MATLAB®

The Language of Technical Computing

Computation

Visualization

Programming

The MATH WORKS Inc.

MATLAB Function Reference
(Volume 2: Graphics)

Version 5

How to Contact The MathWorks:

508-647-7000	Phone
508-647-7001	Fax
The MathWorks, Inc. 24 Prime Park Way Natick, MA 01760-1500	Mail
`http://www.mathworks.com`	Web
`ftp.mathworks.com`	Anonymous FTP server
`comp.soft-sys.matlab`	Newsgroup
`support@mathworks.com`	Technical support
`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`subscribe@mathworks.com`	Subscribing user registration
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information

MATLAB Function Reference

Printing History:	December 1996	First printing	(for MATLAB 5)
	June 1997	Revised for 5.1	(online version)
	October 1997	Revised for 5.2	(online version)
	January 1999	Revised for Release 11	(online version)

Contents

1 Command Summary

1

Command Summary

This chapter lists MATLAB commands by functional area.

General Purpose Commands

Managing Commands and Functions

`addpath`	Add directories to MATLAB's search path
`doc`	Display HTML documentation in Web browser
`docopt`	Display location of help file directory for UNIX platforms
`help`	Online help for MATLAB functions and M-files
`helpdesk`	Display Help Desk page in Web browser, giving access to extensive help
`helpwin`	Display Help Window, providing access to help for all commands
`lasterr`	Last error message
`lastwarn`	Last warning message
`lookfor`	Keyword search through all help entries
`partialpath`	Partial pathname
`path`	Control MATLAB's directory search path
`pathtool`	Start Path Browser, a GUI for viewing and modifying MATLAB's path
`profile`	Start the M-file profiler, a utility for debugging and optimizing code
`profreport`	Generate a profile report
`rmpath`	Remove directories from MATLAB's search path
`type`	List file
`ver`	Display version information for MATLAB, Simulink, and toolboxes
`version`	MATLAB version number
`web`	Point Web browser at file or Web site
`what`	Directory listing of M-files, MAT-files, and MEX-files
`whatsnew`	Display README files for MATLAB and toolboxes
`which`	Locate functions and files

Managing Variables and the Workspace

`clear`	Remove items from memory
`disp`	Display text or array
`length`	Length of vector
`load`	Retrieve variables from disk
`mlock`	Prevent M-file clearing
`munlock`	Allow M-file clearing
`openvar`	Open workspace variable in Array Editor, for graphical editing
`pack`	Consolidate workspace memory
`save`	Save workspace variables on disk
`saveas`	Save figure or model using specified format
`size`	Array dimensions
`who, whos`	List directory of variables in memory
`workspace`	Display the Workspace Browser, a GUI for managing the workspace

Controlling the Command Window

`clc`	Clear command window
`echo`	Echo M-files during execution
`format`	Control the output display format
`home`	Send the cursor home
`more`	Control paged output for the command window

Working with Files and the Operating Environment

`cd`	Change working directory
`copyfile`	Copy file
`delete`	Delete files and graphics objects
`diary`	Save session in a disk file
`dir`	Directory listing
`edit`	Edit an M-file
`fileparts`	Filename parts
`fullfile`	Build full filename from parts
`inmem`	Functions in memory
`ls`	List directory on UNIX
`matlabroot`	Root directory of MATLAB installation
`mkdir`	Make directory
`open`	Open files based on extension
`pwd`	Display current directory
`tempdir`	Return the name of the system's temporary directory
`tempname`	Unique name for temporary file
`!`	Execute operating system command

Starting and Quitting MATLAB

`matlabrc`	MATLAB startup M-file
`quit`	Terminate MATLAB
`startup`	MATLAB startup M-file

Operators and Special Characters

`+`	Plus
`-`	Minus
`*`	Matrix multiplication
`.*`	Array multiplication
`^`	Matrix power
`.^`	Array power
`kron`	Kronecker tensor product

<table>
<tr><td>\</td><td>Backslash or left division</td></tr>
<tr><td>/</td><td>Slash or right division</td></tr>
<tr><td>./ and .\</td><td>Array division, right and left</td></tr>
<tr><td>:</td><td>Colon</td></tr>
<tr><td>( )</td><td>Parentheses</td></tr>
<tr><td>[ ]</td><td>Brackets</td></tr>
<tr><td>{}</td><td>Curly braces</td></tr>
<tr><td>.</td><td>Decimal point</td></tr>
<tr><td>...</td><td>Continuation</td></tr>
<tr><td>,</td><td>Comma</td></tr>
<tr><td>;</td><td>Semicolon</td></tr>
<tr><td>%</td><td>Comment</td></tr>
<tr><td>!</td><td>Exclamation point</td></tr>
<tr><td>'</td><td>Transpose and quote</td></tr>
<tr><td>.'</td><td>Nonconjugated transpose</td></tr>
<tr><td>=</td><td>Assignment</td></tr>
<tr><td>==</td><td>Equality</td></tr>
<tr><td>< ></td><td>Relational operators</td></tr>
<tr><td>&</td><td>Logical AND</td></tr>
<tr><td>|</td><td>Logical OR</td></tr>
<tr><td>~</td><td>Logical NOT</td></tr>
<tr><td>xor</td><td>Logical EXCLUSIVE OR</td></tr>
</table>

Logical Functions

all	Test to determine if all elements are nonzero
any	Test for any nonzeros
exist	Check if a variable or file exists
find	Find indices and values of nonzero elements
is*	Detect state
isa	Detect an object of a given class
logical	Convert numeric values to logical
mislocked	True if M-file cannot be cleared

Language Constructs and Debugging

MATLAB as a Programming Language

builtin	Execute builtin function from overloaded method
eval	Interpret strings containing MATLAB expressions
evalc	Evaluate MATLAB expression with capture

`evalin`	Evaluate expression in workspace
`feval`	Function evaluation
`function`	Function M-files
`global`	Define global variables
`nargchk`	Check number of input arguments
`persistent`	Define persistent variable
`script`	Script M-files

Control Flow

`break`	Terminate execution of `for` loop or `while` loop
`case`	Case switch
`catch`	Begin catch block
`else`	Conditionally execute statements
`elseif`	Conditionally execute statements
`end`	Terminate `for`, `while`, `switch`, `try`, and `if` statements or indicate last index
`error`	Display error messages
`for`	Repeat statements a specific number of times
`if`	Conditionally execute statements
`otherwise`	Default part of `switch` statement
`return`	Return to the invoking function
`switch`	Switch among several cases based on expression
`try`	Begin `try` block
`warning`	Display warning message
`while`	Repeat statements an indefinite number of times

Interactive Input

`input`	Request user input
`keyboard`	Invoke the keyboard in an M-file
`menu`	Generate a menu of choices for user input
`pause`	Halt execution temporarily

Object-Oriented Programming

`class`	Create object or return class of object
`double`	Convert to double precision
`inferiorto`	Inferior class relationship
`inline`	Construct an inline object
`int8`, `int16`, `int32`	Convert to signed integer
`isa`	Detect an object of a given class

`loadobj`	Extends the `load` function for user objects
`saveobj`	Save filter for objects
`single`	Convert to single precision
`superiorto`	Superior class relationship
`uint8`, `uint16`, `uint32`	Convert to unsigned integer

Debugging

`dbclear`	Clear breakpoints
`dbcont`	Resume execution
`dbdown`	Change local workspace context
`dbmex`	Enable MEX-file debugging
`dbquit`	Quit debug mode
`dbstack`	Display function call stack
`dbstatus`	List all breakpoints
`dbstep`	Execute one or more lines from a breakpoint
`dbstop`	Set breakpoints in an M-file function
`dbtype`	List M-file with line numbers
`dbup`	Change local workspace context

Elementary Matrices and Matrix Manipulation

Elementary Matrices and Arrays

`blkdiag`	Construct a block diagonal matrix from input arguments
`eye`	Identity matrix
`linspace`	Generate linearly spaced vectors
`logspace`	Generate logarithmically spaced vectors
`ones`	Create an array of all ones
`rand`	Uniformly distributed random numbers and arrays
`randn`	Normally distributed random numbers and arrays
`zeros`	Create an array of all zeros
`:` (colon)	Regularly spaced vector

Special Variables and Constants

`ans`	The most recent answer
`computer`	Identify the computer on which MATLAB is running
`eps`	Floating-point relative accuracy
`flops`	Count floating-point operations
`i`	Imaginary unit

`Inf`	Infinity
`inputname`	Input argument name
`j`	Imaginary unit
`NaN`	Not-a-Number
`nargin`, `nargout`	Number of function arguments
`pi`	Ratio of a circle's circumference to its diameter,π
`realmax`	Largest positive floating-point number
`realmin`	Smallest positive floating-point number
`varargin`, `varargout`	Pass or return variable numbers of arguments

Time and Dates

`calendar`	Calendar
`clock`	Current time as a date vector
`cputime`	Elapsed CPU time
`date`	Current date string
`datenum`	Serial date number
`datestr`	Date string format
`datevec`	Date components
`eomday`	End of month
`etime`	Elapsed time
`now`	Current date and time
`tic, toc`	Stopwatch timer
`weekday`	Day of the week

Matrix Manipulation

`cat`	Concatenate arrays
`diag`	Diagonal matrices and diagonals of a matrix
`fliplr`	Flip matrices left-right
`flipud`	Flip matrices up-down
`repmat`	Replicate and tile an array
`reshape`	Reshape array
`rot90`	Rotate matrix 90 degrees
`tril`	Lower triangular part of a matrix
`triu`	Upper triangular part of a matrix
`:` (colon)	Index into array, rearrange array

Specialized Matrices

`compan`	Companion matrix
`gallery`	Test matrices
`hadamard`	Hadamard matrix
`hankel`	Hankel matrix
`hilb`	Hilbert matrix
`invhilb`	Inverse of the Hilbert matrix
`magic`	Magic square
`pascal`	Pascal matrix
`toeplitz`	Toeplitz matrix
`wilkinson`	Wilkinson's eigenvalue test matrix

Elementary Math Functions

`abs`	Absolute value and complex magnitude
`acos, acosh`	Inverse cosine and inverse hyperbolic cosine
`acot, acoth`	Inverse cotangent and inverse hyperbolic cotangent
`acsc, acsch`	Inverse cosecant and inverse hyperbolic cosecant
`angle`	Phase angle
`asec, asech`	Inverse secant and inverse hyperbolic secant
`asin, asinh`	Inverse sine and inverse hyperbolic sine
`atan, atanh`	Inverse tangent and inverse hyperbolic tangent
`atan2`	Four-quadrant inverse tangent
`ceil`	Round toward infinity
`complex`	Construct complex data from real and imaginary components
`conj`	Complex conjugate
`cos, cosh`	Cosine and hyperbolic cosine
`cot, coth`	Cotangent and hyperbolic cotangent
`csc, csch`	Cosecant and hyperbolic cosecant
`exp`	Exponential
`fix`	Round towards zero
`floor`	Round towards minus infinity
`gcd`	Greatest common divisor
`imag`	Imaginary part of a complex number
`lcm`	Least common multiple
`log`	Natural logarithm
`log2`	Base 2 logarithm and dissect floating-point numbers into exponent and mantissa
`log10`	Common (base 10) logarithm
`mod`	Modulus (signed remainder after division)
`nchoosek`	Binomial coefficient or all combinations

`real`	Real part of complex number
`rem`	Remainder after division
`round`	Round to nearest integer
`sec, sech`	Secant and hyperbolic secant
`sign`	Signum function
`sin, sinh`	Sine and hyperbolic sine
`sqrt`	Square root
`tan, tanh`	Tangent and hyperbolic tangent

Specialized Math Functions

`airy`	Airy functions
`besselh`	Bessel functions of the third kind (Hankel functions)
`besseli`, `besselk`	Modified Bessel functions
`besselj`, `bessely`	Bessel functions
`beta`, `betainc`, `betaln`	Beta functions
`ellipj`	Jacobi elliptic functions
`ellipke`	Complete elliptic integrals of the first and second kind
`erf`, `erfc`, `erfcx`, `erfinv`	Error functions
`expint`	Exponential integral
`factorial`	Factorial function
`gamma`, `gammainc`, `gammaln`	Gamma functions
`legendre`	Associated Legendre functions
`pow2`	Base 2 power and scale floating-point numbers
`rat, rats`	Rational fraction approximation

Coordinate System Conversion

`cart2pol`	Transform Cartesian coordinates to polar or cylindrical
`cart2sph`	Transform Cartesian coordinates to spherical
`pol2cart`	Transform polar or cylindrical coordinates to Cartesian
`sph2cart`	Transform spherical coordinates to Cartesian

Matrix Functions - Numerical Linear Algebra

Matrix Analysis

`cond`	Condition number with respect to inversion
`condeig`	Condition number with respect to eigenvalues
`det`	Matrix determinant
`norm`	Vector and matrix norms
`null`	Null space of a matrix
`orth`	Range space of a matrix
`rank`	Rank of a matrix7
`rcond`	Matrix reciprocal condition number estimate
`rref`, `rrefmovie`	Reduced row echelon form
`subspace`	Angle between two subspaces
`trace`	Sum of diagonal elements

Linear Equations

`chol`	Cholesky factorization
`inv`	Matrix inverse
`lscov`	Least squares solution in the presence of known covariance
`lu`	LU matrix factorization
`lsqnonneg`	Nonnegative least squares
`pinv`	Moore-Penrose pseudoinverse of a matrix
`qr`	Orthogonal-triangular decomposition

Eigenvalues and Singular Values

`balance`	Improve accuracy of computed eigenvalues
`cdf2rdf`	Convert complex diagonal form to real block diagonal form
`eig`	Eigenvalues and eigenvectors
`gsvd`	Generalized singular value decomposition
`hess`	Hessenberg form of a matrix
`poly`	Polynomial with specified roots
`qz`	QZ factorization for generalized eigenvalues
`rsf2csf`	Convert real Schur form to complex Schur form
`schur`	Schur decomposition
`svd`	Singular value decomposition

Matrix Functions

`expm`	Matrix exponential

`funm`	Evaluate functions of a matrix
`logm`	Matrix logarithm7
`sqrtm`	Matrix square root

Low Level Functions

`qrdelete`	Delete column from QR factorization
`qrinsert`	Insert column in QR factorization

Data Analysis and Fourier Transform Functions

Basic Operations

`convhull`	Convex hull
`cumprod`	Cumulative product
`cumsum`	Cumulative sum
`cumtrapz`	Cumulative trapezoidal numerical integration
`delaunay`	Delaunay triangulation
`dsearch`	Search for nearest point
`factor`	Prime factors
`inpolygon`	Detect points inside a polygonal region
`max`	Maximum elements of an array
`mean`	Average or mean value of arrays
`median`	Median value of arrays
`min`	Minimum elements of an array
`perms`	All possible permutations
`polyarea`	Area of polygon
`primes`	Generate list of prime numbers
`prod`	Product of array elements
`sort`	Sort elements in ascending order
`sortrows`	Sort rows in ascending order
`std`	Standard deviation
`sum`	Sum of array elements
`trapz`	Trapezoidal numerical integration
`tsearch`	Search for enclosing Delaunay triangle
`var`	Variance
`voronoi`	Voronoi diagram

Finite Differences

`del2`	Discrete Laplacian
`diff`	Differences and approximate derivatives

`gradient`	Numerical gradient

Correlation

`corrcoef`	Correlation coefficients
`cov`	Covariance matrix

Filtering and Convolution

`conv`	Convolution and polynomial multiplication
`conv2`	Two-dimensional convolution
`deconv`	Deconvolution and polynomial division
`filter`	Filter data with an infinite impulse response (IIR) or finite impulse response (FIR) filter
`filter2`	Two-dimensional digital filtering

Fourier Transforms

`abs`	Absolute value and complex magnitude
`angle`	Phase angle
`cplxpair`	Sort complex numbers into complex conjugate pairs
`fft`	One-dimensional fast Fourier transform
`fft2`	Two-dimensional fast Fourier transform
`fftshift`	Shift DC component of fast Fourier transform to center of spectrum
`ifft`	Inverse one-dimensional fast Fourier transform
`ifft2`	Inverse two-dimensional fast Fourier transform
`ifftn`	Inverse multidimensional fast Fourier transform
`ifftshift`	Inverse FFT shift
`nextpow2`	Next power of two
`unwrap`	Correct phase angles

Vector Functions

`cross`	Vector cross product
`intersect`	Set intersection of two vectors
`ismember`	Detect members of a set
`setdiff`	Return the set difference of two vector
`setxor`	Set exclusive or of two vectors
`union`	Set union of two vectors
`unique`	Unique elements of a vector

Polynomial and Interpolation Functions

Polynomials

`conv`	Convolution and polynomial multiplication
`deconv`	Deconvolution and polynomial division
`poly`	Polynomial with specified roots
`polyder`	Polynomial derivative
`polyeig`	Polynomial eigenvalue problem
`polyfit`	Polynomial curve fitting
`polyval`	Polynomial evaluation
`polyvalm`	Matrix polynomial evaluation
`residue`	Convert between partial fraction expansion and polynomial coefficients
`roots`	Polynomial roots

Data Interpolation

`griddata`	Data gridding
`interp1`	One-dimensional data interpolation (table lookup)
`interp2`	Two-dimensional data interpolation (table lookup)
`interp3`	Three-dimensional data interpolation (table lookup)
`interpft`	One-dimensional interpolation using the FFT method
`interpn`	Multidimensional data interpolation (table lookup)
`meshgrid`	Generate X and Y matrices for three-dimensional plots
`ndgrid`	Generate arrays for multidimensional functions and interpolation
`spline`	Cubic spline interpolation

Function Functions – Nonlinear Numerical Methods

`dblquad`	Numerical double integration
`fminbnd`	Minimize a function of one variable
`fminsearch`	Minimize a function of several variables
`fzero`	Zero of a function of one variable
`ode45`, `ode23`, `ode113`, `ode15s`, `ode23s`, `ode23t`, `ode23tb`	Solve differential equations
`odefile`	Define a differential equation problem for ODE solvers
`odeget`	Extract properties from options structure created with `odeset`
`odeset`	Create or alter options structure for input to ODE solvers
`quad`, `quad8`	Numerical evaluation of integrals
`vectorize`	Vectorize expression

Sparse Matrix Functions

Elementary Sparse Matrices

`spdiags`	Extract and create sparse band and diagonal matrices
`speye`	Sparse identity matrix
`sprand`	Sparse uniformly distributed random matrix
`sprandn`	Sparse normally distributed random matrix
`sprandsym`	Sparse symmetric random matrix

Full to Sparse Conversion

`find`	Find indices and values of nonzero elements
`full`	Convert sparse matrix to full matrix
`sparse`	Create sparse matrix
`spconvert`	Import matrix from sparse matrix external format

Working with Nonzero Entries of Sparse Matrices

`nnz`	Number of nonzero matrix elements
`nonzeros`	Nonzero matrix elements
`nzmax`	Amount of storage allocated for nonzero matrix elements
`spalloc`	Allocate space for sparse matrix
`spfun`	Apply function to nonzero sparse matrix elements
`spones`	Replace nonzero sparse matrix elements with ones

Visualizing Sparse Matrices

`spy`	Visualize sparsity pattern

Reordering Algorithms

`colmmd`	Sparse column minimum degree permutation
`colperm`	Sparse column permutation based on nonzero count
`dmperm`	Dulmage-Mendelsohn decomposition
`randperm`	Random permutation
`symmmd`	Sparse symmetric minimum degree ordering
`symrcm`	Sparse reverse Cuthill-McKee ordering

Norm, Condition Number, and Rank

`condest`	1-norm matrix condition number estimate
`normest`	2-norm estimate

Sparse Systems of Linear Equations

`bicg`	BiConjugate Gradients method
`bicgstab`	BiConjugate Gradients Stabilized method
`cgs`	Conjugate Gradients Squared method
`cholinc`	Sparse Incomplete Cholesky and Cholesky-Infinity factorizations
`cholupdate`	Rank 1 update to Cholesky factorization
`gmres`	Generalized Minimum Residual method (with restarts)
`luinc`	Incomplete LU matrix factorizations
`pcg`	Preconditioned Conjugate Gradients method
`qmr`	Quasi-Minimal Residual method
`qr`	Orthogonal-triangular decomposition
`qrdelete`	Delete column from QR factorization
`qrinsert`	Insert column in QR factorization
`qrupdate`	Rank 1 update to QR factorization

Sparse Eigenvalues and Singular Values

`eigs`	Find eigenvalues and eigenvectors
`svds`	Find singular values

Miscellaneous

`spparms`	Set parameters for sparse matrix routines

Sound Processing Functions

General Sound Functions

`lin2mu`	Convert linear audio signal to mu-law
`mu2lin`	Convert mu-law audio signal to linear
`sound`	Convert vector into sound
`soundsc`	Scale data and play as sound

SPARCstation-Specific Sound Functions

`auread`	Read NeXT/SUN (.au) sound file
`auwrite`	Write NeXT/SUN (.au) sound file

.WAV Sound Functions

`wavread`	Read Microsoft WAVE (.wav) sound file
`wavwrite`	Write Microsoft WAVE (.wav) sound file

Character String Functions

General

`abs`	Absolute value and complex magnitude
`eval`	Interpret strings containing MATLAB expressions
`real`	Real part of complex number
`strings`	MATLAB string handling

String Manipulation

`deblank`	Strip trailing blanks from the end of a string
`findstr`	Find one string within another
`lower`	Convert string to lower case
`strcat`	String concatenation
`strcmp`	Compare strings
`strcmpi`	Compare strings ignoring case
`strjust`	Justify a character array
`strmatch`	Find possible matches for a string
`strncmp`	Compare the first `n` characters of two strings
`strrep`	String search and replace
`strtok`	First token in string
`strvcat`	Vertical concatenation of strings
`symvar`	Determine symbolic variables in an expression
`texlabel`	Produce the TeX format from a character string
`upper`	Convert string to upper case

String to Number Conversion

`char`	Create character array (string)
`int2str`	Integer to string conversion
`mat2str`	Convert a matrix into a string
`num2str`	Number to string conversion
`sprintf`	Write formatted data to a string
`sscanf`	Read string under format control
`str2double`	Convert string to double-precision value
`str2num`	String to number conversion

Radix Conversion

`bin2dec`	Binary to decimal number conversion
`dec2bin`	Decimal to binary number conversion
`dec2hex`	Decimal to hexadecimal number conversion

`hex2dec`	IEEE hexadecimal to decimal number conversion
`hex2num`	Hexadecimal to double number conversion

Low-Level File I/O Functions

File Opening and Closing

`fclose`	Close one or more open files
`fopen`	Open a file or obtain information about open files

Unformatted I/O

`fread`	Read binary data from file
`fwrite`	Write binary data to a file

Formatted I/O

`fgetl`	Return the next line of a file as a string without line terminator(s)
`fgets`	Return the next line of a file as a string with line terminator(s)
`fprintf`	Write formatted data to file
`fscanf`	Read formatted data from file

File Positioning

`feof`	Test for end-of-file
`ferror`	Query MATLAB about errors in file input or output
`frewind`	Rewind an open file
`fseek`	Set file position indicator
`ftell`	Get file position indicator

String Conversion

`sprintf`	Write formatted data to a string
`sscanf`	Read string under format control

Specialized File I/O

`dlmread`	Read an ASCII delimited file into a matrix
`dlmwrite`	Write a matrix to an ASCII delimited file
`hdf`	HDF interface
`imfinfo`	Return information about a graphics file
`imread`	Read image from graphics file

`imwrite`	Write an image to a graphics file
`textread`	Read formatted data from text file
`wk1read`	Read a Lotus123 WK1 spreadsheet file into a matrix
`wk1write`	Write a matrix to a Lotus123 WK1 spreadsheet file

Bitwise Functions

`bitand`	Bit-wise AND
`bitcmp`	Complement bits
`bitor`	Bit-wise OR
`bitmax`	Maximum floating-point integer
`bitset`	Set bit
`bitshift`	Bit-wise shift
`bitget`	Get bit
`bitxor`	Bit-wise XOR

Structure Functions

`fieldnames`	Field names of a structure
`getfield`	Get field of structure array
`rmfield`	Remove structure fields
`setfield`	Set field of structure array
`struct`	Create structure array
`struct2cell`	Structure to cell array conversion

Object Functions

`class`	Create object or return class of object
`isa`	Detect an object of a given class

Cell Array Functions

`cell`	Create cell array
`cellfun`	Apply a function to each element in a cell array
`cellstr`	Create cell array of strings from character array
`cell2struct`	Cell array to structure array conversion
`celldisp`	Display cell array contents
`cellplot`	Graphically display the structure of cell arrays
`num2cell`	Convert a numeric array into a cell array

Multidimensional Array Functions

`cat`	Concatenate arrays
`flipdim`	Flip array along a specified dimension
`ind2sub`	Subscripts from linear index
`ipermute`	Inverse permute the dimensions of a multidimensional array
`ndgrid`	Generate arrays for multidimensional functions and interpolation
`ndims`	Number of array dimensions
`permute`	Rearrange the dimensions of a multidimensional array
`reshape`	Reshape array
`shiftdim`	Shift dimensions
`squeeze`	Remove singleton dimensions
`sub2ind`	Single index from subscripts

Plotting and Data Visualization

Basic Plots and Graphs

`bar`	Vertical bar chart
`barh`	Horizontal bar chart
`hist`	Plot histograms
`hold`	Hold current graph
`loglog`	Plot using log-log scales
`pie`	Pie plot
`plot`	Plot vectors or matrices.
`polar`	Polar coordinate plot
`semilogx`	Semi-log scale plot
`semilogy`	Semi-log scale plot
`subplot`	Create axes in tiled positions

Three-Dimensional Plotting

`bar3`	Vertical 3-D bar chart
`bar3h`	Horizontal 3-D bar chart
`comet3`	3-D comet plot
`cylinder`	Generate cylinder
`fill3`	Draw filled 3-D polygons in 3-space
`plot3`	Plot lines and points in 3-D space
`quiver3`	3-D quiver (or velocity) plot
`slice`	Volumetric slice plot
`sphere`	Generate sphere
`stem3`	Plot discrete surface data

`waterfall`	Waterfall plot

Plot Annotation and Grids

`clabel`	Add contour labels to a contour plot
`datetick`	Date formatted tick labels
`grid`	Grid lines for 2-D and 3-D plots
`gtext`	Place text on a 2-D graph using a mouse
`legend`	Graph legend for lines and patches
`plotyy`	Plot graphs with Y tick labels on the left and right
`title`	Titles for 2-D and 3-D plots
`xlabel`	X-axis labels for 2-D and 3-D plots
`ylabel`	Y-axis labels for 2-D and 3-D plots
`zlabel`	Z-axis labels for 3-D plots

Surface, Mesh, and Contour Plots

`contour`	Contour (level curves) plot
`contourc`	Contour computation
`contourf`	Filled contour plot
`hidden`	Mesh hidden line removal mode
`meshc`	Combination mesh/contourplot
`mesh`	3-D mesh with reference plane
`peaks`	A sample function of two variables
`surf`	3-D shaded surface graph
`surface`	Create surface low-level objects
`surfc`	Combination surf/contourplot
`surfl`	3-D shaded surface with lighting
`trimesh`	Triangular mesh plot
`trisurf`	Triangular surface plot

Volume Visualization

`coneplot`	Plot velocity vectors as cones in 3-D vector field
`contourslice`	Draw contours in volume slice plane
`isocaps`	Compute isosurface end-cap geometry
`isonormals`	Compute normals of isosurface vertices
`isosurface`	Extract isosurface data from volume data
`reducepatch`	Reduce the number of patch faces
`reducevolume`	Reduce number of elements in volume data set
`shrinkfaces`	Reduce the size of patch faces
`smooth3`	Smooth 3-D data
`stream2`	Compute 2-D stream line data

`stream3`	Compute 3-D stream line data
`streamline`	Draw stream lines from 2- or 3-D vector data
`surf2patch`	Convert srface data to patch data
`subvolume`	Extract subset of volume data set

Domain Generation

`griddata`	Data gridding and surface fitting
`meshgrid`	Generation of X and Y arrays for 3-D plots

Specialized Plotting

`area`	Area plot
`box`	Axis box for 2-D and 3-D plots
`comet`	Comet plot
`compass`	Compass plot
`errorbar`	Plot graph with error bars
`ezcontour`	Easy to use contour plotter
`ezcontourf`	Easy to use filled contour plotter
`ezmesh`	Easy to use 3-D mesh plotter
`ezmeshc`	Easy to use combination mesh/contour plotter
`ezplot`	Easy to use function plotter
`ezplot3`	Easy to use 3-D parametric curve plotter
`ezpolar`	Easy to use polar coordinate plotter
`ezsurf`	Easy to use 3-D colored surface plotter
`ezsurfc`	Easy to use combination surface/contour plotter
`feather`	Feather plot
`fill`	Draw filled 2-D polygons
`fplot`	Plot a function
`pareto`	Pareto char
`pie3`	3-D pie plot
`plotmatrix`	Scatter plot matrix
`pcolor`	Pseudocolor (checkerboard) plot
`rose`	Plot rose or angle histogram
`quiver`	Quiver (or velocity) plot
`ribbon`	Ribbon plot
`stairs`	Stairstep graph
`scatter`	Scatter plot
`scatter3`	3-D scatter plot
`stem`	Plot discrete sequence data
`convhull`	Convex hull
`delaunay`	Delaunay triangulation
`dsearch`	Search Delaunay triangulation for nearest point

`inpolygon`	True for points inside a polygonal region
`polyarea`	Area of polygon
`tsearch`	Search for enclosing Delaunay triangle
`voronoi`	Voronoi diagram

View Control

`camdolly`	Move camera position and target
`camlookat`	View specific objects
`camorbit`	Orbit about camera target
`campan`	Rotate camera target about camera position
`campos`	Set or get camera position
`camproj`	Set or get projection type
`camroll`	Rotate camera about viewing axis
`camtarget`	Set or get camera target
`camup`	Set or get camera up-vector
`camva`	Set or get camera view angle
`camzoom`	Zoom camera in or out
`daspect`	Set or get data aspect ratio
`pbaspect`	Set or get plot box aspect ratio
`view`	3-D graph viewpoint specification.
`viewmtx`	Generate view transformation matrices
`xlim`	Set or get the current *x*-axis limits
`ylim`	Set or get the current *y*-axis limits
`zlim`	Set or get the current *z*-axis limits

Lighting

`camlight`	Cerate or position Light
`diffuse`	Diffuse reflectance
`lighting`	Lighting mode
`lightingangle`	Position light in sphereical coordinates
`material`	Material reflectance mode
`specular`	Specular reflectance

Color Operations

`brighten`	Brighten or darken color map
`bwcontr`	Contrasting black and/or color
`caxis`	Pseudocolor axis scaling
`colorbar`	Display color bar (color scale)
`colorcube`	Enhanced color-cube color map
`colordef`	Set up color defaults

`colormap`	Set the color look-up table
`graymon`	Graphics figure defaults set for grayscale monitor
`hsv2rgb`	Hue-saturation-value to red-green-blue conversion
`rgb2hsv`	RGB to HSVconversion
`rgbplot`	Plot color map
`shading`	Color shading mode
`spinmap`	Spin the colormap
`surfnorm`	3-D surface normals
`whitebg`	Change axes background color for plots

Colormaps

`autumn`	Shades of red and yellow color map
`bone`	Gray-scale with a tinge of blue color map
`contrast`	Gray color map to enhance image contrast
`cool`	Shades of cyan and magenta color map
`copper`	Linear copper-tone color map
`flag`	Alternating red, white, blue, and black color map
`gray`	Linear gray-scale color map
`hot`	Black-red-yellow-white color map
`hsv`	Hue-saturation-value (HSV) color map
`jet`	Variant of HSV
`lines`	Line color colormap
`prism`	Colormap of prism colors
`spring`	Shades of magenta and yellow color map
`summer`	Shades of green and yellow colormap
`winter`	Shades of blue and green color map

Printing

`orient`	Hardcopy paper orientation
`print`	Print graph or save graph to file
`printopt`	Configure local printer defaults
`saveas`	Save figure to graphic file

Handle Graphics, General

`copyobj`	Make a copy of a graphics object and its children
`findobj`	Find objects with specified property values
`gcbo`	Return object whose callback is currently executing
`gco`	Return handle of current object
`get`	Get object properties
`rotate`	Rotate objects about specified origin and direction

`ishandle`	True for graphics objects
`set`	Set object properties

Handle Graphics, Object Creation

`axes`	Create Axes object
`figure`	Create Figure (graph) windows
`image`	Create Image (2-D matrix)
`light`	Create Light object (illuminates Patch and Surface)
`line`	Create Line object (3-D polylines)
`patch`	Create Patch object (polygons)
`rectangle`	Create Rectangle object (2-D rectangle)
surface	Create Surface (quadrilaterals)
`text`	Create Text object (character strings)
`uicontext`	Create context menu (popup associated with object)

Handle Graphics, Figure Windows

`capture`	Screen capture of the current figure
`clc`	Clear figure window
`clf`	Clear figure
`clg`	Clear figure (graph window)
`close`	Close specified window
`gcf`	Get current figure handle
`newplot`	Graphics M-file preamble for `NextPlot` property
`refresh`	Refresh figure
`saveas`	Save figure or model to desired output format

Handle Graphics, Axes

`axis`	Plot axis scaling and appearance
`cla`	Clear Axes
`gca`	Get current Axes handle

Object Manipulation

`propedit`	Edit all properties of any selected object
`reset`	Reset axis or figure
`rotate3d`	Interactively rotate the view of a 3-D plot
`selectmoveresize`	Interactively select, move, or resize objects
`shg`	Show graph window

Interactive User Input

`ginput`	Graphical input from a mouse or cursor
`zoom`	Zoom in and out on a 2-D plot

Region of Interest

`dragrect`	Drag XOR rectangles with mouse
`drawnow`	Complete any pending drawing
`rbbox`	Rubberband box

Graphical User Interface Creation

Dialog Boxes

`dialog`	Create a dialog box
`errordlg`	Create error dialog box
`helpdlg`	Display help dialog box
`inputdlg`	Create input dialog box
`listdlg`	Create list selection dialog box
`msgbox`	Create message dialog box
`pagedlg`	Display page layout dialog box
`printdlg`	Display print dialog box
`questdlg`	Create question dialog box
`uigetfile`	Display dialog box to retrieve name of file for reading
`uiputfile`	Display dialog box to retrieve name of file for writing
`uisetcolor`	Interactively set a `ColorSpec` using a dialog box
`uisetfont`	Interactively set a font using a dialog box
`warndlg`	Create warning dialog box

User Interface Objects

`menu`	Generate a menu of choices for user input
`menuedit`	Menu editor
`uicontextmenu`	Create context menu
`uicontrol`	Create user interface control
`uimenu`	Create user interface menu

Other Functions

`dragrect`	Drag rectangles with mouse
`findfigs`	Display off-screen visible figure windows
`gcbo`	Return handle of object whose callback is executing

`rbbox`	Create rubberband box for area selection
`selectmoveresize`	Select, move, resize, or copy Axes and Uicontrol graphics objects
`textwrap`	Return wrapped string matrix for given Uicontrol
`uiresume`	Used with `uiwait`, controls program execution
`uiwait`	Used with `uiresume`, controls program execution
`waitbar`	Display wait bar
`waitforbuttonpress`	Wait for key/buttonpress over figure

2

Reference

This chapter describes all MATLAB operators, commands, and functions in alphabetical order.

Purpose Area fill of a two-dimensional plot

Syntax

```
area(Y)
area(X,Y)
area(...,ymin)
area(...,'PropertyName',PropertyValue,...)
h = area(...)
```

Description An area plot displays elements in `Y` as one or more curves and fills the area beneath each curve. When `Y` is a matrix, the curves are stacked showing the relative contribution of each row element to the total height of the curve at each *x* interval.

`area(Y)` plots the vector `Y` or the sum of each column in matrix `Y`. The *x*-axis automatically scales depending on `length(Y)` when `Y` is a vector and on `size(Y,1)`when `Y` is a matrix.

`area(X,Y)` plots `Y` at the corresponding values of `X`. If `X` is a vector, `length(X)` must equal `length(Y)` and `X` must be monotonic. If `X` is a matrix, `size(X)` must equal `size(Y)` and each column in `X` must be monotonic. To make a vector or matrix monotonic, use `sort`.

`area(...,ymin)` specifies the lower limit in the *y* direction for the area fill. The default `ymin` is `0`.

`area(...,'`*PropertyName*`',PropertyValue,...)` specifies property name and property value pairs for the patch graphics object created by `area`.

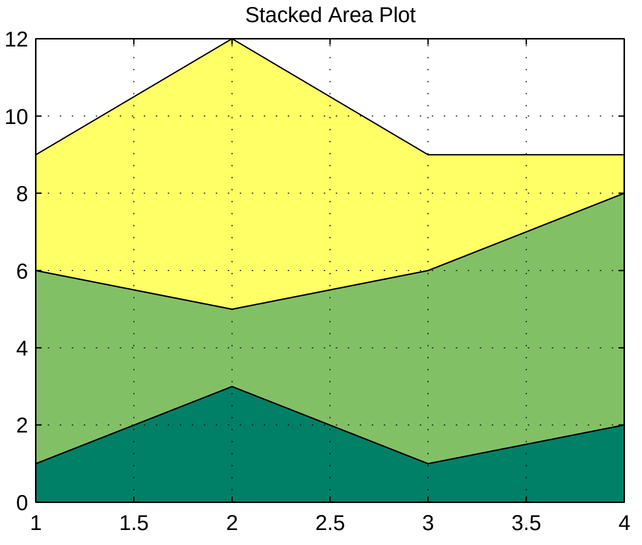
`h = area(...)` returns handles of patch graphics objects. `area` creates one patch object per column in `Y`.

Remarks `area` creates one curve from all elements in a vector or one curve per column in a matrix. The colors of the curves are selected from equally spaced intervals throughout the entire range of the colormap.

Examples

Plot the values in Y as a stacked area plot.

```
Y = [  1, 5, 3;
       3, 2, 7;
       1, 5, 3;
       2, 6, 1];
area(Y)
grid on
colormap summer
set(gca,'Layer','top')
title 'Stacked Area Plot'
```



See Also

plot

Purpose Create axes graphics object

Syntax

```
axes
axes('PropertyName',PropertyValue,...)
axes(h)
h = axes(...)
```

Description `axes` is the low-level function for creating axes graphics objects.

`axes` creates an axes graphics object in the current figure using default property values.

`axes('`*`PropertyName`*`',PropertyValue,...)` creates an axes object having the specified property values. MATLAB uses default values for any properties that you do not explicitly define as arguments.

`axes(h)` makes existing axes `h` the current axes. It also makes `h` the first axes listed in the figure's `Children` property and sets the figure's `CurrentAxes` property to `h`. The current axes is the target for functions that draw image, line, patch, surface, and text graphics objects.

`h = axes(...)` returns the handle of the created axes object.

Remarks MATLAB automatically creates an axes, if one does not already exist, when you issue a command that draws image, light, line, patch, surface, or text graphics objects.

The `axes` function accepts property name/property value pairs, structure arrays, and cell arrays as input arguments (see the `set` and `get` commands for examples of how to specify these data types). These properties, which control various aspects of the axes object, are described in the "Axes Properties" section.

Use the `set` function to modify the properties of an existing axes or the `get` function to query the current values of axes properties. Use the `gca` command to obtain the handle of the current axes.

The `axis` (not `axes`) function provides simplified access to commonly used properties that control the scaling and appearance of axes.

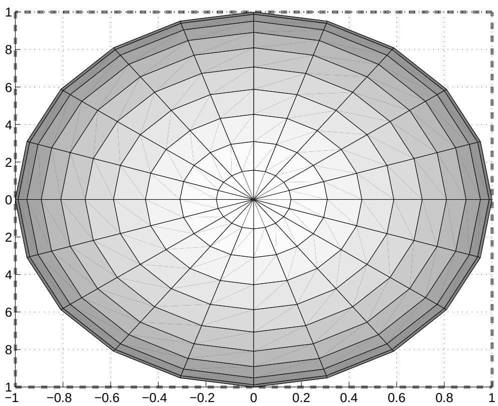
While the basic purpose of an axes object is to provide a coordinate system for plotted data, axes properties provide considerable control over the way MATLAB displays data.

Stretch-to-Fill

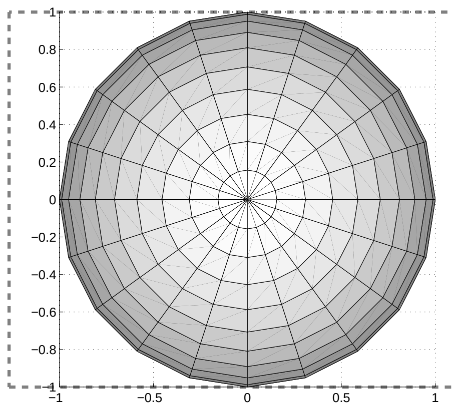
By default, MATLAB stretches the axes to fill the axes position rectangle (the rectangle defined by the last two elements in the `Position` property). This results in graphs that use the available space in the rectangle. However, some 3-D graphs (such as a sphere) appear distorted because of this stretching, and are better viewed with a specific three-dimensional aspect ratio.

Stretch-to-fill is active when the `DataAspectRatioMode`, `PlotBoxAspectRatioMode`, and `CameraViewAngleMode` are all `auto` (the default). However, stretch-to-fill is turned off when the `DataAspectRatio`, `PlotBoxAspectRatio`, or `CameraViewAngle` is user-specified, or when one or more of the corresponding modes is set to `manual` (which happens automatically when you set the corresponding property value).

This picture shows the same sphere displayed both with and without the stretch-to-fill. The dotted lines show the axes `Position` rectangle.



Stretch-to-fill active



Stretch-to-fill disabled

When stretch-to-fill is disabled, MATLAB sets the size of the axes to be as large as possible within the constraints imposed by the `Position` rectangle without introducing distortion. In the picture above, the height of the rectangle constrains the axes size.

Examples

Zooming

Zoom in using aspect ratio and limits:

```
sphere
set(gca,'DataAspectRatio',[1 1 1],...
        'PlotBoxAspectRatio',[1 1 1],'ZLim',[–0.6 0.6])
```

Zoom in and out using the `CameraViewAngle`:

```
sphere
set(gca,'CameraViewAngle',get(gca,'CameraViewAngle')–5)
set(gca,'CameraViewAngle',get(gca,'CameraViewAngle')+5)
```

Note that both examples disable MATLAB's stretch-to-fill behavior.

Positioning the Axes

The axes `Position` property enables you to define the location of the axes within the figure window. For example,

```
h = axes('Position',position_rectangle)
```

creates an axes object at the specified position within the current figure and returns a handle to it. Specify the location and size of the axes with a rectangle defined by a four-element vector,

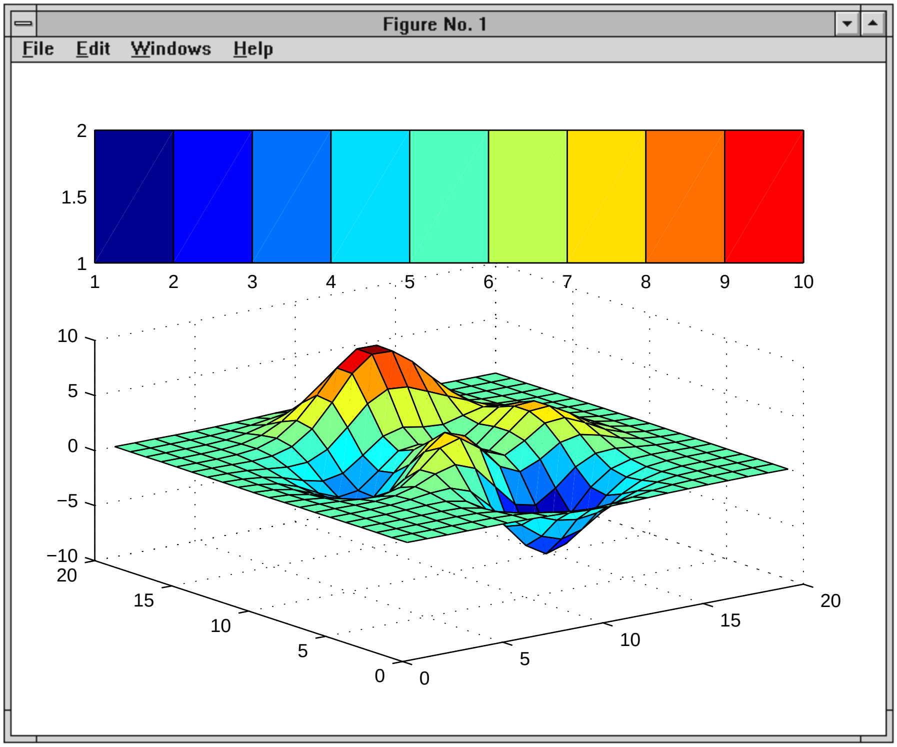
```
position_rectangle = [left, bottom, width, height];
```

The `left` and `bottom` elements of this vector define the distance from the lower-left corner of the figure to the lower-left corner of the rectangle. The `width` and `height` elements define the dimensions of the rectangle. You specify these values in units determined by the `Units` property. By default, MATLAB uses normalized units where (0,0) is the lower-left corner and (1.0,1.0) is the upper-right corner of the figure window.

You can define multiple axes in a single figure window:

```
axes('position',[.1  .1  .8  .6])
mesh(peaks(20));
axes('position',[.1  .7  .8  .2])
pcolor([1:10;1:10]);
```

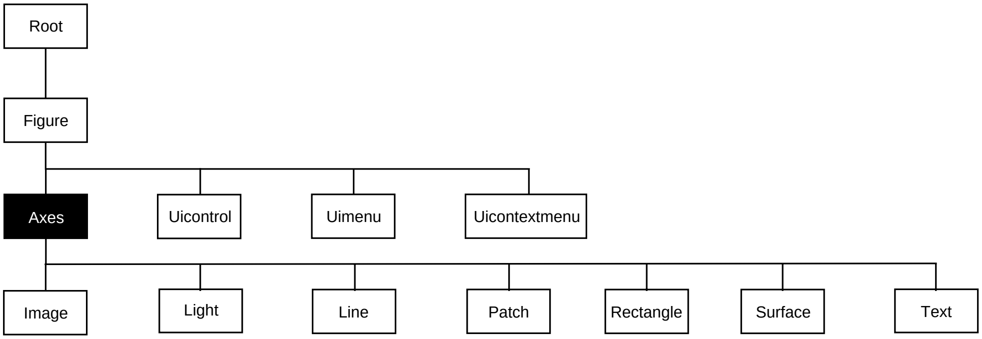
In this example, the first plot occupies the bottom two-thirds of the figure, and the second occupies the top third.



See Also

`axis`, `cla`, `clf`, `figure`, `gca`, `grid`, `subplot`, `title`, `xlabel`, `ylabel`, `zlabel`, `view`

Object Hierarchy



Setting Default Properties

You can set default axes properties on the figure and root levels:

```
set(0,'DefaultAxesPropertyName',PropertyValue,...)
set(gcf,'DefaultAxesPropertyName',PropertyValue,...)
```

where *PropertyName* is the name of the axes property and `PropertyValue` is the value you are specifying. Use `set` and `get` to access axes properties.

Property List

The following table lists all axes properties and provides a brief description of each. The property name links take you an expanded description of the properties.

Property Name	Property Description	Property Value
Controlling Style and Appearance		
`Box`	Toggle axes plot box on and off	Values: `on`, `off` Default: `off`
`Clipping`	This property has no effect; axes are always clipped to the figure window	
`GridLineStyle`	Line style used to draw axes grid lines	Values: `–`, `––`, `:`, `-.`, `none` Default: `:` (dotted line)
`Layer`	Draw axes above or below graphs	Values: `bottom`, `top` Default: `bottom`

Property Name	Property Description	Property Value
LineStyleOrder	Sequence of line styles used for multiline plots	Values: LineSpec Default: – (solid line for)
LineWidth	Width of axis lines, in points (1/72" per point)	Values: number of points Default: 0.5 points
SelectionHighlight	Highlight axes when selected (Selected property set to on)	Values: on, off Default: on
TickDir	Direction of axis tick marks	Values: in, out Default: in (2-D), out (3-D)
TickDirMode	Use MATLAB or user-specified tick mark direction	Values: auto, manual Default: auto
TickLength	Length of tick marks normalized to axis line length, specified as two-element vector	Values: [2-D 3-D] Default: [0.01 0.025}
Visible	Make axes visible or invisible	Values: on, off Default: on
XGrid, YGrid, ZGrid	Toggle grid lines on and off in respective axis	Values: on, off Default: off
General Information About the Axes		
Children	Handles of the images, lights, lines, patches, surfaces, and text objects displayed in the axes	Values: vector of handles
CurrentPoint	Location of last mouse button click defined in the axes data units	Values: a 2-by-3 matrix
HitTest	Specify whether axes can become the current object (see figure CurrentObject property)	Values: on, off Default: on
Parent	Handle of the figure window containing the axes	Values: scalar figure handle

Property Name	Property Description	Property Value
Position	Location and size of axes within the figure	Values: [left bottom width height] Default: [0.1300 0.1100 0.7750 0.8150] in normalized Units
Selected	Indicate whether axes is in a “selected” state	Values: on, off Default: on
Tag	User-specified label	Values: any string Default: '' (empty string)
Type	The type of graphics object (read only)	Value: the string 'axes'
Units	Units used to interpret the Position property	Values: inches, centimeters, characters, normalized, points, pixels Default: normalized
UserData	User-specified data	Values: any matrix Default: [] (empty matrix)
Selecting Fonts and Labels		
FontAngle	Select italic or normal font	Values: normal, italic, oblique Default: normal
FontName	Font family name (e.g., Helvetica, Courier)	Values: a font supported by your system or the string FixedWidth Default: Typically Helvetica
FontSize	Size of the font used for title and labels	Values: an integer in FontUnits Default: 10

Property Name	Property Description	Property Value
FontUnits	Units used to interpret the FontSize property	Values: points, normalized, inches, centimeters, pixels Default: points
FontWeight	Select bold or normal font	Values: normal, bold, light, demi Default: normal
Title	Handle of the title text object	Values: any valid text object handle
XLabel, YLabel, ZLabel	Handles of the respective axis label text objects	Values: any valid text object handle
XTickLabel, YTickLabel, ZTickLabel	Specify tick mark labels for the respective axis	Values: matrix of strings Defaults: numeric values selected automatically by MATLAB
XTickLabelMode, YTickLabelMode, ZTickLabelMode	Use MATLAB or user-specified tick mark labels	Values: auto, manual Default: auto
Controlling Axis Scaling		
XAxisLocation	Specify the location of the *x*-axis	Values: top, bottom Default: bottom
YAxisLocation	Specify the location of the *y*-axis	Values: right left Default: left
XDir, YDir, ZDir	Specify the direction of increasing values for the respective axes	Values: normal, reverse Default: normal
XLim, YLim, ZLim	Specify the limits to the respective axes	Values: [min max] Default: min and max determined automatically by MATLAB

Property Name	Property Description	Property Value
XLimMode, YLimMode, ZLimMode	Use MATLAB or user-specified values for the respective axis limits	Values: auto, manual Default: auto
XScale, YScale, ZScale	Select linear or logarithmic scaling of the respective axis	Values: linear, log Default: linear (changed by plotting commands that create nonlinear plots)
XTick, YTick, ZTick	Specify the location of the axis ticks marks	Values: a vector of data values locating tick marks Default: MATLAB automatically determines tick mark placement
XTickMode, YTickMode, ZTickMode	Use MATLAB or user-specified values for the respective tick mark locations	Values: auto, manual Default: auto
Controlling the View		
CameraPosition	Specify the position of point from which you view the scene	Values: [x,y,z] axes coordinates Default: automatically determined by MATLAB
CameraPositionMode	Use MATLAB or user-specified camera position	Values: auto, manual Default: auto
CameraTarget	Center of view pointed to by camera	Values: [x,y,z] axes coordinates Default: automatically determined by MATLAB
CameraTargetMode	Use MATLAB or user-specified camera target	Values: auto, manual Default: auto

Property Name	Property Description	Property Value
CameraUpVector	Direction that is oriented up	Values: [x,y,z] axes coordinates Default: automatically determined by MATLAB
CameraUpVectorMode	Use MATLAB or user-specified camera up vector	Values: auto, manual Default: auto
CameraViewAngle	Camera field of view	Values: angle in degrees between 0 and 180 Default: automatically determined by MATLAB
CameraViewAngleMode	Use MATLAB or user-specified camera view angle	Values: auto, manual Default: auto
Projection	Select type of projection	Values: orthographic, perspective Default: orthographic
Controlling the Axes Aspect Ratio		
DataAspectRatio	Relative scaling of data units	Values: three relative values [dx dy dz] Default: automatically determined by MATLAB
DataAspectRatioMode	Use MATLAB or user-specified data aspect ratio	Values: auto, manual Default: auto
PlotBoxAspectRatio	Relative scaling of axes plot box	Values: three relative values [dx dy dz] Default: automatically determined by MATLAB
PlotBoxAspectRatioMode	Use MATLAB or user-specified plot box aspect ratio	Values: auto, manual Default: auto
Controlling Callback Routine Execution		

Property Name	Property Description	Property Value
BusyAction	Specify how to handle events that interrupt execution callback routines	Values: cancel, queue Default: queue
ButtonDownFcn	Define a callback routine that executes when a button is pressed over the axes	Values: string Default: an empty string
CreateFcn	Define a callback routine that executes when an axes is created	Values: string Default: an empty string
DeleteFcn	Define a callback routine that executes when an axes is created	Values: string Default: an empty string
Interruptible	Control whether an executing callback routine can be interrupted	Values: on, off Default: on
UIContextMenu	Associate a context menu with the axes	Values: handle of a Uicontextmenu
Specifying the Rendering Mode		
DrawMode	Specify the rendering method to use with the Painters renderer	Values: normal, fast Default: normal
Targeting Axes for Graphics Display		
HandleVisibility	Control access to a specific axes' handle	Values: on, callback, off Default: on
NextPlot	Determine the eligibility of the axes for displaying graphics	Values: add, replace, replacechildren Default: replace
Properties that Specify Color		
AmbientLightColor	Color of the background light in a scene	Values: ColorSpec Default: [1 1 1]
CLim	Control how data is mapped to colormap	Values: [cmin cmax] Default: automatically determined by MATLAB

Property Name	Property Description	Property Value
CLimMode	Use MATLAB or user-specified values for CLim	Values: auto, manual Default: auto
Color	Color of the axes background	Values: none, ColorSpec Default: none
ColorOrder	Line colors used for multiline plots	Values: m-by-3 matrix of RGB values Default: depends on color scheme used
XColor, YColor, ZColor	Colors of the axis lines and tick marks	Values: ColorSpec Default: depends on current color scheme

Axes Properties

This section lists property names along with the types of values each accepts. Curly braces { } enclose default values.

AmbientLightColor `ColorSpec`

The background light in a scene. Ambient light is a directionless light that shines uniformly on all objects in the axes. However, if there are no visible light objects in the axes, MATLAB does not use `AmbientLightColor`. If there are light objects in the axes, the `AmbientLightColor` is added to the other light sources.

AspectRatio (Obsolete)

This property produces a warning message when queried or changed. It has been superseded by the `DataAspectRatio[Mode]` and `PlotBoxAspectRatio[Mode]` properties.

Box `on | {off}`

Axes box mode. This property specifies whether to enclose the axes extent in a box for 2-D views or a cube for 3-D views. The default is to not display the box.

BusyAction `cancel | {queue}`

Callback routine interruption. The `BusyAction` property enables you to control how MATLAB handles events that potentially interrupt executing callback routines. If there is a callback routine executing, subsequently invoked callback routines always attempt to interrupt it. If the `Interruptible` property of the object whose callback is executing is set to `on` (the default), then interruption occurs at the next point where the event queue is processed. If the `Interruptible` property is `off`, the `BusyAction` property (of the object owning the executing callback) determines how MATLAB handles the event. The choices are:

- `cancel` – discard the event that attempted to execute a second callback routine.
- `queue` – queue the event that attempted to execute a second callback routine until the current callback finishes.

ButtonDownFcn string

Button press callback routine. A callback routine that executes whenever you press a mouse button while the pointer is within the axes, but not over another graphics object displayed in the axes. For 3-D views, the active area is defined by a rectangle that encloses the axes.

Define this routine as a string that is a valid MATLAB expression or the name of an M-file. The expression executes in the MATLAB workspace.

CameraPosition `[x, y, z]` axes coordinates

The location of the camera. This property defines the position from which the camera views the scene. Specify the point in axes coordinates.

If you fix `CameraViewAngle`, you can zoom in and out on the scene by changing the `CameraPosition`, moving the camera closer to the `CameraTarget` to zoom in and farther away from the `CameraTarget` to zoom out. As you change the `CameraPosition`, the amount of perspective also changes, if `Projection` is `perspective`. You can also zoom by changing the `CameraViewAngle`; however, this does not change the amount of perspective in the scene.

CameraPositionMode `{auto} | manual`

Auto or manual CameraPosition. When set to `auto`, MATLAB automatically calculates the `CameraPosition` such that the camera lies a fixed distance from the `CameraTarget` along the azimuth and elevation specified by `view`. Setting a value for `CameraPosition` sets this property to `manual`.

CameraTarget `[x, y, z]` axes coordinates

Camera aiming point. This property specifies the location in the axes that the camera points to. The `CameraTarget` and the `CameraPosition` define the vector (the view axis) along which the camera looks.

CameraTargetMode `{auto} | manual`

Auto or manual CameraTarget placement. When this property is `auto`, MATLAB automatically positions the `CameraTarget` at the centroid of the axes plotbox. Specifying a value for `CameraTarget` sets this property to `manual`.

CameraUpVector `[x, y, z]` axes coordinates

Camera rotation. This property specifies the rotation of the camera around the viewing axis defined by the `CameraTarget` and the `CameraPosition` properties. Specify `CameraUpVector` as a three-element array containing the *x*, *y*, and *z* components of the vector. For example, `[0 1 0]` specifies the positive *y*-axis as the up direction.

The default CameraUpVector is [0 0 1], which defines the positive *z*-axis as the up direction.

CameraUpVectorMode {auto} | manual

Default or user-specified up vector. When CameraUpVectorMode is auto, MATLAB uses a value of [0 0 1] (positive *z*-direction is up) for 3-D views and [0 1 0] (positive *y*-direction is up) for 2-D views. Setting a value for CameraUpVector sets this property to manual.

CameraViewAngle scalar greater than 0 and less than or equal to 180 (angle in degrees)

The field of view. This property determines the camera field of view. Changing this value affects the size of graphics objects displayed in the axes, but does not affect the degree of perspective distortion. The greater the angle, the larger the field of view, and the smaller objects appear in the scene.

CameraViewAngleMode{auto} | manual

Auto or manual CameraViewAngle. When in auto mode, MATLAB sets CameraViewAngle to the minimum angle that captures the entire scene (up to 180°).

The following table summarizes MATLAB's automatic camera behavior.

CameraView Angle	Camera Target	Camera Position	Behavior
auto	auto	auto	CameraTarget is set to plot box centroid, CameraViewAngle is set to capture entire scene, CameraPosition is set along the view axis.
auto	auto	manual	CameraTarget is set to plot box centroid, CameraViewAngle is set to capture entire scene.
auto	manual	auto	CameraViewAngle is set to capture entire scene, CameraPosition is set along the view axis.
auto	manual	manual	CameraViewAngle is set to capture entire scene.
manual	auto	auto	CameraTarget is set to plot box centroid, CameraPosition is set along the view axis.

CameraView Angle	Camera Target	Camera Position	Behavior
`manual`	`auto`	`manual`	`CameraTarget` is set to plot box centroid
`manual`	`manual`	`auto`	`CameraPosition` is set along the view axis.
`manual`	`manual`	`manual`	All Camera properties are user-specified.

`Children` vector of graphics object handles

Children of the axes. A vector containing the handles of all graphics objects rendered within the axes (whether visible or not). The graphics objects that can be children of axes are images, lights, lines, patches, surfaces, and text.

The text objects used to label the *x*-, *y*-, and *z*-axes are also children of axes, but their `HandleVisibility` properties are set to `callback`. This means their handles do not show up in the axes `Children` property unless you set the Root `ShowHiddenHandles` property to `on`.

`CLim` `[cmin, cmax]`

Color axis limits. A two-element vector that determines how MATLAB maps the `CData` values of surface and patch objects to the figure's colormap. `cmin` is the value of the data mapped to the first color in the colormap, and `cmax` is the value of the data mapped to the last color in the colormap. Data values in between are linearly interpolated across the colormap, while data values outside are clamped to either the first or last colormap color, whichever is closest.

When `CLimMode` is `auto` (the default), MATLAB assigns `cmin` the minimum data value and `cmax` the maximum data value in the graphics object's `CData`. This maps `CData` elements with minimum data value to the first colormap entry and with maximum data value to the last colormap entry.

If the axes contains multiple graphics objects, MATLAB sets `CLim` to span the range of all objects' `CData`.

`CLimMode` `{auto} | manual`

Color axis limits mode. In `auto` mode, MATLAB sets the `CLim` property to span the `CData` limits of the graphics objects displayed in the axes. If `CLimMode` is `manual`, MATLAB does not change the value of `CLim` when the `CData` limits of axes children change. Setting the `CLim` property sets this property to `manual`.

Clipping {on} | off

This property has no effect on axes.

Color {none} **|** ColorSpec

Color of the axes back planes. Setting this property to none means the axes is transparent and the figure color shows through. A ColorSpec is a three-element RGB vector or one of MATLAB's predefined names. Note that while the default value is none, the matlabrc.m file may set the axes color to a specific color.

ColorOrder m-by-3 matrix of RGB values

Colors to use for multiline plots. ColorOrder is an *m*-by-3 matrix of RGB values that define the colors used by the plot and plot3 functions to color each line plotted. If you do not specify a line color with plot and plot3, these functions cycle through the ColorOrder to obtain the color for each line plotted. To obtain the current ColorOrder, which may be set during startup, get the property value:

```
get(gca,'ColorOrder')
```

Note that if the axes NextPlot property is set to replace (the default), high-level functions like plot reset the ColorOrder property before determining the colors to use. If you want MATLAB to use a ColorOrder that is different from the default, set NextPlot to replacedata. You can also specify your own default ColorOrder.

CreateFcn string

Callback routine executed during object creation. This property defines a callback routine that executes when MATLAB creates an axes object. You must define this property as a default value for axes. For example, the statement,

```
set(0,'DefaultAxesCreateFcn','set(gca,''Color'',''b'')')
```

defines a default value on the Root level that sets the current axes' background color to blue whenever you (or MATLAB) create an axes. MATLAB executes this routine after setting all properties for the axes. Setting this property on an existing axes object has no effect.

The handle of the object whose CreateFcn is being executed is accessible only through the Root CallbackObject property, which can be queried using gcbo.

`CurrentPoint` 2-by-3 matrix

Location of last button click, in axes data units. A 2-by-3 matrix containing the coordinates of two points defined by the location of the pointer. These two points lie on the line that is perpendicular to the plane of the screen and passes through the pointer. The 3-D coordinates are the points, in the axes coordinate system, where this line intersects the front and back surfaces of the axes volume (which is defined by the axes *x*, *y*, and *z* limits).

The returned matrix is of the form:

$$\begin{bmatrix} x_{back} & y_{back} & z_{back} \\ x_{front} & y_{front} & z_{front} \end{bmatrix}$$

MATLAB updates the `CurrentPoint` property whenever a button-click event occurs. The pointer does not have to be within the axes, or even the figure window; MATLAB returns the coordinates with respect to the requested axes regardless of the pointer location.

`DataAspectRatio` `[dx dy dz]`

Relative scaling of data units. A three-element vector controlling the relative scaling of data units in the *x*, *y*, and *z* directions. For example, setting this property t o `[1 2 1]` causes the length of one unit of data in the *x* direction to be the same length as two units of data in the *y* direction and one unit of data in the *z* direction.

Note that the `DataAspectRatio` property interacts with the `PlotBoxAspectRatio`, `XLimMode`, `YLimMode`, and `ZLimMode` properties to control how MATLAB scales the *x*-, *y*-, and *z*-axis. Setting the `DataAspectRatio` will disable the stretch-to-fill behavior, if `DataAspectRatioMode`, `PlotBoxAspectRatioMode`, and `CameraViewAngleMode` are all `auto`. The following table describes the interaction between properties when stretch-to-fill behavior is disabled.

X-, Y-, Z-Limits	DataAspect Ratio	PlotBox AspectRatio	Behavior
auto	auto	auto	Limits chosen to span data range in all dimensions.
auto	auto	manual	Limits chosen to span data range in all dimensions. DataAspectRatio is modified to achieve the requested PlotBoxAspectRatio within the limits selected by MATLAB.
auto	manual	auto	Limits chosen to span data range in all dimensions. PlotBoxAspectRatio is modified to achieve the requested DataAspectRatio within the limits selected by MATLAB.
auto	manual	manual	Limits chosen to completely fit and center the plot within the requested PlotBoxAspectRatio given the requested DataAspectRatio (this may produce empty space around 2 of the 3 dimensions).
manual	auto	auto	Limits are honored. The DataAspectRatio and PlotBoxAspectRatio are modified as necessary.
manual	auto	manual	Limits and PlotBoxAspectRatio are honored. The DataAspectRatio is modified as necessary.
manual	manual	auto	Limits and DataAspectRatio are honored. The PlotBoxAspectRatio is modified as necessary.
1 manual 2 auto	manual	manual	The 2 automatic limits are selected to honor the specified aspect ratios and limit. See “Examples”
2 or 3 manual	manual	manual	Limits and DataAspectRatio are honored; the PlotBoxAspectRatio is ignored.

DataAspectRatioMode{auto} | manual

User or MATLAB controlled data scaling. This property controls whether the values of the DataAspectRatio property are user defined or selected automatically by MATLAB. Setting values for the DataAspectRatio property automatically sets this property to manual. Changing DataAspectRatioMode to manual disables the stretch-to-fill behavior, if DataAspectRatioMode, PlotBoxAspectRatioMode, and CameraViewAngleMode are all auto.

DeleteFcn string

Delete axes callback routine. A callback routine that executes when the axes object is deleted (e.g., when you issue a delete or a close command). MATLAB executes the routine before destroying the object's properties so the callback routine can query these values.

The handle of the object whose DeleteFcn is being executed is accessible only through the Root CallbackObject property, which can be queried using gcbo.

DrawMode {normal} | fast

Rendering method. This property controls the method MATLAB uses to render graphics objects displayed in the axes, when the figure Renderer property is painters.

- normal mode draws objects in back to front ordering based on the current view in order to handle hidden surface elimination and object intersections.
- fast mode draws objects in the order in which you specify the drawing commands, without considering the relationships of the objects in three dimensions. This results in faster rendering because it requires no sorting of objects according to location in the view, but may produce undesirable results because it bypasses the hidden surface elimination and object intersection handling provided by normal DrawMode.

When the figure Renderer is zbuffer, DrawMode is ignored, and hidden surface elimination and object intersection handling are always provided.

FontAngle {normal} | italic | oblique

Select italic or normal font. This property selects the character slant for axes text. normal specifies a nonitalic font. italic and oblique specify italic font.

FontName A name such as Courier or the string FixedWidth

Font family name. The font family name specifying the font to use for axes labels. To display and print properly, FontName must be a font that your system supports. Note that the *x*-, *y*-, and *z*-axis labels do not display in a new font until you manually reset them (by setting the XLabel, YLabel, and ZLabel properties or by using the `xlabel`, `ylabel`, or `zlabel` command). Tick mark labels change immediately.

Specifying a Fixed-Width Font

If you want an axes to use a fixed-width font that looks good in any locale, you should set `FontName` to the string `FixedWidth`:

```
set(axes_handle,'FontName','FixedWidth')
```

This eliminates the need to hardcode the name of a fixed-width font, which may not display text properly on systems that do not use ASCII character encoding (such as in Japan where multibyte character sets are used). A properly written MATLAB application that needs to use a fixed-width font should set `FontName` to `FixedWidth` (note that this string is case sensitive) and rely on `FixedWidthFontName` to be set correctly in the end-user's environment.

End users can adapt a MATLAB application to different locales or personal environments by setting the root `FixedWidthFontName` property to the appropriate value for that locale from `startup.m`.

Note that setting the root `FixedWidthFontName` property causes an immediate update of the display to use the new font.

`FontSize` Font size specified in `FontUnits`

Font size. An integer specifying the font size to use for axes labels and titles, in units determined by the `FontUnits` property. The default point size is 12. The *x*-, *y*-, and *z*-axis text labels do not display in a new font size until you manually reset them (by setting the `XLabel`, `YLabel`, or `ZLabel` properties or by using the `xlabel`, `ylabel`, or `zlabel` command). Tick mark labels change immediately.

`FontUnits` `{points} | normalized | inches | centimeters | pixels`

Units used to interpret the `FontSize` *property*. When set to `normalized`, MATLAB interprets the value of `FontSize` as a fraction of the height of the axes. For example, a `normalized` `FontSize` of 0.1 sets the text characters to a font whose height is one tenth of the axes' height. The default units (`points`), are equal to 1/72 of an inch.

`FontWeight` `{normal} | bold | light | demi`

Select bold or normal font. The character weight for axes text. The *x*-, *y*-, and *z*-axis text labels do not display in bold until you manually reset them (by setting the XLabel, YLabel, and ZLabel properties or by using the xlabel, ylabel, or zlabel commands). Tick mark labels change immediately.

GridLineStyle – | – –| {:} | –. | none

Line style used to draw grid lines. The line style is a string consisting of a character, in quotes, specifying solid lines (–), dashed lines (– –), dotted lines(:), or dash-dot lines (–.). The default grid line style is dotted. To turn on grid lines, use the grid command.

HandleVisibility {on} | callback | off

Control access to object's handle by command-line users and GUIs. This property determines when an object's handle is visible in its parent's list of children. HandleVisibility is useful for preventing command-line users from accidentally drawing into or deleting a figure that contains only user interface devices (such as a dialog box).

Handles are always visible when HandleVisibility is on.

Setting HandleVisibility to callback causes handles to be visible from within callback routines or functions invoked by callback routines, but not from within functions invoked from the command line. This provides a means to protect GUIs from command-line users, while allowing callback routines to have complete access to object handles.

Setting HandleVisibility to off makes handles invisible at all times. This may be necessary when a callback routine invokes a function that might potentially damage the GUI (such as evaluating a user-typed string) and so temporarily hides its own handles during the execution of that function.

When a handle is not visible in its parent's list of children, it cannot be returned by functions that obtain handles by searching the object hierarchy or querying handle properties. This includes get, findobj, gca, gcf, gco, newplot, cla, clf, and close.

When a handle's visibility is restricted using callback or off, the object's handle does not appear in its parent's Children property, figures do not appear in the Root's Currentfigure property, objects do not appear in the Root's CallbackObject property or in the figure's CurrentObject property, and axes do not appear in their parent's Currentaxes property.

You can set the Root `ShowHiddenHandles` property to `on` to make all handles visible, regardless of their `HandleVisibility` settings (this does not affect the values of the `HandleVisibility` properties).

Handles that are hidden are still valid. If you know an object's handle, you can `set` and `get` its properties, and pass it to any function that operates on handles.

HitTest `{on} | off`

Selectable by mouse click. `HitTest` determines if the axes can become the current object (as returned by the `gco` command and the figure `CurrentObject` property) as a result of a mouse click on the axes. If `HiTest` is `off`, clicking on the axes selects the object below it (which is usually the figure containing it).

Interruptible `{on} | off`

Callback routine interruption mode. The `Interruptible` property controls whether an axes callback routine can be interrupted by subsequently invoked callback routines. Only callback routines defined for the `ButtonDownFcn` are affected by the `Interruptible` property. MATLAB checks for events that can interrupt a callback routine only when it encounters a `drawnow`, `figure`, `getframe`, or `pause` command in the routine. See the `BusyAction` property for related information.

Setting `Interruptible` to `on` allows any graphics object's callback routine to interrupt callback routines originating from an axes property. Note that MATLAB does not save the state of variables or the display (e.g., the handle returned by the `gca` or `gcf` command) when an interruption occurs.

Layer `{bottom} | top`

Draw axis lines below or above graphics objects. This property determines if axis lines and tick marks draw on top or below axes children objects for any 2-D view (i.e., when you are looking along the x-, y-, or z-axis). This is useful for placing grid lines and tick marks on top of images.

LineStyleOrder `LineSpec`

Order of line styles and markers used in a plot. This property specifies which line styles and markers to use and in what order when creating multiple-line plots. For example,

```
set(gca,'LineStyleOrder', '–*|:|o')
```

sets `LineStyleOrder` to solid line with asterisk marker, dotted line, and hollow circle marker. The default is (–), which specifies a solid line for all data plotted. Alternatively, you can create a cell array of character strings to define the line styles:

```
set(gca,'LineStyleOrder',{'-*',':','o'})
```

MATLAB supports four line styles, which you can specify any number of times in any order. MATLAB cycles through the line styles only after using all colors defined by the `ColorOrder` property. For example, the first eight lines plotted use the different colors defined by `ColorOrder` with the first line style. MATLAB then cycles through the colors again, using the second line style specified, and so on.

You can also specify line style and color directly with the `plot` and `plot3` functions or by altering the properties of the line objects.

Note that, if the axes `NextPlot` property is set to `replace` (the default), high-level functions like `plot` reset the `LineStyleOrder` property before determining the line style to use. If you want MATLAB to use a `LineStyleOrder` that is different from the default, set `NextPlot` to `replacedata`. You can also specify your own default `LineStyleOrder`.

LineWidth linewidth in points

Width of axis lines. This property specifies the width, in points, of the *x*-, *y*-, and *z*-axis lines. The default line width is 0.5 points (1 point = $^1/_{72}$ inch).

NextPlot `add | {replace} | replacechildren`

Where to draw the next plot. This property determines how high-level plotting functions draw into an existing axes.

- `add` — use the existing axes to draw graphics objects.
- `replace` — reset all axes properties, except `Position`, to their defaults and delete all axes children before displaying graphics (equivalent to `cla reset`).
- `replacechildren` — remove all child objects, but do not reset axes properties (equivalent to `cla`).

The `newplot` function simplifies the use of the `NextPlot` property and is used by M-file functions that draw graphs using only low-level object creation routines. See the M-file `pcolor.m` for an example. Note that figure graphics objects also have a `NextPlot` property.

Parent figure handle

Axes parent. The handle of the axes' parent object. The parent of an axes object is the figure in which it is displayed. The utility function `gcf` returns the handle of the current axes' `Parent`. You can reparent axes to other figure objects.

PlotBoxAspectRatio `[px py pz]`

Relative scaling of axes plotbox. A three-element vector controlling the relative scaling of the plot box in the *x*-, *y*-, and *z*-directions. The plot box is a box enclosing the axes data region as defined by the *x*-, *y*-, and *z*-axis limits.

Note that the `PlotBoxAspectRatio` property interacts with the `DataAspectRatio`, `XLimMode`, `YLimMode`, and `ZLimMode` properties to control the way graphics objects are displayed in the axes. Setting the `PlotBoxAspectRatio` disables stretch-to-fill behavior, if `DataAspectRatioMode`, `PlotBoxAspectRatioMode`, and `CameraViewAngleMode` are all `auto`.

PlotBoxAspectRatioMode `{auto} | manual`

User or MATLAB controlled axis scaling. This property controls whether the values of the `PlotBoxAspectRatio` property are user defined or selected automatically by MATLAB. Setting values for the `PlotBoxAspectRatio` property automatically sets this property to `manual`. Changing the `PlotBoxAspectRatioMode` to `manual` disables stretch-to-fill behavior, if `DataAspectRatioMode`, `PlotBoxAspectRatioMode`, and `CameraViewAngleMode` are all `auto`.

Position four-element vector

Position of axes. A four-element vector specifying a rectangle that locates the axes within the figure window. The vector is of the form:

```
[left bottom width height]
```

where `left` and `bottom` define the distance from the lower-left corner of the figure window to the lower-left corner of the rectangle. `width` and `height` are the dimensions of the rectangle. All measurements are in units specified by the `Units` property.

When axes stretch-to-fill behavior is enabled (when `DataAspectRatioMode`, `PlotBoxAspectRatioMode`, `CameraViewAngleMode` are all `auto`), the axes are stretched to fill the `Position` rectangle. When stretch-to-fill is disabled, the axes are made as large as possible, while obeying all other properties, without extending outside the `Position` rectangle

`Projection` `{orthographic} | perspective`

Type of projection. This property selects between two projection types:

- `orthographic` – This projection maintains the correct relative dimensions of graphics objects with regard to the distance a given point is from the viewer. Parallel lines in the data are drawn parallel on the screen.
- `perspective` – This projection incorporates foreshortening, which allows you to perceive depth in 2-D representations of 3-D objects. Perspective projection does not preserve the relative dimensions of objects; a distant line segment displays smaller than a nearer line segment of the same length. Parallel lines in the data may not appear parallel on screen.

`Selected` `on | off`

Is object selected. When you set this property to `on`, MATLAB displays selection “handles” at the corners and midpoints if the `SelectionHighlight` property is also `on` (the default). You can, for example, define the `ButtonDownFcn` callback routine to set this property to `on`, thereby indicating that the axes has been selected.

`SelectionHighlight` `{on} | off`

Objects highlight when selected. When the `Selected` property is `on`, MATLAB indicates the selected state by drawing four edge handles and four corner handles. When `SelectionHighlight` is `off`, MATLAB does not draw the handles.

`Tag` string

User-specified object label. The `Tag` property provides a means to identify graphics objects with a user-specified label. This is particularly useful when constructing interactive graphics programs that would otherwise need to define object handles as global variables or pass them as arguments between callback routines.

For example, suppose you want to direct all graphics output from an M-file to a particular axes, regardless of user actions that may have changed the current axes. To do this, identify the axes with a `Tag`:

```
axes('Tag','Special Axes')
```

Then make that axes the current axes before drawing by searching for the `Tag` with `findobj`:

```
axes(findobj('Tag','Special Axes'))
```

TickDir `in | out`

Direction of tick marks. For 2-D views, the default is to direct tick marks inward from the axis lines; 3-D views direct tick marks outward from the axis line.

TickDirMode `{auto} | manual`

Automatic tick direction control. In `auto` mode, MATLAB directs tick marks inward for 2-D views and outward for 3-D views. When you specify a setting for `TickDir`, MATLAB sets `TickDirMode` to `manual`. In `manual` mode, MATLAB does not change the specified tick direction.

TickLength `[2DLength 3DLength]`

Length of tick marks. A two-element vector specifying the length of axes tick marks. The first element is the length of tick marks used for 2-D views and the second element is the length of tick marks used for 3-D views. Specify tick mark lengths in units normalized relative to the longest of the visible X-, Y-, or Z-axis annotation lines.

Title handle of text object

Axes title. The handle of the text object that is used for the axes title. You can use this handle to change the properties of the title text or you can set `Title` to the handle of an existing text object. For example, the following statement changes the color of the current title to red:

```
set(get(gca,'Title'),'Color','r')
```

To create a new title, set this property to the handle of the text object you want to use:

```
set(gca,'Title',text('String','New Title','Color','r'))
```

However, it is generally simpler to use the `title` command to create or replace an axes title:

```
title('New Title','Color','r')
```

Type string (read only)

Type of graphics object. This property contains a string that identifies the class of graphics object. For axes objects, `Type` is always set to '`axes`'.

UIContextMenu handle of a uicontextmenu object

Associate a context menu with the axes. Assign this property the handle of a Uicontextmenu object created in the axes' parent figure. Use the `uicontextmenu` function to create the context menu. MATLAB displays the context menu whenever you right-click over the axes.

Units `inches` **|** `centimeters` **|** `{normalized}` **|** `points` **|** `pixels` | `characters`

Position units. The units used to interpret the `Position` property. All units are measured from the lower-left corner of the figure window.

- `normalized` units map the lower-left corner of the figure window to (0,0) and the upper-right corner to (1.0, 1.0).
- `inches`, `centimeters`, and `points` are absolute units (one point equals $^1/_{72}$ of an inch).
- `Character` units are defined by characters from the default system font; the width of one character is the width of the letter x, the height of one character is the distance between the baselines of two lines of text.

UserData matrix

User specified data. This property can be any data you want to associate with the axes object. The axes does not use this property, but you can access it using the `set` and `get` functions.

View Obsolete

The functionality provided by the View property is now controlled by the axes camera properties – `CameraPosition`, `CameraTarget`, `CameraUpVector`, and `CameraViewAngle`. See the `view` command.

Visible `{on} | off`

Visibility of axes. By default, axes are visible. Setting this property to `off` prevents axis lines, tick marks, and labels from being displayed. The visible property does not affect children of axes.

XAxisLocation `top | {bottom}`

Location of x-axis tick marks and labels. This property controls where MATLAB displays the *x*-axis tick marks and labels. Setting this property to `top` moves the *x*-axis to the top of the plot from its default position at the bottom.

YAxisLocation `right | {left}`

Location of y-axis tick marks and labels. This property controls where MATLAB displays the *y*-axis tick marks and labels. Setting this property to `right` moves the *y*-axis to the right side of the plot from its default position on the left side. See the `plotyy` function for a simple way to use two y-axes.

Properties That Control the X-, Y-, or Z-Axis

XColor, YColor, ZColor `ColorSpec`

Color of axis lines. A three-element vector specifying an RGB triple, or a predefined MATLAB color string. This property determines the color of the axis lines, tick marks, tick mark labels, and the axis grid lines of the respective *x*-, *y*-, and *z*-axis. The default axis color is white. See `ColorSpec` for details on specifying colors.

XDir, **YDir**, **ZDir** `{normal} | reverse`

Direction of increasing values. A mode controlling the direction of increasing axis values. axes form a right-hand coordinate system. By default:

- *x*-axis values increase from left to right. To reverse the direction of increasing *x* values, set this property to `reverse`.
- *y*-axis values increase from bottom to top (2-D view) or front to back (3-D view). To reverse the direction of increasing *y* values, set this property to `reverse`.
- *z*-axis values increase pointing out of the screen (2-D view) or from bottom to top (3-D view). To reverse the direction of increasing *z* values, set this property to `reverse`.

XGrid, **YGrid**, **ZGrid** `on | {off}`

Axis gridline mode. When you set any of these properties to `on`, MATLAB draws grid lines perpendicular to the respective axis (i.e., along lines of constant *x*, *y*, or *z* values). Use the `grid` command to set all three properties `on` or `off` at once.

XLabel, **YLabel**, **ZLabel** handle of text object

Axis labels. The handle of the text object used to label the *x*, *y*, or *z*-axis, respectively. To assign values to any of these properties, you must obtain the handle to the text string you want to use as a label. This statement defines a text object and assigns its handle to the `XLabel` property:

```
set(gca,'Xlabel',text('String','axis label'))
```

MATLAB places the string `'axis label'` appropriately for an *x*-axis label. Any text object whose handle you specify as an `XLabel`, `YLabel`, or `ZLabel` property is moved to the appropriate location for the respective label.

Alternatively, you can use the `xlabel`, `ylabel`, and `zlabel` functions, which generally provide a simpler means to label axis lines.

XLim, **YLim**, **ZLim** `[minimum maximum]`

Axis limits. A two-element vector specifying the minimum and maximum values of the respective axis.

Changing these properties affects the scale of the *x*-, *y*-, or z-dimension as well as the placement of labels and tick marks on the axis. The default values for these properties are [0 1].

XLimMode, **YLimMode**, **ZLimMode** `{auto} | manual`

MATLAB or user-controlled limits. The axis limits mode determines whether MATLAB calculates axis limits based on the data plotted (i.e., the `XData`, `YData`, or `ZData` of the axes children) or uses the values explicitly set with the `XLim`, `YLim`, or `ZLim` property, in which case, the respective limits mode is set to `manual`.

XScale, YScale, ZScale `{linear} | log`

Axis scaling. Linear or logarithmic scaling for the respective axis. See also `loglog`, `semilogx`, and `semilogy`.

XTick, YTick, ZTick vector of data values locating tick marks

Tick spacing. A vector of *x*-, *y*-, or *z*-data values that determine the location of tick marks along the respective axis. If you do not want tick marks displayed, set the respective property to the empty vector, [ ]. These vectors must contain monotonically increasing values.

XTickLabel, YTickLabel, ZTickLabelstring

Tick labels. A matrix of strings to use as labels for tick marks along the respective axis. These labels replace the numeric labels generated by MATLAB. If you do not specify enough text labels for all the tick marks, MATLAB uses all of the labels specified, then reuses the specified labels.

For example, the statement,

```
set(gca,'XTickLabel',{'One';'Two';'Three';'Four'})
```

labels the first four tick marks on the *x*-axis and then reuses the labels until all ticks are labeled.

Labels can be specified as cell arrays of strings, padded string matrices, string vectors separated by vertical slash characters, or as numeric vectors (where each number is implicitly converted to the equivalent string using `num2str`). All of the following are equivalent:

```
set(gca,'XTickLabel',{'1';'10';'100'})
set(gca,'XTickLabel','1|10|100')
set(gca,'XTickLabel',[1;10;100])
set(gca,'XTickLabel',['1  ';'10 ';'100'])
```

Note that tick labels do not interpret TeX character sequences (however, the `Title`, `XLabel`, `YLabel`, and `ZLabel` properties do).

XTickMode, YTickMode, ZTickMode `{auto} | manual`

MATLAB or user controlled tick spacing. The axis tick modes determine whether MATLAB calculates the tick mark spacing based on the range of data for the respective axis (`auto` mode) or uses the values explicitly set for any of the `XTick`, `YTick`, and `ZTick` properties (`manual` mode). Setting values for the `XTick`, `YTick`, or `ZTick` properties sets the respective axis tick mode to `manual`.

XTickLabelMode, YTickLabelMode, ZTickLabelMode`{auto} | manual`

MATLAB or user determined tick labels. The axis tick mark labeling mode determines whether MATLAB uses numeric tick mark labels that span the range of the plotted data (`auto` mode) or uses the tick mark labels specified with the `XTickLabel`, `YTickLabel`, or `ZTickLabel` property (`manual` mode). Setting values for the `XTickLabel`, `YTickLabel`, or `ZTickLabel` property sets the respective axis tick label mode to `manual`.

Purpose Axis scaling and appearance

Syntax

```
axis([xmin xmax ymin ymax])
axis([xmin xmax ymin ymax zmin zmax])
v = axis

axis auto
axis manual
axis tight
axis fill

axis ij
axis xy

axis equal
axis image
axis square
axis vis3d
axis normal

axis off
axis on
[mode,visibility,direction] = axis('state')
```

Description `axis` manipulates commonly used axes properties. (See Algorithm section.)

`axis([xmin xmax ymin ymax])` sets the limits for the *x*- and *y*-axis of the current axes.

`axis([xmin xmax ymin ymax zmin zmax])` sets the limits for the *x*-, *y*-, and *z*-axis of the current axes.

`v = axis` returns a row vector containing scaling factors for the *x*-, *y*-, and *z*-axis. `v` has four or six components depending on whether the current axes is 2-D or 3-D, respectively. The returned values are the current axes' `XLim`, `Ylim`, and `ZLim` properties.

`axis auto` sets MATLAB to its default behavior of computing the current axes' limits automatically, based on the minimum and maximum values of *x*, *y*, and *z* data. You can restrict this automatic behavior to a specific axis. For example, `axis 'auto x'` computes only the *x*-axis limits automatically; `axis 'auto yz'` computes the *y*- and *z*-axis limits automatically.

`axis manual` and `axis(axis)` freezes the scaling at the current limits, so that if `hold` is `on`, subsequent plots use the same limits. This sets the `XLimMode`, `YLimMode`, and `ZLimMode` properties to `manual`.

`axis tight` sets the aspect ratio so that the data units are the same in every direction. This differs from `axis equal` because the plot box aspect ratio automatically adjusts.

`axis fill` sets the axis limits to the range of the data.

`axis ij` places the coordinate system origin in the upper-left corner. The *i*-axis is vertical, with values increasing from top to bottom. The *j*-axis is horizontal with values increasing from left to right.

`axis xy` draws the graph in the default Cartesian axes format with the coordinate system origin in the lower-left corner. The *x*-axis is horizontal with values increasing from left to right. The *y*-axis is vertical with values increasing from bottom to top.

`axis equal` sets the aspect ratio so that the data units are the same in every direction. The aspect ratio of the *x*-, *y*-, and *z*-axis is adjusted automatically according to the range of data units in the *x*, *y*, and *z* directions.

`axis image` is the same as `axis equal` except that the plot box fits tightly around the data.

`axis square` makes the current axes region square (or cubed when three-dimensional). MATLAB adjusts the *x*-axis, *y*-axis, and *z*-axis so that they have equal lengths and adjusts the increments between data units accordingly.

`axis vis3d` freezes aspect ratio properties to enable rotation of 3-D objects and overrides stretch-to-fill.

axis normal automatically adjusts the aspect ratio of the axes and the aspect ratio of the data units represented on the axes to fill the plot box.

axis off turns off all axis lines, tick marks, and labels.

axis on turns on all axis lines, tick marks, and labels.

[mode,visibility,direction] = axis('state') returns three strings indicating the current setting of axes properties:

Output Argument	Strings Returned
mode	'auto' | 'manual'
visibility	'on' | 'off'
direction	'xy' | 'ij'

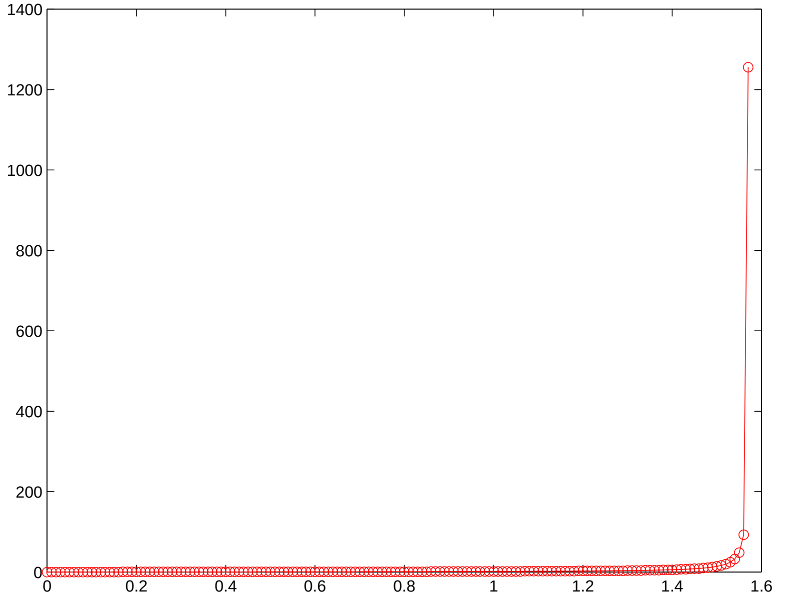
mode is auto if XLimMode, YLimMode, and ZLimMode are all set to auto. If XLimMode, YLimMode, or ZLimMode is manual, mode is manual.

Examples

The statements

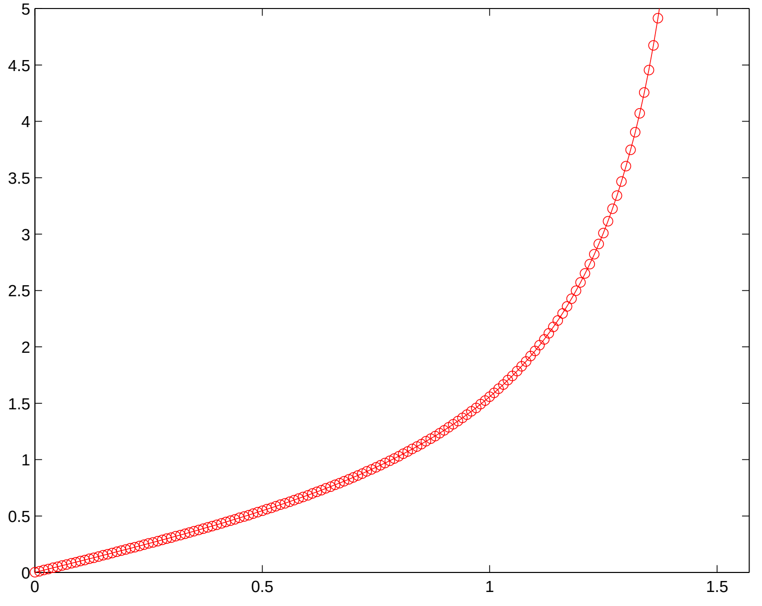
```
x = 0:.025:pi/2;
plot(x,tan(x),'-ro')
```

use the automatic scaling of the *y*-axis based on `ymax = tan(1.57)`, which is well over 1000:



The right figure shows a more satisfactory plot after typing

```
axis([0  pi/2  0  5])
```



Algorithm

When you specify minimum and maximum values for the *x*-, *y*-, and *z*-axes, `axis` sets the `XLim`, `Ylim`, and `ZLim` properties for the current axes to the respective minimum and maximum values in the argument list. Additionally, the `XLimMode`, `YLimMode`, and `ZLimMode` properties for the current axes are set to `manual`.

`axis auto` sets the current axes' `XLimMode`, `YLimMode`, and `ZLimMode` properties to `'auto'`.

`axis manual` sets the current axes' `XLimMode`, `YLimMode`, and `ZLimMode` properties to `'manual'`.

The following table shows the values of the axes properties set by `axis equal`, `axis normal`, `axis square`, and `axis image`.

Axes Property	axis equal	axis normal	axis square	axis tightequal
DataAspectRatio	[1 1 1]	not set	not set	[1 1 1]
DataAspectRatioMode	manual	auto	auto	manual
PlotBoxAspectRatio	[3 4 4]	not set	[1 1 1]	auto
PlotBoxAspectRatioMode	manual	auto	manual	auto
Stretch-to-fill	disabled	active	disabled	disabled

See Also

axes, get, grid, set, subplot

Properties of axes graphics objects

Purpose

Bar chart

Syntax

```
bar(Y)
bar(x,Y)
bar(...,width)
bar(...,'style')
bar(...,LineSpec)
[xb,yb] = bar(...)
h = bar(...)

barh(...)
[xb,yb] = barh(...)
h = barh(...)
```

Description

A bar chart displays the values in a vector or matrix as horizontal or vertical bars.

`bar(Y)` draws one bar for each element in `Y`. If `Y` is a matrix, `bar` groups the bars produced by the elements in each row. The *x*-axis scale ranges from 1 to `length(Y)` when `Y` is a vector, and `1` to `size(Y,1)`, which is the number of rows, when `Y` is a matrix.

`bar(x,Y)` draws a bar for each element in `Y` at locations specified in `x`, where `x` is a monotonically increasing vector defining the *x*-axis intervals for the vertical bars. If `Y` is a matrix, `bar` clusters the elements in the same row in `Y` at locations corresponding to an element in `x`.

`bar(...,width)` sets the relative bar width and controls the separation of bars within a group. The default `width` is `0.8`, so if you do not specify `x`, the bars within a group have a slight separation. If `width` is `1`, the bars within a group touch one another.

`bar(...,'style')` specifies the style of the bars. `'style'` is `'group'` or `'stack'`. `'group'` is the default mode of display.

- `'group'` displays *n* groups of *m* vertical bars, where *n* is the number of rows and *m* is the number of columns in `Y`. The group contains one bar per column in `Y`.
- `'stack'` displays one bar for each row in `Y`. The bar height is the sum of the elements in the row. Each bar is multi-colored, with colors corresponding to distinct elements and showing the relative contribution each row element makes to the total sum.

`bar(...,LineSpec)` displays all bars using the color specified by `LineSpec`.

`[xb,yb] = bar(...)` returns vectors that you plot using `plot(xb,yb)` or `patch(xb,yb,C)`. This gives you greater control over the appearance of a graph, for example, to incorporate a bar chart into a more elaborate `plot` statement.

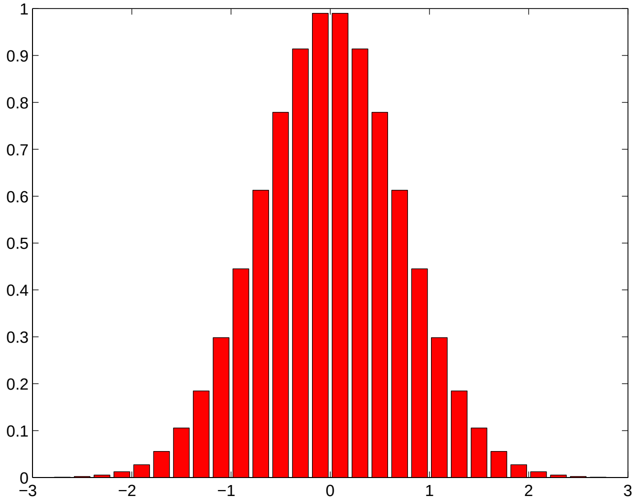
`h = bar(...)` returns a vector of handles to patch graphics objects. `bar` creates one patch graphics object per column in `Y`.

`barh(...)`, `[xb,yb] = barh(...)`, and `h = barh(...)` create horizontal bars. `Y` determines the bar length. The vector `x` is a monotonic vector defining the *y*-axis intervals for horizontal bars.

Examples

Plot a bell shaped curve:

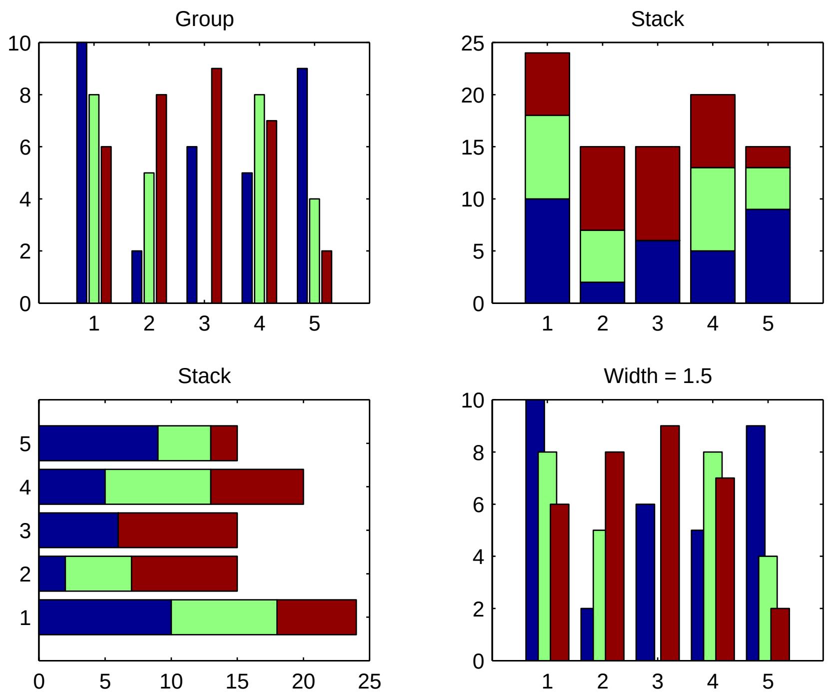
```
x = –2.9:0.2:2.9;
bar(x,exp(–x.*x))
colormap hsv
```



Create four subplots showing the effects of various `bar` arguments:

```
Y = round(rand(5,3)*10);
subplot(2,2,1)
bar(Y,'group')
title 'Group'


subplot(2,2,2)
bar(Y,'stack')
title 'Stack'


subplot(2,2,3)
barh(Y,'stack')
title 'Stack'
```

```
subplot(2,2,4)
bar(Y,1.5)
title 'Width = 1.5'
```



See Also `bar3`, `ColorSpec`, `patch`, `stairs`, `hist`

Purpose Three-dimensional bar chart

Syntax

```
bar3(Y)
bar3(x,Y)
bar3(...,width)
bar3(...,'style')
bar3(...,LineSpec)
h = bar3(...)

bar3h(...)
h = bar3h(...)
```

Description

`bar3` and `bar3h` draw three-dimensional vertical and horizontal bar charts.

`bar3(Y)` draws a three-dimensional bar chart, where each element in `Y` corresponds to one bar. When `Y` is a vector, the *x*-axis scale ranges from `1` to `length(Y)`. When `Y` is a matrix, the *x*-axis scale ranges from `1` to `size(Y,2)`, which is the number of columns, and the elements in each row are grouped together.

`bar3(x,Y)` draws a bar chart of the elements in `Y` at the locations specified in `x`, where `x` is a monotonic vector defining the *y*-axis intervals for vertical bars. If `Y` is a matrix, `bar3` clusters elements from the same row in `Y` at locations corresponding to an element in `x`. Values of elements in each row are grouped together.

`bar3(...,width)` sets the width of the bars and controls the separation of bars within a group. The default `width` is `0.8`, so if you do not specify `x`, bars within a group have a slight separation. If `width` is `1`, the bars within a group touch one another.

`bar3(...,'`*`style`*`')` specifies the style of the bars. `'`*`style`*`'` is `'detached'`, `'grouped'`, or `'stacked'`. `'detached'` is the default mode of display.

- `'detached'` displays the elements of each row in `Y` as separate blocks behind one another in the *x* direction.

