`[msg, id] = lasterr` returns the current error message and error ID.

See also

`lasterror`

Purpose

Get or set the current error message.

Syntax

```
lasterror(s)
s = lasterror
```

Description

`lasterror(s)`, where `s` is a structure, sets the current error message to `s.message` and sets the current error ID to `s.identifier`.

`s = lasterror` returns a structure containing the current error message in the field `message`, the error ID in the field `identifier`, and a detailed error trace in the field `details`.

See also

`lasterr`

Purpose Least common multiple.

Synopsis

```
l = lcm(a,b)
```

Description `l = lcm(a,b)` computes the least common multiple of the elements of arrays `a` and `b`. `a` and `b` must be the same size, or either one can be a scalar.

Example `lcm([120 3 7],9)` returns `[360, 9, 63]`.

See also `gcd`

Purpose

Divide matrices pointwise.

Synopsis

```
d = ldivide(a, b)
```

Description

`d = ldivide(a, b)` computes the pointwise ratio between the two matrices `b` and `a`. For each dimension, `a` and `b` must have the same size or either of them must have size 1. In the latter case, the unit dimension is expanded to the size of the nonunit dimension.

`ldivide(a, b)` is equivalent to `a.\b`.

Examples

```
[1 10 100].\[3 4 5]
```

```
10.\[2 3 5]
```

See also

`minus`, `plus`, `rdivide`, `times`

Purpose Compare matrices pointwise.

Synopsis

```
d = le(a, b)
```

Description `d = le(a, b)` tests if the elements of the matrix `a` are pointwise less than or equal to those of the matrix `b`. For each dimension, `a` and `b` must have the same size or either of them must have size 1. In the latter case, the unit dimension is expanded to the size of the nonunit dimension.

`le(a, b)` is equivalent to `a <= b`.

Examples

```
[2 3 5] <= [1 3 7]
[5 -10 20] <= 0
[1 2 3] <= [1 ; 2]
```

See also `eq`, `ge`, `gt`, `lt`, `ne`

Purpose

Display a legend with a plot.

Synopsis

```
legend(leg1,leg2,leg3,...)
legend('show')
legend('hide')
legend(ax,...)
```

Description

`legend(leg1,leg2,leg3,...)` displays the strings `leg1`, `leg2`, `leg3` and so on as legends with the current plot.

`legend('show')` turns on the display of legends.

`legend('hide')` turns off the display of legends.

`legend(ax,...)` controls legends in the axes `ax` instead of in the current axes.

See also

`plot`

Purpose Get the largest dimension of a matrix.

Syntax
```
l = length(a)
```

Description `l = length(a)`, for a nonempty matrix `a`, returns the maximum length of any dimension of `a`, that is, `max(size(a))`. If `a` is empty, 0 is returned.

See also `size`

Purpose

Create a light.

Synopsis

```
light(...)
```

Description

`light`(...) adds a light to a plot. Several different types of light can be created. To control which type of light to create and what properties to give it, use the properties in the following table.

`h = light(...)` also returns a handle to the created light.

TABLE 1-26: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
`color`	colorspec	`w`	A string or an RGB triplet specifying the color of the light. If it is a string it is one of the letters `r`, `g`, `b`, `c`, `m`, `y` or `k`, meaning red, green, blue, cyan, magenta, yellow and black respectively.
`concentration`	A real value between 0 and 128	0	The concentration for a spotlight.
`direction`	A three element array.	`[0 0 1]`	The direction for a directional light or a spot light.
`parent`	Axes handle	`gca`	What axes to add the light to.
`position`	A three element array.	`[0 0 0]`	The position for a point light or a spotlight.
`style`	`ambient | directional | point | spot`	`point`	The type of light to create.
`spread`	A real number between 0 and pi.	`pi`	The spread angle for a spotlight.

See also

`lighting`, `material`, `patch`, `surface`

Purpose Turn on and off scene light.

Synopsis

```
lighting('phong')
lighting('none')
```

Description

`lighting('phong')` turns on scene lights in the current axes.

`lighting('none')` turns off scene lights in the current axes.

`lighting(ax,...)` controls scene lights in the axes AX instead of in the current axes.

See also `light`, `material`, `patch`, `surface`

Purpose

Create a line.

Syntax

```
line(x,y)
line(x,y,z)
```

Description

`line(x,y)` connects the coordinates in the vectors `x` and `y` to form a line. If `x` and `y` are matrices, one connected line is created for each column in the matrices.

`line(x,y,z)` adds a line in 3D.

`h = line(...)` returns a handle to the created line.

In addition to the fixed arguments, additional property-value pairs can be given at the end of the command to further control how the line is created.

TABLE 1-27: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
`color`	colorspec	`k`	A string or an RGB triplet specifying the color of the line. If it is a string it is one of the letters `r`, `g`, `b`, `c`, `m`, `y` or `k`, meaning red, green, blue, cyan, magenta, yellow, and black, respectively.
`linestyle`	One of the strings `-,:,-.,--`	`-`	String representing solid, dotted, dash-dot, and dashed line styles, respectively
`linewidth`	positive scalar	1	The width of the line
`marker`	`.,v,+,o,*,s,p`		The marker to show along the line. Only available for 2D lines.
`parent`	Axes handle	`gca`	What axes to add the line to.

See also

`plot`, `plot3`

Purpose Create vector containing linearly spaced values.

Syntax

```
v = linspace(a, b, n)
v = linspace(a, b)
```

Description `v = linspace(a, b, n)`, where `a` and `b` are real or complex scalars, creates a vector containing `n` elements linearly spaced between `a` and `b`, that is, `[a a+(b-a)/(n-1) ... b]`.

`v = linspace(a, b)` is equivalent to `v = linspace(a, b, 100)`.

See also `logspace`

Purpose

Create a list box.

Synopsis

```
c = listbox(...)
```

Description

`c = listbox(...)` creates a list box. Values and descriptions for the values in the list box are specified using the properties in the following table

TABLE 1-28: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DESCRIPTION
`descr`	cell array of strings	The strings to display in the list box. If not given the strings specified as `items` will be displayed in the list box.
`items`	cell array of strings	String representing the value corresponding to each entry in the list box. Can then be used to easily set and get the value of the list box using strings instead of indices.

The function returns a list-box object that can then be further manipulated using the methods in the following table.

TABLE 1-29: METHODS FOR MANIPULATING A LISTBOX OBJECT.

METHOD	DESCRIPTION
`addListSelectionListener(name)`	Specifies that the function with the given name should be run when the selection in the list box changes.
`getSelectedIndex`	Returns an index to the currently selected item in the list box.
`getSelectedIndices`	Returns an array with indices to the selected items in the list box.
`getValue`	Returns a string corresponding to the currently selected item in the list box.
`setItems(items)`	Sets the items to display in the list box by passing a cell array of strings.
`setItems(items,descr)`	Sets the descriptions to display in the list box and their corresponding values by passing two cell arrays of strings.
`setSelectedIndex(ind)`	Selects the item with the specified index in the list box.
`setSelectedIndices(ind)`	Selects the items corresponding to the indices in the vector ind.
`setValue(value)`	Selects the item with the specified value in the list box.

See also the reference entry for `component` for property-value pairs and methods that are valid for all components.

See also `component`, `combobox`

Purpose

Load a workspace from a file.

Syntax

```
load(filename)
load(filename, var1, ...)
s = load(filename)
s = load(filename, var1, ...)

load(..., '-mat')
s = load(..., '-mat')

load(..., '-ascii')
s = load(..., '-ascii')
```

Description

`load(filename)`, where `filename` is a string, loads variables and their values from the file `filename`. Existing workspace variables are overwritten.

`load(filename, var1, ...)` loads only the variables `var1`, ... into the workspace. `*` can be used as wildcard character unless `'-ascii'` is given..

`s = load(filename)` loads variables and values into a structure. Each variable corresponds to a field in the structure.

`s = load(filename, var1, ...)` loads only the variables `var1`, ...

`load(..., '-mat')` loads the file as a MATLAB workspace file. (The default behavior is to load the file as a Comsol workspace file.)

`load(..., '-ascii')` reads a text representation of a real matrix from `filename` into a workspace variable with a name derived from `filename`. Each row of the file corresponds to one row in the matrix; hence all rows must have the same number of columns.

`s = load(..., '-ascii')` reads a text representation of a real matrix and returns the matrix.

See also

`save`

Purpose Compute natural logarithm.

Syntax

```
b = log(a)
```

Description `b = log(a)` returns the natural logarithm of the matrix `a` pointwise.

See also `log10`

Purpose Compute a base-10 logarithm.

Syntax

```
b = log10(a)
```

Description `b = log10(a)` returns the base-10 logarithm of the matrix `a` pointwise.

See also `log`

Purpose Compute a base-2 logarithm.

Syntax

```
b = log2(a)
[m, e] = log2(mat)
```

Description `b = log2(a)` returns the base-2 logarithm of the matrix `a` pointwise.

`[m, e] = log2(a)` returns the mantissa `m` and exponent e pointwise for the matrix `a`. They satisfy the relation `a = m.*2.^e`.

See also `log`, `log10`

Purpose

Convert a matrix to a logical matrix.

Syntax

```
l = logical(a)
```

Description

`l = logical(a)` returns a logical matrix with the same size as `a` that is the result of element-wise converting `a` to logical values.

loglog

Purpose	Create a plot with log scales on both the x-axis and the y-axis.
Synopsis	`loglog(...)`
Description	`loglog(...)` has the same functionality as `plot(...)` with the addition that it uses log scales on both the x-axis and the y-axis.
See also	`plot`

Purpose Matrix logarithm.

Syntax `F = logm(A)`

Description `F = logm(A)` returns the principal logarithm of a square matrix `A`.

See also `expm`, `funm`

Purpose

Create a vector containing logarithmically spaced values.

Syntax

```
v = logspace(a, b, n)
v = logspace(a, b)
```

Description

`v = logspace(a, b, n)`, where `a` and `b` are real or complex scalars, returns a vector containing `n` elements logarithmically spaced between `10^a` and `10^b`, that is, `[10^a 10^(a+(b-a)/(n-1) ... 10^(b)]`. If `b` is `pi`, then `10^b` is replaced with `pi` in these expressions.

`v = logspace(a, b)` is equivalent to `v = logspace(a, b, 50)`.

See also

`linspace`

Purpose Search M-files.

Synopsis `lookfor(str)`

Description `lookfor(str)` searches for string `str` in the first line of all .M files on the current path and displays matches.

Purpose Convert string to lower case

Syntax

```
s2 = lower(s1)
```

Description `s2 = lower(s1)` converts the characters in the string `s1` to lower case. `s1` can also be a cell array of strings. In that case, a new cell array is returned where each of the strings has been converted to lower case.

See also `upper`

Purpose

Get a list of the files in a directory.

Synopsis

```
ls
ls(d)
f = ls
f = ls(d)
```

Description

`ls` is a synonym for `dir`.

See also

`dir`

Purpose Compare matrices pointwise.

Synopsis

```
d = lt(a, b)
```

Description `d = lt(a, b)` tests if the elements of the matrix `a` are pointwise less than those of the matrix `b`. For each dimension, `a` and `b` must have the same size or either of them must have size 1. In the latter case, the unit dimension is expanded to the size of the nonunit dimension.

`lt(a, b)` is equivalent to `a < b`.

Examples

```
[2 3 5] < [1 3 7]
[5 -10 20] < 0
[1 2 3] < [1 ; 2]
```

See also `eq`, `ge`, `gt`, `le`, `ne`

Purpose

Compute the LU factorization of matrix.

Synopsis

```
lu(A)
[L,U] = lu(A)
[L,U,P] = lu(A)

[L,U,P,Q] = lu(A)
[L,U,P,Q] = lu(A, thresh)
[L,U,P,Q,R] = lu(A)
[L,U,P,Q,R] = lu(A, thresh)
```

Description

The following syntaxes can be used for a full matrix `A`:

`lu(A)` returns a matrix containing the lower-triangular `L` and the upper-triangular `U` above and below the diagonal, respectively. It is not guaranteed that `A = L*U`.

`[L,U] = lu(A)` returns an upper-triangular `U` and an `L` that is the product of a lower-triangular matrix and a permutation matrix such that `L*U = A`.

`[L,U,P] = lu(A)` returns a lower-triangular `L`, an upper-triangular `U`, and a permutation matrix `P` such that `P*U = L*U`.

The following syntaxes can be used for a sparse matrix `A`:

`[L,U,P,Q] = lu(A)` returns a lower-triangular `L`, an upper-triangular `U`, and permutation matrices `P` and `Q` such that `P*A*Q = L*U`.

`[L,U,P,Q,R] = lu(A)` returns a lower-triangular `L`, an upper-triangular `U`, permutation matrices `P` and `Q`, and a diagonal matrix `R` such that `P*R*A*Q = L*U`. This syntax is more numerically stable than `[L,U,P,Q] = lu(A)`.

`[L,U,P,Q,...] = lu(A, thresh)` uses a threshold `thresh` when pivoting. The default threshold is 0.1. When selecting a pivot element in a column, eligible elements are those that are at least `thresh` times the largest absolute value in that column.

Purpose Create a cell array from a matrix.

Synopsis

```
c = mat2cell(a, part1, part2,...)
```

Description `c = mat2cell(a, part1, part2, ...)` creates a cell array from the matrix `a` where the first dimension of `a` is split into `length(part1)` parts of sizes `part1(1)`, `part1(2)`, and so on. For the partition to be valid, `sum(part`*i*`) == size(a, `*i*`)` must hold for all *i*.

Example `c = mat2cell(rand(5, 15), [2 3], [4 5 6])` creates a 2 x 3 cell array where `c(1,1)` is the 2 x 4 submatrix in the upper-left corner of the random matrix.

See also `cell2mat`

Purpose

Create a string from a value.

Synopsis

```
s = mat2str(a)
```

Description

`s = mat2str(a)` returns a textual expression that evaluates to `a`.

Examples

`mat2str([1 2+3 ; 5+7 11])` returns the string `[1, 5; 12, 11]`.

```
a.b = 12;
a.s = {'abc'};
mat2str(a)
```

returns the string `struct('b', {12}, 's', {{'abc'}})`

Purpose

Control the material for surface reflectance.

Synopsis

```
material(...)
```

Description

`material(...)` specifies properties for the material to use for surface reflectance in the current axes. Use property-value pairs from the following table to specify the type of material to create.

TABLE 1-30: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DESCRIPTION
`Ambient`	colorspec	Specifies the ambient color. A string or an RGB triplet specifying the color of the line. If it is a string it is one of the letters `r`, `g`, `b`, `c`, `m`, `y` or `k`, meaning red, green, blue, cyan, magenta, yellow, and black, respectively.
`Diffusive`	colorspec	Specifies the diffusive color.
`Emissive`	colorspec	Specifies the emissive color.
`Specular`	colorspec	Specifies the specular color.
`Shininess`	Real number >0 and <128.	Specifies the shininess.

`material(ax,...)` controls the material in the axes `ax` instead of in the current axes.

See also

`lighting`, `light`, `patch`, `surface`

Purpose

Compute the maximum value of an array.

Synopsis

```
y = max(x)
y = max(x,[],dim)
[y,i] = max(x,...)
z = max(x,y)
```

Description

`y = max(x)` returns the maximum of `x`. When `x` is a vector, `y` is the largest element of `x`. When `x` is a matrix, `y` is a row vector containing the maximum of each column of `x`. When `x` is an n-dimensional array, `y` is the maximum along the first nonsingleton dimension of y.

`y = max(x,[],dim)` returns the maximum of `x` along the dimension `dim`.

`[y,i] = max(x)` and `[y,i] = max(x,[],dim)` also return `i`, the indices in `x` of the maximum elements. In the case of duplicate elements, `i` refers to the first occurrence.

`z = max(x,y)` compares each element of `x` with the corresponding element in `y` and returns the larger of the two. `x` and `y` must be of equal size, or either one can be a scalar.

When `x` is complex, `max` uses the magnitude and ignores the angle.

`NaN` values are considered smaller than any other value.

Examples

```
x = [0 2 3;-3 1 3;2 4 0];
x2 = [-3 4 1;3 2 0;-1 8 1];
y(:,:,1)=x;y(:,:,2)=x2;
max(x,[],1) returns [2, 4, 3].
max(x,[],2) returns [3; 3; 4].
max(y,[],3) returns [0, 4, 3 ; 3, 2, 3 ; 2, 8, 1].
```

See also

`min`, `mean`, `median`

Purpose

Compute the mean value of an array.

Synopsis

```
y = mean(x)
y = mean(x,dim)
```

Description

`y = mean(x)` returns the mean value of `x`. When `x` is a vector, `y` is the mean value of `x`. When `x` is a matrix, `y` is a row vector containing the mean value of each column of `x`. When `x` is an n-dimensional array, `y` is the mean along the first nonsingleton dimension of `x`.

`y = mean(x,dim)` returns the mean value of `x` along the dimension `dim`.

Examples

```
x = [8 3 3;-4 2 3;2 4 0];
x2 = [-3 4 2;3 5 1;-1 9 7];
y(:,:,1)=x;y(:,:,2)=x2;
```

`mean(x,1)` returns `[2, 3, 2]`.

`mean(x2,2)` returns [1; 3; 5].

`mean(y,3)` returns `[2.5, 3.5, 2.5; -0.5, 3.5, 2; 0.5, 6.5, 3.5]`.

See also

`median`, `max`, `min`

Purpose

Compute the median value of an array.

Synopsis

```
y = median(x)
y = median(x,dim)
```

Description

`y = median(x)` returns the median value of `x`. When `x` is a vector, `y` is the median value of `x`. When `x` is a matrix, `y` is a row vector containing the median value of each column of `x`. When `x` is an n-dimensional array, `y` is the median value along the first nonsingleton dimension of x.

`y = median(x,dim)` returns the median value of `x` along the dimension `dim`.

Examples

```
x = [0 2 3;-3 1 3;2 4 0];
x2 = [-3 4 1;3 2 0;-1 8 1];
y(:,:,1)=x;y(:,:,2)=x2;
```

`median(x,1)` returns `[0, 2, 3]`.
`median(x2,2)` returns [1; 2; 1].
`median(y,3)` returns `[-1.5, 3, 2 ; 0, 1.5, 1.5 ; 0.5, 6, 0.5]`.

See also

`mean`, `max`, `min`

Purpose

Create a menu.

Synopsis

```
m = menu
```

Description

`m = menu(text)` creates a menu that displays the specified text.

The function returns a menu object that can then be further manipulated using the methods in the following table:

TABLE 1-31: METHODS FOR MANIPULATING A MENU OBJECT.

METHOD	DESCRIPTION
`add(menuitem)`	Adds a menu item to this menu.
`addSeparator`	Adds a separator to this menu.

See also the reference entry for `component` for property-value pairs and methods that are valid for all components.

See also

`component`, `menuitem`

Purpose

Create a menu item.

Synopsis

```
m = menuitem(text,action)
m = menuitem(text,action,thread)
```

Description

`m = menuitem(text,action)` creates a menu item that displays the specified text. When the menu item is selected the function with the name `action` is called.

`m = menuitem(text,action,thread)` creates a menu item that displays the specified text. When the menu item is selected the function with the name `action` is called. `thread` is `true` or `false` to indicate if the action should be run in a separate thread.

See also the reference entry for `component` for property-value pairs and methods that are valid for all components.

See also

`component`, `menu`

Purpose

Create a colored wireframe surface of quadrilaterals.

Syntax

```
mesh(x,y,z,c)
mesh(x,y,z)
mesh(z,c)
mesh(z)
```

Description

`mesh(x,y,z,c)` creates a colored wireframe surface of quadrilaterals from the given matrices. The surface is created by placing grid points in $x(i,j)$, $y(i,j)$, and $z(i,j)$ for each element in the matrices. Neighboring coordinates in the matrices are then connected to form quadrilaterals. The matrix `c` is used to color each of the grid points by mapping the range of `c` to the current colormap.

`x` and `y` can also be vectors. In that case, `length(x)` must equal the number of columns in `z`, and `length(y)` must equal the number of rows in `z`. The grid points are then created as $x(j)$, $y(i)$, and $z(i,j)$.

`mesh(x,y,z)` does the same as `mesh(x,y,z,c)` but uses `z` as `c`.

`mesh(z,c)` is the same as `mesh(x,y,z,c)` where `x = 1:nx, y = 1:ny, [ny,nx] = size(z)`.

`mesh(z)` does the same as `mesh(z,c)` but uses `z` as `c`.

`h = mesh(...)` returns a handle to the plotted surface object.

In addition to the fixed arguments, additional property-value pairs can be given at the end of the command to further control how the surface is created. See the reference entry for `patch` for details of allowed properties and corresponding values.

The function `mesh` is the same as the function `surf` with the addition that it sets the property `'facecolor'` to `'w'`.

See also

`meshz`, `surf`

Purpose

Create a 2D or 3D grid.

Synopsis

```
[x, y] = meshgrid(xrange, yrange)
[x, y, z] = meshgrid(xrange, yrange, zrange)
```

Description

`[x, y] = meshgrid(xrange, yrange)` creates a 2D grid from vectors `xrange` and `yrange`. The outputs `x` and `y` are matrices of size `length(yrange)` x `length(xrange)` that can be used, for example, when plotting a function *f*(*x, y*).

`[x, y, z] = meshgrid(xrange, yrange, zrange)` creates a 3D grid from vectors `xrange`, `yrange`, and `zrange`. The outputs `x`, `y`, and `z` are matrices of size `length(yrange)` x `length(xrange)` x `length(zrange)`.

See also

`ndgrid`

Purpose

Create a colored wireframe surface of quadrilaterals with a curtain around it.

