Mapping Toolbox

For Use with MATLAB®

Systems Planning and Analysis, Inc.

Computation

Visualization

Programming

Reference Guide

Version 1

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

Mapping Toolbox Reference Guide

© COPYRIGHT 1998 by The MathWorks, Inc. All Rights Reserved.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

U.S. GOVERNMENT: If Licensee is acquiring the software on behalf of any unit or agency of the U. S. Government, the following shall apply:

(a) for units of the Department of Defense:

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013.

(b) for any other unit or agency:

NOTICE - Notwithstanding any other lease or license agreement that may pertain to, or accompany the delivery of, the computer software and accompanying documentation, the rights of the Government regarding its use, reproduction and disclosure are as set forth in Clause 52.227-19(c)(2) of the FAR.

Contractor/manufacturer is The MathWorks Inc., 24 Prime Park Way, Natick, MA 01760-1500.

MATLAB, Simulink, Handle Graphics, and Real-Time Workshop are registered trademarks and Stateflow and Target Language Compiler are trademarks of The MathWorks, Inc.

Other product or brand names are trademarks or registered trademarks of their respective holders.

Printing History: May 1997 First printing
October 1998 Second printing (online only)

Contents

1 Mapping Reference

Function Tables 1-2
Map Definition And Display 1-2
. 1-5
Map Appearance And Interaction 1-6
Geographic Computation 1-9
Data Operations 1-11
Matrix Map Data Operations 1-13
. 1-14
Reference Data 1-15
Geoid Manipulation 1-16
Data Conversion 1-18
. 1-21
Mapping Toolbox Demos 1-21

2 Projections Reference

How to Use the Projections Reference Pages 2-2

3 GUI Reference

How to Use the GUI Reference Pages 3-2

Graphical User Interface Tables 3-3
Map Definition Tools 3-3
Mapping Tools 3-3
Object Projection Tools 3-4
Display Manipulation Tools 3-4
Thematic Map Tools 3-5
Object Property Tools 3-5
Track Tools 3-6
Map Data Construction Tools 3-6

4 External Data Reference

How to Use the External Data Reference Pages **4-2**

External Data Function Tables **4-3**
External Data Interface **4-3**

1

Mapping Reference

Function Tables

Map Definition And Display

Table 1-1 Projection Control

Function	Description
axesm	Map axes definition and property setting.
defaultm	Initialize default map projection structure.
gcm	Get current map data structure.
ismap	Test if axes have a map definition.
ismapped	Test if object is projected on map axes.
maps	List available map projections and verify id strings.
project	Project displayed graphics object.
usamap	Create a map of the United States of America.
worldmap	Maps a country or region using the worldlo atlas data.

Table 1-2 Projection Calculations

Function	Description
`distortcalc`	3-D contour plot of matrix map data.
`mfwdtran`	Project geographic data without displaying.
`minvtran`	Transform projected data back to geographic coordinates without displaying.
`utmgeoid`	Recommended UTM geoids for zone.
`utmzone`	Universal Transverse Mercator zone.
`vfwdtran`	Transform vector azimuths to a projection space angle.
`vinvtran`	Transform azimuths from a projection space angle.

Table 1-3 Projecting Line Data

Function	Description
`contor3m`	3-D contour plot of matrix map data.
`contorm`	2-D contour plot of matrix map data.
`linem`	Project line objects onto current map axes.
`plot3m`	Project 3-D lines onto current map axes.
`plotm`	Project 2-D lines onto current map axes.

Table 1-4 Projecting Patch Data

Function	Description
`contourfm`	Filled contour map.
`fill3m`	Filled 3-D map polygons in 3-D space.
`fillm`	Filled 2-D map polygons.
`patchesm`	Project patch faces as objects onto current map axes.
`patchm`	Project patch objects onto current map axes.

Table 1-5 Projecting 3-D Map Surface and Mesh Data

Function	Description
`imagem`	Regular matrix map displayed as image.
`meshm`	Regular matrix map warped to projected graticule mesh.
`pcolorm`	Projected matrix map in z = 0 plane.
`surfacem`	Matrix map warped to projected graticule mesh.
`surfm`	Matrix map projected on map axes.

Table 1-6 Projecting Light Objects and Lighted Surfaces

Function	Description
`lightm`	Project light source onto current map.
`meshlsrm`	3-D lighted shaded relief of regular matrix map.
`surflm`	3-D shaded surface with lighting projected on map axes.
`surflsrm`	3-D lighted shaded relief of general matrix map.
`shaderel`	Construct cdata and colormap for colored shaded relief.