- 'grouped' displays *n* groups of *m* vertical bars, where *n* is the number of rows and *m* is the number of columns in Y. The group contains one bar per column in Y.
- 'stacked' displays one bar for each row in Y. The bar height is the sum of the elements in the row. Each bar is multi-colored, with colors corresponding to distinct elements and showing the relative contribution each row element makes to the total sum.

bar3(...,LineSpec) displays all bars using the color specified by LineSpec.

h = bar3(...) returns a vector of handles to patch graphics objects. bar3 creates one patch object per column in Y.

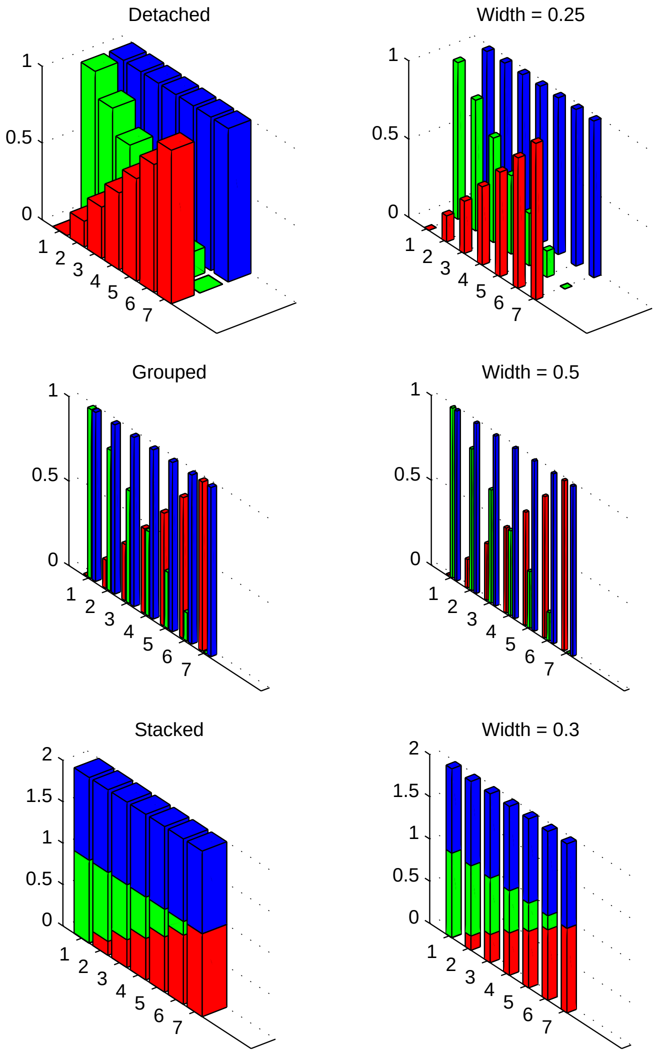
bar3h(...) and h = bar3h(...) create horizontal bars. Y determines the bar length. The vector x is a monotonic vector defining the *y*-axis intervals for horizontal bars.

Examples

This example creates six subplots showing the effects of different arguments for bar3. The data Y is a seven-by-three matrix generated using the cool colormap:

```
Y = cool(7);
subplot(3,2,1)
bar3(Y,'detached')
title('Detached')


subplot(3,2,2)
bar3(Y,0.25,'detached')
title('Width = 0.25')


subplot(3,2,3)
bar3(Y,'grouped')
title('Grouped')
```

```
subplot(3,2,4)
bar3(Y,0.5,'grouped')
title('Width = 0.5')

subplot(3,2,5)
bar3(Y,'stacked')
title('Stacked')

subplot(3,2,6)
bar3(Y,0.3,'stacked')
title('Width = 0.3')

colormap([1 0 0;0 1 0;0 0 1])
```



See Also `bar`, `LineSpec`, `patch`

Purpose Control axes border

Syntax

```
box on
box off
box
```

Description `box on` displays the boundary of the current axes.

`box off` does not display the boundary of the current axes.

`box` toggles the visible state of the current axes' boundary.

Algorithm The `box` function sets the axes `Box` property to `on` or `off`.

See Also `axes`

brighten

Purpose

Brighten or darken colormap

Syntax

```
brighten(beta)
brighten(h,beta)
newmap = brighten(beta)
newmap = brighten(cmap,beta)
```

Description

`brighten` increases or decreases the color intensities in a colormap. The modified colormap is brighter if `0 < beta < 1` and darker if `–1 < beta < 0`.

`brighten(beta)` replaces the current colormap with a brighter or darker colormap of essentially the same colors. `brighten(beta)`, followed by `brighten(–beta)`, where `beta < 1`, restores the original map.

`brighten(h,beta)` brightens all objects that are children of the figure having the handle `h`.

`newmap = brighten(beta)` returns a brighter or darker version of the current colormap without changing the display.

`newmap = brighten(cmap,beta)` returns a brighter or darker version of the colormap `cmap` without changing the display.

Examples

Brighten and then darken the current colormap:

```
beta = .5; brighten(beta);
beta = –.5; brighten(beta);
```

Algorithm

The values in the colormap are raised to the power of gamma, where gamma is

$$\gamma = \begin{cases} 1 - \beta, & \beta > 0 \\ \dfrac{1}{1 + \beta}, & \beta \le 0 \end{cases}$$

`brighten` has no effect on graphics objects defined with true color.

See Also

`colormap`, `rgbplot`

Purpose Move the camera position and target

Syntax

```
camdolly(dx,dy,dz)
camdolly(dx,dy,dz,'targetmode')
camdolly(dx,dy,dz,'targetmode','coordsys')
camdolly(axes_handle,...)
```

Description `camdolly` moves the camera position and the camera target by the specified amounts.

`camdolly(dx,dy,dz)` moves the camera position and the camera target by the specified amounts (see "Coordinate Systems").

`camdolly(dx,dy,dz,'`*`targetmode`*`')` The *targetmode* argument can take on two values that determine how MATLAB moves the camera:

- `movetarget` (default) – move both the camera and the target
- `fixtarget` – move only the camera

`camdolly(dx,dy,dz,'`*`targetmode`*`','`*`coordsys`*`')` The *coordsys* argument can take on three values that determine how MATLAB interprets `dx`, `dy`, and `dz`:

Coordinate Systems

- `camera` (default) – move in the camera's coordinate system. `dx` moves left/right, `dy` moves down/up, and `dz` moves along the viewing axis. The units are normalized to the scene.

  For example, setting `dx` to 1 moves the camera to the right, which pushes the scene to the left edge of the box formed by the axes position rectangle. A negative value moves the scene in the other direction. Setting `dz` to 0.5 moves the camera to a position halfway between the camera position and the camera target
- `pixels` – interpret `dx` and `dy` as pixel offsets. `dz` is ignored.
- `data` – interpret `dx`, `dy`, and `dz` as offesets in axes data coordinates.

`camdolly(axes_handle,...)` operates on the axes identified by the first argument, `axes_handle`. When you do not specify an axes handle, `camdolly` operates on the current axes.

Remarks

`camdolly` sets the axes `CameraPosition` and `CameraTarget` properties, which in turn causes the `CameraPositionMode` and `CameraTargetMode` properties to be set to manual.

Examples

This example moves the camera along the *x*- and *y*-axes in a series of steps.

```
surf(peaks)
axis vis3d
t = 0:pi/20:2*pi;
dx = sin(t)./40;
dy = cos(t)./40;
for i = 1:length(t);
    camdolly(dx(i),dy(i),0)
    drawnow
end
```

See Also

`axes`, `campos`, `camproj`, `camtarget`, `camup`, `camva`

The axes properties `CameraPosition`, `CameraTarget`, `CameraUpVector`, `CameraViewAngle`, `Projection`

The "Camera Properties" section in the online *Using MATLAB Graphics* topic accessed via the `helpdesk` command.

Purpose Create or move a light object in camera coordinates

Syntax

```
camlight headlight
camlight right
camlight left
camlight
camlight(az,el)
camlight(...‘style’)
camlight(light_handle,...)
light_handle = camlight(...)
```

Description `camlight('headlight')` creates a light at the camera position.

`camlight('right')` creates a light right and up from camera.

`camlight('left')` creates a light left and up from camera.

`camlight` with no arguments is the same as `camlight('right')`.

`camlight(az,el)` creates a light at the specified azimuth (`az`) and elevation (`el`) with respect to the camera position. The camera target is the center of rotation and `az` and `el` are in degrees.

`camlight(...,'style')` The style argument can take on the two values:

- `local` (default) – the light is a point source that radiates from the location in all directions.
- `infinite` – the light shines in parallel rays.

`camlight(light_handle,...)` uses the light specified in `light_handle`.

`light_handle = camlight(...)` returns the light's handle.

Remarks `camlight` sets the light object `Position` and `Style` properties. A light created with `camlight` will not track the camera. In order for the light to stay in a constant position relative to the camera, you must call `camlight` whenever you move the camera.

Examples

This example creates a light positioned to the left of the camera and then repositions the light each time the camera is moved:

```
surf(peaks)
axis vis3d
h = camlight('left');
for i = 1:20;
   camorbit(10,0)
   camlight(h, 'left')
   drawnow;
end
```

Purpose

Position the camera to view an object or group of objects

Syntax

```
camlookat(object_handles)
camlookat(axes_handle)
camlookat
```

Description

`camlookat(object_handles)` views the objects identified in the vector `object_handles`. The vector can contain the handles of axes children.

`camlookat(axes_handle)` views the objects that are children of the axes identified by `axes_handle`.

`camlookat` views the objects that are in the current axes.

Remarks

`camlookat` moves the camera position and camera target while preserving the relative view direction and camera view angle. The object (or objects) being viewed roughly fill the axes position rectangle.

`camlookat` sets the axes `CameraPosition` and `CameraTarget` properties.

Examples

This example creates three spheres at different locations and then progressively positions the camera so that each sphere is the object around which the scene is composed:

```
[x y z] = sphere;
s1 = surf(x,y,z);
hold on
s2 = surf(x+3,y,z+3);
s3 = surf(x,y,z+6);
daspect([1 1 1])
view(30,10)
camproj perspective
camlookat(gca) % Compose the scene around the current axes
pause(2)
camlookat(s1)  % Compose the scene around sphere s1
pause(2)
camlookat(s2)  % Compose the scene around sphere s2
pause(2)
camlookat(s3)  % Compose the scene around sphere s3
pause(2)
camlookat(gca)
```

See Also `campos`, `camtarget`

Purpose

Rotate the camera position around the camera target

Syntax

```
camorbit(dtheta,dphi)
camorbit(dtheta,dphi,'coordsys')
camorbit(dtheta,dphi,'coordsys','direction')
camorbit(axes_handle,...)
```

Description

`camorbit(dtheta,dphi)` rotates the camera position around the camera target by the amounts specified in `dtheta` and `dphi` (both in degrees). `dtheta` is the horizontal rotation and `dphi` is the vertical rotation.

`camorbit(dtheta,dphi,'`*`coordsys`*`')` The *coordsys* argument determines the center of rotation. It can take on two values:

- `data` (default) – rotate the camera around an axis defined by the camera target and the `direction` (default is the positive z direction).
- `camera` – rotate the camera about the point defined by the camera target.

`camorbit(dtheta,dphi,'`*`coordsys`*`','direction')` The `direction` argument, in conjunction with the camera target, defines the axis of rotation for the data coordinate system. Specify `direction` as a three-element vector containing the x, y, and z-components of the direction or one of the characters, `x`, `y`, or `z`, to indicate `[1 0 0]`, `[0 1 0]`, or `[0 0 1]` respectively.

`camorbit(axes_handle,...)` operates on the axes identified by the first argument, `axes_handle`. When you do not specify an axes handle, `camorbit` operates on the current axes.

Examples

Compare rotation in the two coordinate systems with these `for` loops. The first rotates the camera horizontally about a line defined by the camera target point and a direction that is parallel to the *y*-axis. Visualize this rotation as a cone formed with the camera target at the apex and the camera position forming the base:

```
surf(peaks)
axis vis3d
for i=1:36
   camorbit(10,0,'data',[0 1 0])
   drawnow
end
```

Rotation in the camera coordinate system orbits the camera around the axes along a circle while keeping the center of a circle at the camera target.

```
surf(peaks)
axis vis3d
for i=1:36
   camorbit(10,0,'camera')
   drawnow
end
```

See Also

axes, axis('vis3d'), camdolly, campan, camzoom, camroll

Purpose Rotate the camera target around the camera position

Syntax

```
campan(dtheta,dphi)
campan(dtheta,dphi,'coordsys')
campan(dtheta,dphi,'coordsys','direction')
campan(axes_handle,...)
```

Description

`campan(dtheta,dphi)` rotates the camera target around the camera position by the amounts specified in `dtheta` and `dphi` (both in degrees). `dtheta` is the horizontal rotation and `dphi` is the vertical rotation.

`campan(dtheta,dphi,'`*coordsys*`')` The `coordsys` argument determines the center of rotation. It can take on two values:

- `data` (default) – rotate the camera target around an axis defined by the camera position and the `direction` (default is the positive z direction)
- `camera` – rotate the camera about the point defined by the camera target.

`campan(dtheta,dphi,'`*coordsys*`','direction')` The `direction` argument, in conjunction with the camera position, defines the axis of rotation for the data coordinate system. Specify `direction` as a three-element vector containing the x, y, and z-components of the direction or one of the characters, `x`, `y`, or `z`, to indicate `[1 0 0]`, `[0 1 0]`, or `[0 0 1]` respectively.

`campan(axes_handle,...)` operates on the axes identified by the first argument, `axes_handle`. When you do not specify an axes handle, `campan` operates on the current axes.

See Also `axes`, `camdolly`, `camorbit`, `camtarget`, `camzoom`, `camroll`

campos

Purpose

Set or query the camera position

Syntax

```
campos
campos([camera_position])
campos('mode')
campos('auto'
campos('manual')
campos(axes_handle,...)
```

Description

`campos` with no arguments returns the camera position in the current axes.

`campos([camera_position])` sets the position of the camera in the current axes to the specified value. Specify the position as a three-element vector containing the x-, y-, and z-coordinates of the desired location in the data units of the axes.

`campos('mode')` returns the value of the camera position mode, which can be either `auto` (the default) or `manual`.

`campos('auto')` sets the camera position mode to `auto`.

`campos('manual')` sets the camera position mode to `manual`.

`campos(axes_handle,...)` performs the set or query on the axes identified by the first argument, `axes_handle`. When you do not specify an axes handle, `campos` operates on the current axes.

Remarks

`campos` sets or queries values of the axes `CameraPosition` and `CameraPositionMode` properties. The camera position is the point in the Cartesian coordinate system of the axes from which you view the scene.

Examples

This example moves the camera along the *x*-axis in a series of steps:

```
surf(peaks)
axis vis3d off
for x = –200:5:200
    campos([x,5,10])
    drawnow
end
```

See Also

axis, camproj, camtarget, camup, camva

The axes properties CameraPosition, CameraTarget, CameraUpVector, CameraViewAngle, Projection

Purpose Set or query the projection type

Syntax

```
camproj
camproj(projection_type)
camproj(axes_handle,...)
```

Description The projection type determines whether MATLAB uses a perspective or orthographic projection for 3-D views.

`camproj` with no arguments returns the projection type setting in the current axes.

`camproj('projection_type')` sets the projection type in the current axes to the specified value. Possible values for *projection_type* are: `orthographic` and `perspective`.

`camproj(axes_handle,...)` performs the set or query on the axes identified by the first argument, `axes_handle`. When you do not specify an axes handle, `camproj` operates on the current axes.

Remarks `camproj` sets or queries values of the axes object `Projection` property.

See Also `campos`, `camtarget`, `camup`, `camva`

The axes properties `CameraPosition`, `CameraTarget`, `CameraUpVector`, `CameraViewAngle`, `Projection`

Purpose Rotate the camera about the view axis

Syntax

```
camroll(dtheta)
camroll(axes_handle,dtheta)
```

Description `camroll(dtheta)` rotates the camera around the camera viewing axis by the amounts specified in `dtheta` (in degrees). The viewing axis is defined by the line passing through the camera position and the camera target.

`camroll(axes_handle,dtheta)` operates on the axes identified by the first argument, `axes_handle`. When you do not specify an axes handle, `camroll` operates on the current axes.

Remarks `camroll` set the axes `CameraUpVector` property and thereby also sets the `CameraUpVectorMode` property to `manual`.

See Also `axes`, `axis('vis3d')`, `camdolly`, `camorbit`, `camzoom`, `campan`

camtarget

Purpose Set or query the location of the camera target

Syntax

```
camtarget
camtarget([camera_target])
camtarget('mode')
camtarget('auto')
camtarget('manual')
camtarget(axes_handle,...)
```

Description The camera target is the location in the axes that the camera points to. The camera remains oriented toward this point regardless of its position.

`camtarget` with no arguments returns the location of the camera target in the current axes.

`camtarget([camera_target])` sets the camera target in the current axes to the specified value. Specify the target as a three-element vector containing the x-, y-, and z-coordinates of the desired location in the data units of the axes.

`camtarget('mode')` returns the value of the camera target mode, which can be either `auto` (the default) or `manual`.

`camtarget('auto')` sets the camera target mode to `auto`.

`camtarget('manual')` sets the camera target mode to `manual`.

`camtarget(axes_handle,...)` performs the set or query on the axes identified by the first argument, `axes_handle`. When you do not specify an axes handle, `camtarget` operates on the current axes.

Remarks `camtarget` sets or queries values of the axes object `Cameratarget` and `CameraTargetMode` properties.

When the camera target mode is `auto`, MATLAB positions the camera target at the center of the axes plot box.

Examples

This example moves the camera position and the camera target along the *x*-axis in a series of steps:

```
surf(peaks);
axis vis3d
xp = linspace(−150,40,50);
xt = linspace(25,50,50);
for i=1:50
     campos([xp(i),25,5]);
     camtarget([xt(i),30,0])
     drawnow
end
```

See Also

`axis`, `camproj`, `campos`, `camup`, `camva`

The axes properties `CameraPosition`, `CameraTarget`, `CameraUpVector`, `CameraViewAngle`, `Projection`

camup

Purpose

Set or query the camera up vector

Syntax

```
camup
camup([up_vector])
camup('mode')
camup('auto')
camup('manual')
camup(axes_handle,...)
```

Description

The camera up vector specifies the direction that is oriented up in the scene.

`camup` with no arguments returns the camera up vector setting in the current axes.

`camup([up_vector])` sets the up vector in the current axes to the specified value. Specify the up vector as x-, y-, and z-components. See Remarks.

`camup('mode')` returns the current value of the camera up vector mode, which can be either `auto` (the default) or `manual`.

`camup('auto')` sets the camera up vector mode to `auto`. In `auto` mode, MATLAB uses a value for the up vector of `[0 1 0]` for 2-D views. This means the z-axis points up.

`camup('manual')` sets the camera up vector mode to `manual`. In `manual` mode, MATLAB does not change the value of the camera up vector.

`camup(axes_handle,...)` performs the set or query on the axes identified by the first argument, `axes_handle`. When you do not specify an axes handle, `camup` operates on the current axes.

Remarks

`camup` sets or queries values of the axes object `CameraUpVector` and `CameraUpVectorMode` properties.

Specify the camera up vector as the x-, y-, and z-coordinates of a point in the axes coordinate system that forms the directed line segment PQ, where P is the point (0,0,0) and Q is the specified x-, y-, and z-coordinates. This line always points up. The length of the line PQ has no effect on the orientation of the scene. This means a value of `[0 0 1]` produces the same results as `[0 0 25]`.

See Also

axis, camproj, campos, camtarget, camva

The axes properties CameraPosition, CameraTarget, CameraUpVector, CameraViewAngle, Projection

camva

Purpose

Set or query the camera view angle

Syntax

```
camva
camva(view_angle)
camva('mode')
camva('auto')
camva('manual')
camva(axes_handle,...)
```

Description

The camera view angle determines the field of view of the camera. Larger angles produce a smaller view of the scene. You can implement zooming by changing the camera view angle.

`camva` with no arguments returns the camera view angle setting in the current axes.

`camva(view_angle)` sets the view angle in the current axes to the specified value. Specify the view angle in degrees.

`camva('mode')` returns the current value of the camera view angle mode, which can be either `auto` (the default) or `manual`. See Remarks.

`camva('auto')` sets the camera view angle mode to `auto`.

`camva('manual')` sets the camera view angle mode to `manual`. See Remarks.

`camva(axes_handle,...)` performs the set or query on the axes identified by the first argument, `axes_handle`. When you do not specify an axes handle, `camva` operates on the current axes.

Remarks

`camva` sets or queries values of the axes object `CameraViewAngle` and `CameraViewAngleMode` properties.

When the camera view angle mode is `auto`, MATLAB adjusts the camera view angle so that the scene fills the available space in the window. If you move the camera to a different position, MATLAB changes the camera view angle to maintain a view of the scene that fills the available area in the window.

Setting a camera view angle or setting the camera view angle to `manual` disables MATLAB's stretch-to-fill feature (stretching of the axes to fit the window). This means setting the camera view angle to its current value,

```
camva(camva)
```

can cause a change in the way the graph looks. See the Remarks section of the axes reference page for more information.

Examples

This example creates two pushbuttons, one that zooms in and another that zooms out.

```
uicontrol('Style','pushbutton',...
  'String','Zoom In',...
  'Position',[20 20 60 20],...
  'Callback','if camva <= 1;return;else;camva(camva-1);end');
uicontrol('Style','pushbutton',...
  'String','Zoom Out',...
  'Position',[100 20 60 20],...
  'Callback','if camva >= 179;return;else;camva(camva+1);end');
```

Now create a graph to zoom in and out on:

```
surf(peaks);
```

Note the range checking in the callback statements. This keeps the values for the camera view angle in the range, greater than zero and less than 180.

See Also

`axis`, `camproj`, `campos`, `camup`, `camtarget`

The axes properties `CameraPosition`, `CameraTarget`, `CameraUpVector`, `CameraViewAngle`, `Projection`

Purpose Zoom in and out on a scene

Syntax

```
camzoom(zoom_factor)
camzoom(axes_handle,...)
```

Description `camzoom(zoom_factor)` zooms in or out on the scene depending on the value specified by `zoom_factor`. If `zoom_factor` is greater than 1, the scene appears larger; if `zoom_factor` is greater than zero and less than 1, the scene appears smaller.

`camzoom(axes_handle,...)` operates on the axes identified by the first argument, `axes_handle`. When you do not specify an axes handle, `camzoom` operates on the current axes.

Remarks `camzoom` sets the axes `CameraViewAngle` property, which in turn causes the `CameraViewAngleMode` property to be set to `manual`. Note that setting the `CameraViewAngle` property disables MATLAB's stretch-to-fill feature (stretching of the axes to fit the window). This may result in a change to the aspect ratio of your graph. See the `axes` function for more information on this behavior.

See Also `axes`, `camdolly`, `camorbit`, `campan`, `camroll`, `camva`

Purpose

`capture` is obsolete in Release 11 (5.3). `getframe` provides the same functionality and supports TrueColor displays by returning TrueColor images.

Syntax

```
capture
capture(h)
[X,cmap] = capture(h)
```

Description

`capture` creates a bitmap copy of the contents of the current figure, including any uicontrol graphics objects. It creates a new figure and displays the bitmap copy as an image graphics object in the new figure.

`capture(h)` creates a new figure that contains a copy of the figure identified by `h`.

`[X,cmap] = capture(h)` returns an image matrix `X` and a colormap. You display this information using the statements

```
colormap(cmap)
image(X)
```

Remarks

The resolution of a bitmap copy is less than that obtained with the `print` command.

See Also

`image`, `print`

Purpose Color axis scaling

Syntax

```
caxis([cmin cmax])
caxis auto
caxis manual
caxis(caxis)
v = caxis
```

Description `caxis` controls the mapping of data values to the colormap. It affects any surfaces, patches, and images with indexed `CData` and `CDataMapping` set to `scaled`. It does not affect surfaces, patches, or images with true color `CData` or with `CDataMapping` set to `direct`.

`caxis([cmin cmax])` sets the color limits to specified minimum and maximum values. Data values less than `cmin` or greater than `cmax` map to `cmin` and `cmax`, respectively. Values between `cmin` and `cmax` linearly map to the current colormap.

`caxis auto` lets MATLAB compute the color limits automatically using the minimum and maximum data values. This is MATLAB's default behavior. Color values set to `Inf` map to the maximum color, and values set to `–Inf` map to the minimum color. Faces or edges with color values set to `NaN` are not drawn.

`caxis manual` and `caxis(caxis)` freeze the color axis scaling at the current limits. This enables subsequent plots to use the same limits when `hold` is `on`.

`v = caxis` returns a two-element row vector containing the `[cmin cmax]` currently in use.

Remarks `caxis` changes the `CLim` and `CLimMode` properties of axes graphics objects.

surface, patch, and image graphics objects with indexed `CData` and `CDataMapping` set to `scaled` map `CData` values to colors in the figure colormap each time they render. `CData` values equal to or less than `cmin` map to the first color value in the colormap, and `CData` values equal to or greater than `cmax` map to the last color value in the colormap. MATLAB performs the following linear transformation on the intermediate values (referred to as `C` below) to map them to an entry in the colormap (whose length is m, and whose row index is referred to as index below).

```
index = fix((C–cmin)/(cmax–cmin)*m)+1
```

Examples

Create (X,Y,Z) data for a sphere and view the data as a surface.

```
[X,Y,Z] = sphere;
C = Z;
surf(X,Y,Z,C)
```

Values of C have the range [–1 1]. Values of C near –1 are assigned the lowest values in the colormap; values of C near 1 are assigned the highest values in the colormap.

To map the top half of the surface to the highest value in the color table, use

```
caxis([–1 0])
```

To use only the bottom half of the color table, enter

```
caxis([–1 3])
```

which maps the lowest CData values to the bottom of the colormap, and the highest values to the middle of the colormap (by specifying a cmax whose value is equal to cmin plus twice the range of the CData).

The command

```
caxis auto
```

resets axis scaling back to auto-ranging and you see all the colors in the surface. In this case, entering

```
caxis
```

returns

```
[-1 1]
```

Adjusting the color axis can be useful when using images with scaled color data. For example, load the image data and colormap for Cape Code, Massachusetts.

```
load cape
```

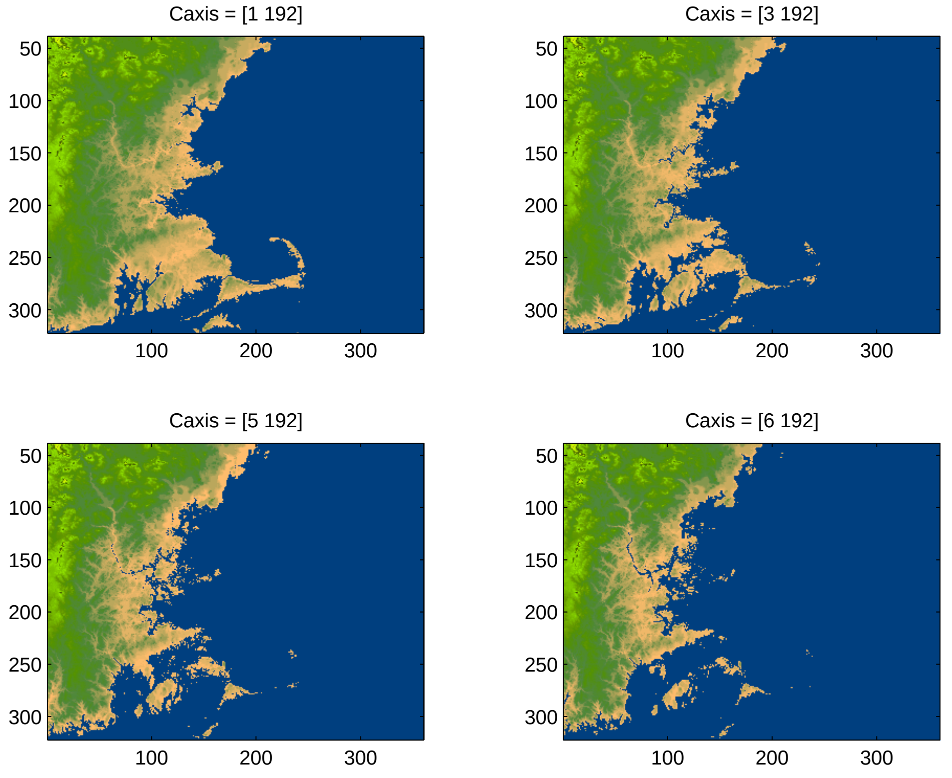
This command loads the images data X and the image's colormap map into the workspace. Now display the image with CDataMapping set to scaled and install the image's colormap.

```
image(X,'CDataMapping','scaled')
colormap(map)
```

MATLAB sets the color limits to span the range of the image data, which is 1 to 192:

```
caxis
ans =
     1   192
```

The blue color of the ocean is the first color in the colormap and is mapped to the lowest data value (1). You can effectively move sealevel by changing the lower color limit value. For example,



See Also

`axes`, `axis`, `colormap`, `get`, `mesh`, `pcolor`, `set`, `surf`

The `CLim` and `CLimMode` properties of axes graphics objects.

The `Colormap` property of figure graphics objects.

The *Using MATLAB Graphics* manual.

cla

Purpose Clear current axes

Syntax

```
cla
cla reset
```

Description `cla` deletes from the current axes all graphics objects whose handles are not hidden (i.e., their `HandleVisibility` property is set to `on`).

`cla reset` deletes from the current axes all graphics objects regardless of the setting of their `HandleVisibility` property and resets all axes properties, except `Position` and `Units`, to their default values.

Remarks The `cla` command behaves the same way when issued on the command line as it does in callback routines – it does not recognize the `HandleVisibility` setting of `callback`. This means that when issued from within a callback routine, `cla` deletes only those objects whose `HandleVisibility` property is set to `on`.

See Also `clf`, `hold`, `newplot`, `reset`

Purpose Contour plot elevation labels

Syntax

```
clabel(C,h)
clabel(C,h,v)
clabel(C,h,'manual')

clabel(C)
clabel(C,v)
clabel(C,'manual')
```

Description The `clabel` function adds height labels to a two-dimensional contour plot.

`clabel(C,h)` rotates the labels and inserts them in the contour lines. The function inserts only those labels that fit within the contour, depending on the size of the contour.

`clabel(C,h,v)` creates labels only for those contour levels given in vector `v`, then rotates the labels and inserts them in the contour lines.

`clabel(C,h,'manual')` places contour labels at locations you select with a mouse. Press the left mouse button (the mouse button on a single-button mouse) or the space bar to label a contour at the closest location beneath the center of the cursor. Press the **Return** key while the cursor is within the figure window to terminate labeling. The labels are rotated and inserted in the contour lines.

`clabel(C)` adds labels to the current contour plot using the contour structure `C` output from `contour`. The function labels all contours displayed and randomly selects label positions.

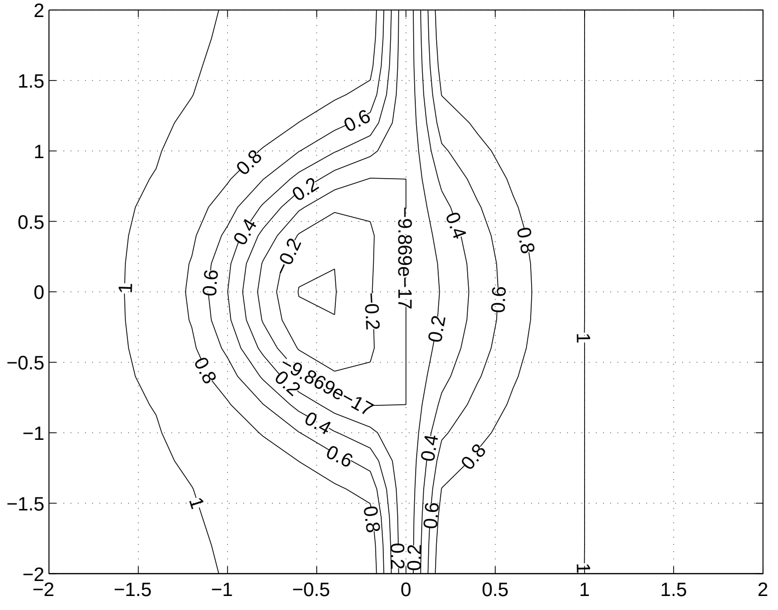
`clabel(C,v)` labels only those contour levels given in vector `v`.

`clabel(C,'manual')` places contour labels at locations you select with a mouse.

Remarks When the syntax includes the argument `h`, this function rotates the labels and inserts them in the contour lines (see Example). Otherwise, the labels are displayed upright and a `'+'` indicates which contour line the label is annotating.

Examples Generate, draw, and label a simple contour plot.

```
[x,y] = meshgrid(–2:.2:2);
z = x.^exp(–x.^2–y.^2);
[C,h] = contour(x,y,z);
clabel(C,h);
```



See Also `contour`, `contourc`, `contourf`

Purpose Clear current figure window

Syntax

```
clf
clf reset
```

Description `clf` deletes from the current figure all graphics objects whose handles are not hidden (i.e., their `HandleVisibility` property is set to `on`).

`clf reset` deletes from the current figure all graphics objects regardless of the setting of their `HandleVisibility` property and resets all figure properties, except `Position`, `Units`, `PaperPosition`, and `PaperUnits` to their default values.

Remarks The `clf` command behaves the same way when issued on the command line as it does in callback routines – it does not recognize the `HandleVisibility` setting of `callback`. This means that when issued from within a callback routine, `clf` deletes only those objects whose `HandleVisibility` property is set to on.

See Also `cla`, `clc`, `hold`, `reset`

close

Purpose Delete specified figure

Syntax

```
close
close(h)
close name
close all
close all hidden
status = close(...)
```

Description

`close` deletes the current figure or the specified figure(s). It optionally returns the status of the close operation.

`close` deletes the current figure (equivalent to `close(gcf)`).

`close(h)` deletes the figure identified by `h`. If `h` is a vector or matrix, `close` deletes all figures identified by `h`.

`close name` deletes the figure with the specified name.

`close all` deletes all figures whose handles are not hidden.

`close all hidden` deletes all figures including those with hidden handles.

`status = close(...)` returns `1` if the specified windows have been deleted and `0` otherwise.

Remarks

The `close` function works by evaluating the specified figure's `CloseRequestFcn` property with the statement:

```
eval(get(h,'CloseRequestFcn'))
```

The default `CloseRequestFcn`, `closereq`, deletes the current figure using `delete(get(0,'CurrentFigure'))`. If you specify multiple figure handles, `close` executes each figure's `CloseRequestFcn` in turn. If MATLAB encounters an error that terminates the execution of a `CloseRequestFcn`, the figure is not deleted. Note that using your computer's window manager (i.e., the **Close** menu item) also calls the figure's `CloseRequestFcn`.

If a figure's handle is hidden (i.e., the figure's `HandleVisibility` property is set to `callback` or `off` and the root `ShowHiddenHandles` property is set on), you must specify the `hidden` option when trying to access a figure using the `all` option.

To delete all figures unconditionally, use the statements:

```
set(0,'ShowHiddenHandles','on')
delete(get(0,'Children'))
```

The delete function does not execute the figure's `CloseRequestFcn`; it simply deletes the specified figure.

The figure `CloseRequestFcn` allows you to either delay or abort the closing of a figure once the `close` function has been issued. For example, you can display a dialog box to see if the user really wants to delete the figure or save and clean up before closing.

See Also

`delete`, `figure`, `gcf`

The figure `HandleVisibility` property

The root `ShowHiddenHandles` property

Purpose

Display colorbar showing the color scale

Syntax

```
colorbar
colorbar('vert')
colorbar('horiz')
colorbar(h)
h = colorbar(...)
```

Description

The `colorbar` function displays the current colormap in the current figure and resizes the current axes to accommodate the colorbar.

`colorbar` updates the most recently created colorbar or, when the current axes does not have a colorbar, `colorbar` adds a new vertical colorbar.

`colorbar('vert')` adds a vertical colorbar to the current axes.

`colorbar('horiz')` adds a horizontal colorbar to the current axes.

`colorbar(h)` places a colorbar in the axes identified by `h`. The colorbar is horizontal if the width of the axes is greater than its height, as determined by the axes `Position` property.

`h = colorbar(...)` returns a handle to the colorbar, which is an axes graphics object.

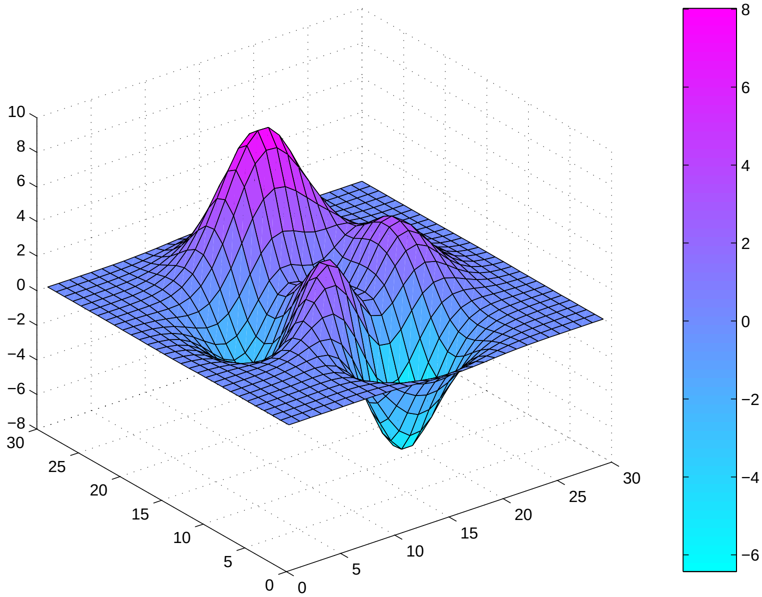
Remarks

`colorbar` works with two-dimensional and three-dimensional plots.

Examples

Display a colorbar beside the axes.

```
surf(peaks(30))
colormap cool
colorbar
```



See Also

`colormap`

colordef

Purpose

Sets default property values to display different color schemes

Syntax

```
colordef white
colordef black
colordef none
colordef(fig,color_option)
h = colordef('new',color_option)
```

Description

`colordef` enables you to select either a white or black background for graphics display. It sets axis lines and labels to show up against the background color.

`colordef white` sets the axis background color to white, the axis lines and labels to black, and the figure background color to light gray.

`colordef black` sets the axis background color to black, the axis lines and labels to white, and the figure background color to dark gray.

`colordef none` sets the figure coloring to that used by MATLAB Version 4 (essentially a black background).

`colordef(fig,color_option)` sets the color scheme of the figure identified by the handle `fig` to the color option `'white'`, `'black'`, or `'none'`.

`h = colordef('new',color_option)` returns the handle to a new figure created with the specified color options (i.e., `'white'`, `'black'`, or `'none'`).

Remarks

`colordef` affects only subsequently drawn figures, not those currently on the display. This is because `colordef` works by setting default property values (on the root or figure level). You can list the currently set default values on the root level with the statement:

```
get(0,'defaults')
```

You can remove all default values using the `reset` command:

```
reset(0)
```

See the `get` and `reset` references pages for more information.

See Also

`whitebg`

Purpose Set and get the current colormap

Syntax

```
colormap(map)
colormap('default')
cmap = colormap
```

Description A colormap is an *m*-by-3 matrix of real numbers between 0.0 and 1.0. Each row is an RGB vector that defines one color. The k^{th} row of the colormap defines the *k*-th color, where `map(k,:) = [r(k) g(k) b(k)])` specifies the intensity of red, green, and blue.

`colormap(map)` sets the colormap to the matrix `map`. If any values in `map` are outside the interval [0 1], MATLAB returns the error: `Colormap must have values in [0,1]`.

`colormap('default')` sets the current colormap to the default colormap.

`cmap = colormap;` retrieves the current colormap. The values returned are in the interval [0 1].

Specifying Colormaps

M-files in the `color` directory generate a number of colormaps. Each M-file accepts the colormap size as an argument. For example,

```
colormap(hsv(128))
```

creates an `hsv` colormap with 128 colors. If you do not specify a size, MATLAB creates a colormap the same size as the current colormap.

Supported Colormaps

MATLAB supports a number of colormaps.

- `autumn` varies smoothly from red, through orange, to yellow.
- `bone` is a grayscale colormap with a higher value for the blue component. This colormap is useful for adding an "electronic" look to grayscale images.
- `colorcube` contains as many regularly spaced colors in RGB colorspace as possible, while attempting to provide more steps of gray, pure red, pure green, and pure blue.
- `cool` consists of colors that are shades of cyan and magenta. It varies smoothly from cyan to magenta.
- `copper` varies smoothly from black to bright copper.
- `flag` consists of the colors red, white, blue, and black. This colormap completely changes color with each index increment.
- `gray` returns a linear grayscale colormap.
- `hot` varies smoothly from black, through shades of red, orange, and yellow, to white.
- `hsv` 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 colormap is particularly appropriate for displaying periodic functions. `hsv(m)` is the same as `hsv2rgb([h ones(m,2)])` where `h` is the linear ramp, `h = (0:m–1)'/m`.
- `jet` ranges from blue to red, and passes through the colors cyan, yellow, and orange. It is a variation of the `hsv` colormap. The `jet` colormap is associated with an astrophysical fluid jet simulation from the National Center for Supercomputer Applications. See the "Examples" section.
- `lines` produces a colormap of colors specified by the axes `ColorOrder` property and a shade of gray.
- `pink` contains pastel shades of pink. The pink colormap provides sepia tone colorization of grayscale photographs.
- `prism` repeats the six colors red, orange, yellow, green, blue, and violet.
- `spring` consists of colors that are shades of magenta and yellow.
- `summer` consists of colors that are shades of green and yellow.
- `white` is an all white monochrome colormap.
- `winter` consists of colors that are shades of blue and green.

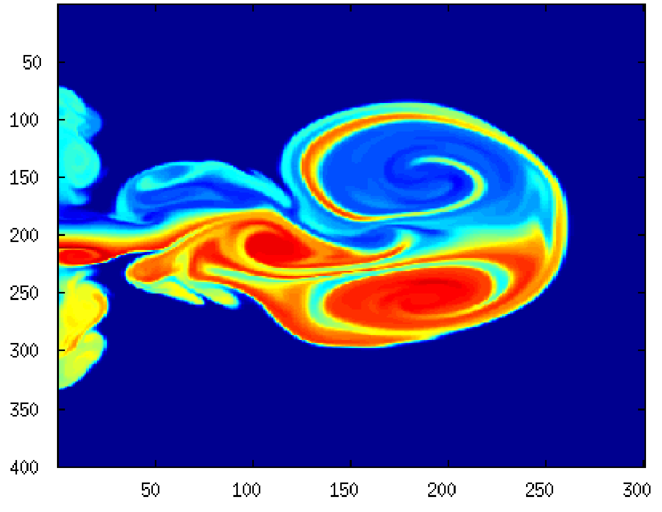
Examples

The images and colormaps demo, `imagedemo`, provides an introduction to colormaps. Select **Color Spiral** from the menu. This uses the `pcolor` function to display a 16-by-16 matrix whose elements vary from 0 to 255 in a rectilinear spiral. The `hsv` colormap starts with red in the center, then passes through yellow, green, cyan, blue, and magenta before returning to red at the outside end of the spiral. Selecting **Colormap Menu** gives access to a number of other colormaps.

The `rgbplot` function plots colormap values. Try `rgbplot(hsv)`, `rgbplot(gray)`, and `rgbplot(hot)`.

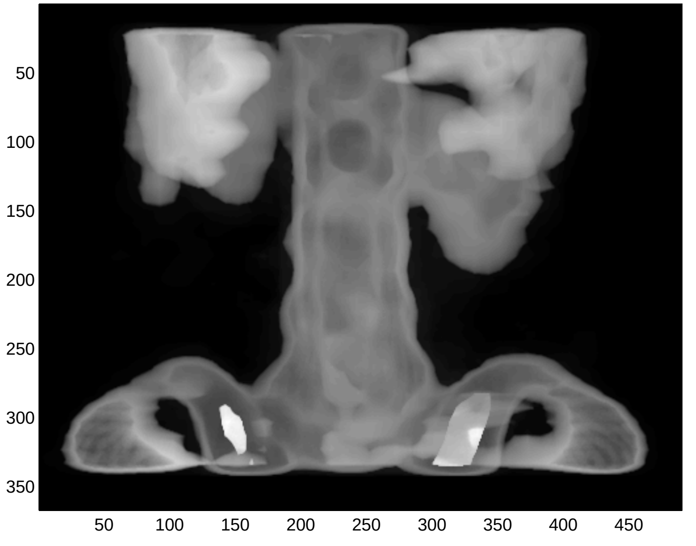
The following commands display the `flujet` data using the `jet` colormap.

```
load flujet
image(X)
colormap(jet)
```



The demos directory contains a CAT scan image of a human spine. To view the image, type the following commands:

```
load spine
image(X)
colormap bone
```



Algorithm

Each figure has its own Colormap property. colormap is an M-file that sets and gets this property.

See Also

brighten, caxis, contrast, hsv2rgb, pcolor, rgb2hsv, rgbplot

The Colormap property of figure graphics objects.

Purpose

Color specification

Description

`ColorSpec` is not a command; it refers to the three ways in which you specify color in MATLAB:

- RGB triple
- Short name
- Long name

The short names and long names are MATLAB strings that specify one of eight predefined colors. The RGB triple is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color; the intensities must be in the range [0 1]. The following table lists the predefined colors and their RGB equivalents.

RGB Value	Short Name	Long Name
`[1 1 0]`	`y`	`yellow`
`[1 0 1]`	`m`	`magenta`
`[0 1 1]`	`c`	`cyan`
`[1 0 0]`	`r`	`red`
`[0 1 0]`	`g`	`green`
`[0 0 1]`	`b`	`blue`
`[1 1 1]`	`w`	`white`
`[0 0 0]`	`k`	`black`

Remarks

The eight predefined colors and any colors you specify as RGB values are not part of a figure's colormap, nor are they affected by changes to the figure's colormap. They are referred to as *fixed* colors, as opposed to *colormap* colors.

Examples

To change the background color of a figure to green, specify the color with a short name, a long name, or an RGB triple. These statements generate equivalent results:

```
whitebg('g')
whitebg('green')
whitebg([0 1 0]);
```

You can use `ColorSpec` anywhere you need to define a color. For example, this statement changes the figure background color to pink:

```
set(gcf,'Color',[1,0.4,0.6])
```

See Also

`bar`, `bar3`, `colordef`, `colormap`, `fill`, `fill3`, `whitebg`

Purpose Two-dimensional comet plot

Syntax

```
comet(y)
comet(x,y)
comet(x,y,p)
```

Description A comet plot is an animated graph in which a circle (the comet *head*) traces the data points on the screen. The comet *body* is a trailing segment that follows the head. The *tail* is a solid line that traces the entire function.

`comet(y)` displays a comet plot of the vector `y`.

`comet(x,y)` displays a comet plot of vector `y` versus vector `x`.

`comet(x,y,p)` specifies a comet body of length `p*length(y)`. `p` defaults to `0.1`.

Remarks Note that the trace left by `comet` is created by using an `EraseMode` of none, which means you cannot print the plot (you get only the comet head) and it disappears if you cause a redraw (e.g., by resizing the window).

Examples Create a simple comet plot:

```
t = 0:.01:2*pi;
x = cos(2*t).*(cos(t).^2);
y = sin(2*t).*(sin(t).^2);
comet(x,y);
```

See Also `comet3`

comet3

Purpose Three-dimensional comet plot

Syntax

```
comet3(z)
comet3(x,y,z)
comet3(x,y,z,p)
```

Description A comet plot is an animated graph in which a circle (the comet *head*) traces the data points on the screen. The comet *body* is a trailing segment that follows the head. The *tail* is a solid line that traces the entire function.

`comet3(z)` displays a three-dimensional comet plot of the vector `z`.

`comet3(x,y,z)` displays a comet plot of the curve through the points `[x(i),y(i),z(i)]`.

`comet3(x,y,z,p)` specifies a comet body of length `p*length(y)`.

Remarks Note that the trace left by `comet3` is created by using an `EraseMode` of `none`, which means you cannot print the plot (you get only the comet head) and it disappears if you cause a redraw (e.g., by resizing the window).

Examples Create a three-dimensional comet plot.

```
t = –10*pi:pi/250:10*pi;
comet3((cos(2*t).^2).*sin(t),(sin(2*t).^2).*cos(t),t);
```

See Also `comet`

Purpose Plot arrows emanating from the origin

Syntax

```
compass(X,Y)
compass(Z)
compass(...,LineSpec)
h = compass(...)
```

Description A compass plot displays direction or velocity vectors as arrows emanating from the origin. `X`, `Y`, and `Z` are in Cartesian coordinates and plotted on a circular grid.

`compass(X,Y)` displays a compass plot having *n* arrows, where *n* is the number of elements in `X` or `Y`. The location of the base of each arrow is the origin. The location of the tip of each arrow is a point relative to the base and determined by `[X(i),Y(i)]`.

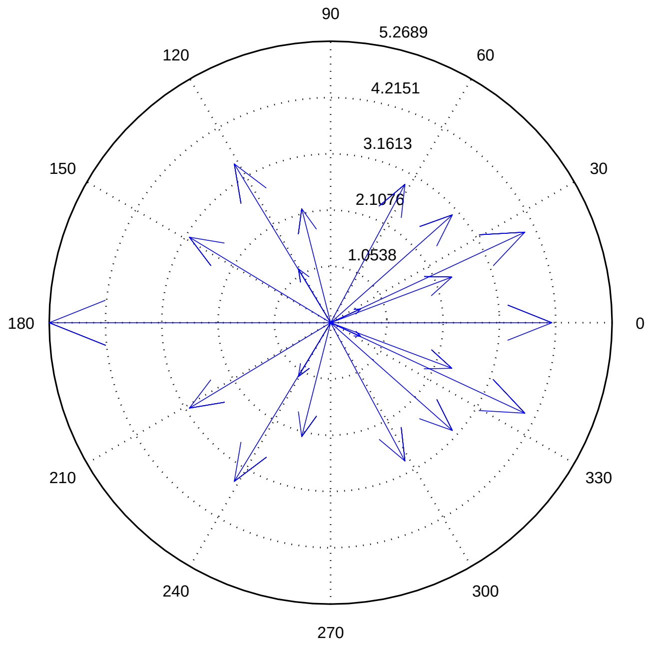
`compass(Z)` displays a compass plot having *n* arrows, where *n* is the number of elements in `Z`. The location of the base of each arrow is the origin. The location of the tip of each arrow is relative to the base as determined by the real and imaginary components of `Z`. This syntax is equivalent to `compass(real(Z),imag(Z))`.

`compass(...,LineSpec)` draws a compass plot using the line type, marker symbol, and color specified by `LineSpec`.

`h = compass(...)` returns handles to line objects.

Examples Draw a compass plot of the eigenvalues of a matrix.

```
Z = eig(randn(20,20));
compass(Z)
```



See Also `feather`, `LineSpec`, `rose`

Purpose

Plot velocity vectors as cones in a 3-D vector field

Syntax

```
coneplot(X,Y,Z,U,V,W,Cx,Cy,Cz)
coneplot(U,V,W,Cx,Cy,Cz)
coneplot(...,s)
coneplot(...,'quiver')
coneplot(...,'method')
h = coneplot(...)
```

Description

`coneplot(X,Y,Z,U,V,W,Cx,Cy,Cz)` plots velocity vectors as cones pointing in the direction of the velocity vector and having a length proportional to the magnitude of the velocity vector.

- `X`, `Y`, `Z` define the coordinates for the vector field.
- `U`, `V`, `W` define the vector field. These arrays must be the same size, monotonic, and 3-D plaid (such as the data produced by `meshgrid`).
- `Cx`, `Cy`, `Cz` define the location of the cones in vector field.

`coneplot(U,V,W,Cx,Cy,Cz)` (omitting the `X`, `Y`, and `Z` arguments) assumes `[X,Y,Z] = meshgrid(1:n,1:m,1:p)` where `[m,n,p]= size(U)`.

`coneplot(...,s)` MATLAB automatically scales the cones to fit the graph and then stretches them by the scale factor `s`. If you do not specify a value for `s`, MATLAB uses a value of 1. Use s = `0` to plot the cones without automatic scaling.

`coneplot(...,'quiver')` draws arrows instead of cones (see `quiver3` for an illustration of a quiver plot).

`coneplot(...,'`*method*`')` specifies the interpolation method to use. *method* can be: `linear`, `cubic`, `nearest`. `linear` is the default (see `interp3` for a discussion of these interpolation methods)

`h = coneplot(...)` returns the handle to the `patch` object used to draw the cones. You can use the `set` command to change the properties of the cones.

Remarks

`coneplot` automatically scales the cones to fit the graph, while keeping them in proportion to the respective velocity vectors.

It is usually best to set the data aspect ratio of the axes before calling coneplot. You can set the ratio using the daspect command,

```
daspect([1,1,1])
```

Examples

This example plots the velocity vector cones for vector volume data representing the motion of air through a rectangular region of space. The final graph employs a number of enhancements to visualize the data more effectively. These include:

- Cone plots indicate the magnitude and direction of the wind velocity.
- Slice planes placed at the limits of the data range provide a visual context for the cone plots within the volume.
- Directional lighting provides visual queues as to the orientation of the cones.
- View adjustments compose the scene to best reveal the information content of the data by selecting the view point, projection type, and magnification.

Load and Inspect Data

The winds data set contains six 3-D arrays: u, v, and w specify the vector components at each of the coordinate specified in x, y, and z. The coordinates define a lattice grid structure where the data is sampled within the volume.

It is useful to establish the range of the data to place the slice planes and to specify where you want the cone plots (min, max).

```
load wind
xmin = min(x(:));
xmax = max(x(:));
ymin = min(y(:));
ymax = max(y(:));
zmin = min(z(:));
```

Create the Cone Plot

- Decide where in data space you want to plot cones. This example selects the full range of x and y in eight steps and the range 3 to 15 in four steps in z (linspace, meshgrid).
- Use daspect to set the data aspect ratio of the axes before calling coneplot so MATLAB can determine the proper size of the cones.

- Draw the cones, setting the scale factor to 5 to make the cones larger than the default size.
- Set the coloring of each cone (FaceColor, EdgeColor).

```
daspect([2,2,1])
xrange = linspace(xmin,xmax,8);
yrange = linspace(ymin,ymax,8);
zrange = 3:4:15;
[cx cy cz] = meshgrid(xrange,yrange,zrange);
hcones = coneplot(x,y,z,u,v,w,cx,cy,cz,5);
set(hcones,'FaceColor','red','EdgeColor','none')
```

Add the Slice Planes

- Calculate the magnitude of the vector field (which represents wind speed) to generate scalar data for the slice command.
- Create slice planes along the x-axis at xmin and xmax, along the y-axis at ymax, and along the z-axis at zmin.
- Specify interpolated face color so the slice coloring indicates wind speed and do not draw edges (hold, slice, FaceColor, EdgeColor).

```
hold on
wind_speed = sqrt(u.^2 + v.^2 + w.^2);
hsurfaces = slice(x,y,z,wind_speed,[xmin,xmax],ymax,zmin);
set(hsurfaces,'FaceColor','interp','EdgeColor','none')
hold off
```

Define the View

- Use the axis command to set the axis limits equal to the range of the data.
- Orient the view to azimuth = 30 and elevation = 40 (rotate3d is a useful command for selecting the best view).
- Select perspective projection to provide a more realistic looking volume (camproj).
- Zoom in on the scene a little to make the plot as large as possible (camzoom).

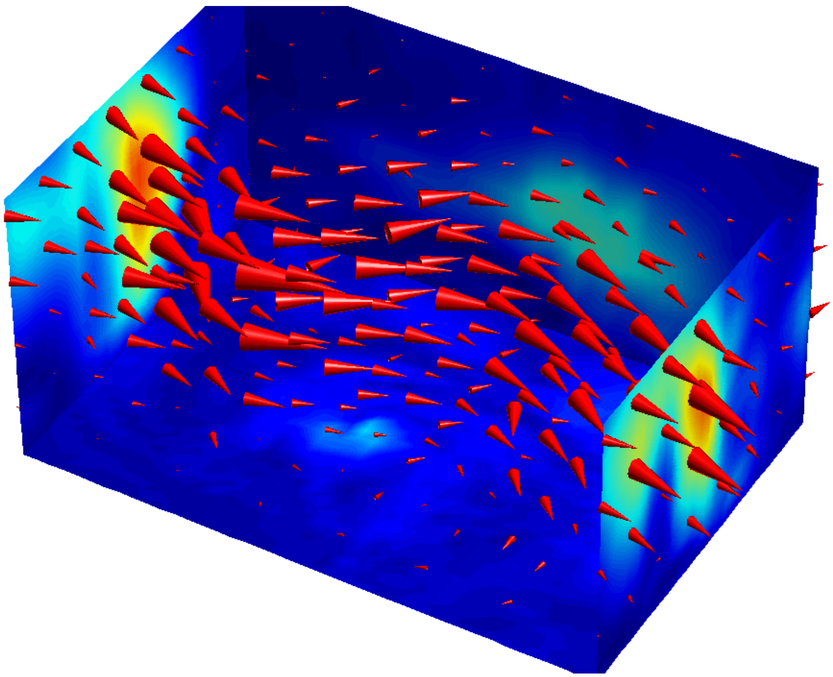
```
axis tight; view(30,40); axis off
camproj perspective; camzoom(1.5)
```

Add Lighting to the Scene

The light source affects both the slice planes (surfaces) and the cone plots (patches). However, you can set the lighting characteristics of each independently.

- Add a light source to the right of the camera and use Phong lighting give the cones and slice planes a smooth, three-dimensional appearance (`camlight`, `lighting`).
- Increase the value of the `AmbientStrength` property for each slice plane to improve the visibility of the dark blue colors. (Note that you can also specify a different `colormap` to change to coloring of the slice planes.)
- Increase the value of the `DiffuseStrength` property of the cones to brighten particularly those cones not showing specular reflections.

```
camlight right; lighting phong
set(hsurfaces,'AmbientStrength',.6)
set(hcones,'DiffuseStrength',.8)
```



See Also

isosurface, patch, reducevolume, smooth3, streamline, stream2, stream3, subvolume

contour

Purpose Two-dimensional contour plot

Syntax

```
contour(Z)
contour(Z,n)
contour(Z,v)
contour(X,Y,Z)
contour(X,Y,Z,n)
contour(X,Y,Z,v)
contour(...,LineSpec)
[C,h] = contour(...)
```

Description A contour plot displays isolines of matrix `Z`. Label the contour lines using `clabel`.

`contour(Z)` draws a contour plot of matrix `Z`, where `Z` is interpreted as heights with respect to the *x*-*y* plane. `Z` must be at least a 2-by-2 matrix. The number of contour levels and the values of the contour levels are chosen automatically based on the minimum and maximum values of `Z`. The ranges of the *x*- and *y*-axis are `[1:n]` and `[1:m]`, where `[m,n] = size(Z)`.

`contour(Z,n)` draws a contour plot of matrix `Z` with `n` contour levels.

`contour(Z,v)` draws a contour plot of matrix `Z` with contour lines at the data values specified in vector `v`. The number of contour levels is equal to `length(v)`. To draw a single contour of level `i`, use `contour(Z,[i i])`.

`contour(X,Y,Z)`, `contour(X,Y,Z,n)`, and `contour(X,Y,Z,v)` draw contour plots of `Z`. `X` and `Y` specify the *x*- and *y*-axis limits. When `X` and `Y` are matrices, they must be the same size as `Z`, in which case they specify a surface as `surf` does.

`contour(...,LineSpec)` draws the contours using the line type and color specified by `LineSpec`. contour ignores marker symbols.

`[C,h] = contour(...)` returns the contour matrix `C` (see `contourc`) and a vector of handles to graphics objects. `clabel` uses the contour matrix `C` to create the labels. `contour` creates patch graphics objects unless you specify `LineSpec`, in which case `contour` creates line graphics objects.

Remarks

If you do not specify LineSpec, colormap and caxis control the color.

If X or Y is irregularly spaced, contour calculates contours using a regularly spaced contour grid, then transforms the data to X or Y.

Examples

To view a contour plot of the function

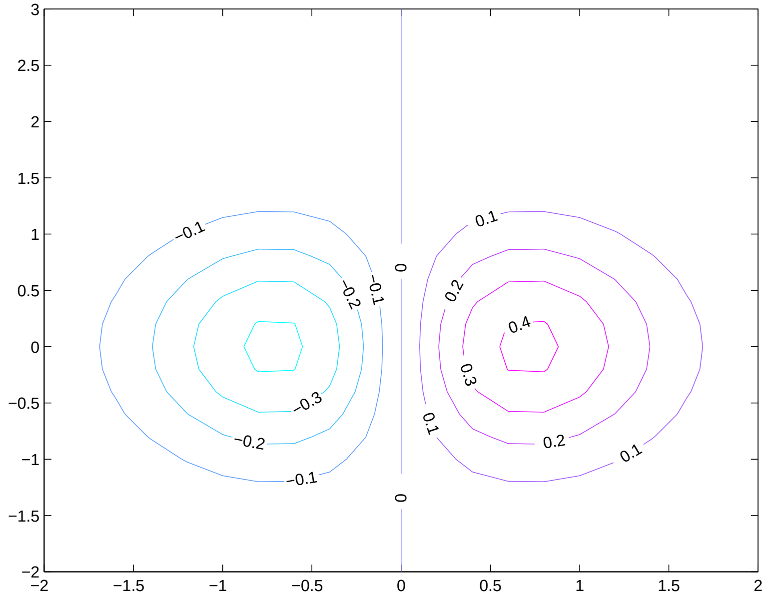
$$z = xe^{(-x^2 - y^2)}$$

over the range $-2 \le x \le 2$, $-2 \le y \le 3$, create matrix Z using the statements

```
[X,Y] = meshgrid(-2:.2:2,-2:.2:3);
Z = X.*exp(-X.^2-Y.^2);
```

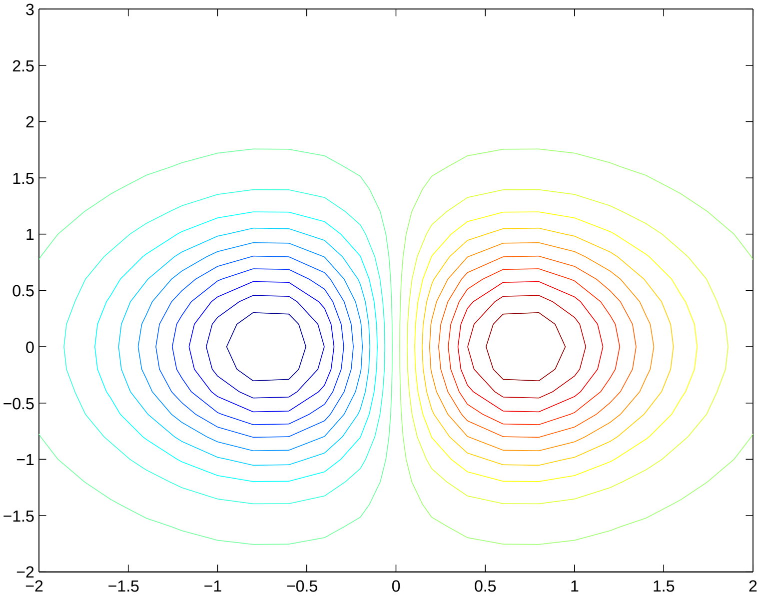
Then, generate a contour plot of Z.

```
[C,h] = contour(X,Y,Z);
clabel(C,h)
colormap cool
```



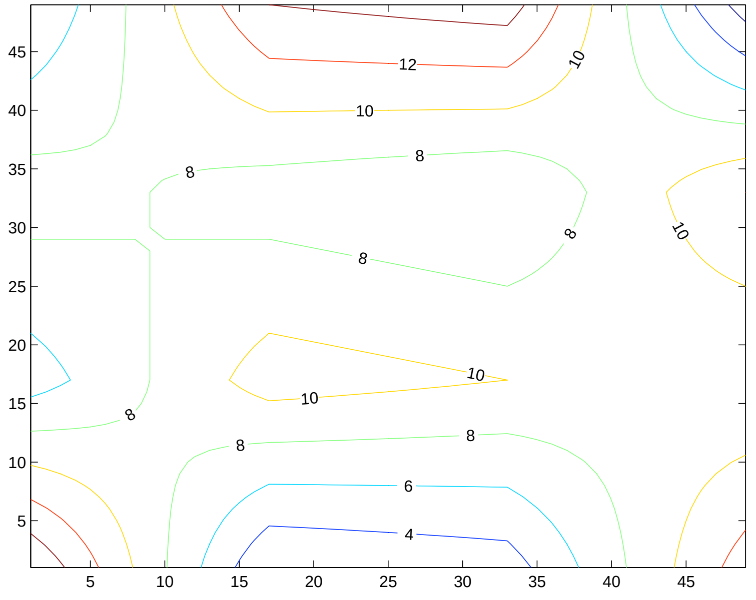
View the same function over the same range with 20 evenly spaced contour lines and colored with the default colormap `jet`.

```
contour(X,Y,Z,20)
```



Use `interp2` and `contour` to create smoother contours.

```
Z = magic(4);
[C,h] = contour(interp2(Z,4));
clabel(C,h)
```



See Also clabel, contour3, contourc, contourf, interp2, quiver

Purpose

Three-dimensional contour plot

Syntax

```
contour3(Z)
contour3(Z,n)
contour3(Z,v)
contour3(X,Y,Z)
contour3(X,Y,Z,n)
contour3(X,Y,Z,v)
contour3(...,LineSpec)
[C,h] = contour3(...)
```

Description

`contour3` creates a three-dimensional contour plot of a surface defined on a rectangular grid.

`contour3(Z)` draws a contour plot of matrix `Z` in a three-dimensional view. `Z` is interpreted as heights with respect to the *x*-*y* plane. `Z` must be at least a 2-by-2 matrix. The number of contour levels and the values of contour levels are chosen automatically. The ranges of the *x*- and *y*-axis are `[1:n]` and `[1:m]`, where `[m,n] = size(Z)`.

`contour3(Z,n)` draws a contour plot of matrix `Z` with `n` contour levels in a three-dimensional view.

`contour3(Z,v)` draws a contour plot of matrix `Z` with contour lines at the values specified in vector `v`. The number of contour levels is equal to `length(v)`. To draw a single contour of level `i`, use `contour(Z,[i i])`.

`contour3(X,Y,Z)`, `contour3(X,Y,Z,n)`, and `contour3(X,Y,Z,v)` use `X` and `Y` to define the *x*- and *y*-axis limits. If `X` is a matrix, `X(1,:)` defines the *x*-axis. If `Y` is a matrix, `Y(:,1)` defines the *y*-axis. When `X` and `Y` are matrices, they must be the same size as `Z`, in which case they specify a surface as `surf` does.

`contour3(...,LineSpec)` draws the contours using the line type and color specified by `LineSpec`.

`[C,h] = contour3(...)` returns the contour matrix `C` as described in the function `contourc` and a column vector containing handles to graphics objects. `contour3` creates patch graphics objects unless you specify `LineSpec`, in which case `contour3` creates line graphics objects.

Remarks

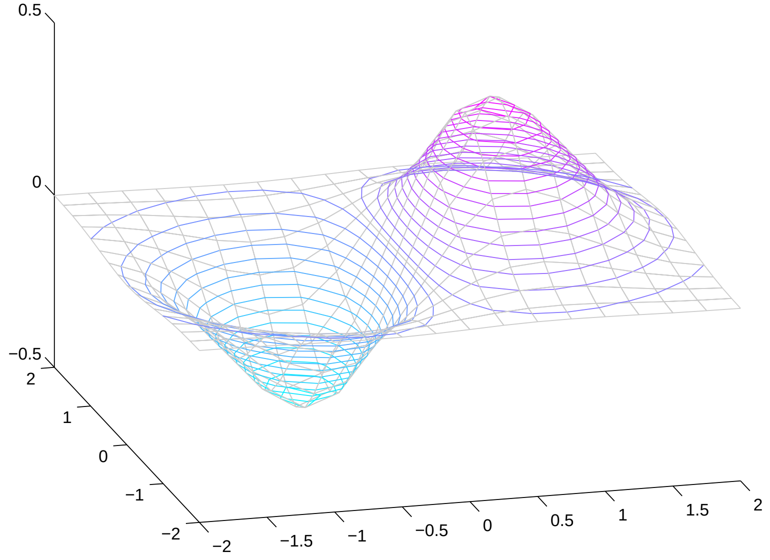
If you do not specify LineSpec, colormap and caxis control the color.

If X or Y is irregularly spaced, contour3 calculates contours using a regularly spaced contour grid, then transforms the data to X or Y.

Examples

Plot the three-dimensional contour of a function and superimpose a surface plot to enhance visualization of the function.

```
[X,Y] = meshgrid([-2:.25:2]);
Z = X.*exp(-X.^2-Y.^2);
contour3(X,Y,Z,30)
surface(X,Y,Z,'EdgeColor',[.8 .8 .8],'FaceColor','none')
grid off
view(-15,25)
colormap cool
```



See Also

contour, contourc, meshc, meshgrid, surfc

contourc

Purpose Low-level contour plot computation

Syntax

```
C = contourc(Z)
C = contourc(Z,n)
C = contourc(Z,v)
C = contourc(x,y,Z)
C = contourc(x,y,Z,n)
C = contourc(x,y,Z,v)
```

Description `contourc` calculates the contour matrix `C` used by `contour`, `contour3`, and `contourf`. The values in `Z` determine the heights of the contour lines with respect to a plane. The contour calculations use a regularly spaced grid determined by the dimensions of `Z`.

`C = contourc(Z)` computes the contour matrix from data in matrix `Z`, where `Z` must be at least a 2-by-2 matrix. The contours are isolines in the units of `Z`. The number of contour lines and the corresponding values of the contour lines are chosen automatically.

`C = contourc(Z,n)` computes contours of matrix `Z` with `n` contour levels.

`C = contourc(Z,v)` computes contours of matrix `Z` with contour lines at the values specified in vector `v`. The length of `v` determines the number of contour levels. To compute a single contour of level `i`, use `contourc(Z,[i i])`.

`C = contourc(x,y,Z)`, `C = contourc(x,y,Z,n)`, and `C = contourc(x,y,Z,v)` compute contours of `Z` using vectors `x` and `y` to determine the *x*- and *y*-axis limits. `x` and `y` must be monotonically increasing.

Remarks `C` is a two-row matrix specifying all the contour lines. Each contour line defined in matrix `C` begins with a column that contains the value of the contour (specified by `v` and used by `clabel`), and the number of `(x,y)` vertices in the contour line. The remaining columns contain the data for the `(x,y)`pairs.

```
C = [value1 xdata(1) xdata(2)...value2 xdata(1) xdata(2)...;
      dim1   ydata(1) ydata(2)...dim2   ydata(1) ydata(2)...]
```

Specifying irregularly spaced `x` and `y` vectors is not the same as contouring irregularly spaced data. If `x` or `y` is irregularly spaced, `contourc` calculates contours using a regularly spaced contour grid, then transforms the data to `x` or `y`.

See Also

`clabel`, `contour`, `contour3`, `contourf`

contourf

Purpose

Filled two-dimensional contour plot

Syntax

```
contourf(Z)
contourf(Z,n)
contourf(Z,v)
contourf(X,Y,Z)
contourf(X,Y,Z,n)
contourf(X,Y,Z,v)
[C,h,CF] = contourf(...)
```

Description

A filled contour plot displays isolines calculated from matrix `Z` and fills the areas between the isolines using constant colors. The color of the filled areas depends on the current figure's colormap.

`contourf(Z)` draws a contour plot of matrix `Z`, where `Z` is interpreted as heights with respect to a plane. `Z` must be at least a 2-by-2 matrix. The number of contour lines and the values of the contour lines are chosen automatically.

`contourf(Z,n)` draws a contour plot of matrix `Z` with `n` contour levels.

`contourf(Z,v)` draws a contour plot of matrix `Z` with contour levels at the values specified in vector `v`.

`contourf(X,Y,Z)`, `contourf(X,Y,Z,n)`, and `contourf(X,Y,Z,v)` produce contour plots of `Z` using `X` and `Y` to determine the *x*- and *y*-axis limits. When `X` and `Y` are matrices, they must be the same size as `Z`, in which case they specify a surface as `surf` does.

`[C,h,CF] = contourf(...)` returns the contour matrix `C` as calculated by the function `contourc` and used by `clabel`, a vector of handles `h` to patch graphics objects, and a contour matrix `CF` for the filled areas.

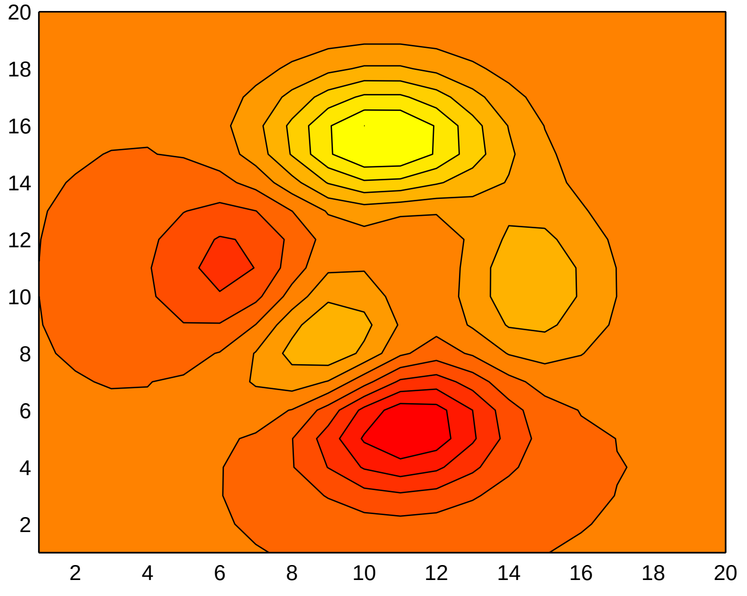
Remarks

If `X` or `Y` is irregularly spaced, `contourf` calculates contours using a regularly spaced contour grid, then transforms the data to `X` or `Y`.

Examples

Create a filled contour plot of the peaks function.

```
[C,h] = contourf(peaks(20),10);
colormap autumn
```



See Also

`clabel`, `contour`, `contour3`, `contourc`, `quiver`

contourslice

Purpose Draw contours in volume slice planes

Syntax

```
contourslice(X,Y,Z,V,Sx,Sy,Sz)
contourslice(X,Y,Z,V,Xi,Yi,Zi)
contourslice(V,Sx,Sy,Sz), contourslice(V,Xi,Yi,Zi)
contourslice(...,n)
contourslice(...,cvals)
contourslice(...,[cv cv])
contourslice(...,'method')
h = contourslice(...)
```

Description

`contourslice(X,Y,Z,V,Sx,Sy,Sz)` draws contours in the x-, y-, and z-axis aligned planes at the points in the vectors `Sx`, `Sy`, `Sz`. The arrays `X`, `Y`, and `Z` define the coordinates for the volume `V` and must be monotonic and 3-D plaid (such as the data produced by `meshgrid`). The color at each contour is determined by the volume `V`, which must be an m-by-n-by-p volume array.

`contourslice(X,Y,Z,V,Xi,Yi,Zi)` draws contours through the volume `V` along the surface defined by the arrays `Xi,Yi,Zi`.

`contourslice(V,Sx,Sy,Sz)` and `contourslice(V,Xi,Yi,Zi)` (omitting the `X`, `Y`, and `Z` arguments) assumes `[X,Y,Z] = meshgrid(1:n,1:m,1:p)` where `[m,n,p]= size(v)`.

`contourslice(...,n)` draws `n` contour lines per plane, overriding the automatic value.

`contourslice(...,cvals)` draws `length(cval)` contour lines per plane at the values specified in vector `cvals`.

`contourslice(...,[cv cv])` computes a single contour per plane at the level `cv`.

`contourslice(...,'`*`method`*`')` specifies the interpolation method to use. *method* can be: `linear`, `cubic`, `nearest`. `nearest` is the default except when the contours are being drawn along the surface defined by `Xi`, `Yi`, `Zi`, in which case `linear` is the default (see `interp3` for a discussion of these interpolation methods).

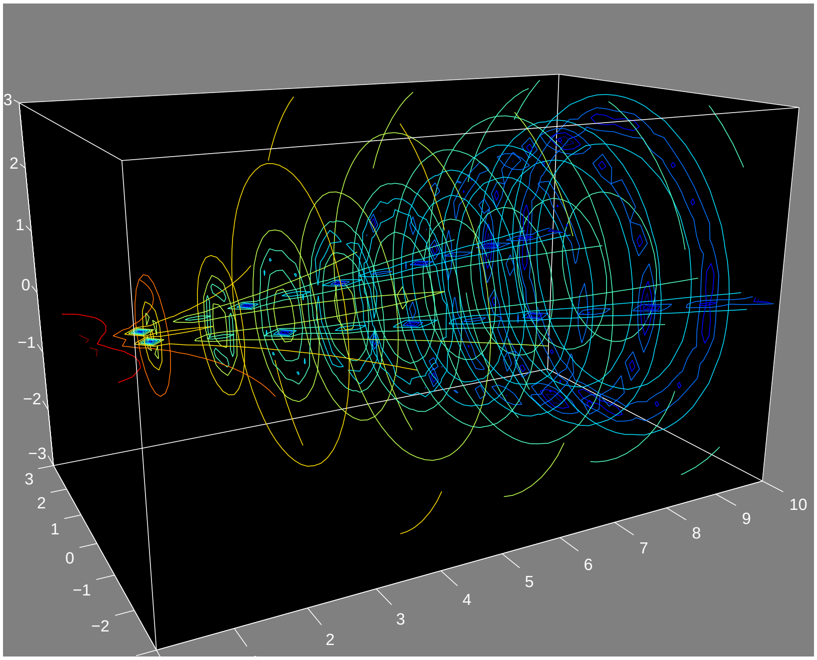
`h = contourslice(...)` returns a vector of handles to `patch` objects that are used to implement the contour lines.

Examples

This example uses the `flow` data set to illustrate the use of contoured slice planes (type `help flow` for more information on this data set). Notice that this example:

- Specifies a vector of `length = 9` for `Sx`, an empty vector for the `Sy`, and a scalar value (0) for `Sz`. This creates nine contour plots along the x direction in the y-z plane, and one in the x-y plane at z = 0.
- Uses `linspace` to define a ten-element linearly spaced vector of values from -8 to 2 that specifies the number of contour lines to draw at each interval.
- Defines the view and projection type (`camva`, `camproj`, `campos`)
- Sets figure (`gcf`) and axes (`gca`) characteristics.

```
[x y z v] = flow;
h = contourslice(x,y,z,v,[1:9],[],[0],linspace(-8,2,10));
axis([0,10,-3,3,-3,3]); daspect([1,1,1])
camva(24); camproj perspective;
campos([-3,-15,5])
set(gcf,'Color',[.5,.5,.5],'Renderer','zbuffer')
set(gca,'Color','black','XColor','white', ...
   'YColor','white','ZColor','white')
box on
```



See Also

`isosurface`, `smooth3`, `subvolume`, `reducevolume`

Purpose

Grayscale colormap for contrast enhancement

Syntax

```
cmap = contrast(X)
cmap = contrast(X,m)
```

Description

The `contrast` function enhances the contrast of an image. It creates a new gray colormap, `cmap`, that has an approximately equal intensity distribution. All three elements in each row are identical.

`cmap = contrast(X)` returns a gray colormap that is the same length as the current colormap.

`cmap = contrast(X,m)` returns an `m`-by-3 gray colormap.

Examples

Add contrast to the clown image defined by `X`.

```
load clown;
cmap = contrast(X);
image(X);
colormap(cmap);
```

See Also

`brighten`, `colormap`, `image`

copyobj

Purpose Copy graphics objects and their descendants

Syntax

```
new_handle = copyobj(h,p)
```

Description

`copyobj` creates copies of graphics objects. The copies are identical to the original objects except the copies have different values for their `Parent` property and a new handle. The new parent must be appropriate for the copied object (e.g., you can copy a line object only to another axes object).

`new_handle = copyobj(h,p)` copies one or more graphics objects identified by `h` and returns the handle of the new object or a vector of handles to new objects. The new graphics objects are children of the graphics objects specified by `p`.

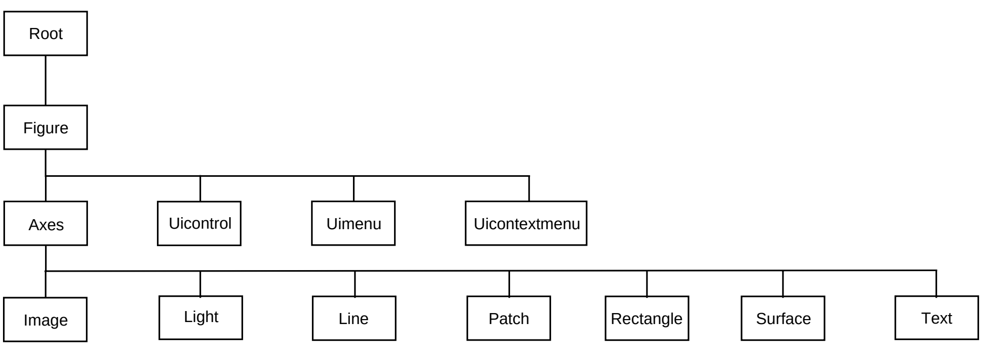
Remarks

`h` and `p` can be scalars or vectors. When both are vectors, they must be the same length and the output argument, `new_handle`, is a vector of the same length. In this case, `new_handle(i)` is a copy of `h(i)` with its `Parent` property set to `p(i)`.

When `h` is a scalar and `p` is a vector, `h` is copied once to each of the parents in `p`. Each `new_handle(i)` is a copy of `h` with its `Parent` property set to `p(i)`, and `length(new_handle)` equals `length(p)`.

When `h` is a vector and `p` is a scalar, each `new_handle(i)` is a copy of `h(i)` with its `Parent` property set to `p`. The length of `new_handle` equals `length(h)`.

Graphics objects are arranged as a hierarchy. Here, each graphics object is shown connected below its appropriate parent object.



Examples

Copy a surface to a new axes within a different figure.

```
h = surf(peaks);
colormap hot
figure     % Create a new figure
axes       % Create an axes object in the figure
new_handle = copyobj(h,gca);
colormap hot
view(3)
grid on
```

Note that while the surface is copied, the colormap (figure property), view, and grid (axes properties) are not copies.

See Also

`findobj`, `gcf`, `gca`, `gco`, `get`, `set`

`Parent` property for all graphics objects

cylinder

Purpose Generate cylinder

Syntax

```
[X,Y,Z] = cylinder
[X,Y,Z] = cylinder(r)
[X,Y,Z] = cylinder(r,n)
cylinder(...)
```

Description `cylinder` generates *x*, *y*, and *z* coordinates of a unit cylinder. You can draw the cylindrical object using `surf` or `mesh`, or draw it immediately by not providing output arguments.

`[X,Y,Z] = cylinder` returns the *x*, *y*, and *z* coordinates of a cylinder with a radius equal to `1`. The cylinder has 20 equally spaced points around its circumference.

`[X,Y,Z] = cylinder(r)` returns the *x*, *y*, and *z* coordinates of a cylinder using `r` to define a profile curve. `cylinder` treats each element in `r` as a radius at equally spaced heights along the unit height of the cylinder. The cylinder has 20 equally spaced points around its circumference.

`[X,Y,Z] = cylinder(r,n)` returns the *x*, *y*, and *z* coordinates of a cylinder based on the profile curve defined by vector `r`. The cylinder has `n` equally spaced points around its circumference.

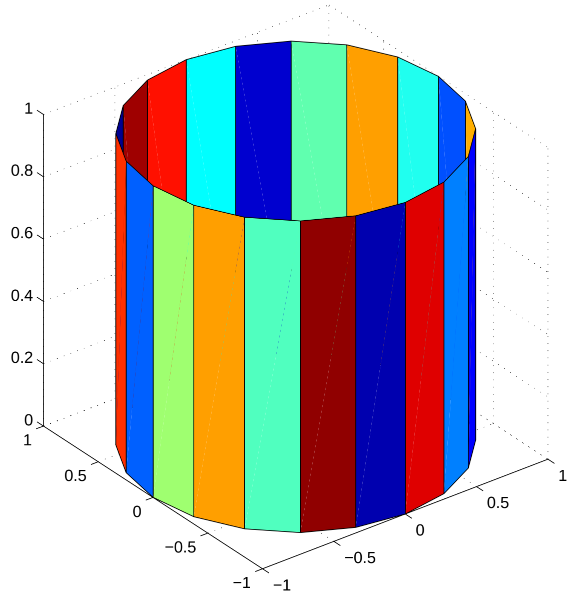
`cylinder(...)`, with no output arguments, plots the cylinder using `surf`.

Remarks `cylinder` treats its first argument as a profile curve. The resulting surface graphics object is generated by rotating the curve about the *x*-axis, and then aligning it with the *z*-axis.

Examples

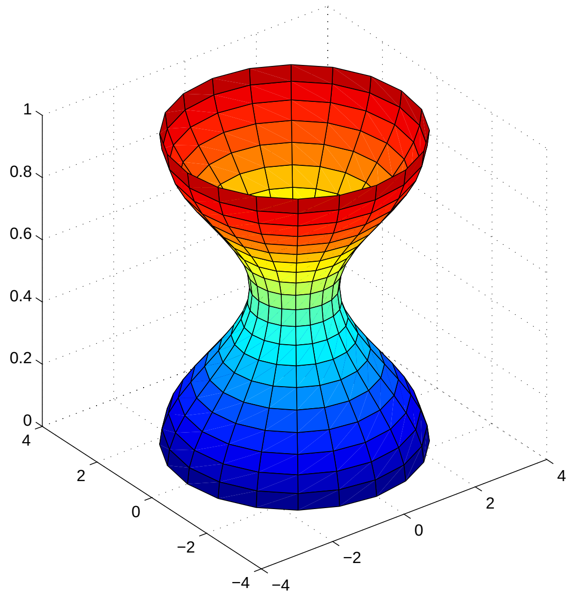
Create a cylinder with randomly colored faces.

```
cylinder
axis square
h = findobj('Type','surface');
set(h,'CData',rand(size(get(h,'CData'))))
```



Generate a cylinder defined by the profile function `2+sin(t)`.

```
t = 0:pi/10:2*pi;
[X,Y,Z] = cylinder(2+cos(t));
surf(X,Y,Z)
axis square
```



See Also sphere, surf

Purpose

Set or query the axes data aspect ratio

Syntax

```
daspect
daspect([aspect_ratio])
daspect('mode')
daspect('auto')
daspect('manual')
daspect(axes_handle,...)
```

Description

The data aspect ratio determines the relative scaling of the data units along the *x*-, *y*-, and *z*-axes.

`daspect` with no arguments returns the data aspect ratio of the current axes.

`daspect([aspect_ratio])` sets the data aspect ratio in the current axes to the specified value. Specify the aspect ratio as three relative values representing the ratio of the *x*-, *y*-, and *z*-axis scaling (e.g., `[1 1 3]` means one unit in *x* is equal in length to one unit in *y* and three unit in *z*).

`daspect('mode')` returns the current value of the data aspect ratio mode, which can be either `auto` (the default) or `manual`. See Remarks.

`daspect('auto')` sets the data aspect ratio mode to `auto`.

`daspect('manual')` sets the data aspect ratio mode to `manual`.

`daspect(axes_handle,...)` performs the set or query on the axes identified by the first argument, `axes_handle`. When you do not specify an axes handle, `daspect` operates on the current axes.

Remarks

`daspect` sets or queries values of the axes object `DataAspectRatio` and `DataAspectRatioMode` properties.

When the data aspect ratio mode is `auto`, MATLAB adjusts the data aspect ratio so that each axis spans the space available in the figure window. If you are displaying a representation of a real-life object, you should set the data aspect ratio to `[1 1 1]` to produce the correct proportions.

Setting a value for data aspect ratio or setting the data aspect ratio mode to `manual` disables MATLAB's stretch-to-fill feature (stretching of the axes to fit the window). This means setting the data aspect ratio to a value, including its current value,

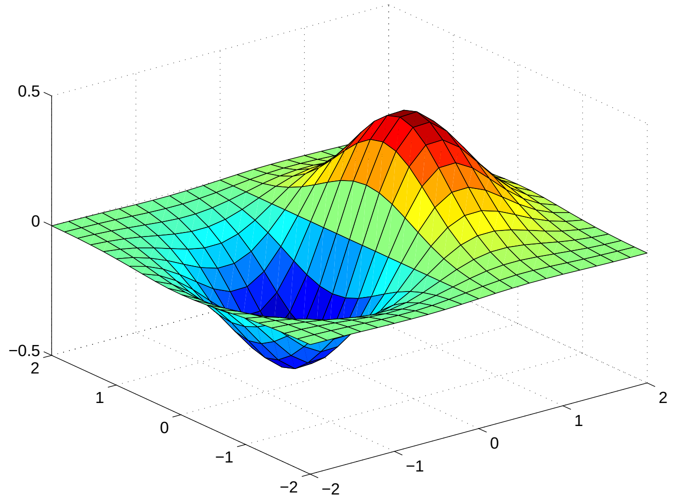
```
daspect(daspect)
```

can cause a change in the way the graphs look. See the Remarks section of the axes description for more information.

Examples

The following surface plot of the function $z = xe^{(-x^2 - y^2)}$ is useful to illustrate the data aspect ratio. First plot the function over the range $-2 \le x \le 2$, $-2 \le y \le 2$,

```
[x,y] = meshgrid([-2:.2:2]);
z = x.*exp(-x.^2 - y.^2);
surf(x,y,z)
```

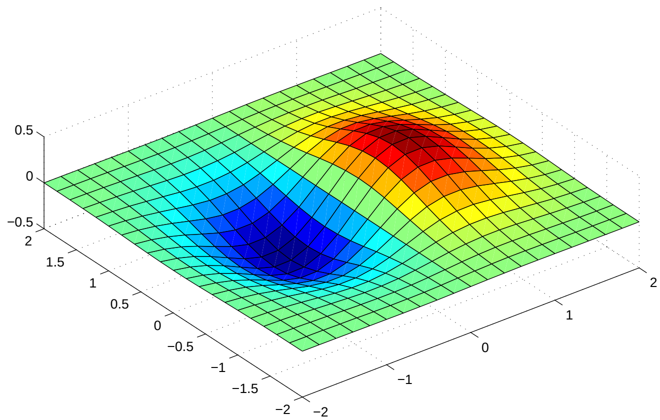


Querying the data aspect ratio shows how MATLAB has drawn the surface.

```
daspect
ans =
     4   4   1
```

Setting the data aspect ratio to [1 1 1] produces a surface plot with equal scaling along each axis.

```
daspect([1 1 1])
```



See Also

axis, pbaspect, xlim, ylim, zlim

The axes properties DataAspectRatio, PlotBoxAspectRatio, XLim, YLim, ZLim

The "Aspect Ratio" section in the *Using MATLAB Graphics* manual.

datetick

Purpose Label tick lines using dates

Syntax

```
datetick(tickaxis)
datetick(tickaxis,dateform)
```

Description `datetick(tickaxis)` labels the tick lines of an axis using dates, replacing the default numeric labels. `tickaxis` is the string `'x'`, `'y'`, or `'z'`. The default is `'x'`. `datetick` selects a label format based on the minimum and maximum limits of the specified axis.

`datetick(tickaxis,dateform)` formats the labels according to the integer `dateform` (see table). To produce correct results, the data for the specified axis must be serial date numbers (as produced by `datenum`).

Dateform	Format	Example
0	day-month-year hour:minute	01-Mar-1995 03:45
1	day-month-year	01-Mar-1995
2	month/day/year	03/01/95
3	month, three letters	Mar
4	month, single letter	M
5	month, numeral	3
6	month/day	03/01
7	day of month	1
8	day of week, three letters	Wed
9	day of week, single letter	W
10	year, four digit	1995
11	year, two digit	95

Dateform	Format	Example
12	month year	Mar95
13	hour:minute:second	15:45:17
14	hour:minute:second AM or PM	03:45:17
15	hour:minute	15:45
16	hour:minute AM or PM	03:45 PM

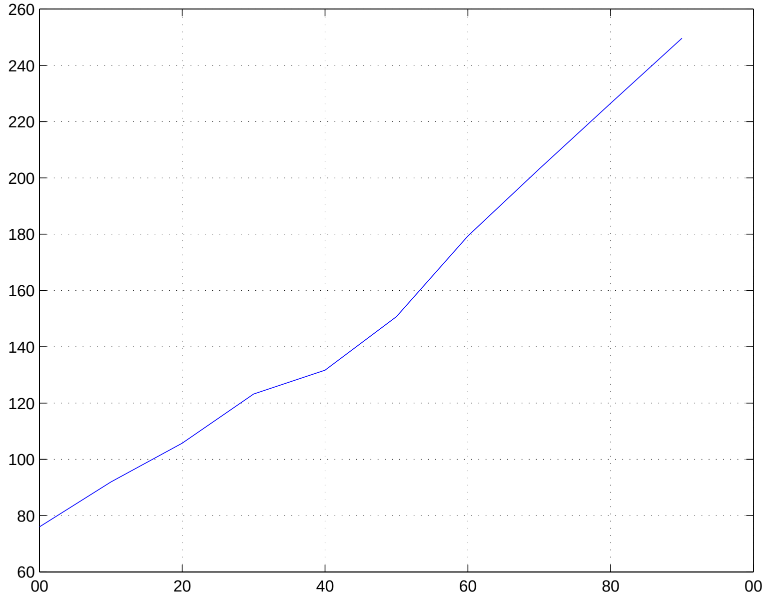
Remarks

`datetick` calls `datestr` to convert date numbers to date strings.

To change the tick spacing and locations, set the appropriate axes property (i.e., `XTick`, `YTick`, or `ZTick`) before calling `datetick`.

Example Consider graphing population data based on the 1990 U.S. census:

```
t = (1900:10:1990)';   % Time interval
p = [75.995 91.972 105.711 123.203 131.669 ...
    150.697 179.323 203.212 226.505 249.633]';  % Population
plot(datenum(t,1,1),p) % Convert years to date numbers and plot
grid on
datetick('x',11)    % Replace x-axis ticks with 2-digit year labels
```



See Also The axes properties XTick, YTick, and ZTick.

datenum, datestr

Purpose MATLAB Version 4.0 figure and axes defaults

Syntax

```
default4
default4(h)
```

Description `default4` sets figure and axes defaults to match MATLAB Version 4.0 defaults.

`default4(h)` only affects the figure with handle `h`.

See Also `colordef`

dialog

Purpose Create and display dialog box

Syntax

```
h = dialog('PropertyName',PropertyValue,...)
```

Description

`h = dialog('`*PropertyName*`',PropertyValue,...)` returns a handle to a dialog box. This function creates a figure graphics object and sets the figure properties recommended for dialog boxes. You can specify any valid figure property value.

See Also

`errordlg`, `figure`, `helpdlg`, `inputdlg`, `pagedlg`, `printdlg`, `questdlg`, `uiwait`, `uiresume`, `warndlg`

Purpose Drag rectangles with mouse

Syntax

```
[finalrect] = dragrect(initialrect)
[finalrect] = dragrect(initialrect,stepsize)
```

Description `[finalrect] = dragrect(initialrect)` tracks one or more rectangles anywhere on the screen. The n-by-4 matrix, `rect`, defines the rectangles. Each row of `rect` must contain the initial rectangle position as `[left bottom width height]` values. `dragrect` returns the final position of the rectangles in `finalrect`.

`[finalrect] = dragrect(initialrect,stepsize)` moves the rectangles in increments of `stepsize`. The lower-left corner of the first rectangle is constrained to a grid of size `stepsize` starting at the lower-left corner of the figure, and all other rectangles maintain their original offset from the first rectangle. `[finalrect] = dragrect(...)` returns the final positions of the rectangles when the mouse button is released. The default stepsize is `1`.

Remarks `dragrect` returns immediately if a mouse button is not currently pressed. Use `dragrect` in a `ButtonDownFcn`, or from the command line in conjunction with `waitforbuttonpress` to ensure that the mouse button is down when `dragrect` is called. `dragrect` returns when you release the mouse button.

Example Drag a rectangle that is 50 pixels wide and 100 pixels in height.

```
waitforbuttonpress
point1 = get(gcf,'CurrentPoint') % button down detected
rect = [point1(1,1) point1(1,2) 50 100]
[r2] = dragrect(rect)
```

See Also `rbbox`, `waitforbuttonpress`

drawnow

Purpose Complete pending drawing events

Syntax

```
drawnow
```

Description `drawnow` flushes the event queue and updates the figure window.

Remarks Other events that cause MATLAB to flush the event queue and draw the figure windows include:

- Returning to the MATLAB prompt
- A `pause` statement
- A `waitforbuttonpress` statement
- A `waitfor` statement
- A `getframe` statement
- A `figure` statement

Examples Executing the statements,

```
x = -pi:pi/20:pi;
plot(x,cos(x))
drawnow
title('A Short Title')
grid on
```

as an M-file updates the current figure after executing the `drawnow` function and after executing the final statement.

See Also `waitfor`, `pause`, `waitforbuttonpress`

Purpose

Plot error bars along a curve

Syntax

```
errorbar(Y,E)
errorbar(X,Y,E)
errorbar(X,Y,L,U)
errorbar(...,LineSpec)
h = errorbar(...)
```

Description

Error bars show the confidence level of data or the deviation along a curve.

`errorbar(Y,E)` plots `Y` and draws an error bar at each element of `Y`. The error bar is a distance of `E(i)` above and below the curve so that each bar is symmetric and `2*E(i)` long.

`errorbar(X,Y,E)` plots `X` versus `Y` with symmetric error bars `2*E(i)` long. `X`, `Y`, `E` must be the same size. When they are vectors, each error bar is a distance of `E(i)` above and below the point defined by `(X(i),Y(i))`. When they are matrices, each error bar is a distance of `E(i,j)` above and below the point defined by `(X(i,j),Y(i,j))`.

`errorbar(X,Y,L,U)` plots `X` versus `Y` with error bars `L(i)+U(i)` long specifying the lower and upper error bars. `X`, `Y`, `L`, and `U` must be the same size. When they are vectors, each error bar is a distance of `L(i)` below and `U(i)` above the point defined by `(X(i),Y(i))`. When they are matrices, each error bar is a distance of `L(i,j)` below and `U(i,j)` above the point defined by `(X(i,j),Y(i,j))`.

`errorbar(...,LineSpec)` draws the error bars using the line type, marker symbol, and color specified by `LineSpec`.

`h = errorbar(...)` returns a vector of handles to line graphics objects.

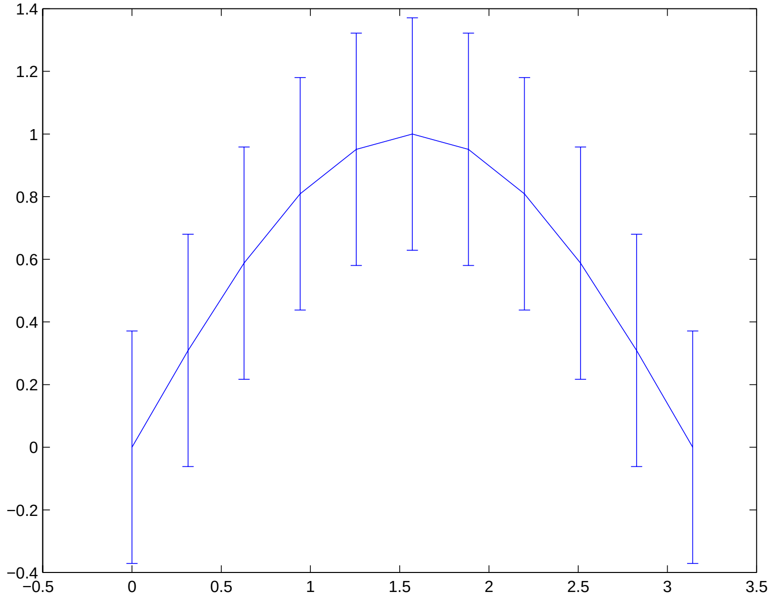
Remarks

When the arguments are all matrices, `errorbar` draws one line per matrix column. If `X` and `Y` are vectors, they specify one curve.

Examples

Draw symmetric error bars that are two standard deviation units in length.

```
X = 0:pi/10:pi;
Y = sin(X);
E = std(Y)*ones(size(X));
errorbar(X,Y,E)
```



See Also

`LineSpec`, `plot`, `std`

Purpose Create and display an error dialog box

Syntax

```
errordlg
errordlg('errorstring')
errordlg('errorstring','dlgname')
errordlg('errorstring','dlgname','on')
h = errordlg(...)
```

Description `errordlg` creates an error dialog box, or if the named dialog exists, `errordlg` pops the named dialog in front of other windows.

`errordlg` displays a dialog box named `'Error Dialog'` that contains the string `'This is the default error string.'`

`errordlg('errorstring')` displays a dialog box named `'Error Dialog'` that contains the string `'errorstring'`.

`errordlg('errorstring','dlgname')` displays a dialog box named `'dlgname'` that contains the string `'errorstring'`.