Syntax

```
meshz(x,y,z,c)
meshz(x,y,z)
meshz(z,c)
meshz(z)
```

Description

`meshz` is the same as the function `mesh` except it also adds a curtain around the plot, which consists of a series of lines extending from the surface to the lowest `z` value anywhere in the plot.

`meshz(x,y,z,c)` creates a colored wireframe surface of quadrilaterals from the given matrices. The surface is created by placing grid points at $x(i,j)$, $y(i,j)$ and $z(i,j)$ for each element in the matrices. Neighboring coordinates in the matrices are then connected to form quadrilaterals. The matrix `c` is used to color each of the grid points by mapping the range of `c` to the current colormap.

`x` and `y` can also be vectors. In that case, `length(x)` must equal the number of columns in `z`, and `length(y)` must equal the number of rows in `z`. The grid points are then created at $x(j)$, $y(i)$, and $z(i,j)$.

`meshz(x,y,z)` does the same as `mesh(x,y,z,c)` but uses `z` as `c`.

`meshz(z,c)` is the same as `mesh(x,y,z,c)` where `x = 1:nx, y = 1:ny, [ny,nx] = size(z)`.

`meshz(z)` does the same as `meshz(z,c)` but uses `z` as `c`.

`h = meshz(...)` returns a handle to the plotted surface object.

In addition to the fixed arguments, additional property-value pairs can be given at the end of the command to further control how the surface is created. See the reference entry for `patch` for details of allowed properties and corresponding values.

See also

`mesh`, `surf`

Purpose

Get the methods provided by a user-defined class.

Synopsis

```
methods cls
c = methods(cls)
c = methods(cls, attr)
c = methods(cls, attr, noattr)
```

Description

`methods cls` displays the public nonstatic methods in the user-defined class `cls`.

`c = methods(cls)` returns a cell array containing the public nonstatic methods in the class `cls`.

`c = methods(cls, attr)`, where `attr` is a string or cell array of strings, returns a cell array containing the methods of `cls` that have at least one of the attributes listed in `attr`. Possible attributes are `'public'`, `'protected'`, `'private'`, `'static'`, and `'transient'`.

`c = methods(cls, attr, noattr)` is like `c = methods(cls, attr)` but excludes any field having an attribute listed in `noattr`, which must be a string or cell array of strings.

See also

`fieldnames`

Purpose Get the name of the function or script being executed.

Synopsis

```
s = mfilename
```

Description `s = mfilename` returns the name of the function or script being executed. When you run it from the command prompt, it returns the empty string.

Purpose

Compute the minimum value of an array.

Synopsis

```
y = min(x)
y = min(x,[],dim)
[y,i] = min(x,...)
z = min(x,y)
```

Description

`y = min(x)` returns the minimum of `x`. When `x` is a vector, `y` is the smallest element of `x`. When `x` is a matrix, `y` is a row vector containing the minimum of each column of `x`. When `x` is an n-dimensional array, `y` is the minimum along the first nonsingleton dimension of y.

`y = min(x,[],dim)` returns the minimum of `x` along the dimension `dim`.

`[y,i] = min(x)` and `[y,i] = min(x,[],dim)` return `i`, the indices in `x` of the minimum elements. In the case of duplicate elements, `i` refers to the first occurrence.

`z = min(x,y)` compares each element of `x` with corresponding element in `y` and returns the smaller of the two. `x` and `y` must be of equal size, or either one can be a scalar.

When `x` is complex, `min` uses the magnitude and ignores the angle.

`NaN` values are considered larger than any other value.

Examples

```
x = [0 2 3;-3 1 3;2 4 0];
x2 = [-3 4 1;3 2 0;-1 8 1];
y(:,:,1)=x;y(:,:,2)=x2;
min(x,[],1) returns [-3,1,0].
min(x2,[],2) returns [-3; 0; -1].
min(y,[],3) returns [-3, 2, 1; -3, 1, 0; -1, 4, 0].
```

See also

`max`, `mean`, `median`

Purpose Subtract matrices pointwise.

Synopsis

```
d = minus(a, b)
```

Description `d = minus(a, b)` computes the pointwise difference between the two matrices `a` and `b`. For each dimension, `a` and `b` must have the same size or either of them must have size 1. In the latter case, the unit dimension is expanded to the size of the nonunit dimension.

`minus(a, b)` is equivalent to `a - b`.

Examples

```
[10 20 30]-[2 3 4]
```

```
[5 10]-3
```

See also `plus`, `ldivide`, `rdivide`, `times`

Purpose

Test if a function is locked in memory.

Synopsis

```
a = mislocked
a = mislocked(func)
```

Description

`a = mislocked`, when called from a function, returns `true` if the function has been locked with `mlock`, otherwise `false`.

`a = mislocked(func)` returns `true` if the function called `func` has been locked, otherwise `false`.

See also

`mlock`, `munlock`

Purpose Create a directory.

Synopsis

```
status = mkdir(name)
status = mkdir(base, name)
```

Description `status = mkdir(name)` creates a directory called `name` in the current directory. 1 is returned if the operation was successful, 0 if it failed.

`status = mkdir(base, name)` creates a directory called `name` in the directory `base`.

See also `isdir`, `rmdir`

Purpose

Make piecewise polynomial.

Synopsis

```
pp = mkpp(breaks, coefficients)
pp = mkpp(breaks, coefficients, dim)
```

Description

`pp = mkpp(breaks, coefficients)` returns a structure representing the piecewise polynomial described by its `breaks` and `coefficients`. `breaks` must be a vector of strictly increasing elements, representing the start and end of each interval. `coefficients` must be a matrix where each row contains the coefficients (in order from highest to lowest exponent) of the polynomial for one interval.

`pp = mkpp(breaks, coefficients, dim)` returns a structure representing the piecewise polynomial where each coefficient is of an array of dimension `dim`.

Example

This example creates a `pp` structure with three polynomial pieces $x^2 + 2x + 1$, $x^2 + 4x + 4$ and $x^2 + 6x + 9$ on the intervals `[1,2]`, `[2,3]` and `[3,4]`, respectively.

```
b = [1 2 3 4];
c = [1 2 1;1 4 4;1 6 9];
pp = mkpp(b,c)
```

See also

`ppval`, `unmkpp`, `pchip`, `spline`

Purpose Solve a linear system of equations.

Synopsis

```
x = mldivide(A, b)
```

Description `x = mldivide(A, b)` returns the solution to the linear system of equations `Ax=b`. Both `A` and `b` must be matrices with the same number of rows. If `A` has more rows than columns, then `x` is the least-squares solution to an overdetermined system. in this case, `x` is the solution to `A'Ax = A'b`.

`mldivide(A, b)` is equivalent to `A \ b`.

See also `ldivide`, `mrdivide`

Purpose Lock a function in memory.

Synopsis

```
mlock(func)
```

Description `mlock(func)` locks the function called `func` so that it is not removed from memory when the command `clear functions` is called.

See also `mislocked`, `munlock`

Purpose Compute the modulus of matrices.

Synopsis

```
m = mod(a, b)
```

Description `m = mod(a, b)` computes `a` mod `b` pointwise. The sizes of `a` and `b` must be identical unless one of them is a scalar; in that case, the scalar is expanded to a matrix of the correct size.

See also `rdivide`, `rem`

Purpose

Create a movie.

Synopsis

```
m = movie(...)
```

Description

`m = movie(...)` creates a movie-generation object. Frames can then be added to the movie from plots in figure windows. The properties `'width'` and `'height'` can be used with movie to specify a desired width and height for the movie.

You can then interact with the movie-generation object using the following methods

METHOD	VALUES	DESCRIPTION
`m.addFrame`		Adds the plot in the current figure window as a frame in the movie.
`m.addFrame(h)`		Adds the plot in the figure window with handle `h` as a frame in the movie.
`m.setFrameRate(rate)`		Sets the frame rate to use when generating the movie.
`m.setQuality(qual)`	A real number between 0 and 1, where 1 is the best quality.	Sets the quality to use when generating the movie.
`m.setFileType(type)`	`'avi'`, `'quicktime'`	Sets which `type` of movie to generate.
`m.listEncodings`		Displays a list of available encoding formats.
`m.setEncoding(enc)`	Any string listed by `listEncodings`.	Sets which encoding format to use.
`m.generate(filename)`		Generates a movie with the name `filename` from the frames that have been added to the movie.

Purpose Matrix power.

Synopsis

```
d = mpower(a, b)
```

Description `d = mpower(a, b)` raises `a` to the power `b`, where `a` must be a square matrix, and `b` must be a positive integer.

`mpower(a,b)` is equivalent to `a ^ b`.

Example

```
a = [-3, 2, 1; -3, 1, 0; -1, 4, 0]
mpower(a,2) returns [2, 0, -3; 6, -5, -3; -9, 2, -1].
```

See also `mtimes`, `power`

Purpose

Solve a linear system of equations.

Synopsis

```
x = mrdivide(A, b)
```

Description

`x = mrdivide(A, b)` returns the solution to the linear system of equations `b'x'=A'`. Both `A` and `b` must be matrices with the same number of columns. If `b` has more columns than rows, then `x` is the least-squares solution to an overdetermined system. In this case, `x` is the solution to `bb'x' = bA'`.

`mrdivide(A, b)` is equivalent to `A / b`.

See also

`mldivide`, `rdivide`

Purpose Compute a matrix product.

Synopsis

```
p = mtimes(a, b)
```

Description `p = mtimes(a, b)` returns the matrix product of `a` and `b`, both of which must be numerical matrices with compatible dimensions; if `a` is an `m1` x `n1`-matrix and b is an `m2` x `n2`-matrix, then `n1==m2` must hold.

`mtimes(a, b)` is equivalent to `a * b`.

See also `times`

Purpose Remove a function lock.

Synopsis `munlock(func)`

Description `munlock(func)` removes the lock on the function called `func` that was set with `mlock`.

See also `mislocked`, `mlock`

Purpose Get the maximum length of variable or function name.

Synopsis

```
len = namelengthmax
```

Description `len = namelengthmax` returns the maximum length of a variable or function name.

Purpose

Get a not-a-number value.

Synopsis

```
nan
m = nan(n)
m = nan(sz)
m = nan(n1,n2,...)
```

Description

`nan` returns a not-a-number value. This value is returned for mathematical operations where the result is ambiguous, for instance, 0/0.

`m = nan(n)`, where `n` is an integer, returns an `n`x`n` all-nan matrix.

`m = nan(sz)`, where `sz` is a vector of integers, returns an all-nan matrix of size `sz`.

`m = nan(n1,n2,...)`, where `n`*i* are integers, returns an `n1`x`n2`x `...` all-nan matrix.

See also

`inf`

Purpose Check that the number of arguments supplied to function is in a specified range.

Synopsis

```
msg = nargchk(lower, upper, actual)
```

Description `msg = nargchk(lower, upper, actual)` returns an error message if actual falls outside the range `[lower, upper]`, otherwise it returns `''`.

Example The intended use of this function is to validate the number of input arguments to a function, for example, by placing a `nargchk(2, 5, nargin)` call at the top of a function expecting between two and five input arguments.

See also `nargoutchk`

Purpose

Get the number of arguments supplied to a function.

Synopsis

```
n = nargin
n = nargin(funcname)
```

Description

`n = nargin`, when invoked from inside a function, returns the number of arguments with which the function was invoked.

`n = nargin(funcname)` returns the number of arguments declared in the definition of the function called `funcname`.

See also

`nargout`

Purpose

Get the number of outputs expected from a function.

Synopsis

```
n = nargout
n = nargout(funcname)
```

Description

`n = nargout`, when invoked from inside a function, returns the number of output arguments that the caller expects the function to return.

`n = nargout(funcname)` returns the number of outputs declared in the definition of the function called `funcname`.

See also

`nargin`

Purpose

Check that the number of outputs expected from a function is in a specified range.

Synopsis

```
msg = nargoutchk(lower, upper, actual)
```

Description

`msg = nargoutchk(lower, upper, actual)` returns an error message if `actual` falls outside the range `[lower, upper]`, otherwise it returns `''`.

Example

The intended use of this function is to validate the number of expected outputs from a function, for example, by placing a `nargoutchk(1, 4, nargout)` call at the top of a function expecting between one and four outputs.

See also

`nargchk`

Purpose

Create an n-dimensional grid.

Synopsis

```
[x, y, z, ...] = ndgrid(xrange, yrange, zrange, ...)
```

Description

`[x, y, z, ...] = ndgrid(xrange, yrange, zrange, ...)` creates an n-dimensional grid from vectors `xrange`, `yrange`, `zrange`, ….

The outputs `x`, `y`, and `z` are matrices of size `length(xrange)`-by-`length(yrange)`-by-`length(zrange)`-by-… .

See also

`meshgrid`

Purpose Get the number of dimensions of a value.

Synopsis

```
n = ndims(a)
```

Description

`n = ndims(a)` returns the number of dimensions in `a`.

`ndims(a)` is equivalent to `length(size(a))`.

See also `length`, `size`

Purpose

Compare matrices pointwise.

Synopsis

```
d = ne(a, b)
```

Description

`d = ne(a, b)` tests if the elements of the two matrices `a` and `b` are unequal pointwise. For each dimension, `a` and `b` must have the same size or either of them must have size 1. In the latter case, the unit dimension is expanded to the size of the nonunit dimension.

`ne(a, b)` is equivalent to `a~=b`.

Examples

```
[2 3 5] ~= [0 3 6]

[10 20 30] ~= 30

[0 1] ~= [0 ; 1]
```

See also

`eq`, `ge`, `gt`, `le`, `lt`

Purpose

Return a fresh axes for plotting.

Synopsis

```
h = newplot
```

Description

`h = newplot` returns a fresh axes into which to plot. If a current axes does not exist, it creates a new one. If a current axes exists, all plots that are not under control of a `hold` command are cleared before a handle to the current axes is returned.

See also

`gca`

Purpose Determine the number of nonzero elements in a matrix.

Synopsis

```
n = nnz(a)
```

Description `n = nnz(a)` returns the number of nonzero elements in `a`.

Purpose

Norm of a matrix or a vector.

Synopsis

```
norm(V)
norm(V,n)
norm(A)
norm(A,n)
```

Description

`norm(V),` for a vector, computes its Euclidian norm.

`norm(V,p)` computes the `p`-norm of the vector.

`norm(V,inf)` and `norm(V,-inf)` compute the vector's maximum and minimum, respectively.

`norm(A)` and `norm(A,2)`, for a matrix, compute its largest singular value.

`norm(A,1)` computes the 1-norm of the matrix.

`norm(A,'fro')` computes the Frobenius norm of the matrix.

`norm(A,inf)` computes the infinity norm.

See also

`cond`

Purpose Compute the logical negation of a matrix.

Synopsis

```
d = not(a)
```

Description `d = not(a)` computes the logical negation of the matrix `a`.

`not(a)` is equivalent to `~a`.

See also `and`, `or`, `xor`

Purpose

Orthonormal basis of the null space of a matrix.

Synopsis

```
null(A)
null(A,tol)
```

Description

`null(A)` computes an orthonormal basis for the null space of `A`.

`null(A,tol)` uses the relative tolerance `tol`.

Purpose

Create a cell array from numerical matrix.

Synopsis

```
c = num2cell(a)
c = num2cell(a, dims)
```

Description

`c = num2cell(a)` returns a cell array with the same size as `a` where each cell contains an element of `a`.

`c = num2cell(a, dims)` returns a cell array `c` where `size(c, i)` is 1 if `i` is listed in the vector `dims`, otherwise `size(c, i)` is `size(a, i)`. In the former case, elements with a different i^{th} index but all other indices equal are put in the same cell.

`num2cell(a)` is equivalent to `num2cell(a, [])`.

Examples

`num2cell([2 3 ; 5 7])` is `{2 3 ; 5 7}`.

`num2cell([2 3 ; 5 7], 1)` is `{[2 5] [5 7]}`.

`num2cell([2 3 ; 5 7], 2)` is `{[2 3] ; [5 7]}`.

`num2cell([2 3 ; 5 7], [1 2])` is `{[2 3 ; 5 7]}`.

See also

`mat2cell`, `cell2mat`

Purpose Convert decimal numbers to IEEE-754 hexadecimal strings.

Synopsis

```
s = num2hex(d)
```

Description `s = num2hex(d)` converts an array of doubles to IEEE-754 hexadecimal string representations 16 characters long. s is a character matrix where each row represents one double.

Example `num2hex([1 ; 20 ; Inf])` returns a character matrix:

```
'3ff0000000000000'
'4034000000000000'
'7ff0000000000000'
```

See also `format`, `base2dec`, `bin2dec`, `hex2dec`, `hex2num`, `dec2base`, `dec2bin`, `dec2hex`

Purpose Convert a number to a string.

Synopsis

```
str = num2str(x)
str = num2str(x,precision)
str = num2str(x,format)
```

Description `str = num2str(x)` converts a 2D array `x` into a string representation with approximately 4-digit precision.

`str = num2str(x,precision)` converts `x` using the maximum precision `precision`.

`str = num2str(x, format)` converts `x` using a specific format string. The default is `'%11.4g'`. (See `sprintf` for possible formats).

Example `num2str([13 0;pi NaN])` returns a character matrix:

```
  '      13             0'
  '3.141593           NaN'
```

See also `int2str`, `mat2str`, `sprintf`

Purpose Get the number of elements in a matrix or cell array.

Synopsis

```
n = numel(m)
```

Description `n = numel(m)` returns the number of elements in `m`, that is, `prod(size(m))`.

See also `size`

Purpose Get the number of nonzero elements for which there has been allocated space in a matrix.

Synopsis

```
nz = nzmax(a)
```

Description `nz = nzmax(a)`, for a dense matrix `a`, is `numel(a)`. For a sparse matrix `a`, the returned value is the number of nonzero elements for which space has been allocated. In neither case need the returned value coincide with the actual number of nonzero elements of `a`.

See also `nnz`

Purpose Get value of an ODE option.

Synopsis

```
value = odeget(options, name)
```

Description `value = odeget(options, name)` returns the value of the property `name` in the ODE options structure `options`.

See also `daspk, odeset`

Purpose

Create an options structure for an ODE solver.

Synopsis

```
opts = odeset
opts = odeset(name, value, ...)
opts = odeset(oldopts, name, value, ...)
```

Description

`opts = odeset` creates an empty options structure.

`opts = odeset(name, value, ...)` creates an options structure where one or more property/value pairs have been set.

`opts = odeset(oldopts, name, value, ...)` adds one or more property-value pairs to an existing options structure.

TABLE 1-32: PROPERTY VALUES FOR ODESET

NAME	VALUE
`'abstol'`	Absolute tolerance, scalar or vector.
`'complex'`	If true, the solution is assumed to be complex even if the initial value is real.
`'consistent'`	Consistent initialization of DAE system. If `'bweuler'` (the default), a consistent initial value is determined using the backward Euler method, if `'off'`, the initial value supplied is assumed to be consistent
`'initialstep'`	Suggested length of first step.
`'Jacobian'`	Matrix or name of function that computes df/dy.
`'Mass'`	Matrix or name of function that computes the mass matrix M(t, y). If omitted, the unit matrix is used.
`'maxorder'`	The maximum order of the backward differentiation formula that is used; must be an integer between 1 and 5.
`'minorder'`	The minimum order of the backward differentiation formula that is used; must be 1 or 2.
`'maxstep'`	Maximum step size.
`'outputfcn'`	Callback function invoked after each step has been taken.
`'reltol'`	Relative tolerance, scalar or vector.
`'stats'`	Display statistics on exit.

See also

`daspk`, `odeget`

Purpose

Create an all-one matrix.

Synopsis

```
m = ones(n)
m = ones(sz)
m = ones(n1, n2, ...)
```

Description

`m = ones(n)`, where `n` is an integer, returns an `n` x `n` all-one matrix.

`m = ones(sz)`, where `sz` is a vector of integers, returns an all-one matrix of size `sz`.

`m = ones(n1, n2, ...)`, where `n`*i* are integers, returns an `n1` x `n2`-... all-one matrix.

See also

`eye`, `repmat`, `zeros`

Purpose

Compute the logical OR of two matrices pointwise.

Synopsis

```
d = or(a, b)
```

Description

`d = or(a, b)` computes the pointwise logical OR of the two matrices `a` and `b`. For each dimension, `a` and `b` must have the same size or either of them must have size 1. In the latter case, the unit dimension is expanded to the size of the nonunit dimension.

`or(a, b)` is equivalent to `a | b`.

Examples

```
[0 0 1 1] | [0 1 0 1]
[0 1] | 0
[0 1] | [1 ; 0]
```

See also

`and`, `not`, `xor`

Purpose

Reorder Schur factorization.

Synopsis

```
[U1,T1] = ordschur(U,T,select)
```

Description

`[U1,T1] = ordschur(U,T,select)` reorders unitary matrix `U` and Schur matrix `T` (typically returned from a call to `schur`) so that a selected cluster of eigenvalues appears in the leading diagonal blocks. `select` is a logical vector with `length(T)` elements, where `true` signifies a selected eigenvalue.

`[U1,T1] = ordschur(U,T,order)` reorders `U` and `T` so that the eigenvalues appear in descending order as specified by the integer vector `order`, where each element corresponds to one eigenvalue.

If `T` is in real Schur form with complex eigenvalues (that is, complex eigenvalues are stored in 2 x 2 block on the diagonal), then said block cannot be separated by `ordschur`. If `select` or `order` contains different values for two elements in the same block, then the block is sorted by the larger of the two.