Table 1-7 Thematic Maps

Function	Description
`comet3m`	3-D comet plot projected on map axes.
`cometm`	2-D comet plot projected on map axes.
`quiver3m`	3-D quiver plot projected on map axes.
`quiverm`	2-D quiver plot projected on map axes.
`stem3m`	Project stem map onto current map axes.
`scatterm`	Construct thematic map with proportional symbols.

Map Appearance And Interaction

Table 1-8 Annotation

Function	Description
clabelm	Map contour labeling.
clegendm	Add legend labels to map contour plot.
demcmap	Generation of color maps.
framem	Toggle display of map frame.
gridm	Grid lines projected on map axes.
mdistort	Displays contours of constant distortion on a map.
mlabel	Meridian labels projected on map axes.
plabel	Parallel labels projected on map axes.
polcmap	Colormap for political maps.
scaleruler	Add or modify graphic scale.
textm	Text annotation projected onto map axes.

Table 1-9 Interaction

Function	Description
gcpmap	Get current mouse point from map.
gtextm	Mouse placement of text on map.
inputm	Latitude and longitude selected by mouse.

Table 1-10 Appearance Control

Function	Description
axesscale	Resize axes for equivalent scale.
camtargm	Camera target from geographic coordinates.
camposm	Camera position from geographic coordinates.
camupm	Camera UpVector from geographic coordinates.
daspectm	Figure DataAspectRatio property for a map.
paperscale	Figure paper size for a given map scale.
previewmap	View map at printed size.
rotatetext	Rotate text to the projected graticule.
tightmap	Remove whitespace around a map.
trimcart	Trim graphic objects to the map frame.

Table 1-11 Object Display

Function	Description
clma	Clear current map axes.
clmo	Clear specified map graphics object.
hidem	Hide specified map graphic objects.
mobjects	Manipulate object sets displayed on map axes.
showaxes	Toggle display of Cartesian MATLAB axes.
showm	Show specified graphic objects.

Table 1-12 Property Manipulation

Function	Description
`cart2grn`	Greenwich coordinates of displayed map object.
`getm`	Get map object properties.
`handlem`	Handle of displayed map objects.
`makemapped`	Make an object a mapped object.
`namem`	Names of displayed graphics objects.
`restack`	Restacks objects within the axes.
`setm`	Set and modify map properties.
`tagm`	Assign names to graphics objects using tag property.
`zdatam`	Adjust z plane of displayed map objects.

Table 1-13 Geographic Data Structures

Function	Description
`displaym`	Project entries of geographic data structure.
`extractm`	Extract vector data from geographic data structure.
`mlayers`	Manipulate map layers defined with structure data.

Geographic Computation

Table 1-14 Points

Function	Description
antipode	Compute point on opposite side of globe.
azimuth	Azimuth between two points.
distance	Distance between two points.
reckon	Reckoning with starting point, azimuth, and range.
departure	Departure of longitudes at specific latitudes.

Table 1-15 Lines

Function	Description
ellipse1	Ellipse defined by its center, semimajor axes, eccentricity, and azimuth.
scircle1	Small circle defined by starting point and arc limits.
scircle2	Small circle defined by its center and perimeter.
scircleg	Display of small circle defined via mouse input.
tissot	Tissot indicatrices projected onto map axes.
track	Connect navigational waypoints with track segments.
track1	Great circle or rhumb line from point, azimuth, and range.
track2	Great circle or rhumb line defined by two points.
trackg	Display of great circle or rhumb line from mouse input.

Table 1-16 Surface Areas

Function	Description
areaint	Compute spherical surface area of polygon.
areamat	Compute geographic area of matrix elements.
areaquad	Compute spherical area of lat-long quadrangle.

Table 1-17 Geostatistics

Function	Description
combntns	Compute all combinations of given set of values.
eqa2grn	Equal area coordinates to Greenwich coordinates.
filterm	Geographic filter for data sets.
grn2eqa	Greenwich coordinates to equal area coordinates.
hista	Spatial equal area histogram.
histr	Spatial equirectangular histogram.
meanm	Mean of geographic points data.
stdm	Standard deviation of geographic data.
stdist	Standard distance of geographic data.