`errordlg('errorstring','dlgname','on')` specifies whether to replace an existing dialog box having the same name. `'on'` brings an existing error dialog having the same name to the foreground. In this case, `errordlg` does not create a new dialog.

`h = errordlg(...)` returns the handle of the dialog box.

Remarks MATLAB sizes the dialog box to fit the string `'errorstring'`. The error dialog box has an OK pushbutton and remains on the screen until you press the OK button or the **Return** key. After pressing the button, the error dialog box disappears.

The appearance of the dialog box depends on the windowing system you use.

Examples The function

```
errordlg('File not found','File Error');
```

displays this dialog box on a UNIX system:

See Also dialog, helpdlg, msgbox, questdlg, warndlg

Purpose Easy to use contour plotter

Syntax

```
ezcontour(f)
ezcontour(f,domain)
ezcontour(...,n)
```

Description

`ezcontour(f)` plots the contour lines of *f(x,y)*, where `f` is a string that represents a mathematical function of two variables, such as *x* and *y*.

The function *f* is plotted over the default domain: $-2\pi < x < 2\pi$, $-2\pi < y < 2\pi$. MATLAB chooses the computational grid according to the amount of variation that occurs; if the function *f* is not defined (singular) for points on the grid, then these points are not plotted.

`ezcontour(f,domain)` plots *f(x,y)* over the specified `domain`. `domain` can be either a 4-by-1 vector [xmin, xmax, ymin, ymax] or a 2-by-1 vector [min, max] (where min < x < max, min < y < max).

If *f* is a function of the variables *u* and *v* (rather than *x* and *y*), then the domain endpoints umin, umax, vmin, and vmax are sorted alphabetically. Thus, `ezcontour('u^2 - v^3',[0,1],[3,6])` plots the contour lines for $u^2 - v^3$ over $0 < u < 1$, $3 < v < 6$.

`ezcontour(...,n)` plots *f* over the default domain using an `n`-by-`n` grid. The default value for `n` is 60.

`ezcontour` automatically adds a title and axis labels.

Remarks

Array multiplication, division, and exponentiation are always implied in the expression you pass to `ezcontour`. For example, the MATLAB syntax for a contour plot of the expression,

```
sqrt(x.^2 + y.^2)
```

is written as:

```
ezcontour('sqrt(x^2 + y^2)')
```

That is, `x^2` is interpreted as `x.^2` in the string you pass to `ezcontour`.

Examples

The following mathematical expression defines a function of two variables, *x* and *y*.

$$f(x, y) = 3(1-x)^2 e^{-x^2-(y+1)^2} - 10\left(\frac{x}{5} - x^3 - y^5\right) e^{-x^2-y^2} - \frac{1}{3} e^{-(x+1)^2-y^2}$$

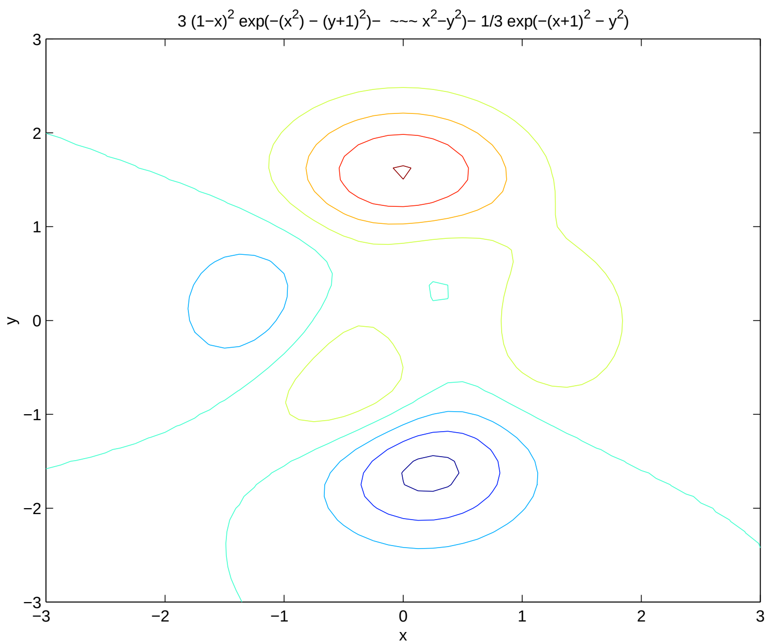
ezcontour requires a string argument that expresses this function using MATLAB syntax to represent exponents, natural logs, etc. This function is represented by the string:

```
f = ['3*(1-x)^2*exp(-(x^2)-(y+1)^2)',...
     '- 10*(x/5 - x^3 - y^5)*exp(-x^2-y^2)',...
     '- 1/3*exp(-(x+1)^2 - y^2)'];
```

For convenience, this string is written on three lines and concatenated into one string using square brackets.

Pass the string variable f to ezcontour along with a domain ranging from −3 to 3 and specify a computational grid of 49-by-49:

```
ezcontour(f,[-3,3],49)
```



In this particular case, the title is too long to fit at the top of the graph so MATLAB abbreviates the string.

See Also

`contour`, `ezcontourf`, `ezmesh`, `ezmeshc`, `ezplot`, `ezplot3`, `ezpolar`, `ezsurf`, `ezsurfc`

ezcontourf

Purpose Easy to use filled contour plotter

Syntax

```
ezcontourf(f)
ezcontourf(f,domain)
ezcontourf(...,n)
```

Description

`ezcontourf(f)` plots the contour lines of *f(x,y)*, where `f` is a string that represents a mathematical function of two variables, such as *x* and *y*.

The function *f* is plotted over the default domain: $-2\pi < x < 2\pi$, $-2\pi < y < 2\pi$. MATLAB chooses the computational grid according to the amount of variation that occurs; if the function *f* is not defined (singular) for points on the grid, then these points are not plotted.

`ezcontourf(f,domain)` plots *f(x,y)* over the specified `domain`. `domain` can be either a 4-by-1 vector [xmin, xmax, ymin, ymax] or a 2-by-1 vector [min, max] (where, min < x < max, min < y < max).

If *f* is a function of the variables *u* and *v* (rather than *x* and *y*), then the domain endpoints umin, umax, vmin, and vmax are sorted alphabetically. Thus, `ezcontourf('u^2 - v^3',[0,1],[3,6])` plots the contour lines for $u^2 - v^3$ over $0 < u < 1$, $3 < v < 6$.

`ezcontourf(...,n)` plots *f* over the default domain using an `n`-by-`n` grid. The default value for `n` is 60.

`ezcontourf` automatically adds a title and axis labels.

Remarks

Array multiplication, division, and exponentiation are always implied in the expression you pass to `ezcontourf`. For example, the MATLAB syntax for a filled contour plot of the expression,

```
sqrt(x.^2 + y.^2);
```

is written as:

```
ezcontourf('sqrt(x^2 + y^2)')
```

That is, `x^2` is interpreted as `x.^2` in the string you pass to `ezcontourf`.

Examples

The following mathematical expression defines a function of two variables, *x* and *y*.

$$f(x, y) = 3(1-x)^2 e^{-x^2-(y+1)^2} - 10\left(\frac{x}{5} - x^3 - y^5\right)e^{-x^2-y^2} - \frac{1}{3}e^{-(x+1)^2-y^2}$$

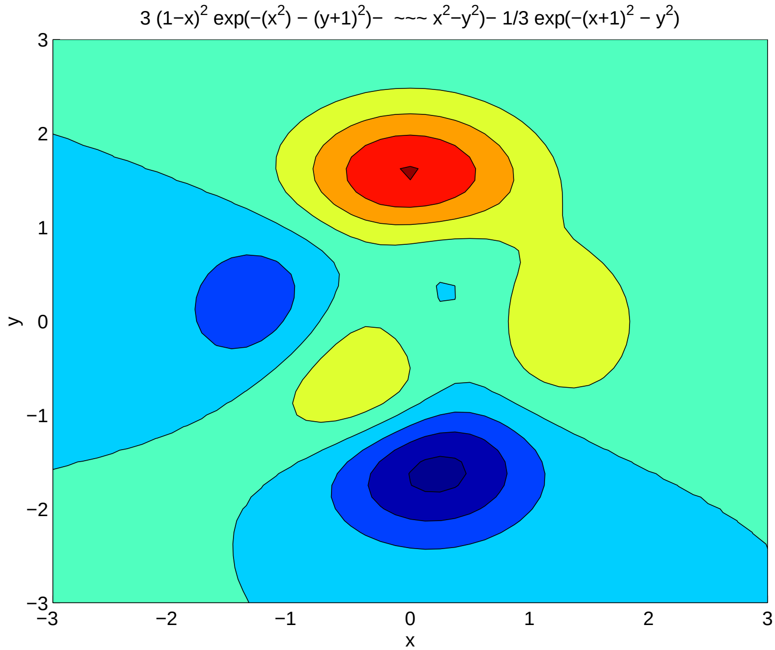
ezcontourf requires a string argument that expresses this function using MATLAB syntax to represent exponents, natural logs, etc. This function is represented by the string:

```
f = ['3*(1–x)^2*exp(–(x^2)–(y+1)^2)',...
      '– 10*(x/5 – x^3 – y^5)*exp(-x^2–y^2)',...
      '- 1/3*exp(–(x+1)^2 – y^2)'];
```

For convenience, this string is written on three lines and concatenated into one string using square brackets.

Pass the string variable f to ezcontourf along with a domain ranging from –3 to 3 and specify a grid of 49-by-49:

```
ezcontourf(f,[-3,3],49)
```



In this particular case, the title is too long to fit at the top of the graph so MATLAB abbreviates the string.

See Also

`contourf`, `ezcontour`, `ezmesh`, `ezmeshc`, `ezplot`, `ezplot3`, `ezpolar`, `ezsurf`, `ezsurfc`

Purpose

Easy to use 3-D mesh plotter

Syntax

```
ezmesh(f)
ezmesh(f,domain)
ezmesh(x,y,z)
ezmesh(x,y,z,[smin,smax,tmin,tmax]) or ezmesh(x,y,z,[min,max])
ezmesh(...,n)
ezmesh(...,'circ')
```

Description

`ezmesh(f)` creates a graph of *f(x,y)*, where `f` is a string that represents a mathematical function of two variables, such as *x* and *y*.

The function *f* is plotted over the default domain: $-2\pi < x < 2\pi$, $-2\pi < y < 2\pi$. MATLAB chooses the computational grid according to the amount of variation that occurs; if the function *f* is not defined (singular) for points on the grid, then these points are not plotted.

`ezmesh(f,domain)` plots *f* over the specified `domain`. `domain` can be either a 4-by-1 vector [xmin, xmax, ymin, ymax] or a 2-by-1 vector [min, max] (where, min < *x* < max, min < *y* < max).

If *f* is a function of the variables *u* and *v* (rather than *x* and *y*), then the domain endpoints umin, umax, vmin, and vmax are sorted alphabetically. Thus, `ezmesh('u^2 - v^3',[0,1],[3,6])` plots $u^2 - v^3$ over $0 < u < 1$, $3 < v < 6$.

`ezmesh(x,y,z)` plots the parametric surface $x = x(s,t)$, $y = y(s,t)$, and $z = z(s,t)$ over the square: $-2\pi < s < 2\pi$, $-2\pi < t < 2\pi$.

`ezmesh(x,y,z,[smin,smax,tmin,tmax])` or `ezmesh(x,y,z,[min,max])` plots the parametric surface using the specified domain.

`ezmesh(...,n)` plots *f* over the default domain using an `n`-by-`n` grid. The default value for `n` is 60.

`ezmesh(...,'circ')` plots *f* over a disk centered on the domain.

Remarks

`rotate3d` is always on. To rotate the graph, click and drag with the mouse.

Array multiplication, division, and exponentiation are always implied in the expression you pass to ezmesh. For example, the MATLAB syntax for a mesh plot of the expression,

```
sqrt(x.^2 + y.^2);
```

is written as:

```
ezmesh('sqrt(x^2 + y^2)')
```

That is, x^2 is interpreted as x.^2 in the string you pass to ezmesh.

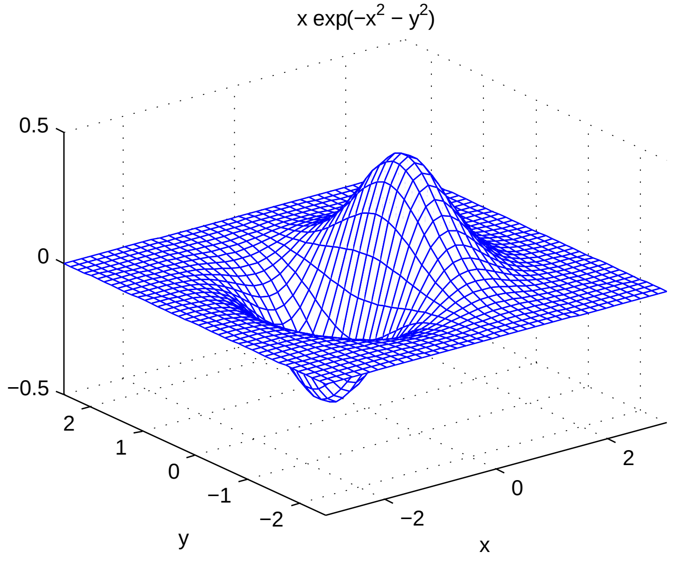
Examples

This example visualizes the function,

$$f(x, y) = xe^{-x^2 - y^2}$$

with a mesh plot drawn on a 40-by-40 grid. The mesh lines are set to a uniform blue color by setting the colormap to a single color:

```
ezmesh('x*exp(-x^2-y^2)',40)
colormap [0 0 1]
```



See Also `ezcontour`, `ezcontourf`, `ezmeshc`, `ezplot`, `ezplot3`, `ezpolar`, `ezsurf`, `ezsurfc`, `mesh`

ezmeshc

Purpose Easy to use combination mesh/contour plotter

Syntax

```
ezmeshc(f)
ezmeshc(f,domain)
ezmeshc(x,y,z)
ezmeshc(x,y,z,[smin,smax,tmin,tmax]) or ezmeshc(x,y,z,[min,max])
ezmeshc(...,n)
ezmeshc(...,'circ')
```

Description

`ezmeshc(f)` creates a graph of $f(x,y)$, where f is a string that represents a mathematical function of two variables, such as x and y.

The function f is plotted over the default domain: $-2\pi < x < 2\pi$, $-2\pi < y < 2\pi$. MATLAB chooses the computational grid according to the amount of variation that occurs; if the function f is not defined (singular) for points on the grid, then these points are not plotted.

`ezmeshc(f,domain)` plots f over the specified `domain`. `domain` can be either a 4-by-1 vector [xmin, xmax, ymin, ymax] or a 2-by-1 vector [min, max] (where, min < x < max, min < y < max).

If f is a function of the variables u and v (rather than x and y), then the domain endpoints umin, umax, vmin, and vmax are sorted alphabetically. Thus, `ezmeshc('u^2 - v^3',[0,1],[3,6])` plots $u^2 - v^3$ over $0 < u < 1$, $3 < v < 6$.

`ezmeshc(x,y,z)` plots the parametric surface $x = x(s,t)$, $y = y(s,t)$, and $z = z(s,t)$ over the square: $-2\pi < s < 2\pi$, $-2\pi < t < 2\pi$.

`ezmeshc(x,y,z,[smin,smax,tmin,tmax])` or `ezmeshc(x,y,z,[min,max])` plots the parametric surface using the specified domain.

`ezmeshc(...,n)` plots f over the default domain using an `n`-by-`n` grid. The default value for `n` is 60.

`ezmeshc(...,'circ')` plots f over a disk centered on the domain.

Remarks

`rotate3d` is always on. To rotate the graph, click and drag with the mouse.

Array multiplication, division, and exponentiation are always implied in the expression you pass to ezmeshc. For example, the MATLAB syntax for a mesh/contour plot of the expression,

```
sqrt(x.^2 + y.^2);
```

is written as:

```
ezmeshc('sqrt(x^2 + y^2)')
```

That is, x^2 is interpreted as x.^2 in the string you pass to ezmeshc.

Examples

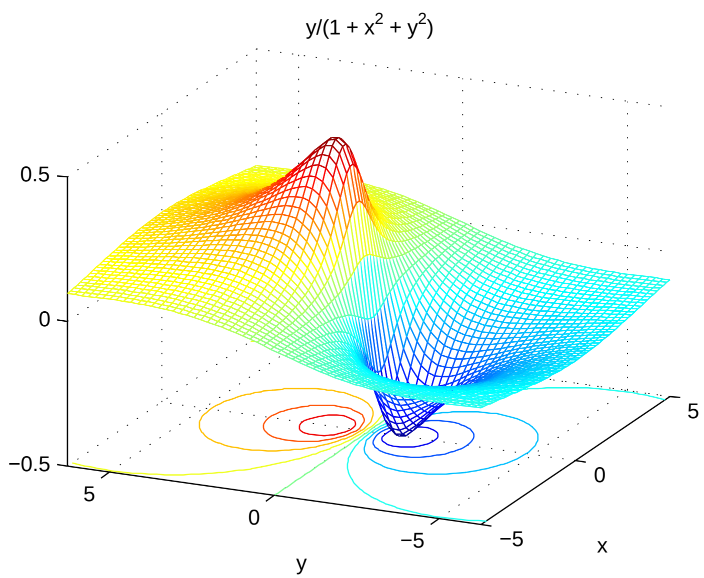
Create a mesh/contour graph of the expression,

$$f(x, y) = \frac{y}{1 + x^2 + y^2}$$

over the domain -5 < x < 5, -2*pi < y < 2*pi:

```
ezmeshc('y/(1 + x^2 + y^2)',[−5,5,−2*pi,2*pi])
```

Use the mouse to rotate the axes to better observe the contour lines (this picture uses a view of azimuth = -65.5 and elevation = 26).



See Also ezcontour, ezcontourf, ezmesh, ezplot, ezplot3, ezpolar, ezsurf, ezsurfc, meshc

Purpose

Easy to use function plotter

Syntax

```
ezplot(f)
ezplot(f,[min,max])
ezplot(f,[xmin,xmax,ymin,ymax])
ezplot(x,y)
ezplot(x,y,[tmin,tmax])
ezplot(...,figure)
```

Description

`ezplot(f)` plots the expression $f = f(x)$ over the default domain: $-2\pi < x < 2\pi$.

`ezplot(f,[min,max])` plots $f = f(x)$ over the domain: `min` < x < `max`.

For implicitly defined functions, $f = f(x,y)$:

`ezplot(f)` plots $f(x,y) = 0$ over the default domain $-2\pi < x < 2\pi$, $-2\pi < y < 2\pi$.

`ezplot(f,[xmin,xmax,ymin,ymax])` plots $f(x,y) = 0$ over `xmin` < x < `xmax` and `ymin` < y < `ymax`.

`ezplot(f,[min,max])`plots $f(x,y) = 0$ over `min` < x < `max` and `min` < y < `max`.

If f is a function of the variables u and v (rather than x and y), then the domain endpoints umin, umax, vmin, and vmax are sorted alphabetically. Thus, `ezplot('u^2 - v^2 - 1',[-3,2,-2,3])` plots $u^2 - v^2 - 1 = 0$ over $-3 < u < 2$, $-2 < v < 3$.

`ezplot(x,y)` plots the parametrically defined planar curve $x = x(t)$ and $y = y(t)$ over the default domain $0 < t < 2\pi$.

`ezplot(x,y,[tmin,tmax])` plots $x = x(t)$ and $y = y(t)$ over `tmin` < t < `tmax`.

`ezplot(...,figure)` plots the given function over the specified domain in the figure window identified by the handle `figure`.

Remarks

Array multiplication, division, and exponentiation are always implied in the expression you pass to `ezplot`. For example, the MATLAB syntax for a plot of the expression,

```
x.^2 - y.^2
```

which represents an implicitly defined function, is written as:

```
ezplot('x^2 - y^2')
```

That is, x^2 is interpreted as x.^2 in the string you pass to ezplot.

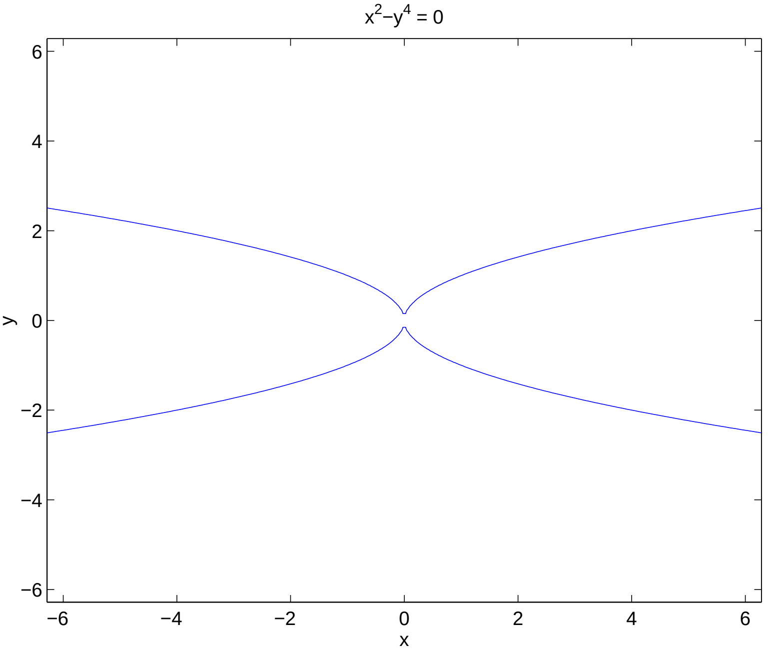
Examples

This example plots the implicitly defined function,

$$x^2 - y^4 = 0$$

over the domain [-2π, 2π]:

```
ezplot('x^2-y^4')
```



See Also

ezcontour, ezcontourf, ezmesh, ezmeshc, ezplot3, ezpolar, ezsurf, ezsurfc, plot

Purpose Easy to use 3-D parametric curve plotter

Syntax

```
ezplot3(x,y,z)
ezplot3(x,y,z,[tmin,tmax])
ezplot3(...,'animate')
```

Description

`ezplot3(x,y,z)` plots the spatial curve $x = x(t)$, $y = y(t)$, and $z = z(t)$ over the default domain $0 < t < 2\pi$.

`ezplot3(x,y,z,[tmin,tmax])` plots the curve $x = x(t)$, $y = y(t)$, and $z = z(t)$ over the domain `tmin` < t < `tmax`.

`ezplot3(...,'animate')` produces an animated trace of the spatial curve.

Remarks

Array multiplication, division, and exponentiation are always implied in the expression you pass to `ezplot3`. For example, the MATLAB syntax for a plot of the expression,

```
x = s./2, y = 2.*s, z = s.^2;
```

which represents a parametric function, is written as:

```
ezplot3('s/2','2*s','s^2')
```

That is, `s/2` is interpreted as `s./2` in the string you pass to `ezplot3`.

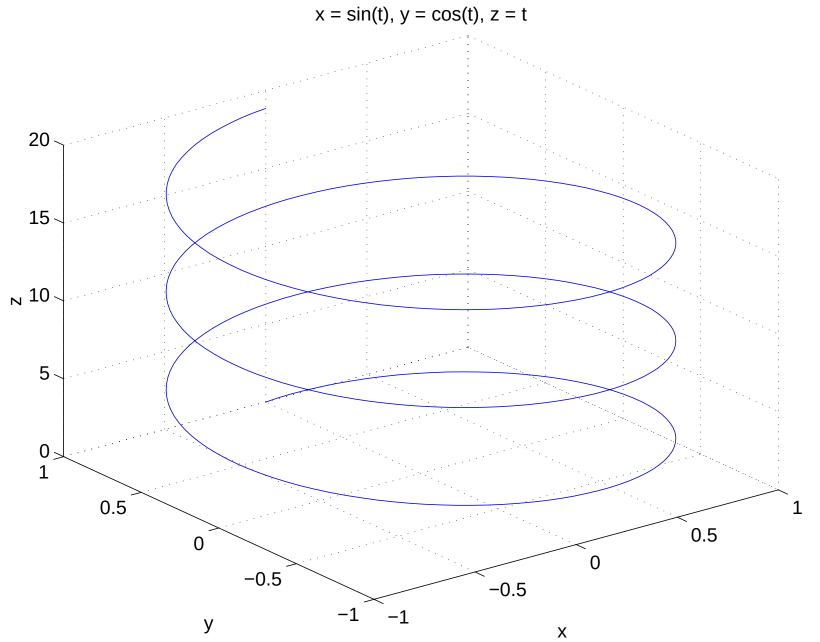
Examples

This example plots the parametric curve,

$$x = \sin t, \quad y = \cos t, \quad z = t$$

over the domain $[0, 6\pi]$:

```
ezplot3('sin(t)','cos(t)','t',[0,6*pi])
```



See Also `ezcontour`, `ezcontourf`, `ezmesh`, `ezmeshc`, `ezplot`, `ezpolar`, `ezsurf`, `ezsurfc`, `plot3`

Purpose

Easy to use polar coordinate plotter

Syntax

```
ezpolar(f)
ezpolar(f,[a,b])
```

Description

`ezpolar(f)` plots the polar curve *rho* = *f(theta)* over the default domain 0 < theta < 2π.

`ezpolar(f,[a,b])` plots *f* for `a` < *theta* < `b`.

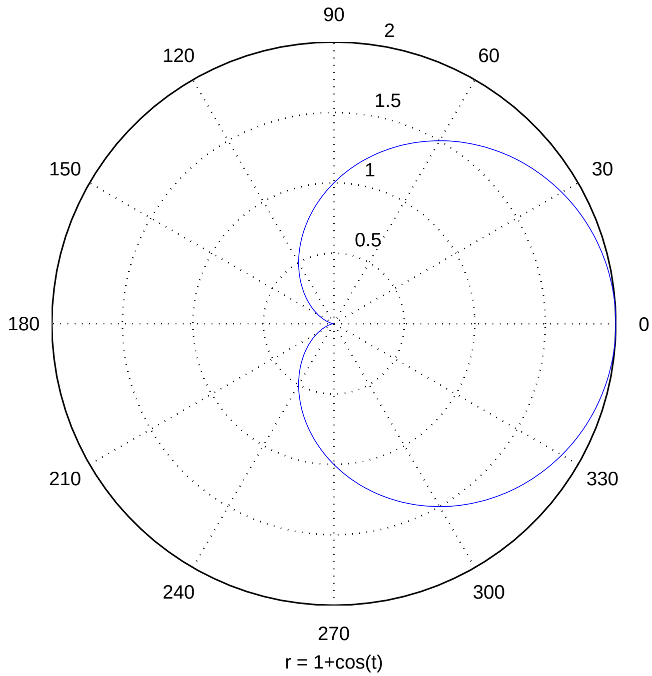
Examples

This example creates a polar plot of the function,

1 + cos(t)

over the domain [0, 2π]:

```
ezpolar('1+cos(t)')
```



See Also

`ezplot`, `ezplot3`, `ezsurf`, `plot`, `plot3`, `polar`

ezsurf

Purpose

Easy to use 3-D colored surface plotter

Syntax

```
ezsurf(f)
ezsurf(f,domain)
ezsurf(x,y,z)
ezsurf(x,y,z,[smin,smax,tmin,tmax]) or ezsurf(x,y,z,[min,max])
ezsurf(...,n)
ezsurf(...,'circ')
```

Description

`ezsurf(f)` creates a graph of $f(x,y)$, where `f` is a string that represents a mathematical function of two variables, such as x and y.

The function f is plotted over the default domain: $-2\pi < x < 2\pi$, $-2\pi < y < 2\pi$. MATLAB chooses the computational grid according to the amount of variation that occurs; if the function f is not defined (singular) for points on the grid, then these points are not plotted.

`ezsurf(f,domain)` plots f over the specified `domain`. `domain` can be either a 4-by-1 vector [xmin, xmax, ymin, ymax] or a 2-by-1 vector [min, max] (where, min < x < max, min < y < max).

If f is a function of the variables u and v (rather than x and y), then the domain endpoints umin, umax, vmin, and vmax are sorted alphabetically. Thus, `ezsurf('u^2 - v^3',[0,1],[3,6])` plots $u^2 - v^3$ over $0 < u < 1$, $3 < v < 6$.

`ezsurf(x,y,z)` plots the parametric surface $x = x(s,t)$, $y = y(s,t)$, and $z = z(s,t)$ over the square: $-2\pi < s < 2\pi$, $-2\pi < t < 2\pi$.

`ezsurf(x,y,z,[smin,smax,tmin,tmax])` or `ezsurf(x,y,z,[min,max])` plots the parametric surface using the specified domain.

`ezsurf(...,n)` plots f over the default domain using an `n`-by-`n` grid. The default value for `n` is 60.

`ezsurf(...,'circ')` plots f over a disk centered on the domain.

Remarks

`rotate3d` is always on. To rotate the graph, click and drag with the mouse.

Array multiplication, division, and exponentiation are always implied in the expression you pass to `ezsurf`. For example, the MATLAB syntax for a surface plot of the expression,

```
sqrt(x.^2 + y.^2);
```

is written as:

```
ezsurf('sqrt(x^2 + y^2)')
```

That is, `x^2` is interpreted as `x.^2` in the string you pass to `ezsurf`.

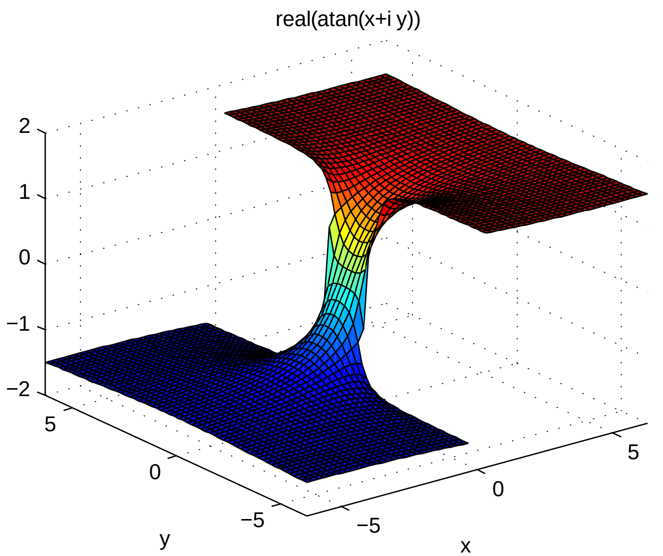
Examples

`ezsurf` does not graph points where the mathematical function is not defined (these data points are set to `NaN`s, which MATLAB does not plot). This example illustrates this filtering of singularities/discontinuous points by graphing the function,

$$f(x, y) = real(atan(x + iy))$$

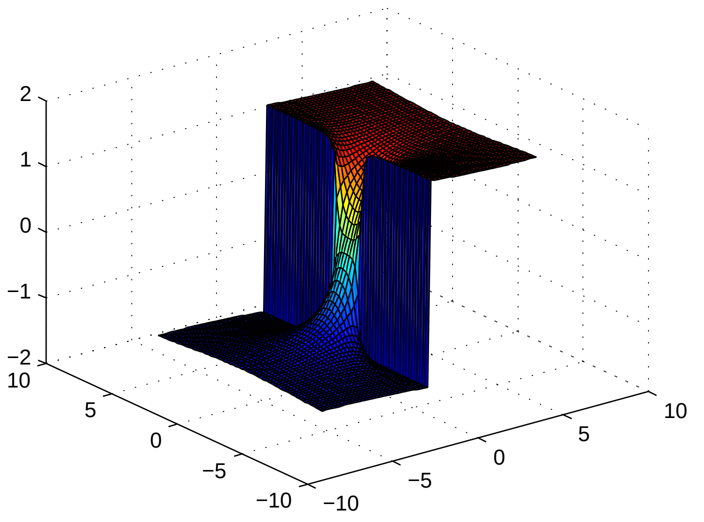
over the default domain $-2\pi < x < 2\pi$, $-2\pi < y < 2\pi$:

```
ezsurf('real(atan(x+i*y))')
```



Using surf to plot the same data produces a graph without filtering of discontinuities (as well as requiring more steps):

```
[x,y] = meshgrid(linspace(-2*pi,2*pi,60));
z = real(atan(x+i.*y));
surf(x,y,z)
```



Note also that ezsurf creates graphs that have axis labels, a title, and extend to the axis limits.

See Also ezcontour, ezcontourf, ezmesh, ezmeshc, ezplot, ezpolar, ezsurfc, surf

Purpose

Easy to use combination surface/contour plotter

Syntax

```
ezsurfc(f)
ezsurfc(f,domain)
ezsurfc(x,y,z)
ezsurfc(x,y,z,[smin,smax,tmin,tmax]) or ezsurfc(x,y,z,[min,max])
ezsurfc(...,n)
ezsurfc(...,'circ')
```

Description

`ezsurfc(f)` creates a graph of *f(x,y)*, where `f` is a string that represents a mathematical function of two variables, such as *x* and *y*.

The function *f* is plotted over the default domain: $-2\pi < x < 2\pi$, $-2\pi < y < 2\pi$. MATLAB chooses the computational grid according to the amount of variation that occurs; if the function *f* is not defined (singular) for points on the grid, then these points are not plotted.

`ezsurfc(f,domain)` plots *f* over the specified `domain`. `domain` can be either a 4-by-1 vector [xmin, xmax, ymin, ymax] or a 2-by-1 vector [min, max] (where, $min < x < max$, $min < y < max$).

If *f* is a function of the variables *u* and *v* (rather than *x* and *y*), then the domain endpoints umin, umax, vmin, and vmax are sorted alphabetically. Thus, `ezsurfc('u^2 - v^3',[0,1],[3,6])` plots $u^2 - v^3$ over $0 < u < 1$, $3 < v < 6$.

`ezsurfc(x,y,z)` plots the parametric surface $x = x(s,t)$, $y = y(s,t)$, and $z = z(s,t)$ over the square: $-2\pi < s < 2\pi$, $-2\pi < t < 2\pi$.

`ezsurfc(x,y,z,[smin,smax,tmin,tmax])` or `ezsurfc(x,y,z,[min,max])` plots the parametric surface using the specified domain.

`ezsurfc(...,n)` plots *f* over the default domain using an `n`-by-`n` grid. The default value for `n` is 60.

`ezsurfc(...,'circ')` plots *f* over a disk centered on the domain.

Remarks

`rotate3d` is always on. To rotate the graph, click and drag with the mouse.

Array multiplication, division, and exponentiation are always implied in the expression you pass to `ezsurfc`. For example, the MATLAB syntax for a surface/contour plot of the experssion,

```
sqrt(x.^2 + y.^2);
```

is written as:

```
ezsurfc('sqrt(x^2 + y^2)')
```

That is, `x^2` is interpreted as `x.^2` in the string you pass to `ezsurfc`.

Examples

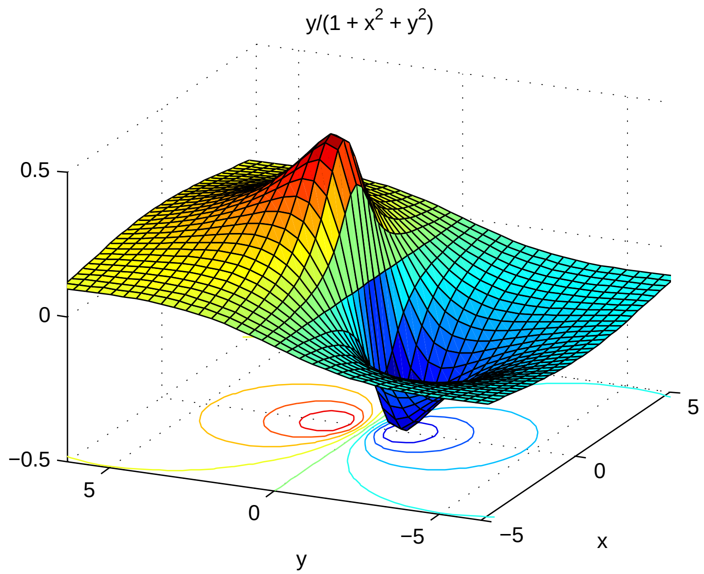
Create a surface/contour plot of the expression,

$$f(x, y) = \frac{y}{1 + x^2 + y^2}$$

over the domain -5 < x < 5, -2*pi < y < 2*pi, with a computational grid of size 35-by-35:

```
ezsurfc('y/(1 + x^2 + y^2)',[–5,5,–2*pi,2*pi],35)
```

Use the mouse to rotate the axes to better observe the contour lines (this picture uses a view of azimuth = -65.5 and elevation = 26)



See Also `ezcontour`, `ezcontourf`, `ezmesh`, `ezmeshc`, `ezplot`, `ezpolar`, `ezsurf`, `surfc`

Purpose

Plot velocity vectors

Syntax

```
feather(U,V)
feather(Z)
feather(...,LineSpec)
```

Description

A feather plot displays vectors emanating from equally spaced points along a horizontal axis. You express the vector components relative to the origin of the respective vector.

`feather(U,V)` displays the vectors specified by `U` and `V`, where `U` contains the *x* components as relative coordinates, and `V` contains the *y* components as relative coordinates.

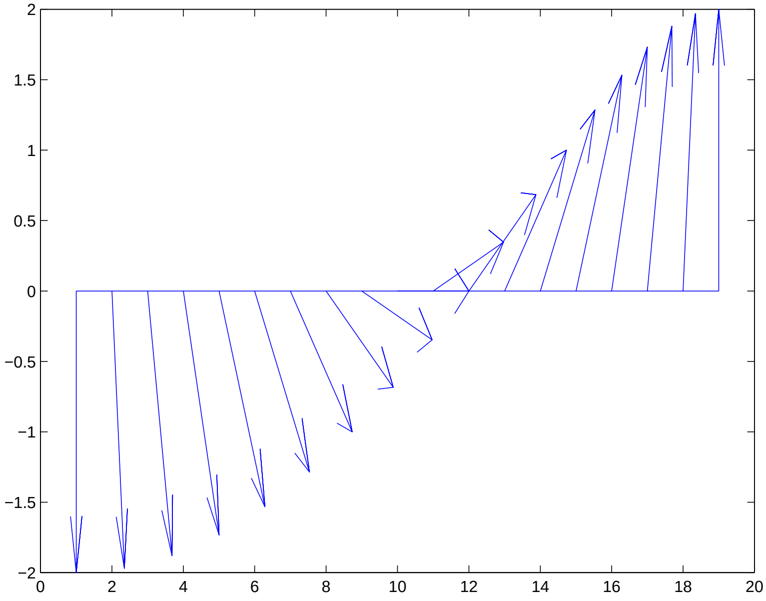
`feather(Z)` displays the vectors specified by the complex numbers in `Z`. This is equivalent to `feather(real(Z),imag(Z))`.

`feather(...,LineSpec)` draws a feather plot using the line type, marker symbol, and color specified by `LineSpec`.

Examples

Create a feather plot showing the direction of `theta`.

```
theta = (-90:10:90)*pi/180;
r = 2*ones(size(theta));
[u,v] = pol2cart(theta,r);
feather(u,v);
```



See Also

`compass`, `LineSpec`, `rose`

figflag

Purpose

Test if figure is on screen

Syntax

```
[flag] = figflag('figurename')
[flag,fig] = figflag('figurename')
[...] = figflag('figurename',silent)
```

Description

Use `figflag` to determine if a particular figure exists, bring a figure to the foreground, or set the window focus to a figure.

`[flag] = figflag('`*`figurename`*`')` returns a `1` if the figure named `'`*`figurename`*`'` exists and pops the figure to the foreground; otherwise this function returns `0`.

`[flag,fig] = figflag('`*`figurename`*`')` returns a `1` in `flag`, returns the figure's handle in `fig`, and pops the figure to the foreground, if the figure named `'`*`figurename`*`'` exists. Otherwise this function returns `0`.

`[...] = figflag('`*`figurename`*`',silent)` pops the figure window to the foreground if `silent` is `0`, and leaves the figure in its current position if `silent` is `1`.

Examples

To determine if a figure window named `'Fluid Jet Simulation'` exists, type

```
[flag,fig] = figflag('Fluid Jet Simulation')
```

MATLAB returns:

```
flag =
    1
fig =
    1
```

If two figures with handles 1 and 3 have the name `'Fluid Jet Simulation'`, MATLAB returns:

```
flag =
    1
fig =
    1 3
```

See Also

`figure`

Purpose Create a figure graphics object

Syntax

```
figure
figure('PropertyName',PropertyValue,...)
figure(h)
h = figure(...)
```

Description `figure` creates figure graphics objects. figure objects are the individual windows on the screen in which MATLAB displays graphical output.

`figure` creates a new figure object using default property values.

`figure('`*PropertyName*`',PropertyValue,...)` creates a new figure object using the values of the properties specified. MATLAB uses default values for any properties that you do not explicitly define as arguments.

`figure(h)` does one of two things, depending on whether or not a figure with handle `h` exists. If `h` is the handle to an existing figure, `figure(h)` makes the figure identified by `h` the current figure, makes it visible, and raises it above all other figures on the screen. The current figure is the target for graphics output. If `h` is not the handle to an existing figure, but is an integer, `figure(h)` creates a figure, and assigns it the handle `h`. `figure(h)` where `h` is not the handle to a figure, and is not an integer, is an error.

`h = figure(...)` returns the handle to the figure object.

Remarks To create a figure object, MATLAB creates a new window whose characteristics are controlled by default figure properties (both factory installed and user defined) and properties specified as arguments. See the properties section for a description of these properties.

You can specify properties as property name/property value pairs, structure arrays, and cell arrays (see the `set` and `get` reference pages for examples of how to specify these data types).

Use `set` to modify the properties of an existing figure or `get` to query the current values of figure properties.

The `gcf` command returns the handle to the current figure and is useful as an argument to the `set` and `get` commands.

Example

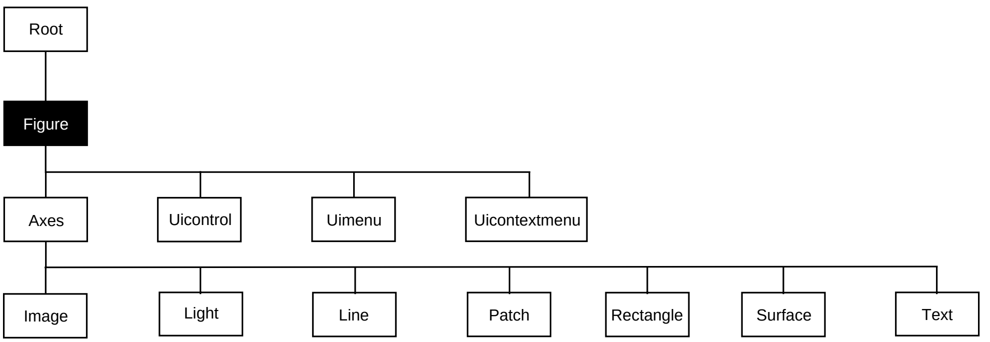
To create a figure window that is one quarter the size of your screen and is positioned in the upper-left corner, use the root object's `ScreenSize` property to determine the size. `ScreenSize` is a four-element vector: [`left`, `bottom`, `width`, `height`]:

```
scrsz = get(0,'ScreenSize');
figure('Position',[1 scrsz(4)/2 scrsz(3)/2 scrsz(4)/2])
```

See Also

`axes`, `uicontrol`, `uimenu`, `close`, `clf`, `gcf`, `rootobject`

Object Hierarchy



Setting Default Properties

You can set default figure properties only on the root level.

```
set(0,'DefaultFigureProperty',PropertyValue...)
```

Where *Property* is the name of the figure property and `PropertyValue` is the value you are specifying. Use `set` and `get` to access figure properties.

Property List

The following table lists all figure properties and provides a brief description of each. The property name links bring you an expanded description of the properties.

Property Name	Property Description	Property Value
Positioning the Figure		
Position	Location and size of figure	Value: a 4-element vector [left, bottom, width, height] Default: depends on display
Units	Units used to interpret the Position property	Values: inches, centimeters, normalized, points, pixels, characters Default: pixels
Specifying Style and Appearance		
Color	Color of the figure background	Values: ColorSpec Default: depends on color scheme (see colordef)
MenuBar	Toggle the figure menu bar on and off	Values: none, figure Default: figure
Name	Figure window title	Values: string Default: '' (empty string)
NumberTitle	Display "Figure No. n", where n is the figure number	Values: on, off Default: on
Resize	Specify whether the figure window can be resized using the mouse	Values: on, off Default: on
SelectionHighlight	Highlight figure when selected (Selected property set to on)	Values: on, off Default: on
Visible	Make the figure visible or invisible	Values: on, off Default: on
WindowStyle	Select normal or modal window	Values: normal, modal Default: normal
Controlling the Colormap		

Property Name	Property Description	Property Value
Colormap	The figure colormap	Values: m-by-3 matrix of RGB values Default: the jet colormap
Dithermap	Colormap used for truecolor data on pseudocolor displays	Values: m-by-3 matrix of RGB values Default: colormap with full range of colors
DithermapMode	Enable MATLAB-generated dithermap	Values: auto, manual Default: manual
FixedColors	Colors not obtained from colormap	Values: m-by-3 matrix of RGB values (read only)
MinColormap	Minimum number of system color table entries to use	Values: scalar Default: 64
ShareColors	Allow MATLAB to share system color table slots	Values on, off Default: on
Specifying the Renderer		
BackingStore	Enable off screen pixel buffering	Values: on, off Default: on
DoubleBuffer	Flash-free rendering for simple animations	Values: on, off Default: off
Renderer	Rendering method used for screen and printing	Values: painters, zbuffer, OpenGL Default: automatic selection by MATLAB
General Information About the Figure		
Children	Handle of any uicontrol, uimenu, and uicontextmenu objects displayed in the figure	Values: vector of handles

Property Name	Property Description	Property Value
Parent	The root object is the parent of all figures	Value: always 0
Selected	Indicate whether figure is in a "selected" state.	Values: on, off Default: on
Tag	User-specified label	Value: any string Default: '' (empty string)
Type	The type of graphics object (read only)	Value: the string 'figure'
UserData	User-specified data	Values: any matrix Default: [] (empty matrix)
RendererMode	Automatic or user-selected renderer	Values: auto, manual Default: auto
Information About Current State		
CurrentAxes	Handle of the current axes in this figure	Values: axes handle
CurrentCharacter	The last key pressed in this figure	Values: single character (read only)
CurrentObject	Handle of the current object in this figure	Values: graphics object handle
CurrentPoint	Location of the last button click in this figure	Values: 2-element vector [x-coord, y-coord]
SelectionType	Mouse selection type (read only)	Values: normal, extended, alt, open
Callback Routine Execution		
BusyAction	Specify how to handle callback routine interruption	Values: cancel, queue Default: queue

Property Name	Property Description	Property Value
ButtonDownFcn	Define a callback routine that executes when a mouse button is pressed on an unoccupied spot in the figure	Values: string Default: empty string
CloseRequestFcn	Define a callback routine that executes when you call the close command	Values: string Default: empty string
CreateFcn	Define a callback routine that executes when a figure is created	Values: string Default: empty string
DeleteFcn	Define a callback routine that executes when the figure is deleted (via close or delete)	Values: string Default: empty string
Interruptible	Determine if callback routine can be interrupted	Values: on, off Default: on (can be interrupted)
KeyPressFcn	Define a callback routine that executes when a key is pressed in the figure window	Values: string Default: empty string
ResizeFcn	Define a callback routine that executes when the figure is resized	Values: string Default: empty string
UIContextMenu	Associate a context menu with the figure	Values: handle of a Uicontrextmenu
WindowButtonDownFcn	Define a callback routine that executes when you press the mouse button down in the figure	Values: string Default: empty string
WindowButtonMotionFcn	Define a callback routine that executes when you move the pointer in the figure	Values: string Default: empty string

Property Name	Property Description	Property Value
WindowButtonUpFcn	Define a callback routine that executes when you release the mouse button	Values: string Default: empty string
Controlling Access to Objects		
IntegerHandle	Specify integer or noninteger figure handle	Values: on, off Default: on (integer handle)
HandleVisibility	Determine if figure handle is visible to users or not	Values: on, callback, off Default: on
HitTest	Determine if the figure can become the current object (see the figure CurrentObject property)	Values: on, off Default: on
NextPlot	Determine how to display additional graphics to this figure	Values: add, replace, replacechildren Default: add
Defining the Pointer		
Pointer	Select the pointer symbol	Values: crosshair, arrow, watch, topl, topr, botl, botr, circle, cross, fleur, left, right, top, bottom, fullcrosshair, ibeam, custom Default: arrow
PointerShapeCData	Data that defines the pointer	Values: 16-by-16 matrix Default: set Pointer to custom and see
PointerShapeHotSpot	Specify the pointer active spot	Values: 2-element vector [row, column] Default: [1,1]
Properties That Affect Printing		

Property Name	Property Description	Property Value
InvertHardcopy	Change figure colors for printing	Values: on, off Default: on
PaperOrientation	Horizontal or vertical paper orientation	Values: portrait, landscape Default: portrait
PaperPosition	Control positioning figure on printed page	Values: 4-element vector [left, bottom, width, height]
PaperPositionMode	Enable WYSIWYG printing of figure	Values: auto, manual Default: manual
PaperSize	Size of the current PaperType specified in PaperUnits (read only)	Values: [width, height]
PaperType	Select from standard paper sizes	Values: see property description Default: usletter
PaperUnits	Units used to specify the PaperSize and PaperPosition	Values: normalized, inches, centimeters, points Default: inches
Controlling the XWindows Display (UNIX only)		
XDisplay	Specify display for MATLAB	Values: display identifier Default: :0.0
XVisual	Select visual used by MATLAB	Values: visual ID
XVisualMode	Auto or manual selection of visual	Values: auto, manual Default: auto

Figure Properties

This section lists property names along with the type of values each accepts. Curly braces { } enclose default values.

BackingStore {on} | off

Off screen pixel buffer. When BackingStore is on, MATLAB stores a copy of the figure window in an off-screen pixel buffer. When obscured parts of the figure window are exposed, MATLAB copies the window contents from this buffer rather than regenerating the objects on the screen. This increases the speed with which the screen is redrawn.

While refreshing the screen quickly is generally desirable, the buffers required do consume system memory. If memory limitations occur, you can set BackingStore to off to disable this feature and release the memory used by the buffers. If your computer does not support backingstore, setting the BackingStore property results in a warning message, but has no other effect.

Setting BackingStore to off can increase the speed of animations because it eliminates the need to draw into both an off-screen buffer and the figure window.

BusyAction cancel | {queue}

Callback routine interruption. The BusyAction property enables you to control how MATLAB handles events that potentially interrupt executing callback routines. If there is a callback routine executing, subsequently invoked callback routines always attempt to interrupt it. If the Interruptible property of the object whose callback is executing is set to on (the default), then interruption occurs at the next point where the event queue is processed. If the Interruptible property is off, the BusyAction property (of the object owning the executing callback) determines how MATLAB handles the event. The choices are:

- cancel – discard the event that attempted to execute a second callback routine.
- queue – queue the event that attempted to execute a second callback routine until the current callback finishes.

ButtonDownFcn string

Button press callback function. A callback routine that executes whenever you press a mouse button while the pointer is in the figure window, but not over a child object (i.e., uicontrol, axes, or axes child). Define this routine as a string that is a valid MATLAB expression or the name of an M-file. The expression executes in the MATLAB workspace.

Children vector of handles

Children of the figure. A vector containing the handles of all axes, uicontrol, and uimenu objects displayed within the figure. You can change the order of the handles and thereby change the stacking of the objects on the display.

Clipping {on} | off

This property has no effect on figures.

CloseRequestFcn string

Function executed on figure close. This property defines a function that MATLAB executes whenever you issue the close command (either a close(figure_handle) or a close all), when you close a figure window from the computer's window manager menu, or when you quit MATLAB.

The CloseRequestFcn provides a mechanism to intervene in the closing of a figure. It allows you to, for example, display a dialog box to ask a user to confirm or cancel the close operation or to prevent users from closing a figure that contains a GUI.

The basic mechanism is:

- A user issues the close command from the command line, by closing the window from the computer's window manager menu, or by quiting MATLAB.
- The close operation executes the function defined by the figure CloseRequestFcn. The default function is named closereq and is predefined as:

```
delete(get(0,'CurrentFigure'))
```

This statement unconditionally deletes the current figure, destroying the window. closereq takes advantage of the fact that the close command makes all figures specified as arguments the current figure before calling the respective close request function.

You can set CloseRequestFcn to any string that is a valid MATLAB statement, including the name of an M-file. For example,

```
set(gcf,'CloseRequestFcn','disp(''This window is immortal'')')
```

This close request function never closes the figure window; it simply echoes "This window is immortal" on the command line. Unless the close request function calls `delete`, MATLAB never closes the figure. (Note that you can always call `delete(`*figure_handle*`)` from the command line if you have created a window with a nondestructive close request function.)

A more useful application of the close request function is to display a question dialog box asking the user to confirm the close operation. The following M-file illustrates how to do this.

```
% my_closereq
% User-defined close request function
% to display a question dialog box

selection = questdlg('Close Specified Figure?',...
                     'Close Request Function',...
                     'Yes','No','Yes');
switch selection,
   case 'Yes',
     delete(gcf)
   case 'No'
     return
end
```

Now assign this M-file to the `CloseRequestFcn` of a figure:

```
set(figure_handle,'CloseRequestFcn','my_closereq')
```

To make this M-file your default close request function, set a default value on the root level.

```
set(0,'DefaultFigureCloseRequestFcn','my_closereq')
```

MATLAB then uses this setting for the `CloseRequestFcn` of all subsequently created figures.

Color `ColorSpec`

Background color. This property controls the figure window background color. You can specify a color using a three-element vector of RGB values or one of MATLAB's predefined names. See `ColorSpec` for more information.

Colormap m-by-3 matrix of RGB values

Figure colormap. This property is an m-by-3 array of red, green, and blue (RGB) intensity values that define m individual colors. MATLAB accesses colors by their row number. For example, an index of 1 specifies the first RGB triplet, an index of 2 specifies the second RGB triplet, and so on. Colormaps can be any length (up to 256 only on MS-Windows), but must be three columns wide. The default figure colormap contains 64 predefined colors.

Colormaps affect the rendering of surface, image, and patch objects, but generally do not affect other graphics objects. See `colormap` and `ColorSpec` for more information.

CreateFcn string

Callback routine executed during object creation. This property defines a callback routine that executes when MATLAB creates a figure object. You must define this property as a default value for figures. For example, the statement,

```
set(0,'DefaultFigureCreateFcn',...
      'set(gcbo,''IntegerHandle'',''off'')')
```

defines a default value on the root level that causes the created figure to use noninteger handles whenever you (or MATLAB) create a figure. MATLAB executes this routine after setting all properties for the figure. Setting this property on an existing figure object has no effect.

The handle of the object whose `CreateFcn` is being executed is accessible only through the root `CallbackObject` property, which you can query using `gcbo`.

CurrentAxes handle of current axes

Target axes in this figure. MATLAB sets this property to the handle of the figure's current axes (i.e., the handle returned by the `gca` command when this figure is the current figure). In all figures for which axes children exist, there is always a current axes. The current axes does not have to be the topmost axes, and setting an axes to be the `CurrentAxes` does not restack it above all other axes.

You can make an axes current using the `axes` and `set` commands. For example, `axes(`*axes_handle*`)` and `set(gcf,'CurrentAxes',`*axes_handle*`)` both make the axes identified by the handle *axes_handle* the current axes. In addition, `axes(`*axes_handle*`)` restacks the axes above all other axes in the figure.

If a figure contains no axes, `get(gcf,'CurrentAxes')` returns the empty matrix. Note that the `gca` function actually creates an axes if one does not exist.

`CurrentCharacter` single character (read only)

Last key pressed. MATLAB sets this property to the last key pressed in the figure window. `CurrentCharacter` is useful for obtaining user input.

`CurrentMenu` (Obsolete)

This property produces a warning message when queried. It has been superseded by the root `CallbackObject` property.

`CurrentObject` object handle

Handle of current object. MATLAB sets this property to the handle of the object that is under the current point (see the `CurrentPoint` property). This object is the front-most object in the stacking order. You can use this property to determine which object a user has selected. The function `gco` provides a convenient way to retrieve the `CurrentObject` of the `CurrentFigure`.

`CurrentPoint` two-element vector: [x-coordinate, y-coordinate]

Location of last button click in this figure. MATLAB sets this property to the location of the pointer at the time of the most recent mouse button press. MATLAB updates this property whenever you press the mouse button while the pointer is in the figure window.

In addition, MATLAB updates `CurrentPoint` before executing callback routines defined for the figure `WindowButtonMotionFcn` and `WindowButtonUpFcn` properties. This enables you to query `CurrentPoint` from these callback routines. It behaves like this:

- If there is no callback routine defined for the `WindowButtonMotionFcn` or the `WindowButtonUpFcn`, then MATLAB updates the `CurrentPoint` only when the mouse button is pressed down within the figure window.
- If there is a callback routine defined for the `WindowButtonMotionFcn`, then MATLAB updates the `CurrentPoint` just before executing the callback. Note that the `WindowButtonMotionFcn` executes only within the figure window unless the mouse button is pressed down within the window and then held down while the pointer is moved around the screen. In this case, the routine

executes (and the CurrentPoint is updated) anywhere on the screen until the mouse button is released.

- If there is a callback routine defined for the WindowButtonUpFcn, MATLAB updates the CurrentPoint just before executing the callback. Note that the WindowButtonUpFcn executes only while the pointer is within the figure window unless the mouse button is pressed down initially within the window. In this case, releasing the button anywhere on the screen triggers callback execution, which is preceded by an update of the CurrentPoint.

The figure CurrentPoint is updated only when certain events occur, as previously described. In some situations, (such as when the WindowButtonMotionFcn takes a long time to execute and the pointer is moved very rapidly) the CurrentPoint may not reflect the actual location of the pointer, but rather the location at the time when the WindowButtonMotionFcn began execution.

The CurrentPoint is measured from the lower-left corner of the figure window, in units determined by the Units property.

The root PointerLocation property contains the location of the pointer updated synchronously with pointer movement. However, the location is measured with respect to the screen, not a figure window.

See uicontrol for information on how this property is set when you click on a uicontrol object.

DeleteFcn string

Delete figure callback routine. A callback routine that executes when the figure object is deleted (e.g., when you issue a delete or a close command). MATLAB executes the routine before destroying the object's properties so these values are available to the callback routine.

The handle of the object whose DeleteFcn is being executed is accessible only through the root CallbackObject property, which you can query using gcbo.

Dithermap m-by-3 matrix of RGB values

Colormap used for true-color data on pseudocolor displays. This property defines a colormap that MATLAB uses to dither true-color CData for display on pseudocolor (8-bit or less) displays. MATLAB maps each RGB color defined as true-color CData to the closest color in the dithermap. The default Dithermap contains colors that span the full spectrum so any color values map reasonably well.

However, if the true-color data contains a wide range of shades in one color, you may achieve better results by defining your own dithermap. See the `DithermapMode` property.

`DithermapMode` `auto | {manual}`

MATLAB generated dithermap. In `manual` mode, MATLAB uses the colormap defined in the `Dithermap` property to display direct color on pseudocolor displays. When `DithermapMode` is `auto`, MATLAB generates a dithermap based on the colors currently displayed. This is useful if the default dithermap does not produce satisfactory results.

The process of generating the dithermap can be quite time consuming and is repeated whenever MATLAB re-renders the display (e.g., when you add a new object or resize the window). You can avoid unnecessary regeneration by setting this property back to `manual` and save the generated dithermap (which MATLAB loaded into the `Dithermap` property).

`DoubleBuffer` `on | {off}`

Flash-free rendering for simple animations. Double buffering is the process of drawing to an off-screen pixel buffer and then blitting the buffer contents to the screen once the drawing is complete. Double buffering generally produces flash-free rendering for simple animations (such as those involving lines, as opposed to objects containing large numbers of polygons). Use double buffering with the animated objects' `EraseMode` property set to `normal`. Use the `set` command to enable double buffering.

```
set(figure_handle,'DoubleBuffer','on')
```

Double buffering works only when the figure `Renderer` property is set to `painters`.

`FixedColors` m-by-3 matrix of RGB values (read only)

Non-colormap colors. Fixed colors define all colors appearing in a figure window that are not obtained from the figure colormap. These colors include axis lines and labels, the color of line, text, uicontrol, and uimenu objects, and any colors that you explicitly define, for example, with a statement like:

```
set(gcf,'Color',[0.3,0.7,0.9]).
```

Fixed color definitions reside in the system color table and do not appear in the figure colormap. For this reason, fixed colors can limit the number of simultaneously displayed colors if the number of fixed colors plus the number of entries in the figure colormap exceed your system's maximum number of colors.

(See the root `ScreenDepth` property for information on determining the total number of colors supported on your system. See the `MinColorMap` and `ShareColors` properties for information on how MATLAB shares colors between applications.)

`HandleVisibility` `{on} | callback | off`

Control access to object's handle by command-line users and GUIs. This property determines when an object's handle is visible in its parent's list of children. `HandleVisibility` is useful for preventing command-line users from accidentally drawing into or deleting a figure that contains only user interface devices (such as a dialog box).

Handles are always visible when `HandleVisibility` is `on`.

Setting `HandleVisibility` to `callback` causes handles to be visible from within callback routines or functions invoked by callback routines, but not from within functions invoked from the command line. This provides a means to protect GUIs from command-line users, while allowing callback routines to have complete access to object handles.

Setting `HandleVisibility` to `off` makes handles invisible at all times. This may be necessary when a callback routine invokes a function that might potentially damage the GUI (such as evaluating a user-typed string), and so temporarily hides its own handles during the execution of that function.

When a handle is not visible in its parent's list of children, it cannot be returned by functions that obtain handles by searching the object hierarchy or querying handle properties. This includes `get`, `findobj`, `gca`, `gcf`, `gco`, `newplot`, `cla`, `clf`, and `close`.

When a handle's visibility is restricted using `callback` or `off`, the object's handle does not appear in its parent's `Children` property, figures do not appear in the root's `CurrentFigure` property, objects do not appear in the root's `CallbackObject` property or in the figure's `CurrentObject` property, and axes do not appear in their parent's `CurrentAxes` property.

You can set the root ShowHiddenHandles property to on to make all handles visible, regardless of their HandleVisibility settings (this does not affect the values of the HandleVisibility properties).

Handles that are hidden are still valid. If you know an object's handle, you can set and get its properties, and pass it to any function that operates on handles.

HitTest {on} | off

Selectable by mouse click. HitTest determines if the figure can become the current object (as returned by the gco command and the figure CurrentObject property) as a result of a mouse click on the figure. If HiTest is off, clicking on the figure sets the CurrentObject to the empty matrix.

IntegerHandle {on} | off

Figure handle mode. Figure object handles are integers by default. When creating a new figure, MATLAB uses the lowest integer that is not used by an existing figure. If you delete a figure, its integer handle can be reused.

If you set this property to off, MATLAB assigns nonreusable real-number handles (e.g., 67.0001221) instead of integers. This feature is designed for dialog boxes where removing the handle from integer values reduces the likelihood of inadvertently drawing into the dialog box.

Interruptible {on} | off

Callback routine interruption mode. The Interruptible property controls whether a figure callback routine can be interrupted by subsequently invoked callback routines. Only callback routines defined for the ButtonDownFcn, KeyPressFcn, WindowButtonDownFcn, WindowButtonMotionFcn, and WindowButtonUpFcn are affected by the Interruptible property. MATLAB checks for events that can interrupt a callback routine only when it encounters a drawnow, figure, getframe, or pause command in the routine. See the BusyAction property for related information.

InvertHardcopy {on} | off

Change hardcopy to black objects on white background. This property affects only printed output. Printing a figure having a background color (Color property) that is not white results in poor contrast between graphics objects and the figure background and also consumes a lot of printer toner.

When InvertHardCopy is on, MATLAB eliminates this effect by changing the color of the figure and axes to white and the axis lines, tick marks, axis labels, etc., to black. lines, text, and the edges of patches and surfaces may be changed depending on the `print` command options specified.

If you set `InvertHardCopy` to `off`, the printed output matches the colors displayed on the screen.

See `print` for more information on printing MATLAB figures.

KeyPressFcn string

Key press callback function. A callback routine invoked by a key press occurring in the figure window. You can define `KeyPressFcn` as any legal MATLAB expression or the name of an M-file.

The callback routine can query the figure's `CurrentCharacter` property to determine what particular key was pressed and thereby limit the callback execution to specific keys.

The callback routine can also query the root `PointerWindow` property to determine in which figure the key was pressed. Note that pressing a key while the pointer is in a particular figure window does not make that figure the current figure (i.e., the one referred by the `gcf` command).

MenuBar `none | {figure}`

Enable-disable figure menu bar. This property enables you to display or hide the menu bar placed at the top of a figure window. The default (`figure`) is to display the menu bar.

This property affects only built in menus. Menus defined with the `uimenu` command are not affected by this property.

MinColormap scalar (default = 64)

Minimum number of color table entries used. This property specifies the minimum number of system color table entries used by MATLAB to store the colormap defined for the figure (see the `ColorMap` property). In certain situations, you may need to increase this value to ensure proper use of colors.

For example, suppose you are running color-intensive applications in addition to MATLAB and have defined a large figure colormap (e.g., 150 to 200 colors). MATLAB may select colors that are close but not exact from the existing colors in the system color table because there are not enough slots available to define all the colors you specified.

To ensure MATLAB uses exactly the colors you define in the figure colormap, set `MinColorMap` equal to the length of the colormap.

```
set(gcf,'MinColormap',length(get(gcf,'ColorMap')))
```

Note that the larger the value of `MinColorMap`, the greater the likelihood other windows (including other MATLAB figure windows) will display in false colors.

Name string

Figure window title. This property specifies the title displayed in the figure window. By default, `Name` is empty and the figure title is displayed as `Figure No. 1`, `Figure No. 2`, and so on. When you set this parameter to a string, the figure title becomes `Figure No. 1: `<*string*>. See the `NumberTitle` property.

NextPlot `{add} | replace | replacechildren`

How to add next plot. `NextPlot` determines which figure MATLAB uses to display graphics output. If the value of the current figure is:

- `add` — use the current figure to display graphics (the default).
- `replace` — reset all figure properties, except `Position`, to their defaults and delete all figure children before displaying graphics (equivalent to `clf reset`).
- `replacechildren` — remove all child objects, but do not reset figure properties (equivalent to `clf`).

The `newplot` function provides an easy way to handle the `NextPlot` property. Also see the `NextPlot` property of axes and the *Using MATLAB Graphics* manual.

NumberTitle `{on} | off`

Figure window title number. This property determines whether the string `Figure No. N` (where `N` is the figure number) is prefixed to the figure window title. See the `Name` property.

PaperOrientation `{portrait} | landscape`

Horizontal or vertical paper orientation. This property determines how printed figures are oriented on the page. `portrait` orients the longest page dimension vertically; `landscape` orients the longest page dimension horizontally. See the `orient` command for more detail.

PaperPosition four-element rect vector

Location on printed page. A rectangle that determines the location of the figure on the printed page. Specify this rectangle with a vector of the form

```
rect = [left, bottom, width, height]
```

where `left` specifies the distance from the left side of the paper to the left side of the rectangle and `bottom` specifies the distance from the bottom of the page to the bottom of the rectangle. Together these distances define the lower-left corner of the rectangle. `width` and `height` define the dimensions of the rectangle. The `PaperUnits` property specifies the units used to define this rectangle.

PaperPositionMode `auto | {manual}`

WYSIWYG printing of figure. In `manual` mode, MATLAB honors the value specified by the `PaperPosition` property. In `auto` mode, MATLAB prints the figure the same size as it appears on the computer screen, centered on the page.

PaperSize `[width height]` (read only)

Paper size. This property contains the size of the current `PaperType`, measured in `PaperUnits`. See `PaperType` to select standard paper sizes.

PaperType Select a value from the following table

Selection of standard paper size. This property sets the `PaperSize` to the one of the following standard sizes.

Property Value	Size (Width x Height)
`usletter (default)`	8.5-by-11 inches
`uslegal`	11-by-14 inches
`tabloid`	11-by-17 inches
`A0`	841-by-1189mm
`A1`	594-by-841mm
`A2`	420-by-594mm
`A3`	297-by-420mm

Property Value	Size (Width x Height)
A4	210-by-297mm
A5	148-by-210mm
B0	1029-by-1456mm
B1	728-by-1028mm
B2	514-by-728mm
B3	364-by-514mm
B4	257-by-364mm
B5	182-by-257mm
arch-A	9-by-12 inches
arch-B	12-by-18 inches
arch-C	18-by-24 inches
arch-D	24-by-36 inches
arch-E	36-by-48 inches
A	8.5-by-11 inches
B	11-by-17 inches
C	17-by-22 inches
D	22-by-34 inches
E	34-by-43 inches

Note that you may need to change the `PaperPosition` property in order to position the printed figure on the new paper size. One solution is to use `normalized` `PaperUnits`, which enables MATLAB to automatically size the figure to occupy the same relative amount of the printed page, regardless of the paper size.

PaperUnits normalized | {inches} | centimeters | points

Hardcopy measurement units. This property specifies the units used to define the PaperPosition and PaperSize properties. All units are measured from the lower-left corner of the page. normalized units map the lower-left corner of the page to (0, 0) and the upper-right corner to (1.0, 1.0). inches, centimeters, and points are absolute units (one point equals 1/72 of an inch).

If you change the value of PaperUnits, it is good practice to return it to its default value after completing your computation so as not to affect other functions that assume PaperUnits is set to the default value.

Parent handle

Handle of figure's parent. The parent of a figure object is the root object. The handle to the root is always 0.

Pointer crosshair | {arrow} | watch | topl | topr | botl | botr | circle | cross | fleur | left | right | top | bottom | fullcrosshair | ibeam | custom

Pointer symbol selection. This property determines the symbol used to indicate the pointer (cursor) position in the figure window. Setting Pointer to custom allows you to define your own pointer symbol. See the PointerShapeCData property for more information. See also the *Using MATLAB Graphics* manual.

PointerShapeCData 16-by-16 matrix

User-defined pointer. This property defines the pointer that is used when you set the Pointer property to custom. It is a 16-by-16 element matrix defining the 16-by-16 pixel pointer using the following values:

- 1 – color pixel black
- 2 – color pixel white
- NaN – make pixel transparent (underlying screen shows through)

Element (1,1) of the PointerShapeCData matrix corresponds to the upper-left corner of the pointer. Setting the Pointer property to one of the predefined pointer symbols does not change the value of the PointerShapeCData. Computer systems supporting 32-by-32 pixel pointers fill only one quarter of the available pixmap.

PointerShapeHotSpot 2-element vector

Pointer active area. A two-element vector specifying the row and column indices in the PointerShapeCData matrix defining the pixel indicating the pointer location. The location is contained in the CurrentPoint property and the root object's PointerLocation property. The default value is element (1,1), which is the upper-left corner.

Position four-element vector

Figure position. This property specifies the size and location on the screen of the figure window. Specify the position rectangle with a four-element vector of the form:

```
rect = [left, bottom, width, height]
```

where left and bottom define the distance from the lower-left corner of the screen to the lower-left corner of the figure window. width and height define the dimensions of the window. See the Units property for information on the units used in this specification. The left and bottom elements can be negative on systems that have more than one monitor.

You can use the get function to obtain this property and determine the position of the figure and you can use the set function to resize and move the figure to a new location.

Renderer painters | zbuffer

Rendering method used for screen and printing. This property enables you to select the method used to render MATLAB graphics. The choices are:

- painters – MATLAB's original rendering method is faster when the figure contains only simple or small graphics objects.
- zbuffer – MATLAB draws graphics object faster and more accurately because objects are colored on a per pixel basis and MATLAB renders only those pixels that are visible in the scene (thus eliminating front-to-back sorting errors). Note that this method can consume a lot of system memory if MATLAB is displaying a complex scene.
- OpenGL – OpenGL is a renderer that is available on many computer systems. This renderer is generally faster than painters or zbuffer and in some cases enables MATLAB to access graphics hardware that is available on some systems.

Using the OpenGL Renderer

The figure `Renderer` property supports a new value that enables MATLAB to use OpenGL as the renderer. The command to enable OpenGL on the current figure is:

```
set(gcf,'Renderer','OpenGL')
```

OpenGL increases performance for most 2-D and 3-D graphs drawn with MATLAB.

Hardware vs. Software OpenGL Implementations

There are two kinds of OpenGL implementations – hardware and software.

The hardware implementation makes use of special graphics hardware to increase performance and is therefore significantly faster than the software version. Many computers have this special hardware available as an option or may come with this hardware right out of the box.

Software implementations of OpenGL are much like the ZBuffer renderer that is available on MATLAB version 5.0, however, OpenGL generally provides superior performance to ZBuffer.

OpenGL Availability

OpenGL is available on all computers that MATLAB runs on with the exception of VMS. MATLAB automatically finds hardware versions of OpenGl if they are available. If the hardware version is not available, then MATLAB uses the software version.

The software versions that are available on different platforms are:

- On UNIX systems, MATLAB uses the software version of OpenGL that is included in the MATLAB distribution.
- On MS-Windows NT 4.0, OpenGL is available as part of the operating system.
- On MS-Windows 95, OpenGL is included in the Windows 95 OSR 2 release. If you do not have this release, the libraries are available on the Microsoft ftp site.

  Microsoft version is available at the URL:

  `ftp://ftp.microsoft.com/softlib/mslfiles/opengl95.exe`

There is also a Silicon Graphics version of OpenGL for Windows 95 that is available at the URL:

```
http://www.sgi.com
```

Tested Hardware Versions

On MS-Windows platforms, there are many graphics boards that accelerate OpenGL. The MathWorks has tested MATLAB on the AccelECLIPSE board from AccelGraphics.

On UNIX platforms, The MathWorks has tested MATLAB on Sparc Ultra with the Creator 3D board and Silicon Graphics computers running IRIX 6.4 or newer.

Determining What Version You Are Using

To determine the version and vendor of the OpenGL library that MATLAB is using on your system, type the following command at the MATLAB prompt.

```
feature GetOpenGLInfo
```

This command also returns a string of extensions to the OpenGL specification that are available with the particular library MATLAB is using. This information is helpful to The MathWorks, so please include this information if you need to report bugs.

OpenGL vs. Other MATLAB Renderers

There are some difference between drawings created with OpenGL and those created with the other renderers. The OpenGL specific differences include:

- OpenGL does not do colormap interpolation. If you create a surface or patch using indexed color and interpolated face or edge coloring, OpenGL will interpolate the colors through the RGB color cube instead of through the colormap.
- OpenGL does not support the `phong` value for the `FaceLighting` and `EdgeLighting` properties of surfaces and patches.

MATLAB issues a warning if you request nonsupported behavior.

Printing from OpenGL

When you print a figure that was drawn with OpenGL, MATLAB switches to the ZBuffer renderer to produce output, which has a resolution determined by the `print` command's `–r` option. This may cause flashing of the figure as the renderer changes.

Implementations of OpenGL Tested by The MathWorks

The following hardware versions have been tested:

- AccelECLIPSE by AccelGraphics
- Sol2/Creator 3D
- SGI

The following software versions have been tested:

- Mesa
- CosmoGL
- Microsoft's Windows 95 implementation
- NT 4.0

RendererMode `{auto} | manual`

Automatic, or user selection of Renderer. This property enables you to specify whether MATLAB should choose the `Renderer` based on the contents of the figure window, or whether the `Renderer` should remain unchanged.

When the `RendererMode` property is set to `auto`, MATLAB selects the rendering method for printing as well as for screen display based on the size and complexity of the graphics objects in the figure.

For printing, MATLAB switches to `zbuffer` at a greater scene complexity than for screen rendering because printing from a Z-buffered figure can be considerably slower than one using the `painters` rendering method, and can result in large PostScript files. However, the output does always match what is on the screen. The same holds true for OpenGL: the output is the same as that produced by the ZBuffer renderer – a bitmap with a resolution determined by the `print` command's `–r` option

When the `RendererMode` property is set to `manual`, MATLAB does not change the `Renderer`, regardless of changes to the figure contents.

Resize {on} | off

Window resize mode. This property determines if you can resize the figure window with the mouse. on means you can resize the window, off means you cannot. When Resize is off, the figure window does not display any resizing controls (such as boxes at the corners) to indicate that it cannot be resized.

ResizeFcn string

Window resize callback routine. MATLAB executes the specified callback routine whenever you resize the figure window. You can query the figure's Position property to determine the new size and position of the figure window. During execution of the callback routine, the handle to the figure being resized is accessible only through the root CallbackObject property, which you can query using gcbo.

You can use ResizeFcn to maintain a GUI layout that is not directly supported by MATLAB's Position/Units paradigm.

For example, consider a GUI layout that maintains an object at a constant height in pixels and attached to the top of the figure, but always matches the width of the figure. The following ResizeFcn accomplishes this; it keeps the uicontrol whose Tag is 'StatusBar' 20 pixels high, as wide as the figure, and attached to the top of the figure. Note the use of the Tag property to retrieve the uicontrol handle, and the gcbo function to retrieve the figure handle. Also note the defensive programming regarding figure Units, which the callback requires to be in pixels in order to work correctly, but which the callback also restores to their previous value afterwards.

```
u = findobj('Tag','StatusBar');
fig = gcbo;
old_units = get(fig,'Units');
set(fig,'Units','pixels');
figpos = get(fig,'Position');
upos = [0, figpos(4) - 20, figpos(3), 20];
set(u,'Position',upos);
set(fig,'Units',old_units);
```

You can change the figure Position from within the ResizeFcn callback; however the ResizeFcn is not called again as a result.

Note that the print command can cause the ResizeFcn to be called if the PaperPositionMode property is set to manual and you have defined a resize function. If you do not want your resize function called by `print`, set the `PaperPositionMode` to `auto`.

`Selected` `on | off`

Is object selected. This property indicates whether the figure is selected. You can, for example, define the `ButtonDownFcn` to set this property, allowing users to select the object with the mouse.

`SelectionHighlight` `{on} | off`

figures do not indicate selection.

`SelectionType` `{normal} | extend | alt | open`
`(this property is read only)`

Mouse selection type. MATLAB maintains this property to provide information about the last mouse button press that occurred within the figure window. This information indicates the type of selection made. Selection types are actions that are generally associated with particular responses from the user interface software (e.g., single clicking on a graphics object places it in move or resize mode; double-clicking on a filename opens it, etc.).

The physical action required to make these selections varies on different platforms. However, all selection types exist on all platforms.

Selection Type	MS-Windows	X-Windows
`Normal`	Click left mouse button	Click left mouse button
`Extend`	**Shift** - click left mouse button or click both left and right mouse buttons	**Shift** - click left mouse button or click middle mouse button
`Alternate`	**Control** - click left mouse button or click right mouse button	**Control** - click left mouse button or click right mouse button
`Open`	Double click any mouse button	Double click any mouse button

Note that the `ListBox` style of uicontrols set the figure `SelectionType` property to `normal` to indicate a single mouse click or to `open` to indicate a double mouse click. See `uicontrol` for information on how this property is set when you click on a uicontrol object.

ShareColors `{on} | off`

Share slots in system colortable with like colors. This property affects the way MATLAB stores the figure colormap in the system color table. By default, MATLAB looks at colors already defined and uses those slots to assign pixel colors. This leads to an efficient use of color resources (which are limited on systems capable of displaying 256 or less colors) and extends the number of figure windows that can simultaneously display correct colors.

However, in situations where you want to change the figure colormap quickly without causing MATLAB to re-render the displayed graphics objects, you should disable color sharing (set `ShareColors` to `off`). In this case, MATLAB can swap one colormap for another without changing pixel color assignments because all the slots in the system color table used for the first colormap are replaced with the corresponding color in the second colormap. (Note that this applies only in cases where both colormaps are the same length and where the computer hardware allows user modification of the system color table.)

Tag string

User-specified object label. The `Tag` property provides a means to identify graphics objects with a user-specified label. This is particularly useful when constructing interactive graphics programs that would otherwise need to define object handles as global variables or pass them as arguments between callback routines.

For example, suppose you want to direct all graphics output from an M-file to a particular figure, regardless of user actions that may have changed the current figure. To do this, identify the figure with a `Tag`.

```
figure('Tag','Plotting Figure')
```

Then make that figure the current figure before drawing by searching for the `Tag` with `findobj`.

```
figure(findobj('Tag','Plotting Figure'))
```

Type `string (read only)`

Object class. This property identifies the kind of graphics object. For figure objects, `Type` is always the string `'figure'`.

UIContextMenu handle of a uicontextmenu object

Associate a context menu with the figure. Assign this property the handle of a uicontextmenu object created in the figure. Use the `uicontextmenu` function to create the context menu. MATLAB displays the context menu whenever you right-click over the figure.

Units `{pixels} | normalized | inches | centimeters | points | characters`

Units of measurement. This property specifies the units MATLAB uses to interpret size and location data. All units are measured from the lower-left corner of the window.

- `normalized` units map the lower-left corner of the figure window to (0,0) and the upper-right corner to (1.0,1.0).
- `inches`, `centimeters`, and `points` are absolute units (one point equals 1/72 of an inch).
- The size of a `pixel` depends on screen resolution.
- `Characters` units are defined by characters from the default system font; the width of one character is the width of the letter x, the height of one character is the distance between the baselines of two lines of text.

This property affects the `CurrentPoint` and `Position` properties. If you change the value of `Units`, it is good practice to return it to its default value after completing your computation so as not to affect other functions that assume `Units` is set to the default value.

When specifying the units as property/value pairs during object creation, you must set the `Units` property before specifying the properties that you want to use these units.

UserData matrix

User specified data. You can specify `UserData` as any matrix you want to associate with the figure object. The object does not use this data, but you can access it using the `set` and `get` commands.

Visible `{on} | off`

Object visibility. The `Visible` property determines whether an object is displayed on the screen. If the `Visible` property of a figure is `off`, the entire figure window is invisible.

WindowButtonDownFcn string

Button press callback function. Use this property to define a callback routine that MATLAB executes whenever you press a mouse button while the pointer is in the figure window. Define this routine as a string that is a valid MATLAB expression or the name of an M-file. The expression executes in the MATLAB workspace.

See `uicontrol` for information on how this property is set when you click on a uicontrol object.

WindowButtonMotionFcn string

Mouse motion callback function. Use this property to define a callback routine that MATLAB executes whenever you move the pointer within the figure window. Define this routine as a string that is a valid MATLAB expression or the name of an M-file. The expression executes in the MATLAB workspace.

WindowButtonUpFcn string

Button release callback function. Use this property to define a callback routine that MATLAB executes whenever you release a mouse button. Define this routine as a string that is a valid MATLAB expression or the name of an M-file. The expression executes in the MATLAB workspace.

The button up event is associated with the figure window in which the preceding button down event occurred. Therefore, the pointer need not be in the figure window when you release the button to generate the button up event.

If the callback routines defined by `WindowButtonDownFcn` or `WindowButtonMotionFcn` contain `drawnow` commands or call other functions that contain `drawnow` commands and the `Interruptible` property is set to `off`, the `WindowButtonUpFcn` may not be called. You can prevent this problem by setting `Interruptible` to `on`.

WindowStyle `{normal} | modal`

Normal or modal window behavior. When `WindowStyle` is set to `modal`, the figure window traps all keyboard and mouse events over all MATLAB windows as long as they are visible. Windows belonging to applications other than MATLAB are unaffected. Modal figures remain stacked above all normal figures and the MATLAB command window. When multiple modal windows exist, the most recently created window keeps focus and stays above all other windows until it becomes invisible, or is returned to WindowStyle normal, or is deleted. At that time, focus reverts to the window that last had focus.

Figures with WindowStyle modal and Visible off do not behave modally until they are made visible, so it is acceptable to hide a modal window instead of destroying it when you want to reuse it.

You can change the WindowStyle of a figure at any time, including when the figure is visible and contains children. However, on some systems this may cause the figure to flash or disappear and reappear, depending on the windowing-system's implementation of normal and modal windows. For best visual results, you should set WindowStyle at creation time or when the figure is invisible.

Modal figures do not display uimenu children or built-in menus, but it is not an error to create uimenus in a modal figure or to change WindowStyle to modal on a figure with uimenu children. The uimenu objects exist and their handles are retained by the figure. If you reset the figure's WindowStyle to normal, the uimenus are displayed.

Use modal figures to create dialog boxes that force the user to respond without being able to interact with other windows. Typing **Control C** at the MATLAB prompt causes all figures with WindowStyle modal to revert to WindowStyle normal, allowing you to type at the command line.

XDisplay display identifier (UNIX only)

Specify display for MATLAB. You can display figure windows on different displays using the XDisplay property. For example, to display the current figure on a system called fred, use the command:

```
set(gcf,'XDisplay','fred:0.0')
```

XVisual visual identifier (UNIX only)

Select visual used by MATLAB. You can select the visual used by MATLAB by setting the XVisual property to the desired visual ID. This can be useful if you want to test your application on an 8-bit or grayscale visual. To see what visuals are avail on your system, use the UNIX xdpyinfo command. From MATLAB, type

```
!xdpyinfo
```

The information returned will contain a line specifying the visual ID. For example,

```
visual id:    0x21
```

To use this visual with the current figure, set the `XVisual` property to the ID.

```
set(gcf,'XVisual','0x21')
```

XVisualMode `auto | manual`

Auto or manual selection of visual. `VisualMode` can take on two values – `auto` (the default) and `manual`. In `auto` mode, MATLAB selects the best visual to use based on the number of colors, availability of the OpenGL extension, etc. In `manual` mode, MATLAB does not change the visual from the one currently in use. Setting the `XVisual` property sets this property to `manual`.

fill

Purpose Filled two-dimensional polygons

Syntax

```
fill(X,Y,C)
fill(X,Y,ColorSpec)
fill(X1,Y1,C1,X2,Y2,C2,...)
fill(...,'PropertyName',PropertyValue)
h = fill(...)
```

Description The `fill` function creates colored polygons.

`fill(X,Y,C)` creates filled polygons from the data in `X` and `Y` with vertex color specified by `C`. `C` is a vector or matrix used as an index into the colormap. If `C` is a row vector, `length(C)` must equal `size(X,2)` and `size(Y,2)`; if `C` is a column vector, `length(C)` must equal `size(X,1)` and `size(Y,1)`. If necessary, `fill` closes the polygon by connecting the last vertex to the first.

`fill(X,Y,ColorSpec)` fills two-dimensional polygons specified by `X` and `Y` with the color specified by `ColorSpec`.

`fill(X1,Y1,C1,X2,Y2,C2,...)` specifies multiple two-dimensional filled areas.

`fill(...,'`*`PropertyName`*`',PropertyValue)` allows you to specify property names and values for a patch graphics object.

`h = fill(...)` returns a vector of handles to patch graphics objects, one handle per patch object.

Remarks If `X` or `Y` is a matrix, and the other is a column vector with the same number of elements as rows in the matrix, `fill` replicates the column vector argument to produce a matrix of the required size. `fill` forms a vertex from corresponding elements in `X` and `Y` and creates one polygon from the data in each column.

The type of color shading depends on how you specify color in the argument list. If you specify color using `ColorSpec`, `fill` generates flat-shaded polygons by setting the patch object's `FaceColor` property to the corresponding RGB triple.

If you specify color using `C`, `fill` scales the elements of `C` by the values specified by the axes property `CLim`. After scaling `C`, `C` indexes the current colormap.

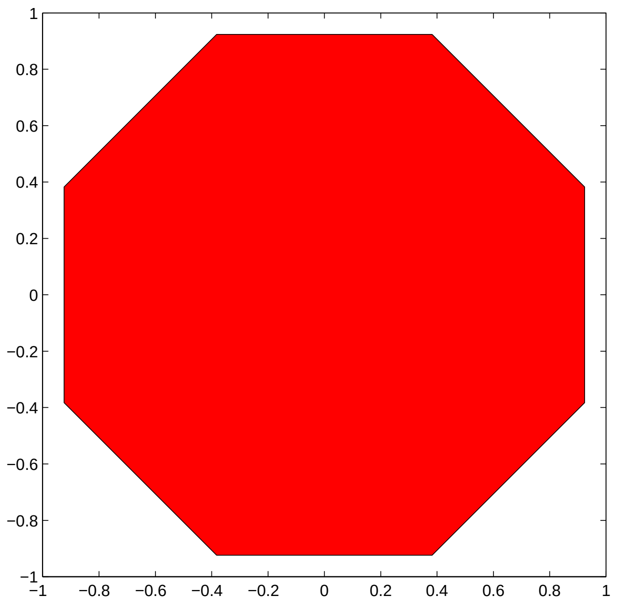
If C is a row vector, fill generates flat-shaded polygons where each element determines the color of the polygon defined by the respective column of the X and Y matrices. Each patch object's FaceColor property is set to 'flat'. Each row element becomes the CData property value for the nth patch object, where *n* is the corresponding column in X or Y.

If C is a column vector or a matrix, fill uses a linear interpolation of the vertex colors to generate polygons with interpolated colors. It sets the patch graphics object FaceColor property to 'interp' and the elements in one column become the CData property value for the respective patch object. If C is a column vector, fill replicates the column vector to produce the required sized matrix.

Examples

Create a red octagon.

```
t = (1/16:1/8:1)'*2*pi;
x = sin(t);
y = cos(t);
fill(x,y,'r')
axis square
```



See Also

axis, caxis, colormap, ColorSpec, fill3, patch

fill3

Purpose Filled three-dimensional polygons

Syntax

```
fill3(X,Y,Z,C)
fill3(X,Y,Z,ColorSpec)
fill3(X1,Y1,Z1,C1,X2,Y2,Z2,C2,...)
fill3(...,'PropertyName',PropertyValue)
h = fill3(...)
```

Description The `fill3` function creates flat-shaded and Gouraud-shaded polygons.

`fill3(X,Y,Z,C)` fills three-dimensional polygons. `X`, `Y`, and `Z` triplets specify the polygon vertices. If `X`, `Y`, or `Z` is a matrix, `fill3` creates *n* polygons, where *n* is the number of columns in the matrix. `fill3` closes the polygons by connecting the last vertex to the first when necessary.

`C` specifies color, where `C` is a vector or matrix of indices into the current colormap. If `C` is a row vector, `length(C)` must equal `size(X,2)` and `size(Y,2)`; if `C` is a column vector, `length(C)` must equal `size(X,1)` and `size(Y,1)`.

`fill3(X,Y,Z,ColorSpec)` fills three-dimensional polygons defined by `X`, `Y`, and `Z` with color specified by `ColorSpec`.

`fill3(X1,Y1,Z1,C1,X2,Y2,Z2,C2,...)` specifies multiple filled three-dimensional areas.

`fill3(...,'`*`PropertyName`*`',PropertyValue)` allows you to set values for specific patch properties.

`h = fill3(...)` returns a vector of handles to patch graphics objects, one handle per patch.

Algorithm If `X`, `Y`, and `Z` are matrices of the same size, `fill3` forms a vertex from the corresponding elements of `X`, `Y`, and `Z` (all from the same matrix location), and creates one polygon from the data in each column.

If `X`, `Y`, or `Z` is a matrix, `fill3` replicates any column vector argument to produce matrices of the required size.

If you specify color using `ColorSpec`, `fill3` generates flat-shaded polygons and sets the patch object `FaceColor` property to an RGB triple.

If you specify color using C, fill3 scales the elements of C by the axes property CLim, which specifies the color axis scaling parameters, before indexing the current colormap.

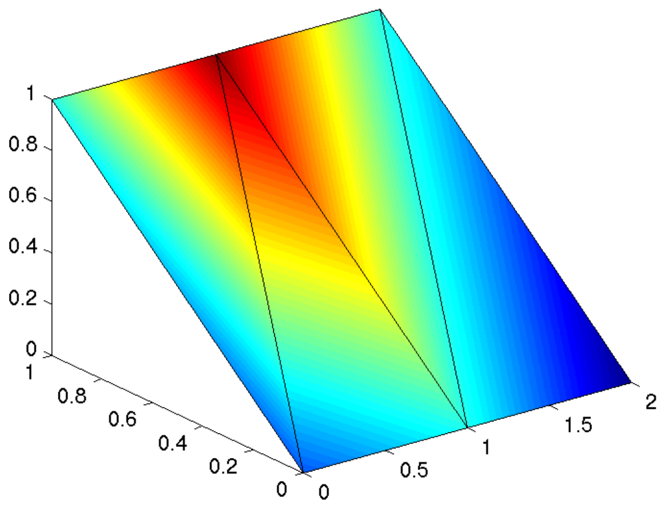
If C is a row vector, fill3 generates flat-shaded polygons and sets the FaceColor property of the patch objects to 'flat'. Each element becomes the CData property value for the respective patch object.

If C is a column vector or a matrix, fill3 generates polygons with interpolated colors and sets the patch object FaceColor property to 'interp'. fill3 uses a linear interpolation of the vertex colormap indices when generating polygons with interpolated colors. The elements in one column become the CData property value for the respective patch object. If C is a column vector, fill3 replicates the column vector to produce the required sized matrix.

Examples

Create four triangles with interpolated colors.

```
X = [0 1 1 2;1 1 2 2;0 0 1 1];
Y = [1 1 1 1;1 0 1 0;0 0 0 0];
Z = [1 1 1 1;1 0 1 0;0 0 0 0];
C = [0.5000 1.0000 1.0000 0.5000;
     1.0000 0.5000 0.5000 0.1667;
     0.3330 0.3330 0.5000 0.5000];
fill3(X,Y,Z,C)
```



See Also `axis`, `caxis`, `colormap`, `ColorSpec`, `fill`, `patch`

Purpose Find visible off-screen figures

Syntax

```
findfigs
```

Description `findfigs` finds all visible figure windows whose display area is off the screen and positions them on the screen.

A window appears to MATLAB to be off-screen when its display area (the area not covered by the window's title bar, menu bar, and toolbar) does not appear on the screen.

This function is useful when bringing an application from a larger monitor to a smaller one (or one with lower resolution). Windows visible on the larger monitor may appear off-screen on a smaller monitor. Using `findfigs` ensures that all windows appear on the screen.

findobj

Purpose Locate graphics objects

Syntax

```
h = findobj
h = findobj('PropertyName',PropertyValue,...)
h = findobj(objhandles,...)
h = findobj(objhandles,'flat','PropertyName',PropertyValue,...)
```

Description

`findobj` locates graphics objects and returns their handles. You can limit the search to objects with particular property values and along specific branches of the hierarchy.

`h = findobj` returns the handles of the root object and all its descendants.

`h = findobj('`*PropertyName*`',PropertyValue,...)` returns the handles of all graphics objects having the property *PropertyName*, set to the value `PropertyValue`. You can specify more than one property/value pair, in which case, `findobj` returns only those objects having all specified values.

`h = findobj(objhandles,...)` restricts the search to objects listed in `objhandles` and their descendants.

`h = findobj(objhandles,'flat','`*PropertyName*`',PropertyValue,...)` restricts the search to those objects listed in `objhandles` and does not search descendants.

Remarks

`findobj` returns an error if a handle refers to a non-existent graphics object.

Findobj correctly matches any legal property value. For example,

```
findobj('Color','r')
```

finds all objects having a `Color` property set to `red`, `r`, or `[1 0 0]`.

When a graphics object is a descendant of more than one object identified in `objhandles`, MATLAB searches the object each time `findobj` encounters its handle. Therefore, implicit references to a graphics object can result in its handle being returned multiple times.

Examples

Find all line objects in the current axes:

```
h = findobj(gca,'Type','line')
```

See Also

`copyobj`, `gcf`, `gca`, `gcbo`, `gco`, `get`, `set`

Graphics objects include:

`axes`, `figure`, `image`, `light`, `line`, `patch`, `surface`, `text`, `uicontrol`, `uimenu`

fplot

Purpose Plot a function between specified limits

Syntax

```
fplot('function',limits)
fplot('function',limits,LineSpec)
fplot('function',limits,tol)
fplot('function',limits,tol,LineSpec)
[x,Y] = fplot(...)
```

Description `fplot` plots a function between specified limits. The function must be of the form $y = f(x)$, where x is a vector whose range specifies the limits, and `y` is a vector the same size as `x` and contains the function's value at the points in x (see the first example). If the function returns more than one value for a given x, then y is a matrix whose columns contain each component of $f(x)$ (see the second example).

`fplot('`*`function`*`',limits)` plots `'`*`function`*`'` between the limits specified by `limits`. `limits` is a vector specifying the x-axis limits (`[xmin xmax]`), or the x- and y-axis limits, (`[xmin xmax ymin ymax]`).

`fplot('`*`function`*`',limits,LineSpec)` plots `'`*`function`*`'` using the line specification `LineSpec`. `'`*`function`*`'` is a name of a MATLAB M-file or a string containing the variable `x`.

`fplot('`*`function`*`',limits,tol)` plots `'`*`function`*`'` using the relative error tolerance `tol` (default is 2e–3). The maximum number of `x` steps is `(1/tol)+1`.

`fplot('`*`function`*`',limits,tol,LineSpec)` plots `'`*`function`*`'` using the relative error tolerance `tol` and a line specification that determines line type, marker symbol, and color.

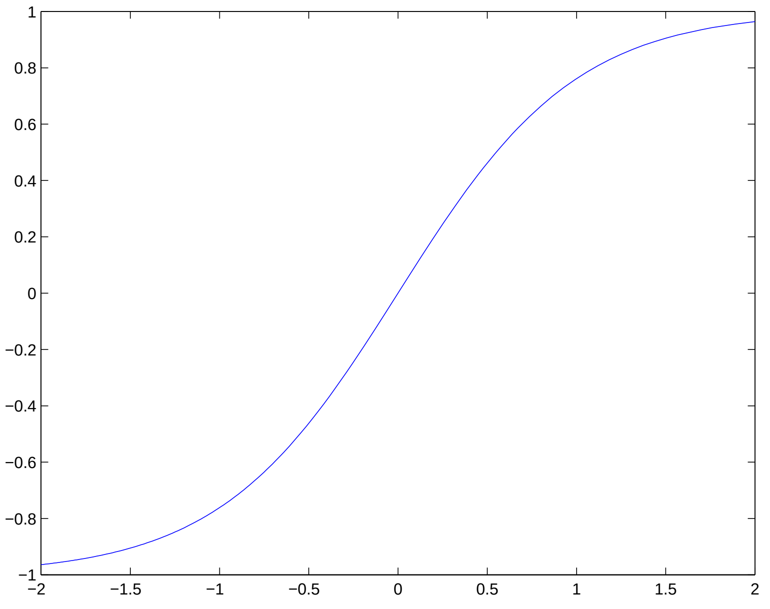
`[x,Y] = fplot(...)` returns the abscissas and ordinates for `'`*`function`*`'` in `x` and `Y`. No plot is drawn on the screen. You plot the function using `plot(x,Y)`.

Remarks `fplot` uses adaptive step control to produce a representative graph, concentrating its evaluation in regions where the function's rate of change is the greatest.

Examples

Plot the hyperbolic tangent function from -2 to 2:

```
fplot('tanh',[-2 2])
```

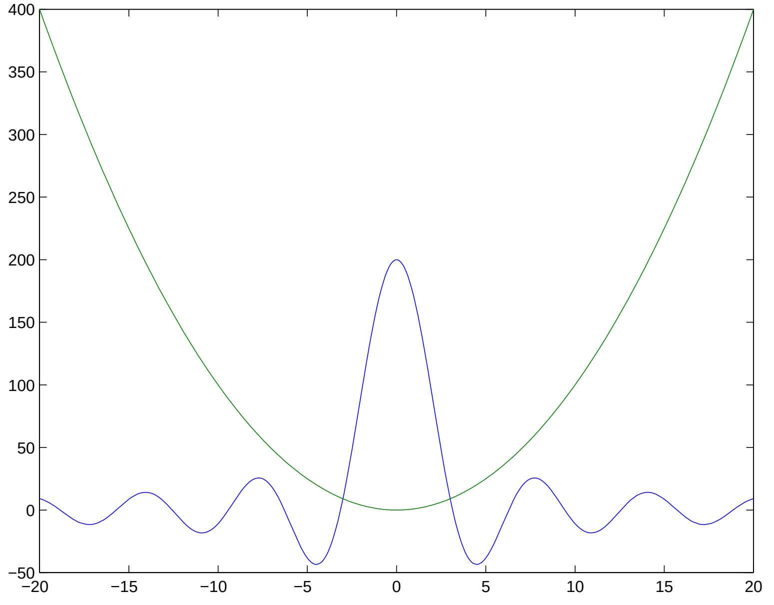


Create an M-file, `myfun`, that returns a two column matrix:

```
function Y = myfun(x)
Y(:,1) = 200*sin(x(:))./x(:);
Y(:,2) = x(:).^2;
```

Plot the function with the statement:

```
fplot('myfun',[-20 20]
```



See Also `eval`, `feval`, `LineSpec`, `plot`

Purpose Convert movie frame to indexed image

Syntax

```
[X,Map] = frame2im(F)
```

Description

`[X,Map] = frame2im(F)` converts the single movie frame `F` into the indexed image `X` and associated colormap `Map`. The functions `getframe` and `im2frame` create a movie frame. If the frame contains truecolor data, then `Map` is empty.