Examples

```
A = [1 0 3 1; 2 2 1 1; 0 0 5 1; 0 0 0 10];
[U,T]=schur(A);
[US,TS] = ordschur(U,T,[0 1 0 1]);

[UO,TO] = ordschur(U,T,[1 2 3 2]);
```

See also

`schur`

Purpose Orthonormal basis of the range of a matrix.

Synopsis

```
orth(A)
orth(A,tol)
```

Description `orth(A)` computes an orthonormal basis for the range of `A`.

`orth(A,tol)` uses the relative tolerance `tol`.

Purpose

Create a panel to add GUI components to.

Synopsis

```
p = panel
```

Description

`p = panel` creates a panel into which you add GUI components.

A panel uses a layout manager called GridBagLayout from Java. With it you add components within cells on a grid. Components can span several cells or be aligned to different positions within a cell. You can also specify that a component should keep its preferred size or fill the cell in which it lies.

When you have added all the desired components to the panel, COMSOL Script automatically determines the size of the panel and each cell in the panel by asking the components for their preferred sizes. This means that there is no need to manually account for different font sizes on different platforms, and so on.

When you have created a panel, you can use the following methods to add more components:

TABLE 1-33: METHODS FOR MANIPULATING PANEL OBJECTS

METHOD	DESCRIPTION
`add(comp,row,col)`	Adds a component to the cell in the given row and column.
`add(comp,row,col,nrows,ncols)`	Adds a component to the cell in the given row and column. The component spans the specified number of rows and columns.
`add(comp,row,col,fill)`	Adds a component and specifies how it should fill the cell that it is assigned to. Fill is a string that tells the component to stretch to fill the cell in certain directions. It can have one of the values `'both'`, `'horizontal'` or `'vertical'`.
`add(comp,row,col,nrwos,ncols, fill)`	The same as `add(comp,row,col,fill)` but also gives the possibility to specify the number of rows and columns that the component should span.
`addBorder(text)`	Adds a border with the specified text around the panel.
`addHSeparator(width,row,col)`	Adds a horizontal separator with the specified width in pixels to the given row and column.

TABLE 1-33: METHODS FOR MANIPULATING PANEL OBJECTS

METHOD	DESCRIPTION
`addVSeparator(height,row,col)`	Adds a vertical separator with the specified height in pixels to the given row and column.
`get(tag)`	Returns the component with the specified tag on this panel or on subpanels to this panel.
`pack`	Packs components on the panel toward the upper left corner. Use, for example, before adding a panel to a `tabbed pane` to avoid that the components on each tab stretch to fill the tab.
`packColumn(row,col)`	Packs components in a column away from the specified row and column.
`packRow(row,col)`	Packs components in a row away from the specified row and column.
`resetWeight`	Resets the weights to their default values, which are1 in both the x and y directions.
`setAlignment(align)`	Components added after calling this method get a certain alignment within the cell. align is a string with one of the following values: `'northwest'`, `'north'`, `'northeast'`, `'west'`, `'center'`, `'east'`, `'southwest'`, `'south'`, or `'east'`.
`setFill(fill)`	Use to set the fill method that is used when adding components after the call. You can used it to avoid having to specify a fill method explicitly in the add calls when adding many components with the same fill style.
`setWeight(x,y)`	Sets the weight in the x and y directions for components that are added after this call. The relative values of the weights of the components are used to determine how extra space within the panel should be distributed if the panel is larger than needed by the preferred size of the components in the panel.

TABLE 1-33: METHODS FOR MANIPULATING PANEL OBJECTS

METHOD	DESCRIPTION
`setWeightX(x)`	Only set the weight in the x direction.
`setWeightY(y)`	Only set the weight in the y direction.

See Also `component`, `dialog`, `frame`

Purpose

Create a patch consisting of triangles or quadrilaterals.

Syntax

```
patch(x,y,c)
patch(x,y,z,c)
```

Description

`patch(x,y,c)` creates one filled triangle or quadrilateral for each column in the matrices `x` and `y`. Both `x` and `y` must have three or four rows.

`c` is a matrix specifying the patch's color, and it can be one of the following:

- one of the strings `'r'`, `'g'`, `'b'`, `'c'`, `'m'`, `'y'`, or `'k'`, specifying the color of the entire patch directly.
- a 3-element vector with values between 0 and 1 representing an RGB triplet of a color for the entire patch.
- a matrix of the same size as `x`, or a matrix with one row and the same number of columns as `x`. If `c` has one row, it specifies the color per triangle or quadrilateral and flat coloring is used. If `c` has the same size as `x`, it specifies the color at the vertices and interpolated coloring is used. The colors are created by mapping the range of `c` to the colormap used.

`patch(x,y,z,c)` is the same as `patch(x,y,c)` but creates a 3D patch by taking coordinates from `z`.

`h = patch(...)` returns a handle to the created patch object.

In addition to the fixed arguments, additional property-value pairs can be given at the end of the command to further control how the patch is created.

PROPERTY	VALUE	DEFAULT	DESCRIPTION
`clim`	2-element vector		Which data values to map to the first and last color in the colormap.
`colormap`	String of the type `'jet(256)'`, or a matrix with three columns		The colormap used to color the patch. It is either a string to be evaluated, or a matrix with one row for each color and one column for red, green, and blue values.
`edgecolor`	`none | flat | interp |` colorspec	`none`	How to color the edges between each element in the patch.

PROPERTY	VALUE	DEFAULT	DESCRIPTION
facecolor	none | flat | interp | colorspec		How to color the interior of each element in the patch.
parent	Axes handle	gca	The axes to which the patch is added.

The interpretation of facecolor and edgecolor for the different allowed values are as follows:

VALUE	DESCRIPTION
none	Either the elements is be filled, or their edges are not drawn. For example, 'facecolor','none' can be used to create a wireframe plot.
flat or interp	How to interpolate the color using the vertex colors. If the value is 'flat', the entire element gets the same color; if the value is 'interp', the color in the interior of the element is created by interpolation from the values at the vertices.
colorspec	A string or an RGB triplet specifying the color of the entire patch. If it is a string, it is one of the letters r, g, b, c, m, y, or k, meaning red, green, blue, cyan, magenta, yellow, and black, respectively.

See also line, surface

Purpose Get or set the M-file path.

Synopsis

```
path
p = path
path(str)
path(str1, str2)
```

Description `path` displays the directories on the path in the order in which they are searched.

`p = path` returns a string containing the directories on the path separated by `pathsep`.

`path(str)` sets the M-file path, where `str` must be a string containing directories separated by `pathsep`.

`path(str1, str2)` sets the M-file path to the union of the paths in `str1` and `str2`.

Example `path('C:/MyProgs', path)` prepends the directory `C:/MyProgs` to the path.

See also `filesep`, `pathsep`

Purpose Get the system path separator.

Synopsis

```
sep = pathsep
```

Description `sep = pathsep` returns the separator between directories in a path list. For Windows this is `';'`, and on all other platforms it is `':'`.

See also `filesep`

Purpose

Pause execution and wait for keypress.

Synopsis

```
pause(t)
pause
pause('off')
pause('on')
```

Description

`pause(t)` pauses execution for `t` seconds.

`pause` with no input pauses the execution and waits for the user to press any key.

`pause('off')` disables any pause calls in the code.

`pause('on')` enables the effect of pause commands again.

Purpose

Piecewise cubic Hermite interpolation.

Synopsis

```
yi = pchip(x,y,xi)
pp = pchip(x,y)
```

Description

`yi = pchip(x,y,xi)` performs piecewise cubic Hermite interpolation of `y` at points `x` and returns an array `yi` corresponding to the values of the underlaying function `y` at `xi`. `x` must be a vector and `y` must be either a vector of the same length as `x` or an array whose last dimension equals the length of `x`. In the latter case, the interpolation is performed along the last dimension of `y`.

`pp = pchip(x,y)` performs piecewise cubic Hermite interpolation of `y` at points `x` and returns the interpolant as a piecewise polynomial structure (described in `ppval`).

Example

This example interpolates points from the sine curve and shows how to reuse the piecewise polynomial.

```
x = linspace(0,2*pi,10); y = sin(x);
xi = linspace(0,2*pi,20);
yi = pchip(x,y,xi);
pp = pchip(x,y);
yip = ppval(pp,xi); %Identical to yi

xi1 = linspace(0,2*pi,100);
yip1 = ppval(pp,xi1);
```

See also

`ppval`, `spline`, `mkpp`, `unmkpp`

Purpose

Permute the order of matrix dimensions.

Synopsis

```
b = permute(a, perm)
b = ipermute(a, perm)
```

Description

`b = permute(a, perm)` returns a matrix with the same elements as `a` but where the matrix dimensions have been reordered using the permutation vector `perm`, which must be a permutation of `1:ndims(a)`.

`b = ipermute(a, perm)` returns a matrix with the same elements as `a` but where the matrix dimensions have been reordered using the inverse of the permutation vector `perm`.

Examples

`permute(a, [2 1])` is equivalent to `a'` if `a` is a 2D matrix.

`permute(ones(2, 3, 5), [2 3 1])` is equivalent to `ones(3, 5, 2)`.

If `b = permute(a, perm)`, then `a(ix(1), ix(2), ...) == b(ix(p(1)), ix(p(2)), ...)`.

Purpose Get pi.

Synopsis `pi`

Description `pi` returns the mathematical constant π.

Purpose Create a colormap with different shades of pink.

Synopsis `pink(n)`

Description `pink(n)` returns a colormap with `n` colors. It is a matrix with `n` rows and 3 columns with RGB values for the colors in the colormap. The colors are different shades of pink.

See also `colormap`, `bone`, `cool`, `gray`, `grayprint`, `jet`, `hot`, `hsv`, `wavemap`

Purpose

Pseudoinverse.

Synopsis

```
pinv(A)
pinv(A,tol)
```

Description

`pinv(A)` computes the pseudoinverse of `A`.

`pinv(A,tol)` uses the relative tolerance `tol`.

Purpose

Create line plots of type y versus x.

Syntax

```
plot(x,y)
plot(y)
plot(x,y,f)
```

Description

`plot(x,y)` creates a plot of `y` versus `x`. If `x` and `y` are vectors, one plot is created. If `x` and `y` are matrices, one plot is created for each column in the matrices.

`plot(y)` plots `y` versus `1:length(y)` if `y` is a vector or versus the row indices if `y` is a matrix.

`plot(y),` where y is complex, is the same as `plot(real(y),imag(y))`.

`plot(x,y,f)` creates a plot with colors, line styles, and markers given by the format string `f,` which has one or more characters from the following table:

TABLE 1-34: STRINGS THAT CAN BE PART OF THE FORMAT STRING F

	COLOR		MARKER		LINE STYLE
r	red	+	plus	-	solid
g	green	o	circle	:	dotted
b	blue	*	star	-.	dashdot
c	cyan	v	triangle	--	dashed
m	magenta	s	square		
y	yellow	p	pentagram		
k	black	.	dot		

`h = plot(...)` returns a handle to the plotted lines.

`plot(x1,y1,f1,x2,y2,f2,x3,y3,f3,...)` can be used to create several different plots with one command.

In addition to the fixed arguments, additional property value pairs can be given at the end of the command to further control the plot. See the reference entry for `line` for details of allowed properties and corresponding values.

Example

Plot sin(x) versus x with a dashed red curve of circular markers:

```
x=linspace(0,2*pi,50);
y=sin(x);
plot(x,y,'ro--');
```

See also

`loglog`, `semilogx`, `semilogy`, `line`, `plot3`

Purpose

Create line plots in 3D.

Syntax

```
plot3(x,y,z)
plot3(x,y,z,c)
```

Description

`plot3(x,y,z)` creates a plot by connecting the coordinates in `x`, `y` and `z` with lines. If `x,y,` and `z` are vectors, one plot is created. If `x,y,` and `z` are matrices, one plot is created for each column in the matrices.

`plot3(x,y,z,c)` creates a plot with a color given by `c`, the single-letter color specification.

`h=plot3(...)` returns a handle to the plotted lines.

In addition to the fixed arguments additional property value pairs can be given at the end to further control the plot. See the reference entry for `line` for details of allowed properties and corresponding values.

See also

`plot`, `line`, `mesh`, `surf`

Purpose Add matrices pointwise.

Synopsis

```
d = plus(a, b)
```

Description `d = plus(a, b)` computes the pointwise sum of the two matrices `a` and `b`. For each dimension, `a` and `b` must have the same size or either of them must have size 1. In the latter case, the unit dimension is expanded to the size of the nonunit dimension.

`plus(a, b)` is equivalent to `a + b`.

Examples

```
[1 2 3]+[4 5 6]
[1 2 ; 3 4]+10
[1 2 3]+[10 20 30]'
```

See also `ldivide`, `minus`, `rdivide`, `times`

Purpose

Create a point.

Syntax

```
point(x,y)
point(x,y,z)
```

Description

`point(x,y)` creates points at the coordinates given by the vectors `x` and `y`.

`point(x,y,z)` creates points in 3D.

`h = point(...)` returns a handle to the created points.

In addition to the fixed arguments, additional property-value pairs can be given at the end of the command to further control how the point is created.

TABLE 1-35: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
`color`	colorspec	`k`	A string or an RGB triplet specifying the color of the points. If it is a string it is one of the letters `r`, `g`, `b`, `c`, `m`, `y` or `k`, meaning red, green, blue, cyan, magenta, yellow, and black, respectively.
`parent`	Axes handle	`gca`	What axes to add the line to.
`size`	Positive real	6	The size of the points.

Purpose

Transform from polar to Cartesian coordinates.

Synopsis

```
[x,y] = pol2cart(theta,r)
[x,y,z] = pol2cart(theta,r,z)
```

Description

`[x,y] = pol2cart(theta,r)` transforms polar coordinates in arrays `theta` and `r` into Cartesian 2D coordinates. `theta` should be the angle in radians and `r` the radius. They must be the same size or either one can be scalar.

`[x,y,z] = pol2cart(theta,r,z)` transforms cylindrical coordinates into Cartesian 3D coordinates. `theta` should be the angle in radians, `r` the radius and `z` the height. They must be the same size or scalar.

Example

`[x,y,z]=pol2cart([0 0 pi/2 0],[0 1 1 0],[0 0 0 1])` returns the Cartesian coordinates for the points (0,0,0), (0,1,0),(pi/2,1,0) and (0,0,1) in cylindrical coordinates, that is points (0,0,0), (1,0,0), (0,1,0) and (0,0,1), respectively.

See also

`cart2pol`, `sph2cart`, `cart2sph`

Purpose

Polynomial with specific roots.

Synopsis

```
p = poly(a)
```

Description

`p = poly(a)`, where `a` is a matrix, returns a vector containing the coefficients of the characteristic polynomial $det(\lambda I - a)$. If `a` is an nxn matrix, `p` is a row vector of length $n+1$.

When `a` is a vector, `p` is a vector containing the coefficients of the polynomial whose roots are `a`.

Example

`a = [1 2 5]; p = poly(a)` returns `p = [1, -8, 17, -10]`, which represents the polynomial $x^3 - 8x^2 + 17x - 10$. Calling `roots(p)` returns the original roots, 1, 2 and 5.

See also

`polyder`, `polyfit`, `polyint`, `polyval`

Purpose

Differentiate a polynomial.

Synopsis

```
q = polyder(p)
q = polyder(a,b)
[n,d] = polyder(b,a)
```

Description

`q = polyder(p)` returns the derivative of polynomial `p`, where `p` is a vector containing the polynomial coefficients.

`q = polyder(a,b)` returns the derivative of the product of two polynomials, `a * b`.

`[n,d] = polyder(a,b)` returns the numerator `n` and the denominator `d` of the derivative of the quotient of two polynomials, `a / b`.

Examples

`Derivative of polynomial` $x^3 - 8x^2 + 17x - 10$:

```
q = polyder([1,-8,17,-10])
```

`q` is `[3, -16, 17]`, that is $3x^2 - 16x + 17$

`Derivative of the product of polynomials` $x^2 + 10x + 2$ and $x + 3$:

```
q = polyder([1,10,2],[1,3])
```

`q` is `[3, 26, 32]`, that is $(x^2 + 10x + 2) \cdot 1 + (2x + 10) \cdot (x + 3) = 3x^2 + 26x + 32$

See also

`poly`, `polyfit`, `polyint`, `polyval`

Purpose

Polynomial fit.

Synopsis

```
p = polyfit(x,y,n)
[p,s] = polyfit(x,y,n)
[p,s,m] = polyfit(x,y,n)
```

Description

`p = polyfit(x,y,n)` returns the coefficients of a least squares polynomial `p(x)` of degree `n` that fits the data `p(x(i))` to `y(i)`.

`[p,s] = polyfit(x,y,n)` also returns a structure s with the fields `R` (the Cholesky factor of the Vandermonde matrix), `df` (degrees of freedom), and `normr` (the norm of the residuals).

`[p,s,m] = polyfit(x,y,n)` uses data `z = (x - m(1)) / m(2)` instead of `x`. `m` is a vector of length two where `m(1)` is the mean value of `x` and `m(2)` is the standard deviation.

See also

`poly`, `polyder`, `polyint`, `polyval`

Purpose Integrate a polynomial.

Synopsis

```
q = polyint(p,k)
q = polyint(p)
```

Description `q = polyint(p,k)` returns the integral of polynomial `p`, where `p` is a vector containing the polynomial coefficients and `k` is a scalar constant of integration.

`q = polyint(p)` returns the integral of polynomial `p` using the default scalar constant of integration `0`.

Example Integral of polynomial $8x^3 - 3x^2 + 6x - 10$ with scalar constant 20:

```
q = polyint([8,-3,6,-10],20)
```

`q` is `[2,-1,3,-10,20]`, that is $2x^4 - x^3 + 3x^2 - 10x + 20$.

See also `poly`, `polyder`, `polyfit`, `polyval`

Purpose

Evaluate a polynomial.

Synopsis

```
y = polyval(p,x)
y = polyval(p,x,[],m)
[y,d] = polyval(p,x,s)
[y,d] = polyval(p,x,s,m)
```

Description

`y = polyval(p,x)` evaluates the polynomial `p` at the elements of an array `x`. `p` is a vector containing the polynomial coefficients.

`y = polyval(p,x,[],m)` evaluates the polynomial `p` using data `z = (x - m(1)) / m(2)` instead of `x`. `m` is a vector of length two where `m(1)` is the mean value of `x` and `m(2)` is the standard deviation, as described in `polyfit`.

`[y,d] = polyval(p,x,s)` and `[y,d] = polyval(p,x,s,m)` use the structure `s` to generate error estimates `d` of `y`. `s` must be on the form returned by `polyfit`, that is, a structure with the fields `R` (the Cholesky factor of the Vandermonde matrix), `df` (degrees of freedom), and `normr` (the norm of the residuals). If the errors in the data are independent normal with constant variance, `polyval` gives error bounds `y`±`d` which contain at least 50% of the predictions.

Example

`Evaluating polynomial` $8x^3 - 3x^2 + 6x - 10$ at `0`,`1` and `2`, that is `y = polyval([8,-3,6,-10],[0 1 2])`, returns `y = [-10,1,54]`.

See also

`poly`, `polyder`, `polyfit`, `polyint`

Purpose Compute or multiply by power of 2.

Synopsis

```
d = pow2(a)
d = pow2(a, exp)
```

Description `d = pow2(a)` is equivalent to `2.^a`.

`d = pow2(a, exp)`, where `exp` is an all-integer matrix, is a faster equivalent to `a.*2.^exp`. The sizes of `a` and `exp` must be identical unless one of them is a scalar; in that case, the scalar is expanded to a matrix of the correct size.

See also `power`

Purpose

Compute a matrix power pointwise.

Synopsis

```
d = power(a, b)
```

Description

`d = power(a, b)` raises `a` to the power `b` pointwise. For each dimension, `a` and `b` must have the same size or either of them must have size 1. In the latter case, the unit dimension is expanded to the size of the nonunit dimension.

`power(a, b)` is equivalent to `a.^b`.

Examples

```
(1:5).^3
(2:6).^(3:7)
```

See also

`times`

Purpose

Evaluate piecewise polynomial.

Synopsis

```
y = ppval(pp,x)
y = ppval(x,pp)
```

Description

`y = ppval(pp,x)` and `y = ppval(x,pp)` evaluate the piecewise polynomial `pp` for the points in the real array `x`. `pp` is a structure returned by for example `spline` or `mkpp`. It should contain the following fields:

TABLE 1-36: FIELDS OF A PP STRUCT

FIELDNAME	DESCRIPTION
form	Indicates the function form and should contain string 'pp' (piecewise polynomial).
breaks	A vector of strictly increasing elements, representing the start and end of each interval.
coefs	A matrix where each row contains the coefficients (in order from highest to lowest exponent) of the polynomial for one interval.
pieces	A scalar indicating the number of pieces.
order	The order of the polynomial.
dim	A vector indicating the size of each coefficient.

See also

`pchip`, `spline`, `mkpp`, `unmkpp`

Purpose Generate prime numbers.

Synopsis

```
p = primes(n)
```

Description `p = primes(n)` generates a row vector `p` of all primes less than or equal to `n`, which must be a real scalar.

Example `primes(20)` returns `[2, 3, 5, 7, 11, 13, 17, 19].`

See also `factor`, `isprime`

Purpose

Compute the product of array elements.

Synopsis

```
y = prod(x)
y = prod(x,dim)
```

Description

`y = prod(x)` computes the product of the elements of `x` along a specific dimension. When `x` is a vector, `y` is the product of the elements of `x`. When `x` is a matrix, `y` is a row vector containing the product of the elements of each column of `x`. When `x` is an n-dimensional array, `y` is the product of the elements along the first nonsingleton dimension of `x`.

`y = prod(x,dim)` returns the product of the elements of `x` along the dimension `dim`.