Table 1-18 Navigation

Function	Description
dreckon	Compute dead reckoning positions for track.
gcwaypts	Compute equally spaced waypoints along great circle.
legs	Compute courses and distances between track waypoints.
navfix	Perform mercator-based navigational fixing.
timezone	Set time zone description for longitudes.

Data Operations

Table 1-19 Interpolation

Function	Description
interpm	Linearly interpolate between latitude and longitude data.
intrplat	Compute interpolated latitude for given longitude.
intrplon	Compute interpolated longitude for given latitude.

Table 1-20 Intersections

Function	Description
crossfix	Compute cross fix positions for bearings and ranges.
gcxgc	Compute intersection points of two great circles.
gcxsc	Compute intersection points of great and small circles.
rhxrh	Find intersection point between two rhumb lines.
scxsc	Calculate intersection points for pairs of small circles.

Table 1-21 Trimming and Reducing Map Data

Function	Description
maptriml	Trim line vector map to specified region.
maptrimp	Trim patch map data to specified region.
maptrims	Trim surface matrix map data to specified region.
reducem	Reduce number of points in vector data.

Table 1-22 Data Precision

Function	Description
epsm	Accuracy of map computations.
roundn	Round data to specified power of 10.

Matrix Map Data Operations

Table 1-23 Manipulation

Function	Description
changem	Substitute codes throughout matrix map.
findm	Lat/lon values of non-zero map entries.
ltln2val	Map code value for latitude and longitude point.
maskm	Mask out specified matrix entries.
resizem	Resize matrix map.

Table 1-24 Regular Matrix Maps

Function	Description
country2mtx	Matrix map for a country in the worldlo database.
encodem	Fill in indexed maps with specified seeds.
getseeds	Get seed locations for indexing regular matrix map.
imbedm	Encode vector data points into regular matrix map.
limitm	Latitude and longitude limits for regular matrix map.
meshgrat	Construct map graticule for surface object display.
neworig	Transform matrix map to new coordinate system origin.
setltln	Matrix coordinates to Greenwich frame.
setpostn	Greenwich coordinates to matrix map coordinates.
sizem	Row and column dimensions for regular matrix map.
vec2mtx	Regular matrix map from vector data.

Table 1-25 Special Regular Matrix Maps

Function	Description
nanm	Full matrix map of NaNs.
spzerom	Sparse regular matrix map of zeros.
onem	Full matrix map of ones.
zerom	Full matrix map of zeros.

Reference Data

Table 1-26 Data Almanac and Atlas Data

Function	Description
almanac	Planetary data for solar system, including Sun and Moon.
coast	World coast lines.
loadcape	Load Cape Cod digital elevation regular matrix map.
loadmoonalb	Load moon albedo image into a regular matrix map.
oceanlo	Ocean mask for worldlo.
usahi	United States vector data.
usalo	United States vector data.
usamtx	United States matrix data.
worldlo	World vector data for mlayers.
worldmtx	World matrix map.

Table 1-27 Interface

Function	Description
grepfields	Identify matching fields in fixed record length files.
nanclip	Clip vector data with NaNs at specified pen-down locations.
rc2yx	Row and column indices to x- and y-coordinates.
readfields	Reads fields or records from a fixed file format.
readmtx	Reads a matrix stored in a file.
spcread	Read ascii file of space delimited data columns.
yx2rc	x- and y-coordinates to row and column indices.

Geoid Manipulation

Table 1-28 Geoid Parameters

Function	Description
axes2ecc	Eccentricity given semimajor and semiminor axes.
ecc2flat	Flattening given eccentricity.
ecc2n	Parameter n given eccentricity.
flat2ecc	Eccentricity given flattening.
majaxis	Semimajor axis given semiminor axis and eccentricity.
minaxis	Semiminor axis given semimajor axis and eccentricity.
n2ecc	Eccentricity given parameter n.
newpole	Origin vector to place specific point at north pole.
org2pol	Location of north pole in transformed coordinates.
putpole	Origin vector given location of north pole.
rcurve	Various radii of curvature for ellipsoid.
rsphere	Radii for auxiliary spheres.