See Also

`getframe`, `im2frame`, `movie`

gca

Purpose Get current axes handle

Syntax

```
h = gca
```

Description

`h = gca` returns the handle to the current axes for the current figure. If no axes exists, MATLAB creates one and returns its handle. You can use the statement

```
get(gcf,'CurrentAxes')
```

if you do not want MATLAB to create an axes if one does not already exist.

The current axes is the target for graphics output when you create axes children. Graphics commands such as `plot`, `text`, and `surf` draw their results in the current axes. Changing the current figure also changes the current axes.

See Also

`axes`, `cla`, `delete`, `gcf`, `gcbo`, `gco`, `hold`, `subplot`, `findobj`

figure `CurrentAxes` property

Purpose

Return the handle of the object whose callback is currently executing

Syntax

```
h = gcbo
[h, figure] = gcbo
```

Description

`h = gcbo` returns the handle of the graphics object whose callback is executing.

`[h, figure] = gcbo` returns the handle of the current callback object and the handle of the figure containing this object.

Remarks

MATLAB stores the handle of the object whose callback is executing in the root's `CallbackObject` property. If a callback interrupts another callback, MATLAB replaces the `CallbackObject` value with the handle of the object whose callback is interrupting. When that callback completes, MATLAB restores the handle of the object whose callback was interrupted.

The root `CallbackObject` property is read-only, so its value is always valid at any time during callback execution. The root `CurrentFigure` property, and the figure `CurrentAxes` and `CurrentObject` properties (returned by `gcf`, `gca`, and `gco` respectively) are user settable, so they can change during the execution of a callback, especially if that callback is interrupted by another callback. Therefore, those functions are not reliable indicators of which object's callback is executing.

When you write callback routines for the `CreateFcn` and `DeleteFcn` of any object and the figure `ResizeFcn`, you must use `gcbo` since those callbacks do not update the root's `CurrentFigure` property, or the figure's `CurrentObject` or `CurrentAxes` properties; they only update the root's `CallbackObject` property.

When no callbacks are executing, `gcbo` returns `[]` (an empty matrix).

See Also

`gca`, `gcf`, `gco`, `rootobject`

gcf

Purpose Get current figure handle

Syntax

```
h = gcf
```

Description

`h = gcf` returns the handle of the current figure. The current figure is the figure window in which graphics commands such as `plot`, `title`, and `surf` draw their results. If no figure exists, MATLAB creates one and returns its handle. You can use the statement

```
get(0,'CurrentFigure')
```

if you do not want MATLAB to create a figure if one does not already exist.

See Also

`axes`, `clf`, `close`, `delete`, `figure`, `gca`, `gcbo`, `gco`, `subplot`

root `CurrentFigure` property

Purpose

Return handle of current object

Syntax

```
h = gco
h = gco(figure_handle)
```

Description

`h = gco` returns the handle of the current object.

`h = gco(figure_handle)` returns the value of the current object for the figure specified by `figure_handle`.

Remarks

The current object is the last object clicked on, excluding uimenus. If the mouse click did not occur over a figure child object, the figure becomes the current object. MATLAB stores the handle of the current object in the figure's `CurrentObject` property.

The `CurrentObject` of the `CurrentFigure` does not always indicate the object whose callback is being executed. Interruptions of callbacks by other callbacks can change the `CurrentObject` or even the `CurrentFigure`. Some callbacks, such as `CreateFcn` and `DeleteFcn`, and uimenu `Callback` intentionally do not update `CurrentFigure` or `CurrentObject`.

`gcbo` provides the only completely reliable way to retrieve the handle to the object whose callback is executing, at any point in the callback function, regardless of the type of callback or of any previous interruptions.

Examples

This statement returns the handle to the current object in figure window 2:

```
h = gco(2)
```

See Also

`gca`, `gcbo`, `gcf`

The `root` object description

get

Purpose Get object properties

Syntax

```
get(h)
get(h,'PropertyName')
<m-by-n value cell array> = get(H,<property cell array>)
a = get(h)
a = get(0,'Factory')
a = get(0,'FactoryObjectTypePropertyName')
a = get(h,'Default')
a = get(h,'DefaultObjectTypePropertyName')
```

Description

`get(h)` returns all properties and their current values of the graphics object identified by the handle `h`.

`get(h,'PropertyName')` returns the value of the property '*PropertyName*' of the graphics object identified by `h`.

`<m-by-n value cell array> = get(H,pn)` returns *n* property values for *m* graphics objects in the *m*-by-*n* cell array, where `m = length(H)` and *n* is equal to the number of property names contained in `pn`.

`a = get(h)` returns a structure whose field names are the object's property names and whose values are the current values of the corresponding properties. `h` must be a scalar. If you do not specify an output argument, MATLAB displays the information on the screen.

`a = get(0,'Factory')` returns the factory-defined values of all user-settable properties. `a` is a structure array whose field names are the object property names and whose field values are the values of the corresponding properties. If you do not specify an output argument, MATLAB displays the information on the screen.

`a = get(0,'FactoryObjectTypePropertyName')` returns the factory-defined value of the named property for the specified object type. The argument, *FactoryObjectTypePropertyName*, is the word `Factory` concatenated with the object type (e.g., `Figure`) and the property name (e.g., `Color`).

```
FactoryFigureColor
```

`a = get(h,'Default')` returns all default values currently defined on object `h`. `a` is a structure array whose field names are the object property names and whose field values are the values of the corresponding properties. If you do not specify an output argument, MATLAB displays the information on the screen.

`a = get(h,'Default`*ObjectTypePropertyName*`')` returns the factory-defined value of the named property for the specified object type. The argument, *DefaultObjectTypePropertyName*, is the word `Default` concatenated with the object type (e.g., `Figure`) and the property name (e.g., `Color`).

```
DefaultFigureColor
```

Examples

You can obtain the default value of the `LineWidth` property for line graphics objects defined on the root level with the statement:

```
get(0,'DefaultLineLineWidth')

ans =
    0.5000
```

To query a set of properties on all axes children define a cell array of property names:

```
props = {'HandleVisibility', 'Interruptible';
        'SelectionHighlight', 'Type'};
output = get(get(gca,'Children'),props);
```

The variable `output` is a cell array of dimension `length(get(gca,'Children')`–by–4.

For example, type

```
patch;surface;text;line
output = get(get(gca,'Children'),props)
output =

    'on'    'on'    'on'    'line'
    'on'    'off'   'on'    'text'
    'on'    'on'    'on'    'surface'
    'on'    'on'    'on'    'patch'
```

See Also

`findobj`, `gca`, `gcf`, `gco`, `set`

getframe

Purpose Get movie frame

Syntax

```
F = getframe
F = getframe(h)
F = getframe(h,rect)
[X,Map] = getframe(...)
```

Description

`getframe` returns a movie frame. The frame is a snapshot (pixmap) of the current axes or figure.

`F = getframe` gets a frame from the current axes.

`F = getframe(h)` gets a frame from the figure or axes identified by the handle `h`.

`F = getframe(h,rect)` specifies a rectangular area from which to copy the pixmap. `rect` is relative to the lower-left corner of the figure or axes `h`, in pixel units. `rect` is a four-element vector in the form `[left bottom width height]`, where `width` and `height` define the dimensions of the rectangle.

`F = getframe(...)` returns a movie frame, which is a structure having two fields:

- `cdata` – The image data stored as a matrix of uint8 values. The dimensions of `F.cdata` are height-by-width-by-3.
- `colormap` – The colormap stored as an n-by-3 matrix of doubles. `F.colormap` is empty on true color systems.

To capture an image, use this approach:

```
F = getframe(gcf);
image(F.cdata)
colromap(F.colormap)
```

`[X,Map] = getframe(...)` returns the frame as an indexed image matrix `X` and a colormap `Map`. This version is obsolete and is supported only for compatible with earlier version of MATLAB. Since indexed images cannot always capture true color displays, you should use the single output argument form of `getframe`. To write code that is compatible with earlier version of

MATLAB and that can take advantage of true color support, use the following approach:

```
F = getframe(gcf);
[X,Map] = frame2im(f);
imshow(X,Map)
```

Remarks

Usually, `getframe` is used in a `for` loop to assemble an array of movie frames for playback using `movie`. For example,

```
for j = 1:n
   plotting commands
   F(j) = getframe;
end
movie(F)
```

To create movies that are compatible with earlier versions of MATLAB (before Release 11/MATLAB 5.3) use this approach:

```
M = moviein(n);
for j = 1:n
   plotting commands
   M(:,j) = getframe;
end
movie(M)
```

Capture Regions

Note that `F = getframe;` returns the contents of the current axes, exclusive of the axis labels, title, or tick labels. `F = getframe(gcf);` captures the entire interior of the current figure window. To capture the figure window menu, use the form `F = getframe(h,rect)` with a rectangle sized to include the menu.

Examples

Make the peaks function vibrate.

```
Z = peaks; surf(Z)
axis tight
set(gca,'nextplot','replacechildren');
for j = 1:20
    surf(sin(2*pi*j/20)*Z,Z)
    F(j) = getframe;
end
movie(F,20) % Play the movie twenty times
```

See Also

`getframe`, `frame2im`, `image`, `im2frame`, `movie`, `moviein`

Purpose

Input data using the mouse

Syntax

```
[x,y] = ginput(n)
[x,y] = ginput
[x,y,button] = ginput(...)
```

Description

`ginput` enables you to select points from the figure using the mouse or arrow keys for cursor positioning. The figure must have focus before `ginput` receives input.

`[x,y] = ginput(n)` enables you to select `n` points from the current axes and returns the *x*- and *y*-coordinates in the column vectors `x` and `y`, respectively. You can press the **Return** key to terminate the input before entering `n` points.

`[x,y] = ginput` gathers an unlimited number of points until you press the **Return** key.

`[x,y,button] = ginput(...)` returns the *x*-coordinates, the *y*-coordinates, and the button or key designation. `button` is a vector of integers indicating which mouse buttons you pressed (1 for left, 2 for middle, 3 for right), or ASCII numbers indicating which keys on the keyboard you pressed.

Remarks

If you select points from multiple axes, the results you get are relative to those axes coordinates systems.

Examples

Pick 10 two-dimensional points from the figure window.

```
[x,y] = ginput(10)
```

Position the cursor with the mouse (or the arrow keys on terminals without a mouse, such as Tektronix emulators). Enter data points by pressing a mouse button or a key on the keyboard. To terminate input before entering 10 points, press the **Return** key.

See Also

`gtext`

gplot

Purpose Plot set of nodes using an adjacency matrix

Synopsis

```
gplot(A,Coordinates)
gplot(A,Coordinates,LineSpec)
```

Description The gplot function graphs a set of coordinates using an adjacency matrix.

gplot(A,Coordinates) plots a graph of the nodes defined in Coordinates according to the *n*-by-*n* adjacency matrix A, where *n* is the number of nodes. Coordinates is an *n*-by-2 or an *n*-by-3 matrix, where *n* is the number of nodes and each coordinate pair or triple represents one node.

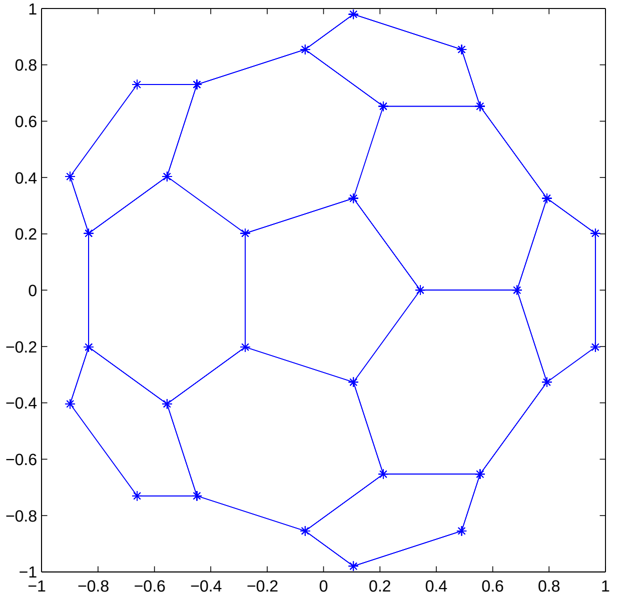
gplot(A,Coordinates,LineSpec) plots the nodes using the line type, marker symbol, and color specified by LineSpec.

Remarks For two-dimensional data, Coordinates(i,:) = [x(i) y(i)] denotes node i, and Coordinates(j,:) = [x(j) y(j)] denotes node j. If node i and node j are joined, A(i,j) or A(j,i) are nonzero; otherwise, A(i,j) and A(j,i) are zero.

Examples

To draw half of a Bucky ball with asterisks at each node:

```
k = 1:30;
[B,XY] = bucky;
gplot(B(k,k),XY(k,:),'-*')
axis square
```



See Also

`LineSpec`, `sparse`, `spy`

graymon

Purpose

Set default figure properties for grayscale monitors

Syntax

```
graymon
```

Description

`graymon` sets defaults for graphics properties to produce more legible displays for grayscale monitors.

See Also

`axes`, `figure`

Purpose Grid lines for two- and three-dimensional plots

Syntax

```
grid on
grid off
grid
```

Description The `grid` function turns the current axes' grid lines on and off.

`grid on` adds grid lines to the current axes.

`grid off` removes grid lines from the current axes.

`grid` toggles the grid visibility state.

Algorithm `grid` sets the `XGrid`, `YGrid`, and `ZGrid` properties of the current axes.

See Also `axes`, `plot`

The `XGrid`, `YGrid`, and `ZGrid` properties of axes objects.

gtext

Purpose Mouse placement of text in two-dimensional view

Syntax

```
gtext('string')
h = gtext('string')
```

Description `gtext` displays a text string in the current figure window after you select a location with the mouse.

`gtext('`*`string`*`')` waits for you to press a mouse button or keyboard key while the pointer is within a figure window. Pressing a mouse button or any key places `'`*`string`*`'` on the plot at the selected location.

`h = gtext('`*`string`*`')` returns the handle to a text graphics object after you place `'`*`string`*`'` on the plot at the selected location.

Remarks As you move the pointer into a figure window, the pointer becomes a crosshair to indicate that `gtext` is waiting for you to select a location. `gtext` uses the functions `ginput` and `text`.

Examples Place a label on the current plot:

```
gtext('Note this divergence!')
```

See Also `ginput`, `text`

Purpose Create a help dialog box

Syntax

```
helpdlg
helpdlg('helpstring')
helpdlg('helpstring','dlgname')
h = helpdlg(...)
```

Description

`helpdlg` creates a help dialog box or brings the named help dialog box to the front.

`helpdlg` displays a dialog box named `'Help Dialog'` containing the string `'This is the default help string.'`

`helpdlg('helpstring')` displays a dialog box named `'Help Dialog'` containing the string specified by `'helpstring'`.

`helpdlg('helpstring','dlgname')` displays a dialog box named `'dlgname'` containing the string `'helpstring'`.

`h = helpdlg(...)` returns the handle of the dialog box.

Remarks

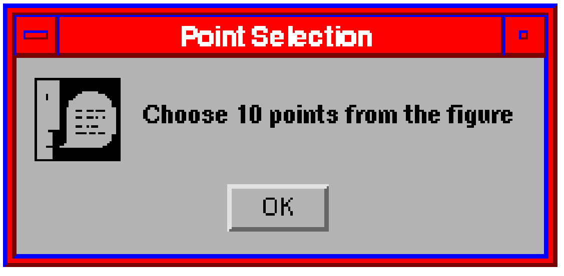
MATLAB wraps the text in `'helpstring'` to fit the width of the dialog box. The dialog box remains on your screen until you press the OK button or the **Return** key. After pressing the button, the help dialog box disappears.

Examples

The statement,

```
helpdlg('Choose 10 points from the figure','Point Selection');
```

displays this dialog box:



See Also `dialog`, `errordlg`, `questdlg`, `warndlg`

Purpose Remove hidden lines from a mesh plot

Syntax

```
hidden on
hidden off
hidden
```

Description Hidden line removal draws only those lines that are not obscured by other objects in the field of view.

`hidden on` turns on hidden line removal for the current graph so lines in the back of a mesh are hidden by those in front. This is the default behavior.

`hidden off` turns off hidden line removal for the current graph.

`hidden` toggles the hidden line removal state.

Algorithm `hidden on` sets the `FaceColor` property of a surface graphics object to the background `Color` of the axes (or of the figure if axes `Color` is none).

Examples Set hidden line removal off and on while displaying the peaks function.

```
mesh(peaks)
hidden off
hidden on
```

See Also `shading`, `mesh`

The surface properties `FaceColor` and `EdgeColor`

Purpose Histogram plot

Syntax

```
n = hist(Y)
n = hist(Y,x)
n = hist(Y,nbins)
[n,xout] = hist(...)
```

Description A histogram shows the distribution of data values.

`n = hist(Y)` bins the elements in `Y` into 10 equally spaced containers and returns the number of elements in each container. If `Y` is a matrix, `hist` works down the columns.

`n = hist(Y,x)` where `x` is a vector, returns the distribution of `Y` among `length(x)` bins with centers specified by `x`. For example, if `x` is a 5-element vector, `hist` distributes the elements of `Y` into five bins centered on the *x*-axis at the elements in `x`. Note: use `histc` if it is more natural to specify bin edges instead of centers.

`n = hist(Y,nbins)` where `nbins` is a scalar, uses `nbins` number of bins.

`[n,xout] = hist(...)` returns vectors `n` and `xout` containing the frequency counts and the bin locations. You can use `bar(xout,n)` to plot the histogram.

`hist(...)` without output arguments, `hist` produces a histogram plot of the output described above. `hist` distributes the bins along the x-axis between the minimum and maximum values of `Y`.

Remarks All elements in vector `Y` or in one column of matrix `Y` are grouped according to their numeric range. Each group is shown as one bin.

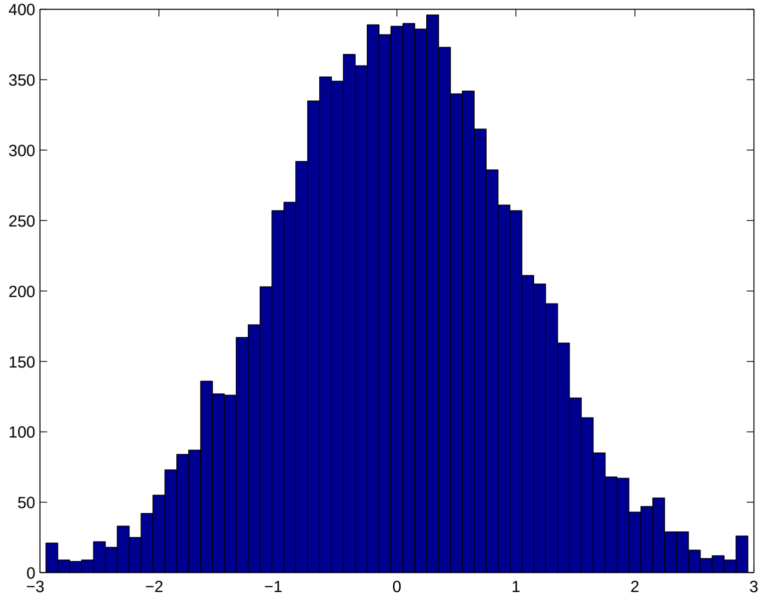
The histogram's *x*-axis reflects the range of values in `Y`. The histogram's *y*-axis shows the number of elements that fall within the groups; therefore, the *y*-axis ranges from `0` to the greatest number of elements deposited in any bin.

The histogram is created with a patch graphics object. If you want to change the color of the graph, you can set patch properties. See the "Example" section for more information. By default, the graph color is controlled by the current colormap, which maps the bin color to the first color in the colormap.

Examples

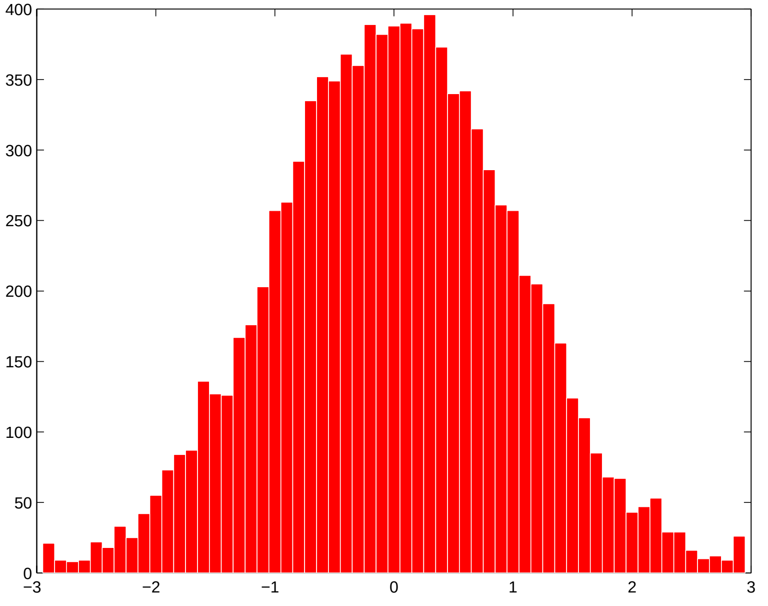
Generate a bell-curve histogram from Gaussian data.

```
x = -2.9:0.1:2.9;
y = randn(10000,1);
hist(y,x)
```



Change the color of the graph so that the bins are red and the edges of the bins are white.

```
h = findobj(gca,'Type','patch');
set(h,'FaceColor','r','EdgeColor','w')
```



See Also

`bar`, `ColorSpec`, `histc`, `patch`, `stairs`

Purpose Hold current graph in the figure

Syntax

```
hold on
hold off
hold
```

Description The hold function determines whether new graphics objects are added to the graph or replace objects in the graph.

hold on retains the current plot and certain axes properties so that subsequent graphing commands add to the existing graph.

hold off resets axes properties to their defaults before drawing new plots. hold off is the default.

hold toggles the hold state between adding to the graph and replacing the graph.

Remarks Test the hold state using the ishold function.

Although the hold state is on, some axes properties change to accommodate additional graphics objects. For example, the axes' limits increase when the data requires them to do so.

The hold function sets the NextPlot property of the current figure and the current axes. If several axes objects exist in a figure window, each axes has its own hold state. hold also creates an axes if one does not exist.

hold on sets the NextPlot property of the current figure and axes to add.

hold off sets the NextPlot property of the current axes to replace.

hold toggles the NextPlot property between the add and replace states.

See Also axis, cla, ishold, newplot

The NextPlot property of axes and figure graphics objects.

hsv2rgb

Purpose Convert HSV colormap to RGB colormap

Syntax

```
M = hsv2rgb(H)
```

Description `M = hsv2rgb(H)` converts a hue-saturation-value (HSV) colormap to a red-green-blue (RGB) colormap. `H` is an *m*-by-3 matrix, where *m* is the number of colors in the colormap. The columns of `H` represent hue, saturation, and value, respectively. `M` is an *m*-by-3 matrix. Its columns are intensities of red, green, and blue, respectively.

Remarks As `H(:,1)` varies from 0 to 1, the resulting color varies from red through yellow, green, cyan, blue, and magenta, and returns to red. When `H(:,2)` is 0, the colors are unsaturated (i.e., shades of gray). When `H(:,2)` is 1, the colors are fully saturated (i.e., they contain no white component). As `H(:,3)` varies from 0 to 1, the brightness increases.

The MATLAB `hsv` colormap uses `hsv2rgb([hue saturation value])` where `hue` is a linear ramp from 0 to 1, and `saturation` and `value` are all 1's.

See Also `brighten`, `colormap`, `rgb2hsv`

Purpose Convert indexed image into movie format

Syntax

```
F = im2frame(X,Map)
```

Description

`F = im2frame(X,Map)` converts the indexed image `X` and associated colormap `Map` into a movie frame F. If X is a truecolor (m-by-n-by-3) image, then `MAP` is optional and has no affect.

`F = im2frame(X)` converts the indexed image `X` into a movie frame `F` using the current colormap.

Example

To use `im2frame` to convert a sequence of images into a movie, first pre-allocate matrix using `movein`.

```
F(1) = im2frame(X1,map);
F(2) = im2frame(X2,map);
...
F(n) = im2frame(Xn,map);
movie(F)
```

See Also

`getframe`, `frame2im`, `movie`

image

Purpose Display image object

Syntax

```
image(C)
image(x,y,C)
image(...,'PropertyName',PropertyValue,...)
image('PropertyName',PropertyValue,...)  Formal synatx – PN/PV only
handle = image(...)
```

Description

`image` creates an image graphics object by interpreting each element in a matrix as an index into the figure's colormap or directly as RGB values, depending on the data specified.

The `image` function has two forms:

- A high-level function that calls `newplot` to determine where to draw the graphics objects and sets the following axes properties:

  `XLim` and `YLim` to enclose the image

  `Layer` to `top` to place the image in front of the tick marks and grid lines

  `YDir` to `reverse`

  `View` to `[0 90]`
- A low-level function that adds the image to the current axes without calling `newplot`. The low-level function argument list can contain only property name/property value pairs.

You can specify properties as property name/property value pairs, structure arrays, and cell arrays (see `set` and `get` for examples of how to specify these data types).

`image(C)` displays matrix `C` as an image. Each element of `C` specifies the color of a rectangular segment in the image.

`image(x,y,C)` where `x` and `y` are two-element vectors, specifies the range of the *x*- and *y*-axis labels, but produces the same image as `image(C)`. This can be useful, for example, if you want the axis tick labels to correspond to real physical dimensions represented by the image.

`image(x,y,C,'`*`PropertyName`*`',PropertyValue,...)` is a high-level function that also specifies property name/property value pairs. This syntax calls `newplot` before drawing the image.

`image('`*`PropertyName`*`',PropertyValue,...)` is the low-level syntax of the `image` function. It specifies only property name/property value pairs as input arguments.

`handle = image(...)` returns the handle of the image object it creates. You can obtain the handle with all forms of the `image` function.

Remarks

image data can be either indexed or true color. An indexed image stores colors as an array of indices into the figure colormap. A true color image does not use a colormap; instead, the color values for each pixel are stored directly as RGB triplets. In MATLAB , the `CData` property of a truecolor image object is a three-dimensional (m-by-n-by-3) array. This array consists of three m-by-n matrices (representing the red, green, and blue color planes) concatenated along the third dimension.

The `imread` function reads image data into MATLAB arrays from graphics files in various standard formats, such as TIFF. You can write MATLAB image data to graphics files using the `imwrite` function. `imread` and `imwrite` both support a variety of graphics file formats and compression schemes.

When you read image data into MATLAB using `imread`, the data is usually stored as an array of 8-bit integers. However, `imread` also supports reading 16-bit-per-pixel data from TIFF and PNG files. These are more efficient storage method than the double-precision (64-bit) floating-point numbers that MATLAB typically uses. However, it is necessary for MATLAB to interpret 8-bit and 16-bit image data differently from 64-bit data. This table summarizes these differences.

Image Type	Double-precision Data (double array)	8-bit Data (uint8 array) 16-bit Data (uint16 array)
indexed (colormap)	Image is stored as a two-dimensional (m-by-n) array of integers in the range [1, length(colormap)]; colormap is an m-by-3 array of floating-point values in the range [0, 1]	Image is stored as a two-dimensional (m-by-n) array of integers in the range [0, 255] (unit8) or [0, 65535] (uint16); colormap is an m-by-3 array of floating-point values in the range [0, 1]
truecolor (RGB)	Image is stored as a three-dimensional (m-by-n-by-3) array of floating-point values in the range [0, 1]	Image is stored as a three-dimensional (m-by-n-by-3) array of integers in the range [0, 255] (unit8) or [0, 65535] (uint16)

Indexed Images

In an indexed image of class double, the value 1 points to the first row in the colormap, the value 2 points to the second row, and so on. In a uint8 or uint16 indexed image, there is an offset; the value 0 points to the first row in the colormap, the value 1 points to the second row, and so on.

If you want to convert a uint8 or uint16 indexed image to double, you need to add 1 to the result. For example,

```
X64 = double(X8) + 1;
```

or

```
X64 = double(X16) + 1;
```

To convert from double to uint8 or unit16, you need to first subtract 1, and then use round to ensure all the values are integers.

```
X8 = uint8(round(X64 – 1));
```

or

```
X16 = uint16(round(X64 – 1));
```

The order of the operations must be as shown in these examples, because you cannot perform mathematical operations on uint8 or uint16 arrays.

When you write an indexed image using `imwrite`, MATLAB automatically converts the values if necessary.

Colormaps

Colormaps in MATLAB are alway `m`-by-3 arrays of double-precision floating-point numbers in the range [0, 1]. In most graphics file formats, colormaps are stored as integers, but MATLAB does not support colormaps with integer values. `imread` and `imwrite` automatically convert colormap values when reading and writing files.

True Color Images

In a truecolor image of class `double`, the data values are floating-point numbers in the range [0, 1]. In a truecolor image of class `uint8`, the data values are integers in the range [0, 255] and for truecolor image of class `uint16` the data values are integers in the range [0, 65535]

If you want to convert a truecolor image from one data type to the other, you must rescale the data. For example, this statement converts a `uint8` truecolor image to `double`,

```
RGB64 = double(RGB8)/255;
```

or for `uint16` images,

```
RGB64 = double(RGB16)/65535;
```

This statement converts a `double` truecolor image to `uint8`.

```
RGB8 = uint8(round(RGB64*255));
```

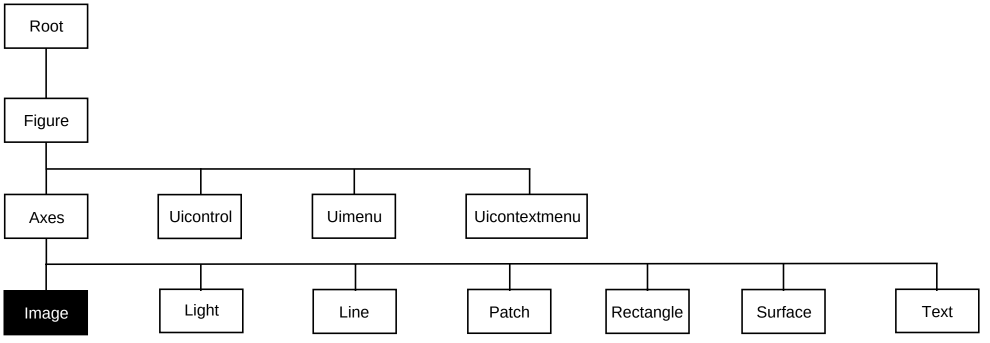
or for `uint16` images,

```
RGB16 = uint16(round(RGB64*65535));
```

The order of the operations must be as shown in these examples, because you cannot perform mathematical operations on `uint8` or `uint16` arrays.

When you write a truecolor image using `imwrite`, MATLAB automatically converts the values if necessary.

Object Hierarchy



Setting Default Properties

You can set default image properties on the axes, figure, and root levels.

```
set(0,'DefaultImageProperty',PropertyValue...)
set(gcf,'DefaultImageProperty',PropertyValue...)
set(gca,'DefaultImageProperty',PropertyValue...)
```

Where *Property* is the name of the image property and *PropertyValue* is the value you are specifying. Use `set` and `get` to access image properties.

The following table lists all image properties and provides a brief description of each. The property name links take you to an expanded description of the properties.

Property Name	Property Description	Property Value
Data Defining the Object		
`CData`	The image data	Values: matrix or m-by-n-by-3 array Default: enter `image;axis image ij` and see
`CDataMapping`	Specify the mapping of data to colormap	Values: `scaled`, `direct` Default: `direct`
`XData`	Control placement of image along *x*-axis	Values: `[min max]` Default: `[1 size(CData,2)]`

Property Name	Property Description	Property Value
YData	Control placement of image along *y*-axis	Values: [min max] Default: [1 size(CData,1)]
Controlling the Appearance		
Clipping	Clipping to axes rectangle	Values: on, off Default: on
EraseMode	Method of drawing and erasing the image (useful for animation)	Values: normal, none, xor, background Default: normal
SelectionHighlight	Highlight image when selected (Selected property set to on)	Values: on, off Default: on
Visible	Make the image visible or invisible	Values: on, off Default: on
Controlling Access to Objects		
HandleVisibility	Determines if and when the the line's handle is visible to other functions	Values: on, callback, off Default: on
HitTest	Determine if image can become the current object (see the figure CurrentObject property)	Values: on, off Default: on
General Information About the Image		
Children	Image objects have no children	Values: [] (empty matrix)
Parent	The parent of an image object is always an axes object	Value: axes handle
Selected	Indicate whether image is in a "selected" state.	Values: on, off Default: on
Tag	User-specified label	Value: any string Default: '' (empty string)
Type	The type of graphics object (read only)	Value: the string 'image'

Property Name	Property Description	Property Value
UserData	User-specified data	Value: any matrix Default: [] (empty matrix)
Properties Related to Callback Routine Execution		
BusyAction	Specify how to handle callback routine interruption	Values: cancel, queue Default: queue
ButtonDownFcn	Define a callback routine that executes when a mouse button is pressed on over the image	Values: string Default: empty string
CreateFcn	Define a callback routine that executes when an image is created	Values: string Default: empty string
DeleteFcn	Define a callback routine that executes when the image is deleted (via close or delete)	Values: string Default: empty string
Interruptible	Determine if callback routine can be interrupted	Values: on, off Default: on (can be interrupted)
UIContextMenu	Associate a context menu with the image	Values: handle of a uicontextmenu

Image Properties

This section lists property names along with the types of values each property accepts.

BusyAction `cancel | {queue}`

Callback routine interruption. The `BusyAction` property enables you to control how MATLAB handles events that potentially interrupt executing callback routines. If there is a callback routine executing, subsequently invoked callback routes always attempt to interrupt it. If the `Interruptible` property of the object whose callback is executing is set to `on` (the default), then interruption occurs at the next point where the event queue is processed. If the `Interruptible` property is `off`, the `BusyAction` property (of the object owning the executing callback) determines how MATLAB handles the event. The choices are:

- `cancel` – discard the event that attempted to execute a second callback routine.
- `queue` – queue the event that attempted to execute a second callback routine until the current callback finishes.

ButtonDownFcn string

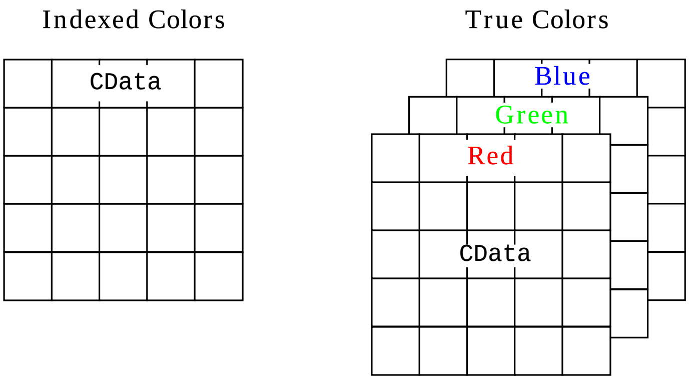
Button press callback routine. A callback routine that executes whenever you press a mouse button while the pointer is over the image object. Define this routine as a string that is a valid MATLAB expression or the name of an M-file. The expression executes in the MATLAB workspace.

CData matrix or m-by-n-by-3 array

The image data. A matrix of values specifying the color of each rectangular area defining the image. `image(C)` assigns the values of `C` to `CData`. MATLAB determines the coloring of the image in one of three ways:

- Using the elements of `CData` as indices into the current colormap (the default)
- Scaling the elements of `CData` to range between the values `min(get(gca,'CLim'))` and `max(get(gca,'CLim'))` (`CDataMapping` set to `scaled`)
- Interpreting the elements of `CData` directly as RGB values (true color specification)

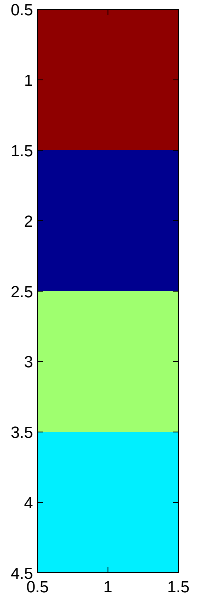
A true color specification for CData requires an m-by-n-by-3 array of RGB values. The first page contains the red component, the second page the green component, and the third page the blue component of each element in the image. RGB values range from 0 to 1. The following picture illustrates the relative dimensions of CData for the two color models.



If CData has only one row or column, the height or width respectively is always one data unit and is centered about the first YData or XData element respectively. For example, using a 4-by-1 matrix of random data,

```
C = rand(4,1);
image(C,'CDataMapping','scaled')
axis image
```

produces:



CDataMapping `scaled | {direct}`

Direct or scaled indexed colors. This property determines whether MATLAB interprets the values in `CData` as indices into the figure colormap (the default) or scales the values according to the values of the axes `CLim` property.

When `CDataMapping` is `direct`, the values of `CData` should be in the range 1 to `length(get(gcf,'Colormap'))`. If you use true color specification for `CData`, this property has no effect.

Children handles

The empty matrix; image objects have no children.

Clipping `on | off`

Clipping mode. By default, MATLAB clips images to the axes rectangle. If you set `Clipping` to `off`, the image can display outside the axes rectangle. For example, if you create an image, set `hold` to `on`, freeze axis scaling (`axis manual`), and then create a larger image, it extends beyond the axis limits.

CreateFcn string

Callback routine executed during object creation. This property defines a callback routine that executes when MATLAB creates an image object. You must define this property as a default value for images. For example, the statement,

```
set(0,'DefaultImageCreateFcn','axis image')
```

defines a default value on the root level that sets the aspect ratio and the axis limits so the image has square pixels. MATLAB executes this routine after setting all image properties. Setting this property on an existing image object has no effect.

The handle of the object whose `CreateFcn` is being executed is accessible only through the root `CallbackObject` property, which you can query using `gcbo`.

DeleteFcn string

Delete image callback routine. A callback routine that executes when you delete the image object (i.e., when you issue a `delete` command or clear the axes or figure containing the image). MATLAB executes the routine before destroying the object's properties so these values are available to the callback routine.

The handle of the object whose `DeleteFcn` is being executed is accessible only through the root `CallbackObject` property, which you can query using `gcbo`.

EraseMode `{normal} | none | xor | background`

Erase mode. This property controls the technique MATLAB uses to draw and erase image objects. Alternative erase modes are useful for creating animated sequences, where control of the way individual objects redraw is necessary to improve performance and obtain the desired effect.

- `normal` (the default) — Redraw the affected region of the display, performing the three-dimensional analysis necessary to ensure that all objects are rendered correctly. This mode produces the most accurate picture, but is the slowest. The other modes are faster, but do not perform a complete redraw and are therefore less accurate.
- `none` – Do not erase the image when it is moved or changed. While the object is still visible on the screen after erasing with `EraseMode` `none`, you cannot print it because MATLAB stores no information about its former location.

- `xor` – Draw and erase the image by performing an exclusive OR (XOR) with the color of the screen beneath it. This mode does not damage the color of the objects beneath the image. However, the image's color depends on the color of whatever is beneath it on the display.
- `background` – Erase the image by drawing it in the axes' background `Color`, or the figure background `Color` if the axes `Color` is set to `none`. This damages objects that are behind the erased image, but images are always properly colored.

Printing with Non-normal Erase Modes. MATLAB always prints figures as if the `EraseMode` of all objects is `normal`. This means graphics objects created with `EraseMode` set to `none`, `xor`, or `background` can look different on screen than on paper. On screen, MATLAB may mathematically combine layers of colors (e.g., XORing a pixel color with that of the pixel behind it) and ignore three-dimensional sorting to obtain greater rendering speed. However, these techniques are not applied to the printed output.

You can use the MATLAB `getframe` command or other screen capture application to create an image of a figure containing non-normal mode objects.

`HandleVisibility` `{on} | callback | off`

Control access to object's handle by command-line users and GUIs. This property determines when an object's handle is visible in its parent's list of children. `HandleVisibility` is useful for preventing command-line users from accidentally drawing into or deleting a figure that contains only user interface devices (such as a dialog box).

Handles are always visible when `HandleVisibility` is `on`.

Setting `HandleVisibility` to `callback` causes handles to be visible from within callback routines or functions invoked by callback routines, but not from within functions invoked from the command line. This provide a means to protect GUIs from command-line users, while allowing callback routines to have complete access to object handles.

Setting `HandleVisibility` to `off` makes handles invisible at all times. This may be necessary when a callback routine invokes a function that might potentially damage the GUI (such as evaling a user-typed string), and so temporarily hides its own handles during the execution of that function.

When a handle is not visible in its parent's list of children, it cannot be returned by functions that obtain handles by searching the object hierarchy or querying handle properties. This includes `get`, `findobj`, `gca`, `gcf`, `gco`, `newplot`, `cla`, `clf`, and `close`.

When a handle's visibility is restricted using `callback` or `off`, the object's handle does not appear in its parent's `Children` property, figures do not appear in the root's `CurrentFigure` property, objects do not appear in the root's `CallbackObject` property or in the figure's `CurrentObject` property, and axes do not appear in their parent's `CurrentAxes` property.

You can set the root `ShowHiddenHandles` property to on to make all handles visible, regardless of their `HandleVisibility` settings (this does not affect the values of the `HandleVisibility` properties).

Handles that are hidden are still valid. If you know an object's handle, you can `set` and `get` its properties, and pass it to any function that operates on handles.

`HitTest` `{on} | off`

Selectable by mouse click. `HitTest` determines if the image can become the current object (as returned by the `gco` command and the figure `CurrentObject` property) as a result of a mouse click on the image. If `HiTest` is `off`, clicking on the image selects the object below it (which maybe the axes containing it).

`Interruptible` `{on} | off`

Callback routine interruption mode. The `Interruptible` property controls whether an image callback routine can be interrupted by subsequently invoked callback routines. Only callback routines defined for the `ButtonDownFcn` are affected by the `Interruptible` property. MATLAB checks for events that can interrupt a callback routine only when it encounters a `drawnow`, `figure`, `getframe`, or `pause` command in the routine.

`Parent` handle of parent axes

Image's parent. The handle of the image object's parent axes. You can move an image object to another axes by changing this property to the new axes handle.

`Selected` `on | {off}`

Is object selected? When this property is `on`, MATLAB displays selection handles if the `SelectionHighlight` property is also `on`. You can, for example, define the ButtonDownFcn to set this property, allowing users to select the object with the mouse.

SelectionHighlight {on} | off

Objects highlight when selected. When the Selected property is on, MATLAB indicates the selected state by drawing four edge handles and four corner handles. When SelectionHighlight is off, MATLAB does not draw the handles.

Tag string

User-specified object label. The Tag property provides a means to identify graphics objects with a user-specified label. This is particularly useful when constructing interactive graphics programs that would otherwise need to define object handles as global variables or pass them as arguments between callback routines. You can define Tag as any string.

Type string (read only)

Type of graphics object. This property contains a string that identifies the class of graphics object. For image objects, Type is always 'image'.

UIContextMenu handle of a uicontextmenu object

Associate a context menu with the image. Assign this property the handle of a uicontextmenu object created in the same figure as the image. Use the uicontextmenu function to create the context menu. MATLAB displays the context menu whenever you right-click over the image.

UserData matrix

User specified data. This property can be any data you want to associate with the image object. The image does not use this property, but you can access it using set and get.

Visible {on} | off

Image visibility. By default, image objects are visible. Setting this property to off prevents the image from being displayed. However, the object still exists and you can set and query its properties.

XData [1 size(CData,2)] by default

Control placement of image along x-axis. A vector specifying the locations of the centers of the elements CData(1,1) and CData(m,n), where CData has a size of m-by-n. Element CData(1,1) is centered over the coordinate defined by the first elements in `XData` and `YData`. Element `CData(m,n)` is centered over the coordinate defined by the last elements in `XData` and `YData`. The centers of the remaining elements of `CData` are evenly distributed between those two points.

The width of each `CData` element is determined by the expression:

```
(XData(2)-XData(1))/(size(CData,2)-1)
```

You can also specify a single value for `XData`. In this case, `image` centers the first element at this coordinate and centers each following element one unit apart.

`YData` `[1 size(CData,1)]` by default

Control placement of image along y-axis. A vector specifying the locations of the centers of the elements `CData(1,1)` and `CData(m,n)`, where `CData` has a size of *m*-by-*n*. Element `CData(1,1)` is centered over the coordinate defined by the first elements in `XData` and `YData`. Element `CData(m,n)` is centered over the coordinate defined by the last elements in `XData` and `YData`. The centers of the remaining elements of `CData` are evenly distributed between those two points.

The height of each `CData` element is determined by the expression:

```
(YData(2)-YData(1))/(size(CData,1)-1)
```

You can also specify a single value for `YData`. In this case, `image` centers the first element at this coordinate and centers each following elements one unit apart.

See Also

`colormap`, `imfinfo`, `imread`, `imwrite`, `pcolor`, `newplot`, `surface`

The *Image* chapter the *Using MATLAB Graphics* manual

Purpose Scale data and display an image object

Syntax

```
imagesc(C)
imagesc(x,y,C)
imagesc(...,clims)
h = imagesc(...)
```

Description The `imagesc` function scales image data to the full range of the current colormap and displays the image. (See the illustration on the following page.)

`imagesc(C)` displays `C` as an image. Each element of `C` corresponds to a rectangular area in the image. The values of the elements of `C` are indices into the current colormap that determine the color of each patch.

`imagesc(x,y,C)` displays `C` as an image and specifies the bounds of the *x*- and *y*-axis with vectors `x` and `y`.

`imagesc(...,clims)` normalizes the values in `C` to the range specified by `clims` and displays `C` as an image. `clims` is a two-element vector that limits the range of data values in `C`. These values map to the full range of values in the current colormap.

`h = imagesc(...)` returns the handle for an image graphics object.

Remarks `x` and `y` do not affect the elements in `C`; they only affect the annotation of the axes. If `length(x) > 2` or `length(y) > 2`, `imagesc` ignores all except the first and last elements of the respective vector.

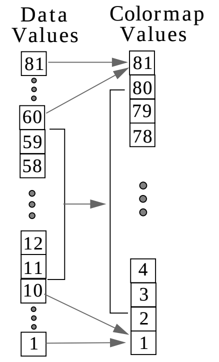
Algorithm `imagesc` creates an image with `CDataMapping` set to `scaled`, and sets the axes `CLim` property to the value passed in `clims`.

Examples

If the size of the current colormap is 81-by-3, the statements

```
clims = [10 60]
imagesc(C,clims)
```

map the data values in C to the colormap, as shown to the right.

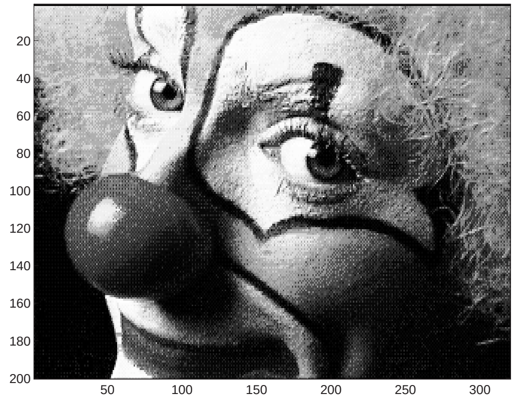


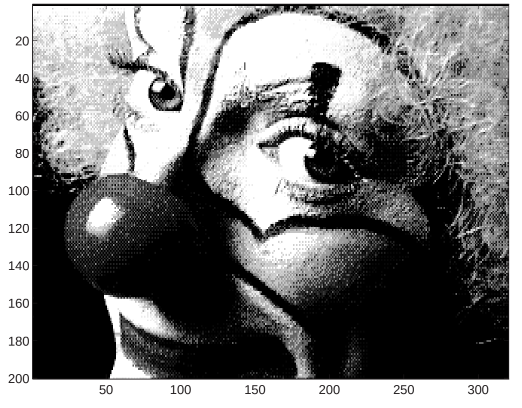
The left image maps to the gray colormap using the statements

```
load clown
imagesc(X)
colormap(gray)
```

The right image has values between `10` and `60` scaled to the full range of the `gray` colormap using the statements

```
load clown
clims = [10 60];
imagesc(X,clims)
colormap(gray)
```





See Also `image`, `colorbar`

ind2rgb

Purpose Convert an indexed image to an RGB image

Syntax

```
RGB = ind2rgb(X,map)
```

Description `RGB = ind2rgb(X,map)` converts thematrix `X` and corresponding colormap `map` to RGB (truecolor) format.

Class Support `X` can be of class `uint8`, `uint16`, or `double`. RGB is an m-by-n-3 array of class `double`.

See Also `image`

Purpose

Create input dialog box

Syntax

```
answer = inputdlg(prompt)
answer = inputdlg(prompt,title)
answer = inputdlg(prompt,title,lineNo)
answer = inputdlg(prompt,title,lineNo,defAns)
answer = inputdlg(prompt,title,lineNo,defAns,Resize)
```

Description

`answer = inputdlg(prompt)` creates a modal dialog box and returns user inputs in the cell array. `prompt` is a cell array containing prompt strings.

`answer = inputdlg(prompt,title)` `title` specifies a title for the dialog box.

`answer = inputdlg(prompt,title,lineNo)` `lineNo` specifies the number of lines for each user entered value. `lineNo` can be a scalar, column vector, or matrix.

- If `lineNo` is a scalar, it applies to all prompts.
- If `lineNo` is a column vector, each element specifies the number of lines of input for a prompt.
- If `lineNo` is a matrix, it should be size m-by-2, where m is the number of prompts on the dialog box. Each row refers to a prompt. The first column specifies the number of lines of input for a prompt. The second column specifies the width of the field in characters.

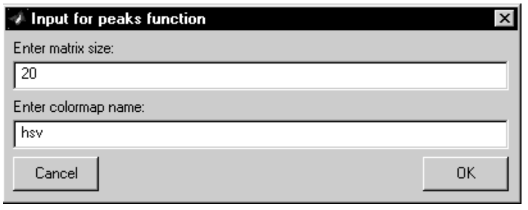
`answer = inputdlg(prompt,title,lineNo,defAns)` `defAns` specifies the default value to display for each prompt. `defAns` must contain the same number of elements as `prompt` and all elements must be strings.

`answer = inputdlg(prompt,title,lineNo,defAns,Resize)` `Resize` specifies whether or not the dialog box can be resized. Permissible values are `'on'` and `'off'` where `'on'` means that the dialog box can be resized and that the dialog box is not modal.

Example

Create a dialog box to input an integer and colormap name. Allow one line for each value.

```
prompt = {'Enter matrix size:','Enter colormap name:'};
title  = 'Input for peaks function';
lines= 1;
def    = {'20','hsv'};
answer = inputdlg(prompt,title,lines,def);
```



See Also

`dialog`, `errordlg`, `helpdlg`, `questdlg`, `warndlg`

Purpose Determines if values are valid graphics object handles

Syntax

```
array = ishandle(h)
```

Description `array = ishandle(h)` returns an array that contains `1`'s where the elements of `h` are valid graphics handles and `0`'s where they are not.

Examples Determine whether the handles previously returned by `fill` remain handles of existing graphical objects:

```
X = rand(4); Y = rand(4);
h = fill(X,Y,'blue')
.
.
.
delete(h(3))
.
.
.
ishandle(h)
ans =
    1
    1
    0
    1
```

See Also `findobj`

ishold

Purpose Return hold state

Syntax

```
k = ishold
```

Description `k = ishold` returns the hold state of the current axes. If `hold` is `on` `k = 1`, if `hold` is `off`, `k = 0`.

Examples `ishold` is useful in graphics M-files where you want to perform a particular action only if `hold` is not `on`. For example, these statements set the view to 3-D only if `hold` is `off`:

```
if ~ishold
   view(3);
end
```

See Also `axes`, `figure`, `hold`, `newplot`

Purpose

Compute isosurface end-cap geometry

Syntax

```
fvc = isocaps(X,Y,Z,V,isovalue)
fvc = isocaps(V,isovalue)
fvc = isocaps(...,'enclose')
fvc = isocaps(...,'whichplane')
[f,v,c] = isocaps(...)
isocaps(...)
```

Description

`fvc = isocaps(X,Y,Z,V,isovalue)` computes isosurface end cap geometry for the volume data `V` at isosurface value `isovalue`. The arrays `X`, `Y`, and `Z` define the coordinates for the volume `V`.

The struct `fvc` contains the face, vertex, and color data for the end caps and can be passed directly to the `patch` command.

`fvc = isocaps(V,isovalue)` assumes the arrays `X`, `Y`, and `Z` are defined as `[X,Y,Z] = meshgrid(1:n,1:m,1:p)` where `[m,n,p] = size(V)`.

`fvc = isocaps(...,'`*`enclose`*`')` specifies whether the end caps enclose data values above or below the value specified in `isovalue`. The string *enclose* can be either `above` (default) or `below`.

`fvc = isocaps(...,'`*`whichplane`*`')` specifies on which planes to draw the end caps. Possible values for *whichplane* are: `all` (default), `xmin`, `xmax`, `ymin`, `ymax`, `zmin`, or `zmax`.

`[f,v,c] = isocaps(...)` returns the face, vertex, and color data for the end caps in three arrays instead of the struct `fvc`.

`isocaps(...)` without output arguments draws a patch with the computed faces, vertices, and colors.

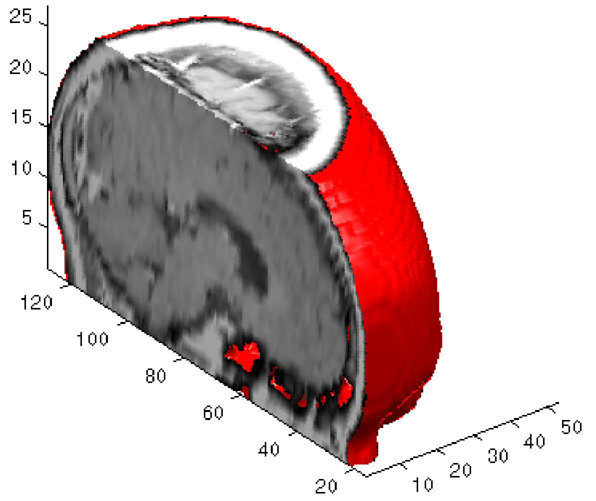
Examples

This example uses a data set that is a collection of MRI slices of a human skull. It illustrates the use of `isocaps` to draw the end caps on this cut-away volume.

The red `isosurface` shows the outline of the volume (skull) and the end caps show what is inside of the volume.

The `patch` created from the end cap data (`p2`) uses interpolated face coloring, which means the gray `colormap` and the light sources determine how it is colored. The isosurface patch (p1) used a flat red face color, which is affected by the lights, but does not use the colormap.

```
load mri
D = squeeze(D);
D(:,1:60,:) = [];
p1 = patch(isosurface(D, 5),'FaceColor','red',...
    'EdgeColor','none');
p2 = patch(isocaps(D, 5),'FaceColor','interp',...
    'EdgeColor','none');
view(3); axis tight; daspect([1,1,.4])
colormap(gray(100))
camlight left; camlight; lighting gouraud
isonormals(D,p1)
```



See Also isosurface, isonormals, smooth3, subvolume, reducevolume, reducepatch

Purpose

Compute normals of isosurface vertices

Syntax

```
n = isonormals(X,Y,Z,V,vertices)
n = isonormals(V,vertices)
n = isonormals(V,p), n = isonormals(X,Y,Z,V,p)
n = isonormals(...,'negate')
isonormals(V,p), isonormals(X,Y,Z,V,p)
```

Description

`n = isonormals(X,Y,Z,V,vertices)` computes the normals of the isosurface vertices from the vertex list, `vertices`, using the gradient of the data `V`. The arrays `X`, `Y`, and `Z` define the coordinates for the volume `V`. The computed normals are returned in `n`.

`n = isonormals(V,vertices)` assumes the arrays `X`, `Y`, and `Z` are defined as `[X,Y,Z] = meshgrid(1:n,1:m,1:p)` where `[m,n,p] = size(V)`.

`n = isonormals(V,p)` and `n = isonormals(X,Y,Z,V,p)` compute normals from the vertices of the patch identified by the handle `p`.

`n = isonormals(...,'negate')` negates (reverses the direction of) the normals.

`isonormals(V,p)` and `isonormals(X,Y,Z,V,p)` set the `VertexNormals` property of the patch identified by the handle `p` to the computed normals rather than returning the values.

Examples

This example compares the effect of different surface normals on the visual appearance of lit isosurfaces. In one case, the triangles used to draw the isosurface define the normals. In the other, the `isonormals` function uses the volume data to calculate the vertex normals based on the gradient of the data points. The latter approach generally produces a smoother-appearing isosurface.

Define a 3-D array of volume data (`cat`, `interp3`):

```
data = cat(3, [0 .2 0; 0 .3 0; 0 0 0], ...
              [.1 .2 0; 0 1 0; .2 .7 0],...
              [0 .4 .2; .2 .4 0;.1 .1 0]);
data = interp3(data,3,'cubic');
```

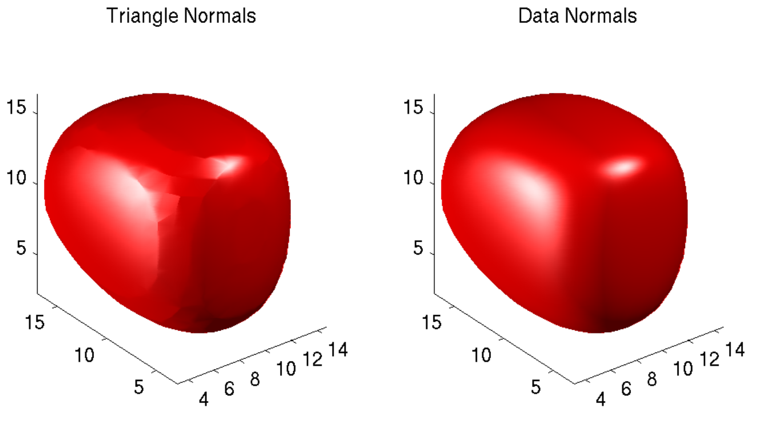
Draw an isosurface from the volume data and add lights. This isosurface uses triangle normals (`patch`, `isosurface`, `view`, `daspect`, `axis`, `camlight`, `lighting`, `title`):

```
subplot(1,2,1)
p1 = patch(isosurface(data,.5),...
'FaceColor','red','EdgeColor','none');
view(3); daspect([1,1,1]); axis tight
camlight; camlight(-80,-10); lighting phong;
title('Triangle Normals')
```

Draw the same lit isosurface using normals calculated from the volume data:

```
subplot(1,2,2)
p2 = patch(isosurface(data,.5),...
    'FaceColor','red','EdgeColor','none');
isonormals(data,p2)
view(3); daspect([1 1 1]); axis tight
camlight;  camlight(-80,-10); lighting phong;
title('Data Normals')
```

These isosurfaces illustrate the difference between triangle and data normals:



See Also

`interp3`, `isosurface`, `isocaps`, `smooth3`, `subvolume`, `reducevolume`, `reducepatch`

Purpose

Extract isosurface data from volume data

Syntax

```
fv = isosurface(X,Y,Z,V,isovalue)
fv = isosurface(V,isovalue)
fv = isosurface(X,Y,Z,V), fv = isosurface(X,Y,Z,V)
fv = isosurface(...,'noshare')
fv = isosurface(...,'verbose')
[f,v] = isosurface(...)
isosurface(...)
```

Description

`fv = isosurface(X,Y,Z,V,isovalue)` computes isosurface data from the volume data `V` at the isosurface value specified in `isovalue`. The arrays `X`, `Y`, and `Z` define the coordinates for the volume `V`. The struct `fv` contains the faces and vertices of the isosurface, which you can pass directly to the `patch` command.

`fv = isosurface(V,isovalue)` assumes the arrays `X`, `Y`, and `Z` are defined as `[X,Y,Z] = meshgrid(1:n,1:m,1:p)` where `[m,n,p] = size(V)`.

`fv = isosurface(...,'noshare')` does not create shared vertices. This is faster, but produces a larger set of vertices.

`fv = isosurface(...,'verbose')` prints progress messages to the command window as the computation progresses.

`[f,v] = isosurface(...)` returns the faces and vertices in two arrays instead of a struct.

`isosurface(...)` with no output arguments creates a patch using the computed faces and vertices.

Remarks

You can pass the `fv` structure created by `isosurface` directly to the `patch` command, but you cannot pass the individual faces and vertices arrays (`f`, `v`) to `patch` without specifying property names. For example,

```
patch(isosurface(X,Y,Z,V,isovalue))
```

or

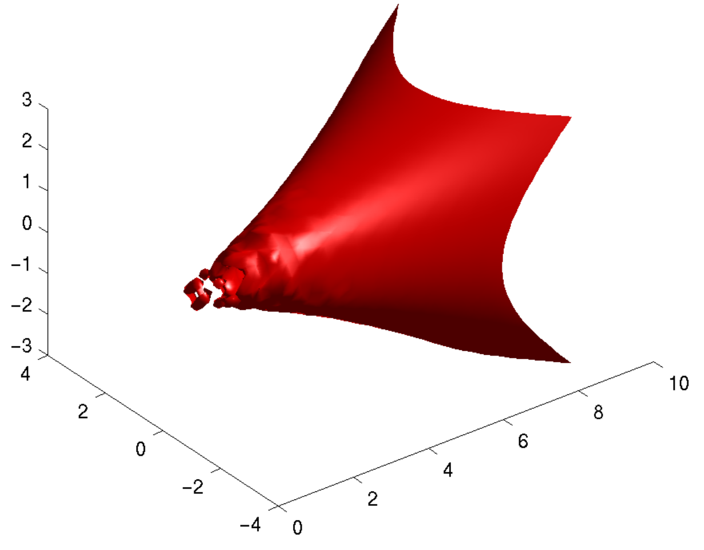
```
[f,v] = isosurface(X,Y,Z,V,isovalue);
patch('Faces',f,'Vertices',v)
```

Examples

This example uses the flow data set, which represents the speed profile of a submerged jet within an infinite tank (type `help flow` for more information). The isosurface is drawn at the data value of -3. The statements that follow the `patch` command prepare the isosurface for lighting by:

- Recalculating the isosurface normals based on the volume data (`isonormals`)
- Setting the face and edge color (`set`, `FaceColor`, `EdgeColor`)
- Specifying the view (`daspect`, `view`)
- Adding lights (`camlight`, `lighting`)

```
[x,y,z,v] = flow;
p = patch(isosurface(x,y,z,v,-3));
isonormals(x,y,z,v,p)
set(p,'FaceColor','red','EdgeColor','none');
daspect([1 1 1])
view(3)
camlight
lighting phong
```



See Also

`isonormals`, `isocaps`, `reducepatch`, `reducevolume`, `shrinkfaces`, `smooth3`, `subvolume`

legend

Purpose Display a legend on graphs

Syntax

```
legend('string1','string2',...)
legend(h,'string1','string2',...)
legend(string_matrix)
legend(h,string_matrix)
legend(axes_handle,...)
legend('off')
legend(h,...)
legend(...,pos)
h = legend(...)
[legend_handle,object_handles] = legend(...)
```

Description

`legend` places a legend on various types of graphs (line plots, bar graphs, pie charts, etc.). For each line plotted, the legend shows a sample of the line type, marker symbol, and color beside the text label you specify. When plotting filled areas (patch or surface objects), the legend contains a sample of the face color next to the text label.

`legend('string1','string2',...)` displays a legend in the current axes using the specified strings to label each set of data.

`legend(h,'string1','string2',...)` displays a legend on the plot containing the handles in the vector `h`, using the specified strings to label the corresponding graphics object (line, bar, etc.).

`legend(string_matrix)` adds a legend containing the rows of the matrix `string_matrix` as labels. This is the same as `legend(string_matrix(1,:),string_matrix(2,:),...)`.

`legend(h,string_matrix)` associates each row of the matrix `string_matrix` with the corresponding graphics object in the vector `h`.

`legend(axes_handle,...)` displays the legend for the axes specified by `axes_handle`.

`legend('off'),legend(axes_handle,'off')` removes the legend from the current axes or the axes specified by `axes_hanlde`.

`legend_handle = legend` returns the handle to the legend on the current axes or an empty vector if no legend exists.

`legend` with no arguments refreshes all the legends in the current figure.

`legend(legend_handle)` refreshes the specified legend.

`legend(...,pos)` uses `pos` to determine where to place the legend.

- `pos = -1` places the legend outside the axes boundary on the right side.
- `pos = 0` places the legend inside the axes boundary, obscuring as few points as possible.
- `pos = 1` places the legend in the upper-right corner of the axes (default).
- `pos = 2` places the legend in the upper-left corner of the axes.
- `pos = 3` places the legend in the lower-left corner of the axes.
- `pos = 4` places the legend in the lower-right corner of the axes.

`[legend_handle,object_handles] = legend(...)` returns the handle of the legend (`legend_handle`), which is an axes graphics object and the handles of the line, patch and text graphics objects (`object_handles`) used in the legend. These handles enable you to modify the properties of the respective objects.

Remarks

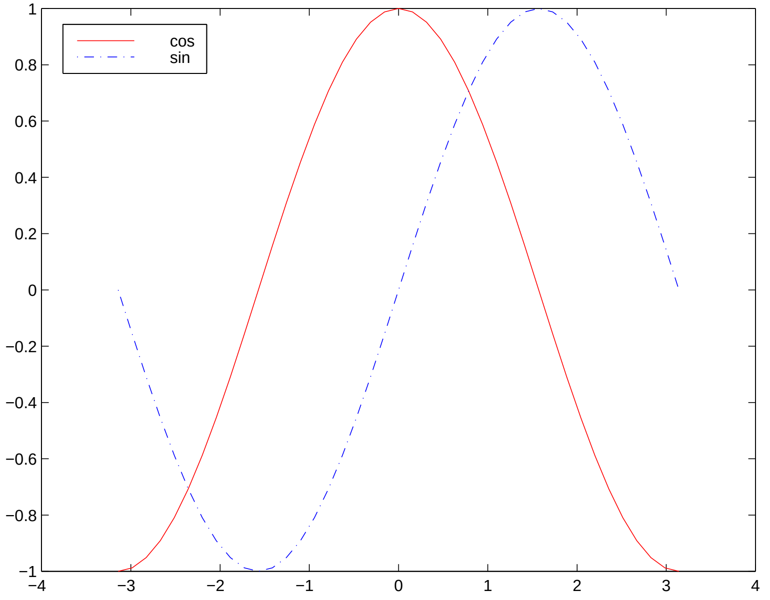
`legend` associates strings with the objects in the axes in the same order that they are listed in the axes `Children` property. By default, the legend annotates the current axes.

MATLAB displays only one legend per axes. `legend` positions the legend based on a variety of factors, such as what objects the legend obscures. You move the legend by pressing the left mouse button while the cursor is over the legend and dragging the legend to a new location. Double clicking on a label allows you to edit the label.

Examples

Add a legend to a graph showing a sine and cosine function:

```
x = -pi:pi/20:pi;
plot(x,cos(x),'-ro',x,sin(x),'-.b')
h = legend('cos','sin',2);
```



In this example, the `plot` command specifies a solid, red line (`'-r'`) for the cosine function and a dash-dot, blue line (`'-.b'`) for the sine function.

See Also

`LineSpec`, `plot`

Purpose

Create a light object

Syntax

```
light('PropertyName',PropertyValue,...)
handle = light(...)
```

Description

`light` creates a light object in the current axes. lights affect only patch and surface object.

`light('PropertyName',PropertyValue,...)` creates a light object using the specified values for the named properties. MATLAB parents the light to the current axes unless you specify another axes with the `Parent` property.

`handle = light(...)` returns the handle of the light object created.

Remarks

You cannot see a light object *per se*, but you can see the effects of the light source on patch and surface objects. You can also specify an axes-wide ambient light color that illuminates these objects. However, ambient light is visible only when at least one light object is present and visible in the axes.

You can specify properties as property name/property value pairs, structure arrays, and cell arrays (see `set` and `get` for examples of how to specify these data types).

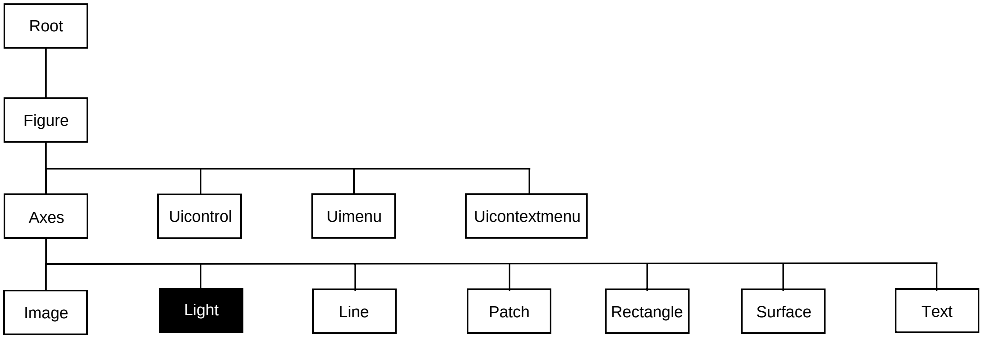
See also the patch and surface `AmbientStrength`, `DiffuseStrength`, `SpecularStrength`, `SpecularExponent`, `SpecularColorReflectance`, and `VertexNormals` properties. Also see the `lighting` and `material` commands.

Examples

Light the `peaks` surface plot with a light source located at infinity and oriented along the direction defined by the vector `[1 0 0]`, that is, along the *x*-axis.

```
h = surf(peaks);
set(h,’FaceLighting’,’phong’,'FaceColor',’interp’,...
      'AmbientStrength',0.5)
light('Position’,[1 0 0],’Style’,’infinite’);
```

Object Hierarchy



Setting Default Properties

You can set default light properties on the axes, figure, and root levels:

```
set(0,'DefaultLightProperty',PropertyValue...)
set(gcf,'DefaultLightProperty',PropertyValue...)
set(gca,'DefaultLightProperty',PropertyValue...)
```

Where *Property* is the name of the light property and `PropertyValue` is the value you are specifying. Use `set` and `get` to access light properties.

The following table lists all light properties and provides a brief description of each. The property name links take you to an expanded description of the properties.

Property Name	Property Description	Property Value
Defining the Light		
`Color`	Color of the light produced by the light object	Values: `ColorSpec`
`Position`	Location of light in the axes	Values: x-, y-, z-coordinates in axes units Default: `[1 0 1]`
`Style`	Parallel or divergent light source	Values: `infinite`, `local`
Controlling the Appearance		

Property Name	Property Description	Property Value
SelectionHighlight	This property is not used by light objects	Values: on, off Default: on
Visible	Make the effects of the light visible or invisible	Values: on, off Default: on
Controlling Access to Objects		
HandleVisibility	Determines if and when the the line's handle is visible to other functions	Values: on, callback, off Default: on
HitTest	This property is not used by light objects	Values: on, off Default: on
General Information About the Light		
Children	Light objects have no children	Values: [] (empty matrix)
Parent	The parent of a light object is always an axes object	Value: axes handle
Selected	This property is not used by light objects	Values: on, off Default: on
Tag	User-specified label	Value: any string Default: '' (empty string)
Type	The type of graphics object (read only)	Value: the string 'light'
UserData	User-specified data	Values: any matrix Default: [] (empty matrix)
Properties Related to Callback Routine Execution		
BusyAction	Specify how to handle callback routine interruption	Values: cancel, queue Default: queue
ButtonDownFcn	This property is not used by light objects	Values: string Default: empty string

Property Name	Property Description	Property Value
`CreateFcn`	Define a callback routine that executes when a light is created	Values: string (command or M-file name) Default: empty string
`DeleteFcn`	Define a callback routine that executes when the light is deleted (via `close` or `delete`)	Values: string (command or M-file name) Default: empty string
`Interruptible`	Determine if callback routine can be interrupted	Values: `on`, `off` Default: `on` (can be interrupted)
`UIContextMenu`	This property is not used by light objects	Values: handle of a Uicontrextmenu

Light Properties

This section lists property names along with the type of values each accepts.

BusyAction `cancel | {queue}`

Callback routine interruption. The `BusyAction` property enables you to control how MATLAB handles events that potentially interrupt executing callback routines. If there is a callback routine executing, subsequently invoked callback routes always attempt to interrupt it. If the `Interruptible` property of the object whose callback is executing is set to `on` (the default), then interruption occurs at the next point where the event queue is processed. If the `Interruptible` property is `off`, the `BusyAction` property (of the object owning the executing callback) determines how MATLAB handles the event. The choices are:

- `cancel` – discard the event that attempted to execute a second callback routine.
- `queue` – queue the event that attempted to execute a second callback routine until the current callback finishes.

ButtonDownFcn string

This property is not useful on lights.

Children handles

The empty matrix; light objects have no children.

Clipping `on | off`

`Clipping` has no effect on light objects.

Color `ColorSpec`

Color of light. This property defines the color of the light emanating from the light object. Define it as three-element RGB vector or one of MATLAB's predefined names. See the `ColorSpec` reference page for more information.

CreateFcn string

Callback routine executed during object creation. This property defines a callback routine that executes when MATLAB creates a light object. You must define this property as a default value for lights. For example, the statement,

```
set(0,'DefaultLightCreateFcn','set(gcf,''Colormap'',hsv)')
```

sets the current figure colormap to `hsv` whenever you create a light object. MATLAB executes this routine after setting all light properties. Setting this property on an existing light object has no effect.

The handle of the object whose `CreateFcn` is being executed is accessible only through the root `CallbackObject` property, which you can query using `gcbo`.

`DeleteFcn` string

Delete light callback routine. A callback routine that executes when you delete the light object (i.e., when you issue a `delete` command or clear the axes or figure containing the light). MATLAB executes the routine before destroying the object's properties so these values are available to the callback routine.

The handle of the object whose `DeleteFcn` is being executed is accessible only through the root `CallbackObject` property, which you can query using `gcbo`.

`HandleVisibility` `{on} | callback | off`

Control access to object's handle by command-line users and GUIs. This property determines when an object's handle is visible in its parent's list of children. `HandleVisibility` is useful for preventing command-line users from accidentally drawing into or deleting a figure that contains only user interface devices (such as a dialog box).

Handles are always visible when `HandleVisibility` is `on`.

Setting `HandleVisibility` to `callback` causes handles to be visible from within callback routines or functions invoked by callback routines, but not from within functions invoked from the command line. This provides a means to protect GUIs from command-line users, while allowing callback routines to have complete access to object handles.

Setting `HandleVisibility` to `off` makes handles invisible at all times. This may be necessary when a callback routine invokes a function that might potentially damage the GUI (such as evaling a user-typed string), and so temporarily hides its own handles during the execution of that function.

When a handle is not visible in its parent's list of children, it cannot be returned by functions that obtain handles by searching the object hierarchy or querying handle properties. This includes `get`, `findobj`, `gca`, `gcf`, `gco`, `newplot`, `cla`, `clf`, and `close`.

When a handle's visibility is restricted using `callback` or `off`, the object's handle does not appear in its parent's `Children` property, figures do not appear in the root's `CurrentFigure` property, objects do not appear in the root's `CallbackObject` property or in the figure's `CurrentObject` property, and axes do not appear in their parent's `CurrentAxes` property.

You can set the root `ShowHiddenHandles` property to on to make all handles visible, regardless of their `HandleVisibility` settings (this does not affect the values of the `HandleVisibility` properties).

Handles that are hidden are still valid. If you know an object's handle, you can `set` and `get` its properties, and pass it to any function that operates on handles.

`HitTest` `{on} | off`

This property is not used by light objects.

`Interruptible` `{on} | off`

Callback routine interruption mode. Light object callback routines defined for the `DeleteFcn` property are not affected by the `Interruptible` property.

`Parent` handle of parent axes

Light objects parent. The handle of the light object's parent axes. You can move a light object to another axes by changing this property to the new axes handle.

`Position` `[x,y,z]` in axes data units

Location of light object. This property specifies a vector defining the location of the light object. The vector is defined from the origin to the specified *x*, *y*, and *z* coordinates. The placement of the light depends on the setting of the `Style` property:

- If the `Style` property is set to `local`, `Position` specifies the actual location of the light (which is then a point source that radiates from the location in all directions).
- If the `Style` property is set to `infinite`, `Position` specifies the direction from which the light shines in parallel rays.

`Selected` `on | off`

This property is not used by light objects.

SelectionHighlight {on} | off

This property is not used by light objects.

Style {infinite} | local

Parallel or divergent light source. This property determines whether MATLAB places the light object at infinity, in which case the light rays are parallel, or at the location specified by the Position property, in which case the light rays diverge in all directions. See the Position property.

Tag string

User-specified object label. The Tag property provides a means to identify graphics objects with a user-specified label. This is particularly useful when constructing interactive graphics programs that would otherwise need to define object handles as global variables or pass them as arguments between callback routines. You can define Tag as any string.

Type string (read only)

Type of graphics object. This property contains a string that identifies the class of graphics object. For light objects, Type is always 'light'.

UIContextMenu handle of a uicontextmenu object

This property is not used by light objects.

UserData matrix

User specified data. This property can be any data you want to associate with the light object. The light does not use this property, but you can access it using set and get.

Visible {on} | off

Light visibility. While light objects themselves are not visible, you can see the light on patch and surface objects. When you set Visible to off, the light emanating from the source is not visible. There must be at least one light object in the axes whose Visible property is on for any lighting features to be enabled (including the axes AmbientLightColor and patch and surface AmbientStrength).

See Also

lighting, material, patch, surface

Purpose

Create or position a light object in spherical coordinates

Syntax

```
lightangle(az,el)
light_handle = lightangle(az,el)
lightangle(light_handle,az,el)
[ax el] = lightangle(light_handle)
```

Description

`lightangle(az,el)` creates a light at the position specified by azimuth and elevation. `az` is the azimuthal (horizontal) rotation and `el` is the vertical elevation (both in degrees). The interpretation of azimuth and elevation is the same as that of the `view` command.

`light_handle = lightangle(az,el)` creates a light and returns the handle of the light in `light_handle`.

`lightangle(light_handle,az,el)` sets the position of the light specified by `light_handle`.

`[az,el] = lightangle(light_handle)` returns the azimuth and elevation of the light specified by `light_handle`.

Remarks

By default, when a light is created, its style is `infinite`. If the light handle passed into `lightangle` refers to a local light, the distance between the light and the camera target is preserved as the position is changed.

Examples

```
surf(peaks)
axis vis3d
h = light;
for az = –50:10:50
   lightangle(h,az,30)
   drawnow
end
```

See Also

`light`, `camlight`, `view`

lighting

Purpose Select the lighting algorithm

Syntax

```
lighting flat
lighting gouraud
lighting phong
lighting none
```

Description `lighting` selects the algorithm used to calculate the effects of light objects on all surface and patch objects in the current axes.

`lighting flat` selects flat lighting.

`lighting gouraund` selects gouraud lighting.

`lighting phong` selects phong lighting.

`lighting none` turns off lighting.

Remarks The `surf`, `mesh`, `pcolor`, `fill`, `fill3`, `surface`, and `patch` functions create graphics objects that are affected by light sources. The `lighting` command sets the `FaceLighting` and `EdgeLighting` properties of surfaces and patches appropriately for the graphics object.

See Also `light`, `material`, `patch`, `surface`

Purpose

Create line object

Syntax

```
line(X,Y)
line(X,Y,Z)
line(X,Y,Z,'PropertyName',PropertyValue,...)
line('PropertyName',PropertyValue,...)  low-level–PN/PV pairs only
h = line(...)
```

Description

`line` creates a line object in the current axes. You can specify the color, width, line style, and marker type, as well as other characteristics.

The `line` function has two forms:

- Automatic color and line style cycling. When you specify matrix coordinate data using the informal syntax (i.e., the first three arguments are interpreted as the coordinates),

  ```
  line(X,Y,Z)
  ```

  MATLAB cycles through the axes `ColorOrder` and `LineStyleOrder` property values the way the `plot` function does. However, unlike `plot`, `line` does not call the `newplot` function.
- Purely low-level behavior. When you call `line` with only property name/property value pairs,

  ```
  line('XData',x,'YData',y,'ZData',z)
  ```

  MATLAB draws a line object in the current axes using the default line color (see the `colordef` function for information on color defaults). Note that you cannot specify matrix coordinate data with the low-level form of the `line` function.

`line(X,Y)` adds the line defined in vectors `X` and `Y` to the current axes. If `X` and `Y` are matrices of the same size, `line` draws one line per column.

`line(X,Y,Z)` creates lines in three-dimensional coordinates.

`line(X,Y,Z,'`*`PropertyName`*`',PropertyValue,...)` creates a line using the values for the property name/property value pairs specified and default values for all other properties.

See the `LineStyle` and `Marker` properties for a list of supported values.

`line('XData',x,'YData',y,'ZData',z,'`*`PropertyName`*`',PropertyValue,...)` creates a line in the current axes using the property values defined as arguments. This is the low-level form of the `line` function, which does not accept matrix coordinate data as the other informal forms described above.

`h = line(...)` returns a column vector of handles corresponding to each line object the function creates.

Remarks

In its informal form, the `line` function interprets the first three arguments (two for 2-D) as the `X`, `Y`, and `Z` coordinate data, allowing you to omit the property names. You must specify all other properties as name/value pairs. For example,

```
line(X,Y,Z,'Color','r','LineWidth',4)
```

The low-level form of the `line` function can have arguments that are only property name/property value paris. For example,

```
line('XData',x,'YData',y,'ZData',z,'Color','r','LineWidth',4)
```

Line properties control various aspects of the line object and are described in the "Line Properties" section. You can also set and query property values after creating the line using `set` and `get`.

You can specify properties as property name/property value pairs, structure arrays, and cell arrays (see the `set` and `get` reference pages for examples of how to specify these data types).

Unlike high-level functions such as `plot`, `line` does not respect the setting of the figure and axes `NextPlot` properties. It simply adds line objects to the current axes. However, axes properties that are under automatic control such as the axis limits can change to accommodate the line within the current axes.

Examples

This example uses the `line` function to add a shadow to plotted data. First, plot some data and save the line's handle:

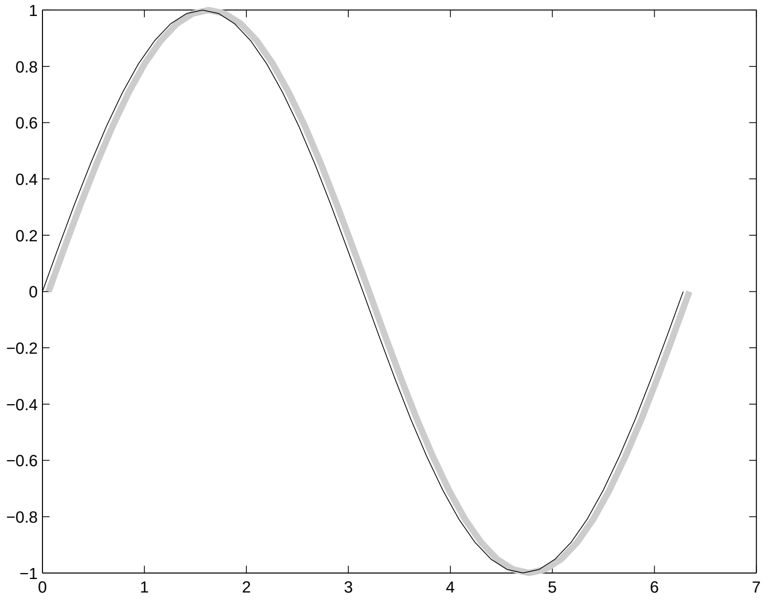
```
t = 0:pi/20:2*pi;
hline1 = plot(t,sin(t),’k’);
```

Next, add a shadow by offsetting the *x* coordinates. Make the shadow line light gray and wider than the default `LineWidth`:

```
hline2 = line(t+.06,sin(t),'LineWidth',4,'Color',[.8 .8 .8]);
```

Finally, pop the first line to the front:

```
set(gca,'Children',[hline1 hline2])
```



Input Argument Dimensions – Informal Form

This statement reuses the one column matrix specified for `ZData` to produce two lines, each having four points.

```
line(rand(4,2),rand(4,2),rand(4,1))
```

If all the data has the same number of columns and one row each, MATLAB transposes the matrices to produce data for plotting. For example,

```
line(rand(1,4),rand(1,4),rand(1,4))
```

is changed to:

```
line(rand(4,1),rand(4,1),rand(4,1))
```

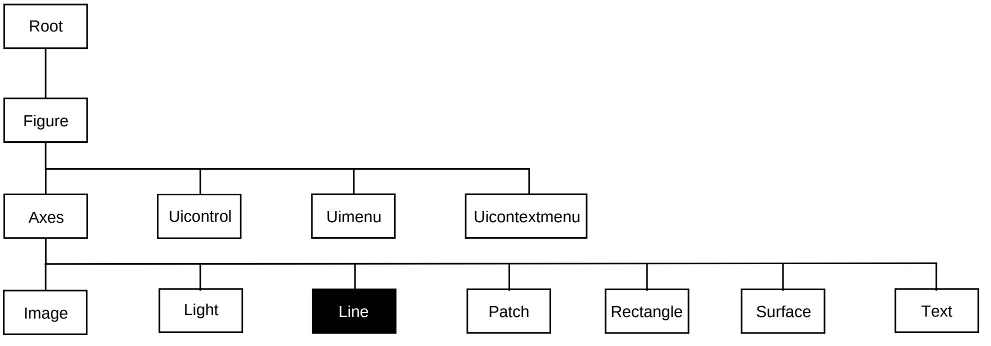
This also applies to the case when just one or two matrices have one row. For example, the statement,

```
line(rand(2,4),rand(2,4),rand(1,4))
```

is equivalent to:

```
line(rand(4,2),rand(4,2),rand(4,1))
```

Object Hierarchy



Setting Default Properties

You can set default line properties on the axes, figure, and root levels.

```
set(0,'DefaultLinePropertyName',PropertyValue,...)
set(gcf,'DefaultLinePropertyName',PropertyValue,...)
set(gca,'DefaultLinePropertyName',PropertyValue,...)
```

Where *PropertyName* is the name of the line property and `PropertyValue` is the value you are specifying. Use `set` and `get` to access line properties.

The following table lists all line properties and provides a brief description of each. The property name links take you to an expanded description of the properties.

Property Name	Property Description	Property Value
Data Defining the Object		
XData	The *x*-coordinates defining the line	Values: vector or matrix Default: [0 1]
YData	The *y*-coordinates defining the line	Values: vector or matrix Default: [0 1]
ZData	The *z*-coordinates defining the line	Values: vector or matrix Default: [] empty matrix
Defining Line Styles and Markers		
LineStyle	Select from five line styles.	Values: –, ––, :, –., none Default: –
LineWidth	The width of the line in points	Values: scalar Default: 0.5 points
Marker	Marker symbol to plot at data points	Values: see Marker property Default: none
MarkerEdgeColor	Color of marker or the edge color for filled markers	Values: ColorSpec, none, auto Default: auto
MarkerFaceColor	Fill color for markers that are closed shapes	Values: ColorSpec, none, auto Default: none
MarkerSize	Size of marker in points	Values: size in points Default: 6
Controlling the Appearance		
Clipping	Clipping to axes rectangle	Values: on, off Default: on
EraseMode	Method of drawing and erasing the line (useful for animation)	Values: normal, none, xor, background Default: normal

Property Name	Property Description	Property Value
SelectionHighlight	Highlight line when selected (Selected property set to on)	Values: on, off Default: on
Visible	Make the line visible or invisible	Values: on, off Default: on
Color	Color of the line	ColorSpec
Controlling Access to Objects		
HandleVisibility	Determines if and when the the line's handle is visible to other functions	Values: on, callback, off Default: on
HitTest	Determines if the line can become the current object (see the figure CurrentObject property)	Values: on, off Default: on
General Information About the Line		
Children	Line objects have no children	Values: [] (empty matrix)
Parent	The parent of a line object is always an axes object	Value: axes handle
Selected	Indicate whether the line is in a "selected" state.	Values: on, off Default: on
Tag	User-specified label	Value: any string Default: '' (empty string)
Type	The type of graphics object (read only)	Value: the string 'line'
UserData	User-specified data	Values: any matrix Default: [] (empty matrix)
Properties Related to Callback Routine Execution		
BusyAction	Specify how to handle callback routine interruption	Values: cancel, queue Default: queue

Property Name	Property Description	Property Value
ButtonDownFcn	Define a callback routine that executes when a mouse button is pressed on over the line	Values: string Default: '' (empty string)
CreateFcn	Define a callback routine that executes when a line is created	Values: string Default: '' (empty string)
DeleteFcn	Define a callback routine that executes when the line is deleted (via close or delete)	Values: string Default: '' (empty string)
Interruptible	Determine if callback routine can be interrupted	Values: on, off Default: on (can be interrupted)
UIContextMenu	Associate a context menu with the line	Values: handle of a Uicontrextmenu

Line Properties

This section lists property names along with the type of values each accepts. Curly braces { } enclose default values.

BusyAction `cancel | {queue}`

Callback routine interruption. The `BusyAction` property enables you to control how MATLAB handles events that potentially interrupt executing callback routines. If there is a callback routine executing, subsequently invoked callback routes always attempt to interrupt it. If the `Interruptible` property of the object whose callback is executing is set to `on` (the default), then interruption occurs at the next point where the event queue is processed. If the `Interruptible` property is `off`, the `BusyAction` property (of the object owning the executing callback) determines how MATLAB handles the event. The choices are:

- `cancel` – discard the event that attempted to execute a second callback routine.
- `queue` – queue the event that attempted to execute a second callback routine until the current callback finishes.

ButtonDownFcn string

Button press callback routine. A callback routine that executes whenever you press a mouse button while the pointer is over the line object. Define this routine as a string that is a valid MATLAB expression or the name of an M-file. The expression executes in the MATLAB workspace.

Children vector of handles

The empty matrix; line objects have no children.

Clipping `{on} | off`

Clipping mode. MATLAB clips lines to the axes plot box by default. If you set `Clipping` to `off`, lines display outside the axes plot box. This can occur if you create a line, set `hold` to `on`, freeze axis scaling (`axis manual`), and then create a longer line.

Color `ColorSpec`

Line color. A three-element RGB vector or one of MATLAB's predefined names, specifying the line color. See the `ColorSpec` reference page for more information on specifying color.

CreateFcn string

Callback routine executed during object creation. This property defines a callback routine that executes when MATLAB creates a line object. You must define this property as a default value for lines. For example, the statement,

```
set(0,'DefaultLineCreateFcn','set(gca,''LineStyleOrder'',''-.|--'')')
```

defines a default value on the root level that sets the axes `LineStyleOrder` whenever you create a line object. MATLAB executes this routine after setting all line properties. Setting this property on an existing line object has no effect.

The handle of the object whose `CreateFcn` is being executed is accessible only through the root `CallbackObject` property, which you can query using `gcbo`.

DeleteFcn string

Delete line callback routine. A callback routine that executes when you delete the line object (e.g., when you issue a `delete` command or clear the axes or figure). MATLAB executes the routine before deleting the object's properties so these values are available to the callback routine.

The handle of the object whose `DeleteFcn` is being executed is accessible only through the root `CallbackObject` property, which you can query using `gcbo`.

EraseMode `{normal} | none | xor | background`

Erase mode. This property controls the technique MATLAB uses to draw and erase line objects. Alternative erase modes are useful for creating animated sequences, where control of the way individual objects redraw is necessary to improve performance and obtain the desired effect.

- `normal` (the default) — Redraw the affected region of the display, performing the three-dimensional analysis necessary to ensure that all objects are rendered correctly. This mode produces the most accurate picture, but is the slowest. The other modes are faster, but do not perform a complete redraw and are therefore less accurate.
- `none` – Do not erase the line when it is moved or destroyed. While the object is still visible on the screen after erasing with `EraseMode none`, you cannot print it because MATLAB stores no information about its former location.
- `xor` – Draw and erase the line by performing an exclusive OR (XOR) with the color of the screen beneath it. This mode does not damage the color of the

objects beneath the line. However, the line's color depends on the color of whatever is beneath it on the display.

- background – Erase the line by drawing it in the axes' background Color, or the figure background Color if the axes Color is set to none. This damages objects that are behind the erased line, but lines are always properly colored.

Printing with Non-normal Erase Modes

MATLAB always prints figures as if the EraseMode of all objects is normal. This means graphics objects created with EraseMode set to none, xor, or background can look different on screen than on paper. On screen, MATLAB may mathematically combine layers of colors (e.g., XORing a pixel color with that of the pixel behind it) and ignore three-dimensional sorting to obtain greater rendering speed. However, these techniques are not applied to the printed output.

You can use the MATLAB getframe command or other screen capture application to create an image of a figure containing non-normal mode objects.

HitTest {on} | off

Selectable by mouse click. HitTest determines if the line can become the current object (as returned by the gco command and the figure CurrentObject property) as a result of a mouse click on the line. If HiTest is off, clicking on the line selects the object below it (which may be the axes containing it).

HandleVisibility {on} | callback | off

Control access to object's handle by command-line users and GUIs. This property determines when an object's handle is visible in its parent's list of children. HandleVisibility is useful for preventing command-line users from accidentally drawing into or deleting a figure that contains only user interface devices (such as a dialog box).

Handles are always visible when HandleVisibility is on.

Setting HandleVisibility to callback causes handles to be visible from within callback routines or functions invoked by callback routines, but not from within functions invoked from the command line. This provides a means to protect GUIs from command-line users, while allowing callback routines to have complete access to object handles.

Setting `HandleVisibility` to `off` makes handles invisible at all times. This may be necessary when a callback routine invokes a function that might potentially damage the GUI (such as evaling a user-typed string), and so temporarily hides its own handles during the execution of that function.

When a handle is not visible in its parent's list of children, it cannot be returned by functions that obtain handles by searching the object hierarchy or querying handle propertes. This includes `get`, `findobj`, `gca`, `gcf`, `gco`, `newplot`, `cla`, `clf`, and `close`.

When a handle's visibility is restricted using `callback` or `off`, the object's handle does not appear in its parent's `Children` property, figures do not appear in the root's `CurrentFigure` property, objects do not appear in the root's `CallbackObject` property or in the figure's `CurrentObject` property, and axes do not appear in their parent's `CurrentAxes` property.

You can set the root `ShowHiddenHandles` property to `on` to make all handles visible, regardless of their `HandleVisibility` settings (this does not affect the values of the `HandleVisibility` properties).

Handles that are hidden are still valid. If you know an object's handle, you can `set` and `get` its properties, and pass it to any function that operates on handles.

`Interruptible` `{on} | off`

Callback routine interruption mode. The `Interruptible` property controls whether a line callback routine can be interrupted by subsequently invoked callback routines. Only callback routines defined for the `ButtonDownFcn` are affected by the `Interruptible` property. MATLAB checks for events that can interrupt a callback routine only when it encounters a `drawnow`, `figure`, `getframe`, or `pause` command in the routine.

`LineStyle` `{–} | –– | : | –. | none`

Line style. This property specifies the line style. Available line styles are shown in the table.

Symbol	Line Style
–	solid line (default)
––	dashed line

Symbol	Line Style
:	dotted line
-.	dash-dot line
none	no line

You can use `LineStyle` none when you want to place a marker at each point but do not want the points connected with a line (see the `Marker` property).

`LineWidth` scalar

The width of the line object. Specify this value in points (1 point = $^1/_{72}$ inch). The default `LineWidth` is 0.5 points.

`Marker` character (see table)

Marker symbol. The Marker property specifies marks that display at data points. You can set values for the `Marker` property independently from the `LineStyle` property. Supported markers include those shown in the table.

Marker Specifier	Description
+	plus sign
o	circle
*	asterisk
.	point
x	cross
s	square
d	diamond
^	upward pointing triangle
v	downward pointing triangle
>	right pointing triangle
<	left pointing triangle

Marker Specifier	Description
p	five-pointed star (pentagram)
h	six-pointed star (hexagram)
none	no marker (default)

MarkerEdgeColor ColorSpec | none | {auto}

Marker edge color. The color of the marker or the edge color for filled markers (circle, square, diamond, pentagram, hexagram, and the four triangles). ColorSpec defines the color to use. none specifies no color, which makes nonfilled markers invisible. auto sets MarkerEdgeColor to the same color as the line's Color property.

MarkerFaceColor ColorSpec | {none} | auto

Marker face color. The fill color for markers that are closed shapes (circle, square, diamond, pentagram, hexagram, and the four triangles). ColorSpec defines the color to use. none makes the interior of the marker transparent, allowing the background to show through. auto sets the fill color to the axes color, or the figure color, if the axes Color property is set to none (which is the factory default for axes).

MarkerSize size in points

Marker size. A scalar specifying the size of the marker, in points. The default value for MarkerSize is six points (1 point = 1/72 inch). Note that MATLAB draws the point marker (specified by the '.' symbol) at one-third the specified size.

Parent handle

Line's parent. The handle of the line object's parent axes. You can move a line object to another axes by changing this property to the new axes handle.

Selected on | off

Is object selected. When this property is on. MATLAB displays selection handles if the SelectionHighlight property is also on. You can, for example, define the ButtonDownFcn to set this property, allowing users to select the object with the mouse.

SelectionHighlight {on} | off

Objects highlight when selected. When the Selected property is on, MATLAB indicates the selected state by drawing handles at each vertex. When SelectionHighlight is off, MATLAB does not draw the handles.

Tag string

User-specified object label. The Tag property provides a means to identify graphics objects with a user-specified label. This is particularly useful when constructing interactive graphics programs that would otherwise need to define object handles as global variables or pass them as arguments between callback routines. You can define Tag as any string.

Type string (read only)

Class of graphics object. For line objects, Type is always the string 'line'.

UIContextMenu handle of a uicontextmenu object

Associate a context menu with the line. Assign this property the handle of a uicontextmenu object created in same figure as the line. Use the uicontextmenu function to create the context menu. MATLAB displays the context menu whenever you right-click over the line.

UserData matrix

User-specified data. Any data you want to associate with the line object. MATLAB does not use this data, but you can access it using the set and get commands.

Visible {on} | off

Line visibility. By default, all lines are visible. When set to off, the line is not visible, but still exists and you can get and set its properties.

XData vector of coordinates

X-coordinates. A vector of *x*-coordinates defining the line. YData and ZData must have the same number of rows. (See "Examples").

YData vector or matrix of coordinates

Y-coordinates. A vector of *y*-coordinates defining the line. XData and ZData must have the same number of rows.

ZData vector of coordinates

Z-coordinates. A vector of *z*-coordinates defining the line. XData and YData must have the same number of rows.

See Also

axes,newplot, plot, plot3

Purpose

Line specification syntax

Description

This page describes how to specify the properties of lines used for plotting. MATLAB enables you to define many characteristics including:

- Line style
- Line width
- Color
- Marker type
- Marker size
- Marker face and edge coloring (for filled markers)

MATLAB defines string specifiers for line styles, marker types, and colors. The following tables list these specifiers.

Line Style Specifiers

Specifier	Line Style
-	solid line (default)
--	dashed line
:	dotted line
-.	dash-dot line

Marker Specifiers

Specifier	Marker Type
+	plus sign
o	circle
*	asterisk
.	point
x	cross
s	square
d	diamond
^	upward pointing triangle
v	downward pointing triangle
>	right pointing triangle
<	left pointing triangle
p	five-pointed star (pentagram)
h	six-pointed star (hexagram)

Color Specifiers

Specifier	Color
`r`	red
`g`	green
`b`	blue
`c`	cyan
`m`	magenta
`y`	yellow
`k`	black
`w`	white

Many plotting commands accept a `LineSpec` argument that defines three components used to specify lines:

- Line style
- Marker symbol
- Color

For example,

```
plot(x,y,'-.or')
```

plots `y` versus `x` using a dash-dot line (`-.`), places circular markers (`o`) at the data points, and colors both line and marker red (`r`). Specify the components (in any order) as a quoted string after the data arguments.

If you specify a marker, but not a line style, MATLAB plots only the markers. For example,

```
plot(x,y,'d')
```

Related Properties

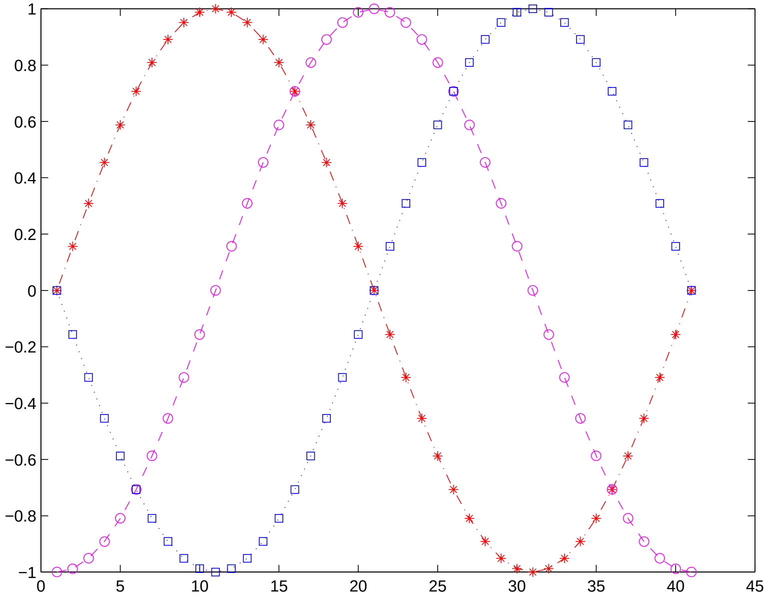
When using the `plot` and `plot3` functions, you can also specify other characteristics of lines using graphics properties:

- `LineWidth` – specifies the width (in points) of the line
- `MarkerEdgeColor` – specifies the color of the marker or the edge color forfilled markers (circle, square, diamond, pentagram, hexagram, and the four triangles).
- `MarkerFaceColor` – specifies the color of the face of filled markers.
- `MarkerSize` – specifies the size of the marker in points.