Examples

```
x = [0 2 3;-3 1 3;2 4 5];
prod(x) returns [0,8,45].
prod(x,2) returns [0;-9;40].
```

See also

`cumprod`, `sum`, `cumsum`

Purpose

Generate profiling information.

Synopsis

```
profile('on')
profile('off')
profile('clear')
profile('report', func, ...)
```

Description

`profile('on')` enables collection of profiling information for all M-file functions and scripts.

`profile('off')` disables collection of profiling information.

`profile('clear')` removes all collected profiling information.

`profile('report', func, ...)` outputs a profile report for the function `func`. The following options can be given:

TABLE I-37:

OPTION	MEANING
`'-html'`	Generate the report as hyperlinked HTML files. The reports of the functions called by func are generated automatically. (Default)
`'-raw'`	Print the report as formatted text.
`'-silent'`	Same as `'-html'` but the HTML files are only generated, not displayed.
`'-dir'` followed by a directory name	Specify the directory where generated HTML files are put.

For each line of `func`, the number of times it has been executed and the relative amount of time spent there is displayed. HTML reports also contain more detailed statistics and links to reports for called functions.

Examples

To generate an HTML report for gradient with HTML files put in /tmp:

```
profile on
gradient(rand(100));
profile report -dir /tmp gradient
```

To generated a report formatted as text:

```
profile report gradient -raw
```

Purpose

Psi function.

Synopsis

```
y = psi(x)
y = psi(k,x)
y = psi(k0:k1,x)
```

Description

`y = psi(x)` computes the psi function (also called the digamma function) for `x`, which must be real and nonnegative.

`y = psi(k,x)` computes the `k`th derivative of the psi function at `x`.

`y = psi(k0:k1,x)` computes the derivatives of order `k0` through `k1` of the psi function at `x`.

See also

`gamma`, `gammaln`

Purpose Print the working directory.

Synopsis `pwd`

Description `pwd` prints the working directory.

See also `cd`, `ls`

Purpose

QR factorization.

Synopsis

```
[q,r] = qr(a)
[q,r,p] = qr(a)
[q,r] = qr(a,0)
[q,r,p] = qr(a,0)
qr(a)
```

Description

`[Q,R] = qr(A)` computes the QR factorization of the dense MxN matrix A, so that $QR = A$. Q is a MxM square unitary matrix and R is a MxN upper triangular matrix.

`[Q,R,P] = qr(A)` computes the QR factorization of the dense matrix A such that $QR = AP$. The absolute value of the diagonal elements of R are in decreasing order.

`[Q,R] = qr(A,0)` computes a reduced size factorization: For $M>N$, only the first N columns of Q and the first N rows of R are computed.

`[Q,R,P] = qr(A,0)` computes the reduced size factorization and in addition returns, `P` such that `Q*R = A(:,P)`.

`qr(A)` returns the output from the LAPACK algorithms `DGEQRF` and `ZGEQRF`, respectively.

See also

`lu`, `chol`, `svd`

Purpose

Evaluate integral numerically using adaptive Simpson quadrature.

Synopsis

```
q = quad(f,a,b)
q = quad(f,a,b,tol)
q = quad(f,a,b,tol,trace)
q = quad(f,a,b,tol,trace,x1,x2,...)
[q,fnr] = quad(f,a,b,...)
```

Description

`q = quad(f,a,b)` approximates the integral of a function `f` from `a` to `b` using adaptive Simpson quadrature with the default tolerance 1e-6.

`q = quad(f,a,b)` approximates the integral to a relative error `tol`.

`q = quad(f,a,b,tol,trace)`, when `trace` is nonzero, displays the number of function evaluations, `a`, `b-a` and `q` during the recursion.

`q = quad(f,a,b,tol,trace,x1,x2,...)` passes any further arguments `x1`,`x2`, ... to the function `f`.

`[q,fnr] = quad(f,a,b,...)` also returns `fnr`, the number of times `quad` evaluated the function `f`.

Example

`q = quad('myfun',-1,2,1e-8)` approximates the integral of a function `myfun` between `-1` and `2` with relative error `1e-8`.

See also

`quadl`

Purpose

Evaluate integral numerically using adaptive Lobatto quadrature.

Synopsis

```
q = quadl(f,a,b)
q = quadl(f,a,b,tol)
q = quadl(f,a,b,tol,trace)
q = quadl(f,a,b,tol,trace,x1,x2,...)
[q,fnr] = quadl(f,a,b,...)
```

Description

`q = quadl(f,a,b)` approximates the integral of a function `f` from `a` to `b` using adaptive Lobatto quadrature with the default tolerance 1e-6.

`q = quadl(f,a,b)` approximates the integral to a relative error `tol`.

`q = quadl(f,a,b,tol,trace)`, when `trace` is nonzero, displays the number of function evaluations, `a`, `b-a` and `q` during the recursion.

`q = quadl(f,a,b,tol,trace,x1,x2,...)` passes any further arguments `x1`,`x2`, ... to the function f.

`[q,fnr] = quadl(f,a,b,...)` also returns `fnr`, the number of times `quad` evaluated the function `f`.

Example

`q = quadl('myfun',-1,2,1e-8)` approximates the integral of a function `myfun` between `-1` and `2` with relative error `1e-8`.

See also

`quad`

Purpose Close the command window.

Synopsis

```
quit
```

Description `quit` closes the command window.

See also `exit`

Purpose Create a radio button.

Synopsis

```
r = radiobutton(text,...)
r = radiobutton(...)
```

Description `r = radiobutton(text)` creates a radio button with the specified text.

A `radiobutton` behaves exactly as does a `togglebutton` except that it is rendered as a radio button. See the reference entry for `togglebutton` for available property values and methods.

See also `togglebutton`

Purpose

Generate random numbers uniformly distributed over [0, 1].

Synopsis

```
a = rand
a = randn

a = rand(n)
a = randn(n)

a = rand(m, n, ...)
a = randn(m, n, ...)

a = rand(sz)
a = randn(sz)

a = rand('state')
a = randn('state')

rand('state', n)
randn('state', n)

rand('state', vec)
randn('state', vec)
```

Description

`rand` generates pseudorandom numbers uniformly distributed over [0, 1] while `randn` generates pseudorandom numbers with the normal distribution. This is the only difference between the two functions; in the description below, you can replace `rand` by `randn` in all places.

`a = rand` returns a random number.

`a = rand(n)`, where `n` is a positive integer, returns an `n`-by-`n`-matrix of random numbers.

`a = rand(m, n, ...)`, where `m`, `n`, … are positive integers, returns a matrix of random numbers of size `(m, n, ...)`.

`a = rand(sz)`, where `sz` is an integer vector, returns a matrix of random numbers of size `sz`.

`a = rand('state')` returns the state vector of the pseudorandom number generator.

`rand('state', n)`, where `n` is an integer, resets the generator using the seed `n`.

`rand('state', vec)`, where `vec` is a vector of doubles, sets the state of the generator to `vec`.

Purpose Random permutation.

Synopsis `p = randperm(n)`

Description `p = randperm(n)`, where `n` is an integer, returns an `n`-long vector that contains a random permutation of `1:n`.

Example `randperm(5)` returns a random permutation of `[1,2,3,4,5]`, for instance `[4,2,3,1,5]` or `[3,2,4,1,5]`.

Purpose

Compute the rank of a matrix.

Synopsis

```
rank(A)
rank(A,tol)
```

Description

`rank(A)` computes the rank of `A`.

`rank(A,tol)` computes the rank of `A` using the relative tolerance tol.

Purpose Rational fraction approximation

Synopsis

```
[n,d] = rat(x)
[n,d] = rat(x,tol)
str = rat(...)
```

Description

`[n,d] = rat(x)` returns arrays `n` and `d` such that `n./d` approximates the elements of `x` within tolerance `1.e-6*norm(X(:),1)`.

`[n,d] = rat(x,tol)` approximates `x` within tolerance `tol`.

`str = rat(...)` returns a string representation of the continued fraction of each element of `x`.

Example `[n,d] = rat([0.3 pi sqrt(2)])` returns `n = [3,355,577]` and `d = [10,113,408]`.

See also `rats`

Purpose

String representation of rational fraction approximation

Synopsis

```
str = rats(x)
str = rats(x,len)
```

Description

`str = rats(x)` returns string representations of the simple rational fraction approximations of `x`. (Unlike `str = rat(...)`, which returns the continued fraction.) An asterisk represents elements that cannot be contained within the default string length of `13`.

`str = rats(x,len)` returns string representations of the elements of `x` within the length `len`. Note that elements are separated by a space character. Hence `rats([0.325 4.442],3)` returns a string of length `8`.

Example

`rats([0.3 pi sqrt(2)])` returns the string

```
'      3/10        355/113        577/408    '.
```

See also

`rat`

Purpose Divide matrices pointwise.

Synopsis

```
d = rdivide(a, b)
```

Description `d = rdivide(a, b)` computes the pointwise ratio between the two matrices `a` and `b`. For each dimension, `a` and `b` must have the same size or either of them must have size 1. In the latter case, the unit dimension is expanded to the size of the nonunit dimension.

`rdivide(a, b)` is equivalent to `a./b`.

Examples

```
[2 3 5]./10
```

```
(2:5)./(3:6)
```

See also `minus`, `plus`, `ldivide`, `times`

Purpose Return real part.

Synopsis

```
b = real(a)
```

Description `b = real(a)` returns the real part of the complex matrix `a`.

See also `imag`

Purpose

Get the smallest and largest values that can be represented as floating-point values.

Synopsis

```
realmin
realmax
```

Description

`realmin` returns the smallest value that can be represented as a floating-point value, 1/2^1022.

`realmax` returns the largest value that can be represented as a floating-point value, 2^1024-1.

Purpose

Compute power of real matrix.

Synopsis

```
m = realpow(a, b)
```

Description

`m = realpow(a, b)` computes a^b pointwise. The sizes of `a` and `b` must be identical unless one of them is a scalar; in that case, the scalar is expanded to a matrix of the correct size.

`realpow` can only be used when the result is real, that is, when all elements of `a.^b` are real.

See also

`power`

Purpose Refreshes the view of the path.

Synopsis

```
rehash
rehash path
```

Description

`rehash` checks for each function on the path if it has been modified since it was loaded into memory and reloads it if this is the case.

`rehash path` refreshes the view of all directories on the path and loads new and modified functions.

`rehash path` is needed if a new function has been created that shadows an existing function on the path.

See also `path`

Purpose Compute a remainder.

Synopsis

```
r = rem(a, b)
```

Description `r = rem(a, b)` computes the remainder for the pointwise division of `a` and `b`, whose sizes must be identical unless one of them is a scalar; in that case, the scalar is expanded to a matrix of the correct size.

See also `mod`, `rdivide`

Purpose

Create matrix by repeating another matrix in a pattern.

Synopsis

```
r = repmat(a, sz)
r = repmat(a, dim1, dim2, ...)
```

Description

`r = repmat(a, sz)` returns a matrix created by repeating the matrix `a` in an `sz(1)`-by-`sz(2)`-... pattern, which results in a matrix of size (`sz(1)*size(a,1)`, `sz(2)*size(a,2)`, ...).

`r = repmat(a, dim1, dim2, ...)` is equivalent to `repmat(a, [dim1 dim2 ...])`.

Examples

`repmat(pi, 5, 3)` returns a 5-by-3-matrix where all the elements are `pi`.

`repmat(eye(2), 2, 3)` returns a 4-by-6-matrix where half the elements are ones and the other half are zeros.

See also

`eye`, `ones`, `zeros`

Purpose

Reshape a matrix.

Synopsis

```
r = reshape(a, sz)
r = reshape(a, dim1, dim2, ...)
```

Description

`r = reshape(a, sz)` returns the matrix `a` reshaped into size `sz`, which must be an integer vector such that `prod(sz)==numel(a)`. The returned matrix `r` has the same column-major order contents as `a` so that `r(:)` equals `a(:)`.

`r = reshape(a, dim1, dim2, ...)` returns the matrix `a` reshaped into size `(dim1, dim2, ...)`. The dimensions must satisfy the relation `dim1*dim2*...==numel(a)`. If one of `the dim`*i* is the empty matrix `[]`, then the size of that dimension is chosen such that the number of elements does not change. This is possible only if the product of the supplied dimensions divides `numel(a)`.

Examples

`reshape(1:4, 2, 2)` returns `[1 3 ; 2 4]`.

`reshape(1:100, 4, [], 5)` returns an 4-by-5-by-5-matrix.

See also

`squeeze`

Purpose Rethrow an error message.

Synopsis

```
rethrow(s)
```

Description `rethrow(s)` throws the error message in the `message` field of the structure `s`.

The typical use of `rethrow` is in a `catch` clause, where `rethrow(lasterror)` throws the error caught by `catch`.

See also `error`, `lasterror`

Purpose

Remove a directory.

Synopsis

```
status = rmdir(name)
status = rmdir(name, 's')
```

Description

`status = rmdir(name)` removes the directory `name`, which must be empty. 1 is returned if the operation was successful, 0 if it failed.

`status = rmdir(name, 's')` removes the directory `name` and its contents recursively.

See also

`isdir`, `mkdir`

Purpose Remove a field from a structure.

Synopsis

```
news = rmfield(s, field)
```

Description `news = rmfield(s, field)` returns a copy of the structure `s` where the field called `field` has been removed.

See also `isfield`, `getfield`, `setfield`

Purpose Remove a directory from the search path.

Synopsis `rmpath(dir)`

Description `rmpath(dir)` removes the directory `dir` from the list of directories where COMSOL Script looks for M-files.

See also `addpath`, `path`

Purpose Compute polynomial roots.

Synopsis `roots(P)`

Description `roots(P)` returns the roots of the polynomial `P`.

Purpose

Rotate a matrix counterclockwise.

Synopsis

```
b = rot90(a)
b = rot90(a, n)
```

Description

`b = rot90(a)` returns `a` rotated 90 degrees counterclockwise.

`b = rot90(a, n)` returns `a` rotated 90`n` degrees counterclockwise where `n` must be an integer.

See also

`fliplr`, `flipud`

Purpose Run a script.

Synopsis `run(scrname)`

Description `run(scrname)` runs the script `scrname`.

Purpose

Save a workspace to file.

Syntax

```
save(filename)
save(filename, var1, var2, ...)

save(..., '-mat')
save(..., '-ascii')
save(..., '-ascii', '-tabs')
save(..., '-ascii', '-double')
save(..., '-ascii', '-double', '-tabs')
```

Description

`save(filename)`, where `filename` is a string, saves variables and their values to the file `filename`.

`save(filename, var1, var2, ...)` saves only the variables `var1`, `var2`, ... to file. `*` can be used as wildcard character unless `'-ascii'` is given.

`save(..., '-mat')` saves the file as a MATLAB workspace file. (The default behavior is to save it as a Comsol workspace file.)

`save(..., '-ascii')` saves text representations of the variables. This is possible only for numerical 2D matrices. The variables are written in the order they were specified. For each matrix variable, a row in the matrix corresponds to a row in the output file.

`save(...,'-ascii','-tabs')` separates the elements on each row of a matrix using tabs instead of spaces.

`save(...,'-ascii','-double')` saves the variables in full precision instead of using 8 significant digits.

`save(...,'-ascii','-double','-tabs')` saves the variables tab-separated in full precision.

Example

Saving only the variables with names beginning with `'ab'`:

```
a = 2;
ab = 3;
abc = 5;
save mydata ab*
```

See also

`load`

Purpose

Save a plot as an image

Synopsis

```
saveimage(filename,...)
```

Description

`saveimage(filename,...)` saves the plot in the current figure as an image with the name `filename`.

The following property-value pairs can be used to control the generated image:

TABLE 1-38: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
`antialias`	`on` | `off`	`on`	Antialiasing.
`autoticks`	`on` | `off`	`on`	Automatic axis tick marks.
`figure`	handle	current figure	Handle to the figure window to generate an image from.
`fontscale`	positive scalar	1	Relative font scale.
`fontscaleabs`	positive scalar	1	Absolute font scale.
`height`	positive scalar	600	The height of the image.
`hideaxis3d`	`on` | `off`	`off`	Hide 3D axes objects.
`includeall`	`on` | `off`	`on`	Include colorbars and legends.
`linescale`	positive scalar	1	Relative line scale.
`linescaleabs`	positive scalar	1	Absolute line scale.
`resolution`	positive integer	300	Image resolution (dpi).
`thingrid`	on | off	`off`	Thin grid lines.
`type`	`bmp` | `jpeg` | `png` | `tiff` | `eps`	`jpeg`	The type of image to create.
`unit`	`cm` | `inch` | `pixel`	`pixel`	Image size unit.
`width`	positive scalar	800	The width of the image.

Purpose

Schur decomposition.

Synopsis

```
T = schur(A)
T = schur(A,str)
[U,T] = schur(A,...)
```

Description

`T = schur(A)` returns the Schur form of a square matrix `A`.

`T = schur(A,str)`, where `str` can be either `'real'` or `'complex'`, returns the corresponding Schur form of `A`. The default is `'real'`, which puts the eigenvalues on the diagonal if they are real and in 2-by-2 block on the diagonal if they are complex. In the latter case, the complex eigenvalues are the eigenvalues of each block. `'complex'` gives the eigenvalues on the diagonal, independent of whether they are real or complex.

`[U,T] = schur(A,...)` also returns a unitary matrix `U` such that `A = U*T*U'` and `U'*U = I`.

See also

`hess`

Purpose Create a scroll pane.

Synopsis

```
s = scrollpane(comp,...)
```

Description `s = scrollpane(comp)` creates a scroll pane that controls the specified component.

The property values listed under the reference entry for `component` can be used to further control how the scroll pane is created. In particular, it is important to specify the `'size'` property because the scroll pane is very small by default.

See also `component`, `panel`

Purpose Create a plot with a log scale on the x-axis.

Synopsis

```
semilogx(...)
```

Description `semilogx(...)` has the same functionality as `plot(...)` except that it uses a log scale on the x-axis.

See also `plot`

Purpose Create a plot with a log scale on the *y*-axis.

Synopsis `semilogy(...)`

Description `semilogy(...)` has the same functionality as `plot(...)` except that it uses a log scale on the *y*-axis.

See also `plot`

Purpose Set the value of a property for a graphics object.

Synopsis `set(h,name,value)`

Description `set(h,name,value)` sets the value of property `name` to `value` for the graphics object to which the handle `h` refers.

See also `get`

Purpose

Set difference.

Synopsis

```
c = setdiff(a,b)
c = setdiff(a,b,'rows')
[c,ai] = setdiff(...)
```

Description

`c = setdiff(a,b)` returns the elements of `a` that are not in `b`. Both of them can be either arrays or cell arrays of strings.

`c = setdiff(a,b,'rows')`, where `a` and `b` must be 2D matrices, returns the row set difference, that is, the rows in `a` that are not in `b`. `a` and `b` must have the same number of columns.

`[c,ai] = setdiff(...)` also returns the index vector `ai`, which contains the linear indices in `a` of the elements in `c`.

In addition to the fixed arguments, additional property-value pairs can be given at the end of the command.

TABLE 1-39: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
`sort`	'on' | 'off'	`'on'`	Controls whether or not output should be sorted.

Examples

```
a = [1 2 0 1 2 3];
b = [2 4 5 7 0 8];
```

`[c,ai] = setdiff(a,b)` returns `c = [1, 3]` and `ai = [4, 6]`.

`[c1,ai1] = setdiff(a,b,'sort','off')` returns the same result unsorted.

```
a = [1 2 3; 2 3 1; 3 4 5; 5 4 3; 4 3 5;1 3 3];
b = [3 4 5; 3 4 5; 1 2 2; 4 3 5];
```

`setdiff(a,b,'rows')` returns `[1, 2, 3; 1, 3, 3; 2, 3, 1; 5, 4, 3]`.

```
a = {'green','yellow','blue','green'};
b = {'red','purple','yellow'};
```

`setdiff(a,b)` returns `{'blue', 'green'}`.

See also

`intersect`, `ismember`, `setxor`, `union`, `unique`

Purpose

Set the value of a structure field.

Synopsis

```
t = setfield(s, field, val)
```

```
t = setfield(s, index1, field, index2, val)
```

Description

`t = setfield(s, field, val)`, for a structure `s`, returns a copy of `s` where the field `field` has been assigned the value `val`. This is equivalent to `t = s; t.(field) = val`.

`t = setfield(s, index1, field, index2, val)`, for a structure `s`, is equivalent to `t = s; t(index1{:}).(field)(index2{:}) = val` where `index1` and `index2` are cell arrays containing array indices.

See also

`getfield`

Purpose

Set exclusive OR.

Synopsis

```
c = setxor(a,b)
c = setxor(a,b,'rows')
[c,ai,bi] = setxor(...)
```

Description

`c = setxor(a,b)` returns the set exclusive or of `a` and `b`, that is, the elements that are not in the intersection of `a` and `b`. `a` and `b` can be either arrays or cell arrays of strings.

`c = setxor(a,b,'rows')`, where `a` and `b` must be 2D matrices, returns the row set XOR, that is, the rows not in the intersection of `a` and `b`. `a` and `b` must have the same number of columns.

`[c,ai,bi] = setxor(...)` also returns the index vectors `ai` and `bi`, where `ai` contains the linear indices of the elements of `c` that belong to `a`, and `bi` contains the linear indices of the elements of `c` that belong to `b`.

In addition to the fixed arguments, additional property-value pairs can be given at the end of the command.

TABLE 1-40: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
`sort`	`'on'` | `'off'`	`'on'`	Controls whether or not output should be sorted.

Examples

```
a = [1 2 0 1 2 3];
b = [2 4 5 7 0 8];
[c,ai,bi] = setxor(a,b)
```

returns `c = [1, 3, 4, 5, 7, 8]`, `ai = [4, 6]`, and `bi = [2, 3, 4, 6]`.

```
a = [1 2 3; 2 3 1; 3 4 5; 5 4 3; 4 3 5;1 3 3];
b = [3 4 5; 3 4 5; 1 2 2; 4 3 5];
c1 = setxor(a,b,'rows')
```

returns `[1, 2, 2 ; 1, 2, 3 ; 1, 3, 3 ; 2, 3, 1 ; 5, 4, 3]`.

`c2 = setxor(a,b,'rows','sort','off')` returns the same result unsorted.

```
a = {'green','yellow','blue','green'};
b = {'red','purple','yellow'};
setxor(a,b) returns {'blue', 'green', 'purple', 'red'}.
```

See also

`intersect`, `ismember`, `setdiff`, `union`, `unique`

Purpose

Control shading of surface and patch objects.

Synopsis

```
shading('flat')
shading('interp')
shading('faceted')
```

Description

`shading('flat')` sets that flat shading should be used on patch and surface objects in the current axes. This means that a constant color picked from one of the corners will be used in each element.

`shading('interp')` sets that the color within elements should be interpolated from color values at the corner nodes.

`shading('faceted')` uses flat shading within the elements but also shows black lines along the edges of each element.

`shading(ax,...)` controls the axes `ax` instead of the current axes.

See also

`patch`, `surface`

Purpose Shift matrix dimensions.

Syntax

```
b = shiftdim(a, n)
[b, n] = shiftdim(a)
```

Description `b = shiftdim(a, n)`, for a positive integer `n`, returns `a` after shifting the matrix dimensions cyclically `n` steps. If `n` is negative, `b` gets the same size as `a` but with `-n` unit dimensions prepended.

`[b, n] = shiftdim(a)` returns `a` with prefix unit dimensions removed; `n` is the number of unit dimensions that were removed.

Examples `size(shiftdim(ones(2, 3, 4), 1))` is `[3 4 2]`.

`size(shiftdim(ones(2, 3, 4), -2))` is `[1 1 2 3 4]`.

`[b, n] = shiftdim(ones(1, 1, 3, 5))` results in `b = ones(3, 5)` and `n = 2`.

See also `circshift`

Purpose Round a matrix to single precision.

Syntax

```
b = single(a)
```

Description `b = single(a)` returns the result of converting `a` to single precision pointwise. The returned matrix `b` is still double precision.

See also `double`

Purpose Get the size of a matrix.

Syntax

```
sz = size(a)
[rows,cols]=size(a)
szd = size(a,dim)
```

Description

`sz = size(a)` returns the size of `a`.

`[rows,cols]=size(a)` returns the number of rows and columns in `a`.

`szd = size(a,dim)` returns the size of dimension `dim` in `a`.

See also `length`

Purpose

Sort an array.

Synopsis

```
y = sort(x)
y = sort(x,dim)
[y,ind] = sort(...)
```

Description

`y = sort(x)` sorts the elements of x in ascending order. When `x` is a vector, `sort` sorts the elements of `x`. When `x` is a matrix, `sort` sorts each column of `x`. When `x` is an n-dimensional array, `sort` sorts along the first nonsingleton dimension of x.

`y = sort(x,dim)` sorts `x` along the dimension `dim`.

`[y,ind] = sort(...)` also returns `ind`, an array of the same size as `x` containing the original index of each element in `y` along the dimension `x` is sorted.

`sort` is stable, hence the relative order of identical elements is preserved.

Note that `NaN` values are sorted as larger than any other value, including `Inf`. Complex values are sorted first by magnitude, then by angle.

Examples

```
a= [1 10 0 3 2 1 7 7 5];
[res,ri] = sort(a); returns
res = [0, 1, 1, 2, 3, 5, 7, 7, 10] and
ri = [3, 1, 6, 5, 4, 9, 7, 8, 2].
```

```
x = [0 2 3;-3 1 3;2 4 0];
x2 = [-3 4 1;3 2 0;-1 8 1];
y(:,:,1)=x;y(:,:,2)=x2;
sort(x,1) returns [-3, 1, 0; 0, 2, 3; 2, 4, 3].
sort(x2,2) returns [-3, 1, 4; 0, 2, 3; -1, 1, 8].
[z,ind] = sort(y,3) returns z and ind such that:
```

```
z(:,:,1) = [-3, 2, 1; -3, 1, 0; -1, 4, 0]
z(:,:,2) = [0, 4, 3; 3, 2, 3; 2, 8, 1]
ind(:,:,1) = [2, 1, 2; 1, 1, 2; 2, 1, 1]
ind(:,:,2) = [1, 2, 1; 2, 2, 1; 1, 2, 2].
```

See also

`sortrows`

Purpose

Sort rows.

Synopsis

```
y = sortrows(x)
y = sortrows(x,col)
[y,ind] = sortrows(x)
```

Description

`y = sortrows(x)` sorts the rows of `x` in ascending order. `x` must be a matrix or a column vector.

`y = sortrows(x,col)` sorts the rows of `x` according to the columns specified in `col`, which must be a vector of positive integers, where each entry specifies one column. `sortrows(x,[3,2])`, for example, sorts first along column three, then, for rows with equal values in column three, along column two.

`[y,ind] = sortrows(...)` also returns `ind`, a vector containing the original index of each row of `y`.

`sortrows` is stable, hence the relative order of identical elements is preserved.

Note that `NaN` values are sorted as larger than any other value, including `Inf`. Complex values are sorted first by magnitude, then by angle.

Examples

```
x = [0 1 2; 1 1 0; 0 1 0; 1 0 2; 0 2 1];
```

`sortrows(x)` returns `[0, 1, 0 ; 0, 1, 2 ; 0, 2, 1 ; 1, 0, 2 ; 1, 1, 0]`.

`sortrows(x,[2 1])` returns

```
[1, 0, 2 ; 0, 1, 2 ; 0, 1, 0 ; 1, 1, 0 ; 0, 2, 1].
```

See also

`sort`

Purpose

Plays sound.

Synopsis

```
sound(data)
sound(data, rate)
sound(data, rate, nbits)

soundsc(data)
soundsc(data, rate)
soundsc(data, rate, nbits)
```

Description

`sound(data)` interprets `data` as a pulse-code modulated signal and plays it with a sample rate of 8192Hz using 16 bits per sample. Signal values outside [-1,1] are clipped.

`sound(data, rate)` uses a sample rate of `rate.`

`sound(data, rate, nbits)` uses a sample rate of `rate` and `nbits` bits per sample (only 8 and 16 are supported).

`soundsc(data, ...)` scales and translates the signal such that the minimum and maximum amplitudes are -1 and 1 respectively when sent to the output device.

`sound` and `soundsc` is only available if the platform has support for sound.

Purpose

Create a sparse matrix.

Synopsis

```
sp = sparse(a)
sp = sparse(m, n)
sp = sparse(rows, cols, data)
sp = sparse(rows, cols, m, n)
```

Description

`sp = sparse(a)` returns a sparse matrix with the same contents as the sparse or full matrix `a`.

`sp = sparse(m, n)` returns an `m`x`n` all-zero real sparse matrix.

`sp = sparse(rows, cols, data)` returns a sparse matrix with a specified sparsity pattern: `rows` and `cols` are vectors containing row and column indices for nonzero elements, and `data` is a vector containing the values of the nonzero elements. The returned matrix has the size (max(`rows`), max(`cols`)), where `rows`, `cols`, and `data` all must have the same length or be scalars; if any of them is a scalar, then it is expanded to a constant vector.

`sp = sparse(rows, cols, data, m, n)` returns an `m`-by-`n` sparse matrix with contents interpreted in the same way as for the syntax `sparse(rows, cols, data)`.

See also

`full`

Purpose

Extract diagonals from a sparse matrix or create a sparse matrix from diagonals.

Synopsis

```
[C,d] = spdiags(S)
C = spdiags(S,d)
S = spdiags(C,d,A)
S = spdiags(C,d,m,n)
```

Description

`[C,d] = spdiags(S)`, where `S` is a 2D matrix, returns a matrix `C`, whose columns are the nonzero diagonal of `S`, and a vector `d` specifying the indices of the diagonals. `0` indicates the diagonal, `-1` the first subdiagonal, `1` the first superdiagonal and so on. C will have `min(m,n)` rows, where `[m,n] = size(S)`. If a column in C is longer than the diagonal in S it represents, elements of superdiagonals correspond to the lower part of the column and elements of subdiagonals to the upper.

`C = spdiags(S,d)` returns a matrix `C` whose columns are the `d` diagonals of S.

`S = spdiags(C,d,A)` returns a sparse copy of a matrix `A` with diagonals `d` replaced by the columns in `C`.

`S = spdiags(C,d,m,n)` returns a sparse matrix of size `m`-by-`n` with diagonals `d` replaced by the columns in `C`.

Example

```
A = reshape(1:16,4,4);
C = [-1,0;0,0;0,-12;0,-14];
S = spdiags(C,[-3,2],A);
```

See also

`diag`

Purpose Create a sparse matrix with ones on the diagonal.

Synopsis

```
e = speye(n)
e = speye(m,n)
```

Description In all cases, a sparse matrix with ones on the main diagonal and zeros elsewhere is returned. Its size is determined as follows:

`speye(n)`, where `n` is a nonnegative integer, returns a square `n`-by-`n`-matrix.

`speye(m,n)`, where `m` and `n` are nonnegative integers returns an `m`-by-`n`-matrix.

See also `eye`

Purpose

Transform from spherical to Cartesian coordinates.

Synopsis

```
[x,y,z] = sph2cart(theta,phi,r)
```

Description

`[x,y,z] = sph2cart(theta,phi,r)` transforms spherical coordinates into Cartesian coordinates. `theta` is the azimuth, `phi` is the elevation, and `r` is the radius. Both `theta` and `phi` must be in radians. All input must be the same size or a scalar.

Example

`[x,y,z]=sph2cart([0 0 pi/2 0],[0 0 0 pi/2],[0 1 1 1])` returns the Cartesian 3D coordinates for the points (0,0,0), (0,0,1), (pi/2,0,1) and (0,pi/2,1) in spherical coordinates, that is points (0,0,0), (1,0,0), (0,1,0) and (0,0,1), respectively.

See also

`cart2sph`, `cart2pol`, `pol2cart`

Purpose

Cubic spline interpolation.

Synopsis

```
yi = spline(x,y,xi)
pp = spline(x,y)
```

Description

`yi = spline(x,y,xi)` performs spline interpolation of `y` at points `x` and returns an array `yi` corresponding to the values of the underlaying function `y` at `xi`. `x` must be a vector and `y` must be either a vector of the same length as `x` or an array whose last dimension equals the length of `x`. In the latter case, the interpolation is performed along the last dimension of `y`.

`pp = spline(x,y)` performs spline interpolation of `y` at points `x` and returns the cubic spline interpolant as a piecewise polynomial structure (described in `ppval`).

Example

This example interpolates points from the sine curve and shows how to reuse the piecewise polynomial.

```
x = linspace(0,2*pi,10); y = sin(x);
xi = linspace(0,2*pi,20);
yi = spline(x,y,xi);
pp = spline(x,y);
yip = ppval(pp,xi); %Identical to yi

xi1 = linspace(0,2*pi,100);
yip1 = ppval(pp,xi1);
```

See also

`ppval`, `pchip`, `mkpp`, `unmkpp`

Purpose Sparse matrix of ones.

Synopsis

```
S = spones(A)
```

Description `spones(A)` returns a sparse matrix with the same sparsity structure as `A` but with ones in the place of nonzero elements.

Example `spones([0 10 2;0 0 0; Inf 0 0])` returns the sparse version of the matrix `[0,1,1; 0,0,0; 1,0,0]`.

See also `nnz`, `spdiags`

Purpose

Sparse random matrix with uniformly distributed numbers.

Synopsis

```
S = sprand(A)
S = sprand(m,n,density)
```

Description

`sprand(A)` returns a sparse matrix with the same sparsity structure as `A` but with random numbers uniformly distributed over $[0, 1]$ in the place of nonzero elements.

`sprand(m,n,density)` returns a sparse matrix of size `m`-by-`n` with approximately `density*m*n` random numbers uniformly distributed over $[0, 1]$.

See also

`sprandn`, `sprandsym`, `rand`

Purpose

Sparse random matrix with normally distributed numbers.

Synopsis

```
S = sprandn(A)
S = sprandn(m,n,density)
```

Description

`sprandn(A)` returns a sparse matrix with the same sparsity structure as `A` but with normally distributed random numbers in the place of nonzero elements.

`sprandn(m,n,density)` returns a sparse matrix of size `m`-by-`n` with approximately `density*m*n` normally distributed random numbers.

See also

`sprand`, `sprandsym`, `rand`

Purpose

Symmetric sparse random matrix.

Synopsis

```
S = sprandsym(A)
S = sprandsym(m,density)
```

Description

`sprandsym(A)` returns a symmetric sparse matrix whose lower triangular part has the same sparsity structure as `A` but with normally distributed random numbers in the place of nonzero elements.

`sprandsym(m,density)` returns a symmetric sparse matrix of size `m`-by-`m` with approximately `density*m*m` normally distributed random numbers.

See also

`sprand`, `sprandn`, `rand`

Purpose

Convert data to a formatted string.

Synopsis

```
s = sprintf(format,m,...)
```

Description

`s = sprintf(format,m,...)` returns a string representation of the input matrices according to the C-style format string `format`.

The format string contains conversion specifications for the input matrices. Each specification begins with the % character followed by optional flags, width and precision fields and the required conversion character as described below:

TABLE 1-41: FLAGS

CHARACTER	DESCRIPTION	EXAMPLE CODE	EXAMPLE OUTPUT
`'-'` (minus)	Result is left justified	`sprintf('x=%-6.2fm',10)`	`x=10.00 m`
`'+'` (plus)	Always print sign	`sprintf('%+d, ',[2 -2])`	`+2, -2,`
`'0'` (zero)	Pad with zeros instead of spaces	`sprintf('%05.1f',2.123)`	`002.1`

Flags can be combined, that is, you can have more than one flag at the same time.

TABLE 1-42: CONVERSION CHARACTERS

CHARACTER	DESCRIPTION
`'d'` (or `'i'`)	Integer notation
`'e'`	Exponential notation using lowercase e
`'E'`	Exponential notation using uppercase E
`'f'`	Fixed-point notation
`'g'`	Exponential or fixed-point notation.
`'G'`	Identical to `'g'`, but using uppercase E for exponential notation.
`'s'`	String

`'%g'` uses exponential notation when the exponent is larger than or equal to the precision, or if the exponent is less than -4. The default precision is 6. Note that precision means number of digits to the right of the decimal point for `'%f'` and the total number of digits for `'%g'`. `'%g'` always removes insignificant zeros.

Examples

EXAMPLE CODE	RESULT
`sprintf('A:%10.4d',12)`	`A:       0012`
`sprintf('A: %-+10.2f ',[10.045,1.02])`	`A: +10.05     A: +1.02`
`sprintf('g1: %.2g g2: %.3g ',100,100)`	`g1: 1e+002 g2: 100`
`sprintf('%%Hello \n World%%')`	`%Hello` ` World%`
`sprintf('%s: %f','X',12.141)`	`X: 12.141000`
`sprintf('f: %.3f g: %.3g ',pi,pi)`	`f: 3.142 g: 3.14`
`sprintf('f: %-10.3f g: %-10.6g (m)',100.000,100.000)`	`f: 100.000    g: 100 (m)`

`sprintf('A: %.1f B: %.3e\n',`
`[1.01 1.00001,1.1],[1e4 1e-4 1])`
	`A: 1.0 B: 1.000e+000`
	`A: 1.1 B: 1.000e+004`
	`A: 0.0 B: 1.000e+000`

See also num2str, int2str, fprintf, sscanf

Purpose

Show sparsity pattern in a sparse matrix.

Synopsis

```
spy(x)
```

Description

`spy(x)` plots a small dot in positions where entries in `x` are nonzero. This can be used to visualize the sparsity pattern of sparse matrices.

Purpose Square root.

Syntax
```
b = sqrt(a)
```

Description `b = sqrt(a)` returns the pointwise square root of `a`.

See also `sqrtm`

Purpose

Matrix square root

Synopsis

```
X = sqrtm(A)
[X,r] = sqrtm(A)
[X,alpha,CX] = sqrtm(A)
```

Description

`X = sqrtm(A)`, where `A` is a square matrix, returns a matrix `X` such that `X * X = A`.

`[X,r] = sqrtm(A)` also returns the residual `r = norm(A - X^2,'fro') / norm(A,'fro')`.

`[X,alpha,CX] = sqrtm(A)` returns `alpha`, a stability factor, and `CX`, an estimate of the matrix square root condition number of `X`.

See also

`expm`, `funm`, `sqrt`

Purpose Remove the unit dimensions.

Synopsis `m = squeeze(a)`

Description `m = squeeze(a)` returns a matrix with the same contents as `a` but where interior unit dimensions have been removed and the elements shifted accordingly.

Example `squeeze(ones(4, 1, 3, 1, 1, 5))` is `ones(4, 3, 5)`.

See also `reshape`, `shiftdim`

Purpose

Read formatted data from a string.

Synopsis

```
a = sscanf(s,format)
a = sscanf(s,format,size)
[a,count,error,nextindex] = sscanf(...)
```

Description

`a = sscanf(s,format)` converts a string `s` into a matrix a according to a specific format string `format`. Formats are defined by a `'%'` character followed by the type identifier (`d`,`e`,`f`,`g`,`s`). (For information about the identifiers, see `sprintf`.) Several different formats can be used in the format string, either one after the other or separated by string tokens or white-space characters. (These tokens must be matched exactly in the base string `s`. A format string of `'%d;'`, for example, will read integers separated by `';'`.)

`a = sscanf(s,format,size)` reads data according to size, which can be a scalar `n`, in which case `sscanf` reads `n` elements into a column vector. If `n` is `Inf`, `sscanf` reads all elements. `size` can also be a matrix `[m,n]`, in which case `a` will be an `mxn` matrix filled in column order. `n` may be `Inf`, but not so `m`.

`[a,count,error,nextindex] = sscanf(...)` also returns `count`, the number of successfully read elements, and `nextindex`, one more than the characters read in `s`. `error` is unused.

Examples

`sscanf('12 Inf -1 12 5 20 0.12 8 10 NaN','%f',[2,3])` returns `[12, -1, 5 ; Inf, 12, 20]`.

`sscanf('12.2 13.1 0.1 ','%d%f %f')` returns `[12 ; 13.1 ; 0.1]`.

`sscanf('1:13,0.1 ','%d:%d,%f')` returns `[1; 13; 0.1]`.

See also

`fscanf`, `sprintf`

Purpose

Stairstep plot

Synopsis

```
stairs(y)
stairs(x,y)
stairs(...,linespec)
h = stairs(...)
[xdata,ydata] = stairs(x,y)
```

Description

`stairs(x,y)` plots a stairstep plot of `y` versus `x`. If the inputs are matrices, one line of stairs is drawn for each column.

`stairs(y)` plots `y` versus default `x`, which is 1:length(`y`) if `y` is a vector and 1:size(`y`,1) if `y` is a matrix.

`stairs(...,linespec)` can be used to control line color and line style. See `plot` for allowed values.

`h = stairs(...)` returns a handle to the drawn lines.

`[xdata,ydata] = stairs(x,y)` does not actually plot the stairs, but instead returns the vectors `xdata` and `ydata` that defines it. (Use for example `plot(xdata,ydata)` to actually plot the stairs.)

The property values for `line` can be passed at the end of the command to further control the plot.

Examples

```
% Stairstep plot of the sine function, with red
% linecolor and markers
x = 1:0.5:10;
y = sin(x);
stairs(x,y,'ro');
```

```
% Plot two functions
x = 1:0.1:10;
y1 = sin(x); y2 = cos(x);
stairs([x(:),x(:)],[y1(:),y2(:)],'linewidth',2);
```

See also

`stem`, `plot`

Purpose

Compute standard deviation.

Synopsis

```
y = std(x)
y = std(x,0)
y = std(x,1)
y = std(x,w)
y = std(x,dim)
```

Description

`y = std(x)` and `y = std(x,0)` compute the standard deviation of `x`, normalizing `y` by `n-1`, where `n` is the sample size.

`y = std(x,1)` computes the standard deviation of `x`, normalizing `y` by `n`.

When `x` is a vector, `y` is the standard deviation of `x`. When `x` is a matrix, `y` is a row vector where each element is the standard deviation of the corresponding column of `x`. When `x` is an n-dimensional array, `y` is the standard deviation along the first nonsingleton dimension of `x`.

`y = std(x,w)` computes the standard deviation of `x` using the weight vector `w`, which `std` normalizes to sum to one. `w` must contain only nonnegative elements and must be of the same length as `x` along the dimension the standard deviation is computed.

`y = std(x,w,dim)` computes the standard deviation along the dimension `dim`.

Examples

```
x = [0 4 1;2 9 2;4 -1 0];
x2 = [-3 4 1;3 2 0;-1 8 1];
y(:,:,1)=x;y(:,:,2)=x2;
```

`std(x)` returns `[2, 5, 1]`.
`std(x,[1 1 3])` returns `[1.6, 4, 0.8]`.
`std(x,[],2)` returns `[2.0817; 4.0415 ; 2.6458]`.
`std(y,[],3)` returns
`[2.121,0,0; 0.707,4.950,1.414; 3.536,6.364,0.707]`.