In addition, you can specify the `LineStyle`, `Color`, and `Marker` properties instead of using the symbol string. This is useful if you want to specify a color that is not in the list by using RGB values. See `ColorSpec` for more information on color.

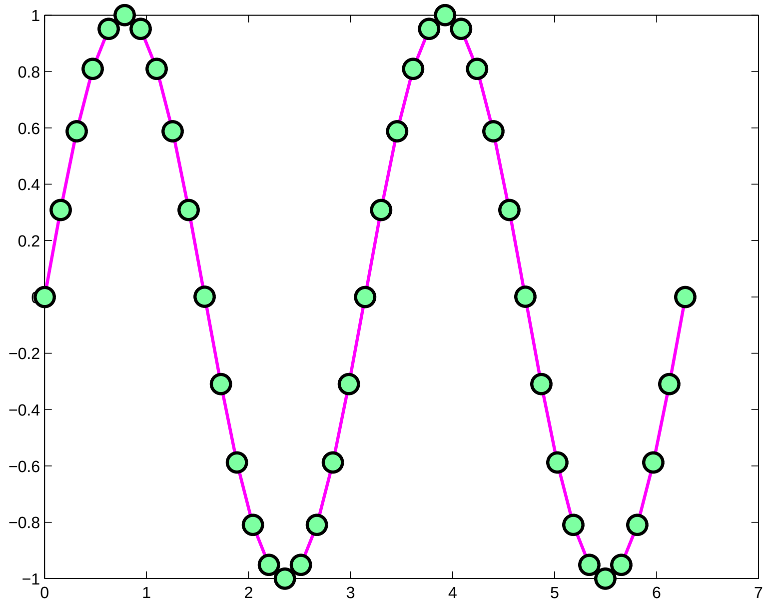
Examples Plot the sine function over three different ranges using different line styles, colors, and markers.

```
t = 0:pi/20:2*pi;
plot(t,sin(t),'-.r*')
hold on
plot(sin(t-pi/2),'--mo')
plot(sin(t-pi),':bs')
hold off
```



Create a plot illustrating how to set line properties.

```
plot(t,sin(2*t),'-mo',...
                'LineWidth',2,...
                'MarkerEdgeColor','k',...
                'MarkerFaceColor',[.49 1 .63],...
                'MarkerSize',12)
```



See Also `line`, `plot`, `patch`, `set`, `surface`, axes `LineStyleOrder` property

listdlg

Purpose Create list selection dialog box

Syntax

```
[Selection,ok] = listdlg('ListString',S,...)
```

Description

`[Selection,ok] = listdlg('ListString',S)` creates a modal dialog box that enables you to select one or more items from a list. `Selection` is a vector of indices of the selected strings (in single selection mode, its length is 1). `Selection` is [] when `ok` is 0. `ok` is 1 if you click the **OK** button, or 0 if you click the **Cancel** button or close the dialog box. Double-clicking on an item or pressing **Return** when multiple items are selected has the same effect as clicking the **OK** button. The dialog box has a **Select all** button (when in multiple selection mode) that enables you to select all list items.

Inputs are in parameter/value pairs:

Parameter	Description
`'ListString'`	Cell array of strings that specify the list box items.
`'SelectionMode'`	String indicating whether one or many items can be selected:`'single'` or `'multiple'` (the default).
`'ListSize'`	List box size in pixels, specified as a two element vector, `[width height]`. Default is `[160 300]`.
`'InitialValue'`	Vector of indices of the list box items that are initially selected. Default is 1, the first item.
`'Name'`	String for the dialog box's title. Default is ''.
`'PromptString'`	String matrix or cell array of strings that appears as text above the list box. Default is {}.
`'OKString'`	String for the OK button. Default is `'OK'`.
`'CancelString'`	String for the Cancel button. Default is '`Cancel`'.
`'uh'`	Uicontrol button height, in pixels. Default is 18.
`'fus'`	Frame/uicontrol spacing, in pixels. Default is 8.
`'ffs'`	Frame/figure spacing, in pixels. Default is 8.

Example

This example displays a dialog box that enables the user to select a file from the current directory. The function returns a vector. Its first element is the index to the selected file; its second element is 0 if no selection is made, or 1 if a selection is made.

```
d = dir;
str = {d.name};
[s,v] = listdlg('PromptString','Select a file:',...
                'SelectionMode','single',...
                'ListString',str)
```

See Also

`dir`

loglog

Purpose Log-log scale plot

Syntax

```
loglog(Y)
loglog(X1,Y1,...)
loglog(X1,Y1,LineSpec,...)
loglog(...,'PropertyName',PropertyValue,...)
h = loglog(...)
```

Description

`loglog(Y)` plots the columns of `Y` versus their index if `Y` contains real numbers. If `Y` contains complex numbers, `loglog(Y)` and `loglog(real(Y),imag(Y))` are equivalent. `loglog` ignores the imaginary component in all other uses of this function.

`loglog(X1,Y1,...)` plots all `Xn` versus `Yn` pairs. If only `Xn` or `Yn` is a matrix, `loglog` plots the vector argument versus the rows or columns of the matrix, depending on whether the vector's row or column dimension matches the matrix.

`loglog(X1,Y1,LineSpec,...)` plots all lines defined by the `Xn,Yn,LineSpec` triples, where `LineSpec` determines line type, marker symbol, and color of the plotted lines. You can mix `Xn,Yn,LineSpec` triples with `Xn,Yn` pairs, for example,

```
loglog(X1,Y1,X2,Y2,LineSpec,X3,Y3)
```

`loglog(...,'`*`PropertyName`*`',PropertyValue,...)` sets property values for all line graphics objects created by `loglog`. See the `line` reference page for more information.

`h = loglog(...)` returns a column vector of handles to line graphics objects, one handle per line.

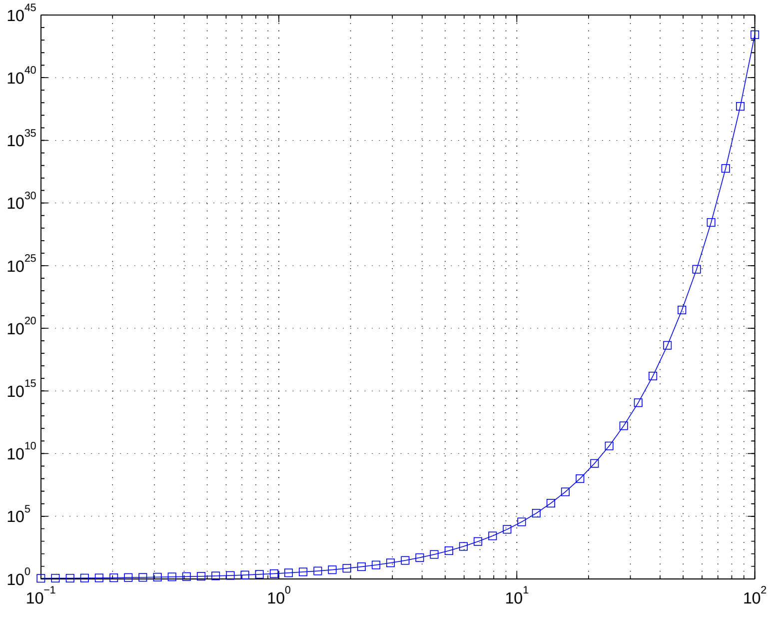
Remarks

If you do not specify a color when plotting more than one line, `loglog` automatically cycles through the colors and line styles in the order specified by the current axes.

Examples

Create a simple loglog plot with square markers.

```
x = logspace(−1,2);
loglog(x,exp(x),'−s')
grid on
```



See Also

line, LineSpec, plot, semilogx, semilogy

material

Purpose Controls the reflectance properties of surfaces and patches

Syntax

```
material shiny
material dull
material metal
material([ka kd ks])
material([ka kd ks n])
material([ka kd ks n sc])
material default
```

Description `material` sets the lighting characteristics of surface and patch objects.

`material shiny` sets the reflectance properties so that the object has a high specular reflectance relative the diffuse and ambient light and the color of the specular light depends only on the color of the light source.

`material dull` sets the reflectance properties so that the object reflects more diffuse light, has no specular highlights, but the color of the reflected light depends only on the light source.

`material metal` sets the reflectance properties so that the object has a very high specular reflectance, very low ambient and diffuse reflectance, and the color of the reflected light depends on both the color of the light source and the color of the object.

`material([ka kd ks])` sets the ambient/diffuse/specular strength of the objects.

`material([ka kd ks n])` sets the ambient/diffuse/specular strength and specular exponent of the objects.

`material([ka kd ks n sc])` sets the ambient/diffuse/specular strength, specular exponent, and specular color reflectance of the objects.

`material default` sets the ambient/diffuse/specular strength, specular exponent, and specular color reflectance of the objects to their defaults.

Remarks The `material` command sets the `AmbientStrength`, `DiffuseStrength`, `SpecularStrength`, `SpecularExponent`, and `SpecularColorReflectance` properties of all surface and patch objects in the axes. There must be visible light objects in the axes for lighting to be enabled. Look at the `materal.m` M-file to see the actual values set (enter the command: `type material`).

See Also `light`, `lighting`, `patch`, `surface`

mesh, meshc, meshz

Purpose Mesh plots

Syntax

```
mesh(X,Y,Z)
mesh(Z)
mesh(...,C)
meshc(...)
meshz(...)
h = mesh(...)
h = meshc(...)
h = meshz(...)
```

Description `mesh`, `meshc`, and `meshz` create wireframe parametric surfaces specified by `X`, `Y`, and `Z`, with color specified by `C`.

`mesh(X,Y,Z)` draws a wireframe mesh with color determined by `Z`, so color is proportional to surface height. If `X` and `Y` are vectors, `length(X) = n` and `length(Y) = m`, where `[m,n] = size(Z)`. In this case, $(X(j), Y(i), Z(i, j))$ are the intersections of the wireframe grid lines; `X` and `Y` correspond to the columns and rows of `Z`, respectively. If `X` and `Y` are matrices, $(X(i, j), Y(i, j), Z(i, j))$ are the intersections of the wireframe grid lines.

`mesh(Z)` draws a wireframe mesh using `X = 1:n` and `Y = 1:m`, where `[m,n] = size(Z)`. The height, `Z`, is a single-valued function defined over a rectangular grid. Color is proportional to surface height.

`mesh(...,C)` draws a wireframe mesh with color determined by matrix `C`. MATLAB performs a linear transformation on the data in `C` to obtain colors from the current colormap. If `X`, `Y`, and `Z` are matrices, they must be the same size as `C`.

`meshc(...)` draws a contour plot beneath the mesh.

`meshz(...)` draws a curtain plot (i.e., a reference plane) around the mesh.

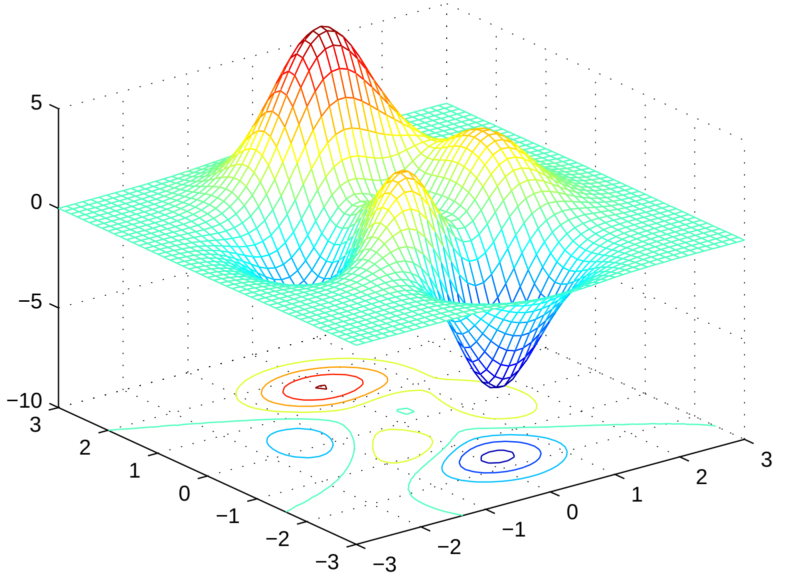
`h = mesh(...)`, `h = meshc(...)`, and `h = meshz(...)` return a handle to a surface graphics object.

Remarks A mesh is drawn as a surface graphics object with the viewpoint specified by `view(3)`. The face color is the same as the background color (to simulate a wireframe with hidden-surface elimination), or none when drawing a standard see-through wireframe. The current colormap determines the edge color. The `hidden` command controls the simulation of hidden-surface elimination in the mesh, and the `shading` command controls the shading model.

Examples

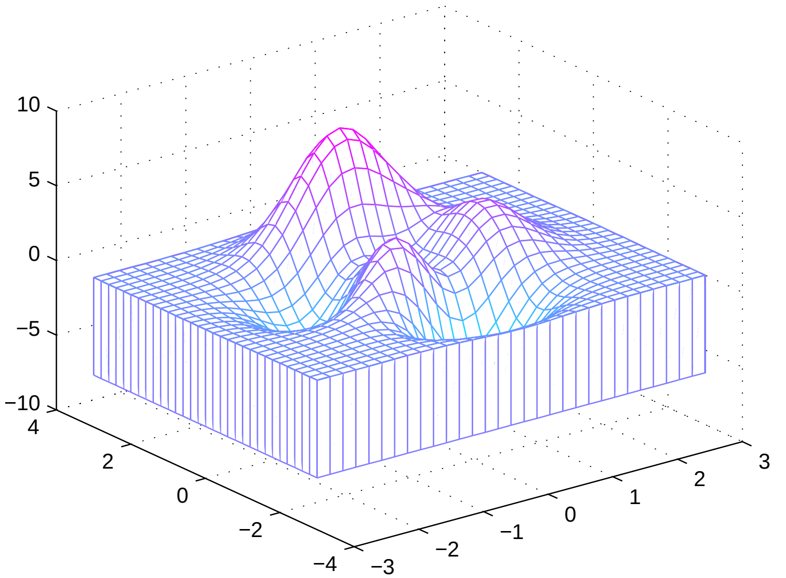
Produce a combination mesh and contour plot of the `peaks` surface:

```
[X,Y] = meshgrid(-3:.125:3);
Z = peaks(X,Y);
meshc(X,Y,Z);
axis([-3 3 -3 3 -10 5])
```



Generate the curtain plot for the peaks function:

```
[X,Y] = meshgrid(-3:.125:3);
Z = peaks(X,Y);
meshz(X,Y,Z)
```



Algorithm

The range of X, Y, and Z, or the current setting of the axes XLimMode, YLimMode, and ZLimMode properties determine the axis limits. axis sets these properties.

The range of C, or the current setting of the axes CLim and CLimMode properties (also set by the caxis function), determine the color scaling. The scaled color values are used as indices into the current colormap.

The mesh rendering functions produce color values by mapping the *z* data values (or an explicit color array) onto the current colormap. MATLAB's default behavior computes the color limits automatically using the minimum and maximum data values (also set using caxis auto). The minimum data value maps to the first color value in the colormap and the maximum data value maps to the last color value in the colormap. MATLAB performs a linear transformation on the intermediate values to map them to the current colormap.

meshc calls mesh, turns hold on, and then calls contour and positions the contour on the *x*-*y* plane. For additional control over the appearance of the contours, you can issue these commands directly. You can combine other types of graphs in this manner, for example surf and pcolor plots.

meshc assumes that X and Y are monotonically increasing. If X or Y is irregularly spaced, contour3 calculates contours using a regularly spaced contour grid, then transforms the data to X or Y.

See Also

contour, hidden, meshgrid, surf, surfc, surfl, waterfall

The functions axis, caxis, colormap, hold, shading, and view all set graphics object properties that affect mesh, meshc, and meshz.

For a discussion of parametric surfaces plots, refer to surf.

movie

Purpose Play recorded movie frames

Syntax

```
movie(M)
movie(M,n)
movie(M,n,fps)
movie(h,...)
movie(h,M,n,fps,loc)
```

Description

`movie` plays the movie defined by a matrix whose columns are movie frames (usually produced by `getframe`).

`movie(M)` plays the movie in matrix `M` once.

`movie(M,n)` plays the movie `n` times. If `n` is negative, each cycle is shown forward then backward. If n is a vector, the first element is the number of times to play the movie, and the remaining elements comprise a list of frames to play in the movie. For example, if `M` has four frames then `n = [10 4 4 2 1]` plays the movie ten times, and the movie consists of frame `4` followed by frame `4` again, followed by frame `2` and finally frame `1`.

`movie(M,n,fps)` plays the movie at `fps` frames per second. The default is 12 frames per second. Computers that cannot achieve the specified speed play as fast as possible.

`movie(h,...)` plays the movie in the figure or axes identified by the handle `h`.

`movie(h,M,n,fps,loc)` specifies a four-element location vector, [x y 0 0], where the lower-left corner of the movie frame is anchored (only the first two elements in the vector are used). The location is relative to the lower-left corner of the figure or axes specified by handle and in units of pixels, regardless of the object's `Units` property.

Remarks

The movie function displays each frame as it loads the data into memory, and then plays the movie. This eliminates long delays with a blank screen when you load a memory-intensive movie. The movie's load cycle is not considered one of the movie repetitions.

Examples Animate the peaks function as you scale the values of Z:

```
Z = peaks; surf(Z);
axis tight
set(gca,'nextplot','replacechildren');

% Record the movie
for j = 1:20
   surf(sin(2*pi*j/20)*Z,Z)
   F(j) = getframe;
end

% Play the movie twenty times
movie(F,20)
```

See Also getframe, frame1im, im2frame

moviein

Purpose

Allocate matrix for movie frames

Syntax

```
M = moviein(n)
M = moviein(n,h)
M = moviein(n,h,rect)
```

Description

`moviein` allocates an appropriately sized matrix for the `getframe` function.

`M = moviein(n)` creates matrix `M` having `n` columns to store `n` frames of a movie based on the size of the current axes.

`M = moviein(n,h)` specifies a handle for a valid figure or axes graphics object on which to base the memory requirement. You must use the same handle with `getframe`. If you want to capture the axis in the frames, specify `h` as the handle of the figure.

`M = moviein(n,h,rect)` specifies the rectangular area from which to copy the bitmap, relative to the lower-left corner of the figure or axes graphics object identified by `h`. `rect = [left bottom width height]`, where `left` and `bottom` specify the lower-left corner of the rectangle, and width and height specify the dimensions of the rectangle. Components of `rect` are in pixel units. You must use the same handle and rectangle with `getframe`.

Remarks

`moviein` is no longer meeded as of MATLAB Release 11 (5.3). In earlier versions, pre-allocating a movie increased performance, but there is no longer a need to do this.

See Also

`getframe`, `movie`

Purpose

Display message box

Syntax

```
msgbox(message)
msgbox(message,title)
msgbox(message,title,'icon')
msgbox(message,title,'custom',iconData,iconCmap)
msgbox(...,'createMode')
h = msgbox(...)
```

Description

`msgbox(message)` creates a message box that automatically wraps `message` to fit an appropriately sized figure. `message` is a string vector, string matrix, or cell array.

`msgbox(message,title)` specifies the title of the message box.

`msgbox(message,title,'icon')` specifies which icon to display in the message box. `'icon'` is `'none'`, `'error'`, `'help'`, `'warn'`, or `'custom'`. The default is `'none'`.


Error Icon


Help Icon


Warning Icon

`msgbox(message,title,'custom',iconData,iconCmap)` defines a customized icon. `iconData` contains image data defining the icon; `iconCmap` is the colormap used for the image.

`msgbox(...,'createMode')` specifies whether the message box is modal or nonmodal, and if it is nonmodal, whether to replace another message box with the same `title`. Valid values for `'createMode'` are `'modal'`, `'non-modal'`, and `'replace'`.

`h = msgbox(...)` returns the handle of the box in `h`, which is a handle to a Figure graphics object.

See Also

`dialog`, `errordlg`, `inputdlg`, `helpdlg`, `questdlg`, `textwrap`, `warndlg`

newplot

Purpose

Determine where to draw graphics objects

Syntax

```
newplot
h = newplot
```

Description

`newplot` prepares a figure and axes for subsequent graphics commands.

`h = newplot` prepares a figure and axes for subsequent graphics commands and returns a handle to the current axes.

Remarks

Use `newplot` at the beginning of high-level graphics M-files to determine which figure and axes to target for graphics output. Calling `newplot` can change the current figure and current axes. Basically, there are three options when drawing graphics in existing figures and axes:

- Add the new graphics without changing any properties or deleting any objects.
- Delete all existing objects whose handles are not hidden before drawing the new objects.
- Delete all existing objects regardless of whether or not their handles are hidden and reset most properties to their defaults before drawing the new objects (refer to the following table for specific information).

The figure and axes `NextPlot` properties determine how `nextplot` behaves. The following two tables describe this behavior with various property values.

First, `newplot` reads the current figure's `NextPlot` property and acts accordingly.

NextPlot	What Happens
`add`	Draw to the current figure without clearing any graphics objects already present.
`replacechildren`	Remove all child objects whose `HandleVisibility` property is set to `on` and reset figure `NextPlot` property to `add`. This clears the current figure and is equivalent to issuing the `clf` command.

NextPlot	What Happens
replace	Remove all child objects (regardless of the setting of the HandleVisibility property) and reset figure properties to their defaults, except: • NextPlot is reset to add regardless of user-defined defaults) • Position, Units, PaperPosition, and PaperUnits are not reset This clears and resets the current figure and is equivalent to issuing the clf reset command.

After newplot establishes which figure to draw in, it reads the current axes' NextPlot property and acts accordingly.

NextPlot	Description
add	Draw into the current axes, retaining all graphics objects already present.
replacechildren	Remove all child objects whose HandleVisibility property is set to on, but do not reset axes properties. This clears the current axes like the cla command.
replace	Removes all child objects (regardless of the setting of the HandleVisibility property) and resets axes properties to their defaults, except Position and Units This clears and resets the current axes like the cla reset command.

See Also

axes, cla, clf, figure, hold, ishold, reset

The NextPlot property for figure and axes graphics objects.

noanimate

Purpose Change EraseMode of all objects to normal

Syntax

```
noanimate(state,fig_handle)
noanimate(state)
```

Description noanimate(state,fig_handle) sets the EraseMode of all image, line, patch surface, and text graphics object in the specified figure to normal. state can be the following strings:

- 'save' – set the values of the EraseMode properties to normal for all the appropriate objects in the designated figure.
- 'restore' – restore the EraseMode properties to the previous values (i.e., the values before calling noanimate with the 'save' argument).

noanimate(state) operates on the current figure.

noanimate is useful if you want to print the figure to a Tiff or JPEG format.

See Also print

Purpose Set paper orientation for printed output

Syntax

```
orient
orient landscape
orient portrait
orient tall
orient(fig_handle), orient(simulink_model)
orient(fig_handle,orientation), orient(simulink_model,orientation)
```

Description

`orient` returns a string with the current paper orientation, either `portrait`, `landscape`, or `tall`.

`orient landscape` sets the paper orientation of the current figure to full-page landscape, orienting the longest page dimension horizontally. The figure is centered on the page and scaled to fit the page with a 0.25 inch border.

`orient portrait` sets the paper orientation of the current figure to portrait, orienting the longest page dimension vertically. The `portrait` option returns the page orientation to MATLAB's default. (Note that the result of using the `portrait` option is affected by changes you make to figure properties. See the "Algorithm" section for more specific information.)

`orient tall` maps the current figure to the entire page in portrait orientation, leaving a 0.25 inch border.

`orient(fig_handle)`, `orient(simulink_model)` returns the current orientation of the specified figure or Simulink model.

`orient(fig_handle,`*`orientation`*`)`, `orient(simulink_model,`*`orientation`*`)` sets the orientation for the specified figure or Simulink model to the specified orientation (`landscape`, `portrait`, or `tall`).

Algorithm

`orient` sets the `PaperOrientation`, `PaperPosition`, and `PaperUnits` properties of the current figure. Subsequent print operations use these properties. The result of using the `portrait` option can be affected by default property values as follows:

- If the current figure `PaperType` is the same as the default figure `PaperType` and the default figure `PaperOrientation` has been set to `landscape`, then the orient portrait command uses the current values of PaperOrientation and PaperPosition to place the figure on the page.

- If the current figure PaperType is the same as the default figure PaperType and the default figure PaperOrientation has been set to landscape, then the orient portrait command uses the default figure PaperPosition with the x, y and width, height values reversed (i.e., [y,x,height,width]) to position the figure on the page.
- If the current figure PaperType is different from the default figure PaperType, then the orient portrait command uses the current figure PaperPosition with the x, y and width, height values reversed (i.e., [y,x,height,width]) to position the figure on the page.

See Also

print, set

PaperOrientation, PaperPosition, PaperSize, PaperType, and PaperUnits properties of figure graphics objects.

Purpose

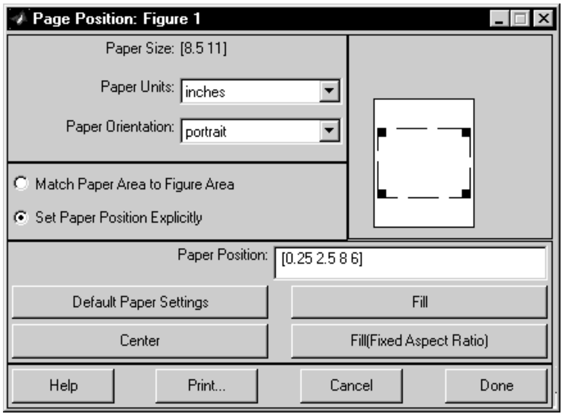
Display page position dialog box

Syntax

```
pagedlg
pagedlg(fig)
```

Description

`pagedlg` displays a page position dialog box for the current figure. The dialog box enables you to set page layout properties.



`pagedlg(fig)` displays a page position dialog box for the figure identified by the handle `fig`.

Remarks

This dialog box enables you to set figure properties that determine how MATLAB lays out the figure on the printed paper. See the dialog box help for more information.

See Also

The figure properties – `PaperPosition`, `PaperOrientation`, `PaperUnits`

Purpose Pareto chart

Syntax

```
pareto(Y)
pareto(Y,names)
pareto(Y,X)
H = pareto(...)
```

Description Pareto charts display the values in the vector `Y` as bars drawn in descending order.

`pareto(Y)` labels each bar with its element index in `Y`.

`pareto(Y,names)` labels each bar with the associated name in the string matrix or cell array `names`.

`pareto(Y,X)` labels each bar with the associated value from `X`.

`H = pareto(...)` returns a combination of patch and line object handles.

See Also `hist`, `bar`

Purpose

Create patch graphics object

Syntax

```
patch(X,Y,C)
patch(X,Y,Z,C)
patch(...'PropertyName',PropertyValue...)
patch('PropertyName',PropertyValue...)  PN/PV pairs only
handle = patch(...)
```

Description

`patch` is the low-level graphics function for creating patch graphics objects. A patch object is one or more polygons defined by the coordinates of its vertices. You can specify the coloring and lighting of the patch. See the "3-D Modeling" topic in *Using MATLAB Graphics* for more information on patches.

`patch(X,Y,C)` adds the filled two-dimensional patch to the current axes. The elements of `X` and `Y` specify the vertices of a polygon. If `X` and `Y` are matrices, MATLAB draws one polygon per column. `C` determines the color of the patch. It can be a single `ColorSpec`, one color per face, or one color per vertex (see "Remarks"). If C is a 1-by-3 vector, it is assumed to be an RGB triplet, specifying a color directly.

`patch(X,Y,Z,C)` creates a patch in three-dimensional coordinates.

`patch(...'`*`PropertyName`*`',PropertyValue...)` follows the `X`, `Y`, (`Z`), and `C` arguments with property name/property value pairs to specify additional patch properties.

`patch('`*`PropertyName`*`',PropertyValue,...)` specifies all properties using property name/property value pairs. This form enables you to omit the color specification because MATLAB uses the default face color and edge color, unless you explicitly assign a value to the `FaceColor` and `EdgeColor` properties. This form also allows you to specify the patch using the `Faces` and `Vertices` properties instead of *x*-, *y*-, and *z*-coordinates. See the "Examples" section for more information.

`handle = patch(...)` returns the handle of the patch object it creates.

Remarks

Unlike high-level area creation functions, such as `fill` or `area`, `patch` does not check the settings of the figure and axes `NextPlot` properties. It simply adds the patch object to the current axes.

If the coordinate data does not define closed polygons, `patch` closes the polygons. The data can define concave or intersecting polygons. However, if the edges of an individual patch face intersect themselves, the resulting face may or may not be completely filled. In that case, it is better to break up the face into smaller polygons.

Specifying Patch Properties

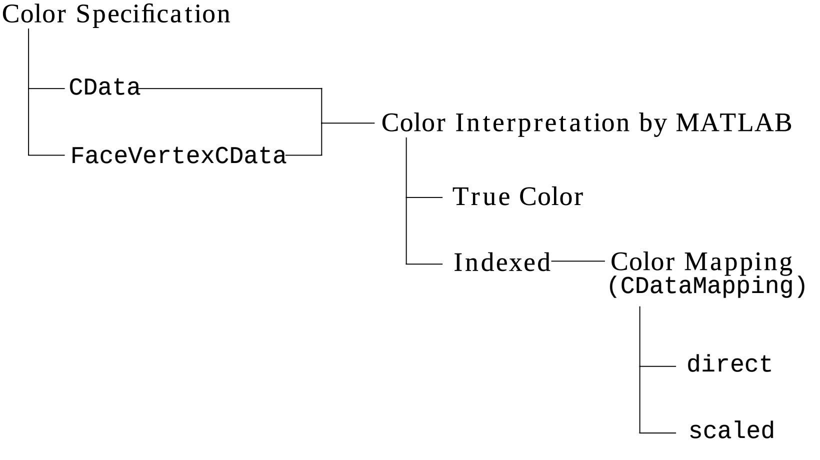
You can specify properties as property name/property value pairs, structure arrays, and cell arrays (see the `set` and `get` reference pages for examples of how to specify these data types).

There are two patch properties that specify color:

- `CData` – use when specifying *x*-, *y*-, and *z*-coordinates (`XData`, `YData`, `ZData`).
- `FaceVertexCData` – use when specifying vertices and connection matrix (`Vertices` and `Faces`).

The `CData` and `FaceVertexCData` properties accept color data as indexed or true color (RGB) values. See the `CData` and `FaceVertexCData` property descriptions for information on how to specify color.

Indexed color data can represent either direct indices into the colormap or scaled values that map the data linearly to the entire colormap (see the `caxis` function for more information on this scaling). The `CDataMapping` property determines how MATLAB interprets indexed color data.



Color Data Interpretation

You can specify patch colors as:

- A single color for all faces
- One color for each face enabling flat coloring
- One color for each vertex enabling interpolated coloring

The following tables summarize how MATLAB interprets color data defined by the `CData` and `FaceVertexCData` properties.

Interpretation of the CData Property

[X,Y,Z]Data Dimensions	CData Required for Indexed	True Color	Results Obtained
m-by-n	scalar	1-by-1-by-3	Use the single color specified for all patch faces. Edges can be only a single color.
m-by-n	1-by-n (n >= 4)	1-by-n-by-3	Use one color for each patch face. Edges can be only a single color.
m-by-n	m-by-n	m-by-n-3	Assign a color to each vertex. patch faces can be flat (a single color) or interpolated. Edges can be flat or interpolated.

Interpretation of the FaceVertexCData Property

Vertices Dimensions	Faces Dimensions	FaceVertexCData Required for Indexed	True Color	Results Obtained
m-by-n	k-by-3	scalar	1-by-3	Use the single color specified for all patch faces. Edges can be only a single color.

Vertices Dimensions	Faces Dimensions	FaceVertexCData Required for Indexed	True Color	Results Obtained
m-by-n	k-by-3	k-by-1	k-by-3	Use one color for each patch face. Edges can be only a single color.
m-by-n	k-by-3	m-by-1	m-by-3	Assign a color to each vertex. patch faces can be flat (a single color) or interpolated. Edges can be flat or interpolated.

Examples

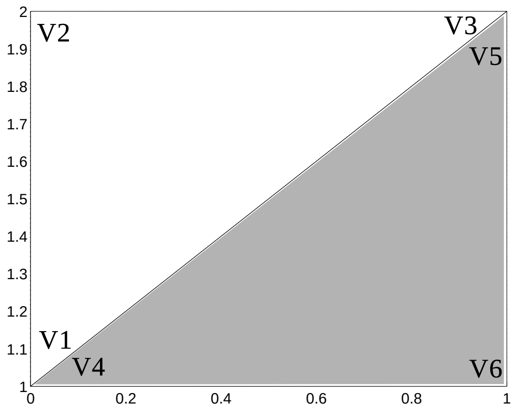
This example creates a patch object using two different methods:

- Specifying *x*-, *y*-, and *z*-coordinates and color data (`XData`, `YData`, `ZData`, and `CData` properties).
- Specifying vertices, the connection matrix, and color data (`Vertices`, `Faces`, `FaceVertexCData`, and `FaceColor` properties).

Specifying X, Y, and Z Coordinates

The first approach specifies the coordinates of each vertex. In this example, the coordinate data defines two triangular faces, each having three vertices. Using true color, the top face is set to white and the bottom face to gray.

```
x = [0 0;0 1;1 1];
y = [1 1;2 2;2 1];
z = [1 1;1 1;1 1];
tcolor(1,1,1:3) = [1 1 1];
tcolor(1,2,1:3) = [.7 .7 .7];
patch(x,y,z,tcolor)
```



Notice that each face shares two vertices with the other face (V_1-V_4 and V_3-V_5).

Specifying Vertices and Faces

The `Vertices` property contains the coordinates of each *unique* vertex defining the patch. The `Faces` property specifies how to connect these vertices to form each face of the patch. For this example, two vertices share the same location so you need to specify only four of the six vertices. Each row contains the *x*, *y*, and *z*-coordinates of each vertex.

```
vert = [0 1 1;0 2 1;1 2 1;1 1 1];
```

There are only two faces, defined by connecting the vertices in the order indicated.

```
fac = [1 2 3;1 3 4];
```

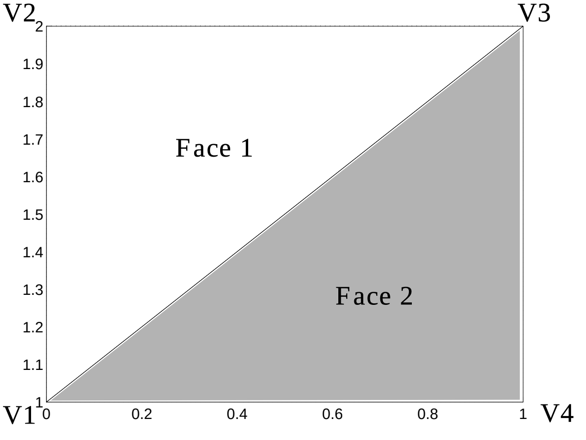
To specify the face colors, define a 2-by-3 matrix containing two RGB color definitions.

```
tcolor = [1 1 1;.7 .7 .7];
```

With two faces and two colors, MATLAB can color each face with flat shading. This means you must set the `FaceColor` property to `flat`, since the faces/vertices technique is available only as a low-level function call (i.e., only by specifying property name/property value pairs).

Create the patch by specifying the `Faces`, `Vertices`, and `FaceVertexCData` properties as well as the `FaceColor` property.

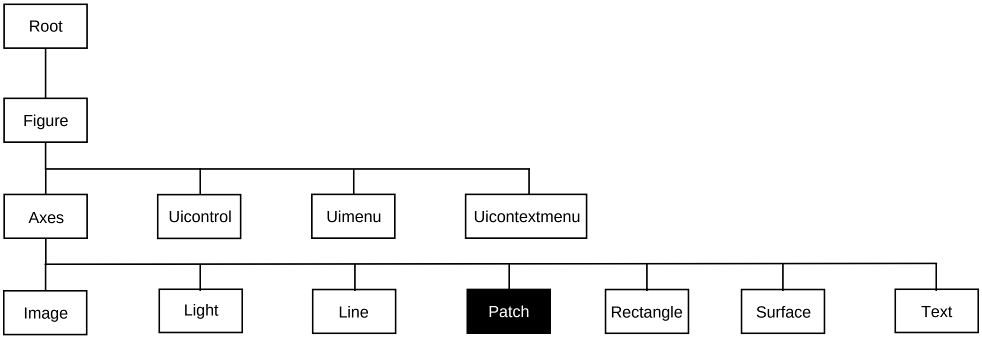
```
patch('faces',fac,'vertices',vert,'FaceVertexCData',tcolor,...
      'FaceColor','flat')
```



Specifying only unique vertices and their connection matrix can reduce the size of the data for patches having many faces. See the descriptions of the `Faces`, `Vertices`, and `FaceVertexCData` properties for information on how to define them.

MATLAB does not require each face to have the same number of vertices. In cases where they do not, pad the `Faces` matrix with `NaN`s. To define a patch with faces that do not close, add one or more `NaN` to the row in the `Vertices` matrix that defines the vertex you do not want connected.

Object Hierarchy



Setting Default Properties

You can set default patch properties on the axes, figure, and root levels.

```
set(0,'DefaultPatchPropertyName',PropertyValue...)
set(gcf,'DefaultPatchPropertyName',PropertyValue...)
set(gca,'DefaultPatchPropertyName',PropertyValue...)
```

PropertyName is the name of the patch property and `PropertyValue` is the value you are specifying. Use `set` and `get` to access patch properties.

Property List

The following table lists all patch properties and provides a brief description of each. The property name links take you to an expanded description of the properties.

Property Name	Property Description	Property Value
Data Defining the Object		
`Faces`	Connection matrix for `Vertices`	Values: m-by-n matrix Default: `[1,2,3]`
`Vertices`	Matrix of *x*-, *y*-, and *z*-coordinates of the vertices (used with `Faces`)	Values: matrix Default: `[0,1;1,1;0,0]`
`XData`	The *x*-coordinates of the vertices of the patch	Values: vector or matrix Default: `[0;1;0]`
`YData`	The *y*-coordinates of the vertices of the patch	Values: vector or matrix Default: `[1;1;0]`

Property Name	Property Description	Property Value
ZData	The *z*-coordinates of the vertices of the patch	Values: vector or matrix Default: [] empty matrix
Specifying Color		
CData	Color data for use with the XData/YData/ZData method	Values: scalar, vector, or matrix Default: [] empty matrix
CDataMapping	Controls mapping of CData to colormap	Values: scaled, direct Default: scaled
EdgeColor	Color of face edges	Values: ColorSpec, none, flat, interp Default: ColorSpec
FaceColor	Color of face	Values: ColorSpec, none, flat, interp Default: ColorSpec
FaceVertexCData	Color data for use with Faces/Vertices method	Values: matrix Default: [] empty matrix
MarkerEdgeColor	Color of marker or the edge color for filled markers	Values: ColorSpec, none, auto Default: auto
MarkerFaceColor	Fill color for markers that are closed shapes	Values: ColorSpec, none, auto Default: none
Controlling the Effects of Lights		
AmbientStrength	Intensity of the ambient light	Values: scalar >=0 and <=1 Default: 0.3
BackFaceLighting	Controls lighting of faces pointing away from camera	Values: unlit, lit, reverselit Default: reverselit

Property Name	Property Description	Property Value
DiffuseStrength	Intensity of diffuse light	Values: scalar >=0 and <=1 Default: 0.6
EdgeLighting	Method used to light edges	Values: none, flat, gouraud, phong Default: none
FaceLighting	Method used to light edges	Values: none, flat, gouraud, phong Default: none
NormalMode	MATLAB-generated or user-specified normal vectors	Values: auto, manual Default: auto
SpecularColorReflectance	Composite color of specularly reflected light	Values: scalar 0 to 1 Default: 1
SpecularExponent	Harshness of specular reflection	Values: scalar >= 1 Default: 10
SpecularStrength	Intensity of specular light	Values: scalar >=0 and <=1 Default: 0.9
VertexNormals	Vertex normal vectors	Values: matrix
Defining Edges and Markers		
LineStyle	Select from five line styles.	Values: –, ––, :, –., none Default: –
LineWidth	The width of the edge in points	Values: scalar Default: 0.5 points
Marker	Marker symbol to plot at data points	Values: see Marker property Default: none
MarkerSize	Size of marker in points	Values: size in points Default: 6
Controlling the Appearance		

Property Name	Property Description	Property Value
Clipping	Clipping to axes rectangle	Values: on, off Default: on
EraseMode	Method of drawing and erasing the patch (useful for animation)	Values: normal, none, xor, background Default: normal
SelectionHighlight	Highlight patch when selected (Selected property set to on)	Values: on, off Default: on
Visible	Make the patch visible or invisible	Values: on, off Default: on
Controlling Access to Objects		
HandleVisibility	Determines if and when the the patch's handle is visible to other functions	Values: on, callback, off Default: on
HitTest	Determines if the patch can become the current object (see the figure CurrentObject property)	Values: on, off Default: on
Controlling Callback Routine Execution		
BusyAction	Specify how to handle callback routine interruption	Values: cancel, queue Default: queue
ButtonDownFcn	Define a callback routine that executes when a mouse button is pressed on over the patch	Values: string Default: '' (empty string)
CreateFcn	Define a callback routine that executes when an patch is created	Values: string Default: '' (empty string)
DeleteFcn	Define a callback routine that executes when the patch is deleted (via close or delete)	Values: string Default: '' (empty string)

Property Name	Property Description	Property Value
Interruptible	Determine if callback routine can be interrupted	Values: on, off Default: on (can be interrupted)
UIContextMenu	Associate a context menu with the patch	Values: handle of a Uicontrextmenu
General Information About the Patch		
Children	Patch objects have no children	Values: [] (empty matrix)
Parent	The parent of a patch object is always an axes object	Value: axes handle
Selected	Indicate whether the patch is in a “selected” state.	Values: on, off Default: on
Tag	User-specified label	Value: any string Default: '' (empty string)
Type	The type of graphics object (read only)	Value: the string 'patch'
UserData	User-specified data	Values: any matrix Default: [] (empty matrix)

Patch Properties

This section lists property names along with the type of values each accepts. Curly braces { } enclose default values.

`AmbientStrength` scalar >= 0 and <= 1

Strength of ambient light. This property sets the strength of the ambient light, which is a nondirectional light source that illuminates the entire scene. You must have at least one visible light object in the axes for the ambient light to be visible. The axes `AmbientColor` property sets the color of the ambient light, which is therefore the same on all objects in the axes.

You can also set the strength of the diffuse and specular contribution of light objects. See the `DiffuseStrength` and `SpecularStrength` properties.

`BackFaceLighting` `unlit | lit | {reverselit}`

Face lighting control. This property determines how faces are lit when their vertex normals point away from the camera:

- `unlit` – face is not lit
- `lit` – face lit in normal way
- `reverselit` – face is lit as if the vertex pointed towards the camera

This property is useful for discriminating between the internal and external surfaces of an object. See the *Using MATLAB Graphics* manual for an example.

`BusyAction` `cancel | {queue}`

Callback routine interruption. The `BusyAction` property enables you to control how MATLAB handles events that potentially interrupt executing callback routines. If there is a callback routine executing, subsequently invoked callback routes always attempt to interrupt it. If the `Interruptible` property of the object whose callback is executing is set to `on` (the default), then interruption occurs at the next point where the event queue is processed. If the `Interruptible` property is `off`, the `BusyAction` property (of the object owning the executing callback) determines how MATLAB handles the event. The choices are:

- `cancel` – discard the event that attempted to execute a second callback routine.
- `queue` – queue the event that attempted to execute a second callback routine until the current callback finishes.

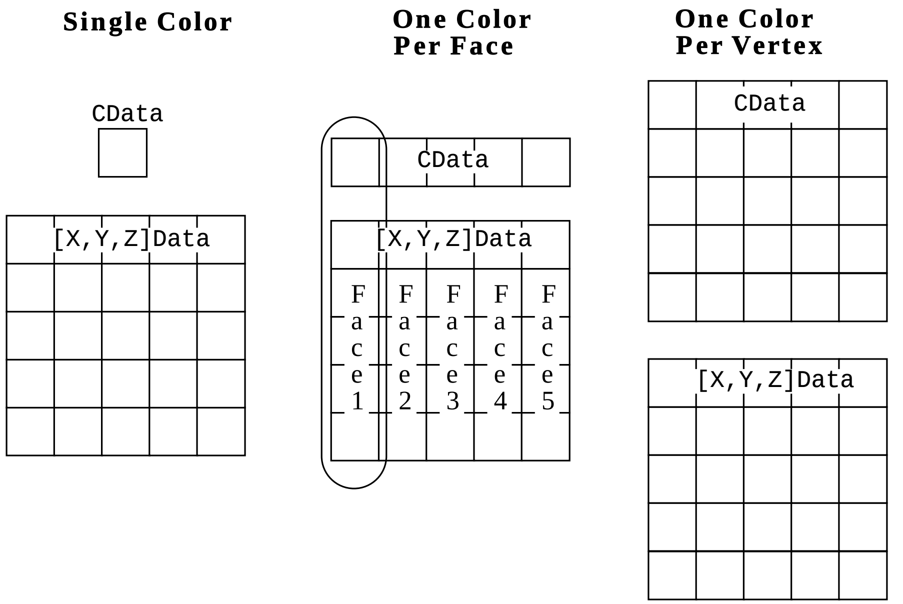
ButtonDownFcn string

Button press callback routine. A callback routine that executes whenever you press a mouse button while the pointer is over the patch object. Define this routine as a string that is a valid MATLAB expression or the name of an M-file. The expression executes in the MATLAB workspace.

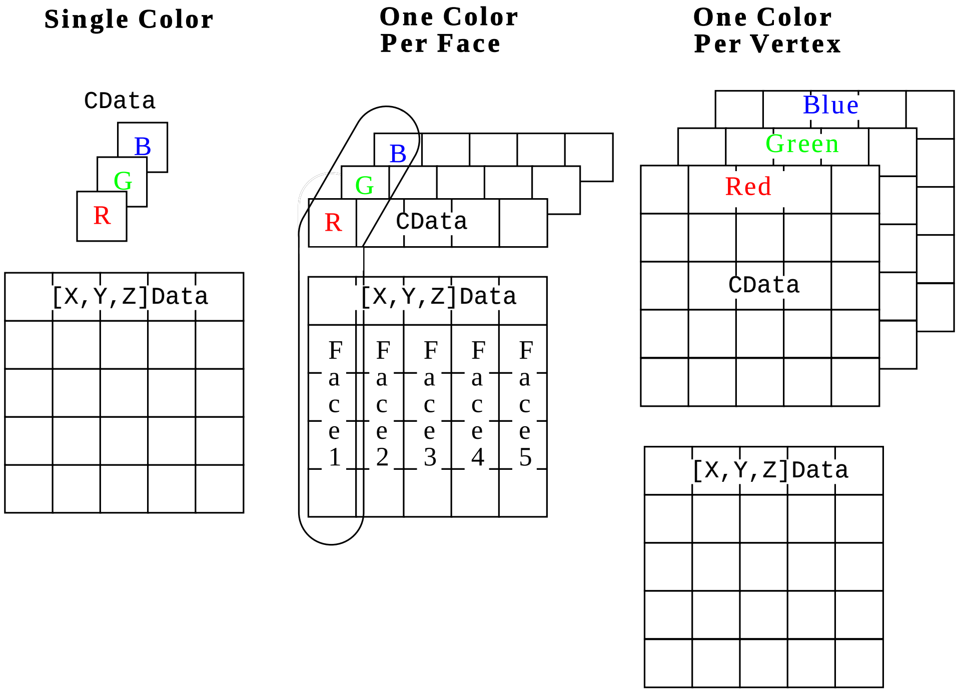
CData scalar, vector, or matrix

Patch colors. This property specifies the color of the patch. You can specify color for each vertex, each face, or a single color for the entire patch. The way MATLAB interprets `CData` depends on the type of data supplied. The data can be numeric values that are scaled to map linearly into the current colormap, integer values that are used directly as indices into the current colormap, or arrays of RGB values. RGB values are not mapped into the current colormap, but interpreted as the colors defined. On true color systems, MATLAB uses the actual colors defined by the RGB triples. On pseudocolor systems, MATLAB uses dithering to approximate the RGB triples using the colors in the figure's `Colormap` and `Dithermap`.

The following two diagrams illustrate the dimensions of `CData` with respect to the coordinate data arrays, `XData`, `YData`, and `ZData`. The first diagram illustrates the use of indexed color.


Single Color
CData
[X,Y,Z]Data
One Color Per Face
CData
[X,Y,Z]Data
Face 1
Face 2
Face 3
Face 4
Face 5
One Color Per Vertex
CData
[X,Y,Z]Data

The second diagram illustrates the use of true color. True color requires *m*-by-*n*-by-3 arrays to define red, green, and blue components for each color.



Note that if `CData` contains `NaNs`, MATLAB does not color the faces.

See also the `Faces`, `Vertices`, and `FaceVertexCData` properties for an alternative method of patch definition.

`CDataMapping` `{scaled} | direct`

Direct or scaled color mapping. This property determines how MATLAB interprets indexed color data used to color the patch. (If you use true color specification for `CData` or `FaceVertexCData`, this property has no effect.)

- `scaled` – transform the color data to span the portion of the colormap indicated by the axes `CLim` property, linearly mapping data values to colors. See the `caxis` command for more information on this mapping.
- `direct` – use the color data as indices directly into the colormap. When not scaled, the data are usually integer values ranging from 1 to

`length(colormap)`. MATLAB maps values less than 1 to the first color in the colormap, and values greater than `length(colormap)` to the last color in the colormap. Values with a decimal portion are fixed to the nearest, lower integer.

`Children` matrix of handles

Always the empty matrix; patch objects have no children.

`Clipping` `{on} | off`

Clipping to axes rectangle. When `Clipping` is `on`, MATLAB does not display any portion of the patch outside the axes rectangle.

`CreateFcn` string

Callback routine executed during object creation. This property defines a callback routine that executes when MATLAB creates a patch object. You must define this property as a default value for patches. For example, the statement,

```
set(0,'DefaultPatchCreateFcn','set(gcf,''DitherMap'',my_dither_map)')
```

defines a default value on the root level that sets the figure `DitherMap` property whenever you create a patch object. MATLAB executes this routine after setting all properties for the patch created. Setting this property on an existing patch object has no effect.

The handle of the object whose `CreateFcn` is being executed is accessible only through the root `CallbackObject` property, which you can query using `gcbo`.

`DeleteFcn` string

Delete patch callback routine. A callback routine that executes when you delete the patch object (e.g., when you issue a `delete` command or clear the axes (`cla`) or figure (`clf`) containing the patch). MATLAB executes the routine before deleting the object's properties so these values are available to the callback routine.

The handle of the object whose `DeleteFcn` is being executed is accessible only through the root `CallbackObject` property, which you can query using `gcbo`.

`DiffuseStrength` scalar >= 0 and <= 1

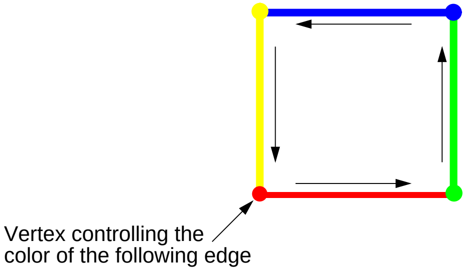
Intensity of diffuse light. This property sets the intensity of the diffuse component of the light falling on the patch. Diffuse light comes from light objects in the axes.

You can also set the intensity of the ambient and specular components of the light on the patch object. See the AmbientStrength and SpecularStrength properties.

EdgeColor {ColorSpec} | none | flat | interp

Color of the patch edge. This property determines how MATLAB colors the edges of the individual faces that make up the patch.

- ColorSpec – A three-element RGB vector or one of MATLAB's predefined names, specifying a single color for edges. The default edge color is black. See ColorSpec for more information on specifying color.
- none – Edges are not drawn.
- flat – The color of each vertex controls the color of the edge that follows it. This means flat edge coloring is dependent on the order you specify the vertices:



- interp – Linear interpolation of the CData or FaceVertexCData values at the vertices determines the edge color.

EdgeLighting {none} | flat | gouraud | phong

Algorithm used for lighting calculations. This property selects the algorithm used to calculate the effect of light objects on patch edges. Choices are:

- none – Lights do not affect the edges of this object.
- flat – The effect of light objects is uniform across each edge of the patch.
- gouraud – The effect of light objects is calculated at the vertices and then linearly interpolated across the edge lines.
- phong – The effect of light objects is determined by interpolating the vertex normals across each edge line and calculating the reflectance at each pixel.

Phong lighting generally produces better results than Gouraud lighting, but takes longer to render.

EraseMode {normal} | none | xor | background

Erase mode. This property controls the technique MATLAB uses to draw and erase patch objects. Alternative erase modes are useful in creating animated sequences, where control of the way individual objects redraw is necessary to improve performance and obtain the desired effect.

- normal – Redraw the affected region of the display, performing the three-dimensional analysis necessary to ensure that all objects are rendered correctly. This mode produces the most accurate picture, but is the slowest. The other modes are faster, but do not perform a complete redraw and are therefore less accurate.
- none – Do not erase the patch when it is moved or destroyed. While the object is still visible on the screen after erasing with EraseMode none, you cannot print it because MATLAB stores no information about its former location.
- xor– Draw and erase the patch by performing an exclusive OR (XOR) with each pixel index of the screen behind it. Erasing the patch does not damage the color of the objects behind it. However, patch color depends on the color of the screen behind it and is correctly colored only when over the axes background Color, or the figure background Color if the axes Color is set to none.
- background – Erase the patch by drawing it in the axes' background Color, or the figure background Color if the axes Color is set to none. This damages objects that are behind the erased patch, but the patch is always properly colored.

Printing with Non-normal Erase Modes. MATLAB always prints figures as if the EraseMode of all objects is normal. This means graphics objects created with EraseMode set to none, xor, or background can look different on screen than on paper. On screen, MATLAB may mathematically combine layers of colors (e.g., XORing a pixel color with that of the pixel behind it) and ignore three-dimensional sorting to obtain greater rendering speed. However, these techniques are not applied to the printed output.

You can use the MATLAB getframe command or other screen capture application to create an image of a figure containing non-normal mode objects.

FaceColor `{ColorSpec} | none | flat | interp`

Color of the patch face. This property can be any of the following:

- `ColorSpec` – A three-element RGB vector or one of MATLAB's predefined names, specifying a single color for faces. See `ColorSpec` for more information on specifying color.
- `none` – Do not draw faces. Note that edges are drawn independently of faces.
- `flat` – The values of `CData` or `FaceVertexCData` determine the color for each face in the patch. The color data at the first vertex determines the color of the entire face.
- `interp` – Bilinear interpolation of the color at each vertex determines the coloring of each face.

FaceLighting `{none} | flat | gouraud | phong`

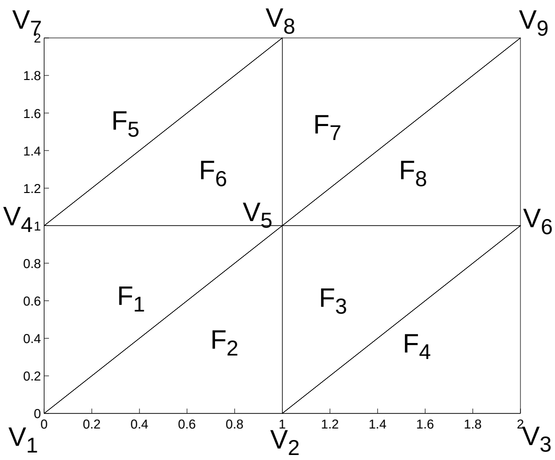
Algorithm used for lighting calculations. This property selects the algorithm used to calculate the effect of light objects on patch faces. Choices are:

- `none` – Lights do not affect the faces of this object.
- `flat` – The effect of light objects is uniform across the faces of the patch. Select this choice to view faceted objects.
- `gouraud` – The effect of light objects is calculated at the vertices and then linearly interpolated across the faces. Select this choice to view curved surfaces.
- `phong` – The effect of light objects is determined by interpolating the vertex normals across each face and calculating the reflectance at each pixel. Select this choice to view curved surfaces. Phong lighting generally produces better results than Gouraud lighting, but takes longer to render.

Faces m-by-n matrix

Vertex connection defining each face. This property is the connection matrix specifying which vertices in the `Vertices` property are connected. The `Faces` matrix defines *m* faces with up to *n* vertices each. Each row designates the connections for a single face, and the number of elements in that row that are not `NaN` defines the number of vertices for that face.

The `Faces` and `Vertices` properties provide an alternative way to specify a patch that can be more efficient than using *x*, *y*, and *z* coordinates in most cases. For example, consider the following patch. It is composed of eight triangular faces defined by nine vertices.



`Faces` property

F_1	V_1	V_4	V_5
F_2	V_1	V_5	V_2
F_3	V_2	V_5	V_6
F_4	V_2	V_6	V_3
F_5	V_4	V_7	V_8
F_6	V_4	V_8	V_5
F_7	V_5	V_8	V_9
F8	V_5	V_9	V_6

`Vertices` property

V_1	X_1	Y_1	Z_1
V_2	X_2	Y_2	Z_2
V_3	X_3	Y_3	Z_3
V_4	X_4	Y_4	Z_4
V_5	X_5	Y_5	Z_5
V_6	X_6	Y_6	Z_6
V_7	X_7	Y_7	Z_7
V_8	X_8	Y_8	Z_8
V_9	X_9	Y_9	Z_9

The corresponding `Faces` and `Vertices` properties are shown to the right of the patch. Note how some faces share vertices with other faces. For example, the fifth vertex (V5) is used six times, once each by faces one, two, and three and six, seven, and eight. Without sharing vertices, this same patch requires 24 vertex definitions.

`FaceVertexCData` matrix

Face and vertex colors. The `FaceVertexCData` property specifies the color of patches defined by the Faces and Vertices properties, and the values are used when `FaceColor`, `EdgeColor`, `MarkerFaceColor`, or `MarkerEdgeColor` are set appropriately. The interpretation of the values specified for `FaceVertexCData` depends on the dimensions of the data.

For indexed colors, `FaceVertexCData` can be:

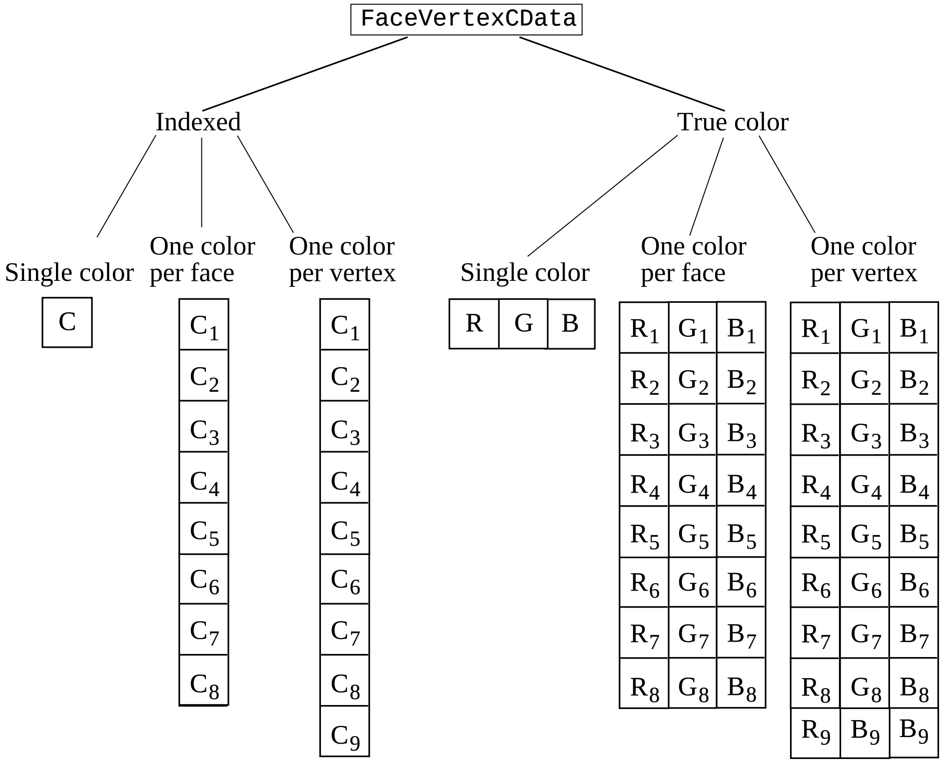
- A single value, which applies a single color to the entire patch
- An *n*-by-1 matrix, where *n* is the number of rows in the `Faces` property, which specifies one color per face

- An *n*-by-1 matrix, where *n* is the number of rows in the `Vertices` property, which specifies one color per vertex

For true colors, `FaceVertexCData` can be:

- A 1-by-3 matrix , which applies a single color to the entire patch
- An *n*-by-3 matrix, where *n* is the number of rows in the `Faces` property, which specifies one color per face
- An *n*-by-3 matrix, where *n* is the number of rows in the `Vertices` property, which specifies one color per vertex

The following diagram illustrates the various forms of the `FaceVertexCData` property for a patch having eight faces and nine vertices. The `CDataMapping` property determines how MATLAB interprets the `FaceVertexCData` property when you specify indexed colors.



HandleVisibility `{on} | callback | off`

Control access to object's handle by command-line users and GUIs. This property determines when an object's handle is visible in its parent's list of children. `HandleVisibility` is useful for preventing command-line users from accidentally drawing into or deleting a figure that contains only user interface devices (such as a dialog box).

Handles are always visible when `HandleVisibility` is `on`.

Setting `HandleVisibility` to `callback` causes handles to be visible from within callback routines or functions invoked by callback routines, but not from within functions invoked from the command line. This provides a means to protect GUIs from command-line users, while allowing callback routines to have complete access to object handles.

Setting `HandleVisibility` to `off` makes handles invisible at all times. This may be necessary when a callback routine invokes a function that might potentially damage the GUI (such as evaluating a user-typed string), and so temporarily hides its own handles during the execution of that function.

When a handle is not visible in its parent's list of children, it cannot be returned by functions that obtain handles by searching the object hierarchy or querying handle properties. This includes `get`, `findobj`, `gca`, `gcf`, `gco`, `newplot`, `cla`, `clf`, and `close`.

When a handle's visibility is restricted using `callback` or `off`, the object's handle does not appear in its parent's `Children` property, figures do not appear in the root's `CurrentFigure` property, objects do not appear in the root's `CallbackObject` property or in the figure's `CurrentObject` property, and axes do not appear in their parent's `Currentaxes` property.

You can set the root `ShowHiddenHandles` property to `on` to make all handles visible, regardless of their `HandleVisibility` settings (this does not affect the values of the `HandleVisibility` properties).

Handles that are hidden are still valid. If you know an object's handle, you can `set` and `get` its properties, and pass it to any function that operates on handles.

`HitTest` `{on} | off`

Selectable by mouse click. `HitTest` determines if the patch can become the current object (as returned by the `gco` command and the figure `CurrentObject` property) as a result of a mouse click on the patch. If `HiTest` is `off`, clicking on the patch selects the object below it (which maybe the axes containing it).

`Interruptible` `{on} | off`

Callback routine interruption mode. The `Interruptible` property controls whether a patch callback routine can be interrupted by subsequently invoked callback routines. Only callback routines defined for the `ButtonDownFcn` are affected by the `Interruptible` property. MATLAB checks for events that can interrupt a callback routine only when it encounters a `drawnow`, `figure`, `getframe`, or `pause` command in the routine. See the `BusyAction` property for related information.

LineStyle {–} | –– | : | –. | none

Edge linestyle. This property specifies the line style of the patch edges. The following table lists the available line styles.

Symbol	Line Style
–	solid line (default)
––	dashed line
:	dotted line
–.	dash-dot line
none	no line

You can use LineStyle none when you want to place a marker at each point but do not want the points connected with a line (see the Marker property).

LineWidth scalar

Edge line width. The width, in points, of the patch edges (1 point = $^1/_{72}$ inch). The default LineWidth is 0.5 points.

Marker character (see table)

Marker symbol. The Marker property specifies marks that locate vertices. You can set values for the Marker property independently from the LineStyle property. The following tables lists the available markers.

Marker Specifier	Description
+	plus sign
o	circle
*	asterisk
.	point
x	cross
s	square

Marker Specifier	Description
d	diamond
^	upward pointing triangle
v	downward pointing triangle
>	right pointing triangle
<	left pointing triangle
p	five-pointed star (pentagram)
h	six-pointed star (hexagram)
none	no marker (default)

MarkerEdgeColor ColorSpec | none | {auto} | flat

Marker edge color. The color of the marker or the edge color for filled markers (circle, square, diamond, pentagram, hexagram, and the four triangles). ColorSpec defines the color to use. none specifies no color, which makes nonfilled markers invisible. auto sets MarkerEdgeColor to the same color as the EdgeColor property.

MarkerFaceColor ColorSpec | {none} | auto | flat

Marker face color. The fill color for markers that are closed shapes (circle, square, diamond, pentagram, hexagram, and the four triangles). ColorSpec defines the color to use. none makes the interior of the marker transparent, allowing the background to show through. auto sets the fill color to the axes color, or the figure color, if the axes Color property is set to none.

MarkerSize size in points

Marker size. A scalar specifying the size of the marker, in points. The default value for MarkerSize is six points (1 point = $^{1}/_{72}$ inch). Note that MATLAB draws the point marker at 1/3 of the specified size.

NormalMode {auto} | manual

MATLAB-generated or user-specified normal vectors. When this property is auto, MATLAB calculates vertex normals based on the coordinate data. If you specify your own vertex normals, MATLAB sets this property to manual and does not generate its own data. See also the `VertexNormals` property.

`Parent` axes handle

Patch's parent. The handle of the patch's parent object. The parent of a patch object is the axes in which it is displayed. You can move a patch object to another axes by setting this property to the handle of the new parent.

`Selected` `on | {off}`

Is object selected? When this property is `on`, MATLAB displays selection handles or a dashed box (depending on the number of faces) if the `SelectionHighlight` property is also `on`. You can, for example, define the `ButtonDownFcn` to set this property, allowing users to select the object with the mouse.

`SelectionHighlight` `{on} | off`

Objects highlight when selected. When the `Selected` property is `on`, MATLAB indicates the selected state by:

- Drawing handles at each vertex for a single-faced patch.
- Drawing a dashed bounding box for a multi-faced patch.

When `SelectionHighlight` is `off`, MATLAB does not draw the handles.

`SpecularColorReflectance` scalar in the range 0 to 1

Color of specularly reflected light. When this property is 0, the color of the specularly reflected light depends on both the color of the object from which it reflects and the color of the light source. When set to 1, the color of the specularly reflected light depends only on the color or the light source (i.e., the light object `Color` property). The proportions vary linearly for values in between.

`SpecularExponent` scalar >= 1

Harshness of specular reflection. This property controls the size of the specular spot. Most materials have exponents in the range of 5 to 20.

`SpecularStrength` scalar >= 0 and <= 1

Intensity of specular light. This property sets the intensity of the specular component of the light falling on the patch. Specular light comes from light objects in the axes.

You can also set the intensity of the ambient and diffuse components of the light on the patch object. See the `AmbientStrength` and `DiffuseStrength` properties.

Tag string

User-specified object label. The `Tag` property provides a means to identify graphics objects with a user-specified label. This is particularly useful when constructing interactive graphics programs that would otherwise need to define object handles as global variables or pass them as arguments between callback routines.

For example, suppose you use patch objects to create borders for a group of uicontrol objects and want to change the color of the borders in a uicontrol's callback routine. You can specify a `Tag` with the patch definition:

```
patch(X,Y,'k','Tag','PatchBorder')
```

Then use `findobj` in the uicontrol's callback routine to obtain the handle of the patch and set its `FaceColor` property.

```
set(findobj('Tag','PatchBorder'),'FaceColor','w')
```

Type string (read only)

Class of the graphics object. For patch objects, `Type` is always the string `'patch'`.

UIContextMenu handle of a uicontextmenu object

Associate a context menu with the patch. Assign this property the handle of a uicontextmenu object created in the same figure as the patch. Use the `uicontextmenu` function to create the context menu. MATLAB displays the context menu whenever you right-click over the patch.

UserData matrix

User-specified data. Any matrix you want to associate with the patch object. MATLAB does not use this data, but you can access it using `set` and `get`.

VertexNormals matrix

Surface normal vectors. This property contains the vertex normals for the patch. MATLAB generates this data to perform lighting calculations. You can supply your own vertex normal data, even if it does not match the coordinate data. This can be useful to produce interesting lighting effects.

Vertices matrix

Vertex coordinates. A matrix containing the *x*-, *y*-, *z*-coordinates for each vertex. See the Faces property for more information.

Visible {on} | off

Patch object visibility. By default, all patches are visible. When set to off, the patch is not visible, but still exists and you can query and set its properties.

XData vector or matrix

X-coordinates. The *x*-coordinates of the points at the vertices of the patch. If XData is a matrix, each column represents the *x*-coordinates of a single face of the patch. In this case, XData, YData, and ZData must have the same dimensions.

YData vector or matrix

Y-coordinates. The *y*-coordinates of the points at the vertices of the patch. If YData is a matrix, each column represents the *y*-coordinates of a single face of the patch. In this case, XData, YData, and ZData must have the same dimensions.

ZData vector or matrix

Z-coordinates. The *z*-coordinates of the points at the vertices of the patch. If ZData is a matrix, each column represents the *z*-coordinates of a single face of the patch. In this case, XData, YData, and ZData must have the same dimensions.

See Also

area, caxis, fill, fill3, surface

Purpose

Set or query the plot box aspect ratio

Syntax

```
pbaspect
pbaspect([aspect_ratio])
pbaspect('mode')
pbaspect('auto')
pbaspect('manual')
pbaspect(axes_handle,...)
```

Description

The plot box aspect ratio determines the relative size of the x-, y-, and z-axes.

`pbaspect` with no arguments returns the plot box aspect ratio of the current axes.

`pbaspect([aspect_ratio])` sets the plot box aspect ratio in the current axes to the specified value. Specify the aspect ratio as three relative values representing the ratio of the x-, y-, and z-axes size. For example, a value of `[1 1 1]` (the default) means the plot box is a cube (although with stretch-to-fill enabled, it may not appear as a cube). See Remarks.

`pbaspect('mode')` returns the current value of the plot box aspect ratio mode, which can be either `auto` (the default) or `manual`. See Remarks.

`pbaspect('auto')` sets the plot box aspect ratio mode to `auto`.

`pbaspect('manual')` sets the plot box aspect ratio mode to `manual`.

`pbaspect(axes_handle,...)` performs the set or query on the axes identified by the first argument, `axes_handle`. If you do not specify an axes handle, `pbaspect` operates on the current axes.

Remarks

`pbaspect` sets or queries values of the axes object `PlotBoxAspectRatio` and `PlotBoxAspectRatioMode` properties.

When the plot box aspect ratio mode is `auto`, MATLAB sets the ratio to `[1 1 1]`, but may change it to accommodate manual settings of the data aspect ratio, camera view angle, or axis limits. See the axes `DataAspectRatio` property for a table listing the interactions between various properties.

Setting a value for the plot box aspect ratio or setting the plot box aspect ratio mode to `manual` disables MATLAB's stretch-to-fill feature (stretching of the axes to fit the window). This means setting the plot box aspect ratio to its current value,

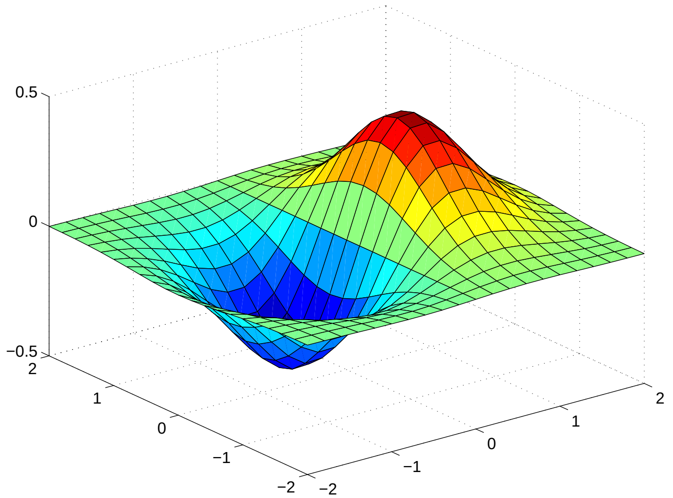
```
pbaspect(pbaspect)
```

can cause a change it the way the graphs look. See the Remarks section of the axes reference description and the "Aspect Ratio" section in the *Using MATLAB Graphics* manual for a discussion of stretch-to-fill.

Examples

The following surface plot of the function $z = xe^{(-x^2 - y^2)}$ is useful to illustrate the plot box aspect ratio. First plot the function over the range $-2 \le x \le 2$, $-2 \le y \le 2$,

```
[x,y] = meshgrid([-2:.2:2]);
z = x.*exp(-x.^2 - y.^2);
surf(x,y,z)
```



Querying the plot box aspect ratio shows that the plot box is square.

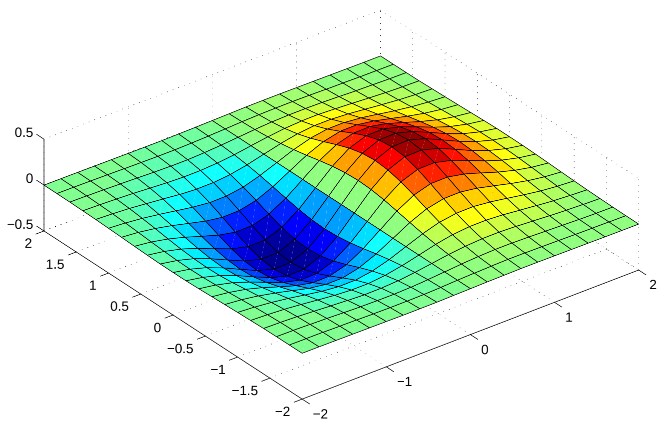
```
pbaspect
ans =
     1   1   1
```

It is also interesting to look at the data aspect ratio selected by MATLAB.

```
daspect
ans =
     4   4   1
```

To illustrate the interaction between the plot box and data aspect ratios, set the data aspect ratio to [1 1 1] and again query the plot box aspect ratio.

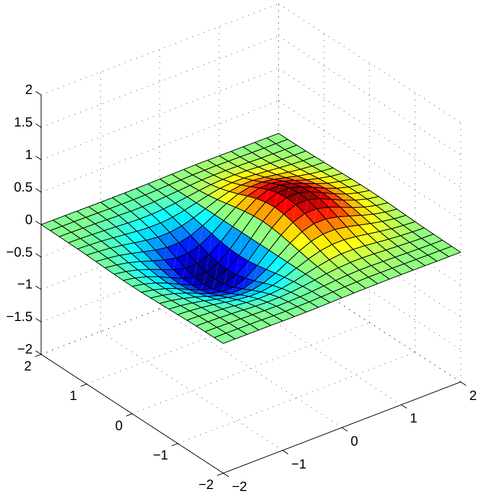
```
daspect([1 1 1])
```




```
pbaspect
ans =
     4   4   1
```

The plot box aspect ratio has changed to accommodate the specified data aspect ratio. Now suppose you want the plot box aspect ratio to be [1 1 1] as well.

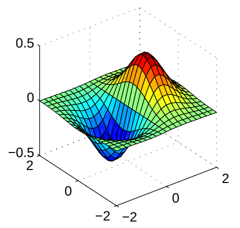
```
pbaspect([1 1 1])
```

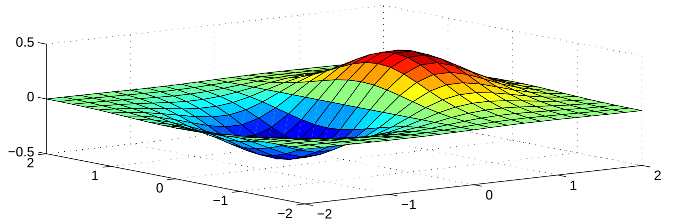


Notice how MATLAB changed the axes limits because of the constraints introduced by specifying both the plot box and data aspect ratios.

You can also use pbaspect to disable stretch-to-fill. For example, displaying two subplots in one figure can give surface plots a squashed appearance. Disabling stretch-to-fill.

```
upper_plot = subplot(211);
surf(x,y,z)
lower_plot = subplot(212);
surf(x,y,z)
pbaspect(upper_plot,'manual')
```





See Also

`axis`, `daspect`, `xlim`, `ylim`, `zlim`

The axes properties `DataAspectRatio`, `PlotBoxAspectRatio`, `XLim`, `YLim`, `ZLim`

The "Aspect Ratio" section in the *Using MATLAB Graphics* manual.

Purpose Pseudocolor plot

Syntax

```
pcolor(C)
pcolor(X,Y,C)
h = pcolor(...)
```

Description A pseudocolor plot is a rectangular array of cells with colors determined by `C`. MATLAB creates a pseudocolor plot by using each set of four adjacent points in `C` to define a surface patch (i.e., cell).

`pcolor(C)` draws a pseudocolor plot. The elements of `C` are linearly mapped to an index into the current colormap. The mapping from `C` to the current colormap is defined by `colormap` and `caxis`.

`pcolor(X,Y,C)` draws a pseudocolor plot of the elements of `C` at the locations specified by `X` and `Y`. The plot is a logically rectangular, two-dimensional grid with vertices at the points `[X(i,j), Y(i,j)]`. `X` and `Y` are vectors or matrices that specify the spacing of the grid lines. If `X` and `Y` are vectors, `X` corresponds to the columns of `C` and `Y` corresponds to the rows. If `X` and `Y` are matrices, they must be the same size as `C`.

`h = pcolor(...)` returns a handle to a surface graphics object.

Remarks A pseudocolor plot is a flat surface plot viewed from above. `pcolor(X,Y,C)` is the same as viewing `surf(X,Y,0*Z,C)` using `view([0 90])`.

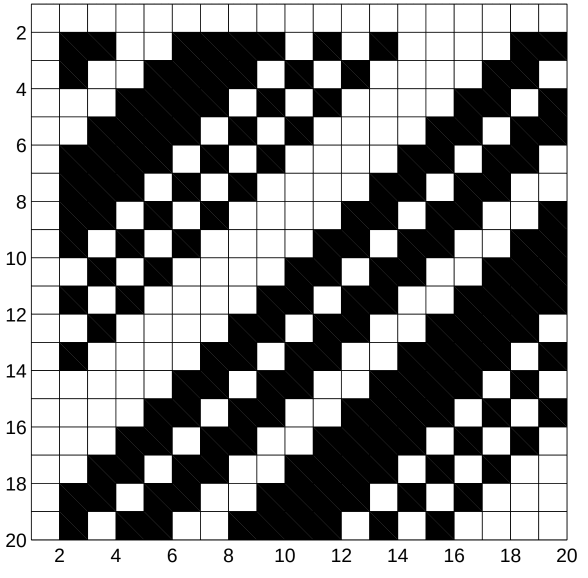
When you use `shading faceted` or `shading flat`, the constant color of each cell is the color associated with the corner having the smallest *x-y* coordinates. Therefore, `C(i,j)` determines the color of the cell in the *i*th row and *j*th column. The last row and column of `C` are not used.

When you use `shading interp`, each cell's color results from a bilinear interpolation of the colors at its four vertices and all elements of `C` are used.

Examples

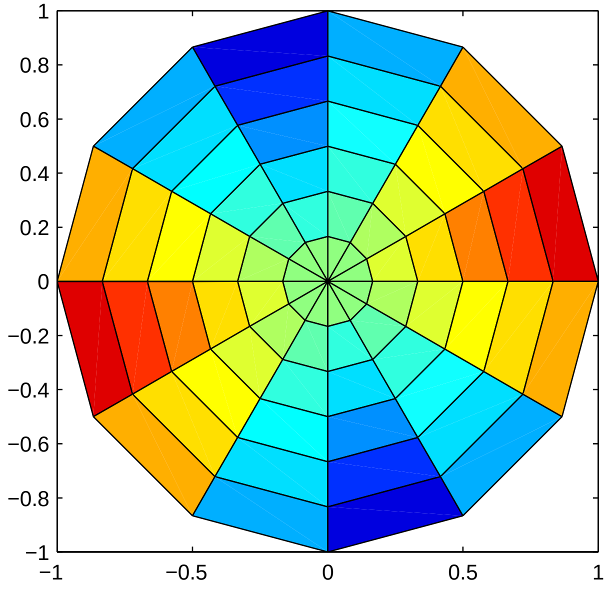
A Hadamard matrix has elements that are +1 and –1. A colormap with only two entries is appropriate when displaying a pseudocolor plot of this matrix.

```
pcolor(hadamard(20))
colormap(gray(2))
axis ij
axis square
```



A simple color wheel illustrates a polar coordinate system.

```
n = 6;
r = (0:n)'/n;
theta = pi*(-n:n)/n;
X = r*cos(theta);
Y = r*sin(theta);
C = r*cos(2*theta);
pcolor(X,Y,C)
axis equal tight
```



Algorithm

The number of vertex colors for `pcolor(C)` is the same as the number of cells for `image(C)`. `pcolor` differs from `image` in that `pcolor(C)` specifies the colors of vertices, which are scaled to fit the colormap; changing the axes `clim` property changes this color mapping. `image(C)` specifies the colors of cells and directly indexes into the colormap without scaling. Additionally, `pcolor(X,Y,C)` can produce parametric grids, which is not possible with `image`.

See Also

`caxis`, `image`, `mesh`, `shading`, `surf`, `view`

peaks

Purpose

A sample function of two variables.

Syntax

```
Z = peaks;
Z = peaks(n);
Z = peaks(V);
Z = peaks(X,Y);

peaks;
peaks(N);
peaks(V);
peaks(X,Y);

[X,Y,Z] = peaks;
[X,Y,Z] = peaks(n);
[X,Y,Z] = peaks(V);
```

Description

`peaks` is a function of two variables, obtained by translating and scaling Gaussian distributions, which is useful for demonstrating `mesh`, `surf`, `pcolor`, `contour`, and so on.

`Z = peaks;` returns a 49-by-49 matrix.

`Z = peaks(n);` returns an `n`-by-`n` matrix.

`Z = peaks(V);` returns an `n`-by-`n` matrix, where `n = length(V)`.

`Z = peaks(X,Y);` evaluates `peaks` at the given `X` and `Y` (which must be the same size) and returns a matrix the same size.

`peaks(...)` (with no output argument) plots the peaks function with `surf`.

`[X,Y,Z] = peaks(...);` returns two additional matrices, `X` and `Y`, for parametric plots, for example, `surf(X,Y,Z,del2(Z))`. If not given as input, the underlying matrices `X` and `Y` are:

```
[X,Y] = meshgrid(V,V)
```

where `V` is a given vector, or `V` is a vector of length `n` with elements equally spaced from –3 to 3. If no input argument is given, the default `n` is 49.

See Also

`meshgrid`, `surf`

Purpose

Pie chart

Syntax

```
pie(X)
pie(X,explode)
h = pie(...)
```

Description

`pie(X)` draws a pie chart using the data in `X`. Each element in `X` is represented as a slice in the pie chart.

`pie(X,explode)` offsets a slice from the pie. `explode` is a vector or matrix of zeros and nonzeros that correspond to `X`. A non-zero value offsets the corresponding slice from the center of the pie chart, so that `X(i,j)` is offset from the center if `explode(i,j)` is nonzero. `explode` must be the same size as `X`.

`h = pie(...)` returns a vector of handles to patch and text graphics objects.

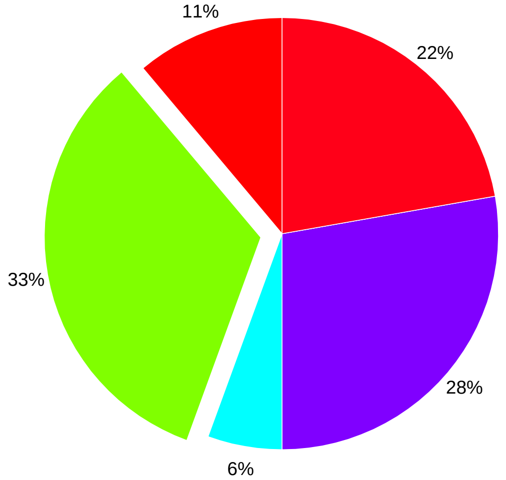
Remarks

The values in `X` are normalized via `X/sum(X)` to determine the area of each slice of the pie. If `sum(X)≤1`, the values in `X` directly specify the are of the pie slices. MATLAB draws only a partial pie if `sum(X)<1`.

Examples

Emphasize the second slice in the chart by setting its corresponding explode element to 1.

```
x = [1 3 0.5 2.5 2];
explode = [0 1 0 0 0];
pie(x,explode)
colormap jet
```



See Also

pie3

pie3

Purpose

Three-dimensional pie chart

Syntax

```
pie3(X)
pie3(X,explode)
h = pie3(...)
```

Description

`pie3(X)` draws a three-dimensional pie chart using the data in `X`. Each element in `X` is represented as a slice in the pie chart.

`pie3(X,explode)` specifies whether to offset a slice from the center of the pie chart. `X(i,j)` is offset from the center of the pie chart if `explode(i,j)` is nonzero. `explode` must be the same size as `X`.

`h = pie(...)` returns a vector of handles to patch, surface, and text graphics objects.

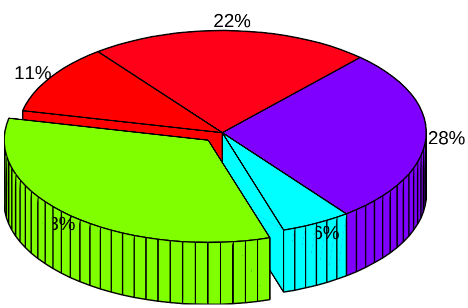
Remarks

The values in `X` are normalized via `X/sum(X)` to determine the area of each slice of the pie. If `sum(X)≤1`, the values in `X` directly specify the area of the pie slices. MATLAB draws only a partial pie if `sum(X)<1`.

Examples

Offset a slice in the pie chart by setting the corresponding `explode` element to `1`:

```
x = [1 3 0.5 2.5 2]
explode = [0 1 0 0 0]
pie3(x,explode)
colormap hsv
```



See Also

`pie`

Purpose Linear 2–D plot

Syntax

```
plot(Y)
plot(X1,Y1,...)
plot(X1,Y1,LineSpec,...)
plot(...,'PropertyName',PropertyValue,...)
h = plot(...)
```

Description

`plot(Y)` plots the columns of `Y` versus their index if `Y` is a real number. If `Y` is complex, `plot(Y)` is equivalent to `plot(real(Y),imag(Y))`. In all other uses of `plot`, the imaginary component is ignored.

`plot(X1,Y1,...)` plots all lines defined by `Xn` versus `Yn` pairs. If only `Xn` or `Yn` is a matrix, the vector is plotted versus the rows or columns of the matrix, depending on whether the vector's row or column dimension matches the matrix.

`plot(X1,Y1,LineSpec,...)` plots all lines defined by the `Xn,Yn,LineSpec` triples, where `LineSpec` is a line specification that determines line type, marker symbol, and color of the plotted lines. You can mix `Xn,Yn,LineSpec` triples with `Xn,Yn` pairs: `plot(X1,Y1,X2,Y2,LineSpec,X3,Y3)`.

`plot(...,'`*`PropertyName`*`',PropertyValue,...)` sets properties to the specified property values for all line graphics objects created by `plot`. (See the "Examples" section for examples.)

`h = plot(...)` returns a column vector of handles to line graphics objects, one handle per line.

Remarks

If you do not specify a color when plotting more than one line, `plot` automatically cycles through the colors in the order specified by the current axes `ColorOrder` property. After cycling through all the colors defined by `ColorOrder`, `plot` then cycles through the line styles defined in the axes `LineStyleOrder` property.

Note that, by default, MATLAB resets the `ColorOrder` and `LineStyleOrder` properties each time you call `plot`. If you want changes you make to these properties to persist, then you must define these changes as default values. For example,

```
set(0,'DefaultAxesColorOrder',[0 0 0],...
      'DefaultAxesLineStyleOrder','-|-.|--|:')
```

sets the default `ColorOrder` to use only the color black and sets the `LineStyleOrder` to use solid, dash-dot, dash-dash, and dotted line styles.

Additional Information

- See the “Creating 2-D Graphs” and “Labeling Graphs” in *Using MATLAB Graphics* for more information on plotting.
- See `LineSpec` for more information on specifying line styles and colors.

Examples

Specifying the Color and Size of Markers

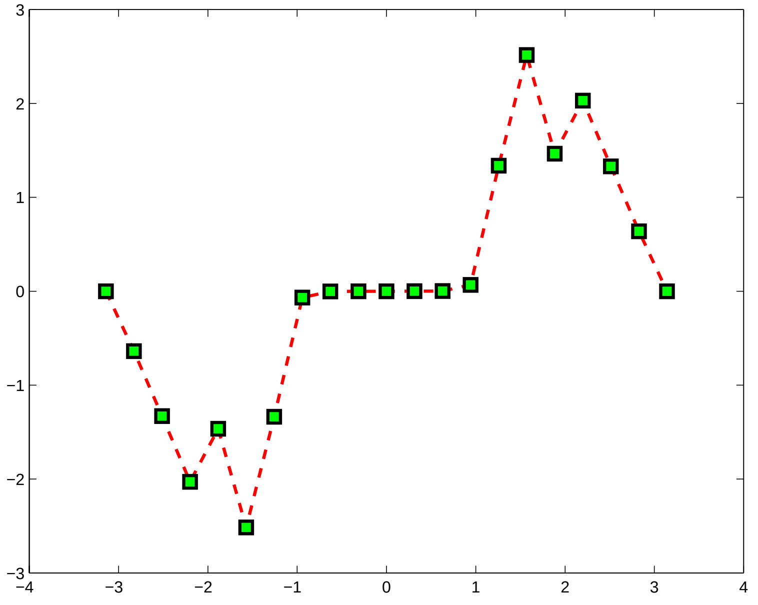
You can also specify other line characteristics using graphics properties (see `line` for a description of these properties):

- `LineWidth` – specifies the width (in points) of the line.
- `MarkerEdgeColor` – specifies the color of the marker or the edge color for filled markers (circle, square, diamond, pentagram, hexagram, and the four triangles).
- `MarkerFaceColor` – specifies the color of the face of filled markers.
- `MarkerSize` – specifies the size of the marker in units of points.

For example, these statements,

```
x = –pi:pi/10:pi;
y = tan(sin(x)) – sin(tan(x));
plot(x,y,'––rs','LineWidth',2,...
                'MarkerEdgeColor','k',...
                'MarkerFaceColor','g',...
                'MarkerSize',10)
```

produce this graph.

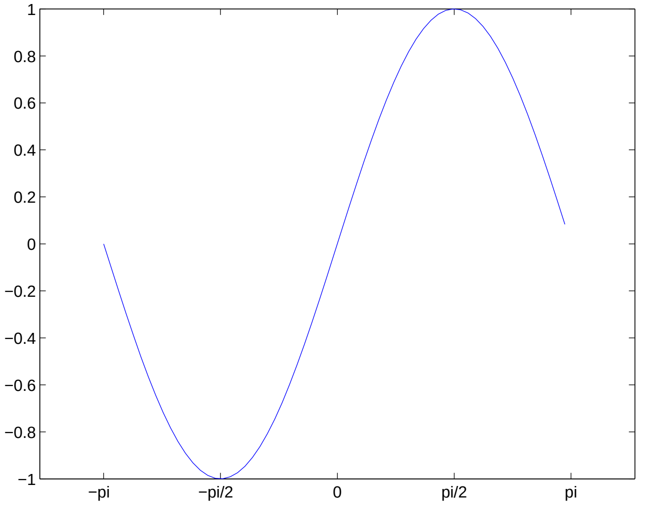


Specifying Tick Mark Location and Labeling

You can adjust the axis tick-mark locations and the labels appearing at each tick. For example, this plot of the sine function relabels the x-axis with more meaningful values,

```
x = −pi:.1:pi;
y = sin(x);
plot(x,y)
set(gca,'XTick',−pi:pi/2:pi)
set(gca,'XTickLabel',{'−pi','−pi/2','0','pi/2','pi'})
```

Now add axis labels and annotate the point −pi/4, sin(−pi/4).



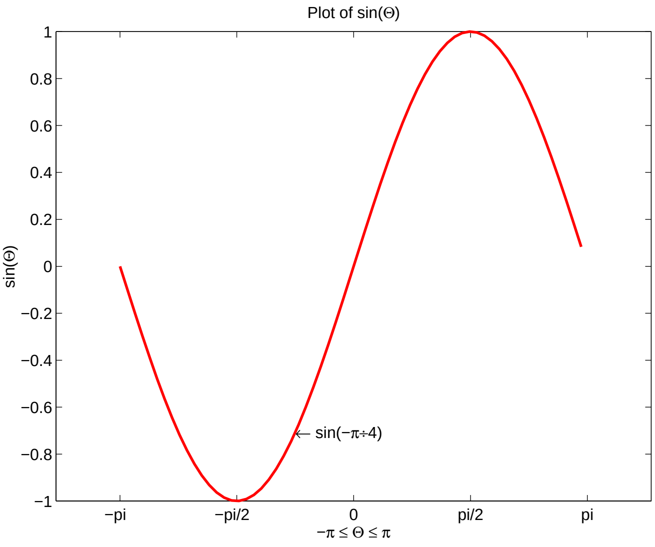
Adding Titles, Axis Labels, and Annotations

MATLAB enables you to add axis labels and titles. For example, using the graph from the previous example, add an x- and y-axis label,

```
xlabel('−\pi \leq \Theta \leq \pi')
ylabel('sin(\Theta)')
title('Plot of sin(\Theta)')
text(−pi/4,sin(−pi/4),'\leftarrow sin(−\pi\div4)',...
     'HorizontalAlignment','left')
```

Now change the line color to red by first finding the handle of the line object created by `plot` and then setting its `Color` property. In the same statement, set the `LineWidth` property to 2 points.

```
set(findobj(gca,'Type','line','Color',[0 0 1]),...
    'Color','red',...
    ‘LineWidth',2)
```



See Also

`axis`, `bar`, `grid`, `legend`, `line`, `LineSpec`, `loglog`, `plotyy`, `semilogx`, `semilogy`, `xlabel`, `xlim`, `ylabel`, `ylim`, `zlabel`, `zlim`

See the `text` `String` property for a list of symbols and how to display them.

See `plotedit` for information on using the plot annotation tools in the figure window toolbar.

plot3

Purpose

Linear 3-D plot

Syntax

```
plot3(X1,Y1,Z1,...)
plot3(X1,Y1,Z1,LineSpec,...)
plot3(...,'PropertyName',PropertyValue,...)
h = plot3(...)
```

Description

The `plot3` function displays a three-dimensional plot of a set of data points.

`plot3(X1,Y1,Z1,...)`, where `X1`, `Y1`, `Z1` are vectors or matrices, plots one or more lines in three-dimensional space through the points whose coordinates are the elements of `X1`, `Y1`, and `Z1`.

`plot3(X1,Y1,Z1,LineSpec,...)` creates and displays all lines defined by the `Xn,Yn,Zn,LineSpec` quads, where `LineSpec` is a line specification that determines line style, marker symbol, and color of the plotted lines.

`plot3(...,'`*`PropertyName`*`',PropertyValue,...)` sets properties to the specified property values for all Line graphics objects created by `plot3`.

`h = plot3(...)` returns a column vector of handles to line graphics objects, with one handle per line.

Remarks

If one or more of `X1`, `Y1`, `Z1` is a vector, the vectors are plotted versus the rows or columns of the matrix, depending whether the vectors' lengths equal the number of rows or the number of columns.

You can mix `Xn,Yn,Zn` triples with `Xn,Yn,Zn,LineSpec` quads, for example,

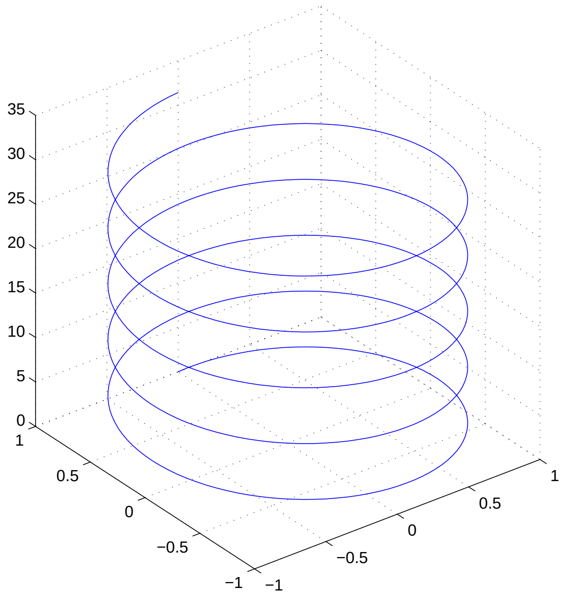
```
plot3(X1,Y1,Z1,X2,Y2,Z2,LineSpec,X3,Y3,Z3)
```

See `LineSpec` and `plot` for information on line types and markers.

Examples

Plot a three-dimensional helix.

```
t = 0:pi/50:10*pi;
plot3(sin(t),cos(t),t)
grid on
axis square
```



See Also `axis`, `bar3`, `grid`, `line`, `LineSpec`, `loglog`, `plot`, `semilogx`, `semilogy`

plotmatrix

Purpose Draw scatter plots

Syntax

```
plotmatrix(X,Y)
plotmatrix(...,'LineSpec')
[H,AX,BigAx,P] = plotmatrix(...)
```

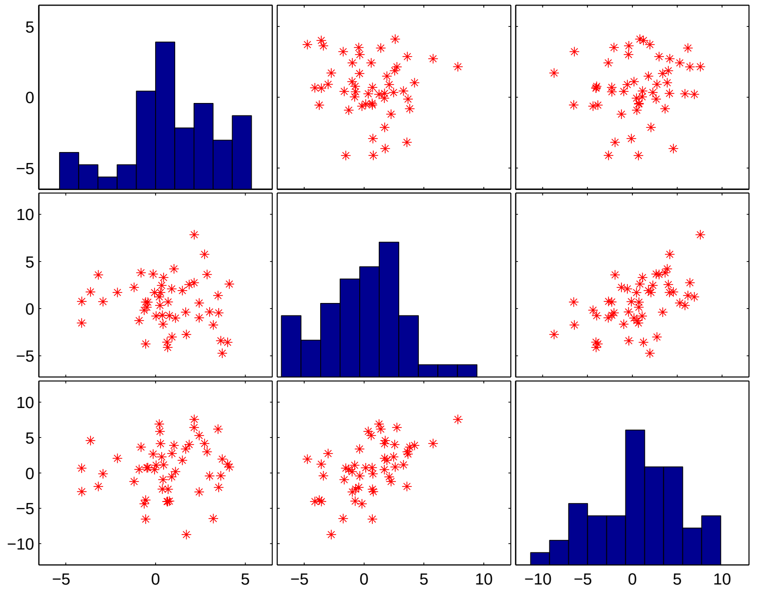
Description `plotmatrix(X,Y)` scatter plots the columns of `X` against the columns of `Y`. If `X` is *p*-by-*m* and `Y` is *p*-by-*n*, `plotmatrix` produces an *n*-by-*m* matrix of axes. `plotmatrix(Y)` is the same as `plotmatrix(Y,Y)` except that the diagonal is replaced by `hist(Y(:,i))`.

`plotmatrix(...,'LineSpec')` uses a `LineSpec` to create the scatter plot.The default is `'.'` .

`[H,AX,BigAx,P] = plotmatrix(...)` returns a matrix of handles to the objects created in `H`, a matrix of handles to the individual subaxes in `AX`, a handle to a big (invisible) axes that frames the subaxes in `BigAx`, and a matrix of handles for the histogram plots in `P`. `BigAx` is left as the current axes so that a subsequent `title`, `xlabel`, or `ylabel` commands are centered with respect to the matrix of axes.

Examples Generate plots of random data.

```
x = randn(50,3); y = x*[-1 2 1;2 0 1;1 -2 3;]';
plotmatrix(y,'*r')
```



See Also `scatter`, `scatter3`

Purpose Create graphs with y axes on both left and right side

Syntax

```
plotyy(X1,Y1,X2,Y2)
plotyy(X1,Y1,X2,Y2,'function')
plotyy(X1,Y1,X2,Y2,'function1','function2')
[AX,H1,H2] = plotyy(...)
```

Description

`plotyy(X1,Y1,X2,Y2)` plots `X1` versus `Y1` with y-axis labeling on the left and plots `X2` versus `Y2` with y-axis labeling on the right.