See also

`corrcoef`, `cov`, `var`

Purpose

Stem plot in 2D

Synopsis

```
stem(y)
stem(x,y)
stem(...,linespec)
h = stem(...)
```

Description

`stem(x,y)` plots `y` versus `x` as stems. If the inputs are matrices, one line with stems is drawn for each column.

`stem(y)` plots `y` versus default `x`, which is `1:length(y)` if `y` is a vector and `1:size(y,1)` if `y` is a matrix.

`stem(..., linespec)` can be used to control line color and line style. See `plot` for allowed values.

`h=stem(...)` returns handles to the drawn lines.

The property values for `line` can be passed at the end of the command to further control the plot.

Examples

```
% Plot ten red, dotted stems from the sine function
x = 1:10;
y = sin(x);
stem(x,y,'r--');

% Plot two functions and modify the plot afterwards
x1 = 1:10; x2 = x1+0.3;
y1 = sin(x1); y2 = cos(x2);
h=stem([x1(:),x2(:)],[y1(:),y2(:)],'linewidth',2);
set(h(2),'marker','cycle')
```

See also

`stem3`, `plot`

Purpose

Stem plot in 3D

Synopsis

```
stem3(z)
stem3(x,y,z)
stem(...,linespec)
h = stem(...)
```

Description

`stem3(x,y,z)` plots `z` versus `x` and `y` as stems. The stems are created by placing grid points in `x(i,j)`, `y(i,j)` and `z(i,j)` for each element in the matrices. `x` and `y` can also be vectors. In that case `length(x)`, must equal the number of columns in `z`, and `length(y)` must equal the number of rows in `z`. The grid points are then created as `x(j)`, `y(i)` and `z(i,j)`.

`stem3(z)` plots `z` versus default `x` and `y`, which are `1:size(z,2)` and `1:size(z,1)`, respectively.

`stem3(...,linespec)` can be used to control line color and line style. See `plot` for allowed values.

`h = stem3(...)` returns handles to the drawn lines.

The property values for `line` can be passed at the end of the command to further control the plot.

Examples

```
% Create a stem plot of the function x*y.
x=0:10; y=0:10;
z = x'*y;
stem3(x,y,z)
```

See also

`stem`, `plot`

Purpose Store application data in a frame or a dialog box.

Synopsis

```
storedata(f,data)
```

Description `storedata(f,data)` stores the data `data` in `f`. `f` can be a frame or a dialog box. The data can be any of the data types available in COMSOL Script. It can be retrieved later on using the `getdata` function.

See also `getdata`

Purpose String to number conversion.

Synopsis `x = str2num(str)`

Description `x = str2num(str)` converts a string `str` to a numeric value or a matrix. `str` must contain a valid expression representing a number or a matrix, but you can omit the enclosing brackets.

Example `str2num('1 2;3 4')` returns `[1 2; 3 4]`

See also `num2str`, `sscanf`

Purpose Concatenate strings.

Synopsis

```
s = strcat(s1,...)
```

Description `s = strcat(s1,...)` concatenates input arguments horizontally. The input can be strings, character arrays, or cell arrays of strings.

With the exception of cell arrays, `strcat` ignores trailing blanks at the end of each string. To retain these blanks in the output, use `horzcat`.

Example `strcat({'one','two'},'abc','def')` gives `{'oneabcdef','twoabcdef'}`.

`strcat({'one';'two'},['abc';'def'],'ghi')` gives `{'oneabcghi';'twodefghi'}`

See also `strvcat`, `horzcat`

Purpose

Compare strings.

Synopsis

```
r = strcmp(s1,s2)
```

Description

`r = strcmp(s1,s2)` compares `s1` and `s2`, both of which can be strings or cell arrays of strings. If both are cell arrays, they must be of equal size.

When `s1` and `s2` are both strings, `r` is logical `true` if they are identical and `false` otherwise. When one or both are cell arrays, `strcmp` compares corresponding elements and returns an array `r` containing `true` for matching elements and `false` otherwise.

Examples

`strcmp('blue',{'blue','Blue','red'})` returns `[true false false]`.

`strcmp({'blue','Blue','Red'},{'blue','Blue','red'})` returns `[true true false]`.

See also

`strcmpi`, `strncmp`, `strread, textread`

Purpose

Compare strings ignoring case.

Synopsis

```
r = strcmpi(s1, s2)
```

Description

`r = strcmpi(s1, s2)` compares `s1` and `s2` ignoring case. `s1` and `s2` can be strings or cell arrays of strings. If both are cell arrays, they must be of equal size.

When `s1` and `s2` are both strings, `r` is logical `true` if they are identical except for case, and `false` otherwise. When one or both are cell arrays, `strcmpi` compares corresponding elements and returns an array `r` containing `true` for elements that match except for case, and `false` otherwise.

Examples

`strcmpi('blue',{'blue','Blue','bluer'})` returns `[true true false]`.

`strcmpi({'blue','Blue','green'},{'blue','Blue','red'})` returns `[true true false]`.

See also

`strcmp`, `strncmp`, `strncmpi`

Purpose Find one string within another.

Synopsis `ind = strfind(s,pattern)`

Description `ind = strfind(s,pattern)` returns the first index of each occurrence of a string pattern in `s`. `s` can be either a string, in which case `ind` is the index array indicating occurrences of `pattern` in `s`, or a cell array of strings, in which case `ind` is a cell array of index arrays.

Example `strfind('blue yellow green red','e')` returns `[4, 7, 15, 16, 20]`.

See also `findstr`, `strmatch`, `strcmp`

Purpose

Justify a character array.

Synopsis

```
r = strjust(s)
r = strjust(s,alignment)
```

Description

`r = strjust(s)` returns a right-justified copy of `s`, where `s` must be a string or a character array.

`r = strjust(s,alignment)` returns a justified copy of `s` with a specific alignment: `'right'`, `'left'`, or `'center'`.

Examples

```
strjust('   red ') returns '    red'.
strjust('   red ','left') returns 'red    '.
strjust('  red ','center') returns ' red  '.
```

Purpose

Find string matches.

Synopsis

```
r = strmatch(s,strs)
r = strmatch(s,strs,'exact')
```

Description

`r = strmatch(s,strs)` finds strings in `strs` that begin with `s` and returns the index of each match. `strs` can be a character arrray, in which case `strmatch` returns row indices of the matches, or a cell array of strings, in which case `strmatch` returns the linear index of each match.

`r = strmatch(s,strs,'exact')` returns the exact matches of `s` and the strings in `strs`. Note however that trailing blanks in `strs` are ignored.

Examples

`strmatch('abc',{'abcde','abdde','bcd','abc'})` returns `[1 ; 4].`
`strmatch('abc',{'abcde','abdde','bcd','abc'},'exact')` returns `4.`

See also

`strtok`, `strfind`, `findstr`, `strcmp`

Purpose

Compare a specific number of characters in two strings.

Synopsis

```
r = strncmp(s1,s2,n)
```

Description

`r = strncmp(s1,s2,n)` compares the first `n` characters of `s1` and `s2`. `s1` and `s2` can be strings or cell arrays of strings. If both are cell arrays, they must be of equal size.

When `s1` and `s2` are both strings, `r` is logical `true` if the first `n` characters are identical and `false` otherwise. When one or both are cell arrays, `strncmp` compares corresponding elements and returns an array `r` containing `true` for elements whose first `n` characters match and `false` otherwise.

Examples

`strncmp('blue',{'black','Black','red'},2)` returns `[true false false]`.

`strncmp({'blue','Blue','green'},{'black','Black','red'},2)` returns `[true true false]`.

See also

`strncmpi`, `strcmp`, `strcmpi`

Purpose

Compare a specific number of characters in two strings ignoring case.

Synopsis

```
r = strncmpi(s1,s2,n)
```

Description

`r = strncmpi(s1,s2,n)` compares the first `n` characters of `s1` and `s2` ignoring case. `s1` and `s2` can be strings or cell arrays of strings. If both are cell arrays, they must be of equal size.

When `s1` and `s2` are both strings, `r` is logical `true` if the first `n` characters are identical except for case and `false` otherwise. When one or both are cell arrays, `strncmpi` compares corresponding elements and returns an array `r` containing `true` for elements whose first `n` characters match except for case and `false` otherwise.

Examples

`strncmpi('blue',{'black','Black','red'},2)` returns `[true true false]`.

`strncmpi({'blue','Blue','green'},{'black','blue','red'},3)` returns `[false true false]`.

See also

`strncmp`, `strcmp`, `strcmpi`

Purpose

Read formatted text.

Syntax

```
d = strread(str)
d = textread(filename)

[d1, ...] = strread(str, format, ...)
[d1, ...] = textread(filename, format, ...)

[d1, ...] = strread(str, format, n, ...)
[d1, ...] = textread(filename, format, n, ...)
```

Description

`d = strread(str)` reads a numerical matrix from the string `str`. Each nonempty line corresponds to one line in the output. All lines of `str` must contain the same number of columns.

`d = textread(filename)` reads a numerical matrix from the file called `filename`. Each nonempty line corresponds to one line in the output. All lines of the file must contain the same number of columns.

`[d1, ...] = strread(str, format, ...)` reads data from the string `str` interpreted using the format string `format`. Options to control how the data is read can be given in optional parameter pairs using the syntax `strread(str, format, par1, val1, ...)`.

`[d1, ...] = textread(filename, format, ...)` reads data from the file called `filename` interpreted using the format string `format`. Options to control how the data is read can be given in optional parameter pairs using the syntax `textread(filename, format, par1, val1, ...)`.

`[d1, ...] = strread(str, format, n, ...)` uses the format string at most `n` times. The default is to read to the end of the string.

`[d1, ...] = textread(filename, format, n, ...)` uses the format string at most `n` times. The default is to read to the end of the file.

The syntax of the format string is a subset of the syntax accepted by the function `sscanf` in the C programming language. The number of `'%'` elements in the format string must be identical with the number of outputs.

TABLE 1-43: STRREAD/TEXTREAD FORMAT STRING

FORMAT	MATCHES
Literal string	The same literal string
`'%d'`	Integer.
`'%f'`	Floating-point number.

TABLE 1-43: STRREAD/TEXTREAD FORMAT STRING

FORMAT	MATCHES
'%q''	String quoted within “ characters.
'%s'	Sequence of characters.
'[...]'	Sequence of characters from the bracketed list.
'[^...]''	Sequence of characters not from the bracketed list
'%*...'	Matches using the above rules but does not return the matched characters.

The tokens read using the format string are separated by white space except for quoted strings read with '%q'; they are read until the end of the string.

For the numerical format strings '%d' and '%f', the output is real matrices. For the other format strings, the output is cell arrays of strings.

The following property-value pairs can be used to set options:

TABLE 1-44: STRREAD/TEXTREAD PROPERTIES AND VALUES

PROPERTY	VALUE
'headerlines'	Number of lines at beginning of files that are skipped.
'delimiter'	Delimiter character.
'commentstyle'	'matlab' ignores text after % on each row, 'shell' ignores text after # on each row, 'c'ignores text between /* and */, 'c++'ignores text after // on each row.

Examples

Suppose the file 'elements' has the following contents:

```
Hydrogen 1 1.008
Oxygen 8 16.000
...
```

Then [name nr wt] = textread('elements', '%s %f %f') reads the names into the cell array name and the numbers and weights into the matrices nr and wt respectively.

Suppose that the file 'magic' has the following contents:

```
8 1 6
3 5 7
4 9 2
```

Then m = textread('magic') reads from the file into a 3-by-3-matrix.

See also `dlmread`

Purpose Search and replace strings.

Synopsis `s = strrep(str,pattern,replacement)`

Description `s = strrep(str,pattern,replacement)` replaces all occurrences of `pattern` in `str` with `replacement`. The input can be strings or cell arrays of strings, in any mix. In the case of cell arrays, `strrep` works on corresponding elements, and `s` is a cell array of the same size as the input.

Example `strrep('abc','bc','de')` returns `'ade'`.

`strrep({'abc','bcd'},'bc',{'de','ef'})` returns `{'ade','efd'}`.

See also `strtok`, `strfind`, `findstr`, `strcmp`

Purpose

Retrieve first token.

Synopsis

```
token = strtok(s)
token = strtok(s,delimiter)
[token,remainder] = strtok(...)
```

Description

`token = strtok(s)` finds the first token using white-space characters as delimiters. (See `isspace` for the definition of white-space characters.) When `s` is a string, `token` is the first token of `s`. When `s` is a cell array of strings, `token` is a cell array of the first tokens of the elements of `s`.

`token = strtok(s, delimiter)` finds the first token using the delimiter characters in `delimiter`, which can be a string or a cell array of strings.

Note that delimiter characters are not considered tokens.

`[token, remainder] = strtok(...)` also returns the remainder of `s`. When `s` is a string, `remainder` is the remainder of s. When `s` is a cell array of string, `remainder` is a cell array with the remainders of the elements of `s`. The remainder consists of all characters after the token substring.

Examples

`strtok('yellow green')` returns `'yellow'`.

`[tok, rem] = strtok('123.12:44.19:12.3',':')` returns `tok = '123.12'` and `rem = ':44.19:12.3'`.

A subsequent call to `strtok` extracts the next token:

```
tok2 = strtok(rem,':')
```

```
To extract all tokens from a string:
str = '123.12: 44.19:12.3:Inf:-10 000';
[tok,rem] = strtok(str,':');
while ~isempty(rem)
  [tok,rem] = strtok(rem,':');
  disp(tok)
end
```

See also

`strfind`, `findstr`, `strmatch`, `strcmp`

Purpose

Remove leading and trailing white-space characters.

Synopsis

```
r = strtrim(s)
```

Description

`r = strtrim(s)` removes leading and trailing white-space characters from `s`. (See `isspace` for the definition of white-space characters.) When `s` is a string, `r` is a copy of `s` without leading and trailing white-space characters. When `s` is a cell array of strings, `r` is a copy of `s` with leading and trailing white-space characters removed from each string.

See also

`deblank`

Purpose

Create a structure array.

Syntax

```
s = struct([])
s = struct(obj)
s = struct(field, val, ...)
```

Description

`struct([])` creates an empty structure.

`s = struct(obj)` converts the object `obj` to a structure; each visible field of `obj` corresponds to a field in the returned structure.

`s = struct(field, val, ...)` creates a structure from a list of pairs of field names and values. The field names must be strings; the values can be of any data type. If any value is a cell array, then the returned struct is an array with the same size as the cell array. In this case, the sizes of all nonscalar values must be identical.

Examples

`s = struct('a', 47, 'b', 11)` creates a structure with the two fields `a` and `b`.

`s = struct('a', {2 3}, 'b', 5)` creates a 1 x 2 structure array.

`s = struct('a', {{5}})` creates a structure where the field `a` has the value `{5}`.

See also

`cell`

Purpose Convert a structure array into a cell array.

Syntax

```
c = struct2cell(s)
```

Description `c = struct2cell(s)` returns the structure `s` converted to a cell array. The returned cell array has the size `[length(fieldnames(s)) size(s)]`, that is, the field names are mapped to the first dimension of the cell array where the order of the fields is that returned by `fieldnames`.

See also `cell2struct`

Purpose

Concatenate strings vertically.

Synopsis

```
s = strvcat(s1,...)
s = strvcat(cell)
```

Description

`s = strvcat(s1,...)` concatenates strings or character arrays vertically. This is the same as `vertcat` except that empty input arguments are ignored and nonempty input is automatically padded with Zeros.

`s = strvcat(cell)` concatenates strings or character arrays contained in a cell array.

Example

`strvcat('red','green','blue','yellow')` returns the character matrix:

```
  'red   '
  'green '
  'blue  '
  'yellow'
```

See also

`vertcat`, `strcat`

Purpose Convert a multidimensional index vector into an equivalent 1D matrix index.

Synopsis

```
ix = sub2ind(sz, ix1, ...)
```

Description `ix = sub2ind(sz, ix1, ...)` returns the 1D matrix index that corresponds to the multidimensional index vector (`ix1`, ...) a matrix of size `sz`.

Example `sub2ind([3 3], 1, 2)` gives 4 because `M(4)` and `M(1, 2)` refer to the same element in a 3-by-3 matrix `M`.

See also `ind2sub`

Purpose

Creates a grid containing multiple sets of plot axes in a figure window.

Syntax

```
subplot(rows,cols,current)
```

Description

`subplot(rows,cols,current)` Creates a grid containing multiple sets of plot axes in a figure window.

Syntax

```
subplot(rows,cols,current)
```

Description

`subplot(rows,cols,current)` creates a grid of smaller plot axes in the specified number of rows and columns in the current figure. The axis number `current` is made the current axis. The axis numbering increases along the columns of the first row, then along the second row, and so on. If `rows` and `cols` is the same as the current number of rows and columns in the subplot grid, then the plots are kept and only the current axis is changed. If either the number of rows or the number of columns is changed, a new subplot grid with an empty axis is created.

`current` can also be an array of numbers. In that case a smaller axes that covers those positions in the grid will be created.

`subplot(abc)`, where `abc` is a 3-digit number, is a alternative syntax where `a` is then the same as `rows`, `b` is equivalent to `cols`, and `c` is equivalent to `current` in the above syntax.

`h = subplot(...)` also returns a handle to the current axis

Example

Create a 2-by-2 grid with some different plots.

```
x=linspace(0,10,100);
y=sin(x);
subplot(2,2,1);
plot(x,y);
subplot(2,2,2);
plot(x,x.*y);
subplot(2,2,3);
plot(x,x.*y-x);
subplot(2,2,4);
plot(x,sqrt(y+2));
```

Purpose

Principal angle between subspaces.

Synopsis

```
theta = subspace(F,G)
```

Description

`theta = subspace(F,G)` returns the largest principal angle between two subspaces spanned by the columns of matrices `F` and `G`. The cosine of a principal angle is the canonical correlation.

Purpose Compute the sum of an array

Synopsis

```
y = sum(x)
y = sum(x,dim)
```

Description `y = sum(x)` adds the values of `x`. When `x` is a vector, `y` is the sum of `x`. When `x` is a matrix, `y` is a row vector containing the sum of each column of `x`. When `x` is an n-dimensional array, `y` is the sum along the first nonsingleton dimension of `x`.

`y = sum(x,dim)` returns the sum of `x` along the dimension `dim`.

Examples

```
x = [0 2 3;-3 1 3;2 4 0];
sum(x) returns [-1,7,6].
sum(x,2) returns [5 ; 1 ; 6].
```

See also `cumsum`, `prod`, `cumprod`

Purpose Run superclass constructor.

Synopsis `super(...)`

Description `super(...)`, when run from the constructor of a user-defined class, runs the constructor of the superclass, if any.

See also `this`

Purpose

Create a colored surface of quadrilaterals.

Syntax

```
surf(x,y,z,c)
surf(x,y,z)
surf(z,c)
surf(z)
```

Description

`surf(x,y,z,c)` creates a colored surface of quadrilaterals from the given matrices. The surface is created by placing grid points at $x(i,j)$, $y(i,j)$, and $z(i,j)$ for each element in the matrices. Neighboring coordinates in the matrices are then connected to form quadrilaterals. The matrix `c` is used to color each of the grid points by mapping the range of `c` to the current colormap.

`x` and `y` can also be vectors. In that case, `length(x)` must equal the number of columns in `z`, and `length(y)` must equal the number of rows in `z`. The grid points are then created at $x(j)$, $y(i)$, and $z(i,j)$.

`surf(x,y,z)` does the same as `surf(x,y,z,c)` but uses `z` as `c`.

`surf(z,c)` is the same as `surf(x,y,z,c)` where `x = 1:nx, y = 1:ny, [ny,nx] = size(z)`.

`surf(z)` does the same as `surf(z,c)` but uses `z` as `c`.

`h = surf(...)` returns a handle to the plotted surface object.

In addition to the fixed arguments, additional property-value pairs can be given at the end of the command to further control how the surface is created. See the reference entry for `patch` to get details about allowed properties and corresponding values.

Example

Create a surface plot of the function $x \cdot y$.

```
x=0:10;
y=0:10;
[xx,yy]=meshgrid(x,y);
zz=xx.*yy;
surf(xx,yy,zz)
```

See also

`mesh`

Purpose

Create a colored surface of quadrilaterals.

Syntax

```
surface(x,y,z,c)
surface(x,y,z)
surface(z,c)
surface(z)
```

Description

`surface(x,y,z,c)` creates a colored surface of quadrilaterals from the given matrices. The surface is created by placing grid points at *x(i,j)*, *y(i,j)*, and *z(i,j)* for each element in the matrices. Neighboring coordinates in the matrices are then connected to form quadrilaterals. The matrix `c` is used to color each of the grid points by mapping the range of `c` to the current colormap.

`x` and `y` can also be vectors. In that case, `length(x)` must equal the number of columns in `z`, and `length(y)` must equal the number of rows in `z`. The grid points are then created at *x(j)*, *y(i)*, and *z(i,j)*.

`surface(x,y,z)` does the same as `surface(x,y,z,c)` but uses `z` as `c`.

`surface(z,c)` is the same as `surface(x,y,z,c)` where `x = 1:nx, y = 1:ny, [ny,nx] = size(z)`.

`surface(z)` does the same as `surface(z,c)` but uses `z` as `c`.

`h = surface(...)` returns a handle to the plotted surface object.

In addition to the fixed arguments, additional property-value pairs can be given at the end of the command to further control how the surface is created. See the reference entry for `patch` to get details about allowed properties and corresponding values.

`surface` is the same as `surf` except that it does not clear the axes before adding the surface to it.

See also

`line`, `patch`, `surf`

Purpose Singular values.

Synopsis

```
svd(A)
[U,S,V] = svd(A)
```

Description `svd(A)` computes the singular values of a matrix `A`.