`plotyy(X1,Y1,X2,Y2,'function')` uses the plotting function specified by the string '`function`' instead of `plot` to produce each graph. '`function`' can be `plot`, `semilogx`, `semilogy`, `loglog`, `stem` or any MATLAB function that accepts the syntax:

```
h = function(x,y)
```

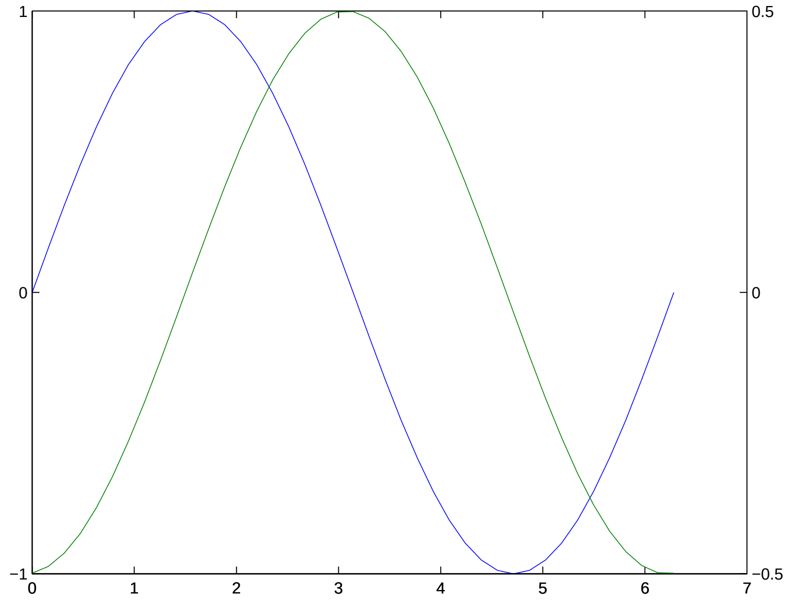
`plotyy(X1,Y1,X2,Y2,'function1','function2')` uses `function1(X1,Y1)` to plot the data for the left axis and `function1(X2,Y2)` to plot the data for the right axis.

`[AX,H1,H2] = plotyy(...)` returns the handles of the two axes created in `AX` and the handles of the graphics objects from each plot in `H1` and `H2`. `AX(1)` is the left axes and `AX(2)` is the right axes.

Examples

Create a graph using the plot function and two *y*-axes.

```
t = 0:pi/20:2*pi;
y1 = sin(t);
y2 = 0.5*sin(t-1.5);
plotyy(t,y1,t,y2,’plot’)
```



See Also `plot`, `loglog`, `semilogx`, `semilogy`, axes properties: `XAxisLocation`, `YAxisLocation`

The axes chapter in the *Using MATLAB Graphics* manual for information on multi-axis axes.

polar

Purpose

Plot polar coordinates

Syntax

```
polar(theta,rho)
polar(theta,rho,LineSpec)
```

Description

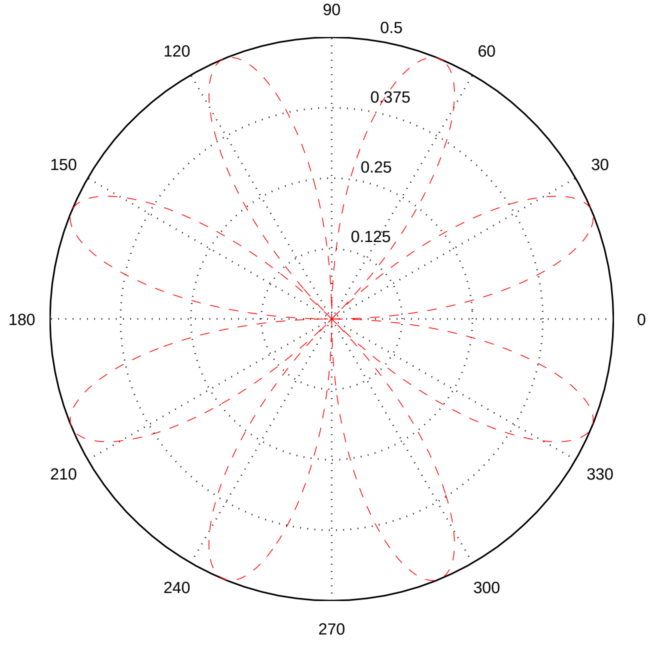
The `polar` function accepts polar coordinates, plots them in a Cartesian plane, and draws the polar grid on the plane.

`polar(theta,rho)` creates a polar coordinate plot of the angle `theta` versus the radius `rho`. `theta` is the angle from the *x*-axis to the radius vector specified in radians; `rho` is the length of the radius vector specified in dataspace units.

`polar(theta,rho,LineSpec)` `LineSpec` specifies the line type, plot symbol, and color for the lines drawn in the polar plot.

Examples

```
Create a simple polar plot using a dashed, red line:
t = 0:.01:2*pi;
polar(t,sin(2*t).*cos(2*t),'--r')
```



See Also cart2pol, compass, LineSpec, plot, pol2cart, rose

print, printopt

Purpose Create hardcopy output

Syntax

```
print
print -devicetype -options filename
[pcmd,dev] = printopt
```

Description

`print` and `printopt` produce hardcopy output. All arguments to the `print` command are optional. You can use them in any combination or order.

`print` sends the contents of the current figure, including bitmap representations of any user interface controls, to the printer using the device and system print command defined by `printopt`.

`print` *–devicetype* specifies a print device (an output format such as TIFF or PostScript or a print driver that controls what MATLAB sends to your printer), overriding the value returned by `printopt`. The Devices section lists all supported device types.

`print` *–options* specifies print options that modify the action of the `print` command. (For example, the `–noui` option suppresses printing of user interface controls.) The Options section lists available options.

`print` *filename* directs the output to the file designated by *filename*. If *filename* does not include an extension, `print` appends an appropriate extension, depending on the device (e.g., `.eps`). If you omit *filename*, `print` sends the file to the default output device (except for `–dmeta` and `–dbitmap`, which place their output on the clipboard).

`print(...)` the function form of print, with arguments in parentheses, enables you to pass variables for any input arguments. This form is useful passing file names and handles. See Batch Processing for an example.

`[pcmd,dev] = printopt` returns strings containing the current system-dependent print command and output device. `printopt` is an M-file used by `print` to produce the hardcopy output. You can edit the M-file `printopt.m` to set your default printer type and destination.

`pcmd` and `dev` are platform-dependent strings. `pcmd` contains the command that `print` uses to send a file to the printer. `dev` contains the device option for the `print` command. Their defaults are platform dependent.

Platform	pcmd	dev
UNIX	`lpr -r -s`	`-dps2`
VMS	`PRINT/DELETE`	`-dps2`
Windows	`COPY /B %s LPT1:`	`-dwin`

Devices

The table below lists device types supported by MATLAB's built-in drivers. Generally, Level 2 PostScript files are smaller and render more quickly when printing than Level 1 PostScript files. However, not all PostScript printers support Level 2, so determine the capabilities of your printer before using those drivers. Level 2 PostScript is the default for UNIX, and VAX/VMS. You can change this default by editing the `printopt.m` file.

The TIFF image format is supported on all platforms by almost all word processors for importing images. JPEG is a lossy, highly compressed format that is supported on all platforms for image processing and for inclusion into HTML documents on the World Wide Web. To create these formats, MATLAB renders the figure using the Z-buffer rendering method and the resulting pixmap is then saved to the specified file. You can specify the resolution of the image using the `-r` resolution switch.

Device	Description
`-dps`	Level 1 black and white PostScript
`-dpsc`	Level 1 color PostScript
`-dps2`	Level 2 black and white PostScript
`-dpsc2`	Level 2 color PostScript
`-deps`	Level 1 black and white Encapsulated PostScript (EPS)
`-depsc`	Level 1 color Encapsulated PostScript (EPS)

Device	Description
`-deps2`	Level 2 black and white Encapsulated PostScript (EPS)
`-depsc2`	Level 2 color Encapsulated PostScript (EPS)
`-dhpgl`	HPGL compatible with HP 7475A plotter
`-dill`	Adobe Illustrator 88 compatible illustration file
`-dtiff`	24-bit RGB TIFF with packbits compression (figures only)
`-dtiffn`	24-bit RGB TIFF with no compression (figures only)
`-djpeg`	Baseline JPEG image, quality factor defaults to 75 (figures only)
`-djpeg`*nn*	Baseline JPEG image with *nn* (0-100) quality factor (figures only)

This table lists additional devices supported via the Ghostscript post-processor, which converts PostScript files into other formats.

Device	Description
`-dlaserjet`	HP LaserJet
`-dljetplus`	HP LaserJet+
`-dljet2p`	HP LaserJet IIP
`-dljet3`	HP LaserJet III
`-ddeskjet`	HP DeskJet and DeskJet Plus
`-ddjet500`	HP Deskjet 500
`-dcdjmono`	HP DeskJet 500C printing black only
`-dcdjcolor`	HP DeskJet 500C with 24 bit/pixel color and high-quality color (Floyd-Steinberg) dithering
`-dcdj500`	HP DeskJet 500C

Device	Description
-dcdj550	HP Deskjet 550C (UNIX only)
-dpaintjet	HP PaintJet color printer
-dpjxl	HP PaintJet XL color printer
-dpjetxl	HP PaintJet XL color printer
-dpjxl300	HP PaintJet XL300 color printer
-ddnj650c	HP DesignJet 650C
-dbj10e	Canon BubbleJet BJ10e
-dbj200	Canon BubbleJet BJ200
-dbjc600	Canon Color BubbleJet BJC-600 and BJC-4000
-dln03	DEC LN03 printer
-depson	Epson-compatible dot matrix printers (9- or 24-pin)
-depsonc	Epson LQ-2550 and Fujitsu 3400/2400/1200
-deps9high	Epson-compatible 9-pin, interleaved lines (triple resolution)
-dibmpro	IBM 9-pin Proprinter
-dbmp256	8-bit (256-color) BMP file format
-dbmp16m	24-bit BMP file format
-dpcxmono	Monochrome PCX file format
-dpcx16	Older color PCX file format (EGA/VGA, 16-color)
-dpcx256	Newer color PCX file format (256-color)
-dpcx24b	24-bit color PCX file format, three 8-bit planes
-dpbm	Portable Bitmap (plain format)
-dpbmraw	Portable Bitmap (raw format)

Device	Description
`-dpgm`	Portable Graymap (plain format)
`-dpgmraw`	Portable Graymap (raw format)
`-dppm`	Portable Pixmap (plain format)
`-dppmraw`	Portable Pixmap (raw format)

This table summarizes additional devices available on Windows systems.

Device	Description
`-dwin`	Use Windows printing services (black and white)
`-dwinc`	Use Windows printing services (color)
`-dmeta`	Copy to clipboard in Enhanced Windows metafile format (color)
`-dbitmap`	Copy to clipboard in Windows bitmap (BMP) format (color)
`-dsetup`	Display **Print Setup** dialog box, but do not print
`-v`	Verbose mode to display **Print** dialog box (suppressed by default)

Options

This table summarizes printing options that you can specify when you enter the `print` command.

Option	Description
`-tiff`	Add color TIFF preview to Encapsulated PostScript
`-loose`	Use loose bounding box for EPS and PS devices
`-cmyk`	Use CMYK colors in PostScript instead of RGB
`-append`	Append to existing PostScript file without overwriting

Option	Description
`-r`*number*	Specify resolution in dots per inch
`-adobecset`	Use PostScript default character set encoding
`-P`*printer*	Specify printer to use (UNIX only)
`-f`*handle*	Handle of a figure graphics object to print (current figure by default; see `gcf`)
`-s`*windowtitle*	Name of SIMULINK system window to print (current system by default; see `gcs`)
`-painters`	Render using painter's algorithm
`-zbuffer`	Render using Z-buffer
`-noui`	Suppress printing of user interface controls

Paper Sizes

MATLAB supports a number of standard paper sizes. You can select from the following list by setting the `PaperType` property of the figure or selecting a supported paper size from the print dialog box.

Property Value	Size (Width-by-Height)
`usletter`	8.5-by-11 inches
`uslegal`	11-by-14 inches
`tabloid`	11-by-17 inches
`A0`	841-by-1189mm
`A1`	594-by-841mm
`A2`	420-by-594mm
`A3`	297-by-420mm
`A4`	210-by-297mm
`A5`	148-by-210mm

Property Value	Size (Width-by-Height)
B0	1029-by-1456mm
B1	728-by-1028mm
B2	514-by-728mm
B3	364-by-514mm
B4	257-by-364mm
B5	182-by-257mm
arch-A	9-by-12 inches
arch-B	12-by-18 inches
arch-C	18-by-24 inches
arch-D	24-by-36 inches
arch-E	36-by-48 inches
A	8.5-by-11 inches
B	11-by-17 inches
C	17-by-22 inches
D	22-by-34 inches
E	34-by-43 inches

Printing Tips

This section includes information about specific printing issues.

Figures with Resize Functions

The `print` command produces a warning when you print a figure having a callback routine defined for the figure `ResizeFcn`. To avoid the warning, set the figure `PaperPositionMode` property to `auto` or select **Match Figure Screen Size** in the **File->Page Setup...** dialog box.

Troubleshooting MS-Windows Printing

If you encounter problems such as segmentation violations , general protection faults, application errors, or the output does not appear as you expect when using MS-Windows printer drivers, try the following:

- If your printer is PostScript compatible, print with one of MATLAB's built-in PostScript drivers. There are various PostScript device options that you can use with the `print` command: they all start with `–dps`, or `–deps`.
- The behavior you are experiencing may occur only with certain versions of the print driver. Contact the print driver vendor for information on how to obtain and install a different driver. If you are using Windows 95, try installing the drivers that ship with the Windows 95 CD-ROM.
- Try printing with one of MATLAB's built-in GhostScript devices. These devices use GhostScript to convert PostScript files into other formats, such as HP LaserJet, PCX, Canon BubbleJet, and so on.
- Copy the figure as a Windows Metafile using the **Edit-->CopyFigure** menu item on the figure window menu or the `print –dmeta` option at the command line. You can then import the file into another application for printing.

  You can set copy options in the figure's **File-->Preferences...-->Copying Options** dialog box. The Windows Metafile clipboard format produces a better quality image than Windows Bitmap.

Printing Thick Lines on Windows95

Due to a limitation in Windows95, MATLAB is set up to print lines as either:

- Solid lines of the specified thickness (`LineWidth`)
- Thin (one pixel wide) lines with the specified line style (`LineStyle`)

If you create lines that are thicker than one pixel and use nonsolid line styles, MATLAB prints these lines with the specified line style, but one pixel wide (i.e., as thin lines).

However, you can change this behavior so that MATLAB prints thick, styled lines as thick, solid lines by editing your `matlab.ini` file, which is in your Windows directory. In this file, find the section,

```
[Matlab Settings]
```

and in this section change the assignment,

```
ThinLineStyles=1
```

to

```
ThinLineStyles=0
```

then restart MATLAB.

Printing MATLAB GUIs

You can generally obtain better results when printing a figure window that contains MATLAB `uicontrols` by setting these key properties:

- Set the figure `PaperPositionMode` property to `auto`. This ensures the printed version is the same size as the onscreen version. With `PaperPositionMode` set to `auto` MATLAB does not resize the figure to fit the current value of the `PaperPosition`. This is particularly important if you have specified a figure `ResizeFcn` because if MATLAB resizes the figure during the print operation, the `ResizeFcn` is automatically called.

  To set `PaperPositionMode` on the current figure, use the command:

  ```
  set(gcf,'PaperPositionMode','auto')
  ```

- Set the figure `InvertHardcopy` property to `off`. By default, MATLAB changes the figure background color of printed output to white, but does not change the color of uicontrols. If you have set the background color to, for example, match the gray of the GUI devices, you must set `InvertHardcopy` to `off` to preserve the color scheme.

  To set `InvertHardcopy` on the current figure, use the command:

  ```
  set(gcf,'InvertHardcopy','off')
  ```

- Use a color device if you want lines and text that are in color on the screen to be written to the output file as colored objects. Black and white devices convert colored lines and text to black or white to provide the best contrast with the background and to aviod dithering.
- Use the `print` command's `–loose` option to prevent MATLAB from using a bounding box that is tightly wrapped around objects contained in the figure. This is important if you have intentionally used space between uicontrols or axes and the edge of the figure and you want to maintain this appearance in the printed output.

Notes on Printing Interpolated Shading with PostScript Drivers

MATLAB can print surface objects (such as graphs created with `surf` or `mesh`) using interpolated colors. However, only patch objects that are composed of triangular faces can be printed using interpolated shading.

Printed output is always interpolated in RGB space, not in the colormap colors. This means, if you are using indexed color and interpolated face coloring, the printed output can look different from what is displayed on screen. See "Interpolating in Indexed vs. Truecolor" in the *Using MATLAB Graphics* manual for an illustration of each type of interpolation.

PostScript files generated for interpolated shading contain the color information of the graphics object's vertices and require the printer to perform the interpolation calculations. This can take an excessive amount of time and in some cases, printers may actually "time-out" before finishing the print job. One solution to this problem is to interpolate the data and generate a greater number of faces, which can then be flat shaded.

To ensure that the printed output matches what you see on the screen, print using the `-zbuffer` option. To obtain higher resolution (for example, to make text look better), use the `–r` option to increase the resolution. There is, however, a trade-off between the resolution and the size of the created PostScript file, which can be quite large at higher resolutions. The default resolution of 150 dpi generally produces good results. You can reduce the size of the output file by making the figure smaller before printing it and setting the figure `PaperPositionMode` to `auto`, or by just setting the `PaperPosition` property to a smaller size.

Note that in some UNIX environments, the default `lpr` command cannot print files larger than 1 Mbyte unless you use the `–s` option, which MATLAB does by default. See the `lpr` man page for more information.

Examples

This example prints a surface plot with interpolated shading. Setting the current figure's (`gcf`) `PaperPositionMode` to `auto` enables you to resize the figure window and print it at the size you see on the screen. See Options and the previous section for information on the `–zbuffer` and `–r200` options.

```
surf(peaks)
shading interp
set(gcf,'PaperPositionMode','auto')
print –dpsc2 –zbuffer –r200
```

Batch Processing

You can use the function form of print to pass variables containing file names. For example, this for loop creates a series of graphs and prints each one with a different file name.

```
for i=1:length(fnames)
    surf(Z(:,:,i))
    print('-dtiff','-r200',fnames(i))
end
```

Tiff Preview

The command:

```
print -depsc -tiff -r300 picture1
```

saves the current figure at 300 dpi, in a color Encapsulated PostScript file named `picture1.eps`. The `-tiff` option creates a 72 dpi TIFF preview, which many word processor applications can display on screen after you import the EPS file. This enables you to view the picture on screen within your word processor and print the document to a PostScript printer using a resolution of 300 dpi.

See Also

`orient`, `figure`

See the *Using MATLAB Graphics* manual for detailed information about printing in MATLAB.

Purpose Display print dialog box

Syntax

```
printdlg
printdlg(fig)
printdlg('-crossplatform',fig)
```

Description `printdlg` prints the current figure.

`printdlg(fig)` creates a dialog box from which you can print the figure window identified by the handle `fig`. Note that uimenus do not print.

`printdlg('-crossplatform',fig)` displays the standard cross-platform MATLAB printing dialog rather than the built-in printing dialog box for Microsoft Windows and Macintosh computers. Insert this option before the `fig` argument.

questdlg

Purpose Create and display question dialog box

Syntax

```
button = questdlg('qstring')
button = questdlg('qstring','title')
button = questdlg('qstring','title','default')
button = questdlg('qstring','title','str1','str2','default')
button =
  questdlg('qstring','title','str1','str2','str3','default')
```

Description

`button = questdlg('qstring')` displays a modal dialog presenting the question `'qstring'`. The dialog has three default buttons, **Yes**, **No**, and **Cancel**. `'qstring'` is a cell array or a string that automatically wraps to fit within the dialog box. `button` contains the name of the button pressed.

`button = questdlg('qstring','title')` displays a question dialog with '*title*' displayed in the dialog's title bar.

`button = questdlg('qstring','title','default')` specifies which push button is the default in the event that the **Return** key is pressed. '*default*' must be `'Yes'`, `'No'`, or `'Cancel'`.

`button = questdlg('qstring','title','str1','str2','default')` creates a question dialog box with two push buttons labeled `'str1'` and `'str2'`. `'default'` specifies the default button selection and must be `'str1'` or `'str2'`.

`button = questdlg('qstring','title','str1','str2','str3','default')` creates a question dialog box with three push buttons labeled `'str1'`, `'str2'`, and `'str3'`. `'default'` specifies the default button selection and must be `'str1'`, `'str2'`, or `'str3'`.

Example

Create a question dialog asking the user whether to continue a hypothetical operation:

```
button = questdlg('Do you want to continue?',...
'Continue Operation','Yes','No','Help','No');
if strcmp(button,'Yes')
   disp('Creating file')
elseif strcmp(button,'No')
   disp('Canceled file operation')
elseif strcmp(button,'Help')
   disp('Sorry, no help available')
end
```

See Also

dialog, errordlg, helpdlg, inputdlg, msgbox, warndlg

quiver

Purpose Quiver or velocity plot

Syntax

```
quiver(U,V)
quiver(X,Y,U,V)
quiver(...,scale)
quiver(...,LineSpec)
quiver(...,LineSpec,'filled')
h = quiver(...)
```

Description A quiver plot displays vectors with components (u,v) at the points (x,y).

`quiver(U,V)` draws vectors specified by `U` and `V` at the coordinates defined by `x = 1:n` and `y = 1:m`, where `[m,n] = size(U) = size(V)`. This syntax plots `U` and `V` over a geometrically rectangular grid. `quiver` automatically scales the vectors based on the distance between them to prevent them from overlapping.

`quiver(X,Y,U,V)` draws vectors at each pair of elements in `X` and `Y`. If `X` and `Y` are vectors, `length(X) = n` and `length(Y) = m`, where `[m,n] = size(U) = size(V)`. The vector `X` corresponds to the columns of `U` and `V`, and vector `Y` corresponds to the rows of `U` and `V`.

`quiver(...,scale)` automatically scales the vectors to prevent them from overlapping, then multiplies them by `scale`. `scale = 2` doubles their relative length and `scale = 0.5` halves them. Use `scale = 0` to plot the velocity vectors without the automatic scaling.

`quiver(...,LineSpec)` specifies line style, marker symbol, and color using any valid `LineSpec`. `quiver` draws the markers at the origin of the vectors.

`quiver(...,LineSpec,'filled')` fills markers specified by `LineSpec`.

`h = quiver(...)` returns a vector of line handles.

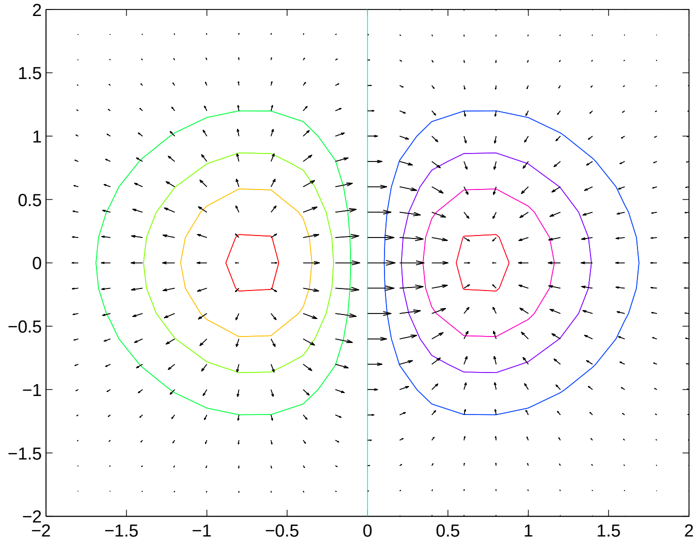
Remarks If `X` and `Y` are vectors, this function behaves as

```
[X,Y] = meshgrid(x,y)
quiver(X,Y,U,V)
```

Examples

Plot the gradient field of the function $z = xe^{(-x^2 - y^2)}$.

```
[X,Y] = meshgrid(-2:.2:2);
Z = X.*exp(-X.^2 - Y.^2);
[DX,DY] = gradient(Z,.2,.2);
contour(X,Y,Z)
hold on
quiver(X,Y,DX,DY)
colormap hsv
grid off
hold off
```



See Also

`contour`, `LineSpec`, `plot`, `quiver3`

quiver3

Purpose Three-dimensional velocity plot

Syntax

```
quiver3(Z,U,V,W)
quiver3(X,Y,Z,U,V,W)
quiver3(...,scale)
quiver3(...,LineSpec)
quiver3(...,LineSpec,'filled')
h = quiver3(...)
```

Description A three-dimensional quiver plot displays vectors with components (u,v,w) at the points (x,y,z).

`quiver3(Z,U,V,W)` plots the vectors at the equally spaced surface points specified by matrix `Z`. `quiver3` automatically scales the vectors based on the distance between them to prevent them from overlapping.

`quiver3(X,Y,Z,U,V,W)` plots vectors with components (u,v,w) at the points (x,y,z). The matrices `X`, `Y`, `Z`, `U`, `V`, `W` must all be the same size and contain the corresponding position and vector components.

`quiver3(...,scale)` automatically scales the vectors to prevent them from overlapping, then multiplies them by `scale`. `scale = 2` doubles their relative length and `scale = 0.5` halves them. Use `scale = 0` to plot the vectors without the automatic scaling.

`quiver3(...,LineSpec)` specify line type and color using any valid `LineSpec`.

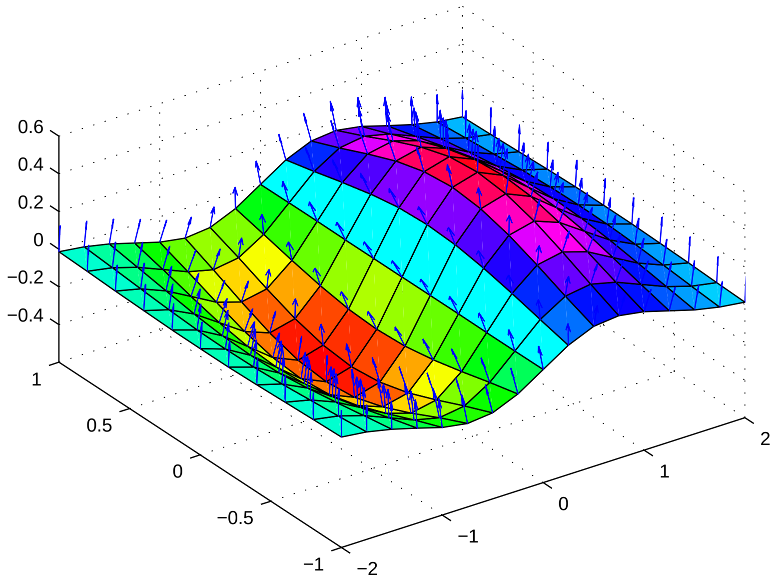
`quiver3(...,LineSpec,'filled')` fills markers specified by `LineSpec`.

`h = quiver3(...)` returns a vector of line handles.

Examples

Plot the surface normals of the function $z = xe^{(-x^2 - y^2)}$.

```
[X,Y] = meshgrid(-2:0.25:2,-1:0.2:1);
Z = X.* exp(-X.^2 - Y.^2);
[U,V,W] = surfnorm(X,Y,Z);
quiver3(X,Y,Z,U,V,W,0.5);
hold on
surf(X,Y,Z);
colormap hsv
view(-35,45)
axis ([-2 2 -1 1 -.6 .6])
hold off
```



See Also

`axis`, `contour`, `LineSpec`, `plot`, `plot3`, `quiver`, `surfnorm`, `view`

Purpose Create rubberband box for area selection

Synopsis

```
rbbox
rbbox(initialRect)
rbbox(initialRect,fixedPoint)
rbbox(initialRect,fixedPoint,stepSize)
finalRect = rbbox(...)
```

Description

`rbbox` initializes and tracks a rubberband box in the current figure. It sets the initial rectangular size of the box to 0, anchors the box at the figure's `CurrentPoint`, and begins tracking from this point.

`rbbox(initialRect)` specifies the initial location and size of the rubberband box as `[x y width height]`, where `x` and `y` define the lower-left corner, and `width` and `height` define the size. `initialRect` is in the units specified by the current figure's `Units` property, and measured from the lower-left corner of the figure window. The corner of the box closest to the pointer position follows the pointer until `rbbox` receives a button-up event.

`rbbox(initialRect,fixedPoint)` specifies the corner of the box that remains fixed. All arguments are in the units specified by the current figure's `Units` property, and measured from the lower-left corner of the figure window. `fixedPoint` is a two-element vector, `[x y]`. The tracking point is the corner diametrically opposite the anchored corner defined by `fixedPoint`.

`rbbox(initialRect,fixedPoint,stepSize)` specifies how frequently the rubberband box is updated. When the tracking point exceeds `stepSize` figure units, `rbbox` redraws the rubberband box. The default stepsize is `1`.

`finalRect = rbbox(...)` returns a four-element vector, `[x y width height]`, where `x` and `y` are the *x* and *y* components of the lower-left corner of the box, and `width` and `height` are the dimensions of the box.

Remarks

`rbbox` is useful for defining and resizing a rectangular region:

- For box definition, `initialRect` is `[x y 0 0]`, where `(x,y)` is the figure's `CurrentPoint`.
- For box resizing, `initialRect` defines the rectangular region that you resize (e.g., a legend). `fixedPoint` is the corner diametrically opposite the tracking point.

`rbbox` returns immediately if a button is not currently pressed. Therefore, you use `rbbox` with `waitforbuttonpress` so that the mouse button is down when `rbbox` is called. `rbbox` returns when you release the mouse button.

Examples

Assuming the current view is `view(2)`, use the current axes' `CurrentPoint` property to determine the extent of the rectangle in dataspace units:

```
k = waitforbuttonpress;

point1 = get(gca,'CurrentPoint');    % button down detected
finalRect = rbbox;                   % return figure units
point2 = get(gca,'CurrentPoint');    % button up detected

point1 = point1(1,1:2);              % extract x and y
point2 = point2(1,1:2);

p1 = min(point1,point2);             % calculate locations
offset = abs(point1-point2);         % and dimensions

x = [p1(1) p1(1)+offset(1) p1(1)+offset(1) p1(1) p1(1)];
y = [p1(2) p1(2) p1(2)+offset(2) p1(2)+offset(2) p1(2)];

hold on
axis manual
plot(x,y)                            % redraw in dataspace units
```

See Also

`axis`, `dragrect`, `waitforbuttonpress`

Purpose

Create a 2-D rectangle object

Syntax

```
rectangle
rectangle('Position',[x,y,w,h])
rectangle(...,'Curvature',[x,y])
h = rectangle(...)
```

Description

`rectangle` draws a rectangle with `Position` `[0,0,1,1]` and `Curvature` `[0,0]` (i.e., no curvature).

`rectangle('Position',[x,y,w,h])` draws the rectangle from the point x,y and having a width of `w` and a height of `h`. Specify values in axes data units.

Note that, to display a rectangle in the specified proportions, you need to set the axes data aspect ratio so that one unit is of equal length along both the x and y axes. You can do this with the command `axis equal` or `daspect([1,1,1])`.

`rectangle(...,'Curvature',[x,y])` specifies the curvature of the rectangle sides, enabling it to vary from a rectangle to an ellipse. The horizontal curvature `x` is the fraction of width of the rectangle that is curved along the top and bottom edges. The vertical curvature `y` is the fraction of the height of the rectangle that is curved along the left and right edges.

The values of `x` and `y` can range from `0` (no curvature) to `1` (maximum curvature). A value of `[0,0]` creates a rectangle with square sides. A value of `[1,1]` creates an ellipse. If you specify only one value for `Curvature`, then the same length (in axes data units) is curved along both horizontal and vertical sides. The amount of curvature is determined by the shorter dimension.

`h = rectangle(...)` returns the handle of the rectangle object created.

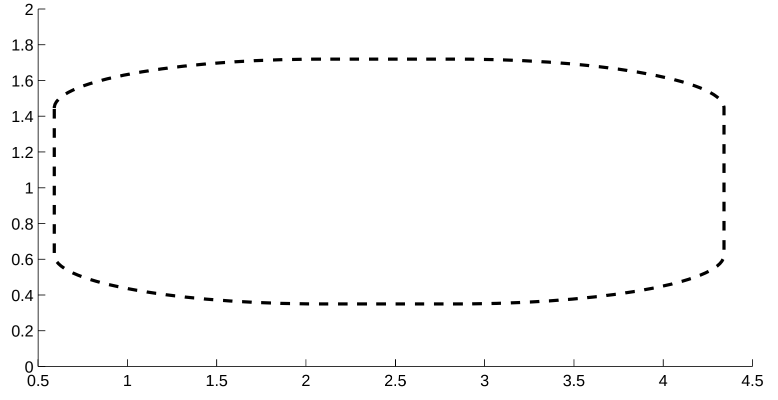
Remarks

Rectangle objects are 2-D and can be drawn in an axes only if the view is `[0 90]` (i.e., `view(2)`). Rectangles are children of axes and are defined in coordinates of the axes data.

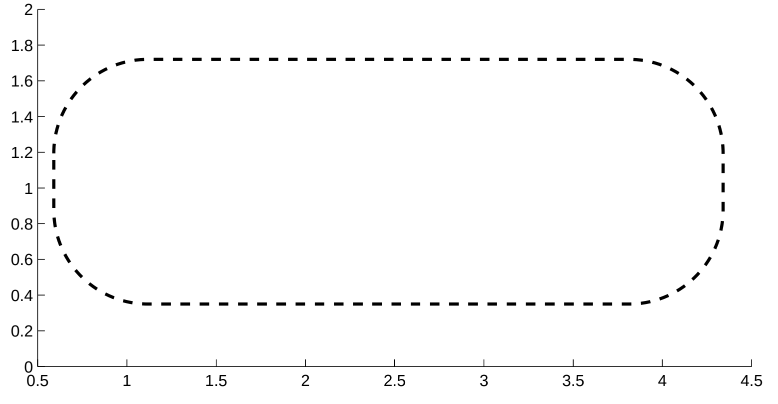
Examples

This example sets the data aspect ratio to `[1,1,1]` so that the rectangle displays in the specified proportions (`daspect`). Note that the horizontal and vertical curvature can be different. Also, note the effects of using a single value for Curvature.

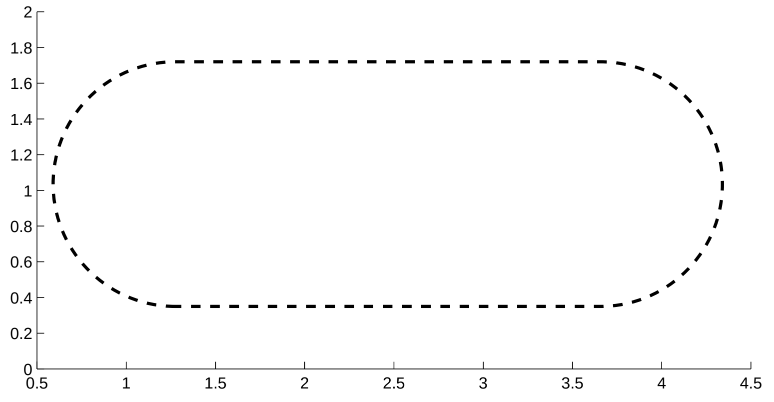
```
rectangle('Position',[0.59,0.35,3.75,1.37],...
          'Curvature',[0.8,0.4],...
         'LineWidth',2,'LineStyle','--')
daspect([1,1,1])
```



Specifying a single value of [0.4] for Curvature produces:

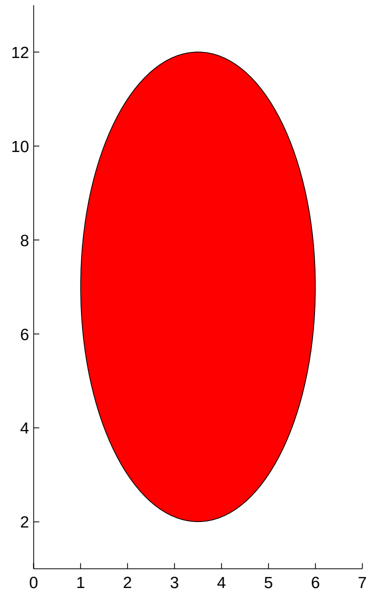


A Curvature of [1] produces a rectangle with the shortest side completely round:



This example creates an ellipse and colors the face red.

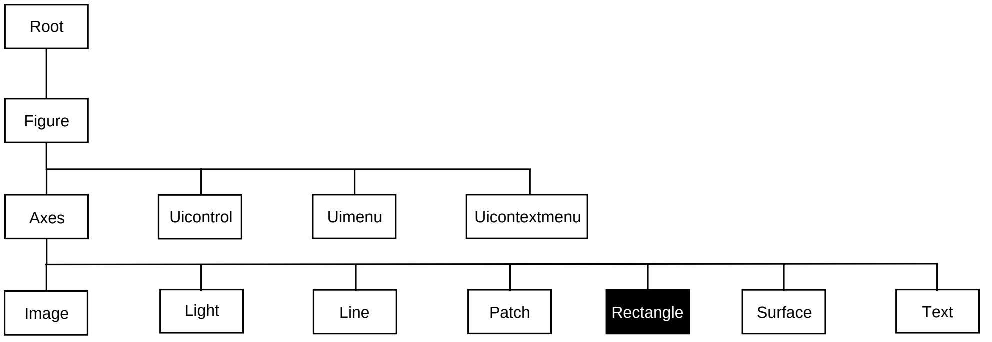
```
rectangle('Position',[1,2,5,10],'Curvature',[1,1],...
          'FaceColor','r')
daspect([1,1,1])
xlim([0,7])
ylim([1,13])
```



See Also

`line`, `patch`, `plot`, `plot3`, `set`, `text`, rectangle properties

Object Hierarchy



Setting Default Properties

You can set default rectangle properties on the axes, figure, and root levels.

```
set(0,'DefaultRectangleProperty',PropertyValue...)
set(gcf,'DefaultRectangleProperty',PropertyValue...)
set(gca,'DefaultRectangleProperty',PropertyValue...)
```

Where *Property* is the name of the rectangle property whose default value you want to set and `PropertyValue` is the value you are specifying. Use `set` and `get` to access the surface properties.

Property List

The following table lists all rectangle properties and provides a brief description of each. The property name links take you to an expanded description of the properties.

Property Name	Property Description	Property Value
Defining the Rectangle Object		
`Curvature`	Degree of horizontal and vertical curvature	Value: two-element vector with values between `0` and `1` Default: `[0,0]`
`EraseMode`	Method of drawing and erasing the rectangle (useful for animation)	Values: `normal`, `none`, `xor`, `background` Default: `normal`

Property Name	Property Description	Property Value
EdgeColor	Color of rectangle edges	Value: Colorspec or none Default: ColorSpec [0,0,0]
FaceColor	Color of rectangle interior	Value: Colorspec or none Default: none
LineStyle	Line style of edges	Values: –, ––, :, –., none Default: –
LineWidth	Width of edge lines in points	Value: scalar Default: 0.5 points
Position	Location and width and height of rectangle	Value: [x,y,width,height] Default: [0,0,1,1]
General Information About Rectangle Objects		
Children	Rectangle objects have no children	
Parent	Axes object	Value: handle of axes
Selected	Indicate if the rectangle is in a "selected" state.	Value: on, off Default: off
Tag	User-specified label	Value: any string Default: '' (empty string)
Type	The type of graphics object (read only)	Value: the string 'rectangle'
UserData	User-specified data	Value: any matrix Default: [] (empty matrix)
Properties Related to Callback Routine Execution		
BusyAction	Specify how to handle callback routine interruption	Value: cancel, queue Default: queue
ButtonDownFcn	Define a callback routine that executes when a mouse button is pressed on over the rectangle	Value: string Default: '' (empty string)

Property Name	Property Description	Property Value
CreateFcn	Define a callback routine that executes when a rectangle is created	Value: string Default: '' (empty string)
DeleteFcn	Define a callback routine that executes when the rectangle is deleted (via close or delete)	Values: string Default: '' (empty string)
Interruptible	Determine if callback routine can be interrupted	Values: on, off Default: on (can be interrupted)
UIContextMenu	Associate a context menu with the rectangle	Values: handle of a Uicontextmenu
Controlling Access to Objects		
HandleVisibility	Determines if and when the rectangle's handle is visible to other functions	Values: on, callback, off Default: on
HitTest	Determines if the rectangle can become the current object (see the Figure CurrentObject property)	Values: on, off Default: on
Controlling the Appearance		
Clipping	Clipping to axes rectangle	Values: on, off Default: on
SelectionHighlight	Highlight rectangle when selected (Selected property set to on)	Values: on, off Default: on
Visible	Make the rectangle visible or invisible	Values: on, off Default: on

Rectangle Properties

This section lists property names along with the type of values each accepts. Curly braces { } enclose default values.

BusyAction `cancel | {queue}`

Callback routine interruption. The `BusyAction` property enables you to control how MATLAB handles events that potentially interrupt executing callback routines. If there is a callback routine executing, subsequently invoked callback routes always attempt to interrupt it. If the `Interruptible` property of the object whose callback is executing is set to `on` (the default), then interruption occurs at the next point where the event queue is processed. If the `Interruptible` property is `off`, the `BusyAction` property (of the object owning the executing callback) determines how MATLAB handles the event. The choices are:

- `cancel` – discard the event that attempted to execute a second callback routine.
- `queue` – queue the event that attempted to execute a second callback routine until the current callback finishes.

ButtonDownFcn string

Button press callback routine. A callback routine that executes whenever you press a mouse button while the pointer is over the rectangle object. Define this routine as a string that is a valid MATLAB expression or the name of an M-file. The expression executes in the MATLAB workspace.

Children vector of handles

The empty matrix; rectangle objects have no children.

Clipping `{on} | off`

Clipping mode. MATLAB clips rectangles to the axes plot box by default. If you set `Clipping` to `off`, rectangles display outside the axes plot box. This can occur if you create a rectangle, set `hold` to `on`, freeze axis scaling (`axis manual`), and then create a larger rectangle.

CreateFcn string

Callback routine executed during object creation. This property defines a callback routine that executes when MATLAB creates a rectangle object. You must define this property as a default value for rectangles. For example, the statement,

```
set(0,'DefaultRectangleCreateFcn',...
      'set(gca,''DataAspectRatio'',[1,1,1])')
```

defines a default value on the root level that sets the axes `DataAspectRatio` whenever you create a rectangle object. MATLAB executes this routine after setting all rectangle properties. Setting this property on an existing rectangle object has no effect.

The handle of the object whose `CreateFcn` is being executed is accessible only through the root `CallbackObject` property, which you can query using `gcbo`.

Curvature one- or two-element vector `[x,y]`

Amount of horizontal and vertical curvature. This property specifies the curvature of the property sides, which enables the shape of the rectangle to vary from rectangular to ellipsoidal. The horizontal curvature `x` is the fraction of width of the rectangle that is curved along the top and bottom edges. The vertical curvature `y` is the fraction of the height of the rectangle that is curved along the left and right edges.

The values of `x` and `y` can range from `0` (no curvature) to `1` (maximum curvature). A value of `[0,0]` creates a rectangle with square sides. A value of `[1,1]` creates an ellipse. If you specify only one value for `Curvature`, then the same length (in axes data units) is curved along both horizontal and vertical sides. The amount of curvature is determined by the shorter dimension.

DeleteFcn string

Delete rectangle callback routine. A callback routine that executes when you delete the rectangle object (e.g., when you issue a `delete` command or clear the axes or figure). MATLAB executes the routine before deleting the object's properties so these values are available to the callback routine.

The handle of the object whose `DeleteFcn` is being executed is accessible only through the root `CallbackObject` property, which you can query using `gcbo`.

EdgeColor `{Colorspec} | none`

Color of the rectangle edges. This property specifies the color of the rectangle edges as a color or specifies that no edges be drawn.

EraseMode `{normal} | none | xor | background`

Erase mode. This property controls the technique MATLAB uses to draw and erase rectangle objects. Alternative erase modes are useful for creating animated sequences, where control of the way individual objects redraw is necessary to improve performance and obtain the desired effect.

- `normal` (the default) – Redraw the affected region of the display, performing the three-dimensional analysis necessary to ensure that all objects are rendered correctly. This mode produces the most accurate picture, but is the slowest. The other modes are faster, but do not perform a complete redraw and are therefore less accurate.
- `none` – Do not erase the rectangle when it is moved or destroyed. While the object is still visible on the screen after erasing with `EraseMode` `none`, you cannot print it because MATLAB stores no information about its former location.
- `xor` – Draw and erase the rectangle by performing an exclusive OR (XOR) with the color of the screen beneath it. This mode does not damage the color of the objects beneath the rectangle. However, the rectangle's color depends on the color of whatever is beneath it on the display.
- `background` – Erase the rectangle by drawing it in the Axes' background `Color`, or the Figure background `Color` if the Axes `Color` is set to `none`. This damages objects that are behind the erased rectangle, but rectangles are always properly colored.

Printing with Non-normal Erase Modes.

MATLAB always prints Figures as if the `EraseMode` of all objects is `normal`. This means graphics objects created with `EraseMode` set to `none`, `xor`, or `background` can look different on screen than on paper. On screen, MATLAB may mathematically combine layers of colors (e.g., XORing a pixel color with that of the pixel behind it) and ignore three-dimensional sorting to obtain greater rendering speed. However, these techniques are not applied to the printed output.

You can use the MATLAB `getframe` command or other screen capture application to create an image of a Figure containing non-normal mode objects.

FaceColor ColorSpec | {none}

Color of rectangle face. This property specifies the color of the rectangle face, which is not colored by default.

HandleVisibility {on} | callback | off

Control access to object's handle by command-line users and GUIs. This property determines when an object's handle is visible in its parent's list of children. HandleVisibility is useful for preventing command-line users from accidentally drawing into or deleting a figure that contains only user interface devices (such as a dialog box).

Handles are always visible when HandleVisibility is on.

Setting HandleVisibility to callback causes handles to be visible from within callback routines or functions invoked by callback routines, but not from within functions invoked from the command line. This provides a means to protect GUIs from command-line users, while allowing callback routines to have complete access to object handles.

Setting HandleVisibility to off makes handles invisible at all times. This may be necessary when a callback routine invokes a function that might potentially damage the GUI (such as evaling a user-typed string), and so temporarily hides its own handles during the execution of that function.

When a handle is not visible in its parent's list of children, it cannot be returned by functions that obtain handles by searching the object hierarchy or querying handle properties. This includes get, findobj, gca, gcf, gco, newplot, cla, clf, and close.

When a handle's visibility is restricted using callback or off, the object's handle does not appear in its parent's Children property, figures do not appear in the root's CurrentFigure property, objects do not appear in the root's CallbackObject property or in the figure's CurrentObject property, and Axes do not appear in their parent's CurrentAxes property.

You can set the Root ShowHiddenHandles property to on to make all handles visible, regardless of their HandleVisibility settings (this does not affect the values of the HandleVisibility properties).

Handles that are hidden are still valid. If you know an object's handle, you can set and get its properties, and pass it to any function that operates on handles.

HitTest `{on} | off`

Selectable by mouse click. `HitTest` determines if the rectangle can become the current object (as returned by the `gco` command and the figure `CurrentObject` property) as a result of a mouse click on the rectangle. If `HiTest` is `off`, clicking on the rectangle selects the object below it (which may be the axes containing it).

Interruptible `{on} | off`

Callback routine interruption mode. The `Interruptible` property controls whether a rectangle callback routine can be interrupted by subsequently invoked callback routines. Only callback routines defined for the `ButtonDownFcn` are affected by the `Interruptible` property. MATLAB checks for events that can interrupt a callback routine only when it encounters a `drawnow`, `figure`, `getframe`, or `pause` command in the routine.

LineStyle `{-} | -- | : | -. | none`

Line style. This property specifies the line style of the edges. The available line styles are:

Symbol	Line Style
`-`	solid line (default)
`--`	dashed line
`:`	dotted line
`-.`	dash-dot line
`none`	no line

LineWidth scalar

The width of the rectangle object. Specify this value in points (1 point = $^1/_{72}$ inch). The default `LineWidth` is 0.5 points.

Parent handle

rectangle's parent. The handle of the rectangle object's parent axes. You can move a rectangle object to another axes by changing this property to the new axes handle.

Position four-element vecotr [x,y,width,height]

Location and size of rectangle. This property specifies the location and size of the rectangle in the data units of the axes. The point defined by x, y specifies one corner of the rectangle, and width and height define the size in units along the x and y axes respecitvely.

Selected on | off

Is object selected? When this property is on MATLAB displays selection handles if the SelectionHighlight property is also on. You can, for example, define the ButtonDownFcn to set this property, allowing users to select the object with the mouse.

SelectionHighlight {on} | off

Objects highlight when selected. When the Selected property is on, MATLAB indicates the selected state by drawing handles at each vertex. When SelectionHighlight is off, MATLAB does not draw the handles.

Tag string

User-specified object label. The Tag property provides a means to identify graphics objects with a user-specified label. This is particularly useful when constructing interactive graphics programs that would otherwise need to define object handles as global variables or pass them as arguments between callback routines. You can define Tag as any string.

Type string (read only)

Class of graphics object. For rectangle objects, Type is always the string 'rectangle'.

UIContextMenu handle of a uicontextmenu object

Associate a context menu with the rectangle. Assign this property the handle of a uicontextmenu object created in the same figure as the rectangle. Use the uicontextmenu function to create the context menu. MATLAB displays the context menu whenever you right-click over the rectangle.

UserData matrix

User-specified data. Any data you want to associate with the rectangle object. MATLAB does not use this data, but you can access it using the set and get commands.

Visible {on} | off

rectangle visibility. By default, all rectangles are visible. When set to off, the rectangle is not visible, but still exists and you can get and set its properties.

reducepatch

Purpose Reduce the number of patch faces

Syntax

```
reducepatch(p,r)
nfv = reducepatch(p,r)
nfv = reducepatch(fv,r)
nfv = reducepatch(f,v,r)
[nf,nv] = reducepatch(...)
```

Description

`reducepatch(p,r)` reduces the number of faces of the patch identified by handle `p`, while attempting to preserve the overall shape of the original object. MATLAB interprets the reduction factor `r` in one of two ways depending on its value:

- If `r` is less than 1, `r` is interpreted as a fraction of the original number of faces. For example, if you specify `r` as 0.2, then the number of faces is reduced to 20% of the number in the original patch.
- If `r` is greater than or equal to 1, then `r` is the target number of faces. For example, if you specify `r` as 400, then the number of faces is reduced until there are 400 faces remaining.

`nfv = reducepatch(p,r)` returns the reduced set of faces and vertices but does not set the `Faces` and `Vertices` properties of patch `p`. The struct `nfv` contains the faces and vertices after reduction.

`nfv = reducepatch(fv,r)` performs the reduction on the faces and vertices in the struct `fv`.

`nfv = reducepatch(p)` or `nfv = reducepatch(fv)` uses a reduction value of `0.5`.

`nfv = reducepatch(f,v,r)` performs the reduction on the faces in `f` and the vertices in `v`.

`[nf,nv] = reducepatch(...)` returns the faces and vertices in the arrays `nf` and `nv`.

Remarks

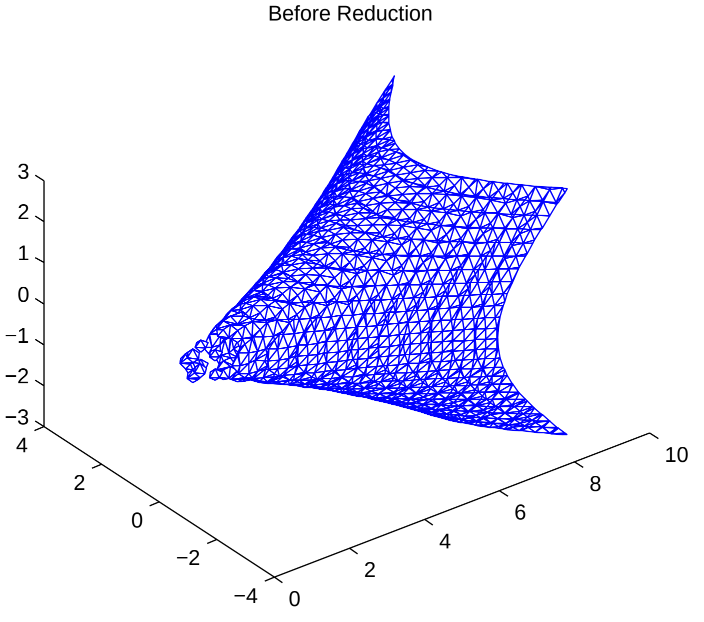
If the patch contains nonshared vertices, MATLAB computes shared vertices before reducing the number of faces. If the faces of the patch are not triangles, MATLAB triangulates the faces before reduction. The faces returned are always defined as triangles.

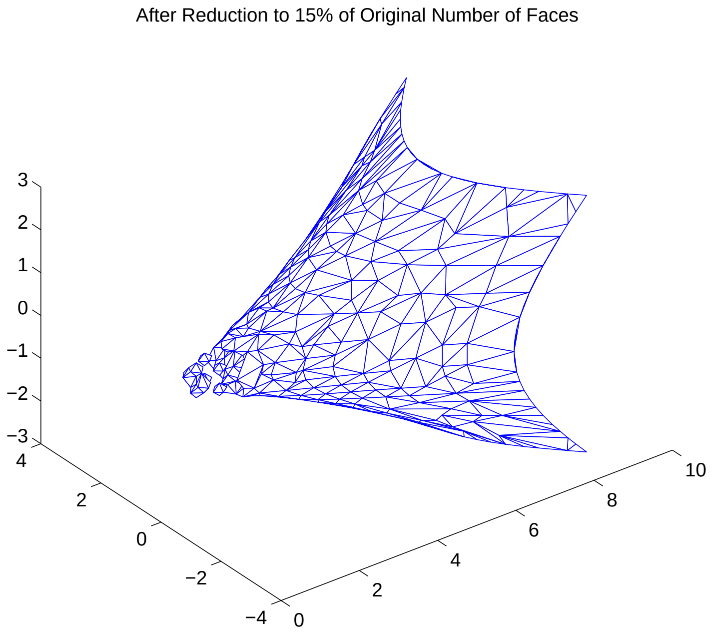
The number of output triangles may not be exactly the number specified with the reduction factor argument (r), particularly if the faces of the original patch are not triangles.

Examples

This example illustrates the effect of reducing the number of faces to only 15% of the original value.

```
[x,y,z,v] = flow;
p = patch(isosurface(x,y,z,v,-3));
set(p,'facecolor','w','EdgeColor','b');
daspect([1,1,1])
view(3)
figure;
h = axes;
p2 = copyobj(p,h);
reducepatch(p2,0.15)
daspect([1,1,1])
view(3)
```





See Also `isosurface`, `isocaps`, `isonormals`, `smooth3`, `subvolume`, `reducevolume`

Purpose Reduce the number of elements in a volume data set

Syntax

```
[nx,ny,nz,nv] = reducevolume(X,Y,Z,V,[Rx,Ry,Rz])
[nx,ny,nz,nv] = reducevolume(V,[Rx,Ry,Rz])
nv = reducevolume(...)
```

Description

`[nx,ny,nz,nv] = reducevolume(X,Y,Z,V,[Rx,Ry,Rz])` reduces the number of elements in the volume by retaining every `Rx`th element in the x direction, every `Ry`th element in the y direction, and every `Rz`th element in the z direction. If a scalar `R` is used to indicate the amount or reduction instead of a 3-element vector, MATLAB assumes the reduction to be `[R R R]`.

The arrays `X`, `Y`, and `Z` define the coordinates for the volume `V`. The reduced volume is returned in `nv` and the coordinates of the reduced volume are returned in `nx`, `ny`, and `nz`.

`[nx,ny,nz,nv] = reducevolume(V,[Rx,Ry,Rz])` assumes the arrays `X`, `Y`, and `Z` are defined as `[X,Y,Z] = meshgrid(1:n,1:m,1:p)` where `[m,n,p] = size(V)`.

`nv = reducevolume(...)` returns only the reduced volume.

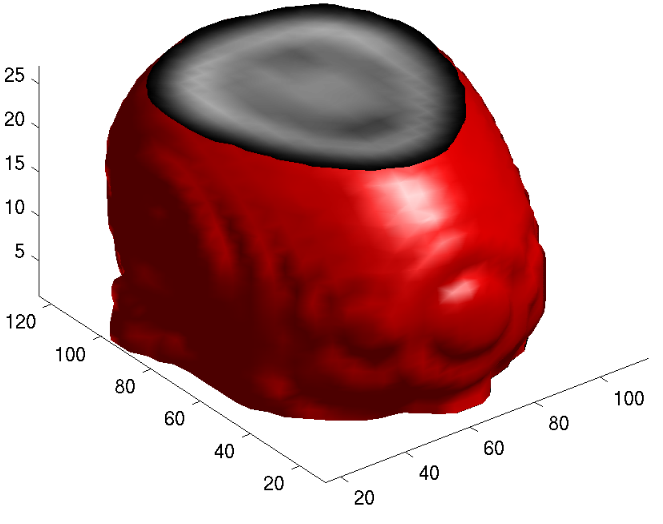
Examples

This example uses a data set that is a collection of MRI slices of a human skull. This data is processed in a variety of ways:

- The 4-D array is squeezed (`squeeze`) into three dimensions and then reduced (`reducevolume`) so that what remains is every 4th element in the x and y directions and every element in the z direction.
- The reduced data is smoothed (`smooth3`).
- The outline of the skull is an isosurface generated as a patch (`p1`) whose vertex normals are recalculated to improve the appearance when lighting is applied (`patch`, `isosurface`, `isonormals`).
- A second patch (`p2`) with an interpolated face color draws the end caps (`FaceColor`, `isocaps`).
- The view of the object is set (`view`, `axis`, `daspect`).

- A 100-element grayscale colormap provides coloring for the end caps (colormap).
- Adding a light to the right of the camera illuminates the object (camlight, lighting).

```
load mri
D = squeeze(D);
[x,y,z,D] = reducevolume(D,[4,4,1]);
D = smooth3(D);
p1 = patch(isosurface(x,y,z,D, 5,'verbose'),...
    'FaceColor','red','EdgeColor','none');
isonormals(x,y,z,D,p1);

p2 = patch(isocaps(x,y,z,D, 5),...
    'FaceColor','interp','EdgeColor','none');
view(3); axis tight; daspect([1,1,.4])
colormap(gray(100))
camlight; lighting gouraud
```



See Also `isosurface`, `isocaps`, `isonormals`, `smooth3`, `subvolume`, `reducepatch`

refresh

Purpose Redraw current figure

Syntax

```
refresh
refresh(h)
```

Description `refresh` erases and redraws the current figure.

`refresh(h)` redraws the figure identified by `h`.

Purpose

Reset graphics object properties to their defaults

Syntax

```
reset(h)
```

Description

`reset(h)` resets all properties having factory defaults on the object identified by `h`. To see the list of factory defaults, use the statement,

```
get(0,'factory')
```

If `h` is a figure, MATLAB does not reset `Position`, `Units`, `PaperPosition`, and `PaperUnits`. If `h` is an axes, MATLAB does not reset `Position` and `Units`.

Examples

`reset(gca)` resets the properties of the current axes.

`reset(gcf)` resets the properties of the current figure.

See Also

`cla`, `clf`, `gca`, `gcf`, `hold`

rgb2hsv

Purpose Convert RGB colormap to HSV colormap

Syntax

```
cmap = rgb2hsv(M)
```

Description

`cmap = rgb2hsv(M)` converts an RGB colormap, `M`, to an HSV colormap, `cmap`. Both colormaps are *m*-by-3 matrices. The elements of both colormaps are in the range 0 to 1.

The columns of the input matrix, `M`, represent intensities of red, green, and blue, respectively. The columns of the output matrix, `cmap`, represent hue, saturation, and value, respectively.

See Also `brighten`, `colormap`, `hsv2rgb`,`rgbplot`

Purpose

Plot colormap

Syntax

```
rgbplot(cmap)
```

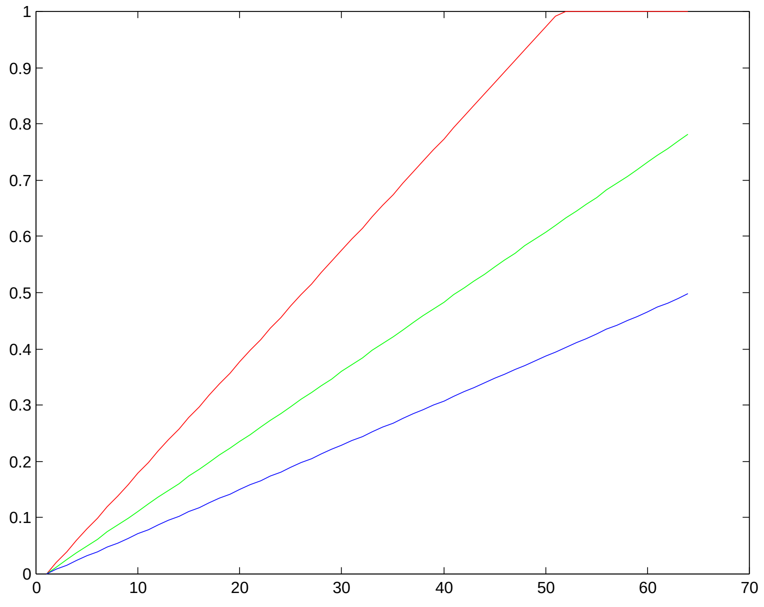
Description

`rgbplot(cmap)` plots the three columns of `cmap`, where `cmap` is an *m*-by-3 colormap matrix. `rgbplot` draws the first column in red, the second in green, and the third in blue.

Examples

Plot the RGB values of the `copper` colormap.

```
rgbplot(copper)
```



See Also

`colormap`

ribbon

Purpose Ribbon plot

Syntax

```
ribbon(Y)
ribbon(X,Y)
ribbon(X,Y,width)
h = ribbon(...)
```

Description

`ribbon(Y)` plots the columns of `Y` as separate three-dimensional ribbons using `X = 1:size(Y,1)`.

`ribbon(X,Y)` plots `X` versus the columns of `Y` as three-dimensional strips. `X` and `Y` are vectors of the same size or matrices of the same size. Additionally, `X` can be a row or a column vector, and `Y` a matrix with `length(X)` rows.

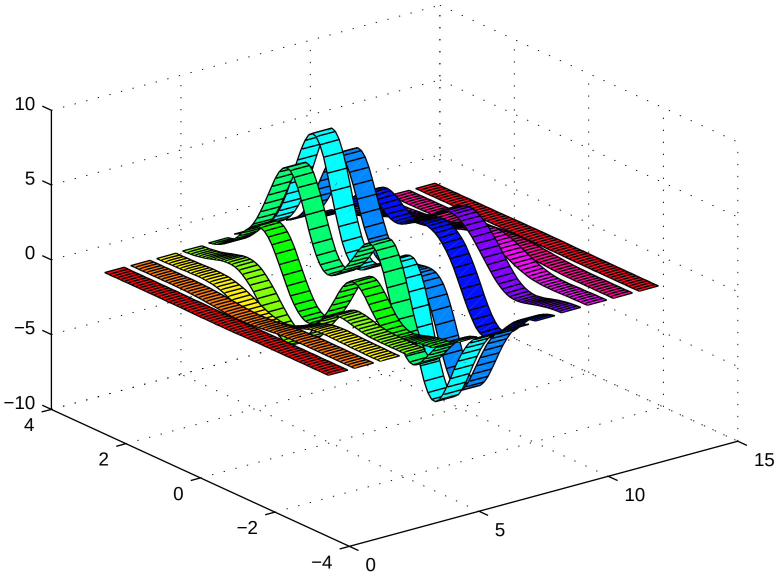
`ribbon(X,Y,width)` specifies the width of the ribbons. The default is `0.75`.

`h = ribbon(...)` returns a vector of handles to surface graphics objects. `ribbon` returns one handle per strip.

Examples

Create a ribbon plot of the peaks function.

```
[x,y] = meshgrid(-3:.5:3,-3:.1:3);
z = peaks(x,y);
ribbon(y,z)
colormap hsv
```



See Also

`plot`, `plot3`, `surface`

Purpose

Root object properties

Description

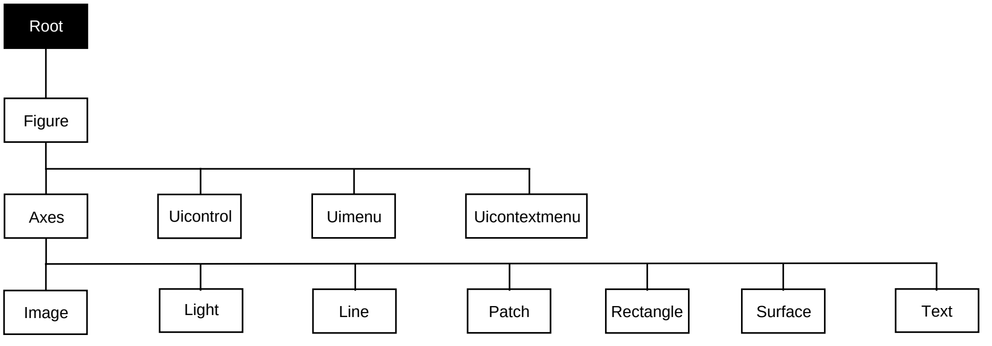
The root is a graphics object that corresponds to the computer screen. There is only one root object and it has no parent. The children of the root object are figures.

The root object exists when you start MATLAB; you never have to create it and you cannot destroy it. Use `set` and `get` to access the root properties.

See Also

`diary`, `echo`, `figure`, `format`, `gcf`, `get`, `set`

Object Hierarchy



Property List

The following table lists all root properties and provides a brief description of each. The property name links take you to an expanded description of the properties. This table does not include properties that are defined for, but not used by, the root object.

Property Name	Property Description	Property Value
Information about MATLAB's state		
`CallbackObject`	Handle of object whose callback is executing	Values: object handle
`CurrentFigure`	Handle of current figure	Values: object handle

Property Name	Property Description	Property Value
ErrorMessage	Text of last error message	Value: character string
PointerLocation	Current location of pointer	Values: *x*-, and *y*-coordinates
PointerWindow	Handle of window containing the pointer	Values: figure handle
ShowHiddenHandles	Show or hide handles marked as hidden	Values: on, off Default: off
Controlling MATLAB's behavior		
Diary	Enable the diary file	Values: on, off Default: off
DiaryFile	Name of the diary file	Values: filename (string) Default: diary
Echo	Display each line of script M-file as executed	Values: on, off Default: off
Format	Format used to display numbers	Values: short, shortE, long, longE, bank, hex, +, rat Default: shortE
FormatSpacing	Display or omit extra line feed	Values: compact, loose Default: loose
Language	System environment setting	Values: string Default: english
RecursionLimit	Maximum number of nested M-file calls	Values: integer Defalut: 2.1478e+009
Units	Units for PointerLocation and ScreenSize properties	Values: pixels, normalized, inches, centimeters, points, characters Default: pixels
Information about the display		

Property Name	Property Description	Property Value
FixedWidthFontName	Value for axes, text, and uicontrol FontName property	Values: font name Default: Courier
ScreenDepth	Depth of the display bitmap	Values: bits per pixel
ScreenSize	Size of the screen	Values: [left, bottom, width, height]
Information about terminals (X-Windows only)		
TerminalHideGraphCommands	Command to hide graphics window	Values: string
TerminalOneWindow	Indicates if there is only one graphics window	Values: on, off Default: on
TerminalDimensions	Size of the terminal	Values: scalar in pixels
TerminalProtocol	Identifies the type of terminal	Values: none, x, tek401x, tek410x
TerminalShowGraphCommand	Command to display graphics window	Value: string
General Information About Root Objects		
Children	Handles of all nonhidden Figue objects	Values: vector of handles
Parent	The root object has no parent	Value: [] (empty matrix)
Selected	This property is not used by the root object.	
Tag	User-specified label	Value: any string Default: '' (empty string)
Type	The type of graphics object (read only)	Value: the string 'root'
UserData	User-specified data	Values: any matrix Default: [] (empty matrix)

Property Name	Property Description	Property Value
MATLAB profiler		
`Profile`	Enable/disable profiler	Values: `on`, `off` Default: `off`
`ProfileFile`	Specify the name of M-file to profile	Values: pathname to M-file
`ProfileCount`	Output of profiler	Values: *n*-by-1 vector
`ProfileInterval`	Time increment at with to profile M-file	Values: scalar (seconds) Default: `0.01` seconds

Root Properties

This section lists property names along with the type of values each accepts. Curly braces { } enclose default values.

BusyAction `cancel | {queue}`

Not used by the root object.

ButtonDownFcn string

Not used by the root object.

CallbackObject handle (read only)

Handle of current callback's object. This property contains the handle of the object whose callback routine is currently executing. If no callback routines are executing, this property contains the empty matrix [ ]. See also the `gco` command.

CaptureMatrix (obsolete)

This property has been superseded by the `getframe` command.

CaptureRect (obsolete)

This property has been superseded by the `getframe` command.

Children vector of handles

Handles of child objects. A vector containing the handles of all nonhidden figure objects. You can change the order of the handles and thereby change the stacking order of the figures on the display.

Clipping `{on} | off`

Clipping has no effect on the root object.

CreateFcn

The root does not use this property.

CurrentFigure figure handle

Handle of the current figure window, which is the one most recently created, clicked in, or made current with the statement:

```
figure(h)
```

which restacks the figure to the top of the screen, or

```
set(0,'CurrentFigure',h)
```

which does not restack the figures. In these statements, h is the handle of an existing figure. If there are no figure objects,

```
get(0,'CurrentFigure')
```

returns the empty matrix. Note, however, that gcf always returns a figure handle, and creates one if there are no figure objects.

DeleteFcn string

This property is not used since you cannot delete the root object

Diary on | {off}

Diary file mode. When this property is on, MATLAB maintains a file (whose name is specified by the DiaryFile property) that saves a copy of all keyboard input and most of the resulting output. See also the diary command.

DiaryFile string

Diary filename. The name of the diary file. The default name is diary.

Echo on | {off}

Script echoing mode. When Echo is on, MATLAB displays each line of a script file as it executes. See also the echo command.

ErrorMessage string

Text of last error message. This property contains the last error message issued by MATLAB.

FixedWidthFontName font name

Fixed-width font to use for axes, text, and uicontrols whose FontName *is set to* FixedWidth. MATLAB uses the font name specified for this property as the value for axes, text, and uicontrol FontName properties when their FontName property is set to FixedWidth. Specifying the font name with this property eliminates the need to hardcode font names in MATLAB applications and thereby enables these applications to run without modification in locales where non-ASCII character sets are required. In these cases, MATLAB attempts to set the value of FixedWidthFontName to the correct value for a given locale.

MATLAB application developers should not change this property, but should create axes, text, and uicontrols with FontName properties set to FixedWidth when they want to use a fixed width font for these objects.

MATLAB end users can set this property if they do not want to use the preselected value. In locales where Latin-based characters are used, Courier is the default.

Format `short | {shortE} | long | longE | bank | hex | + | rat`

Output format mode. This property sets the format used to display numbers. See also the `format` command.

- `short` – Fixed-point format with 5 digits.
- `shortE` – Floating-point format with 5 digits.
- `shortG` – Fixed- or floating-point format displaying as many significant figures as possible with 5 digits.
- `long` – Scaled fixed-point format with 15 digits.
- `longE` – Floating-point format with 15 digits.
- `longG` – Fixed- or floating-point format displaying as many significant figures as possible with 15 digits.
- `bank` – Fixed-format of dollars and cents.
- `hex` – Hexadecimal format.
- `+` – Displays + and – symbols.
- `rat` – Approximation by ratio of small integers.

FormatSpacing `compact | {loose}`

Output format spacing (see also `format` command).

- `compact` — Suppress extra line feeds for more compact display.
- `loose` — Display extra line feeds for a more readable display.

HandleVisibility `{on} | callback | off`

This property is not useful on the root object.

HitTest `{on} | off`

This property is not useful on the root object.

Interruptible `{on} | off`

This property is not useful on the root object.

`Language` string

System environment setting.

`Parent` handle

Handle of parent object. This property always contains the empty matrix, as the root object has no parent.

`PointerLocation` `[x,y]`

Current location of pointer. A vector containing the *x*- and *y*-coordinates of the pointer position, measured from the lower-left corner of the screen. You can move the pointer by changing the values of this property. The `Units` property determines the units of this measurement.

This property always contains the instantaneous pointer location, even if the pointer is not in a MATLAB window. A callback routine querying the `PointerLocation` can get a different value than the location of the pointer when the callback was triggered. This difference results from delays in callback execution caused by competition for system resources.

`PointerWindow` handle (read only)

Handle of window containing the pointer. MATLAB sets this property to the handle of the figure window containing the pointer. If the pointer is not in a MATLAB window, the value of this property is 0. A callback routine querying the `PointerWindow` can get the wrong window handle if you move the pointer to another window before the callback executes. This error results from delays in callback execution caused by competition for system resources.

`Profile` on | {off}

M-file profiler on or off. Setting this property to `on` activates the profiler when you execute the M-files named in `ProfileFile`. The profiler determines what percentage of time MATLAB spends executing each line of the M-file. See also the `profile` command.

`ProfileFile` M-file name

M-file to profile. Set this property to the full path name of the M-file to profile.

`ProfileCount` vector

Profiler output. This property is a *n*-by-1 vector, where *n* is the number of lines of code in the profiled M-file. Each element in this vector represents the number of times the profiler found MATLAB executing a particular line of code. The `ProfileInterval` property determines how often MATLAB profiles (i.e., determines which line is executing).

`ProfileInterval` scalar

Time increment to profile M-file. This property sets the time interval at which the profiler checks to see what line in the M-file is executing.

`RecursionLimit` integer

Number of nested M-file calls. This property sets a limit to the number of nested calls to M-files MATLAB will make before stoping (or potentially running out of memory). By default the value is set to a large value. Setting this property to a smaller value (something like 150, for example) should prevent MATLAB from running out of memory and will instead cause MATLAB to issue an error when the limit is reached.

`ScreenDepth` bits per pixel

Screen depth. The depth of the display bitmap (i.e., the number of bits per pixel). The maximum number of simultaneously displayed colors on the current graphics device is 2 raised to this power.

`ScreenDepth` supersedes the `BlackAndWhite` property. To override automatic hardware checking, set this property to `1`. This value causes MATLAB to assume the display is monochrome. This is useful if MATLAB is running on color hardware but is displaying on a monochrome terminal. Such a situation can cause MATLAB to determine erroneously that the display is color.

`ScreenSize` 4-element rectangle vector (read only)

Screen size. A four-element vector,

```
[left,bottom,width,height]
```

that defines the display size. `left` and `bottom` are 0 for all `Units` except `pixels`, in which case `left` and `bottom` are 1. `width` and `height` are the screen dimensions in units specified by the `Units` property.

`Selected` `on | off`

This property has no effect on the root level.

`SelectionHighlight` `{on} | off`

This property has no effect on the root level.

ShowHiddenHandles on | {off}

Show or hide handles marked as hidden. When set to on, this property disables handle hiding and exposes all object handles, regardless of the setting of an object's HandleVisibility property. When set to off, all objects so marked remain hidden within the graphics hierarchy.

Tag string

User-specified object label. The Tag property provides a means to identify graphics objects with a user-specified label. While it is not necessary to identify the root object with a tag (since its handle is always 0), you can use this property to store any string value that you can later retrieve using set.

TerminalHideGraphCommandstring X-Windows only

Hide graph window command. This property specifies the escape sequence that MATLAB issues to hide the graph window when switching from graph mode back to command mode. This property is used only by the terminal graphics driver. Consult your terminal manual for the correct escape sequence.

TerminalOneWindow {on} | off X-Windows only

One window terminal. This property indicates whether there is only one window on your terminal. If the terminal uses only one window, MATLAB waits for you to press a key before it switches from graphics mode back to command mode. This property is used only by the terminal graphics driver.

TerminalDimensions pixels X-Windows only

Size of default terminal. This property defines the size of the terminal.

TerminalProtocol none | x | tek401x | tek410x X-Windows only

Type of terminal. This property tells MATLAB what type of terminal you are using. Specify tek401x for terminals that emulate Tektronix 4010/4014 terminals. Specify tek410x for terminals that emulate Tektronix 4100/4105 terminals. If you are using X Windows and MATLAB can connect to your X display server, this property is automatically set to x.

Once this property is set, you cannot change it unless you quit and restart MATLAB.

TerminalShowGraphCommandstring X-Windows only

Display graph window command. This property specifies the escape sequence that MATLAB issues to display the graph window when switching from command mode to graph mode. This property is only used by the terminal graphics driver. Consult your terminal manual for the appropriate escape sequence.

Type string (read only)

Class of graphics object. For the root object, `Type` is always `'root'`.

UIContextMenu handle

This property has no effect on the root level.

Units `{pixels} | normalized | inches | centimeters | points | characters`

Unit of measurement. This property specifies the units MATLAB uses to interpret size and location data. All units are measured from the lower-left corner of the screen. Normalized units map the lower-left corner of the screen to (0,0) and the upper right corner to (1.0,1.0). `inches`, `centimeters`, and `points` are absolute units (one point equals 1/72 of an inch). Characters are units defined by characters from the default system font; the width of one unit is the width of the letter x, the height of one character is the distance between the baselines of two lines of text.

This property affects the `PointerLocation` and `ScreenSize` properties. If you change the value of `Units`, it is good practice to return it to its default value after completing your operation so as not to affect other functions that assume `Units` is set to the default value.

UserData matrix

User specified data. This property can be any data you want to associate with the root object. MATLAB does not use this property, but you can access it using the `set` and `get` functions.

Visible `{on} | off`

Object visibility. This property has no effect on the root object.

Purpose Angle histogram

Syntax

```
rose(theta)
rose(theta,x)
rose(theta,nbins)
[tout,rout] = rose(...)
```

Description

`rose` creates an angle histogram, which is a polar plot showing the distribution of values grouped according to their numeric range. Each group is shown as one bin.

`rose(theta)` plots an angle histogram showing the distribution of `theta` in 20 angle bins or less. The vector `theta`, expressed in radians, determines the angle from the origin of each bin. The length of each bin reflects the number of elements in `theta` that fall within a group, which ranges from 0 to the greatest number of elements deposited in any one bin.

`rose(theta,x)` uses the vector `x` to specify the number and the locations of bins. `length(x)` is the number of bins and the values of `x` specify the center angle of each bin. For example, if `x` is a five-element vector, `rose` distributes the elements of `theta` in five bins centered at the specified `x` values.

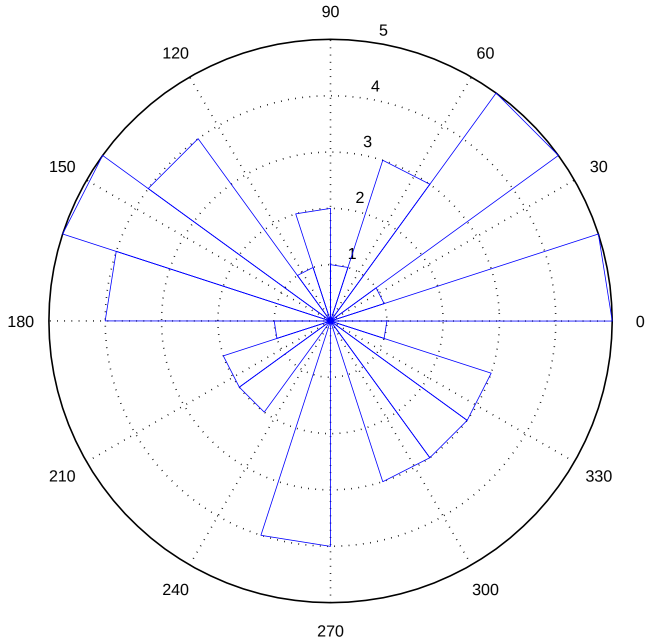
`rose(theta,nbins)` plots `nbins` equally spaced bins in the range `[0, 2*pi]`. The default is `20`.

`[tout,rout] = rose(...)` returns the vectors `tout` and `rout` so `polar(tout,rout)` generates the histogram for the data. This syntax does not generate a plot.

Example

Create a rose plot showing the distribution of 50 random numbers.

```
theta = 2*pi*rand(1,50);
rose(theta)
```



See Also `compass`, `feather`, `hist`, `polar`

Purpose Rotate object about a specified direction

Syntax

```
rotate(h,direction,alpha)
rotate(...,origin)
```

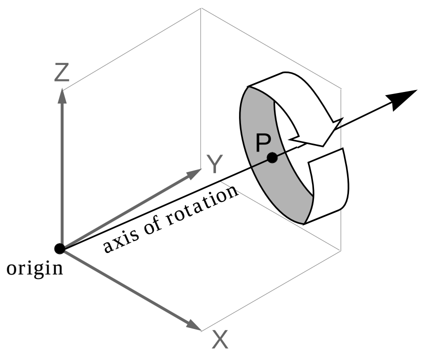
Description The `rotate` function rotates a graphics object in three-dimensional space, according to the right-hand rule.

`rotate(h,direction,alpha)` rotates the graphics object `h` by `alpha` degrees. `direction` is a two- or three-element vector that describes the axis of rotation in conjunction with the origin.

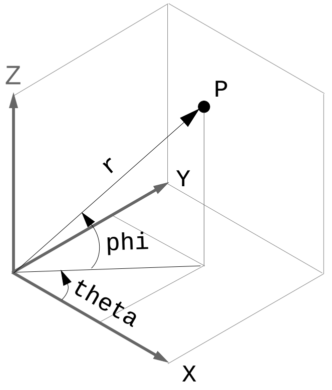
`rotate(...,origin)` specifies the origin of the axis of rotation as a three-element vector. The default origin is the center of the plot box.

Remarks The graphics object you want rotated must be a child of the same axes. The object's data is modified by the rotation transformation. This is in contrast to `view` and `rotate3d`, which only modify the viewpoint.

The axis of rotation is defined by an origin and a point *P* relative to the origin. *P* is expressed as the spherical coordinates `[theta phi]`, or as Cartesian coordinates.



The two-element form for `direction` specifies the axis direction using the spherical coordinates `[theta phi]`. `theta` is the angle in the *xy* plane counterclockwise from the positive *x*-axis. `phi` is the elevation of the direction vector from the *xy* plane.



The three-element form for `direction` specifies the axis direction using Cartesian coordinates. The direction vector is the vector from the origin to (X,Y,Z).

Examples

Rotate a graphics object 180° about the *x*-axis.

```
h = surf(peaks(20));
rotate(h,[1 0 0],180)
```

Rotate a surface graphics object 45° about its center in the *z* direction.

```
h = surf(peaks(20));
zdir = [0 0 1];
center = [10 10 0];
rotate(h,zdir,45,center)
```

Remarks

`rotate` changes the `Xdata`, `Ydata`, and `Zdata` properties of the appropriate graphics object.

See Also

`rotate3d`, `sph2cart`, `view`

The axes `CameraPosition`, `CameraTarget`, `CameraUpVector`, `CameraViewAngle`

Purpose

Rotate axes using mouse

Syntax

```
rotate3d
rotate3d on
rotate3d off
```

Description

`rotate3d on` enables interactive axes rotation within the current figure using the mouse. When interactive axes rotation is enabled, clicking on an axes draws an animated box, which rotates as the mouse is dragged, showing the view that will result when the mouse button is released. A numeric readout appears in the lower-left corner of the figure during this time, showing the current azimuth and elevation of the animated box. Releasing the mouse button removes the animated box and the readout, and changes the view of the axes to correspond to the last orientation of the animated box.

`rotate3d off` disables interactive axes rotation in the current figure.

`rotate3d` toggles interactive axes rotation in the current figure.

Double clicking on the figure restores the original view.

See Also

`camorbit`, `rotate`, `view`

Purpose

2-D Scatter plot

Syntax

```
scatter(X,Y,S,C)
scatter(X,Y)
scatter(X,Y,S)
scatter(...,markertype)
scatter(...,'filled')
h = scatter(...,)
```

Description

`scatter(X,Y,S,C)` displays colored circles at the locations specified by the vectors `X` and `Y` (which must be the same size).

`S` determines the size of each marker (specified in points^2). `S` can be a vector the same length as `X` and `Y` or a scalar. If `S` is a scalar, MATLAB draws all the markers the same size.

`C` determines the colors of each marker. When `C` is a vector the same length as `X` and `Y`, the values in `C` are linearly mapped to the colors in the current colormap. When `C` is a `length(X)`-by-3 matrix, it specifies the colors of the markers as RGB values. `C` can also be a color string (see `ColorSpec` for a list of color string specifiers)

`scatter(X,Y)` draws the markers in the default size and color.

`scatter(X,Y,S)` draws the markers at the specified sizes (`S`) with a single color.

`scatter(...,`*markertype*`)` uses the marker type specified instead of 'o' (see `LineSpec` for a list of marker specifiers).

`scatter(...,'filled')` fills the markers.

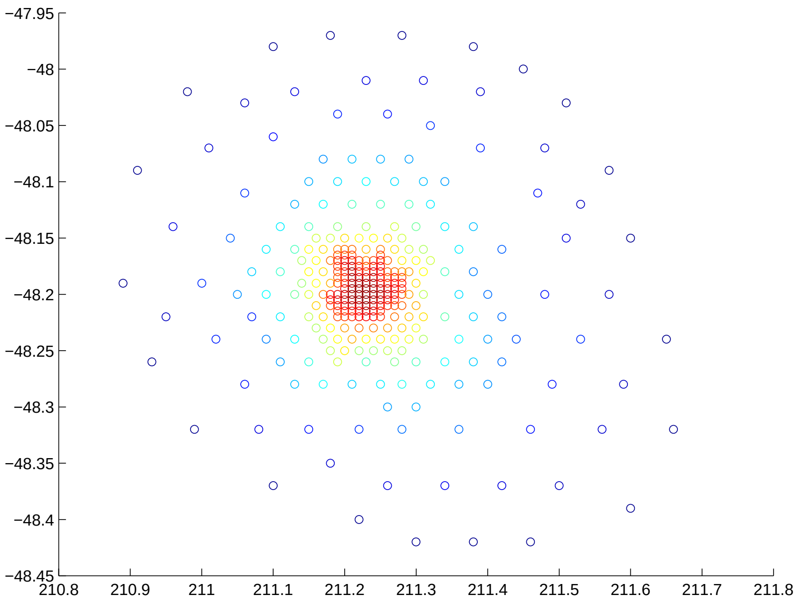
`h = scatter(...)` returns the handles to the line objects created by `scatter` (see `line` for a list of properties you can specify using the object handles and `set`).

Remarks

Use `plot` for single color, single marker size scatter plots.

Examples

```
load seamount
scatter(x,y,5,z)
```



See Also

`scatter3`, `plot`, `plotmatrix`

scatter3

Purpose 3-D scatter plot

Syntax

```
scatter3(X,Y,Z,S,C)
scatter3(X,Y,Z)
scatter3(X,Y,Z,S)
scatter3(...,markertype)
scatter3(...,'filled')
h = scatter3(...,)
```

Description

`scatter3(X,Y,Z,S,C)` displays colored circles at the locations specified by the vectors `X`, `Y`, and `Z` (which must all be the same size).

`S` determines the size of each marker (specified in points). `S` can be a vector the same length as `X`, `Y`, and `Z` or a scalar. If `S` is a scalar, MATLAB draws all the markers the same size.

`C` determines the colors of each marker. When `C` is a vector the same length as `X`, `Y`, and `Z`, the values in `C` are linearly mapped to the colors in the current colormap. When `C` is a `length(X)`-by-3 matrix, it specifies the colors of the markers as RGB values. `C` can also be a color string (see `ColorSpec` for a list of color string specifiers)

`scatter3(X,Y,Z)` draws the markers in the default size and color.

`scatter3(X,Y,Z,S)` draws the markers at the specified sizes (`S`) with a single color.

`scatter3(...,`*markertype*`)` uses the marker type specified instead of '`o`' (see `LineSpec` for a list of marker specifiers).

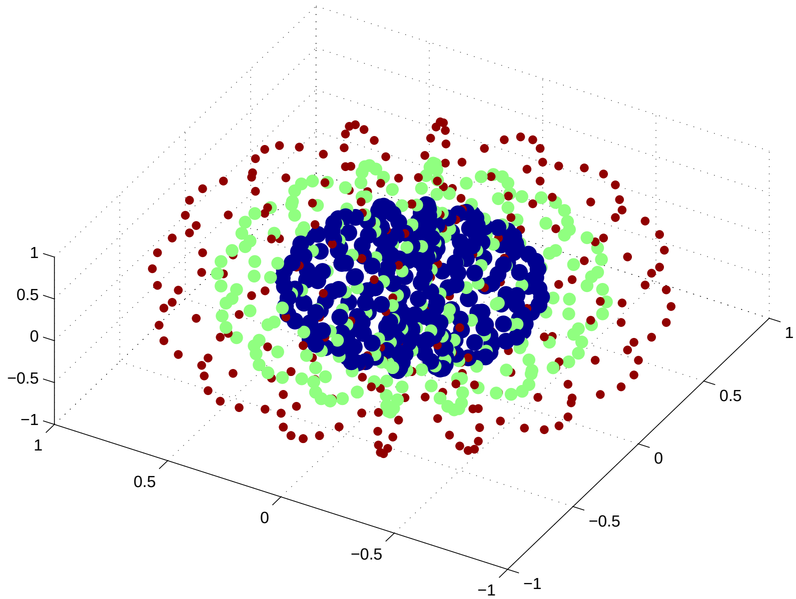
`scatter3(...,'filled')` fills the markers.

`h = scatter3(...)` returns the handles to the line objects created by `scatter3` (see `line` for a list of properties you can specify using the object handles and `set`).

Remarks Use `plot3` for single color, single marker size 3-D scatter plots.

Examples

```
[x,y,z] = sphere(16);
X = [x(:)*.5 x(:)*.75 x(:)];
Y = [y(:)*.5 y(:)*.75 y(:)];
Z = [z(:)*.5 z(:)*.75 z(:)];
S = repmat([1 .75 .5]*10,prod(size(x)),1);
C = repmat([1 2 3],prod(size(x)),1);
scatter3(X(:),Y(:),Z(:),S(:),C(:),'filled'), view(-60,60)
```



See Also

`scatter`, `plot3`

selectmoveresize

Purpose

Select, move, resize, or copy axes and uicontrol graphics objects

Syntax

```
A = selectmoveresize;
set(h,'ButtonDownFcn','selectmoveresize')
```

Description

`selectmoveresize` is useful as the callback routine for axes and uicontrol button down functions. When executed, it selects the object and allows you to move, resize, and copy it.

For example, this statement sets the `ButtonDownFcn` of the current axes to `selectmoveresize`:

```
set(gca,'ButtonDownFcn','selectmoveresize')
```

`A = selectmoveresize` returns a structure array containing:

- `A.Type`: a string containing the action type, which can be `Select`, `Move`, `Resize`, or `Copy`.
- `A.Handles`: a list of the selected handles or for a `Copy` an m-by-2 matrix containing the original handles in the first column and the new handles in the second column.

See Also

The `ButtonDownFcn` of axes and uicontrol graphics objects

Purpose

Semi-logarithmic plots

Syntax

```
semilogx(Y)
semilogx(X1,Y1,...)
semilogx(X1,Y1,LineSpec,...)
semilogx(...,'PropertyName',PropertyValue,...)
h = semilogx(...)

semilogy(...)
h = semilogy(...)
```

Description

`semilogx` and `semilogy` plot data as logarithmic scales for the *x*- and *y*-axis, respectively. logarithmic

`semilogx(Y)` creates a plot using a base 10 logarithmic scale for the *x*-axis and a linear scale for the *y*-axis. It plots the columns of `Y` versus their index if `Y` contains real numbers. `semilogx(Y)` is equivalent to `semilogx(real(Y), imag(Y))` if `Y` contains complex numbers. `semilogx` ignores the imaginary component in all other uses of this function.

`semilogx(X1,Y1,...)` plots all `Xn` versus `Yn` pairs. If only `Xn` or `Yn` is a matrix, `semilogx` plots the vector argument versus the rows or columns of the matrix, depending on whether the vector's row or column dimension matches the matrix.

`semilogx(X1,Y1,LineSpec,...)` plots all lines defined by the `Xn,Yn,LineSpec` triples. `LineSpec` determines line style, marker symbol, and color of the plotted lines.

`semilogx(...,'`*`PropertyName`*`',PropertyValue,...)` sets property values for all line graphics objects created by `semilogx`.

`semilogy(...)` creates a plot using a base 10 logarithmic scale for the *y*-axis and a linear scale for the *x*-axis.

`h = semilogx(...)` and `h = semilogy(...)` return a vector of handles to line graphics objects, one handle per line.

Remarks

If you do not specify a color when plotting more than one line, `semilogx` and `semilogy` automatically cycle through the colors and line styles in the order specified by the current axes `ColorOrder` and `LineStyleOrder` properties.

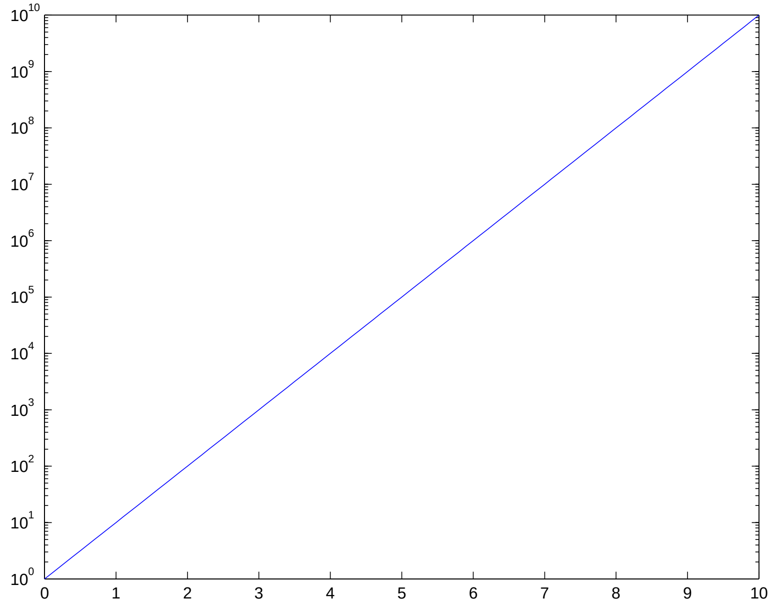
You can mix `Xn,Yn` pairs with `Xn,Yn,LineSpec` triples; for example,

```
semilogx(X1,Y1,X2,Y2,LineSpec,X3,Y3)
```

Examples

Create a simple `semilogy` plot.

```
x = 0:.1:10;
semilogy(x,10.^x)
```



See Also

`line`, `LineSpec`, `loglog`, `plot`

Purpose Set object properties

Syntax

```
set(H,'PropertyName',PropertyValue,...)
set(H,a)
set(H,pn,pv...)
set(H,pn,<m-by-n cell array>)
a= set(h)
a= set(0,'Factory')
a= set(0,'FactoryObjectTypePropertyName')
a= set(h,'Default')
a= set(h,'DefaultObjectTypePropertyName')
<cell array> = set(h,'PropertyName')
```

Description

`set(H,'`*`PropertyName`*`',PropertyValue,...)` sets the named properties to the specified values on the object(s) identified by `H`. `H` can be a vector of handles, in which case `set` sets the properties' values for all the objects.

`set(H,a)` sets the named properties to the specified values on the object(s) identified by `H`. `a` is a structure array whose field names are the object property names and whose field values are the values of the corresponding properties.

`set(H,pn,pv,...)` sets the named properties specified in the cell array `pn` to the corresponding value in the cell array `pv` for all objects identified in `H`.

`set(H,pn,<m-by-n cell array>)` sets `n` property values on each of `m` graphics objects, where `m = length(H)` and `n` is equal to the number of property names contained in the cell array `pn`. This allows you to set a given group of properties to different values on each object.

`a = set(h)` returns the user-settable properties and possible values for the object identified by `h`. `a` is a structure array whose field names are the object's property names and whose field values are the possible values of the corresponding properties. If you do not specify an output argument, MATLAB displays the information on the screen. `h` must be scalar.

`a = set(0,'Factory')` returns the properties whose defaults are user settable for all objects and lists possible values for each property. `a` is a structure array whose field names are the object's property names and whose field values are the possible values of the corresponding properties. If you do not specify an output argument, MATLAB displays the information on the screen.

`a = set(0,'Factory`*ObjectTypePropertyName*`')` returns the possible values of the named property for the specified object type, if the values are strings. The argument `Factory`*ObjectTypePropertyName* is the word `Factory` concatenated with the object type (e.g., `axes`) and the property name (e.g., `CameraPosition`).

`a = set(h,'Default')` returns the names of properties having default values set on the object identified by `h`. `set` also returns the possible values if they are strings. `h` must be scalar.

`a = set(h,'Default`*ObjectTypePropertyName*`')` returns the possible values of the named property for the specified object type, if the values are strings. The argument `Default`*ObjectTypePropertyName* is the word `Default` concatenated with the object type (e.g., `axes`) and the property name (e.g., `CameraPosition`). For example, `DefaultAxesCameraPosition`. `h` must be scalar.

`pv = set(h,'`*PropertyName*`')` returns the possible values for the named property. If the possible values are strings, `set` returns each in a cell of the cell array, `pv`. For other properties, `set` returns an empty cell array. If you do not specify an output argument, MATLAB displays the information on the screen. `h` must be scalar.

Remarks

You can use any combination of property name/property value pairs, structure arrays, and cell arrays in one call to `set`.

Examples

Set the `Color` property of the current axes to blue.

```
set(gca,'Color','b')
```

Change all the lines in a plot to black.

```
plot(peaks)
set(findobj('Type','line'),'Color','k')
```

You can define a group of properties in a structure to better organize your code. For example, these statements define a structure called `active`, which contains a set of property definitions used for the uicontrol objects in a particular figure. When this figure becomes the current figure, MATLAB changes colors and enables the controls.

```
active.BackgroundColor = [.7 .7 .7];
active.Enable = 'on';
active.ForegroundColor = [0 0 0];

if gcf == control_fig_handle
   set(findobj(control_fig_handle,'Type','uicontrol'),active)
end
```

You can use cell arrays to set properties to different values on each object. For example, these statements define a cell array to set three properties,

```
PropName(1) = {'BackgroundColor'};
PropName(2) = {'Enable'};
PropName(3) = {'ForegroundColor'};
```

These statements define a cell array containing three values for each of three objects (i.e., a 3-by-3 cell array).

```
PropVal(1,1) = {[.5 .5 .5]};
PropVal(1,2) = {'off'};
PropVal(1,3) = {[.9 .9 .9]};

PropVal(2,1) = {[1 0 0]};
PropVal(2,2) = {'on'};
PropVal(2,3) = {[1 1 1]};

PropVal(3,1) = {[.7 .7 .7]};
PropVal(3,2) = {'on'};
PropVal(3,3) = {[0 0 0]};
```

Now pass the arguments to `set`,

```
set(H,PropName,PropVal)
```

where `length(H) = 3` and each element is the handle to a uicontrol.

See Also

`findobj`, `gca`, `gcf`, `gco`, `gcbo`, `get`

shading

Purpose

Set color shading properties

Syntax

```
shading flat
shading faceted
shading interp
```

Description

The `shading` function controls the color shading of surface and patch graphics objects.

`shading flat` each mesh line segment and face has a constant color determined by the color value at the end point of the segment or the corner of the face that has the smallest index or indices.

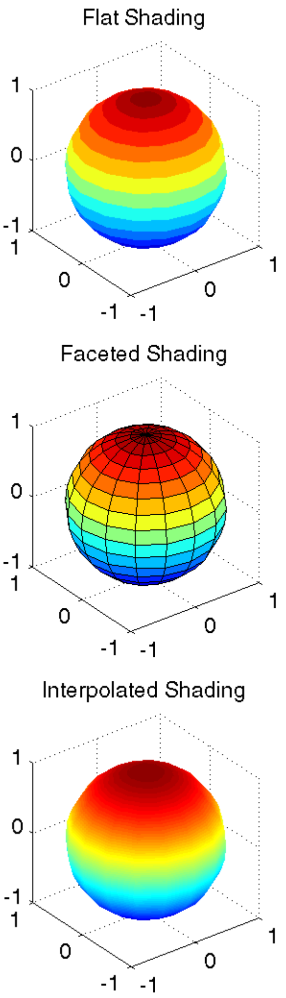
`shading faceted` flat shading with superimposed black mesh lines. This is the default shading mode.

`shading interp` varies the color in each line segment and face by interpolating the colormap index or true color value across the line or face.

Examples

Compare a flat, faceted, and interpolated-shaded sphere.

```
subplot(3,1,1)
sphere(16)
axis square
shading flat
title('Flat Shading')

subplot(3,1,2)
sphere(16)
axis square
shading faceted
title('Faceted Shading')

subplot(3,1,3)
sphere(16)
axis square
shading interp
title('Interpolated Shading')
```



Algorithm

`shading` sets the `EdgeColor` and `FaceColor` properties of all surface and patch graphics objects in the current axes. `shading` sets the appropriate values, depending on whether the surface or patch objects represent meshes or solid surfaces.