`[U,S,V] = svd(A)` computes a singular value decomposition of `A`: the singular value matrix `S` and the unitary matrices `U` and `V` such that `A = U*S*V'`.

Purpose

Find identifiers in an expression.

Synopsis

```
c = symvar(expr)
```

Description

`c = symvar(expr)` parses the expression string `expr` and returns a cell array containing the identifiers it contains. An identifier is a variable name not followed by parentheses or brackets.

`symvar` ignores the following common identifiers: `eps`, `i`, `inf`, `Inf`, `nan`, `NaN`, and `pi`.

See also

`inline`

Purpose Run a system command.

Synopsis

```
status = system(cmd)
[status output] = system(cmd)
```

Description `status = system(cmd)` runs the system command `cmd` in the operating system and returns the exit code, which is 0 if the execution was successful and nonzero otherwise.

`[status output] = system(cmd)` runs the system command `cmd` and returns any output to the standard output stream in `output`.

See also `dos`, `unix`

Purpose

Create a tabbed pane.

Synopsis

```
t = tabbedpane
```

Description

`t = tabbedpane` creates a tabbed pane.

The methods in the following table can be used to add panels as tabs to a tabbed pane.

TABLE 1-45: METHODS FOR ADDING PANELS TO A TABBEDPANE.

METHOD	DESCRIPTION
`addTab(title,panel)`	Adds the given panel as a tab with the specified title.
`addTab(title,panel,pos)`	Inserts a tab with the given panel at the specified position.

See also

`component`, `panel`

Purpose

Create a table.

Synopsis

```
t = table(...)
```

Description

`t = table` creates a table.

The property value pairs in the following table can be used to control how the table is created.

TABLE 1-46: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
`autoadd`	`on` | `off`	`off`	Should rows automatically be added to the end of the table as needed when the user enters values.
`cols`	integer	2	The number of columns in the table.
`editablecols`	integer array	all	Indices to the columns that should be editable.
`rows`	integer	10	The number of rows in the table.
`titles`	cell array of strings		The headings for each column.
`width`	integer array		The desired width of the columns. Either a scalar specifying the same width for all columns or a vector of the same length as the number of columns. If not given each column will be given a suitable width to fit its title.

The function returns a table object that can then be further manipulated using the methods in the following table.

TABLE 1-47: METHODS FOR MANIPULATING A TABLE OBJECT.

METHOD	DESCRIPTION
`getValue`	Returns a matrix with all the values in the table.
`getValue(rows,cols)`	Returns a matrix with values taken from the rows and columns given by the index vectors `rows` and `cols`.
`setValue(data)`	Sets the values of a cells in the table from a numerical matrix.

See also the reference entry for `component` for property-value pairs and methods that are valid for all components.

Purpose Get a directory where temporary files can be created.

Synopsis

```
d = tempdir
```

Description `d = tempdir` returns a directory where temporary files can be created.

See also `tempname`

Purpose Create a temporary file name.

Synopsis

```
f = tempname
```

Description `f = tempname` returns a file name suitable for a temporary file. Successive calls try to return different file names, but no guarantee about this is made.

See also `tempdir`

Purpose

Add text at a specified location.

Synopsis

```
text(x,y,string)
text(x,y,z,string)
```

Description

`text(x,y,string)` adds the text `string` at the coordinates `x` and `y`. Both `x` and `y` can also be vectors, and `string` a cell array of strings of the same length.

`text(x,y,z,string)` adds text in 3D.

`h = text(...)` returns a handle to the created text.

In addition to the fixed arguments, additional property-value pairs can be given at the end of the command to further control how the text is created.

TABLE 1-48: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
`color`	colorspec	`k`	A string or an RGB triplet specifying the color of the text. If it is a string it is one of the letters `r`, `g`, `b`, `c`, `m`, `y` or `k`, meaning red, green, blue, cyan, magenta, yellow and black respectively.
`parent`	Axes handle	`gca`	The axes to which to add the text.

The following HTML tags are supported in the text command string.

TABLE 1-49: VALID HTML TAGS

HTML TAG	DESCRIPTION
`<B> </B>`	Enclosed text will be rendered using a bold font style.
` `	Line break.
`<CENTER> </CENTER>`	Centered text.
`<I> </I>`	Enclosed text will be rendered using an italic font style.
`<LI>`	List item. When the list used is `<OL>`, ordered list, the LI element will be rendered with a number. When the list used is `<UL>`, unordered list, the LI element will be rendered with a bullet.
`<OL> </OL>`	Ordered list (see also: `<LI>`).
`<P> </P>`	Paragraph. This tag will create a line break and a space between lines.
`<PRE> </PRE>`	Enclosed text preserves spaces and line breaks. The text will be rendered using a monospaced font.

TABLE 1-49: VALID HTML TAGS

HTML TAG	DESCRIPTION
<STRIKE> </STRIKE>	Enclosed text will be rendered in a strike-through appearance.
	Enclosed text will be rendered in subscript, with the enclosed text slightly lower than the surrounding text.
	Enclosed text will be rendered in superscript, with the enclosed text slightly higher than the surrounding text.
<TT> </TT>	Enclosed text will be rendered using a monospaced font.
<U> </U>	Enclosed text will be underlined.
<UL> </UL>	Unordered list (see also: <LI>).

The following Greek symbol tags are supported in the text command string.

TABLE 1-50: VALID GREEK SYMBOL TAGS

TAG	SYMBOL	TAG	SYMBOL
\ALPHA	Α	\alpha	α
\BETA	Β	\beta	β
\GAMMA	Γ	\gamma	γ
\DELTA	Δ	\delta	δ
\EPSILON	Ε	\epsilon	ε
\ZETA	Ζ	\zeta	ζ
\ETA	Η	\eta	η
\THETA	Θ	\theta	θ
\IOTA	Ι	\iota	ι
\KAPPA	Κ	\kappa	κ
\LAMBDA	Λ	\lambda	λ
\MU	Μ	\mu	μ
\NU	Ν	\nu	ν
\XI	Ξ	\xi	ξ
\OMICRON	Ο	\omicron	ο
\PI	Π	\pi	π
\RHO	Ρ	\rho	ρ
\SIGMA	Σ	\sigma	σ
\TAU	Τ	\tau	τ
\UPSILON	Υ	\upsilon	υ

TABLE 1-50: VALID GREEK SYMBOL TAGS

TAG	SYMBOL	TAG	SYMBOL
\PHI	Φ	\phi	φ
\CHI	Χ	\chi	χ
\PSI	Ψ	\psi	ψ
\OMEGA	Ω	\omega	ω

The following math symbol tags are supported in the text command string.

TABLE 1-51: VALID MATH SYMBOL TAGS

TAG	SYMBOL	TAG	SYMBOL
\approx	≈	\bullet	•
\lequal	≤	\partial	∂
\gequal	≥	\nabla	∇
\plusmin	±	\sqrt	√
\infinity	∞	\integral	∫

In addition to the greek and math symbols above, you can specify additional characters using Unicode numbers (see example below). Visit www.unicode.org for more information about Unicode characters.

Examples

Create a text object with the following text: βξ + x•y

```
text(0,0,'\beta<sup>\xi</sup> + x\bullety')
```

Create a text object by using the Unicode number `00A9` (©, the copyright character):

```
text(0,0,'\u00A9')
```

See also

`xlabel`, `ylabel`, `zlabel`, `title`

Purpose

Create a text area.

Synopsis

```
t = textarea(rows,cols,...)
```

Description

`t = textarea(rows,cols)` creates a text area to hold the specified number of rows and columns of text.

TABLE 1-52: METHODS FOR MANIPULATING A TEXTAREA OBJECT.

METHOD	DESCRIPTION
`getValue`	Returns the text in the text area.
`setValue(text)`	Sets the text in the text area.

See also the reference entry for `component` for property-value pairs and methods that are valid for all components.

See also

`component`, `textfield`

Purpose

Create a text field.

Synopsis

```
t = textfield(width, ...)
```

Description

`t = textfield(width)` creates a text field wide enough to hold `width` characters.

TABLE 1-53: METHODS FOR MANIPULATING A TEXTFIELD OBJECT.

METHOD	DESCRIPTION
`getValue`	Returns the text in the text field.
`setValue(text)`	Sets the text in the text field.

See also the reference entry for `component` for property-value pairs and methods that are valid for all components.

See also

`component`, `textarea`

Purpose Get the instance for which an instance method is run.

Synopsis

```
obj = this
```

Description `obj = this`, when called from an instance method of a user-defined class, returns the instance for which the method is run.

See also `clone`, `super`

Purpose

Start or stop timer.

Synopsis

```
tic
toc
t = toc
```

Description

`tic` starts a timer.

`toc` displays the time elapsed since the timer was started with `tic`.

`t = toc` returns the time in seconds elapsed since the timer was started with `tic`.

Purpose

Multiply matrices pointwise.

Synopsis

```
d = times(a, b)
```

Description

`d = times(a, b)` computes the pointwise product of the two matrices `a` and `b`. For each dimension, `a` and `b` must have the same size or either of them must have size 1. In the latter case, the unit dimension is expanded to the size of the nonunit dimension.

`times(a, b)` is equivalent to `a.*b`.

Examples

```
[1 2 3].*[4 5 6]
[1 2 ; 3 4].*10
[1 2 3].*[10 20 30]'
```

See also

`plus`, `rdivide`

Purpose

Interpolation on delaunay triangulation.

Synopsis

```
yi = tinterp(s,y)
```

Description

`yi = tinterp(s,y)` uses a delaunay triangulation stored in the struct `s` and interpolates linearly to determine `yi` when `y = f(x1,x2,...)`. `y` must match the size of the original points, used for the triangulation.

`s` is a struct as produced by the `griddata` functions and contains the following fields:

TABLE 1-54: FIELDS OF THE S STRUCT

FIELDNAME	DESCRIPTION
method	Interpolation method. Can be either 'linear' (denoting linear interpolation) or 'nearest' (denoting nearest neighbor interpolation). Nearest neighbor in this case signifies the closest vertex in the nearest delaunay element.
strategy	Search strategy. Not actually used by `tinterp`, but serves to indicate whether the indexation, `s.ind` was created using the `'boxonly'` or `'closest'` search strategy. `'boxonly'` means that `s.ind` contains `NaN` for points outside the mesh, whereas `'closest'` indicates that a nearest element was located for all points. (For further details, see `griddata`.)
t	2D or 3D Delaunay triangulation of original points.
ind	A column vector containing row indices into t for all search points.
coord	Barycentric coordinates for each search point.
size	Denotes the expected size of yi.

Note that if `s` was created through a call to one of the `griddata` functions, then the interpolation method was also used when creating the indexation, `s.ind`. Though `griddata` does not perform any interpolation in this case (when a struct is requested), the different methods are slightly different insofar that `'linear'` returns `NaN` for points outside the mesh whereas `'nearest'` locates the nearest element for all points.

Examples

```
rand('state',0);
x = 4*rand(1,100)-2;y = 4*rand(1,100)-2;
ti = -2:.1:2;
[xi,yi] = meshgrid(ti,ti);
g = griddata(x,y,xi,yi, 'linear',[],'closest');

z=sin(x).*sin(y).*exp(-x.^2-y.^2);
zi1 = tinterp(g,z);
z2 = sin(x).*sin(y);
```

```
zi2 = tinterp(g,z2);

plot3(x,y,z,'*');
hold on;
mesh(xi,yi,zi1);
hold off;
figure;
plot3(x,y,z2,'*');
hold on;
mesh(xi,yi,zi2);
hold off;
```

See also griddata, griddata3, griddatan, tsearch, tsearchn, delaunay, delaunay3

Purpose

Add a title above a plot.

Synopsis

```
title(string)
```

Description

`title(string)` sets the text `string` as a title above the plot in the current axes.

See the `text` command for a list of valid Greek symbols and HTML formatting syntax.

See also

`text`, `xlabel`, `ylabel`, `zlabel`

Purpose

Create a toggle button.

Synopsis

```
t = togglebutton(text,...)
t = togglebutton(...)
```

Description

`t = togglebutton(text)` creates a toggle button with the specified text.

To make the toggle button synchronize its state with other toggle buttons, you can add them all to the same `buttongroup`.

In addition to the fixed arguments, additional property-value pairs can be given at the end of the command to further control how the button is created.

TABLE 1-55: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DESCRIPTION
`image`	iconimage	An image to display on the toggle button.
`text`	string	A text to display on the toggle button.

The function returns a `togglebutton` object that can then be further manipulated using the methods in the following table.

TABLE 1-56: METHODS FOR MANIPULATING A TOGGLEBUTTON OBJECT.

METHOD	DESCRIPTION
`addActionListener(name)`	Specifies that the function with the given name should be run when the button is clicked.
`addActionListenerThread(name)`	Specifies that the function with the given name should be run when the button is clicked. The function will be run in a separate thread. This can be used for operations that run for a long time and need to update graphics while running.
`getSelected`	Returns the selected state of the toggle button as a logical.
`getText`	Returns the text on the button.
`getValue`	Returns the selected state of the toggle button as the string `'on'` or `'off'`.
`setSelected(sel)`	Sets the selected state of the toggle button as a logical.
`setText(text)`	Sets the text on the button.
`setValue(val)`	Sets the selected state of the toggle button using the string `'on'` or `'off'`.

See also the reference entry for `component` for property value pairs and methods that are valid for all components.

See also `checkbox`, `radiobutton`

Purpose

Tensor product and contraction.

Synopsis

```
C = tprod(A, B, IA, IB)
```

Description

`C = TPROD(A, B, IA, IB)` computes the tensor product of the arrays `A` and `B`, optionally followed by contractions and setting some indices equal. The mapping from input indices to output indices, as well as how to contract, is described by the vectors IA and IB.

This function is best explained by an example: Let `A` be a 4-dimensional array, and `B` a 3-dimensional array. Then

```
C = TPROD(A, B, [2 -1 1 -2], [-2 2 -1])
```

creates a 2-dimensional array (matrix) `C` in the following way. First, the product `D(%-2, %-1, %1, %2) = A(%2, %-1, %1, %-2)*B(%-2, %2, %-1)` is formed. This is a 4-dimensional array D, where %-1, %-2, %1, %2 denote index variables. `D` is the tensor (outer) product of `A` and `B`, followed by a permutation of the indices and setting some indices equal. It is assumed that `SIZE(A,1)=SIZE(B,2)`, `SIZE(A,2)=SIZE(B,3)`, and `SIZE(A,4)=SIZE(B,1)`. Secondly, we sum over the index variables corresponding to negative numbers (%-1 and %-2): `C(%1, %2) = sum of D(%-2, %-1, %1, %2)` where the indices %-1 and %-2 (independently) run through all their possible values.

The arguments are assumed to have the following format:

- `A` and `B` are real or complex arrays.
- `IA` and `IB` are vectors of doubles, containing nonzero integers.
- The length of `IA` (`IB`) has to be equal to the number of dimensions of `A` (`B`).
- `A` is padded with singleton dimensions if the number of dimensions of `A` is less than the length of `IA` (and similarly for `B`).
- The numbers in `IA` (`IB`) have to be distinct.
- If a number occurs both in `IA` and `IB`, it is required that the corresponding dimensions in `A` and `B` have the same size.
- If a negative number occurs in `A` (`B`), it must also occur in `B` (`A`).
- It is assumed that the union of the numbers in `IA` and `IB` together with 0 form a contiguous sequence of integers.

Examples

`C = TPROD(A, B, [1 2], [3 4])` is the tensor (outer) product of the matrices `A` and `B`.

`C = TPROD(A, B, [1 -1], [-1 2])` is the ordinary matrix product of the matrices `A` and `B`.

`C = TPROD(A, 1, [2 1], [3 4])` is the transpose of the matrix `A`. Note that trailing singleton dimensions are removed, so `C` is a matrix.

`C = TPROD(A, ONES(SIZE(A)), [-1 -2], [-1 -2])` is the sum of all entries in the matrix `A`.

`C = TPROD(A, EYE(SIZE(A)), [-1 -2], [-1 -2])` is the sum of all diagonal entries in the matrix `A` (the trace).

Purpose The trace of a matrix.

Synopsis `trace(A)`

Description `trace(A)` computes the sum of the diagonal elements of `A`.

Purpose Transposes a matrix.

Synopsis

```
d = transpose(a)
```

Description `d = transpose(a)` computes the transpose of the matrix `a`.

`transpose(a)` is equivalent to `a.'`.

See also `ctranspose`

Purpose

Trapezoidal numerical integration.

Synopsis

```
z = trapz(y)
z = trapz(x,y)
z = trapz(y,dim)
z = trapz(x,y,dim)
```

Description

`z = trapz(y)` computes the integral of `y` using the trapezoidal method with unit spacing. (To compute the integral for different spacing, multiply `z` by the spacing increment.) When `y` is a vector, `z` is the integral of `y`. When `y` is a matrix, `z` is a row vector containing the integral over each column of `y`. When `y` is an n-dimensional array, `z` is the integral along the first nonsingleton dimension of `y`.

`z = trapz(x,y)` computes the integral of `y` with respect to `x`, which must be a vector with the same length as the first nonsingleton dimension of `y`. Alternatively, both `x` and `y` must be vectors of equal length.

`z = trapz(y,dim)` or `z = trapz(x,y,dim)` integrate across the dimension `dim` of `y`. `x`, if given, must be a vector with the same length as `y` along the dimension `dim`.

Examples

```
y = reshape(0:11,3,4);
trapz(y) returns [2 8 14 20]
trapz(y,2) returns [13.5 ; 16.5 ; 19.5].
```

See also

`cumtrapz`

Purpose

Extract elements above or below the main diagonal of a matrix.

Synopsis

```
l = tril(a)
u = triu(a)

l = tril(a, n)
u = triu(a, n)
```

Description

`l = tril(a)` returns a matrix containing the elements on or below the main diagonal of `a`.

`u = triu(a)` returns a matrix containing the elements on or above the main diagonal of `a`.

`l = tril(a, n)` returns a matrix containing the elements on or below the n^{th} superdiagonal of `a`.

`u = triu(a, n)` returns a matrix containing the elements on or above the n^{th} superdiagonal of `a`.

See also

`diag`

Purpose

Create a mesh plot with triangles.

Synopsis

```
trimesh(tri, x, y, z, c)
trimesh(tri, x, y, z)
trimesh(tri, x, y)
h=trimesh(...)
```

Description

`trimesh(tri, x, y, z, c)` creates a mesh plot with triangles. `tri` is a N-by-3 matrix where each row corresponds to a triangle. The entries in `tri` are indices into `x`, `y`, `z` and `c`.

`trimesh(tri, x, y, z)` uses `c=z`.

`trimesh(tri, x, y)` displays the mesh in 2D using `line`.

`h = trimesh(...)` returns a handle to the created object.

Additional property values from `patch` or `line` can be given at the end of the command to further control the created object.

See also

`trisurf`

Purpose

Create a surface plot with triangles.

Synopsis

```
trisurf(tri, x, y, z, c)
trisurf(tri, x, y, z)
h=trisurf(...)
```

Description

`trisurf(tri, x, y, z, c)` creates a surface plot with triangles. `tri` is a N-by-3 matrix where each row corresponds to a triangle. The entries in `tri` are indices into `x`, `y`, `z` and `c`.

`trisurf(tri, x, y, z)` uses `c=z`.

`h = trisurf(...)` returns a handle to the created patch object.

Additional property values from `patch` can be given at the end of the command to further control the created object.

See also

`trimesh`

Purpose Create an all-true logical matrix.

Synopsis

```
f = true
f = true(n)
f = true(m, n, ...)
f = true(sz)
```

Description In all cases, an all-true logical matrix is returned. Its size is determined as follows:

`f = true` returns a scalar.

`f = true(n)`, where `n` is a nonnegative integer, returns an `n` x `n` matrix.

`f = true(m, n, ...)`, where `m`, `n`, ... are nonnegative integers, returns an `m` x `n` x ...-matrix.

`f = true(sz)`, where `sz` is an integer vector, returns a matrix of size `sz`.

See also `false`

Purpose

Find Delaunay element.

Synopsis

```
ind = tsearch(x,y,t,xi,yi)
```

Description

`ind = tsearch(x,y,t,xi,yi)` finds the Delaunay element for each point (`xi`, `yi`). `ind` is a column vector containing row indices into `t` (or `NaN` for points outside the mesh), where `t` is a triangulation of `x` and `y` as returned by `delaunay`.

To get the barycentric coordinates for `xi` and `yi`, use `tsearchn`:

```
[ind,coord] = tsearchn([x(:),y(:)],t,[xi(:),tiyi(:)])
```

Example

```
x = rand(20,1);y = rand(20,1);
tri = delaunay(x,y);
xi = rand(5,5); yi = rand(5,5);
tsearch(x,y,tri,xi,yi)
```

See also

`delaunay`, `delaunay3`, `tsearchn`, `griddata`, `griddata3`, `griddatan`

Purpose

Find Delaunay element in nD.

Synopsis

```
ind = tsearchn(pts,t,ptsi)
[ind,coord] = tsearchn(pts,t,ptsi)
```

Description

`ind = tsearchn(pts,t,ptsi)` finds the Delaunay element for each point in `ptsi`. `ind` is a column vector containing row indices into `t` (or `NaN` for points outside the mesh), where `t` is a triangulation of `pts` as returned by `delaunay` or `delaunay3`. `pts` and `ptsi` are nx2 or nx3 matrices, for 2D and 3D space respectively.

`[ind,coord] = tsearchn(pts,t,ptsi)` also returns the barycentric or area coordinates for all points in `ptsi`.