See Also

`fill`, `fill3`, `hidden`, `mesh`, `patch`, `pcolor`, `surf`

The `EdgeColor` and `FaceColor` properties for surface and patch graphics objects.

Purpose

Reduce the size of patch faces

Syntax

```
shrinkfaces(p,sf)
nfv = shrinkfaces(p,sf)
nfv = shrinkfaces(fv,sf)
shrinkfaces(p), shrinkfaces(fv)
nfv = shrinkfaces(f,v,sf)
[nf,nv] = shrinkfaces(...)
```

Description

`shrinkfaces(p,sf)` shrinks the area of the faces in patch `p` to shrink factor `sf`. A shrink factor of 0.6 shrinks each face to 60% of its original area. If the patch contains shared vertices, MATLAB creates nonshared vertices before performing the face-area reduction.

`nfv = shrinkfaces(p,sf)` returns the face and vertex data in the struct `nfv`, but does not set the `Faces` and `Vertices` properties of patch `p`.

`nfv = shrinkfaces(fv,sf)` uses the face and vertex data from the struct `fv`.

`shrinkfaces(p)` and `shrinkfaces(fv)` (without specifying a shrink factor) assume a shrink factor of 0.3.

`nfv = shrinkfaces(f,v,sf)` uses the face and vertex data from the arrays `f` and `v`.

`[nf,nv] = shrinkfaces(...)` returns the face and vertex data in two separate arrays instead of a struct.

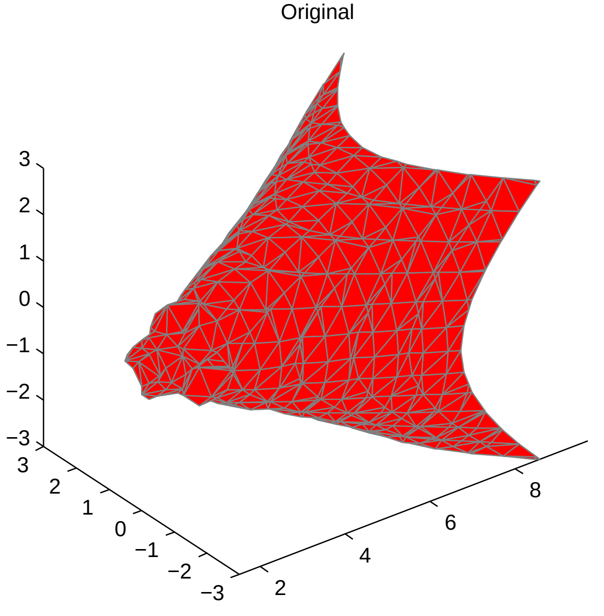
Examples

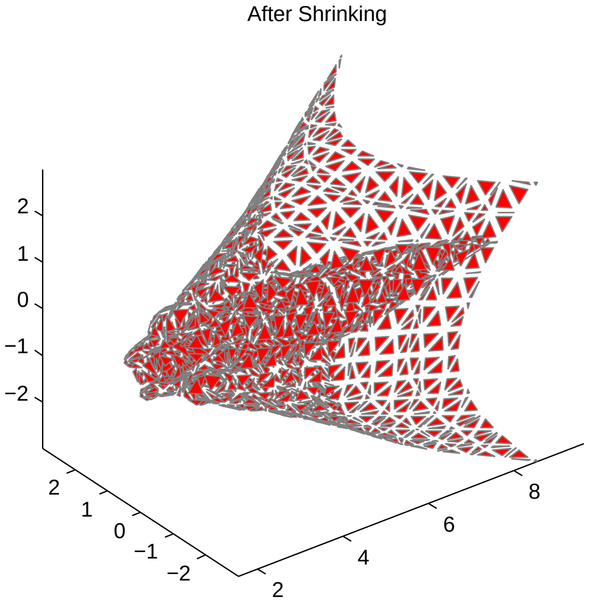
This example uses the flow data set, which represents the speed profile of a submerged jet within a infinite tank (type `help flow` for more information). Two isosurfaces provide a before and after view of the effects of shrinking the face size.

- First `reducevolume` samples the flow data at every other point and then `isosurface` generates the faces and vertices data.
- The `patch` command accepts the face/vertex struct and draws the first (`p1`) isosurface.

- Use the daspect, view, and axis commands to set up the view and then add a title.
- The shrinkfaces command modifies the face/vertex data and passes it directly to patch.

```
[x,y,z,v] = flow;
[x,y,z,v] = reducevolume(x,y,z,v,2);
fv = isosurface(x,y,z,v,-3);
p1 = patch(fv);
set(p1,'FaceColor','red','EdgeColor',[.5,.5,.5]);
daspect([1 1 1]); view(3); axis tight
title('Original')

figure
p2 = patch(shrinkfaces(fv,.3));
set(p2,'FaceColor','red','EdgeColor',[.5,.5,.5]);
daspect([1 1 1]); view(3); axis tight
title('After Shrinking')
```





See Also isocaps, isonormals, isosurface, reducepatch, reducevolume, smooth3, subvolume

slice

Purpose

Volumetric slice plot

Syntax

```
slice(V,sx,sy,sz)
slice(X,Y,Z,V,sx,sy,sz)
slice(V,XI,YI,ZI)
slice(X,Y,Z,V,XI,YI,ZI)
slice(...,'method')
h = slice(...)
```

Description

`slice` displays orthogonal slice planes through volumetric data.

`slice(V,sx,sy,sz)` draws slices along the x, y, z directions in the volume `V` at the points in the vectors `sx`, `sy`, and `sz`. `V` is an *m*-by-*n*-by-*p* volume array containing data values at the default location `X = 1:n`, `Y = 1:m`, `Z = 1:p`. Each element in the vectors `sx`, `sy`, and `sz` defines a slice plane in the *x*-, *y*-, or *z*-axis direction.

`slice(X,Y,Z,V,sx,sy,sz)` draws slices of the volume `V`. `X`, `Y`, and `Z` are three-dimensional arrays specifying the coordinates for `V`. `X`, `Y`, and `Z` must be monotonic and orthogonally spaced (as if produced by the function `meshgrid`). The color at each point is determined by 3-D interpolation into the volume `V`.

`slice(V,XI,YI,ZI)` draws data in the volume `V` for the slices defined by `XI`, `YI`, and `ZI`. `XI`, `YI`, and `ZI` are matrices that define a surface, and the volume is evaluated at the surface points. `XI`, `YI`, and `ZI` must all be the same size.

`slice(X,Y,Z,V,XI,YI,ZI)` draws slices through the volume `V` along the surface defined by the arrays `XI`, `YI`, `ZI`.

`slice(...,'`*`method`*`')` specifies the interpolation method. `'`*`method`*`'` is `'linear'`, `'cubic'`, or `'nearest'`.

- `linear` specifies trilinear interpolation (the default).
- `cubic` specifies tricubic interpolation.
- `nearest` specifies nearest neighbor interpolation.

`h = slice(...)` returns a vector of handles to surface graphics objects.

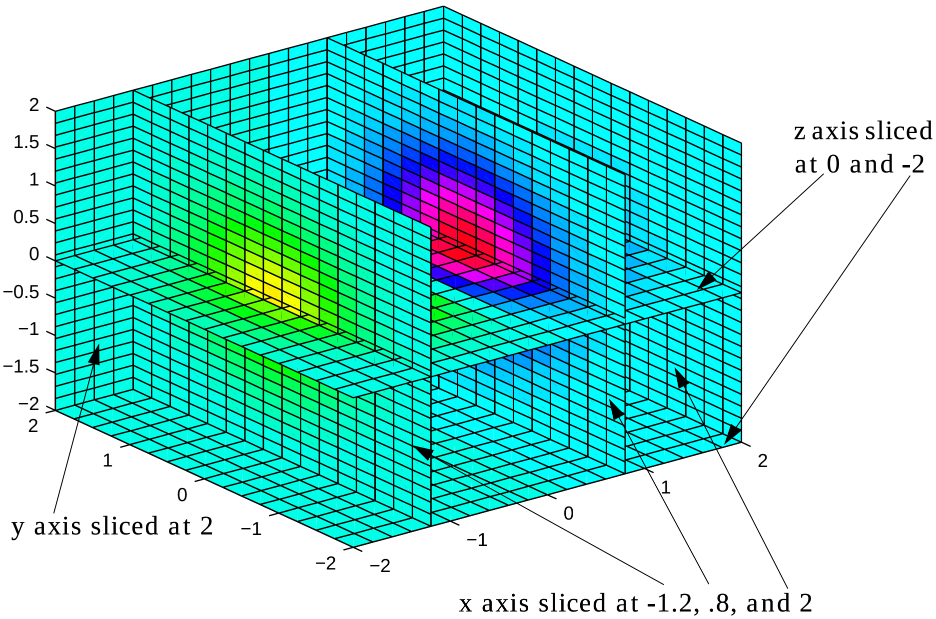
Remarks The color drawn at each point is determined by interpolation into the volume V.

Examples Visualize the function

$$v = xe^{(-x^2 - y^2 - z^2)}$$

over the range $-2 \le x \le 2$, $-2 \le y \le 2$, $-2 \le z \le 2$:

```
[x,y,z] = meshgrid(-2:.2:2,-2:.25:2,-2:.16:2);
v = x.*exp(-x.^2-y.^2-z.^2);
xslice = [-1.2,.8,2]; yslice = 2; zslice = [-2,0];
slice(x,y,z,v,xslice,yslice,zslice)
colormap hsv
```



Slicing At Arbitrary Angles

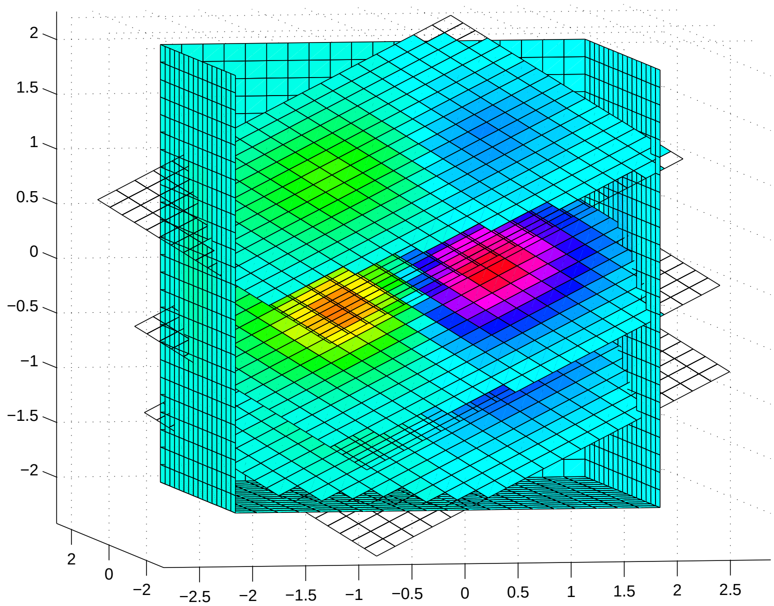
You can also create slices that are oriented in arbitrary planes. To do this,

- Create a slice surface in the domain of the volume (surf, linspace).
- Orient this surface with respect the the axes (rotate).
- Get the XData, YData, and ZData of the surface (get).
- Use this data to draw the slice plane within the volume.

For example, these statements slice the volume in the first example with a rotated plane. Placing these commands within a for loop "passes" the plane through the volume along the z-axis.

```
for i = –2:.5:2
   hsp = surf(linspace(–2,2,20),linspace(–2,2,20),zeros(20)+i);
   rotate(hsp,[1,–1,1],30)
   xd = get(hsp,'XData');
   yd = get(hsp,'YData');
   zd = get(hsp,'ZData');
   delete(hsp)
   slice(x,y,z,v,[–2,2],2,-2) % Draw some volume boundaries
   hold on
   slice(x,y,z,v,xd,yd,zd)
   hold off
   axis tight
   view(–5,10)
   drawnow
end
```

The following picture illustrates three positions of the same slice surface as it passes through the volume.

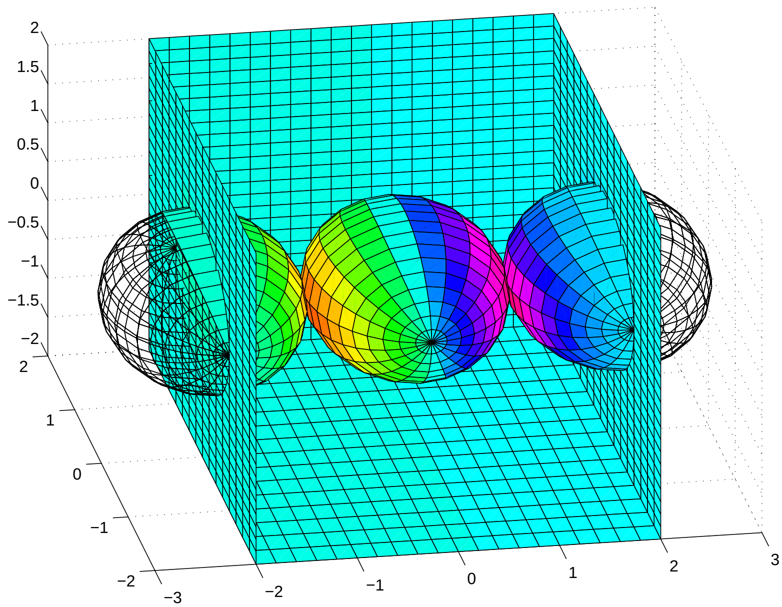


Slicing with a Nonplanar Surface

You can slice the volume with any surface. This example probes the volume created in the previous example by passing a spherical slice surface through the volume.

```
[xsp,ysp,zsp] = sphere;
slice(x,y,z,v,[-2,2],2,-2)  % Draw some volume boundaries

for i = -3:.2:3
   hsp = surface(xsp+i,ysp,zsp);
   rotate(hsp,[1 0 0],90)
   xd = get(hsp,'XData');
   yd = get(hsp,'YData');
   zd = get(hsp,'ZData');
   delete(hsp)
   hold on
   hslicer = slice(x,y,z,v,xd,yd,zd);
   axis tight
   xlim([-3,3])
   view(-10,35)
   drawnow
   delete(hslicer)
   hold off
end
```

The following picture illustrates three positions of the spherical slice surface as it passes through the volume.



See Also

`interp3`, `meshgrid`

smooth3

Purpose Smooth 3-D data

Syntax

```
W = smooth3(V)
W = smooth3(V,'filter')
W = smooth3(V,'filter',size)
W = smooth3(V,'filter',size,sd)
```

Description `W = smooth3(V)` smooths the input data `V` and returns the smoothed data in `W`.

`W = smooth3(V,'`*filter*`')` *filter* determines the convolution kernel and can be the strings `gaussian` or `box` (default).

`W = smooth3(V,'`*filter*`',size)` sets the size of the convolution kernel (default is `[3 3 3]`). If `size` is scalar, then `size` is interpreted as [`size`, `size`, `size`].

`W = smooth3(V,'`*filter*`',size,sd)` sets an attribute of the convolution kernel. When *filter* is `gaussian`, `sd` is the standard deviation (default is .65).

Examples This example smooths some random 3-D data and then creates an isosurface with end caps.

```
data = rand(10,10,10);
data = smooth3(data,'box',5);
p1 = patch(isosurface(data,.5), ...
   'FaceColor','blue','EdgeColor','none');
p2 = patch(isocaps(data,.5), ...
    'FaceColor','interp','EdgeColor','none');
isonormals(data,p1)
view(3); axis vis3d tight
camlight; lighting phong
```

See Also `isocaps`, `isonormals`, `isosurface`, `patch`, `reducepatch`, `reducevolume`, `subvolume`

Purpose Generate sphere

Syntax

```
sphere
sphere(n)
[X,Y,Z] = sphere(...)
```

Description The `sphere` function generates the *x*-, *y*-, and *z*-coordinates of a unit sphere for use with `surf` and `mesh`.

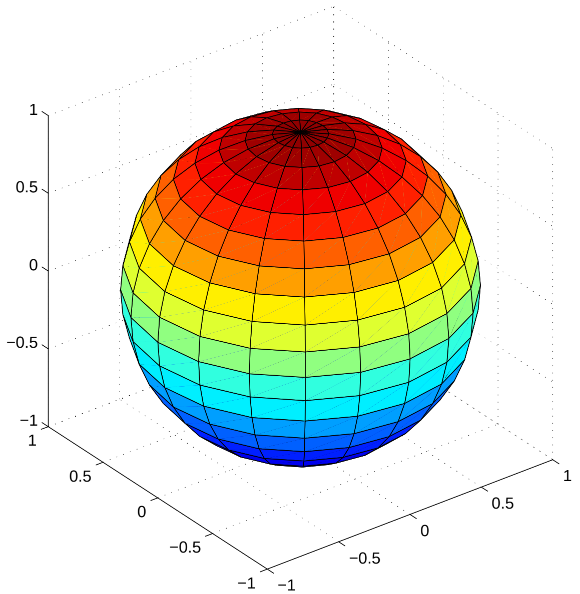
`sphere` generates a sphere consisting of 20-by-20 faces.

`sphere(n)` draws a `surf` plot of an `n`-by-`n` sphere in the current figure.

`[X,Y,Z] = sphere(n)` returns the coordinates of a sphere in three matrices that are `(n+1)`–by–`(n+1)` in size. You draw the sphere with `surf(X,Y,Z)` or `mesh(X,Y,Z)`.

Examples Generate and plot a sphere.

```
sphere
axis equal
```



See Also `cylinder`, `axis`

Purpose Spin colormap

Syntax

```
spinmap
spinmap(t)
spinmap(t,inc)
spinmap('inf')
```

Description The `spinmap` function shifts the colormap RGB values by some incremental value. For example, if the increment equals 1, color 1 becomes color 2, color 2 becomes color 3, etc.

`spinmap` cyclically rotates the colormap for approximately five seconds using an incremental value of 2.

`spinmap(t)` rotates the colormap for approximately `10*t` seconds. The amount of time specified by `t` depends on your hardware configuration (e.g., if you are running MATLAB over a network).

`spinmap(t,inc)` rotates the colormap for approximately `10*t` seconds and specifies an increment `inc` by which the colormap shifts. When `inc` is 1, the rotation appears smoother than the default (i.e., 2). Increments greater than 2 are less smooth than the default. A negative increment (e.g., –2) rotates the colormap in a negative direction.

`spinmap('inf')` rotates the colormap for an infinite amount of time. To break the loop, press **Ctrl-C**.

See Also `colormap`

Purpose Stairstep plot

Syntax

```
stairs(Y)
stairs(X,Y)
stairs(...,LineSpec)
[xb,yb] = stairs(Y)
[xb,yb] = stairs(X,Y)
```

Description Stairstep plots are useful for drawing time-history plots of digitally sampled data systems.

`stairs(Y)` draws a stairstep plot of the elements of `Y`. When `Y` is a vector, the *x*-axis scale ranges from `1` to `size(Y)`. When `Y` is a matrix, the *x*-axis scale ranges from `1` to the number of rows in `Y`.

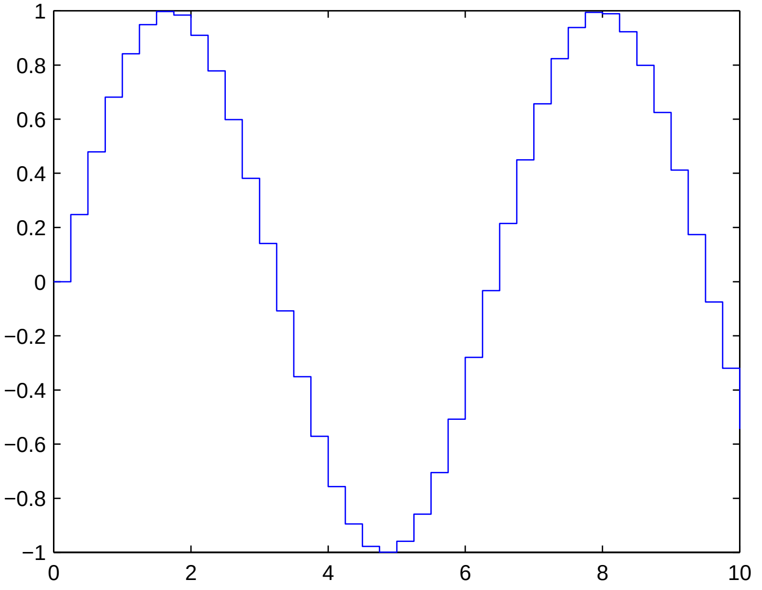
`stairs(X,Y)` plots `X` versus the columns of `Y`. `X` and `Y` are vectors of the same size or matrices of the same size. Additionally, `X` can be a row or a column vector, and `Y` a matrix with `length(X)` rows.

`stairs(...,LineSpec)` specifies a line style, marker symbol, and color for the plot (see `LineSpec` for more information).

`[xb,yb] = stairs(Y)` and `[xb,yb] = stairs(x,Y)` do not draw graphs, but return vectors `xb` and `yb` such that `plot(xb,yb)` plots the stairstep graph.

Examples Create a stairstep plot of a sine wave.

```
x = 0:.25:10;
stairs(x,sin(x))
```



See Also `bar`, `hist`

Purpose

Plot discrete sequence data

Syntax

```
stem(Y)
stem(X,Y)
stem(...,'fill')
stem(...,LineSpec)
h = stem(...)
```

Description

A two-dimensional stem plot displays data as lines extending from the *x*-axis. A circle (the default) or other marker whose *y*-position represents the data value terminates each stem.

`stem(Y)` plots the data sequence `Y` as stems that extend from equally spaced and automatically generated values along the *x*-axis. When `Y` is a matrix, `stem` plots all elements in a row against the same *x* value.

`stem(X,Y)` plots `X` versus the columns of `Y`. `X` and `Y` are vectors or matrices of the same size. Additionally, `X` can be a row or a column vector and `Y` a matrix with `length(X)` rows.

`stem(...,'fill')` specifies whether to color the circle at the end of the stem.

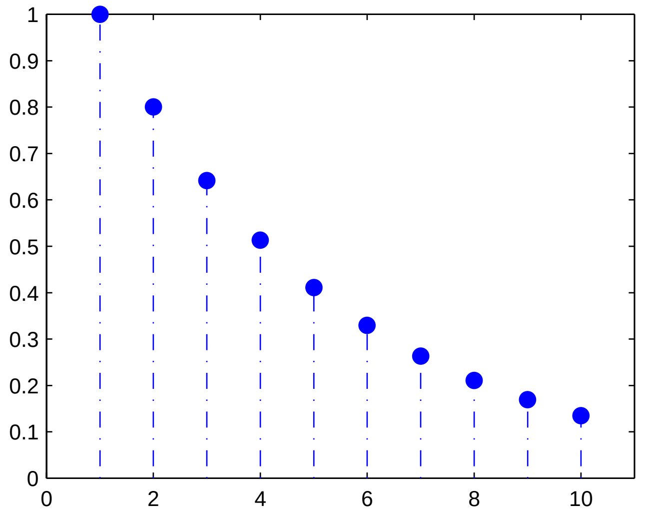
`stem(...,LineSpec)` specifies the line style, marker symbol, and color for the stem plot. See `LineSpec` for more information.

`h = stem(...)` returns handles to line graphics objects.

Examples

Create a stem plot of 10 random numbers.

```
y = linspace(0,2,10);
stem(exp(-y),'fill','-.')
axis ([0 11 0 1])
```



See Also

`bar`, `plot`, `stairs`, `stem3`

Purpose

Plot three-dimensional discrete sequence data

Syntax

```
stem3(Z)
stem3(X,Y,Z)
stem3(...,'fill')
stem3(...,LineSpec)
h = stem3(...)
```

Description

Three-dimensional stem plots display lines extending from the *xy*-plane. A circle (the default) or other marker symbol whose *z*-position represents the data value terminates each stem.

`stem3(Z)` plots the data sequence `Z` as stems that extend from the *xy*-plane. *x* and *y* are generated automatically. When `Z` is a row vector, `stem3` plots all elements at equally spaced *x* values against the same *y* value. When `Z` is a column vector, `stem3` plots all elements at equally spaced *y* values against the same *x* value.

`stem3(X,Y,Z)` plots the data sequence `Z` at values specified by `X` and `Y`. `X`, `Y`, and `Z` must all be vectors or matrices of the same size.

`stem3(...,'fill')` specifies whether to color the interior of the circle at the end of the stem.

`stem3(...,LineSpec)` specifies the line style, marker symbol, and color for the stems. See `LineSpec` for more information.

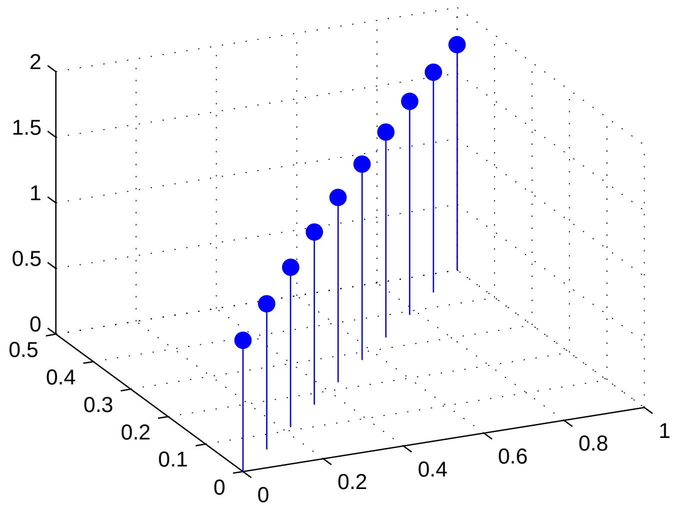
`h = stem3(...)` returns handles to line graphics objects.

Examples

Create a three-dimensional stem plot to visualize a function of two variables.

```
X = linspace(0,1,10);
Y = X./2;
Z = sin(X) + cos(Y);
stem3(X,Y,Z,'fill')
view(-25,30)
```

:



See Also `bar`, `plot`, `stairs`, `stem`

stream2

Purpose

Compute 2-D stream line data

Syntax

```
XY = stream2(x,y,u,v,startx,starty)
XY = stream2(u,v,startx,starty)
XY = stream2(...,options)
```

Description

`XY = stream2(x,y,u,v,startx,starty)` computes stream lines from vector data `u` and `v`. The arrays `x` and `y` define the coordinates for `u` and `v` and must be monotonic and 2-D plaid (such as the data produced by `meshgrid`). `startx` and `starty` define the starting positions of the stream lines. The returned value `XY` contains a cell array of vertex arrays.

`XY = stream2(u,v,startx,starty)` assumes the arrays `x` and `y` are defined as `[x,y] = meshgrid(1:n,1:m)` where `[m,n] = size(u)`.

`XY = stream2(...,options)` specifies the options used when creating the stream lines. Define `options` as a one or two element vector containing the step size or the step size and the maximum number of vertices in a stream line:

```
[stepsize]
```

or

```
[stepsize, max_number_vertices]
```

If you do not specify a value, MATLAB uses the default:

- stepsize = 0.1 (one tenth of a cell)
- naximum number of vertices = 1000

Use the `streamline` command to plot the data returned by `stream2`.

Examples

This example draws 2-D stream lines from data representing air currents over regions of North America.

```
load wind
[sx,sy] = meshgrid(80,20:10:50);
streamline(stream2(x(:,:,5),y(:,:,5),u(:,:,5),v(:,:,5),sx,sy));
```

See Also

`coneplot`, `isosurface`, `reducevolume` `smooth3`, `stream3`, `streamline`, `subvolume`

Purpose Compute 3-D stream line data

Syntax

```
XYZ = stream3(X,Y,Z,U,V,W,startx,starty,startz)
XYZ = stream3(U,V,W,startx,starty,startz)
```

Description

XYZ = stream3(X,Y,Z,U,V,W,startx,starty,startz) computes stream lines from vector data U, V, W. The arrays X, Y, Z define the coordinates for U, V, W and must be monotonic and 3-D plaid (such as the data produced by meshgrid). startx, starty, and startz define the starting positions of the stream lines. The returned value XYZ contains a cell array of vertex arrays.

XYZ = stream3(U,V,W,startx,starty,startz) assumes the arrays X, Y, and Z are defined as [X,Y,Z] = meshgrid(1:N,1:M,1:P) where [M,N,P] = size(U).

XYZ = stream3(...,options) specifies the options used when creating the stream lines. Define options as a one or two element vector containing the step size or the step size and the maximum number of vertices in a stream line:

```
[stepsize]
```

or

```
[stepsize, max_number_vertices]
```

If you do not specify values, MATLAB uses the default:

- stepsize = 0.1 (one tenth of a cell)
- naximum number of vertices = 1000

Use the streamline command to plot the data returned by stream3.

Examples

This example draws 3-D stream lines from data representing air currents over regions of North America.

```
load wind
[sx sy sz] = meshgrid(80,20:10:50,0:5:15);
streamline(stream3(x,y,z,u,v,w,sx,sy,sz))
view(3)
```

See Also

coneplot, isosurface, reducevolume smooth3, stream2, streamline, subvolume

Purpose

Draw stream lines from 2-D or 3-D vector data

Syntax

```
h = streamline(X,Y,Z,U,V,W,startx,starty,startz)
h = streamline(U,V,W,startx,starty,startz)
h = streamline(XYZ)
h = streamline(X,Y,U,V,startx,starty)
h = streamline(U,V,startx,starty)
h = streamline(XY)
h = streamline(...,options)
```

Description

`h = streamline(X,Y,Z,U,V,W,startx,starty,startz)` draws stream lines from 3-D vector data `U`, `V`, `W`. The arrays `X`, `Y`, `Z` define the coordinates for `U`, `V`, `W` and must be monotonic and 3-D plaid (such as the data produced by `meshgrid`). `startx`, `starty`, `startz` define the starting positions of the stream lines. The output argument `h` contains a vector of line handles, one handle for each stream line.

`h = streamline(U,V,W,startx,starty,startz)` assumes the arrays `X`, `Y`, and `Z` are defined as `[X,Y,Z] = meshgrid(1:N,1:M,1:P)` where `[M,N,P] = size(U)`.

`h = streamline(XYZ)` assumes `XYZ` is a precomputed cell array of vertex arrays (as produced by `stream3`).

`h = streamline(X,Y,U,V,startx,starty)` draws stream lines from 2-D vector data `U`, `V`. The arrays `X`, `Y` define the coordinates for `U`, `V` and must be monotonic and 2-D plaid (such as the data produced by `meshgrid`). `startx` and `starty` define the starting positions of the stream lines. The output argument `h` contains a vector of line handles, one handle for each stream line.

`h = streamline(U,V,startx,starty)` assumes the arrays `X` and `Y` are defined as `[X,Y] = meshgrid(1:N,1:M)` where `[M,N] = size(U)`.

`h = streamline(XY)` assumes `XY` is a precomputed cell array of vertex arrays (as produced by `stream2`).

`streamline(...,options)` specifies the options used when creating the stream lines. Define `options` as a one or two element vector containing the step size or the step size and the maximum number of vertices in a stream line:

```
[stepsize]
```

or

```
[stepsize, max_number_vertices]
```

If you do not specify values, MATLAB uses the default:

- stepsize = 0.1 (one tenth of a cell)
- naximum number of vertices = 1000

Examples

This example draws stream lines from data representing air currents over a region of North America. Loading the `wind` data set creates the variables `x`, `y`, `z`, `u`, `v`, and `w` in the MATLAB workspace.

The plane of stream lines indicates the flow of air from the west to the east (the x direction) beginning at x = 80 (which is close to the minimum value of the x coordinates). The y and z coordinate starting points are multivalued and approximately span the range of these coordinates. `meshgrid` generates the starting positions of the stream lines.

```
load wind
[sx,sy,sz] = meshgrid(80,20:10:50,0:5:15);
h = streamline(x,y,z,u,v,w,sx,sy,sz);
set(h,'Color','red')
view(3)
```

See Also

`stream2`, `stream3`, `coneplot`, `isosurface`, `smooth3`, `subvolume`, `reducevolume`

Purpose Create and control multiple axes

Syntax

```
subplot(m,n,p)
subplot(h)
subplot('Position',[left bottom width height])
h = subplot(...)
```

Description `subplot` divides the current figure into rectangular panes that are numbered row-wise. Each pane contains an axes. Subsequent plots are output to the current pane.

`subplot(m,n,p)` creates an axes in the `p`-th pane of a figure divided into an `m`-by-`n` matrix of rectangular panes. The new axes becomes the current axes.

`subplot(h)` makes the axes with handle `h` current for subsequent plotting commands.

`subplot('Position',[left bottom width height])` creates an axes at the position specified by a four-element vector. `left`, `bottom`, `width`, and `height` are in normalized coordinates in the range from 0.0 to 1.0.

`h = subplot(...)` returns the handle to the new axes.

Remarks If a `subplot` specification causes a new axes to overlap an existing axes, `subplot` deletes the existing axes. `subplot(1,1,1)` or `clf` deletes all axes objects and returns to the default `subplot(1,1,1)` configuration.

You can omit the parentheses and specify subplot as.

```
subplot mnp
```

where `m` refers to the row, `n` refers to the column, and `p` specifies the pane.

Special Case – subplot(111)

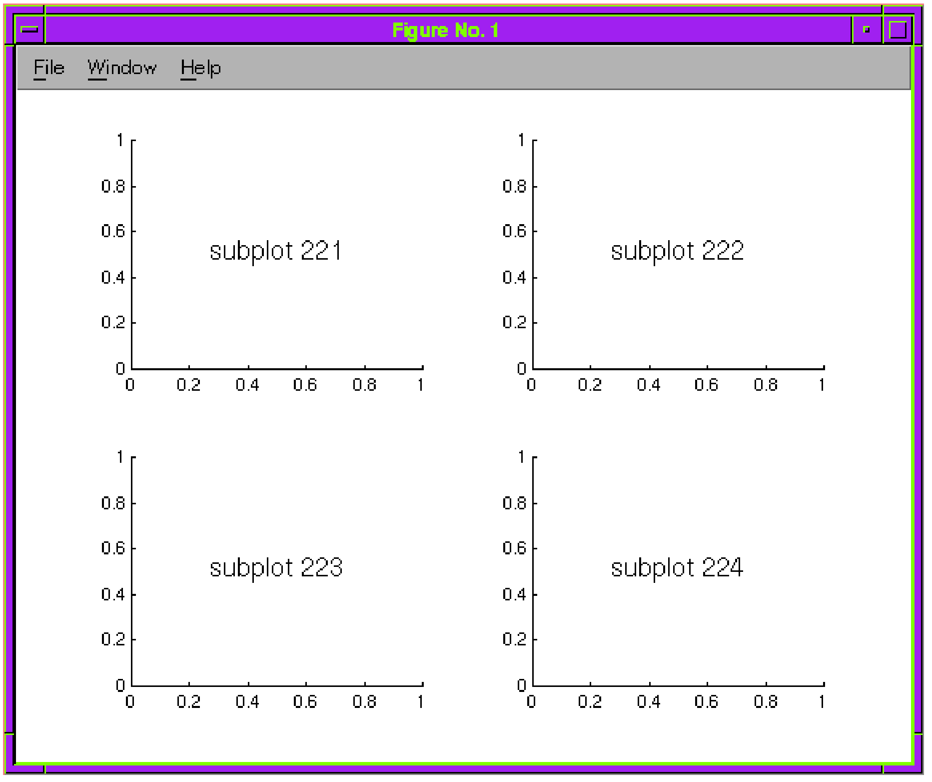
The command `subplot(111)` is not identical in behavior to `subplot(1,1,1)` and exists only for compatibility with previous releases. This syntax does not immediately create an axes, but instead sets up the figure so that the next graphics command executes a `clf reset` (deleting all figure children) and creates a new axes in the default position. This syntax does not return a handle, so it is an error to specify a return argument. (This behavior is implemented by setting the figure's NextPlot property to replace.)

Examples

To plot income in the top half of a figure and outgo in the bottom half,

```
income = [3.2 4.1 5.0 5.6];
outgo = [2.5 4.0 3.35 4.9];
subplot(2,1,1); plot(income)
subplot(2,1,2); plot(outgo)
```

The following illustration shows four subplot regions and indicates the command used to create each.



See Also

axes, cla, clf, figure, gca

subvolume

Purpose

Extract subset of volume data set

Syntax

```
[Nx,Ny,Nz,Nv] = subvolume(X,Y,Z,V,limits)
[Nx,Ny,Nz,Nv] = subvolume(V,limits)
Nv = subvolume(...)
```

Description

`[Nx,Ny,Nz,Nv] = subvolume(X,Y,Z,V,limits)` extracts a subset of the volume data set `V` using the specified axis-aligned `limits`. `limits = [xmin,xmax,ymin, ymax,zmin,zmax]` (Any `NaN`s in the limits indicate that the volume should not be cropped along that axis).

The arrays `X`, `Y`, and `Z` define the coordinates for the volume `V`. The subvolume is returned in `NV` and the coordinates of the subvolume are given in `NX`, `NY`, and `NZ`.

`[Nx,Ny,Nz,Nv] = subvolume(V,limits)` assumes the arrays `X`, `Y`, and `Z` are defined as `[X,Y,Z] = meshgrid(1:N,1:M,1:P)` where `[M,N,P] = size(V)`.

`Nv = subvolume(...)` returns only the subvolume.

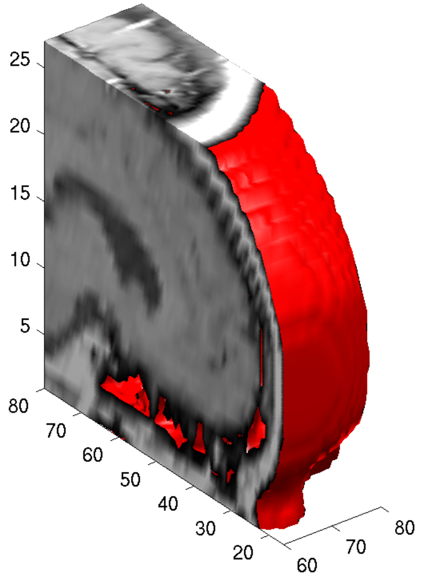
Examples

This example uses a data set that is a collection of MRI slices of a human skull. The data is processed in a variety of ways:

- The 4-D array is squeezed (`squeeze`) into three dimensions and then a subset of the data is extracted (`subvolume`).
- The outline of the skull is an isosurface generated as a patch (`p1`) whose vertex normals are recalculated to improve the appearance when lighting is applied (`patch`, `isosurface`, `isonormals`).
- A second patch (`p2`) with interpolated face color draws the end caps (`FaceColor`, `isocaps`).
- The view of the object is set (`view`, `axis`, `daspect`).

- A 100-element grayscale colormap provides coloring for the end caps (`colormap`).
- Adding lights to the right and left of the camera illuminates the object (`camlight`, `lighting`).

```
load mri
D = squeeze(D);
[x,y,z,D] = subvolume(D,[60,80,nan,80,nan,nan]);
p1 = patch(isosurface(x,y,z,D, 5),...
     'FaceColor','red','EdgeColor','none');
isonormals(x,y,z,D,p1);
p2 = patch(isocaps(x,y,z,D, 5),...
     'FaceColor','interp','EdgeColor','none');
view(3); axis tight; daspect([1,1,.4])
colormap(gray(100))
camlight right; camlight left; lighting gouraud
```



See Also `isocaps`, `isonormals`, `isosurface`, `reducepatch`, `reducevolume`, `smooth3`

surf, surfc

Purpose

3-D shaded surface plot

Syntax

```
surf(Z)
surf(X,Y,Z)
surf(X,Y,Z,C)
surf(...,'PropertyName',PropertyValue)
surfc(...)
h = surf(...)
h = surfc(...)
```

Description

Use `surf` and `surfc` to view mathematical functions over a rectangular region. `surf` and `surfc` create colored parametric surfaces specified by `X`, `Y`, and `Z`, with color specified by `Z` or `C`.

`surf(Z)` creates a a three-dimensional shaded surface from the *z* components in matrix `Z`, using `x = 1:n` and `y = 1:m`, where `[m,n] = size(Z)`. The height, `Z`, is a single-valued function defined over a geometrically rectangular grid. `Z` specifies the color data as well as surface height, so color is proportional to surface height.

`surf(X,Y,Z)` creates a shaded surface using `Z` for the color data as well as surface height. `X` and `Y` are vectors or matrices defining the *x* and *y* components of a surface. If `X` and `Y` are vectors, `length(X) = n` and `length(Y) = m`, where `[m,n] = size(Z)`. In this case, the vertices of the surface faces are $(X(j), Y(i), Z(i, j))$ triples.

`surf(X,Y,Z,C)` creates a shaded surface, with color defined by `C`. MATLAB performs a linear transformation on this data to obtain colors from the current colormap.

`surf(...,'`*`PropertyName`*`',PropertyValue)` specifies surface properties along with the data.

`surfc(...)` draws a contour plot beneath the surface.

`h = surf(...)` and `h = surfc(...)` return a handle to a surface graphics object.

Algorithm

Abstractly, a parametric surface is parametrized by two independent variables, i and j, which vary continuously over a rectangle; for example, 1 ≤ i ≤ m and 1 ≤ j ≤ n. The three functions, x(i,j), y(i,j), and z(i,j), specify the surface. When i and j are integer values, they define a rectangular grid with integer grid points. The functions x(i,j), y(i,j), and z(i,j) become three m-by-n matrices, X, Y and Z. surface color is a fourth function, c(i,j), denoted by matrix C.

Each point in the rectangular grid can be thought of as connected to its four nearest neighbors.

```
         i–1,j
           |
i,j–1 – i,j – i,j+1
           |
         i+1,j
```

This underlying rectangular grid induces four-sided patches on the surface. To express this another way, [X(:) Y(:) Z(:)] returns a list of triples specifying points in 3-space. Each interior point is connected to the four neighbors inherited from the matrix indexing. Points on the edge of the surface have three neighbors; the four points at the corners of the grid have only two neighbors. This defines a mesh of quadrilaterals or a *quad-mesh*.

Surface color can be specified in two different ways – at the vertices or at the centers of each patch. In this general setting, the surface need not be a single-valued function of x and y. Moreover, the four-sided surface patches need not be planar. For example, you can have surfaces defined in polar, cylindrical, and spherical coordinate systems.

The shading function sets the shading. If the shading is interp, C must be the same size as X, Y, and Z; it specifies the colors at the vertices. The color within a surface patch is a bilinear function of the local coordinates. If the shading is faceted (the default) or flat, C(i,j) specifies the constant color in the surface patch:

```
  (i,j)    –   (i,j+1)
    |    C(i,j)   |
 (i+1,j)  –  (i+1,j+1)
```

In this case, C can be the same size as X, Y, and Z and its last row and column are ignored, Alternatively, its row and column dimensions can be one less than those of X, Y, and Z.

The surf and surfc functions specify the view point using view(3).

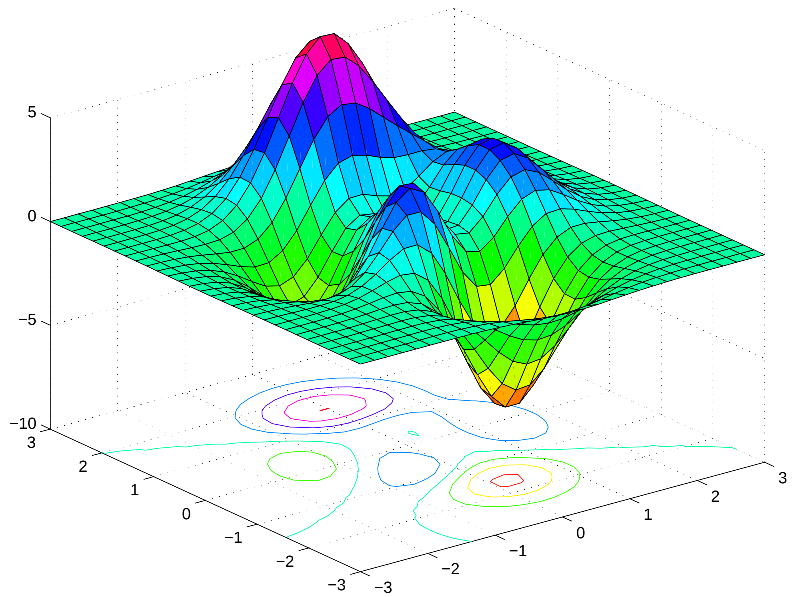
The range of X, Y, and Z, or the current setting of the axes XLimMode, YLimMode, and ZLimMode properties (also set by the axis function) determine the axis labels.

The range of C, or the current setting of the axes CLim and ClimMode properties (also set by the caxis function) determine the color scaling. The scaled color values are used as indices into the current colormap.

Examples

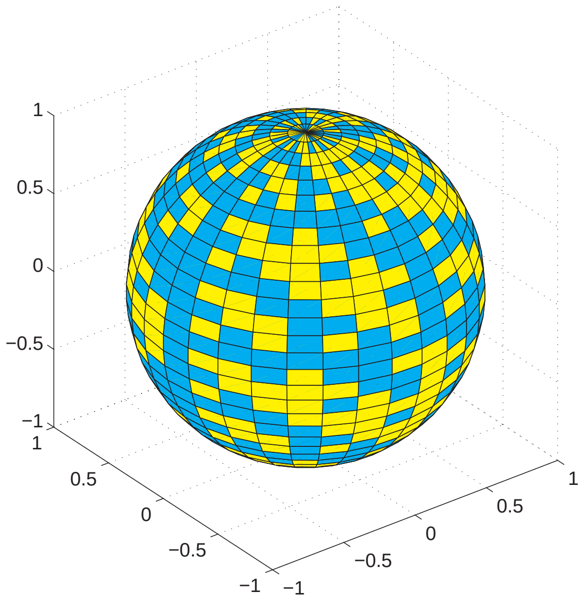
Display a surface and contour plot of the peaks surface.

```
[X,Y,Z] = peaks(30);
surfc(X,Y,Z)
colormap hsv
axis([−3 3 −3 3 −10 5])
```



Color a sphere with the pattern of +1s and -1s in a Hadamard matrix.

```
k = 5;
n = 2^k-1;
[x,y,z] = sphere(n);
c = hadamard(2^k);
surf(x,y,z,c);
colormap([1  1  0; 0  1  1])
axis equal
```



See Also

axis, caxis, colormap, contour, mesh, pcolor, shading, view

Properties for surface graphics objects

Purpose Convert surface data to patch data

Syntax

```
fvc = surf2patch(h)
fvc = surf2patch(Z)
fvc = surf2patch(Z,C)
fvc = surf2patch(X,Y,Z)
fvc = surf2patch(X,Y,Z,C)
fvc = surf2patch(...,'triangles')
[f,v,c] = surf2patch(...)
```

Description

`fvc = surf2patch(h)` converts the geometry and color data from the `surface` object identified by the handle `h` into patch format and returns the face, vertex, and color data in the struct `fvc`. You can pass this struct directly to the `patch` command.

`fvc = surf2patch(Z)` calculates the patch data from the surface's `ZData` matrix `Z`.

`fvc = surf2patch(Z,C)` calculates the patch data from the surface's `ZData` and `CData` matrices `Z` and `C`.

`fvc = surf2patch(X,Y,Z)` calculates the patch data from the surface's `XData`, `YData`, and `ZData` matrices `X`, `Y`, and `Z`.

`fvc = surf2patch(X,Y,Z,C)` calculates the patch data from the surface's `XData`, `YData`, `ZData`, and `CData` matrices `X`, `Y`, `Z`, and `C`.

`fvc = surf2patch(...,'triangles')` creates triangular faces instead of the quadrilaterals that compose surfaces.

`[f,v,c] = surf2patch(...)` returns the face, vertex, and color data in the three arrays `f`, `v`, and `c` instead of a struct.

Examples

The first example uses the `sphere` command to generate the `XData`, `YData`, and `ZData` of a surface, which is then converted to a patch. Note that the `ZData` (`z`) is passed to `surf2patch` as both the third and fourth arguments – the third argument is the `ZData` and the fourth argument is taken as the `CData`. This is because the `patch` command does not automatically use the z-coordinate data for the color data, as does the `surface` command.

Also, because patch is a low-level command, you must set the view to 3-D and shading to faceted to produce the same results produced by the surf command.

```
[x y z] = sphere;
patch(surf2patch(x,y,z,z));
shading faceted; view(3)
```

In the second example surf2patch calculates face, vertex, and color data from a surface whose handle has been passed as an argument.

```
s = surf(peaks);
pause
patch(surf2patch(s));
delete(s)
shading faceted; view(3)
```

See Also

patch, reducepatch, shrinkfaces, surface, surf

surface

Purpose Create surface object

Syntax

```
surface(Z)
surface(Z,C)
surface(X,Y,Z)
surface(X,Y,Z,C)
surface(...'PropertyName',PropertyValue,...)
h = surface(...)
```

Description

`surface` is the low-level function for creating surface graphics objects. surfaces are plots of matrix data created using the row and column indices of each element as the *x*- and *y*-coordinates and the value of each element as the z-coordinate.

`surface(Z)` plots the surface specified by the matrix `Z`. Here, `Z` is a single-valued function, defined over a geometrically rectangular grid.

`surface(Z,C)` plots the surface specified by `Z` and colors it according to the data in `C` (see “Examples”).

`surface(X,Y,Z)` uses `C = Z`, so color is proportional to surface height above the *x*-*y* plane.

`surface(X,Y,Z,C)` plots the parametric surface specified by `X`, `Y` and `Z`, with color specified by `C`.

`surface(x,y,Z)`, `surface(x,y,Z,C)` replaces the first two matrix arguments with vectors and must have `length(x) = n` and `length(y) = m` where `[m,n] = size(Z)`. In this case, the vertices of the surface facets are the triples `(x(j),y(i),Z(i,j))`. Note that `x` corresponds to the columns of `Z` and `y` corresponds to the rows of `Z`. For a complete discussion of parametric surfaces, see the `surf` function.

`surface(...'PropertyName',PropertyValue,...)` follows the `X`, `Y`, `Z`, and `C` arguments with property name/property value pairs to specify additional surface properties. These properties are described in the “Surface Properties” section.

`h = surface(...)` returns a handle to the created surface object.

Remarks

Unlike high-level area creation functions, such as `surf` or `mesh`, `surface` does not respect the settings of the figure and axes `NextPlot` properties. It simply adds the surface object to the current axes.

If you do not specify separate color data (`C`), MATLAB uses the matrix (`Z`) to determine the coloring of the surface. In this case, color is proportional to values of `Z`. You can specify a separate matrix to color the surface independently of the data defining the area of the surface.

You can specify properties as property name/property value pairs, structure arrays, and cell arrays (see `set` and `get` for examples of how to specify these data types).

`surface` provides convenience forms that allow you to omit the property name for the `XData`, `YData`, `ZData`, and `CData` properties. For example,

```
surface('XData',X,'YData',Y,'ZData',Z,'CData',C)
```

is equivalent to:

```
surface(X,Y,Z,C)
```

When you specify only a single matrix input argument,

```
surface(Z)
```

MATLAB assigns the data properties as if you specified,

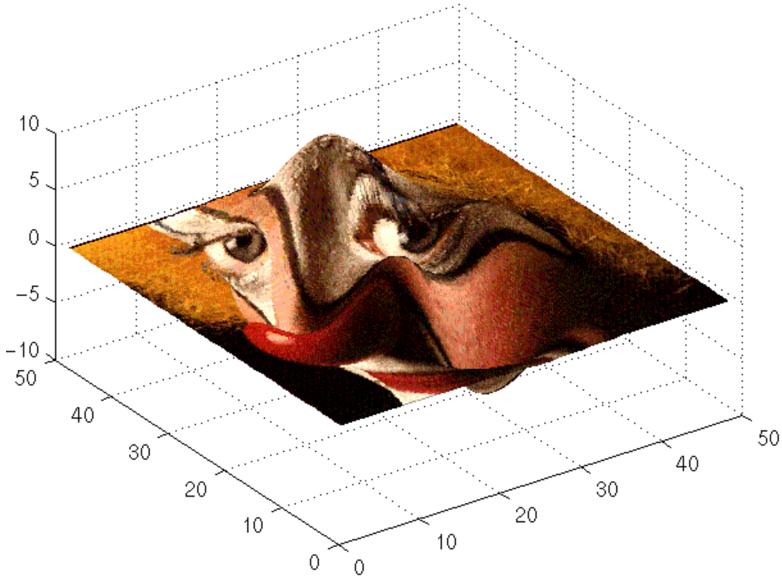
```
surface('XData',[1:size(Z,2)],...
        'YData',[1:size(Z,1)],...
        'ZData',Z,...
        'CData',Z)
```

The `axis`, `caxis`, `colormap`, `hold`, `shading`, and `view` commands set graphics properties that affect surfaces. You can also set and query surface property values after creating them using the `set` and `get` commands.

Example

This example creates a surface using the `peaks` M-file to generate the data, and colors it using the clown image. The `ZData` is a 49-by-49 element matrix, while the CData is a 200-by-320 matrix. You must set the surface's FaceColor to texturemap to use ZData and CData of different dimensions.

```
load clown
surface(peaks,flipud(X),...
        'FaceColor','texturemap',...
        'EdgeColor','none',...
        'CDataMapping','direct')
colormap(map)
view(-35,45)
```



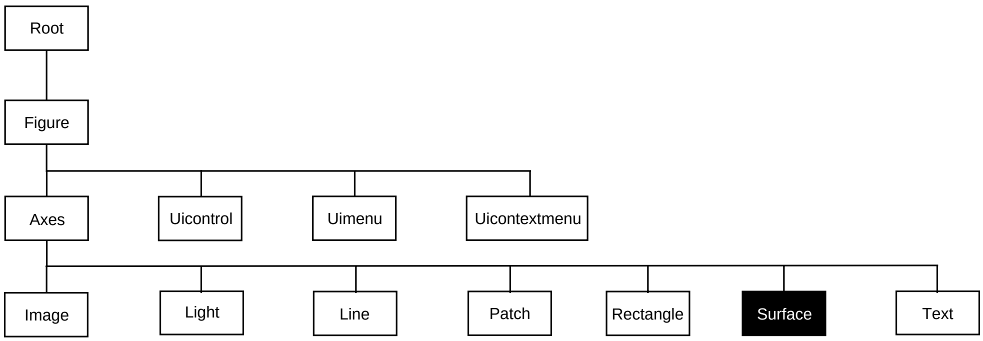
Note the use of the surface(Z,C) convenience form combined with property name/property value pairs.

Since the clown data (X) is typically viewed with the image command, which MATLAB normally displays with 'ij' axis numbering and direct CDataMapping, this example reverses the data in the vertical direction using flipud and sets the CDataMapping property to direct.

See Also

ColorSpec, mesh, patch, pcolor, surf

Object Hierarchy



Setting Default Properties

You can set default surface properties on the axes, figure, and root levels.

```
set(0,'DefaultSurfaceProperty',PropertyValue...)
set(gcf,'DefaultSurfaceProperty',PropertyValue...)
set(gca,'DefaultSurfaceProperty',PropertyValue...)
```

Where *Property* is the name of the surface property whose default value you want to set and `PropertyValue` is the value you are specifying. Use `set` and `get` to access the surface properties.

Property List

The following table lists all surface properties and provides a brief description of each. The property name links take you to an expanded description of the properties.

Property Name	Property Description	Property Value
Data Defining the Object		
`XData`	The *x*-coordinates of the vertices of the surface	Values: vector or matrix
`YData`	The *y*-coordinates of the vertices of the surface	Values: vector or matrix

Property Name	Property Description	Property Value
ZData	The *z*-coordinates of the vertices of the surface	Values: matrix
Specifying Color		
CData	Color data	Values: scalar, vector, or matrix Default: [] empty matrix
CDataMapping	Controls mapping of CData to colormap	Values: scaled, direct Default: scaled
EdgeColor	Color of face edges	Values: ColorSpec, none, flat, interp Default: ColorSpec
FaceColor	Color of face	Values: ColorSpec, none, flat, interp Default: ColorSpec
MarkerEdgeColor	Color of marker or the edge color for filled markers	Values: ColorSpec, none, auto Default: auto
MarkerFaceColor	Fill color for markers that are closed shapes	Values: ColorSpec, none, auto Default: none
Controlling the Effects of Lights		
AmbientStrength	Intensity of the ambient light	Values: scalar >=0 and <=1 Default: 0.3
BackFaceLighting	Controls lighting of faces pointing away from camera	Values: unlit, lit, reverselit Default: reverselit
DiffuseStrength	Intensity of diffuse light	Values: scalar >=0 and <=1 Default: 0.6

Property Name	Property Description	Property Value
EdgeLighting	Method used to light edges	Values: none, flat, gouraud, phong Default: none
FaceLighting	Method used to light edges	Values: none, flat, gouraud, phong Default: none
NormalMode	MATLAB-generated or user-specified normal vectors	Values: auto, manual Default: auto
SpecularColorReflectance	Composite color of specularly reflected light	Values: scalar 0 to 1 Default: 1
SpecularExponent	Harshness of specular reflection	Values: scalar >= 1 Default: 10
SpecularStrength	Intensity of specular light	Values: scalar >=0 and <=1 Default: 0.9
VertexNormals	Vertex normal vectors	Values: matrix
Defining Edges and Markers		
LineStyle	Select from five line styles.	Values: –, ––, :, –., none Default: –
LineWidth	The width of the edge in points	Values: scalar Default: 0.5 points
Marker	Marker symbol to plot at data points	Values: see Marker property Default: none
MarkerSize	Size of marker in points	Values: size in points Default: 6
Controlling the Appearance		
Clipping	Clipping to axes rectangle	Values: on, off Default: on

Property Name	Property Description	Property Value
EraseMode	Method of drawing and erasing the surface (useful for animation)	Values: normal, none, xor, background Default: normal
MeshStyle	Specifies whether to draw all edge lines or just row or column edge lines	Values: both, row, column Defaults: both
SelectionHighlight	Highlight surface when selected (Selected property set to on)	Values: on, off Default: on
Visible	Make the surface visible or invisible	Values: on, off Default: on
Controlling Access to Objects		
HandleVisibility	Determines if and when the the surface's handle is visible to other functions	Values: on, callback, off Default: on
HitTest	Determines if the surface can become the current object (see the figure CurrentObject property)	Values: on, off Default: on
Properties Related to Callback Routine Execution		
BusyAction	Specifies how to handle callback routine interruption	Values: cancel, queue Default: queue
ButtonDownFcn	Defines a callback routine that executes when a mouse button is pressed on over the surface	Values: string Default: '' (empty string)
CreateFcn	Defines a callback routine that executes when an surface is created	Values: string Default: '' (empty string)
DeleteFcn	Defines a callback routine that executes when the surface is deleted (via close or delete)	Values: string Default: '' (empty string)

Property Name	Property Description	Property Value
Interruptible	Determines if callback routine can be interrupted	Values: on, off Default: on (can be interrupted)
UIContextMenu	Associates a context menu with the surface	Values: handle of a uicontextmenu
General Information About the Surface		
Children	Surface objects have no children	Values: [] (empty matrix)
Parent	The parent of a surface object is always an axes object	Value: axes handle
Selected	Indicates whether the surface is in a “selected” state.	Values: on, off Default: on
Tag	User-specified label	Value: any string Default: '' (empty string)
Type	The type of graphics object (read only)	Value: the string 'surface'
UserData	User-specified data	Values: any matrix Default: [] (empty matrix)

Surface Properties

This section lists property names along with the types of values each accepts. Curly braces { } enclose default values.

AmbientStrength scalar >= 0 and <= 1

Strength of ambient light. This property sets the strength of the ambient light, which is a nondirectional light source that illuminates the entire scene. You must have at least one visible light object in the axes for the ambient light to be visible. The axes `AmbientLightColor` property sets the color of the ambient light, which is therefore the same on all objects in the axes.

You can also set the strength of the diffuse and specular contribution of light objects. See the surface `DiffuseStrength` and `SpecularStrength` properties.

BackFaceLighting `unlit | lit | reverselit`

Face lighting control. This property determines how faces are lit when their vertex normals point away from the camera.

- `unlit` – face is not lit
- `lit` – face lit in normal way
- `reverselit` – face is lit as if the vertex pointed towards the camera

This property is useful for discriminating between the internal and external surfaces of an object. See the *Using MATLAB Graphics* manual for an example.

BusyAction `cancel | {queue}`

Callback routine interruption. The `BusyAction` property enables you to control how MATLAB handles events that potentially interrupt executing callback routines. If there is a callback routine executing, subsequently invoked callback routines always attempt to interrupt it. If the `Interruptible` property of the object whose callback is executing is set to `on` (the default), then interruption occurs at the next point where the event queue is processed. If the `Interruptible` property is `off`, the `BusyAction` property (of the object owning the executing callback) determines how MATLAB handles the event. The choices are:

- `cancel` – discard the event that attempted to execute a second callback routine.
- `queue` – queue the event that attempted to execute a second callback routine until the current callback finishes.

ButtonDownFcn string

Button press callback routine. A callback routine that executes whenever you press a mouse button while the pointer is over the surface object. Define this routine as a string that is a valid MATLAB expression or the name of an M-file. The expression executes in the MATLAB workspace.

CData matrix

Vertex colors. A matrix containing values that specify the color at every point in `ZData`. If you set the `FaceColor` property to `texturemap`, `CData` does not need to be the same size as `ZData`. In this case, MATLAB maps `CData` to conform to the surface defined by `ZData`.

You can specify color as indexed values or true color. Indexed color data specifies a single value for each vertex. These values are either scaled to map linearly into the current colormap (see `caxis`) or interpreted directly as indices into the colormap, depending on the setting of the `CDataMapping` property.

True color defines an RGB value for each vertex. If the coordinate data (`XData` for example) are contained in *m*-by-*n* matrices, then `CData` must be an *m*-by-*n*-3 array. The first page contains the red components, the second the green components, and the third the blue components of the colors.

On computer displays that cannot display true color (e.g., 8-bit displays), MATLAB uses dithering to approximate the RGB triples using the colors in the figure's `Colormap` and `Dithermap`. By default, `Dithermap` uses the `colorcube(64)` colormap. You can also specify your own dithermap.

CDataMapping `{scaled} | direct`

Direct or scaled color mapping. This property determines how MATLAB interprets indexed color data used to color the surface. (If you use true color specification for `CData`, this property has no effect.)

- `scaled` – transform the color data to span the portion of the colormap indicated by the axes `CLim` property, linearly mapping data values to colors. See the `caxis` reference page for more information on this mapping.
- `direct` – use the color data as indices directly into the colormap. The color data should then be integer values ranging from 1 to `length(colormap)`. MATLAB maps values less than 1 to the first color in the colormap, and values greater than `length(colormap)` to the last color in the colormap. Values with a decimal portion are fixed to the nearest, lower integer.

Children matrix of handles

Always the empty matrix; surface objects have no children.

Clipping {on} | off

Clipping to axes rectangle. When Clipping is on, MATLAB does not display any portion of the surface that is outside the axes rectangle.

CreateFcn string

Callback routine executed during object creation. This property defines a callback routine that executes when MATLAB creates a surface object. You must define this property as a default value for surfaces. For example, the statement,

```
set(0,'DefaultSurfaceCreateFcn',...
      'set(gcf,''DitherMap'',my_dithermap)')
```

defines a default value on the root level that sets the figure DitherMap property whenever you create a surface object. MATLAB executes this routine after setting all surface properties. Setting this property on an existing surface object has no effect.

The handle of the object whose CreateFcn is being executed is accessible only through the root CallbackObject property, which you can query using gcbo.

DeleteFcn string

Delete surface callback routine. A callback routine that executes when you delete the surface object (e.g., when you issue a delete command or clear the axes or figure). MATLAB executes the routine before destroying the object's properties so these values are available to the callback routine.

The handle of the object whose DeleteFcn is being executed is accessible only through the root CallbackObject property, which you can query using gcbo.

DiffuseStrength scalar >= 0 and <= 1

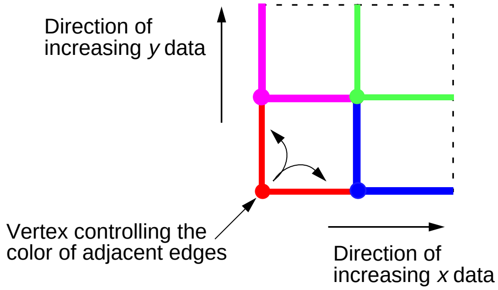
Intensity of diffuse light. This property sets the intensity of the diffuse component of the light falling on the surface. Diffuse light comes from light objects in the axes.

You can also set the intensity of the ambient and specular components of the light on the surface object. See the AmbientStrength and SpecularStrength properties.

EdgeColor `{ColorSpec} | none | flat | interp`

Color of the surface edge. This property determines how MATLAB colors the edges of the individual faces that make up the surface:

- `ColorSpec` — A three-element RGB vector or one of MATLAB's predefined names, specifying a single color for edges. The default `EdgeColor` is black. See `ColorSpec` for more information on specifying color.
- `none` — Edges are not drawn.
- `flat` — The `CData` value of the first vertex for a face determines the color of each edge.



- `interp` — Linear interpolation of the `CData` values at the face vertices determines the edge color.

EdgeLighting `{none} | flat | gouraud | phong`

Algorithm used for lighting calculations. This property selects the algorithm used to calculate the effect of light objects on surface edges. Choices are:

- `none` – Lights do not affect the edges of this object.
- `flat` – The effect of light objects is uniform across each edge of the surface.
- `gouraud` – The effect of light objects is calculated at the vertices and then linearly interpolated across the edge lines.
- `phong` – The effect of light objects is determined by interpolating the vertex normals across each edge line and calculating the reflectance at each pixel. Phong lighting generally produces better results than Gouraud lighting, but takes longer to render.

EraseMode `{normal} | none | xor | background`

Erase mode. This property controls the technique MATLAB uses to draw and erase surface objects. Alternative erase modes are useful for creating animated sequences, where control of the way individual objects redraw is necessary to improve performance and obtain the desired effect.

- `normal` — Redraw the affected region of the display, performing the three-dimensional analysis necessary to ensure that all objects are rendered correctly. This mode produces the most accurate picture, but is the slowest. The other modes are faster, but do not perform a complete redraw and are therefore less accurate.
- `none` — Do not erase the surface when it is moved or destroyed. While the object is still visible on the screen after erasing with `EraseMode` `none`, you cannot print it because MATLAB stores no information about its former location.
- `xor` — Draw and erase the surface by performing an exclusive OR (XOR) with each pixel index of the screen behind it. Erasing the surface does not damage the color of the objects behind it. However, surface color depends on the color of the screen behind it and is correctly colored only when over the axes background `Color`, or the figure background `Color` if the axes `Color` is set to `none`.
- `background` — Erase the surface by drawing it in the axes' background `Color`, or the figure background `Color` if the axes `Color` is set to `none`. This damages objects that are behind the erased object, but surface objects are always properly colored.

Printing with Non-normal Erase Modes. MATLAB always prints figures as if the `EraseMode` of all objects is `normal`. This means graphics objects created with `EraseMode` set to `none`, `xor`, or `background` can look different on screen than on paper. On screen, MATLAB may mathematically combine layers of colors (e.g., XORing a pixel color with that of the pixel behind it) and ignore three-dimensional sorting to obtain greater rendering speed. However, these techniques are not applied to the printed output.

You can use the MATLAB `getframe` command or other screen capture application to create an image of a figure containing non-normal mode objects.

FaceColor `ColorSpec | none | {flat} | interp`

Color of the surface face. This property can be any of the following:

- `ColorSpec` — A three-element RGB vector or one of MATLAB's predefined names, specifying a single color for faces. See `ColorSpec` for more information on specifying color.
- `none` — Do not draw faces. Note that edges are drawn independently of faces.
- `flat` — The values of `CData` determine the color for each face of the surface. The color data at the first vertex determines the color of the entire face.
- `interp` — Bilinear interpolation of the values at each vertex (the `CData`) determines the coloring of each face.
- `texturemap` — Texture map the `CData` to the surface. MATLAB transforms the color data so that it conforms to the surface. (See the texture mapping example.)

FaceLighting `{none} | flat | gouraud | phong`

Algorithm used for lighting calculations. This property selects the algorithm used to calculate the effect of light objects on the surface. Choices are:

- `none` – Lights do not affect the faces of this object.
- `flat` – The effect of light objects is uniform across the faces of the surface. Select this choice to view faceted objects.
- `gouraud` – The effect of light objects is calculated at the vertices and then linearly interpolated across the faces. Select this choice to view curved surfaces.
- `phong` – The effect of light objects is determined by interpolating the vertex normals across each face and calculating the reflectance at each pixel. Select this choice to view curved surfaces. Phong lighting generally produces better results than Gouraud lighting, but takes longer to render.

HandleVisibility `{on} | callback | off`

Control access to object's handle by command-line users and GUIs. This property determines when an object's handle is visible in its parent's list of children. This property is useful for preventing command-line users from accidentally drawing into or deleting a figure that contains only user interface devices (such as a dialog box).

Handles are always visible when `HandleVisibility` is `on`.

Setting `HandleVisibility` to `callback` causes handles to be visible from within callback routines or functions invoked by callback routines, but not from within functions invoked from the command line. This provides a means to protect GUIs from command-line users, while allowing callback routines to have complete access to object handles.

Setting `HandleVisibility` to `off` makes handles invisible at all times. This may be necessary when a callback routine invokes a function that might potentially damage the GUI (such as evaluating a user-typed string), and so temporarily hides its own handles during the execution of that function.

When a handle is not visible in its parent's list of children, it cannot be returned by functions that obtain handles by searching the object hierarchy or querying handle properties. This includes `get`, `findobj`, `gca`, `gcf`, `gco`, `newplot`, `cla`, `clf`, and `close`.

When a handle's visibility is restricted using `callback` or `off`, the object's handle does not appear in its parent's `Children` property, figures do not appear in the root's `CurrentFigure` property, objects do not appear in the root's `CallbackObject` property or in the figure's `CurrentObject` property, and axes do not appear in their parent's `CurrentAxes` property.

You can set the root `ShowHiddenHandles` property to `on` to make all handles visible, regardless of their `HandleVisibility` settings (this does not affect the values of the `HandleVisibility` properties).

Handles that are hidden are still valid. If you know an object's handle, you can `set` and `get` its properties, and pass it to any function that operates on handles.

`HitTest` `{on} | off`

Selectable by mouse click. `HitTest` determines if the surface can become the current object (as returned by the `gco` command and the figure `CurrentObject` property) as a result of a mouse click on the surface. If `HiTest` is `off`, clicking on the surface selects the object below it (which maybe the axes containing it).

`Interruptible` `{on} | off`

Callback routine interruption mode. The `Interruptible` property controls whether a surface callback routine can be interrupted by subsequently invoked callback routines. Only callback routines defined for the `ButtonDownFcn` are affected by the `Interruptible` property. MATLAB checks for events that can interrupt a callback routine only when it encounters a `drawnow`, `figure`, getframe, or pause command in the routine. See the BusyAction property for related information.

LineStyle {-} | -- | : | -. | none

Edge line type. This property determines the line style used to draw surface edges. The available line styles are shown in this table.

Symbol	Line Style
–	solid line (default)
––	dashed line
:	dotted line
–.	dash-dot line
none	no line

LineWidth scalar

Edge line width. The width of the lines in points used to draw surface edges. The default width is 0.5 points (1 point = 1/72 inch).

Marker marker symbol (see table)

Marker symbol. The Marker property specifies symbols that display at vertices. You can set values for the Marker property independently from the LineStyle property.

You can specify these markers.

Marker Specifier	Description
+	plus sign
o	circle
*	asterisk
.	point
x	cross

Marker Specifier	Description
s	square
d	diamond
^	upward pointing triangle
v	downward pointing triangle
>	right pointing triangle
<	left pointing triangle
p	five-pointed star (pentagram)
h	six-pointed star (hexagram)
none	no marker (default)

MarkerEdgeColor `ColorSpec | none | {auto}`

Marker edge color. The color of the marker or the edge color for filled markers (circle, square, diamond, pentagram, hexagram, and the four triangles).

- `ColorSpec` defines a single color to use for the edge (see `ColorSpec` for more information).
- `none` specifies no color, which makes nonfilled markers invisible.
- `auto` uses the same color as the `EdgeColor` property.

MarkerFaceColor `ColorSpec | {none} | auto`

Marker face color. The fill color for markers that are closed shapes (circle, square, diamond, pentagram, hexagram, and the four triangles).

- `ColorSpec` defines a single color to use for all marker on the surface (see `ColorSpec` for more information).
- `none` makes the interior of the marker transparent, allowing the background to show through.
- `auto` uses the `CData` for the vertex located by the marker to determine the color.

MarkerSize size in points

Marker size. A scalar specifying the marker size, in points. The default value for MarkerSize is six points (1 point = 1/72 inch). Note that MATLAB draws the point marker at 1/3 the specified marker size.

MeshStyle {both} | row | column

Row and column lines. This property specifies whether to draw all edge lines or just row or column edge lines.

- both draws edges for both rows and columns.
- row draws row edges only.
- column draws column edges only.

NormalMode {auto} | manual

MATLAB -generated or user-specified normal vectors. When this property is auto, MATLAB calculates vertex normals based on the coordinate data. If you specify your own vertex normals, MATLAB sets this property to manual and does not generate its own data. See also the VertexNormals property.

Parent handle

Surface's parent object. The parent of a surface object is the axes in which it is displayed. You can move a surface object to another axes by setting this property to the handle of the new parent.

Selected on | {off}

Is object selected? When this property is on, MATLAB displays a dashed bounding box around the surface if the SelectionHighlight property is also on. You can, for example, define the ButtonDownFcn to set this property, allowing users to select the object with the mouse.

SelectionHighlight {on} | off

Objects highlight when selected. When the Selected property is on, MATLAB indicates the selected state by drawing a dashed bounding box around the surface. When SelectionHighlight is off, MATLAB does not draw the handles.

SpecularColorReflectance scalar in the range 0 to 1

Color of specularly reflected light. When this property is 0, the color of the specularly reflected light depends on both the color of the object from which it reflects and the color of the light source. When set to 1, the color of the specularly reflected light depends only on the color or the light source (i.e., the light object `Color` property). The proportions vary linearly for values in between.

SpecularExponent scalar >= 1

Harshness of specular reflection. This property controls the size of the specular spot. Most materials have exponents in the range of 5 to 20.

SpecularStrength scalar >= 0 and <= 1

Intensity of specular light. This property sets the intensity of the specular component of the light falling on the surface. Specular light comes from light objects in the axes.

You can also set the intensity of the ambient and diffuse components of the light on the surface object. See the `AmbientStrength` and `DiffuseStrength` properties. Also see the `material` function.

Tag string

User-specified object label. The `Tag` property provides a means to identify graphics objects with a user-specified label. This is particularly useful when constructing interactive graphics programs that would otherwise need to define object handles as global variables or pass them as arguments between callback routines. You can define `Tag` as any string.

Type string (read only)

Class of the graphics object. The class of the graphics object. For surface objects, `Type` is always the string `'surface'`.

UIContextMenu handle of a uicontextmenu object

Associate a context menu with the surface. Assign this property the handle of a uicontextmenu object created in the same figure as the surface. Use the `uicontextmenu` function to create the context menu. MATLAB displays the context menu whenever you right-click over the surface.

UserData matrix

User-specified data. Any matrix you want to associate with the surface object. MATLAB does not use this data, but you can access it using the `set` and `get` commands.

VertexNormals vector or matrix

Surface normal vectors. This property contains the vertex normals for the surface. MATLAB generates this data to perform lighting calculations. You can supply your own vertex normal data, even if it does not match the coordinate data. This can be useful to produce interesting lighting effects.

Visible `{on} | off`

Surface object visibility. By default, all surfaces are visible. When set to `off`, the surface is not visible, but still exists and you can query and set its properties.

XData vector or matrix

X-coordinates. The *x*-position of the surface points. If you specify a row vector, `surface` replicates the row internally until it has the same number of columns as `ZData`.

YData vector or matrix

Y-coordinates. The *y*-position of the surface points. If you specify a row vector, `surface` replicates the row internally until it has the same number of rows as `ZData`.

ZData matrix

Z-coordinates. *Z*-position of the surface points. See the Description section for more information.

surfl

Purpose Surface plot with colormap-based lighting

Syntax

```
surfl(Z)
surfl(X,Y,Z)
surfl(...,'light')
surfl(...,s)
surfl(X,Y,Z,s,k)
h = surfl(...)
```

Description The `surfl` function displays a shaded surface based on a combination of ambient, diffuse, and specular lighting models.

`surfl(Z)` and `surfl(X,Y,Z)` create three-dimensional shaded surfaces using the default direction for the light source and the default lighting coefficients for the shading model. `X`, `Y`, and `Z` are vectors or matrices that define the *x*, *y*, and *z* components of a surface.

`surfl(...,'light')` produces a colored, lighted surface using a MATLAB light object. This produces results different from the default lighting method, surfl(...,'cdata'), which changes the color data for the surface to be the reflectance of the surface.

`surfl(...,s)` specifies the direction of the light source. `s` is a two- or three-element vector that specifies the direction from a surface to a light source. `s = [sx sy sz]` or `s = [azimuth elevation]`. The default `s` is 45° counterclockwise from the current view direction.

`surfl(X,Y,Z,s,k)` specifies the reflectance constant. `k` is a four-element vector defining the relative contributions of ambient light, diffuse reflection, specular reflection, and the specular shine coefficient. `k = [ka kd ks shine]` and defaults to `[.55,.6,.4,10]`.

`h = surfl(...)` returns a handle to a surface graphics object.

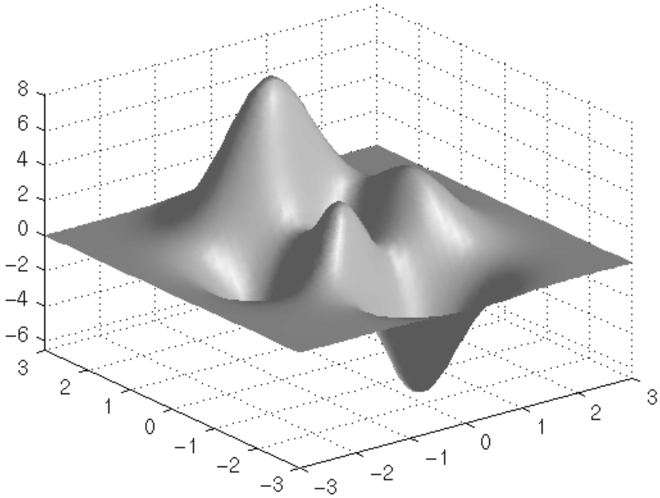
Remarks For smoother color transitions, use colormaps that have linear intensity variations (e.g., `gray`, `copper`, `bone`, `pink`).

The ordering of points in the `X`, `Y`, and `Z` matrices define the inside and outside of parametric surfaces. If you want the opposite side of the surface to reflect the light source, use `surfl(X',Y',Z')`. Because of the way surface normal vectors are computed, `surfl` requires matrices that are at least 3-by-3.

Examples

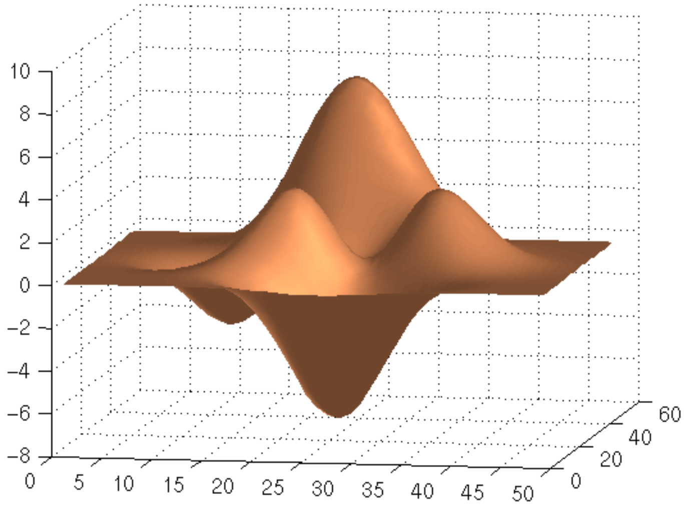
View peaks using colormap-based lighting.

```
[x,y] = meshgrid(-3:1/8:3);
z = peaks(x,y);
surfl(x,y,z);
shading interp
colormap(gray);
axis([-3  3  -3  3  -8  8])
```



To plot a lighted surface from a view direction other than the default.

```
view([10 10])
grid on
hold on
surfl(peaks)
shading interp
colormap copper
hold off
```



See Also colormap, shading, light

Purpose

Compute and display 3-D surface normals

Syntax

```
surfnorm(Z)
surfnorm(X,Y,Z)
[Nx,Ny,Nz] = surfnorm(...)
```

Description

The `surfnorm` function computes surface normals for the surface defined by `X`, `Y`, and `Z`. The surface normals are unnormalized and valid at each vertex. Normals are not shown for surface elements that face away from the viewer.

`surfnorm(Z)` and `surfnorm(X,Y,Z)` plot a surface and its surface normals. `Z` is a matrix that defines the z component of the surface. `X` and `Y` are vectors or matrices that define the *x* and *y* components of the surface.

`[Nx,Ny,Nz] = surfnorm(...)` returns the components of the three-dimensional surface normals for the surface.

Remarks

The direction of the normals is reversed by calling `surfnorm` with transposed arguments:

```
surfnorm(X',Y',Z')
```

`surfl` uses `surfnorm` to compute surface normals when calculating the reflectance of a surface.

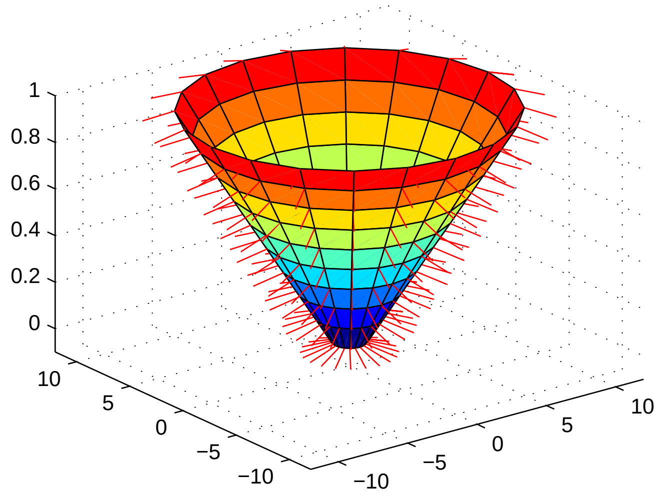
Algorithm

The surface normals are based on a bicubic fit of the data in `X`, `Y`, and `Z`. For each vertex, diagonal vectors are computed and crossed to form the normal.

Examples

Plot the normal vectors for a truncated cone.

```
[x,y,z] = cylinder(1:10);
surfnorm(x,y,z)
axis([–12 12 –12 12 –0.1 1])
```



See Also `surf`, `quiver3`

Purpose

Set graphics terminal type

Syntax

```
terminal
terminal('type')
```

Description

To add terminal-specific settings (e.g., escape characters, line length), edit the file `terminal.m`.

`terminal` displays a menu of graphics terminal types, prompts for a choice, then configures MATLAB to run on the specified terminal.

`terminal('`*type*`')` accepts a terminal type string. Valid `'`*type*`'` strings are shown in the table.

Type	Description
`tek401x`	Tektronix 4010/4014
`tek4100`	Tektronix 4100
`tek4105`	Tektronix 4105
`retro`	Retrographics card
`sg100`	Selanar Graphics 100
`sg200`	Selanar Graphics 200
`vt240tek`	VT240 & VT340 Tektronix mode
`ergo`	Ergo terminal
`graphon`	Graphon terminal
`citoh`	C.Itoh terminal
`xtermtek`	xterm, Tektronix graphics
`wyse`	Wyse WY-99GT
`kermit`	MS-DOS Kermit 2.23
`hp2647`	Hewlett-Packard 2647

Type	Description (Continued)
hds	Human Designed Systems

Purpose Produce TeX format from character string

Syntax

```
texlabel(f)
texlabel(f,'literal')
```

Description `texlabel(f)` converts the MATLAB expression `f` into the TeX equivalent for use in text strings. It processes Greek variable names (e.g., lambda, delta, etc.) into a string that displays as actual Greek letters.

`texlabel(f,'literal')` prints Greek variable names as literals.

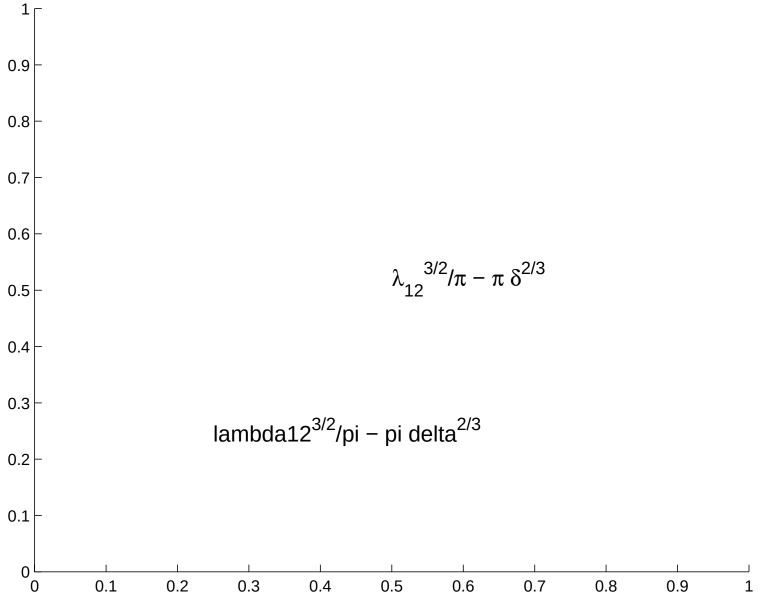
If the string is too long to fit into a figure window, then the center of the expression is replaced with a tilde ellipsis (~~~).

Examples You can use `texlabel` as an argument to the `title`, `xlabel`, `ylabel`, `zlabel`, and `text` commands. For example,

```
title(texlabel('sin(sqrt(x^2 + y^2))/sqrt(x^2 + y^2)'))
```

By default, `texlabel` translates Greek variable names to the equivalent Greek letter. You can select literal interpretation by including the `literal` argument. For example, compare these two commands.

```
text(.5,.5,...
     texlabel('lambda12^(3/2)/pi - pi*delta^(2/3)'))
text(.25,.25,...
     texlabel('lambda12^(3/2)/pi - pi*delta^(2/3)','literal'))
```



See Also `text`, `title`, `xlabel`, `ylabel`, `zlabel`, the text `String` property

Purpose Create text object in current axes

Syntax

```
text(x,y,'string')
text(x,y,z,'string')
text(...'PropertyName',PropertyValue...)
h = text(...)
```

Description `text` is the low-level function for creating text graphics objects. Use `text` to place character strings at specified locations.

`text(x,y,'string')` adds the string in quotes to the location specified by the point `(x,y)`.

`text(x,y,z,'string')` adds the string in 3-D coordinates.

`text(x,y,z,'string','`*PropertyName*`',PropertyValue....)` adds the string in quotes to location defined by the coordinates and uses the values for the specified text properties. See the text property list section at the end of this page for a list of text properties.

`text('`*PropertyName*`',PropertyValue....)` omits the coordinates entirely and specifies all properties using property name/property value pairs.

`h = text(..)` returns a column vector of handles to text objects, one handle per object. All forms of the `text` function optionally return this output argument.

See the `String` property for a list of symbols, including Greek letters.

Remarks Specify the text location coordinates (the `x`, `y`, and `z` arguments) in the data units of the current axes (see "Examples"). The `Extent`, `VerticalAlignment`, and `HorizontalAlignment` properties control the positioning of the character string with regard to the text location point.

If the coordinates are vectors, `text` writes the string at all locations defined by the list of points. If the character string is an array the same length as `x`, `y`, and `z`, `text` writes the corresponding row of the string array at each point specified.

When specifying strings for multiple text objects, the string can be

- a cell array of strings
- a padded string matrix
- a string vector using vertical slash characters ('|') as separators.

Each element of the specified string array creates a different text object.

When specifying the string for a single text object, cell arrays of strings and padded string matrices result in a text object with a multiline string, while vertical slash characters are not interpreted as separators and result in a single line string containing vertical slashes.

`text` is a low-level function that accepts property name/property value pairs as input arguments, however; the convenience form,

```
text(x,y,z,'string')
```

is equivalent to:

```
text('XData',x,'YData',y,'ZData',z,'String','string')
```

You can specify other properties only as property name/property value pairs. See the text property list at the end of this page for a description of each property. You can specify properties as property name/property value pairs, structure arrays, and cell arrays (see the `set` and `get` reference pages for examples of how to specify these data types).

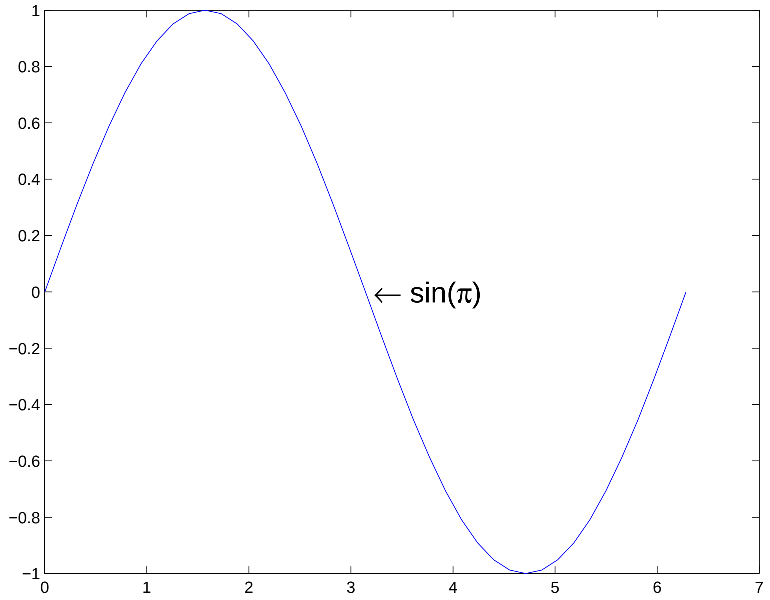
`text` does not respect the setting of the figure or axes `NextPlot` property. This allows you to add text objects to an existing axes without setting `hold` to `on`.

Examples

The statements,

```
plot(0:pi/20:2*pi,sin(0:pi/20:2*pi))
text(pi,0,' \leftarrow sin(\pi)','FontSize',18)
```

annotate the point at `(pi,0)` with the string `sin(π)`.



The statement,

```
text(x,y,'\ite^{i\omega\tau} = cos(\omega\tau) + i sin(\omega\tau)')
```

uses embedded TeX sequences to produce:

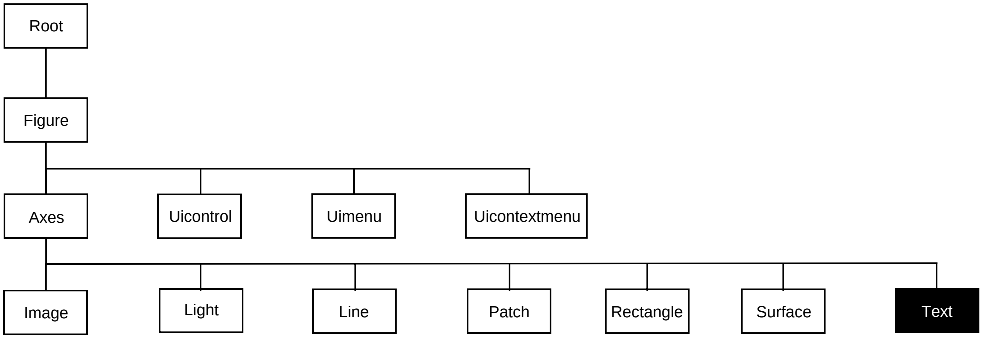
$$e^{i\omega\tau} = cos(\omega\tau) + i\ sin(\omega\tau)$$

See Also

`gtext`, `int2str`, `num2str`, `title`, `xlabel`, `ylabel`, `zlabel`

The “Labeling Graphs” topic in the online *Using MATLAB Graphics* manual discusses positioning text.

Object Hierarchy



Setting Default Properties

You can set default text properties on the axes, figure, and root levels.

```
set(0,'DefaulttextProperty',PropertyValue...)
set(gcf,'DefaulttextProperty',PropertyValue...)
set(gca,'DefaulttextProperty',PropertyValue...)
```

Where *Property* is the name of the text property and `PropertyValue` is the value you are specifying. Use `set` and `get` to access text properties.

Property List

The following table lists all text properties and provides a brief description of each. The property name links take you to an expanded description of the properties.

Property Name	Property Description	Property Value
Defining the character string		
`Editing`	Enable or disable editing mode.	Values: `on`, `off` Default: `off`
`Interpreter`	Enable or disable TeX interpretation	Values: `tex`, `none` Default: `tex`
`String`	The character string (including list of TeX character sequences)	Value: character string
Positioning the character string		

Property Name	Property Description	Property Value
Extent	Position and size of text object	Values: [left, bottom, width, height]
HorizontalAlignment	Horizontal alignment of text string	Values: left, center, right Default: left
Position	Position of text Extent rectangle	Values: [x, y, z] coordinates Default: [] empty matrix
Rotation	Orientation of text object	Values: scalar (degrees) Default: 0
Units	Units for Extent and Position properties	Values: pixels, normalized, inches, centimeters, points, data Default: data
VerticalAlignment	Vertical alignment of text string	Values: top, cap, middle, baseline, bottom Default: middle
Specifying the Font		
FontAngle	Select italic-style font	Values: normal, italic, oblique Default: normal
FontName	Select font family	Values: a font supported by your system or the string FixedWidth Default: Helvetica
FontSize	Size of font	Values: size in FontUnits Default: 10 points
FontUnits	Units for FontSize property	Values: points, normalized, inches, centimeters, pixels Default: points

Property Name	Property Description	Property Value
FontWeight	Weight of text characters	Values: light, normal, demi, bold Default: normal
Controlling the Appearance		
Clipping	Clipping to axes rectangle	Values: on, off Default: on
EraseMode	Method of drawing and erasing the text (useful for animation)	Values: normal, none, xor, background Default: normal
SelectionHighlight	Highlight text when selected (Selected property set to on)	Values: on, off Default: on
Visible	Make the text visible or invisible	Values: on, off Default: on
Color	Color of the text	ColorSpec
Controlling Access to Text Objects		
HandleVisibility	Determines if and when the the text's handle is visible to other functions	Values: on, callback, off Default: on
HitTest	Determines if the text can become the current object (see the figure CurrentObject property)	Values: on, off Default: on
General Information About Text Objects		
Children	Text objects have no children	Values: [] (empty matrix)
Parent	The parent of a text object is always an axes object	Value: axes handle
Selected	Indicate whether the text is in a "selected" state.	Values: on, off Default: off

Property Name	Property Description	Property Value
Tag	User-specified label	Value: any string Default: '' (empty string)
Type	The type of graphics object (read only)	Value: the string 'text'
UserData	User-specified data	Values: any matrix Default: [] (empty matrix)
Controlling Callback Routine Execution		
BusyAction	Specifies how to handle callback routine interruption	Values: cancel, queue Default: queue
ButtonDownFcn	Defines a callback routine that executes when a mouse button is pressed on over the text	Values: string Default: '' (empty string)
CreateFcn	Defines a callback routine that executes when an text is created	Values: string Default: '' (empty string)
DeleteFcn	Defines a callback routine that executes when the text is deleted (via close or delete)	Values: string Default: '' (empty string)
Interruptible	Determines if callback routine can be interrupted	Values: on, off Default: on (can be interrupted)
UIContextMenu	Associates a context menu with the text	Values: handle of a uicontextmenu

Text Properties

This section lists property names along with the types of values each accepts. Curly braces { } enclose default values.

BusyAction `cancel | {queue}`

Callback routine interruption. The `BusyAction` property enables you to control how MATLAB handles events that potentially interrupt executing callback routines. If there is a callback routine executing, subsequently invoked callback routines always attempt to interrupt it. If the `Interruptible` property of the object whose callback is executing is set to `on` (the default), then interruption occurs at the next point where the event queue is processed. If the `Interruptible` property is `off`, the `BusyAction` property (of the object owning the executing callback) determines how MATLAB handles the event. The choices are:

- `cancel` – discard the event that attempted to execute a second callback routine.
- `queue` – queue the event that attempted to execute a second callback routine until the current callback finishes.

ButtonDownFcn string

Button press callback routine. A callback routine that executes whenever you press a mouse button while the pointer is over the text object. Define this routine as a string that is a valid MATLAB expression or the name of an M-file. The expression executes in the MATLAB workspace.

Children matrix (read only)

The empty matrix; text objects have no children.

Clipping `on | {off}`

Clipping mode. When `Clipping` is `on`, MATLAB does not display any portion of the text that is outside the axes.

Color `ColorSpec`

Text color. A three-element RGB vector or one of MATLAB's predefined names, specifying the text color. The default value for `Color` is white. See `ColorSpec` for more information on specifying color.

CreateFcn string

Callback routine executed during object creation. This property defines a callback routine that executes when MATLAB creates a text object. You must define this property as a default value for text. For example, the statement,

```
set(0,'DefaultTextCreateFcn',...
    'set(gcf,''Pointer'',’'crosshair'')')
```

defines a default value on the root level that sets the figure `Pointer` property to a crosshair whenever you create a text object. MATLAB executes this routine after setting all text properties. Setting this property on an existing text object has no effect.

The handle of the object whose `CreateFcn` is being executed is accessible only through the root `CallbackObject` property, which you can query using `gcbo`.

DeleteFcn string

Delete text callback routine. A callback routine that executes when you delete the text object (e.g., when you issue a `delete` command or clear the axes or figure). MATLAB executes the routine before destroying the object’s properties so these values are available to the callback routine.

The handle of the object whose `DeleteFcn` is being executed is accessible only through the root `CallbackObject` property, which you can query using `gcbo`.

Editing on | {off}

Enable or disable editing mode. When this property is set to the default `off`, you cannot edit the text string interactively (i.e., you must change the `String` property to change the text). When this property is set to on, MATLAB places an insert cursor at the beginning of the text string and enables editing. To apply the new text string:

- Press the **ESC** key
- Clicking in any figure window (including the current figure)
- Reset the `Editing` property to `off`

MATLAB then updates the `String` property to contain the new text and resets the `Editing` property to `off`. You must reset the `Editing` property to on to again resume editing.

EraseMode `{normal} | none | xor | background`

Erase mode. This property controls the technique MATLAB uses to draw and erase text objects. Alternative erase modes are useful for creating animated sequences, where controlling the way individual object redraw is necessary to improve performance and obtain the desired effect.

- `normal` — Redraw the affected region of the display, performing the three-dimensional analysis necessary to ensure that all objects are rendered correctly. This mode produces the most accurate picture, but is the slowest. The other modes are faster, but do not perform a complete redraw and are therefore less accurate.
- `none` — Do not erase the text when it is moved or destroyed. While the object is still visible on the screen after erasing with `EraseMode` `none`, you cannot print it because MATLAB stores no information about its former location.
- `xor` — Draw and erase the text by performing an exclusive OR (XOR) with each pixel index of the screen beneath it. When the text is erased, it does not damage the objects beneath it. However, when text is drawn in `xor` mode, its color depends on the color of the screen beneath it and is correctly colored only when over axes background `Color`, or the figure background `Color` if the axes `Color` is set to `none`.
- `background` — Erase the text by drawing it in the background `Color`, or the figure background `Color` if the axes `Color` is set to `none`. This damages objects that are behind the erased text, but text is always properly colored.

Printing with Non-normal Erase Modes. MATLAB always prints figures as if the `EraseMode` of all objects is `normal`. This means graphics objects created with `EraseMode` set to `none`, `xor`, or `background` can look different on screen than on paper. On screen, MATLAB may mathematically combine layers of colors (e.g., XORing a pixel color with that of the pixel behind it) and ignore three-dimensional sorting to obtain greater rendering speed. However, these techniques are not applied to the printed output.

You can use the MATLAB `getframe` command or other screen capture application to create an image of a figure containing non-normal mode objects.

`Extent` position rectangle (read only)

Position and size of text. A four-element read-only vector that defines the size and position of the text string.

```
[left,bottom,width,height]
```

If the `Units` property is set to `data` (the default), `left` and `bottom` are the *x* and *y* coordinates of the lower-left corner of the text `Extent` rectangle.

For all other values of `Units`, `left` and `bottom` are the distance from the lower-left corner of the axes position rectangle to the lower-left corner of the text `Extent` rectangle. `width` and `height` are the dimensions of the `Extent` rectangle. All measurements are in units specified by the `Units` property.

`FontAngle` `{normal} | italic | oblique`

Character slant. MATLAB uses this property to select a font from those available on your particular system. Generally, setting this property to `italic` or `oblique` selects a slanted font.

`FontName` A name such as `Courier` or the string `FixedWidth`

Font family. A string specifying the name of the font to use for the text object. To display and print properly, this must be a font that your system supports. The default font is Helvetica.