Example

```
pts = [0 0 0;0 0 1;0 1 0; 0 1 1; 1 0 0;1 0 1; 1 1 0; 1 1 1];
tri = delaunay3(pts(:,1),pts(:,2),pts(:,3));
t = 0:0.1:1;
[xi,yi,zi] = meshgrid(t,t,t);
ptsi = [xi(:),yi(:),zi(:)];
[ind,coord] = tsearchn(pts,tri,ptsi);
```

See also

`delaunay`, `delaunay3`, `tsearch`, `griddata`, `griddata3`, `griddatan`

Purpose

Display the contents in a text file on the command line.

Synopsis

```
type(filename)
```

Description

`type(filename)` displays the contents of the file `filename` on the command line. `filename` can be an absolute file name or an M-file on the path.

Purpose

Convert matrix to an unsigned integer matrix.

Synopsis

```
m = uint8(a)
m = uint16(a)
m = uint32(a)
m = uint64(a)
```

Description

`m = uint8(a)` converts the real matrix `a` to an unsigned integer matrix by rounding each element to the closest unsigned 8-bit integer. Elements too large or too small to be represented using unsigned 8-bit integers are rounded to the largest and smallest 8-bit integers, respectively.

`uint16`, `uint32`, and `uint64` instead round to 16-, 32-, and 64-bit unsigned integers, respectively.

The maximum and minimum values of *n*-bit unsigned integers are as follows:.

TABLE 1-57:

FUNCTION	MIN	MAX
uint8	0	255
uint16	0	65535
uint32	0	4294967295
uint64	0	18446744073709551615

See also

`int8, int16, int32, int64`

Purpose

Compute the unary negation of a matrix.

Synopsis

```
d = uminus(a)
```

Description

`d = uminus(a)` computes the unary negation of the matrix `a`.

`uminus(a)` is equivalent to `-a`.

See also

`uplus`

Purpose

Set union.

Synopsis

```
c = union(a,b)
c = union(a,b,'rows')
[c,ai,bi] = union(...)
```

Description

`c = union(a,b)` returns the set union of `a` and `b`. `a` and `b` can be either arrays or cell arrays of strings.

`c = union(a,b,'rows')`, where `a` and `b` must be 2D matrices, returns the row set union, that is, the unique rows of `a` and `b` combined. `a` and `b` must have the same number of columns.

`[c,ai,bi] = union(...)` also returns the index vectors `ai` and `bi`, where `ai` contains the linear indices of the elements of `c` that belong to `a`, and `bi` contains the linear indices of the elements of `c` that belong to `b`. Elements that occur in both `a` and `b` are indexed in `bi`.

In addition to the fixed arguments, additional property-value pairs can be given at the end of the command.

TABLE 1-58: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
`sort`	'on' | 'off'	`'on'`	Controls whether or not output should be sorted.

Examples

```
a = [1 2 0 1 2 3];
b = [2 4 5 7 0 8];
```

`c= union(a,b)` returns `[0, 1, 2, 3, 4, 5, 7, 8]`.

`c1 = union(a,b,'sort','off')` returns the same result unsorted.

```
a = [1 2 3; 2 3 1; 3 4 5; 5 4 3; 4 3 5;1 3 3];
b = [3 4 5; 3 4 5; 1 2 2; 4 3 5];
union(a,b,'rows') returns
[1, 2, 2 ; 1, 2, 3 ; 1, 3, 3 ; 2, 3, 1 ; 3, 4, 5 ; 4, 3, 5 ; 5, 4, 3].
```

```
a = {'green','yellow','blue','green'};
b = {'red','purple','yellow'};
```

`union(a,b)` returns `{'blue', 'green', 'purple', 'red', 'yellow'}`.

.See also

`intersect`, `ismember`, `setdiff`, `setxor`, `unique`

Purpose

Retrieve unique elements.

Synopsis

```
b = unique(a)
b = unique(a,'rows')
[b,m,n] = unique(...)
```

Description

`b = unique(a)` returns a copy of `a` without repetitions. `a` can be either an array or a cell arrays of strings.

`b = unique(a,'rows')`, where `a` must be a 2D matrix, returns the unique rows of `a`.

`[b,ai,bi] = unique(...)` also returns the index vectors `ai` and `bi`. `ai` contains the linear indices of the last occurrence of each element in `a`, while `bi` contains the linear indices of where each element of `a` is in `b`.

In addition to the fixed arguments, additional property-value pairs can be given at the end of the command.

TABLE 1-59: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
`sort`	'on' | 'off'	`'on'`	Controls whether or not output should be sorted.

Examples

`a = [1 2 0 1 2 3];`
`unique(a)` returns `[0, 1, 2, 3]`.

`a = [1 2 3; 2 3 1; 3 4 5; 3 4 5; 4 3 5;1 2 3];`
`[b,ai,bi] = unique(a,'rows')` returns
`b = [1, 2, 3 ; 2, 3, 1 ; 3, 4, 5 ; 4, 3, 5]`,
`ai = [6 ; 2 ; 4 ; 5]` and `bi = [1 ; 2 ; 3 ; 3 ; 4 ; 1]`.

`[b1,ai1,bi1] = unique(a,'rows','sort','off')` returns the same result unsorted.

`a = {'green','yellow','blue','green', 'blue'};`
`unique(a)` returns `{'blue', 'green', 'yellow'}`.

See also

`intersect`, `ismember`, `setdiff`, `setxor`, `union`

Purpose Run a system command.

Synopsis

```
status = unix(cmd)
[status output] = unix(cmd)
```

Description `unix` is a synonym for `system`.

See also `dos`, `system`

Purpose Extract details from piecewise polynomial.

Synopsis

```
[breaks,coefs,pieces,order,dim] = unmkpp(pp)
```

Description `[breaks,coefs,pieces,order,dim] = unmkpp(pp)` returns the breaks, coefficients, number of pieces, order and dimension of the piecewise polynomial `pp`, represented by a structure (described in `ppval`).

See also `ppval`, `mkpp`, `pchip`, `spline`

Purpose Remove phase jumps.

Synopsis

```
b = unwrap(a)
b = unwrap(a, tol)
b = unwrap(a, tol, dim)
```

Description `b = unwrap(a)`, for a matrix `a`, returns a matrix with the same size as `a` but where jumps along the first nonunit dimension larger than `pi` have been replaced with the equivalent angle closest to 0.

`b = unwrap(a, tol)` replaces only jumps larger than `tol`.

`b = unwrap(a, tol, dim)` unwraps along the dimension `dim`.

Example `unwrap([0 1 2 2+1.5*pi])` is `[0 1 2 2-0.5*pi]`.

Purpose Compute the unary plus of a matrix.

Synopsis

```
d = uplus(a)
```

Description `d = uplus(a)` computes the unary plus of the matrix `a`.

`uplus(a)` is equivalent to `+a`.

See also `uminus`

Purpose Convert string to upper case

Synopsis

```
s2 = upper(s1)
```

Description `s2 = upper(s1)` converts the characters in the string `s1` to upper case. `s1` can also be a cell array of strings. In that case, a new cell array is returned where each of the strings has been converted to upper case.

See also `lower`

Purpose

Compute variance.

Synopsis

```
y = var(x)
y = var(x,w)
y = var(x,w,dim)
```

Description

`y = var(x)` and `y = var(x,0)` compute the variance of `x` using normalization by `n-1`, where `n` is the sample size.

`y = var(x,1)` computes the variance of `x` using normalization by `n`.

When `x` is a vector, `y` is the variance of `x`. When `x` is a matrix, `y` is a row vector where each element is the variance of the corresponding column of `x`. When `x` is an n-dimensional array, `y` is the variance along the first nonsingleton dimension of `x`.

`y = var(x,w)` computes the variance of `x` using the weight vector `w`, which `var` normalizes to sum to one. `w` must contain only nonnegative elements and must be of the same length as `x` along the dimension the variance is computed.

`y = var(x,w,dim)` computes the variance along the dimension `dim`.

Examples

```
x = [0 4 1;2 9 2;4 -1 0];
x2 = [-3 4 1;3 2 0;-1 8 1];
y(:,:,1)=x;y(:,:,2)=x2;
var(x) returns [4,25,1].
var(x,[1 1 3]) returns [2.56,16,0.64].
var(x,[],2) returns [4.333; 16.333 ; 7].
var(y,[],3) returns
[4.5,0,0; 0.5,24.5,2; 12.5,40.5,0.5].
```

See also

`corrcoef`, `cov`, `std`

Purpose Retrieve arguments to a function that has a variable number of input arguments.

Synopsis

```
c = varargin
```

Description `c = varargin` returns a cell array containing the last arguments to a function that has a variable number of input arguments. This can be done only in a function where the last input argument is `varargin`.

Example Suppose that a function `func` has the following declaration:

```
function out = func(x, varargin)
```

and that it is called with `func(2, 3, 5, 7)`. Then `varargin` returns `{3 5 7}`.

See also `varargout`

Purpose

Set the outputs from a function that has a variable number of outputs.

Synopsis

```
varargout
```

Description

After execution of a function where the last output argument is `varargout`, the value of the cell array `varargout` is used to determine the values of the last output arguments.

Example

Suppose that a function `func` has the following declaration:

```
function varargout = func(x)
```

and that it is called with `[a b c] = func(10)`. If `varargout` is a cell array with the contents `{2 3 5}` when `func` has executed, then the assignments `a = 2`, `b = 3`, and `c = 5` are made.

See also

`varargin`

Purpose Vectorize an expression.

Synopsis

```
r = vectorize(s)
```

Description `r = vectorize(s)` returns a copy of a string s where every occurrence of `'*'`, `'/'` and `'^'` are replaced with `'.*'`, `'./'`, and `'.^'`.

Example `vectorize('x*y - x.^2/y.^2 + 12')` returns `'x.*y - x.^2./y.^2 + 12'`.

Purpose Return the current version as a string.

Synopsis

```
version
version java
```

Description `version` returns the current version of COMSOL Script as a string.

`version java` returns the Java version used in COMSOL Script as a string.

Purpose Concatenate matrices or cell arrays vertically.

Synopsis

```
c = vertcat(arg1, ...)
```

Description `c = vertcat(arg1, ...)` returns the vertical concatenation of its input arguments. The arguments need not be of the same type; if they differ, the result is the common base type of all arguments.

`vertcat(arg1, ...)` is equivalent to `[arg1 ; ...]` or `cat(1, arg1, ...)`.

See also `cat`, `horzcat`

Purpose

Control position of view point.

Synopsis

```
view(2)
view(3)
view('xy')
view('yz')
view('zx')
view(az,elev)
view(ax,...)
```

Description

`view(2)` specifies that the plot should be viewed as a 2-D plot.

`view(3)` specifies that the plot should be viewed from the default 3D view.

`view('xy')` specifies that the plot should be viewed in the xy-plane.

`view('yz')` specifies that the plot should be viewed in the yz-plane.

`view('zx')` specifies that the plot should be viewed in the zx-plane.

`view(az,elev)` sets the view point at the azimuth `az` and the elevation `elev`. `az` is the horizontal rotation and `elev` is the vertical elevation. Both are given in degrees.

`view(ax,...)` controls the view in the axes `ax` instead of in the current axes.

Purpose

Display a warning.

Synopsis

```
warning(msg)
warning(msg, id)

warning('on')
warning('off')

s = warning('on', id)
s = warning('off', id)
```

Description

`warning(msg)` displays the warning message `msg`.

`warning(msg, id)` displays the warning message `msg` belonging to the category `id`.

`warning('on')` and `warning('off')` enable and disable, respectively, display of warnings.

`s = warning('on', id)` and `s = warning('off', id)` enable and disable, respectively, display of warnings belonging to the category `id`. It returns a structure containing the previous state of the warning category `id`.

See also

`error`

Purpose

Create a colormap suitable for wave phenomena.

Synopsis

```
wavemap(n)
```

Description

`wavemap(n)` returns a colormap with `n` colors. It is a matrix with `n` rows and 3 columns with RGB values for the colors in the colormap. The colors are blue to white and white to blue, suitable for wave phenomena.

See also

`colormap`, `bone`, `cool`, `gray`, `grayprint`, `jet`, `hot`, `hsv`, `pink`

Purpose

Read a .wav sound file.

Synopsis

```
data = wavread(name)
data = wavread(name, sz)
[data, rate] = wavread(name)
[data, rate] = wavread(name, sz)
[data, rate, nbits] = wavread(name)
[data, rate, nbits] = wavread(name, sz)
[data, rate, nbits, desc] = wavread(name)
[data, rate, nbits, desc] = wavread(name, sz)

[nframes, nchannels] = wavread(name, 'size')
```

Description

`data = wavread(name)` reads and returns pulse-code modulated signal data from the .wav file `name`. The number of bits per sample must be 8 or 16. For a mono signal, a column vector is returned, and for a stereo signal, an N-by-2 matrix is returned.

`[data, rate] = wavread(name)` also returns the sample rate.

`[data, rate, nbits] = wavread(name)` also returns the sample rate and the number of bits per sample.

`[data, rate, nbits, desc] = wavread(name)` also returns the sample rate, the number of bits per sample, and a structure containing a further description of the data (if available).

`wavread(..., sz)` only reads a part of the signal: If `sz` is a scalar, then the first `sz` signal values are read. If `sz` is a vector of length 2, then the signal values for positions `sz(1)..sz(2)` are read.

`[nframes, nchannels] = wavread(name, 'size')` returns the number of frames and channels but ignores the signal.

See also

`sound`, `soundsc`, `wavwrite`

Purpose

Write a .wav sound file.

Synopsis

```
wavwrite(data, name)
wavwrite(data, rate, name)
wavwrite(data, rate, nbits, name)
```

Description

`wavwrite(data, name)` writes the pulse-code modulated signal `data` to the .wav-file `name` using a sample rate of 8000Hz and 16 bits per sample. For a mono signal, `data` is a column vector, and for a stereo signal, `data` is an N-by-2 matrix. Signal values outside [-1,1] are clipped.

`wavwrite(data, rate, name)` writes the pulse-code modulated signal `data` to the .wav-file `name` using a sample rate of `rate` and 16 bits per sample.

`wavwrite(data, rate, nbits, name)` writes the pulse-code modulated signal `data` to the .wav-file `name` using a sample rate of `rate` and `nbits` bits per sample (must be 8 or 16).

See also

`wavread`

Purpose

Display the function or variable to which a name is mapped.

Synopsis

```
which(name)
w = which(name)

which(..., '-all')
which(..., '-subfun')
```

Description

`which(name)` displays the function, variable, or built-in function to which `name` is mapped.

`w = which(name)` returns the name of the function, variable, or built-function to which `name` is mapped.

`which(..., '-all')` displays or returns all candidate maps listed in order of decreasing priority.

`which(name, '-subfun')` displays or returns all subfunctions in the function called `name`.

See also

`exist`

Purpose

Get the names of variables in the workspace.

Synopsis

```
who
v = who

who(name1, ...)
v = who(name1, ...)
```

Description

`who` displays the names of all workspace variables.

`v = who` returns a cell array containing the names of all workspace variables.

`who(name1, ...)` displays the names of all workspace variables matching any of the `name`*`i`*. The variable names may contain the wildcard `*`, which matches any character sequence.

`v = who(name1, ...)` returns a cell array containing the names of all workspace variables matching any of the `name`*`i`*.

See also

`whos`

Purpose

Get information about variables in the workspace.

Synopsis

```
whos
v = whos

whos(name1, ...)
v = whos(name1, ...)
```

Description

`whos` displays information about the variables in the workspace.

`v = whos` returns a structure array with one element for each variable in the workspace. It contains the following fields:.

FIELD	CONTENTS
`name`	variable name
`size`	dimensions
`bytes`	approximate number of bytes occupied
`class`	class

`whos(name1, ...)` displays information about the variables in the workspace with names matching any of the `name`*`i`*. The variable names may contain the wildcard `*`, which matches any character sequence.

`v = whos(name1, ...)` returns a structure array with one element for each workspace variable matching any of the `name`*`i`*.

See also

`who`

Purpose

Controls axis limits.

Synopsis

```
lim = xlim
mode = xlim('mode')
xlim(limits)
xlim(mode)
xlim(ax,...)
```

Description

`lim = xlim` returns the *x*-axis limits for the current axes.

`mode = xlim('mode')` returns `'auto'` or `'manual'` for the *x*-axis limits mode.

`xlim(limits)` sets the *x*-axis limits to the limits given by the 2-element vector `limits`.

`xlim(mode)`, where `mode` is the string `'auto'` or `'manual'`, sets the *x*-axis limits mode.

`xlim(ax,...)` uses the axes `ax` instead of the current axes.

The `ylim` and `zlim` functions have the same functionality as `xlim` but operate on the *y*- and *z*-axis, respectively.

Purpose Specify an x-axis label.

Synopsis `xlabel(string)`

Description `xlabel(string)` places the text `string` as the label on the x-axis.

See the `text` command for a list of valid Greek symbols and HTML formatting syntax.

See also `text`

Purpose

Read from .xls file

Synopsis

```
num = xlsread(filename,sheet)
[num,txt] = xlsread(filename,sheet)
[num,txt,raw] = xlsread(filename,sheet)
[num,txt,raw] = xlsread(filename,range)
[num,txt,raw] = xlsread(filename,sheet,range)
xlsread(filename,...)
```

Description

`[num,txt,raw] = xlsread(filename)` reads all entries from the first sheet of an .xls file `filename` and returns numerical data in matrix `num`, strings in cell array `txt` and mixed output in cell array `raw`. Excel cells containing string values will appear as `NaN` in `num` while numerical values will appear as empty strings in `txt`. Likewise for empty or unreadable cells (such as error codes, images etc). Empty leading rows or columns will be ignored.

`xlsread(filename,sheet)`, `xlsread(filename,range)` and `xlsread(filename,sheet,range)` read from a specific sheet and/or range. `sheet` must be given either as a number or a string containing the sheet's name. `range` must be a string in Excels A1-notation, for example `'A1:F24'`.

`xlsread(filename,...)` supports property value pairs as follows:

TABLE 1-60: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
`out`	`'num' | 'raw' | 'text'`	`{'num', 'text', 'raw'}`	Output variables.
`range`	`string`	`':'`	Range to read from.

TABLE 1-60: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
sheet	string or positive integer	1	Sheet to read from.
trim	'on' | 'off'	'on'	If 'on', xlsread trims leading and trailing rows/columns of NaNs (for num, raw) or empty strings (for text). For example, a leading row containing only strings will automatically be removed from num, as opposed to appearing as a row of NaN's. Note that if range has been given, setting trim to 'off' does not guarantee output of corresponding size, as completely empty trailing rows and columns will still not be read.

xlsread supports Excel 97 format and later.

Note: Excel cells stored in date format will be returned as numbers or strings depending on the format in which they were stored in the xls file. Numerical dates are based on the number of serial days elapsed since January 1, 1900.

See also xlswrite

Purpose

Write to .xls file

Synopsis

```
xlswrite(filename,data)
xlswrite(filename,data,sheet)
xlswrite(filename,data,range)
xlswrite(filename,data,sheet,range)
xlswrite(filename,data,...)
```

Description

`xlswrite(filename, data)` writes entries in data to the first sheet of an .xls file `filename`. `data` can be a real matrix or a cell array. In the latter case, only numerical and string entries are printed to `filename`. Empty strings or `NaN` entries are ignored.

`xlsread(filename,data,sheet)`, `xlsread(filename,data,range)` and `xlsread(filename,data,sheet,range)` write to a specific sheet and/or range. `sheet` must be given either as a number or a string containing the sheet's name. `range` must be a string in Excels A1-notation, for example `'A1:F24'`.

`xlswrite(filename,data,...)` supports property value pairs as follows:

TABLE 1-61: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
`range`	`string`	`':'`	Range to write to.
`sheet`	`string or positive integer`	1	Sheet to write to.

Examples

```
mat = [1:10; sin(1:10); exp(1:10)]';
xlswrite('myfile.xls',mat,'range','A1:B5');

c = [{'Header 1','Header 2','Header 3'};num2cell(mat)];
xlswrite('myfile.xls',c,'My sheet');
```

See also

`xlsread`

Purpose Specify a *y*-axis label.

Synopsis

```
ylabel(string)
```

Description `ylabel(string)` places the text `string` as the label on the *y*-axis.

See the `text` command for a list of valid Greek symbols and HTML formatting syntax.

See also `text`

Purpose

Compute the logical XOR of two matrices pointwise.

Synopsis

```
d = xor(a, b)
```

Description

`d = xor(a, b)` computes the pointwise logical XOR of the two matrices `a` and `b`. For each dimension, `a` and `b` must have the same size or either of them must have size 1. In the latter case, the unit dimension is expanded to the size of the nonunit dimension.

Examples

```
xor([0 0 1 1], [0 1 0 1])
```

```
xor([0 1], 0)
```

```
xor([0 1], [1 ; 0])
```

See also

`and`, `not`, `or`

Purpose Create an all-zero matrix.

Synopsis

```
m = zeros(n)
m = zeros(sz)
m = zeros(n1, n2, ...)
```

Description `m = zeros(n)`, where `n` is an integer, returns an `n`-by-`n` all-zero matrix.

`m = zeros(sz)`, where `sz` is a vector of integers, returns an all-zero matrix of size `sz`.

`m = zeros(n1,n2,...)`, where `ni` are integers, returns an `n1`x`n2`x `...` all-zero matrix.

See also `eye`, `ones`, `repmat`, `zeros`

Purpose Specify a *z*-axis label.

Synopsis `zlabel(string)`

Description `zlabel(string)` places the text `string` as the label on the *z*-axis.

See the `text` command for a list of valid Greek symbols and HTML formatting syntax.

See also `text`

INDEX