Specifying a Fixed-Width Font

If you want text to use a fixed-width font that looks good in any locale, you should set `FontName` to the string `FixedWidth`:

```
set(text_handle,'FontName','FixedWidth')
```

This eliminates the need to hardcode the name of a fixed-width font, which may not display text properly on systems that do not use ASCII character encoding (such as in Japan where multibyte character sets are used). A properly written MATLAB application that needs to use a fixed-width font should set `FontName` to `FixedWidth` (note that this string is case sensitive) and rely on `FixedWidthFontName` to be set correctly in the end-user's environment.

End users can adapt a MATLAB application to different locales or personal environments by setting the root `FixedWidthFontName` property to the appropriate value for that locale from `startup.m`.

Note that setting the root `FixedWidthFontName` property causes an immediate update of the display to use the new font.

FontSize size in `FontUnits`

Font size. An integer specifying the font size to use for text, in units determined by the `FontUnits` property. The default point size is 10 (1 point = 1/72 inch).

FontWeight `light | {normal} | demi | bold`

Weight of text characters. MATLAB uses this property to select a font from those available on your particular system. Generally, setting this property to `bold` or `demi` causes MATLAB to use a bold font.

FontUnits `{points} | normalized | inches | centimeters | pixels`

Font size units. MATLAB uses this property to determine the units used by the `FontSize` property. `Normalized` units interpret `FontSize` as a fraction of the height of the parent axes. When you resize the axes, MATLAB modifies the screen `FontSize` accordingly. `pixels`, `inches`, `centimeters`, and `points` are absolute units (1 point = 1/72 inch).

HandleVisibility `{on} | callback | off`

Control access to object's handle by command-line users and GUIs. This property determines when an object's handle is visible in its parent's list of children. `HandleVisibility` is useful for preventing command-line users from accidentally drawing into or deleting a figure that contains only user interface devices (such as a dialog box).

Handles are always visible when `HandleVisibility` is `on`.

Setting `HandleVisibility` to `callback` causes handles to be visible from within callback routines or functions invoked by callback routines, but not from within functions invoked from the command line. This provides a means to protect GUIs from command-line users, while allowing callback routines to have complete access to object handles.

Setting `HandleVisibility` to `off` makes handles invisible at all times. This may be necessary when a callback routine invokes a function that might potentially damage the GUI (such as evaluating a user-typed string), and so temporarily hides its own handles during the execution of that function.

When a handle is not visible in its parent's list of children, it cannot be returned by functions that obtain handles by searching the object hierarchy or querying handle properties. This includes `get`, `findobj`, `gca`, `gcf`, `gco`, `newplot`, `cla`, `clf`, and `close`.

When a handle's visibility is restricted using `callback` or `off`, the object's handle does not appear in its parent's `Children` property, figures do not appear in the root's `CurrentFigure` property, objects do not appear in the root's `CallbackObject` property or in the figure's `CurrentObject` property, and axes do not appear in their parent's `CurrentAxes` property.

You can set the root `ShowHiddenHandles` property to on to make all handles visible, regardless of their `HandleVisibility` settings (this does not affect the values of the `HandleVisibility` properties).

Handles that are hidden are still valid. If you know an object's handle, you can `set` and `get` its properties, and pass it to any function that operates on handles.

HitTest `{on} | off`

Selectable by mouse click. `HitTest` determines if the text can become the current object (as returned by the `gco` command and the figure `CurrentObject` property) as a result of a mouse click on the text. If `HiTest` is `off`, clicking on the text selects the object below it (which is usually the axes containing it).

For example, suppose you define the button down function of an image (see the `ButtonDownFcn` property) to display text at the location you click on with the mouse.

First define the callback routine.

```
function bd_function
pt = get(gca,'CurrentPoint');
text(pt(1,1),pt(1,2),pt(1,3),...
   '{\fontsize{20}\oplus} The spot to label',...
   'HitTest','off')
```

Now display an image, setting its `ButtonDownFcn` property to the callback routine.

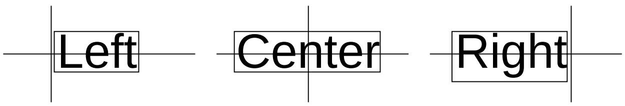
```
load earth
image(X,'ButtonDownFcn','bd_function'); colormap(map)
```

When you click on the image, MATLAB displays the text string at that location. With `HitTest` set to `off`, existing text cannot intercept any subsequent button down events that occur over the text. This enables the image's button down function to execute.

`HorizontalAlignment` `{left} | center | right`

Horizontal alignment of text. This property specifies the horizontal justification of the text string. It determines where MATLAB places the string with regard to the point specified by the `Position` property. The following picture illustrates the alignment options.

Text `HorizontalAlignment` property viewed with the `VerticalAlignment` property set to `middle` (the default).



See the `Extent` property for related information.

`Interpreter` `{tex} | none`

Interpret Tex instructions. This property controls whether MATLAB interprets certain characters in the `String` property as Tex instructions (default) or displays all characters literally. See the `String` property for a list of support Tex instructions.

`Interruptible` `{on} | off`

Callback routine interruption mode. The `Interruptible` property controls whether a text callback routine can be interrupted by subsequently invoked callback routines. text objects have four properties that define callback routines: `ButtonDownFcn`, `CreateFcn`, and `DeleteFcn`. See the `BusyAction` property for information on how MATLAB executes callback routines.

`Parent` handle

Text object's parent. The handle of the text object's parent object. The parent of a text object is the axes in which it is displayed. You can move a text object to another axes by setting this property to the handle of the new parent.

Position [x,y,[z]]

Location of text. A two- or three-element vector, [x y [z]], that specifies the location of the text in three dimensions. If you omit the z value, it defaults to 0. All measurements are in units specified by the Units property. Initial value is [0 0 0].

Rotation scalar (default = 0)

Text orientation. This property determines the orientation of the text string. Specify values of rotation in degrees (positive angles cause counterclockwise rotation).

Selected on | {off}

Is object selected? When this property is on, MATLAB displays selection handles if the SelectionHighlight property is also on. You can, for example, define the ButtonDownFcn to set this property, allowing users to select the object with the mouse.

SelectionHighlight {on} | off

Objects highlight when selected. When the Selected property is on, MATLAB indicates the selected state by drawing four edge handles and four corner handles. When SelectionHighlight is off, MATLAB does not draw the handles.

String string

The text string. Specify this property as a quoted string for single-line strings, or as a cell array of strings or a padded string matrix for multiline strings. MATLAB displays this string at the specified location. Vertical slash characters are not interpreted as linebreaks in text strings, and are drawn as part of the text string. See the "Remarks" section for more information.

When the text Interpreter property is Tex (the default), you can use a subset of TeX commands embedded in the string to produce special characters such as Greek letters and mathematical symbols. The following table lists these characters and the character sequence used to define them.

Character Sequence	Symbol	Character Sequence	Symbol	Character Sequence	Symbol
\alpha	α	\upsilon	υ	\sim	~
\beta	β	\phi	ϕ	\leq	≤
\gamma	γ	\chi	χ	\infty	∞
\delta	δ	\psi	ψ	\clubsuit	♣
\epsilon	ε	\omega	ω	\diamondsuit	♦
\zeta	ζ	\Gamma	Γ	\heartsuit	♥
\eta	η	\Delta	Δ	\spadesuit	♠
\theta	θ	\Theta	Θ	\leftrightarrow	↔
\vartheta	ϑ	\Lambda	Λ	\leftarrow	←
\iota	ι	\Xi	Ξ	\uparrow	↑
\kappa	κ	\Pi	Π	\rightarrow	→
\lambda	λ	\Sigma	Σ	\downarrow	↓
\mu	µ	\Upsilon	ϒ	\circ	º
\nu	ν	\Phi	Φ	\pm	±
\xi	ξ	\Psi	Ψ	\geq	≥
\pi	π	\Omega	Ω	\propto	∝
\rho	ρ	\forall	∀	\partial	∂
\sigma	σ	\exists	∃	\bullet	•
\varsigma	ς	\ni	∍	\div	÷
\tau	τ	\cong	≅	\neq	≠
\equiv	≡	\approx	≈	\aleph	ℵ
\Im	ℑ	\Re	ℜ	\wp	℘

Character Sequence	Symbol	Character Sequence	Symbol	Character Sequence	Symbol
\otimes	⊗	\oplus	⊕	\oslash	∅
\cap	∩	\cup	∪	\supseteq	⊇
\supset	⊃	\subseteq	⊆	\subset	⊂
\int	∫	\in	∈	\o	ο
\rfloor	⌋	\lceil	⌈	\nabla	∇
\lfloor	⌊	\cdot	·	\ldots	...
\perp	⊥	\neg	¬	\prime	′
\wedge	∧	\times	×	\0	∅
\rceil	⌉	\surd	√	\mid	|
\vee	∨	\varpi	ϖ	\copyright	©
\langle	〈	\rangle	〉		

You can also specify stream modifiers that control the font used. The first four modifiers are mutually exclusive. However, you can use `\fontname` in combination with one of the other modifiers:

- `\bf` – bold font
- `\it` – italics font
- `\sl` – oblique font (rarely available)
- `\rm` – normal font
- `\fontname{`*fontname*`}` – specify the name of the font family to use.
- `\fontsize{`*fontsize*`}` – specify the font size in `FontUnits`.

Stream modifiers remain in effect until the end of the string or only within the context defined by braces { }.

Specifying Subscript and Superscript Characters

The subscript character “_” and the superscript character “^” modify the character or substring defined in braces immediately following.

To print the special characters used to define the Tex strings when Interpreter is Tex, prefix them with the backslash “\” character: \\, \{, \} _, \^.

See the example for more information.

When Interpreter is none, no characters in the String are interpreted, and all are displayed when the text is drawn.

Tag string

User-specified object label. The Tag property provides a means to identify graphics objects with a user-specified label. This is particularly useful when constructing interactive graphics programs that would otherwise need to define object handles as global variables or pass them as arguments between callback routines. You can define Tag as any string.

Type string (read only)

Class of graphics object. For text objects, Type is always the string 'text'.

Units pixels | normalized | inches | centimeters | points | {data}

Units of measurement. This property specifies the units MATLAB uses to interpret the Extent and Position properties. All units are measured from the lower-left corner of the axes plotbox. Normalized units map the lower-left corner of the rectangle defined by the axes to (0,0) and the upper-right corner to (1.0,1.0). pixels, inches, centimeters, and points are absolute units (1 point = $^1/_{72}$ inch). data refers to the data units of the parent axes.

If you change the value of Units, it is good practice to return it to its default value after completing your computation so as not to affect other functions that assume Units is set to the default value.

UserData matrix

User-specified data. Any data you want to associate with the text object. MATLAB does not use this data, but you can access it using set and get.

UIContextMenu handle of a uicontextmenu object

Associate a context menu with the text. Assign this property the handle of a uicontextmenu object created in the same figure as the text. Use the uicontextmenu function to create the context menu. MATLAB displays the context menu whenever you right-click over the text.

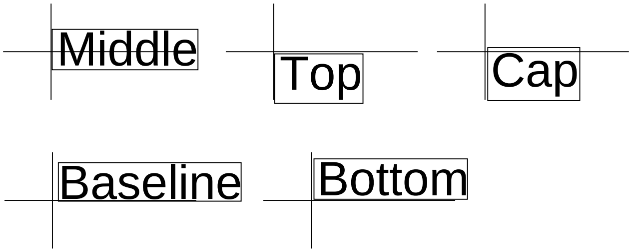
VerticalAlignment `top | cap | {middle} | baseline | bottom`

Vertical alignment of text. This property specifies the vertical justification of the text string. It determines where MATLAB places the string with regard to the value of the `Position` property. The possible values mean:

- `top` – Place the top of the string's `Extent` rectangle at the specified *y*-position.
- `cap` – Place the string so that the top of a capital letter is at the specified *y*-position.
- `middle` – Place the middle of the string at specified *y*-position.
- `baseline` – Place font baseline at the specified *y*-position.
- `bottom` – Place the bottom of the string's `Extent` rectangle at the specified *y*-position.

The following picture illustrates the alignment options.

Text `VerticalAlignment` property viewed with the `HorizontalAlignment` property set to `left` (the default).



Visible `{on} | off`

Text visibility. By default, all text is visible. When set to `off`, the text is not visible, but still exists and you can query and set its properties.

textwrap

Purpose Return wrapped string matrix for given uicontrol

Syntax

```
outstring = textwrap(h,instring)
[outstring,position] = textwrap(h,instring)
```

Description `outstring = textwrap(h,instring)` returns a wrapped string cell array, `outstring`, that fits inside the uicontrol with handle `h`. `instring` is a cell array, with each cell containing a single line of text. `outstring` is the wrapped string matrix in cell array format. Each cell of the input string is considered a paragraph.

`[outstring,position]=textwrap(h,instring)` returns the recommended position of the uicontrol in the units of the uicontrol. `position` considers the extent of the multiline text in the *x* and *y* directions.

Example Place a textwrapped string in a uicontrol:

```
pos = [10 10 100 10];
h = uicontrol('Style','Text','Position',pos);
string = {'This is a string for the uicontrol.',
          'It should be correctly wrapped inside.'};
[outstring,newpos] = textwrap(h,string);
pos(4) = newpos(4);
set(h,'String',outstring,'Position',[pos(1),pos(2),pos(3)+10,pos(4)])
```

See Also `uicontrol`

Purpose Add title to current axes

Syntax

```
title('string')
title(fname)
title(...,'PropertyName',PropertyValue,...)
h = title(...)
```

Description Each axes graphics object can have one title. The title is located at the top and in the center of the axes.

`title('string')` outputs the string at the top and in the center of the current axes.

`title(fname)` evaluates the function that returns a string and displays the string at the top and in the center of the current axes.

`title(...,'PropertyName',PropertyValue,...)` specifies property name and property value pairs for the text graphics object that `title` creates.

`h = title(...)` returns the handle to the text object used as the title.

Examples Display today's date in the current axes:

```
title(date)
```

Include a variable's value in a title:

```
f = 70;
c = (f–32)/1.8;
title(['Temperature is ',num2str(c),'C'])
```

Include a variable's value in a title and set the color of the title to yellow:

```
n = 3;
title(['Case number #',int2str(n)],'Color','y')
```

Include Greek symbols in a title:

```
title('\ite^{\omega\tau} = cos(\omega\tau) + isin(\omega\tau)’)
```

Include a superscript character in a title:

```
title('\alpha^2’)
```

Include a subscript character in a title:

```
title('X_1')
```

The text object `String` property lists the available symbols.

Remarks

`title` sets the `Title` property of the current axes graphics object to a new text graphics object. See the text `String` property for more information.

See Also

`gtext`, `int2str`, `num2str`, `plot`, `text`, `xlabel`, `ylabel`, `zlabel`

Purpose Triangular mesh plot

Syntax

```
trimesh(Tri,X,Y,Z)
trimesh(Tri,X,Y,Z,C)
trimesh(...'PropertyName',PropertyValue...)
h = trimesh(...)
```

Description `trimesh(Tri,X,Y,Z)` displays triangles defined in the *m*-by-3 face matrix `Tri` as a mesh. Each row of `Tri` defines a single triangular face by indexing into the vectors or matrices that contain the `X`, `Y`, and `Z` vertices.

`trimesh(Tri,X,Y,Z,C)` specifies color defined by `C` in the same manner as the `surf` function. MATLAB performs a linear transformation on this data to obtain colors from the current colormap.

`trimesh(...'`*`PropertyName`*`',PropertyValue...)` specifies additional patch property names and values for the patch graphics object created by the function.

`h = trimesh(...)` returns a handle to a patch graphics object.

Example Create vertex vectors and a face matrix, then create a triangular mesh plot.

```
x = rand(1,50);
y = rand(1,50);
z = peaks(6*x–3,6*x–3);
tri = delaunay(x,y);
trimesh(tri,x,y,z)
```

See Also `patch`, `trisurf`, `delaunay`

trisurf

Purpose Triangular surface plot

Syntax

```
trisurf(Tri,X,Y,Z)
trisurf(Tri,X,Y,Z,C)
trisurf(...'PropertyName',PropertyValue...)
h = trisurf(...)
```

Description

`trisurf(Tri,X,Y,Z)` displays triangles defined in the *m*-by-3 face matrix `Tri` as a surface. Each row of `Tri` defines a single triangular face by indexing into the vectors or matrices that contain the `X`, `Y`, and `Z` vertices.

`trisurf(Tri,X,Y,Z,C)` specifies color defined by `C` in the same manner as the `surf` function. MATLAB performs a linear transformation on this data to obtain colors from the current colormap.

`trisurf(...'`*`PropertyName`*`',PropertyValue...)` specifies additional patch property names and values for the patch graphics object created by the function.

`h = trisurf(...)` returns a patch handle.

Example Create vertex vectors and a face matrix, then create a triangular surface plot.

```
x = rand(1,50);
y = rand(1,50);
z = peaks(6*x–3,6*x–3);
tri = delaunay(x,y);
trisurf(tri,x,y,z)
```

See Also `patch`, `surf`, `trimesh`, `delaunay`

Purpose Create a context menu

Syntax

```
handle = uicontextmenu('PropertyName',PropertyValue,...);
```

Description

uicontextmenu creates a context menu, which is a menu that appears when the user right-clicks on a graphics object.

You create context menu items using the uimenu function. Menu items appear in the order the uimenu statements appear. You associate a context menu with an object using the UIContextMenu property for the object and specifying the context menu's handle as the property value.

More information about context menus.

Properties

This table lists the properties that are useful to uicontextmenu objects, grouping them by function. Each property name acts as a link to a description of the property.

Property Name	Property Description	Property Value
Controlling Style and Appearance		
Visible	Uicontextmenu visibility	Value: on, off Default: off
Position	Location of uicontextmenu when Visible is set to on	Value: two-element vector Default: [0 0]
General Information About the Object		
Children	The uimenus defined for the uicontextmenu	Value: matrix
Parent	Uicontextmenu object's parent	Value: scalar figure handle
Tag	User-specified object identifier	Value: string
Type	Class of graphics object	Value: string (read-only) Default: uicontrol
UserData	User-specified data	Value: matrix

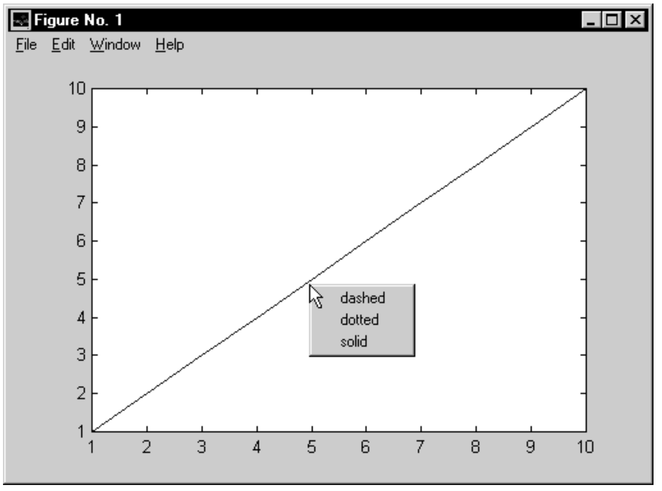
Property Name	Property Description	Property Value
Controlling Callback Routine Execution		
BusyAction	Callback routine interruption	Value: cancel, queue Default: queue
Callback	Control action	Value: string
CreateFcn	Callback routine executed during object creation	Value: string
DeleteFcn	Callback routine executed during object deletion	Value: string
Interruptible	Callback routine interruption mode	Value: on, off Default: on
Controlling Access to Objects		
HandleVisibility	Whether handle is accessible from command line and GUIs	Value: on, callback, off Default: on

Example

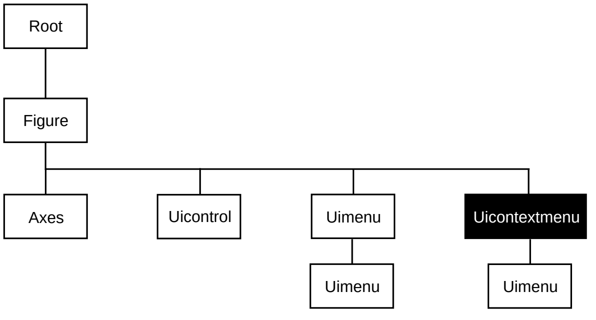
These statements define a context menu associated with a line. When the user extend-clicks anywhere on the line, the menu appears. Menu items enable the user to change the line style.

```
% Define the context menu
cmenu = uicontextmenu;
% Define the line and associate it with the context menu
hline = plot(1:10, 'UIContextMenu', cmenu);
% Define callbacks for context menu items
cb1 = ['set(hline, ''LineStyle'', ''--'')'];
cb2 = ['set(hline, ''LineStyle'', '':'')'];
cb3 = ['set(hline, ''LineStyle'', ''-'')'];
% Define the context menu items
item1 = uimenu(cmenu, 'Label', 'dashed', 'Callback', cb1);
item2 = uimenu(cmenu, 'Label', 'dotted', 'Callback', cb2);
item3 = uimenu(cmenu, 'Label', 'solid', 'Callback', cb3);
```

When the user extend-clicks on the line, the context menu appears, as shown in this figure:



Object Hierarchy



See Also

`uicontrol`, `uimenu`

Uicontextmenu Properties

BusyAction `cancel | {queue}`

Callback routine interruption. The `BusyAction` property enables you to control how MATLAB handles events that potentially interrupt executing callback routines. If a callback routine is executing, subsequently invoked callback routines always attempt to interrupt it. If the `Interruptible` property of the object whose callback is executing is set to `on` (the default), then interruption occurs at the next point where the event queue is processed. If the `Interruptible` property is `off`, the `BusyAction` property of the object whose callback is executing determines how MATLAB handles the event. The choices are:

- `cancel` – discard the event that attempted to execute a second callback routine.
- `queue` – queue the event that attempted to execute a second callback routine until the current callback finishes.

ButtonDownFcn string

This property has no effect on uicontextmenu objects.

Callback string

Control action. A routine that executes whenever you right-click on an object for which a context menu is defined. The routine executes immediately before the context menu is posted. Define this routine as a string that is a valid MATLAB expression or the name of an M-file. The expression executes in the MATLAB workspace.

Children matrix

The uimenus defined for the uicontextmenu.

Clipping `{on} | off`

This property has no effect on uicontextmenu objects.

CreateFcn string

Callback routine executed during object creation. This property defines a callback routine that executes when MATLAB creates a uicontextmenu object. You must define this property as a default value for uicontextmenus. For example, this statement:

```
set(0,'DefaultUicontextmenuCreateFcn',...
      'set(gcf,''IntegerHandle'',''off'')')
```

defines a default value on the root level that sets the figure `IntegerHandle` property to `off` whenever you create a uicontextmenu object. MATLAB executes this routine after setting all property values for the uicontextmenu. Setting this property on an existing uicontextmenu object has no effect.

The handle of the object whose `CreateFcn` is being executed is accessible only through the root `CallbackObject` property, which can be queried using `gcbo`.

`DeleteFcn` string

Delete uicontextmenu callback routine. A callback routine that executes when you delete the uicontextmenu object (e.g., when you issue a `delete` command or clear the figure containing the uicontextmenu). MATLAB executes the routine before destroying the object's properties so these values are available to the callback routine.

The handle of the object whose `DeleteFcn` is being executed is accessible only through the root `CallbackObject` property, which you can query using `gcbo`.

`HandleVisibility` `{on} | callback | off`

Control access to object's handle by command-line users and GUIs. This property determines when an object's handle is visible in its parent's list of children. `HandleVisibility` is useful for preventing command-line users from accidentally drawing into or deleting a figure that contains only user interface devices (such as a dialog box).

Handles are always visible when `HandleVisibility` is `on`.

Setting `HandleVisibility` to `callback` causes handles to be visible from within callback routines or functions invoked by callback routines, but not from within functions invoked from the command line. This provides a means to protect GUIs from command-line users, while allowing callback routines to have complete access to object handles.

Setting `HandleVisibility` to `off` makes handles invisible at all times. This may be necessary when a callback routine invokes a function that might potentially damage the GUI (such as evaluating a user-typed string), and so temporarily hides its own handles during the execution of that function.

When a handle is not visible in its parent's list of children, it cannot be returned by functions that obtain handles by searching the object hierarchy or querying handle properties. This includes `get`, `findobj`, `gca`, `gcf`, `gco`, `newplot`, `cla`, `clf`, and `close`.

When a handle's visibility is restricted using `callback` or `off`, the object's handle does not appear in its parent's `Children` property, figures do not appear in the root's `CurrentFigure` property, objects do not appear in the root's `CallbackObject` property or in the figure's `CurrentObject` property, and axes do not appear in their parent's `CurrentAxes` property.

You can set the root `ShowHiddenHandles` property to on to make all handles visible, regardless of their `HandleVisibility` settings (this does not affect the values of the `HandleVisibility` properties).

Handles that are hidden are still valid. If you know an object's handle, you can `set` and `get` its properties, and pass it to any function that operates on handles.

HitTest `{on} | off`

This property has no effect on uicontextmenu objects.

Interruptible `{on} | off`

Callback routine interruption mode. The `Interruptible` property controls whether a uicontextmenu callback routine can be interrupted by subsequently invoked callback routines. By default (on), execution of a callback routine can be interrupted.

Only callback routines defined for the `ButtonDownFcn` and `Callback` properties are affected by the `Interruptible` property. MATLAB checks for events that can interrupt a callback routine only when it encounters a `drawnow`, `figure`, `getframe`, `pause`, or `waitfor` command in the routine.

Parent handle

Uicontextmenu's parent. The handle of the uicontextmenu's parent object. The parent of a uicontextmenu object is the figure in which it appears. You can move a uicontextmenu object to another figure by setting this property to the handle of the new parent.

Position vector

Uicontextmenu's position. A two-element vector that defines the location of a context menu posted by setting the `Visible` property value to on. Specify `Position` as

```
[left bottom]
```

where vector elements represent the distance in pixels from the bottom left corner of the figure window to the top left corner of the context menu.

Selected on | {off}

This property has no effect on uicontextmenu objects.

SelectionHighlight {on} | off

This property has no effect on uicontextmenu objects.

Tag string

User-specified object label. The `Tag` property provides a means to identify graphics objects with a user-specified label. This is particularly useful when constructing interactive graphics programs that would otherwise need to define object handles as global variables or pass them as arguments between callback routines. You can define `Tag` as any string.

Type string

Class of graphics object. For uicontextmenu objects, `Type` is always the string `'uicontextmenu'`.

UIContextMenu handle

This property has no effect on uicontextmenus.

UserData matrix

User-specified data. Any data you want to associate with the uicontextmenu object. MATLAB does not use this data, but you can access it using `set` and `get`.

Visible on | {off}

Uicontextmenu visibility. The `Visible` property can be used in two ways:

- Its value indicates whether the context menu is currently posted. While the context menu is posted, the property value is `on`; when the context menu is not posted, its value is `off`.
- Its value can be set to `on` to force the posting of the context menu. Similarly, setting the value to `off` forces the context menu to be removed. When used in this way, the `Position` property determines the location of the posted context menu.

uicontrol

Purpose Create user interface control object

Syntax

```
handle = uicontrol(parent)
handle = uicontrol(...,'PropertyName',PropertyValue,...)
```

Description `uicontrol` creates uicontrol graphics objects (user interface controls). You implement graphical user interfaces using uicontrols. When selected, most uicontrol objects perform a predefined action. MATLAB supports numerous styles of uicontrols, each suited for a different purpose:

- Check boxes
- Editable text
- Frames
- List boxes
- Pop-up menus
- Push buttons
- Radio buttons
- Sliders
- Static text
- Toggle buttons

Check boxes generate an action when clicked on. These devices are useful when providing the user with a number of independent choices. To activate a check box, click the mouse button on the object. The state of the device is indicated on the display.

Editable text boxes are fields that enable users to enter or modify text values. Use editable text when you want text as input.

On Microsoft Windows systems, if an editable text box has focus, clicking on the menu bar does not cause the editable text callback routine to execute. However, it does cause execution on UNIX systems. Therefore, after clicking on the menu bar, the statement

```
get(edit_handle,'String')
```

does not return the current contents of the edit box on Microsoft Windows systems because MATLAB must execute the callback routine to update the

`String` property (even though the text string has changed on the screen). This behavior is consistent with the respective platform conventions.

Frames are boxes that visually enclose regions of a figure window. Frames can make a user interface easier to understand by visually grouping related controls. Frames have no callback routines associated with them. Only uicontrols can appear within frames.

Frames are opaque, not transparent, so the order you define uicontrols is important in determining whether uicontrols within a frame are covered by the frame or are visible. *Stacking order* determines the order objects are drawn: objects defined first are drawn first; objects defined later are drawn over existing objects. If you use a frame to enclose objects, you must define the frame before you define the objects.

List boxes display a list of items (defined using the `String` property) and enable users to select one or more items. The `Min` and `Max` properties control the selection mode. The `Value` property indicates selected entries and contains the indices into the list of strings; a vector value indicates multiple selections. MATLAB evaluates the list box's callback routine after any mouse button up event that changes the `Value` property. Therefore, you may need to add a "Done" button to delay action caused by multiple clicks on list items. List boxes differentiate between single and double clicks and set the figure `SelectionType` property to `normal` or `open` accordingly before evaluating the list box's `Callback` property.

Pop-up menus open to display a list of choices (defined using the `String` property) when pressed. When not open, a pop-up menu indicates the current choice. Pop-up menus are useful when you want to provide users with a number of mutually exclusive choices, but do not want to take up the amount of space that a series of radio buttons requires. You must specify a value for the `String` property.

Push buttons generate an action when pressed. To activate a push button, click the mouse button on the push button.

Radio buttons are similar to check boxes, but are intended to be mutually exclusive within a group of related radio buttons (i.e., only one is in a pressed state at any given time). To activate a radio button, click the mouse button on the object. The state of the device is indicated on the display. Note that your code can implement the mutually exclusive behavior of radio buttons.

Sliders accept numeric input within a specific range by enabling the user to move a sliding bar. Users move the bar by pressing the mouse button and dragging the pointer over the bar, or by clicking in the trough or on an arrow. The location of the bar indicates a numeric value, which is selected by releasing the mouse button. You can set the minimum, maximum, and current values of the slider.

Static text boxes display lines of text. Static text is typically used to label other controls, provide directions to the user, or indicate values associated with a slider. Users cannot change static text interactively and there is no way to invoke the callback routine associated with it.

Toggle buttons are controls that execute callbacks when clicked on and indicate their state, either on or off. Toggle buttons are useful for building toolbars. More information about toggle buttons.

Remarks

The `uicontrol` function accepts property name/property value pairs, structures, and cell arrays as input arguments and optionally returns the handle of the created object. You can also set and query property values after creating the object using the `set` and `get` functions.

Uicontrol objects are children of figures and therefore do not require an axes to exist when placed in a figure window.

Properties

This table lists all properties useful for `uicontrol` objects, grouping them by function. Each property name acts as a link to a description of the property.

Property Name	Property Description	Property Value
Controlling Style and Appearance		
`BackgroundColor`	Object background color	Value: `ColorSpec` Default: system dependent
`CData`	Truecolor image displayed on the control	Value: matrix
`ForegroundColor`	Color of text	Value: `ColorSpec` Default: `[0 0 0]`

Property Name	Property Description	Property Value
SelectionHighlight	Object highlighted when selected	Value: on, off Default: on
String	Uicontrol label, also list box and pop-up menu items	Value: string
Visible	Uicontrol visibility	Value: on, off Default: on
General Information About the Object		
Children	Uicontrol objects have no children	
Enable	Enable or disable the uicontrol	Value: on, inactive, off Default: on
Parent	Uicontrol object's parent	Value: scalar figure handle
Selected	Whether object is selected	Value: on, off Default: off
SliderStep	Slider step size	Value: two-element vector Default: [0.01 0.1]
Style	Type of uicontrol object	Value: pushbutton, togglebutton, radiobutton, checkbox, edit, text, slider, frame, listbox, popupmenu Default: pushbutton
Tag	User-specified object identifier	Value: string
TooltipString	Content of object's tooltip	Value: string
Type	Class of graphics object	Value: string (read-only) Default: uicontrol
UserData	User-specified data	Value: matrix
Controlling the Object Position		

Property Name	Property Description	Property Value
`Position`	Size and location of uicontrol object	Value: position rectangle Default: `[20 20 60 20]`
`Units`	Units to interpret position vector	Value: `pixels`, `normalized`, `inches`, `centimeters`, `points`, `characters` Default: `pixels`
Controlling Fonts and Labels		
`FontAngle`	Character slant	Value: `normal`, `italic`, `oblique` Default: `normal`
`FontName`	Font family	Value: string Default: system dependent
`FontSize`	Font size	Value: size in `FontUnits` Default: system dependent
`FontUnits`	Font size units	Value: `points`, `normalized`, `inches`, `centimeters`, `pixels` Default: `points`
`FontWeight`	Weight of text characters	Value: `light`, `normal`, `demi`, `bold` Default: `normal`
`HorizontalAlignment`	Alignment of label string	Value: `left`, `center`, `right` Default: depends on uicontrol object
`String`	Uicontrol object label, also list box and pop-up menu items	Value: string
Controlling Callback Routine Execution		
`BusyAction`	Callback routine interruption	Value: `cancel`, `queue` Default: `queue`

Property Name	Property Description	Property Value
ButtonDownFcn	Button press callback routine	Value: string
Callback	Control action	Value: string
CreateFcn	Callback routine executed during object creation	Value: string
DeleteFcn	Callback routine executed during object deletion	Value: string
Interruptible	Callback routine interruption mode	Value: on, off Default: on
UIContextMenu	Uicontextmenu object associated with the uicontrol	Value: handle
Information About the Current State		
ListboxTop	Index of top-most string displayed in list box	Value: scalar Default: [1]
Max	Maximum value (depends on uicontrol object)	Value: scalar Default: object dependent
Min	Minimum value (depends on uicontrol object)	Value: scalar Default: object dependent
Value	Current value of uicontrol object	Value: scalar or vector Default: object dependent
Controlling Access to Objects		
HandleVisibility	Whether handle is accessible from command line and GUIs	Value: on, callback, off Default: on
HitTest	Whether selectable by mouse click	Value: on, off Default: on

Examples

The following statement creates a push button that clears the current axes when pressed:

```
h = uicontrol('Style', 'pushbutton', 'String', 'Clear',...
    'Position', [20 150 100 70], 'Callback', 'cla');
```

You can create a uicontrol object that changes figure colormaps by specifying a pop-up menu and supplying an M-file name as the object's `Callback`:

```
hpop = uicontrol('Style', 'popup',...
       'String', 'hsv|hot|cool|gray',...
       'Position', [20 320 100 50],...
       'Callback', 'setmap');
```

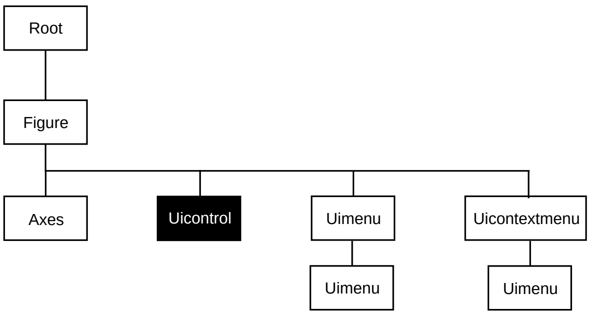
The above call to `uicontrol` defines four individual choices in the menu: `hsv`, `hot`, `cool`, and `gray`. You specify these choices with the `String` property, separating the choices with the "|" character.

The `Callback`, in this case `setmap`, is the name of an M-file that defines a more complicated set of instructions than a single MATLAB command. `setmap` contains these statements:

```
val = get(hpop,'Value');
if val == 1
    colormap(hsv)
elseif val == 2
    colormap(hot)
elseif val == 3
    colormap(cool)
elseif val == 4
    colormap(gray)
end
```

The `Value` property contains a number that indicates the selected choice. The choices are numbered sequentially from one to four. The `setmap` M-file can get and then test the contents of the `Value` property to determine what action to take.

Object Hierarchy



See Also

`textwrap`, `uimenu`

Uicontrol Properties

You can set default uicontrol properties on the root and figure levels:

```
set(0,'DefaultUicontrolProperty',PropertyValue...)
set(gcf,'DefaultUicontrolProperty',PropertyValue...)
```

where *Property* is the name of the uicontrol property whose default value you want to set and `PropertyValue` is the value you are specifying. Use `set` and `get` to access uicontrol properties.

Curly braces { } enclose the default value.

BackgroundColor `ColorSpec`

Object background color. The color used to fill the uicontrol rectangle. Specify a color using a three-element RGB vector or one of MATLAB's predefined names. The default color is determined by system settings. See `ColorSpec` for more information on specifying color.

BusyAction `cancel | {queue}`

Callback routine interruption. If a callback is executing and the user triggers an event (such as a mouse click) on an object for which a callback is defined, that callback attempts to interrupt the first callback. The first callback can be interrupted only at a `drawnow`, `figure`, `getframe`, `pause`, or `waitfor` command; if the callback does not contain any of these commands, it cannot be interrupted.

If the `Interruptible` property of the object whose callback is executing is `off` (the default value is `on`), the callback cannot be interrupted (except by certain callbacks; see the note below). The `BusyAction` property of the object whose callback is waiting to execute determines what happens to the callback:

- If the value is `queue`, the callback is added to the event queue and executes after the first callback finishes execution.
- If the value is `cancel`, the event is discarded and the callback is not executed.

Note If the interrupting callback is a `DeleteFcn` or `CreateFcn` callback or a figure's `CloseRequest` or `ResizeFcn` callback, it interrupts an executing callback regardless of the value of that object's `Interruptible` property. The interrupting callback starts execution at the next `drawnow`, `figure`, `getframe`, `pause`, or `waitfor` statement.

`ButtonDownFcn` string

Button press callback routine. A callback routine that executes whenever you press a mouse button while the pointer is in a five-pixel wide border around the uicontrol. When the uicontrol's `Enable` property is set to `inactive` or `off`, the `ButtonDownFcn` executes when you click the mouse in the five-pixel border or on the control itself. This is useful for implementing actions to interactively modify control object properties, such as size and position, when they are clicked on (using `selectmoveresize`, for example).

Define this routine as a string that is a valid MATLAB expression or the name of an M-file. The expression executes in the MATLAB workspace.

The `Callback` property defines the callback routine that executes when you activate the enabled uicontrol (e.g., click on a push button).

`Callback` string

Control action. A routine that executes whenever you activate the uicontrol object (e.g., when you click on a push button or move a slider). Define this routine as a string that is a valid MATLAB expression or the name of an M-file. The expression executes in the MATLAB workspace.

To execute the callback routine for an editable text control, type in the desired text, then either:

- Move the focus off the object (click the mouse someplace else in the GUI),
- For a single line editable text box, press **Return**, or
- For a multiline editable text box, press **Ctl-Return**.

Callback routines defined for frames and static text do not execute because no action is associated with these objects.

`CData` matrix

Truecolor image displayed on control. A three-dimensional matrix of RGB values that defines a truecolor image displayed on either a push button or toggle button. Each value must be between 0.0 and 1.0. More information about this property.

Children matrix

The empty matrix; uicontrol objects have no children.

Clipping {on} | off

This property has no effect on uicontrols.

CreateFcn string

Callback routine executed during object creation. This property defines a callback routine that executes when MATLAB creates a uicontrol object. You must define this property as a default value for uicontrols. For example, this statement:

```
set(0,'DefaultUicontrolCreateFcn',...
      'set(gcf,''IntegerHandle'',''off'')')
```

defines a default value on the root level that sets the figure `IntegerHandle` property to `off` whenever you create a uicontrol object. MATLAB executes this routine after setting all property values for the uicontrol. Setting this property on an existing uicontrol object has no effect.

The handle of the object whose `CreateFcn` is being executed is accessible only through the root `CallbackObject` property, which can be queried using `gcbo`.

DeleteFcn string

Delete uicontrol callback routine. A callback routine that executes when you delete the uicontrol object (e.g., when you issue a `delete` command or clear the figure containing the uicontrol). MATLAB executes the routine before destroying the object's properties so these values are available to the callback routine.

The handle of the object whose `DeleteFcn` is being executed is accessible only through the root `CallbackObject` property, which you can query using `gcbo`.

Enable {on} | inactive | off

Enable or disable the uicontrol. This property controls how uicontrols respond to mouse button clicks, including which callback routines execute.

- `on` – The uicontrol is operational (the default).
- `inactive` – The uicontrol is not operational, but looks the same as when `Enable` is `on`.
- `off` – The uicontrol is not operational and its label (set by the `string` property) is grayed out.

When you left-click on a uicontrol whose `Enable` property is `on`, MATLAB performs these actions in this order:

1. Sets the figure's `SelectionType` property.
2. Executes the control's `Callback` routine.
3. Does not set the figure's `CurrentPoint` property and does not execute either the control's `ButtonDownFcn` or the figure's `WindowButtonDownFcn` callback.

When you left-click on a uicontrol whose `Enable` property is `inactive` or `off`, or when you right-click on a uicontrol whose `Enable` property has any value, MATLAB performs these actions in this order:

1. Sets the figure's `SelectionType` property.
2. Sets the figure's `CurrentPoint` property.
3. Executes the figure's `WindowButtonDownFcn` callback.
4. On a right-click, if the uicontrol is associated with a context menu, posts the context menu.
5. Executes the control's `ButtonDownFcn` callback.
6. Executes the selected context menu item's `Callback` routine.
7. Does not execute the control's `Callback` routine.

Setting this property to inactive or off enables you to implement object dragging or resizing using the `ButtonDownFcn` callback routine.

`Extent` position rectangle (read only)

Size of uicontrol character string. A four-element vector that defines the size and position of the character string used to label the uicontrol. It has the form:

```
[0,0,width,height]
```

The first two elements are always zero. `width` and `height` are the dimensions of the rectangle. All measurements are in units specified by the `Units` property.

Since the Extent property is defined in the same units as the uicontrol itself, you can use this property to determine proper sizing for the uicontrol with regard to its label. Do this by

- Defining the String property and selecting the font using the relevant properties.
- Getting the value of the Extent property.
- Defining the width and height of the Position property to be somewhat larger than the width and height of the Extent.

For multiline strings, the Extent rectangle encompasses all the lines of text. For single line strings, the Extent is returned as a single line, even if the string wraps when displayed on the control.

FontAngle {normal} | italic | oblique

Character slant. MATLAB uses this property to select a font from those available on your particular system. Setting this property to italic or oblique selects a slanted version of the font, when it is available on your system.

FontName string

Font family. The name of the font in which to display the String. To display and print properly, this must be a font that your system supports. The default font is system dependent.

To use a fixed-width font that looks good in any locale (and displays properly in Japan, where multibyte character sets are used), set FontName to the string FixedWidth (this string value is case sensitive):

```
set(uicontrol_handle, 'FontName', 'FixedWidth')
```

This parameter value eliminates the need to hard code the name of a fixed-width font, which may not display text properly on systems that do not use ASCII character encoding (such as in Japan). A properly written MATLAB application that needs to use a fixed-width font should set FontName to FixedWidth and rely on the root FixedWidthFontName property to be set correctly in the end user's environment.

End users can adapt a MATLAB application to different locales or personal environments by setting the root FixedWidthFontName property to the appropriate value for that locale from startup.m. Setting the root

FixedWidthFontName property causes an immediate update of the display to use the new font.

FontSize size in FontUnits

Font size. A number specifying the size of the font in which to display the String, in units determined by the FontUnits property. The default point size is system dependent.

FontUnits {points} | normalized | inches | centimeters | pixels

Font size units. This property determines the units used by the FontSize property. Normalized units interpret FontSize as a fraction of the height of the uicontrol. When you resize the uicontrol, MATLAB modifies the screen FontSize accordingly. pixels, inches, centimeters, and points are absolute units (1 point = $^1/_{72}$ inch).

FontWeight light | {normal} | demi | bold

Weight of text characters. MATLAB uses this property to select a font from those available on your particular system. Setting this property to bold causes MATLAB to use a bold version of the font, when it is available on your system.

ForegroundColor ColorSpec

Color of text. This property determines the color of the text defined for the String property (the uicontrol label). Specify a color using a three-element RGB vector or one of MATLAB 's predefined names. The default text color is black. See ColorSpec for more information on specifying color.

HandleVisibility {on} | callback | off

Control access to object's handle by command-line users and GUIs. This property determines when an object's handle is visible in its parent's list of children. HandleVisibility is useful for preventing command-line users from accidentally drawing into or deleting a figure that contains only user interface devices (such as a dialog box).

Handles are always visible when HandleVisibility is on.

Setting HandleVisibility to callback causes handles to be visible from within callback routines or functions invoked by callback routines, but not from within functions invoked from the command line. This provides a means to protect GUIs from command-line users, while allowing callback routines to have complete access to object handles.

Setting HandleVisibility to off makes handles invisible at all times. This may be necessary when a callback routine invokes a function that might potentially damage the GUI (such as evaluating a user-typed string), and so temporarily hides its own handles during the execution of that function.

When a handle is not visible in its parent's list of children, it cannot be returned by functions that obtain handles by searching the object hierarchy or querying handle properties. This includes get, findobj, gca, gcf, gco, newplot, cla, clf, and close.

When a handle's visibility is restricted using callback or off, the object's handle does not appear in its parent's Children property, figures do not appear in the root's CurrentFigure property, objects do not appear in the root's CallbackObject property or in the figure's CurrentObject property, and axes do not appear in their parent's CurrentAxes property.

You can set the root ShowHiddenHandles property to on to make all handles visible, regardless of their HandleVisibility settings (this does not affect the values of the HandleVisibility properties).

Handles that are hidden are still valid. If you know an object's handle, you can set and get its properties, and pass it to any function that operates on handles.

HitTest {on} | off

Selectable by mouse click. This property has no effect on uicontrol objects.

HorizontalAlignment left | {center} | right

Horizontal alignment of label string. This property determines the justification of the text defined for the String property (the uicontrol label):

- left — Text is left justified with respect to the uicontrol.
- center — Text is centered with respect to the uicontrol.
- right — Text is right justified with respect to the uicontrol.

On Microsoft Windows systems, this property affects only edit and text uicontrols.

Interruptible {on} | off

Callback routine interruption mode. If a callback is executing and the user triggers an event (such as a mouse click) on an object for which a callback is defined, that callback attempts to interrupt the first callback. MATLAB processes the callbacks according to these factors:

- The `Interruptible` property of the object whose callback is executing
- Whether the executing callback contains `drawnow`, `figure`, `getframe`, `pause`, or `waitfor` statements
- The `BusyAction` property of the object whose callback is waiting to execute

If the `Interruptible` property of the object whose callback is executing is `on` (the default), the callback can be interrupted. The callback interrupts execution at the next `drawnow`, `figure`, `getframe`, `pause`, or `waitfor` statement, and processes the events in the event queue, which includes the waiting callback.

If the `Interruptible` property of the object whose callback is executing is `off`, the callback cannot be interrupted (except by certain callbacks; see the note below). The `BusyAction` property of the object whose callback is waiting to execute determines what happens to the callback.

Note If the interrupting callback is a `DeleteFcn` or `CreateFcn` callback or a figure's `CloseRequest` or `ResizeFcn` callback, it interrupts an executing callback regardless of the value of that object's `Interruptible` property. The interrupting callback starts execution at the next `drawnow`, `figure`, `getframe`, `pause`, or `waitfor` statement. A figure's `WindowButtonDownFcn` callback routine, or an object's `ButtonDownFcn` or `Callback` routine are processed according to the rules described above.

`ListboxTop` scalar

Index of top-most string displayed in list box. This property applies only to the `listbox` style of uicontrol. It specifies which string appears in the top-most position in a list box that is not large enough to display all list entries. `ListboxTop` is an index into the array of strings defined by the `String` property and must have a value between 1 and the number of strings. Noninteger values are fixed to the next lowest integer.

`Max` scalar

Maximum value. This property specifies the largest value allowed for the `Value` property. Different styles of uicontrols interpret `Max` differently:

- Check boxes – `Max` is the setting of the `Value` property while the check box is selected.
- Editable text – If `Max` – `Min` > 1, then editable text boxes accept multiline input. If `Max` – `Min` <= 1, then editable text boxes accept only single line input.
- List boxes – If `Max` – `Min` > 1, then list boxes allow multiple item selection. If `Max` – `Min` <= 1, then list boxes do not allow multiple item selection.
- Radio buttons – `Max` is the setting of the `Value` property when the radio button is selected.
- Sliders – `Max` is the maximum slider value and must be greater than the `Min` property. The default is 1.
- Toggle buttons – `Max` is the value of the `Value` property when the toggle button is selected. The default is 1.
- Frames, pop-up menus, push buttons, and static text do not use the `Max` property.

`Min` scalar

Minimum value. This property specifies the smallest value allowed for the `Value` property. Different styles of uicontrols interpret `Min` differently:

- Check boxes – `Min` is the setting of the `Value` property while the check box is not selected.
- Editable text – If `Max` – `Min` > 1, then editable text boxes accept multiline input. If `Max` – `Min` <= 1, then editable text boxes accept only single line input.
- List boxes – If `Max` – `Min` > 1, then list boxes allow multiple item selection. If `Max` – `Min` <= 1, then list boxes allow only single item selection.
- Radio buttons – `Min` is the setting of the `Value` property when the radio button is not selected.
- Sliders – `Min` is the minimum slider value and must be less than `Max`. The default is 0.

- Toggle buttons – `Min` is the value of the `Value` property when the toggle button is not selected. The default is 0.
- Frames, pop-up menus, push buttons, and static text do not use the `Min` property.

`Parent` handle

Uicontrol's parent. The handle of the uicontrol's parent object. The parent of a uicontrol object is the figure in which it appears. You can move a uicontrol object to another figure by setting this property to the handle of the new parent.

`Position` position rectangle

Size and location of uicontrol. The rectangle defined by this property specifies the size and location of the control within the figure window. Specify `Position` as

```
[left bottom width height]
```

`left` and `bottom` are the distance from the lower-left corner of the figure window to the lower-left corner of the uicontrol object. `width` and `height` are the dimensions of the uicontrol rectangle. All measurements are in units specified by the `Units` property.

On Microsoft Windows systems, the height of pop-up menus is automatically determined by the size of the font. The value you specify for the `height` of the `Position` property has no effect.

`Selected` `on | {off}`

Is object selected. When this property is `on`, MATLAB displays selection handles if the `SelectionHighlight` property is also `on`. You can, for example, define the `ButtonDownFcn` to set this property, allowing users to select the object with the mouse.

`SelectionHighlight` `{on} | off`

Object highlight when selected. When the `Selected` property is `on`, MATLAB indicates the selected state by drawing four edge handles and four corner handles. When `SelectionHighlight` is `off`, MATLAB does not draw the handles.

SliderStep `[min_step max_step]`

Slider step size. This property controls the amount the slider `Value` changes when you click the mouse on the arrow button (`min_step`) or on the slider trough (`max_step`). Specify `SliderStep` as a two-element vector; each value must be in the range `[0, 1]`. The actual step size is a function of the specified `SliderStep` and the total slider range (`Max – Min`). The default, `[0.01 0.10]`, provides a 1 percent change for clicks on the arrow button and a 10 percent change for clicks in the trough.

For example, if you create the following slider,

```
uicontrol('Style','slider','Min',1,'Max',7,...
          'SliderStep',[0.1 0.6])
```

clicking on the arrow button moves the indicator by,

```
0.1*(7-1)
ans =
     0.6000
```

and clicking in the trough moves the indicator by,

```
0.6*(7-1)
ans =
     3.6000
```

Note that if the specified step size moves the slider to a value outside the range, the indicator moves only to the `Max` or `Min` value.

See also the `Max`, `Min`, and `Value` properties.

String string

Uicontrol label, list box items, pop-up menu choices. For check boxes, editable text, push buttons, radio buttons, static text, and toggle buttons, the text displayed on the object. For list boxes and pop-up menus, the set of entries or items displayed in the object.

For uicontrol objects that display only one line of text, if the string value is specified as a cell array of strings or padded string matrix, only the first string of a cell array or of a padded string matrix is displayed; the rest are ignored. Vertical slash (‘`|`’) characters are not interpreted as line breaks and instead show up in the text displayed in the uicontrol.

For multiple line editable text or static text controls, line breaks occur between each row of the string matrix, each cell of a cell array of strings, and after any `\n` characters embedded in the string. Vertical slash ('|') characters are not interpreted as line breaks, and instead show up in the text displayed in the uicontrol.

For multiple items on a list box or pop-up menu, you can specify items as a cell array of strings, a padded string matrix, or within a string vector separated by vertical slash ('|') characters.

For editable text, this property value is set to the string entered by the user.

Style `{pushbutton} | togglebutton | radiobutton | checkbox | edit | text | slider | frame | listbox | popupmenu`

Style of uicontrol object to create. The `Style` property specifies the kind of uicontrol to create. See the "Description" section for information on each type.

Tag string

User-specified object label. The `Tag` property provides a means to identify graphics objects with a user-specified label. This is particularly useful when constructing interactive graphics programs that would otherwise need to define object handles as global variables or pass them as arguments between callback routines. You can define `Tag` as any string.

TooltipString string

Content of tooltip for object. The `TooltipString` property specifies the text of the tooltip associated with the uicontrol. When the user moves the mouse pointer over the control and leaves it there, the tooltip is displayed. More information about this property.

Type string (read only)

Class of graphics object. For uicontrol objects, `Type` is always the string `'uicontrol'`.

UIContextMenu handle

Associate a context menu with uicontrol. Assign this property the handle of a Uicontextmenu object. MATLAB displays the context menu whenever you right-click over the uicontrol. Use the `uicontextmenu` function to create the context menu. More information about this property.

Units {pixels} | normalized | inches | centimeters | points | characters

Units of measurement. The units MATLAB uses to interpret the Extent and Position properties. All units are measured from the lower-left corner of the figure window. Normalized units map the lower-left corner of the figure window to (0,0) and the upper-right corner to (1.0,1.0). pixels, inches, centimeters, and points are absolute units (1 point = 1/72 inch). Character units are characters using the default system font; the width of one character is the width of the letter x, the height of one character is the distance between the baselines of two lines of text. More information about character units.

If you change the value of Units, it is good practice to return it to its default value after completing your computation so as not to affect other functions that assume Units is set to the default value.

UserData matrix

User-specified data. Any data you want to associate with the uicontrol object. MATLAB does not use this data, but you can access it using set and get.

Value scalar or vector

Current value of uicontrol. The uicontrol style determines the possible values this property can have:

- Check boxes set Value to Max when they are on (when selected) and Min when off (not selected).
- List boxes set Value to a vector of indices corresponding to the selected list entries, where 1 corresponds to the first item in the list.
- Pop-up menus set Value to the index of the item selected, where 1 corresponds to the first item in the menu. The "Examples" section shows how to use the Value property to determine which item has been selected.
- Radio buttons set Value to Max when they are on (when selected) and Min when off (not selected).
- Sliders set Value to the number indicated by the slider bar.
- Toggle buttons set Value to Max when they are down (selected) and Min when up (not selected).
- Editable text, frames, push buttons, and static text do not set this property.

Set the `Value` property either interactively with the mouse or through a call to the `set` function. The display reflects changes made to `Value`.

`Visible` `{on} | off`

Uicontrol visibility. By default, all uicontrols are visible. When set to `off`, the uicontrol is not visible, but still exists and you can query and set its properties.

uigetfile

Purpose Interactively retrieve a filename

Syntax

```
uigetfile
uigetfile('FilterSpec')
uigetfile('FilterSpec','DialogTitle')
uigetfile('FilterSpec','DialogTitle',x,y)
[fname,pname] = uigetfile(...)
```

Description `uigetfile` displays a dialog box used to retrieve a file. The dialog box lists the files and directories in the current directory.

`uigetfile('FilterSpec')` displays a dialog box that lists files in the current directory. `FilterSpec` determines the initial display of files and can be a full filename or include the * wildcard. For example, `'*.m'` (the default) causes the dialog box list to show only MATLAB M-files.

`uigetfile('FilterSpec','DialogTitle')` displays a dialog box that has the title `DialogTitle`.

`uigetfile('FilterSpec','DialogTitle',x,y)` positions the dialog box at position [x,y], where `x` and `y` are the distance in pixel units from the left and top edges of the screen. Note that some platforms may not support dialog box placement.

`[fname,pname] = uigetfile(...)` returns the name and path of the file selected in the dialog box. After you press the **Done** button, `fname` contains the name of the file selected and `pname` contains the name of the path selected. If you press the **Cancel** button or if an error occurs, `fname` and `pname` are set to `0`.

Remarks If you select a file that does not exist, an error dialog appears. You can then enter another filename, or press the **Cancel** button.

Examples This statement displays a dialog box that enables you to retrieve a file. The statement lists all MATLAB M-files within a selected directory. The name and path of the selected file are returned in `fname` and `pname`.

```
[fname,pname] = uigetfile('*.m','Sample Dialog Box')
```

The exact appearance of the dialog box depends on your windowing system.

See Also `uiputfile`

uimenu

Purpose Create menus on figure windows

Syntax

```
uimenu('PropertyName',PropertyValue,...)
uimenu(parent,'PropertyName',PropertyValue,...)
handle = uimenu('PropertyName',PropertyValue,...)
handle = uimenu(parent,'PropertyName',PropertyValue,...)
```

Description uimenu creates a hierarchy of menus and submenus that are displayed in the figure window's menu bar. You can also use uimenu to create menu items for context menus. More information about context menus.

handle = uimenu('*PropertyName*',PropertyValue,...) creates a menu in the current figure's menu bar using the values of the specified properties and assigns the menu handle to handle.

handle = uimenu(parent,'*PropertyName*',PropertyValue,...) creates a submenu of a parent menu or a menu item on a context menu specified by parent and assigns the menu handle to handle. If parent refers to a figure instead of another uimenu object or a Uicontextmenu, MATLAB creates a new menu on the referenced figure's menu bar.

Remarks MATLAB adds the new menu to the existing menu bar. Each menu choice can itself be a menu that displays its submenu when selected.

uimenu accepts property name/property value pairs, as well as structures and cell arrays of properties as input arguments. The uimenu Callback property defines the action taken when you activate the menu item. uimenu optionally returns the handle to the created uimenu object.

Uimenus only appear in figures whose WindowStyle is normal. If a figure containing uimenu children is changed to WindowStyle modal, the uimenu children still exist and are contained in the Children list of the figure, but are not displayed until the WindowStyle is changed to normal.

The value of the figure MenuBar property affects the location of the uimenu on the figure menu bar. When MenuBar is figure, a set of built-in menus precedes the uimenus on the menu bar (MATLAB controls the built-in menus and their handles are not available to the user). When MenuBar is none, uimenus are the only items on the menu bar (that is, the built-in menus do not appear).

You can set and query property values after creating the menu using `set` and `get`.

Properties

This table lists all properties useful to `uimenu` objects, grouping them by function. Each property name acts as a link to a description of the property.

Property Name	Property Description	Property Value
Controlling Style and Appearance		
`Checked`	Menu check indicator	Value: `on`, `off` Default: `off`
`ForegroundColor`	Color of text	Value: `ColorSpec` Default: `[0 0 0]`
`Label`	Menu label	Value: string
`SelectionHighlight`	Object highlighted when selected	Value: `on`, `off` Default: `on`
`Separator`	Separator line mode	Value: `on`, `off` Default: `off`
`Visible`	Uimenu visibility	Value: `on`, `off` Default: `on`
General Information About the Object		
`Accelerator`	Keyboard equivalent	Value: character
`Children`	Handles of submenus	Value: vector of handles
`Enable`	Enable or disable the uimenu	Value: `on`, `off` Default: `on`
`Parent`	Uimenu object's parent	Value: handle
`Tag`	User-specified object identifier	Value: string
`Type`	Class of graphics object	Value: string (read-only) Default: `uimenu`

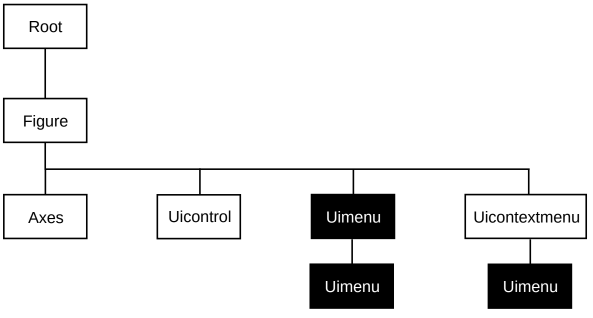
Property Name	Property Description	Property Value
UserData	User-specified data	Value: matrix
Controlling the Object Position		
Position	Relative uimenu position	Value: scalar Default: [1]
Controlling Callback Routine Execution		
BusyAction	Callback routine interruption	Value: cancel, queue Default: queue
ButtonDownFcn	Button press callback routine	Value: string
Callback	Control action	Value: string
CreateFcn	Callback routine executed during object creation	Value: string
DeleteFcn	Callback routine executed during object deletion	Value: string
Interruptible	Callback routine interruption mode	Value: on, off Default: on
Controlling Access to Objects		
HandleVisibility	Whether handle is accessible from command line and GUIs	Value: on, callback, off Default: on
HitTest	Whether selectable by mouse click	Value: on, off Default: on

Examples

This example creates a menu labeled **Workspace** whose choices allow users to create a new figure window, save workspace variables, and exit out of MATLAB. In addition, it defines an accelerator key for the Quit option.

```
f = uimenu('Label','Workspace');
    uimenu(f,'Label','New Figure','Callback','figure');
    uimenu(f,'Label','Save','Callback','save');
    uimenu(f,'Label','Quit','Callback','exit',...
           'Separator','on','Accelerator','Q');
```

Object Hierarchy



See Also

`uicontrol`, `uicontextmenu`, `gcbo`, `set`, `get`, `figure`

Uimenu Properties

This section lists property names along with the type of values each accepts. Curly braces { } enclose default values.

You can set default uimenu properties on the figure and root levels:

```
set(0,'DefaultUimenuPropertyName',PropertyValue...)
set(gcf,'DefaultUimenuPropertyName',PropertyValue...)
set(menu_handle,'DefaultUimenuProperty',PropertyValue...)
```

Where *PropertyName* is the name of the uimenu property and `PropertyValue` is the value you are specifying. Use `set` and `get` to access uimenu properties.

Accelerator character

Keyboard equivalent. A character specifying the keyboard equivalent for the menu item. This allows users to select a particular menu choice by pressing the specified character in conjunction with another key, instead of selecting the menu item with the mouse. The key sequence is platform specific:

- For Microsoft Windows systems, the sequence is **Ctrl-**`Accelerator`. These keys are reserved for default menu items: c, v, and x.
- For UNIX systems, the sequence is **Ctrl-**`Accelerator`. These keys are reserved for default menu items: o, p, s, and w.

You can define an accelerator only for menu items that do not have children menus. Accelerators work only for menu items that directly execute a callback routine, not items that bring up other menus.

Note that the menu item does not have to be displayed (e.g., a submenu) for the accelerator key to work. However, the window focus must be in the figure when the key sequence is entered.

BusyAction `cancel | {queue}`

Callback routine interruption. If a callback is executing and the user triggers an event (such as a mouse click) on an object for which a callback is defined, that callback attempts to interrupt the first callback. The first callback can be interrupted only at a `drawnow`, `figure`, `getframe`, `pause`, or `waitfor` command; if the callback does not contain any of these commands, it cannot be interrupted.

If the `Interruptible` property of the object whose callback is executing is `off` (the default value is on), the callback cannot be interrupted (except by certain callbacks; see the note below). The `BusyAction` property of the object whose callback is waiting to execute determines what happens to the callback:

- If the value is `queue`, the callback is added to the event queue and executes after the first callback finishes execution.
- If the value is `cancel`, the event is discarded and the callback is not executed.

Note If the interrupting callback is a `DeleteFcn` or `CreateFcn` callback or a figure's `CloseRequest` or `ResizeFcn` callback, it interrupts an executing callback regardless of the value of that object's `Interruptible` property. The interrupting callback starts execution at the next `drawnow`, `figure`, `getframe`, `pause`, or `waitfor` statement.

`ButtonDownFcn` string

The button down function has no effect on uimenu objects.

`Callback` string

Menu action. A callback routine that executes whenever you select the menu. Define this routine as a string that is a valid MATLAB expression or the name of an M-file. The expression executes in the MATLAB workspace.

A menu with children (submenus) executes its callback routine before displaying the submenus. A menu without children executes its callback routine when you *release* the mouse button (i.e., on the button up event).

`Checked` `on | {off}`

Menu check indicator. Setting this property to `on` places a check mark next to the corresponding menu item. Setting it to `off` removes the check mark. You can use this feature to create menus that indicate the state of a particular option. Note that there is no formal mechanism for indicating that an unchecked menu item will become checked when selected. Also, this property does not check top level menus or submenus, although you can change the value of the property for these menus.

Children vector of handles

Handles of submenus. A vector containing the handles of all children of the uimenu object. The children objects of uimenus are other uimenus, which function as submenus. You can use this property to re-order the menus.

Clipping {on} | off

`Clipping` has no effect on uimenu objects.

CreateFcn string

Callback routine executed during object creation. This property defines a callback routine that executes when MATLAB creates a uimenu object. You must define this property as a default value for uimenus. For example, the statement,

```
set(0,'DefaultUimenuCreateFcn','set(gcf,''IntegerHandle'',...
''off''’))
```

defines a default value on the root level that sets the figure `IntegerHandle` property to `off` whenever you create a uimenu object. Setting this property on an existing uimenu object has no effect. MATLAB executes this routine after setting all property values for the uimenu.

The handle of the object whose `CreateFcn` is being executed is accessible only through the root `CallbackObject` property, which can be queried using `gcbo`.

DeleteFcn string

Delete uimenu callback routine. A callback routine that executes when you delete the uimenu object (e.g., when you issue a `delete` command or cause the figure containing the uimenu to reset). MATLAB executes the routine before destroying the object’s properties so these values are available to the callback routine.

The handle of the object whose `DeleteFcn` is being executed is accessible only through the root `CallbackObject` property, which is more simply queried using `gcbo`.

Enable {on} | off

Enable or disable the uimenu. This property controls whether a menu item can be selected. When not enabled (set to `off`), the menu `Label` appears dimmed, indicating the user cannot select it.

ForegroundColor ColorSpec X-Windows only

Color of menu label string. This property determines color of the text defined for the Label property. Specify a color using a three-element RGB vector or one of MATLAB's predefined names. The default text color is black. See ColorSpec for more information on specifying color.

HandleVisibility {on} | callback | off

Control access to object's handle by command-line users and GUIs. This property determines when an object's handle is visible in its parent's list of children. HandleVisibility is useful for preventing command-line users from accidentally drawing into or deleting a figure that contains only user interface devices (such as a dialog box).

Handles are always visible when HandleVisibility is on.

Setting HandleVisibility to callback causes handles to be visible from within callback routines or functions invoked by callback routines, but not from within functions invoked from the command line. This provide a means to protect GUIs from command-line users, while allowing callback routines to have complete access to object handles.

Setting HandleVisibility to off makes handles invisible at all times. This may be necessary when a callback routine invokes a function that might potentially damage the GUI (such as evaluating a user-typed string), and so temporarily hides its own handles during the execution of that function.

When a handle is not visible in its parent's list of children, it cannot be returned by functions that obtain handles by searching the object hierarchy or querying handle properties. This includes get, findobj, gca, gcf, gco, newplot, cla, clf, and close.

When a handle's visibility is restricted using callback or off, the object's handle does not appear in its parent's Children property, figures do not appear in the root's CurrentFigure property, objects do not appear in the root's CallbackObject property or in the figure's CurrentObject property, and axes do not appear in their parent's CurrentAxes property.

You can set the root ShowHiddenHandles property to on to make all handles visible, regardless of their HandleVisibility settings (this does not affect the values of the HandleVisibility properties).

Handles that are hidden are still valid. If you know an object's handle, you can set and get its properties, and pass it to any function that operates on handles.

Interruptible {on} | off

Callback routine interruption mode. If a callback is executing and the user triggers an event (such as a mouse click) on an object for which a callback is defined, that callback attempts to interrupt the first callback. MATLAB processes the callbacks according to these factors:

- The Interruptible property of the object whose callback is executing
- Whether the executing callback contains drawnow, figure, getframe, pause, or waitfor statements
- The BusyAction property of the object whose callback is waiting to execute

If the Interruptible property of the object whose callback is executing is on (the default), the callback can be interrupted. The callback interrupts execution at the next drawnow, figure, getframe, pause, or waitfor statement, and processes the events in the event queue, which includes the waiting callback.

If the Interruptible property of the object whose callback is executing is off, the callback cannot be interrupted (except by certain callbacks; see the note below). The BusyAction property of the object whose callback is waiting to execute determines what happens to the callback.

Note If the interrupting callback is a DeleteFcn or CreateFcn callback or a figure's CloseRequest or ResizeFcn callback, it interrupts an executing callback regardless of the value of that object's Interruptible property. The interrupting callback starts execution at the next drawnow, figure, getframe, pause, or waitfor statement. A figure's WindowButtonDownFcn callback routine, or an object's ButtonDownFcn or Callback routine are processed according to the rules described above.

Label string

Menu label. A string specifying the text label on the menu item. You can specify a mnemonic using the "&" character. Whatever character follows the "&" in the string appears underlined and selects the menu item when you type that character while the menu is visible. The "&" character is not displayed. To display the "&" character in a label, use two "&" characters in the string:

`'O&pen selection'` yields **Open selection**

`'Save && Go'` yields **Save & Go**

`Parent` handle

Uimenu's parent. The handle of the uimenu's parent object. The parent of a uimenu object is the figure on whose menu bar it displays, or the uimenu of which it is a submenu. You can move a uimenu object to another figure by setting this property to the handle of the new parent.

`Position` scalar

Relative menu position. The value of `Position` indicates placement on the menu bar or within a menu. Top-level menus are placed from left to right on the menu bar according to the value of their `Position` property, with 1 representing the left-most position. The individual items within a given menu are placed from top to bottom according to the value of their `Position` property, with 1 representing the top-most position.

`Selected` `on | {off}`

This property is not used for uimenu objects.

`SelectionHighlight` `on | off`

This property is not used for uimenu objects.

`Separator` `on | {off}`

Separator line mode. Setting this property to `on` draws a dividing line above the menu item.

`Tag` string

User-specified object label. The `Tag` property provides a means to identify graphics objects with a user-specified label. This is particularly useful when constructing interactive graphics programs that would otherwise need to define object handles as global variables or pass them as arguments between callback routines. You can define `Tag` as any string.

`Type` string (read only)

Class of graphics object. For uimenu objects, `Type` is always the string `'uimenu'`.

`UserData` matrix

User-specified data. Any matrix you want to associate with the uimenu object. MATLAB does not use this data, but you can access it using the `set` and `get` commands.

`Visible` `{on} | off`

Uimenu visibility. By default, all uimenus are visible. When set to `off`, the uimenu is not visible, but still exists and you can query and set its properties.

Purpose

Interactively select a file for writing

Syntax

```
uiputfile
uiputfile('InitFile')
uiputfile('InitFile','DialogTitle')
uiputfile('InitFile','DialogTitle',x,y)
[fname,pname] = uiputfile(...)
```

Description

`uiputfile` displays a dialog box used to select a file for writing. The dialog box lists the files and directories in the current directory.

`uiputfile('InitFile')` displays a dialog box that contains a list of files in the current directory determined by `InitFile`. `InitFile` is a full filename or includes the * wildcard. For example, specifying `'*.m'` (the default) causes the dialog box list to show only MATLAB M-files.

`uiputfile('InitFile','DialogTitle')` displays a dialog box that has the title `DialogTitle`.

`uiputfile('InitFile','DialogTitle',x,y)` positions the dialog box at screen position [x,y], where `x` and `y` are the distance in pixel units from the left and top edges of the screen. Note that positioning may not work on all platforms.

`[fname,pname] = uiputfile(...)` returns the name and path of the file selected in the dialog box. If you press the **Cancel** button or an error occurs, `fname` and `pname` are set to `0`.

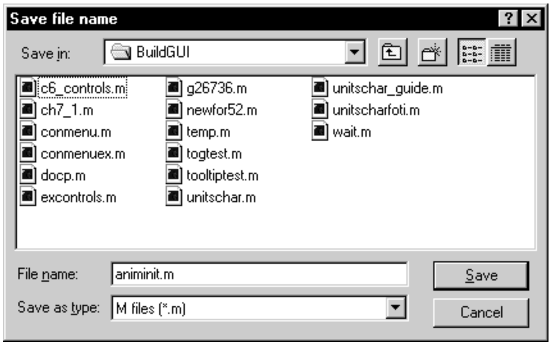
Remarks

If you select a file that already exists, a prompt asks whether you want to overwrite the file. If you choose to, the function successfully returns but does not delete the existing file (which is the responsibility of the calling routines). If you select **Cancel** in response to the prompt, the function returns control back to the dialog box so you can enter another filename.

Examples

This statement displays a dialog box titled 'Save file name' (the exact appearance of the dialog box depends on your windowing system) with the filename animinit.m.

```
[newfile,newpath] = uiputfile('animinit.m','Save file name');
```



Microsoft Windows

See Also

uigetfile

Purpose Control program execution

Syntax

```
uiwait(h)
uiwait
uiresume(h)
```

Description The `uiwait` and `uiresume` functions block and resume MATLAB program execution.

`uiwait` blocks execution until `uiresume` is called or the current figure is deleted. This syntax is the same as `uiwait(gcf)`.

`uiwait(h)` blocks execution until `uiresume` is called or the figure `h` is deleted.

`uiresume(h)` resumes the M-file execution that `uiwait` suspended.

Remarks When creating a dialog, you should have a uicontrol with a callback that calls `uiresume` or a callback that destroys the dialog box. These are the only methods that resume program execution after the `uiwait` function blocks execution.

`uiwait` is a convenient way to use the `waitfor` command. You typically use it in conjunction with a dialog box. It provides a way to block the execution of the M-file that created the dialog, until the user responds to the dialog box. When used in conjunction with a modal dialog, `uiwait/uiresume` can block the execution of the M-file *and* restrict user interaction to the dialog only.

See Also `uicontrol`, `uimenu`, `waitfor`, `figure`, `dialog`

uisetcolor

Purpose Set an object's `ColorSpec` from a dialog box interactively

Syntax

```
c = uisetcolor(h_or_c, 'DialogTitle');
```

Description `uisetcolor` displays a dialog box for the user to fill in, then applies the selected color to the appropriate property of the graphics object identified by the first argument.

`h_or_c` can be either a handle to a graphics object or an RGB triple. If you specify a handle, it must specify a graphics object that have a `Color` property. If you specify a color, it must be a valid RGB triple (e.g., [1 0 0] for red). The color specified is used to initialize the dialog box. If no initial RGB is specified, the dialog box initializes the color to black.

`DialogTitle` is a string that is used as the title of the dialog box.

`c` is the RGB value selected by the user. If the user presses **Cancel** from the dialog box, or if any error occurs, `c` is set to the input RGB triple, if provided; otherwise, it is set to 0.

See Also `ColorSpec`

Purpose Modify font characteristics for objects interactively

Syntax

```
uisetfont
uisetfont(h)
uisetfont(S)
uisetfont(h,'DialogTitle')
uisetfont(S,'DialogTitle')
S = uisetfont(...)
```

Description `uisetfont` enables you to change font properties (`FontName`, `FontUnits`, `FontSize`, `FontWeight`, and `FontAngle`) for a text, axes, or uicontrol object. The function returns a structure consisting of font properties and values. You can specify an alternate title for the dialog box.

`uisetfont` displays the dialog box and returns the selected font properties.

`uisetfont(h)` displays a dialog box, initializing the font property values with the values of those properties for the object whose handle is `h`. Selected font property values are applied to the current object. If a second argument is supplied, it specifies a name for the dialog box.

`uisetfont(S)` displays a dialog box, initializing the font property values with the values defined for the specified structure (`S`). `S` must define legal values for one or more of these properties: `FontName`, `FontUnits`, `FontSize`, `FontWeight`, and `FontAngle` and the field names must match the property names exactly. If other properties are defined, they are ignored. If a second argument is supplied, it specifies a name for the dialog box.

`uisetfont('DialogTitle')` displays a dialog box with the title `DialogTitle` and returns the values of the font properties selected in the dialog box.

If a left-hand argument is specified, the properties `FontName`, `FontUnits`, `FontSize`, `FontWeight`, and `FontAngle` are returned as fields in a structure. If the user presses **Cancel** from the dialog box or if an error occurs, the output value is set to 0.

Example These statements create a text object, then display a dialog box (labeled Update Font) that enables you to change the font characteristics:

```
h = text(.5,.5,'Figure Annotation');
uisetfont(h,'Update Font')
```

These statements create two push buttons, then set the font properties of one based on the values set for the other:

```
% Create push button with string ABC
c1 = uicontrol('Style', 'pushbutton', ...
     'Position', [10 10 100 20], 'String', 'ABC');
% Create push button with string XYZ
c2 = uicontrol('Style', 'pushbutton', ...
     'Position', [10 50 100 20], 'String', 'XYZ');
% Display set font dialog box for c1, make selections, save to d
d = uisetfont(c1)
% Apply those settings to c2
set(c2, d)
```

See Also `axes`, `text`, `uicontrol`

Purpose Viewpoint specification

Syntax

```
view(az,el)
view([az,el])
view([x,y,z])
view(2)
view(3)
view(T)

[az,el] = view
T = view
```

Description The position of the viewer (the viewpoint) determines the orientation of the axes. You specify the viewpoint in terms of azimuth and elevation, or by a point in three-dimensional space.

`view(az,el)` and `view([az,el])` set the viewing angle for a three-dimensional plot. The azimuth, `az`, is the horizontal rotation about the *z*-axis as measured in degrees from the negative *y*-axis. Positive values indicate counterclockwise rotation of the viewpoint. `el` is the vertical elevation of the viewpoint in degrees. Positive values of elevation correspond to moving above the object; negative values correspond to moving below the object.

`view([x,y,z])` sets the viewpoint to the Cartesian coordinates `x`, `y`, and `z`. The magnitude of `(x,y,z)` is ignored.

`view(2)` sets the default two-dimensional view, `az = 0`, `el = 90`.

`view(3)` sets the default three-dimensional view, `az = –37.5`, `el = 30`.

`view(T)` sets the view according to the transformation matrix `T`, which is a 4-by-4 matrix such as a perspective transformation generated by `viewmtx`.

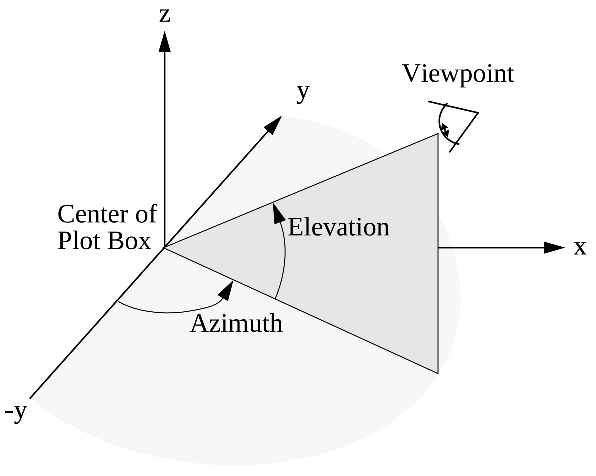
`[az,el] = view` returns the current azimuth and elevation.

`T = view` returns the current 4-by-4 transformation matrix.

Remarks

Azimuth is a polar angle in the *x*-*y* plane, with positive angles indicating counterclockwise rotation of the viewpoint. Elevation is the angle above (positive angle) or below (negative angle) the *x*-*y* plane.

This diagram illustrates the coordinate system. The arrows indicate positive directions.



Examples

View the object from directly overhead.

```
az = 0;
el = 90;
view(az, el);
```

Set the view along the *y*-axis, with the *x*-axis extending horizontally and the *z*-axis extending vertically in the figure.

```
view([0 0]);
```

Rotate the view about the *z*-axis by 180°.

```
az = 180;
el = 90;
view(az, el);
```

See Also

`viewmtx`, `axes`

axes graphics object properties: CameraPosition, CameraTarget, CameraViewAngle, Projection.

viewmtx

Purpose View transformation matrices

Syntax

```
T = viewmtx(az,el)
T = viewmtx(az,el,phi)
T = viewmtx(az,el,phi,xc)
```

Description

`viewmtx` computes a 4-by-4 orthographic or perspective transformation matrix that projects four-dimensional homogeneous vectors onto a two-dimensional view surface (e.g., your computer screen).

`T = viewmtx(az,el)` returns an *orthographic* transformation matrix corresponding to azimuth `az` and elevation `el`. `az` is the azimuth (i.e., horizontal rotation) of the viewpoint in degrees. `el` is the elevation of the viewpoint in degrees. This returns the same matrix as the commands

```
view(az,el)
T = view
```

but does not change the current view.

`T = viewmtx(az,el,phi)` returns a *perspective* transformation matrix. `phi` is the perspective viewing angle in degrees. `phi` is the subtended view angle of the normalized plot cube (in degrees) and controls the amount of perspective distortion.

Phi	Description
0 degrees	Orthographic projection
10 degrees	Similar to telephoto lens
25 degrees	Similar to normal lens
60 degrees	Similar to wide angle lens

You can use the matrix returned to set the view transformation with `view(T)`. The 4-by-4 perspective transformation matrix transforms four-dimensional homogeneous vectors into unnormalized vectors of the form (x,y,z,w), where w is not equal to 1. The x- and y-components of the normalized vector (x/w, y/w, z/w, 1) are the desired two-dimensional components (see example below).

`T = viewmtx(az,el,phi,xc)` returns the perspective transformation matrix using `xc` as the target point within the normalized plot cube (i.e., the camera is looking at the point `xc`). `xc` is the target point that is the center of the view. You specify the point as a three-element vector, `xc = [xc,yc,zc]`, in the interval [0,1]. The default value is `xc = [0,0,0]`.

Remarks

A four-dimensional homogenous vector is formed by appending a 1 to the corresponding three-dimensional vector. For example, `[x,y,z,1]` is the four-dimensional vector corresponding to the three-dimensional point `[x,y,z]`.

Examples

Determine the projected two-dimensional vector corresponding to the three-dimensional point (0.5,0.0,–3.0) using the default view direction. Note that the point is a column vector.

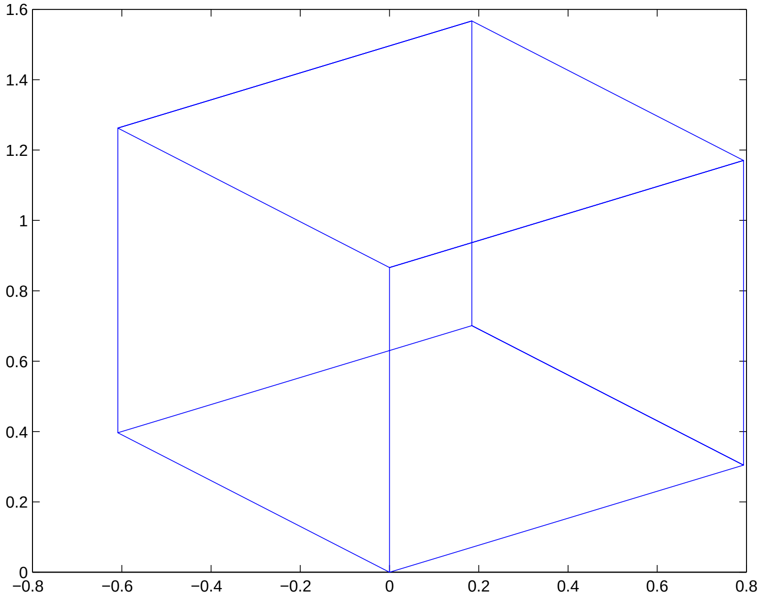
```
A = viewmtx(–37.5,30);
x4d = [.5  0  –3  1]';
x2d = A∗x4d;
x2d = x2d(1:2)
x2d =
     0.3967
    –2.4459
```

Vectors that trace the edges of a unit cube are

```
x = [0  1  1  0  0  0  1  1  0  0  1  1  1  1  0  0];
y = [0  0  1  1  0  0  0  1  1  0  0  0  1  1  1  1];
z = [0  0  0  0  0  1  1  1  1  1  1  0  0  1  1  0];
```

Transform the points in these vectors to the screen, then plot the object.

```
A = viewmtx(–37.5,30);
[m,n] = size(x);
x4d = [x(:),y(:),z(:),ones(m*n,1)]';
x2d = A*x4d;
x2 = zeros(m,n); y2 = zeros(m,n);
x2(:) = x2d(1,:);
y2(:) = x2d(2,:);
plot(x2,y2)
```

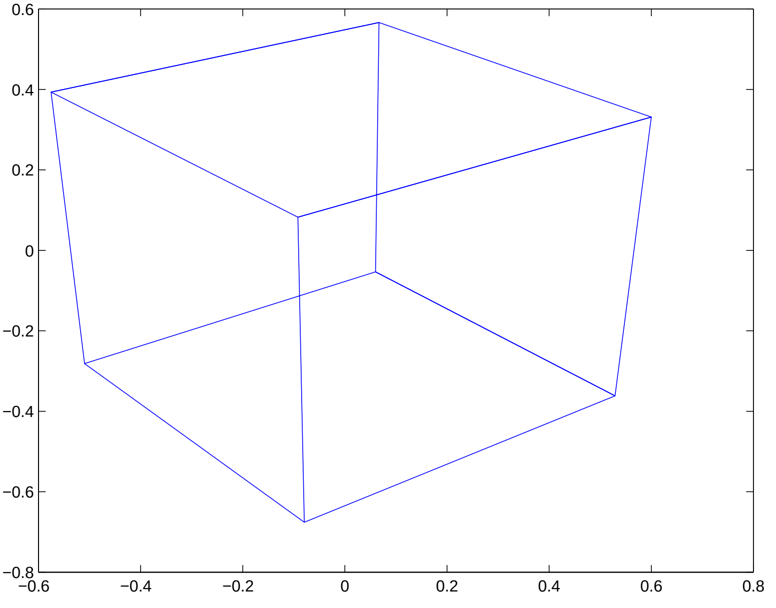


Use a perspective transformation with a 25 degree viewing angle:

```
A = viewmtx(–37.5,30,25);
x4d = [.5  0  –3  1]';
x2d = A∗x4d;
x2d = x2d(1:2)/x2d(4)   % Normalize
x2d =
     0.1777
    –1.8858
```

Transform the cube vectors to the screen and plot the object:

```
A = viewmtx(-37.5,30,25);
[m,n] = size(x);
x4d = [x(:),y(:),z(:),ones(m*n,1)]';
x2d = A*x4d;
x2 = zeros(m,n); y2 = zeros(m,n);
x2(:) = x2d(1,:)./x2d(4,:);
y2(:) = x2d(2,:)./x2d(4,:);
plot(x2,y2)
```



See Also

`view`

waitbar

Purpose Display waitbar

Syntax `h = waitbar(x,'title')`

Description A waitbar shows what percentage of a calculation is complete, as the calculation proceeds.

`h = waitbar(x,'title')` creates and displays a waitbar of fractional length `x`. The handle to the waitbar figure is returned in `h`. `x` should be between 0 and 1. Each subsequent call to waitbar, `waitbar(x)`, extends the length of the bar to the new position `x`.

Example `waitbar` is typically used inside a `for` loop that performs a lengthy computation. For example,

```
h = waitbar(0,'Please wait...');

for i=1:100, % computation here %
waitbar(i/100)
end

close(h)
```

Purpose Wait for condition

Syntax

```
waitfor(h)
waitfor(h,'PropertyName')
waitfor(h,'PropertyName',PropertyValue)
```

Description The `waitfor` function blocks the caller's execution stream so that command-line expressions, callbacks, and statements in the blocked M-file do not execute until a specified condition is satisfied.

`waitfor(h)` returns when the graphics object identified by `h` is deleted or when a **Ctrl-C** is typed in the Command Window. If `h` does not exist, `waitfor` returns immediately without processing any events.

`waitfor(h,'PropertyName')`, in addition to the conditions in the previous syntax, returns when the value of `'PropertyName'` for the graphics object `h` changes. If `'PropertyName'` is not a valid property for the object, `waitfor` returns immediately without processing any events.

`waitfor(h,'PropertyName',PropertyValue)`, in addition to the conditions in the previous syntax, `waitfor` returns when the value of `'PropertyName'` for the graphics object `h` changes to `PropertyValue`. `waitfor` returns immediately without processing any events if `'PropertyName'` is set to `PropertyValue`.

Remarks While `waitfor` blocks an execution stream, other execution streams in the form of callbacks may execute as a result of various events (e.g., pressing a mouse button).

`waitfor` can block nested execution streams. For example, a callback invoked during a `waitfor` statement can itself invoke `waitfor`.

See Also `uiresume`, `uiwait`

waitforbuttonpress

Purpose Wait for key or mouse button press

Syntax

```
k = waitforbuttonpress
```

Description `k = waitforbuttonpress` blocks the caller's execution stream until the function detects that the user has pressed a mouse button or a key while the figure window is active. The function returns

- `0` if it detects a mouse button press
- `1` if it detects a key press

Additional information about the event that causes execution to resume is available through the figure's `CurrentCharacter`, `SelectionType`, and `CurrentPoint` properties.

If a `WindowButtonDownFcn` is defined for the figure, its callback is executed before `waitforbuttonpress` returns a value.

Example These statements display text in the Command Window when the user either clicks a mouse button or types a key in the figure window:

```
w = waitforbuttonpress;
if w == 0
    disp('Button press')
else
    disp('Key press')
end
```

See Also `dragrect`, `figure`, `gcf`, `ginput`, `rbbox`, `waitfor`

Purpose Display warning dialog box

Syntax

```
h = warndlg('warningstring','dlgname')
```

Description

`warndlg` displays a dialog box named `'Warning Dialog'` containing the string `'This is the default warning string.'` The warning dialog box disappears after you press the **OK** button.

`warndlg('warningstring')` displays a dialog box with the title `'Warning Dialog'` containing the string specified by `warningstring`.

`warndlg('warningstring','dlgname')` displays a dialog box with the title `dlgname` that contains the string `warningstring`.

`h = warndlg(...)` returns the handle of the dialog box.

Examples

The statement

```
warndlg('Pressing OK will clear memory','!! Warning !!')
```

displays this dialog box:

See Also

`dialog`, `errordlg`, `helpdlg`, `msgbox`

Purpose Waterfall plot

Syntax

```
waterfall(Z)
waterfall(X,Y,Z)
waterfall(...,C)

h = waterfall(...)
```

Description The `waterfall` function draws a mesh similar to the `meshz` function, but it does not generate lines from the columns of the matrices. This produces a “waterfall” effect.

`waterfall(Z)` creates a waterfall plot using `x = 1:size(Z,1)` and `y = 1:size(Z,1)`. `Z` determines the color, so color is proportional to surface height.

`waterfall(X,Y,Z)` creates a waterfall plot using the values specified in `X`, `Y`, and `Z`. `Z` also determines the color, so color is proportional to the surface height. If `X` and `Y` are vectors, `X` corresponds to the columns of `Z`, and `Y` corresponds to the rows, where `length(x) = n`, `length(y) = m`, and `[m,n] = size(Z)`. `X` and `Y` are vectors or matrices that define the *x* and *y* coordinates of the plot. `Z` is a matrix that defines the *z* coordinates of the plot (i.e., height above a plane). If `C` is omitted, color is proportional to `Z`.

`waterfall(...,C)` uses scaled color values to obtain colors from the current colormap. Color scaling is determined by the range of `C`, which must be the same size as `Z`. MATLAB performs a linear transformation on `C` to obtain colors from the current colormap.

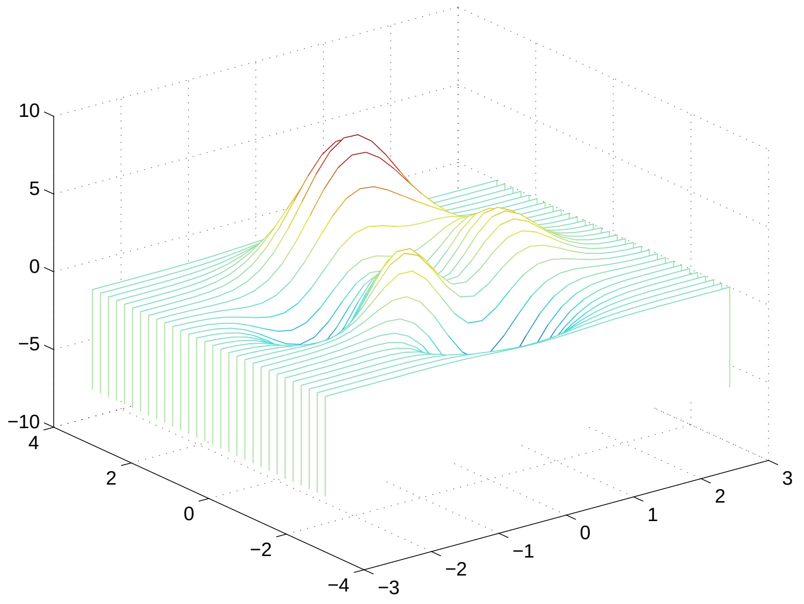
`h = waterfall(...)` returns the handle of the patch graphics object used to draw the plot.

Remarks For column-oriented data analysis, use `waterfall(Z')` or `waterfall(X',Y',Z')`.

Examples

Produce a waterfall plot of the peaks function.

```
[X,Y,Z] = peaks(30);
waterfall(X,Y,Z)
```



Algorithm

The range of X, Y, and Z, or the current setting of the axes Llim, YLim, and ZLim properties, determines the range of the axes (also set by axis). The range of C, or the current setting of the axes Clim property, determines the color scaling (also set by caxis).

The CData property for the patch graphics objects specifies the color at every point along the edge of the patch, which determines the color of the lines.

The waterfall plot looks like a mesh surface; however, it is a patch graphics object. To create a surface plot similar to waterfall, use the meshz function and set the MeshStyle property of the surface to 'Row'. For a discussion of parametric surfaces and related color properties, see surf.

See Also

axes, axis, caxis, meshz, surf

Properties for patch graphics objects.

whitebg

Purpose

Change axes background color

Syntax

```
whitebg
whitebg(h)
whitebg(ColorSpec)
whitebg(h,ColorSpec)
```

Description

`whitebg` complements the colors in the current figure.

`whitebg(h)` complements colors in all figures specified in the vector `h`.

`whitebg(ColorSpec)` and `whitebg(h,ColorSpec)` change the color of the axes, which are children of the figure, to the color specified by `ColorSpec`.

Remarks

`whitebg` changes the colors of the figure's children, with the exception of shaded surfaces. This ensures that all objects are visible against the new background color. `whitebg` sets the default properties on the root such that all subsequent figures use the new background color.

Examples

Set the background color to blue-gray.

```
whitebg([0 .5 .6])
```

Set the background color to blue.

```
whitebg('blue')
```

See Also

`ColorSpec`

The figure graphics object property `InvertHardCopy`.

Purpose Label the *x*-, *y*-, and *z*-axis

Syntax

```
xlabel('string')
xlabel(fname)
xlabel(...,'PropertyName',PropertyValue,...)
h = xlabel(...)

ylabel(...)
h = ylabel(...)

zlabel(...)
h = zlabel(...)
```

Description Each axes graphics object can have one label for the *x*-, *y*-, and *z*-axis. The label appears beneath its respective axis in a two-dimensional plot and to the side or beneath the axis in a three-dimensional plot.

`xlabel('string')` labels the *x*-axis of the current axes.

`xlabel(fname)` evaluates the function `fname`, which must return a string, then displays the string beside the *x*-axis.

`xlabel(...,'PropertName',PropertyValue,...)` specifies property name and property value pairs for the text graphics object created by `xlabel`.

`h = xlabel(...)`, `h = ylabel(...)`, and `h = zlabel(...)` return the handle to the text object used as the label.

`ylabel(...)` and `zlabel(...)` label the *y*-axis and *z*-axis, respectively, of the current axes.

Remarks Re-issuing an `xlabel`, `ylabel`, or `zlabel` command causes the new label to replace the old label.

For three-dimensional graphics, MATLAB puts the label in the front or side, so that it is never hidden by the plot.

See Also `text`, `title`

xlim, ylim, zlim

Purpose Set or query axis limits

Syntax Note that the syntax for each of these three functions is the same; only the `xlim` function is used for simplicity. Each operates on the respective x-, y-, or z-axis.

```
xlim
xlim([xmin xmax])
xlim('mode')
xlim('auto')
xlim('manual')
xlim(axes_handle,...)
```

Description `xlim` with no arguments returns the respective limits of the current axes.

`xlim([xmin xmax])` sets the axis limits in the current axes to the specified values.

`xlim('mode')` returns the current value of the axis limits mode, which can be either `auto` (the default) or `manual`.

`xlim('auto')` sets the axis limit mode to `auto`.

`xlim('manual')` sets the respective axis limit mode to `manual`.

`xlim(axes_handle,...)` performs the set or query on the axes identified by the first argument, `axes_handle`. When you do not specify an axes handle, these functions operate on the current axes.

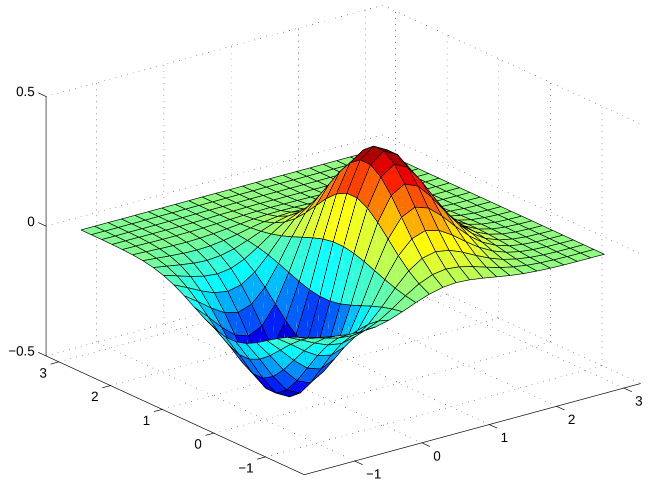
Remarks `xlim`, `ylim`, and `zlim` set or query values of the axes object `XLim`, `YLim`, `ZLim`, and `XLimMode`, `YLimMode`, `ZLimMode` properties.

When the axis limit modes are `auto` (the default), MATLAB uses limits that span the range of the data being displayed and are round numbers. Setting a value for any of the limits also sets the corresponding mode to `manual`. Note that high-level plotting functions like `plot` and `surf` reset both the modes and the limits. If you set the limits on an existing graph and want to maintain these limits while adding more graphs, use the `hold` command.

Examples

This example illustrates how to set the *x*- and *y*-axis limits to match the actual range of the data, rather than the rounded values of [-2 3] for the *x*-axis and [-2 4] for the *y*-axis originally selected by MATLAB.

```
[x,y] = meshgrid([-1.75:.2:3.25]);
z = x.*exp(-x.^2-y.^2);
surf(x,y,z)
xlim([-1.75 3.25])
ylim([-1.75 3.25])
```



See Also

axis

The axes properties XLim, YLim, ZLim

The "Aspect Ratio" section in the online *Using MATLAB Graphics* manual.

zoom

Purpose

Zoom in and out on a 2-D plot

Syntax

```
zoom on
zoom off
zoom out
zoom reset
zoom
zoom xon
zoom yon
zoom(factor)
zoom(fig, option)
```

Description

`zoom on` turns on interactive zooming. When interactive zooming is enabled in a figure, pressing a mouse button while your cursor is within an axes zooms into the point or out from the point beneath the mouse. Zooming changes the axes limits.

- For a single-button mouse, zoom in by pressing the mouse button and zoom out by simultaneously pressing **Shift** and the mouse button.
- For a two- or three-button mouse, zoom in by pressing the left mouse button and zoom out by pressing the right mouse button.

Clicking and dragging over an axes when interactive zooming is enabled draws a rubber-band box. When the mouse button is released, the axes zoom in to the region enclosed by the rubber-band box.

Double-clicking over an axes returns the axes to its initial zoom setting.

`zoom off` turns interactive zooming off.

`zoom out` returns the plot to its initial zoom setting.

`zoom reset` remembers the current zoom setting as the initial zoom setting. Later calls to `zoom out`, or double-clicks when interactive `zoom` mode is enabled, will return to this zoom level.

`zoom` toggles the interactive zoom status.

`zoom xon` and `zoom yon` set `zoom on` for the *x*- and *y*-axis, respectively.

`zoom(factor)` zooms in or out by the specified zoom factor, without affecting the interactive zoom mode. Values greater than 1 zoom in by that amount, while numbers greater than 0 and less than 1 zoom out by `1/factor`.

`zoom(fig, option)` Any of the above options can be specified on a figure other than the current figure using this syntax.

Remarks

`zoom` changes the axes limits by a factor of two (in or out) each time you press the mouse button while the cursor is within an axes. You can also click and drag the mouse to define a zoom area, or double-click to return to the initial zoom level.

A

List of Commands

Function Names

Index

D

E

Q

R

S

T

U

V

W

X

Y

Z