Table 1-29 Auxiliary Latitude Conversions

Function	Description
`aut2geod`	Authalic latitude to geodetic latitude.
`cen2geod`	Geocentric latitude to geodetic latitude.
`cnf2geod`	Conformal latitude to geodetic latitude.
`geod2aut`	Geodetic latitude to authalic latitude.
`geod2cen`	Geodetic latitude to geocentric latitude.
`geod2cnf`	Geodetic latitude to conformal latitude.
`geod2iso`	Geodetic latitude to isometric latitude.
`geod2par`	Geodetic latitude to parametric latitude.
`geod2rec`	Geodetic latitude to rectifying latitude.
`iso2geod`	Isometric latitude to geodetic latitude.
`par2geod`	Parametric latitude to geodetic latitude.
`rec2geod`	Rectifying latitude to geodetic latitude.

Data Conversion

Table 1-30 Angles

Function	Description
angledim	Convert angles using recognized units strings.
deg2dm	Convert angles from degrees to deg:min.
deg2dms	Convert angles from degrees to deg:min:sec.
deg2rad	Convert angles from degrees to radians.
dms2deg	Convert angles from deg:min:sec to degrees.
dms2dm	Convert angles from deg:min:sec to deg:min.
dms2mat	Convert dms vector format to [d m s] matrix.
dms2rad	Convert angles from deg:min:sec to radians.
eastof	Wraps longitudes to values east of a meridian.
mat2dms	Convert [d m s] matrix to dms vector format.
npi2pi	Truncate angles into [-180 180] degrees range.
rad2deg	Convert angles from radians to degrees.
rad2dm	Convert angles from radians to deg:min.
rad2dms	Convert angles from radians to deg:min:sec.
smoothlong	Removes discontinuities in longitude data.
westof	Wraps longitudes to values west of a meridian.
zero2pi	Truncate angles into [0 360] range.

Table 1-31 Time

Function	Description
hms2hm	Convert times from hrs:min:sec to hrs:min.
hms2hr	Convert times from hrs:min:sec to hours.
hms2mat	Convert hms vector format to [h m s] matrix.
hms2sec	Convert times from hrs:min:sec to seconds.
hr2hm	Convert times from hours to hrs:min.
hr2hms	Convert times from hours to hrs:min:sec.
hr2sec	Convert times from hours to seconds.
mat2hms	Convert [h m s] matrix to hms vector format.
sec2hm	Convert times from seconds to hrs:min.
sec2hms	Convert times from seconds to hrs:min:sec.
sec2hr	Convert times from seconds to hours.
timedim	Convert times using recognized units strings.

Table 1-32 Distance

Function	Description
deg2km	Convert distance from degrees to kilometers.
deg2nm	Convert distance from degrees to nautical miles.
deg2sm	Convert distance from degrees to statute miles.
distdim	Convert distances using recognized units strings.
km2deg	Convert distance from kilometers to degrees.
km2nm	Convert distance from kilometers to nautical miles.
km2rad	Convert distance from kilometers to radians.
km2sm	Convert distance from kilometers to statute miles.
nm2deg	Convert distance from nautical miles to degrees.
nm2km	Convert distance from nautical miles to kilometers.
nm2rad	Convert distance from nautical miles to radians.
nm2sm	Convert distance from nautical miles to statute miles.
rad2km	Convert distance from radians to kilometers.
rad2nm	Convert distance from radians to nautical miles.
rad2sm	Convert distance from radians to statute miles.
sm2deg	Convert distance from statute miles to degrees.
sm2km	Convert distance from statute miles to kilometers.
sm2nm	Convert distance from statute miles to nautical miles.
sm2rad	Convert distance from statute miles to radians.

Table 1-33 String Manipulation

Function	Description
angl2str	Angle conversion to string.
dist2str	Distance conversion to string.
time2str	Time conversion to string.
unitstr	Test for valid unit strings or abbreviations.

Mapping Toolbox Demos

Function	Description
orbits	GUI demonstrating orbit calculations and map interaction.
qrydemo	Demonstration of linked map queries.
viewmaps	GUI demonstrating map projections.
vote92	GUI-based analysis of 1992 presidential election results.
vote96	GUI-based analysis of 1996 presidential election results.
wrlddemo	World map display and associated query.

almanac

Purpose

Display planetary data for the nine planets, the Sun, and the Moon

Syntax

```
almanac
almanac(body)
data = almanac(body,parameter)
data = almanac(body,parameter,units)
data = almanac(body,parameter,units,referencebody)
```

Description

`almanac` displays the names of the celestial objects available in the almanac.

`almanac(body)` lists the options, or parameters, available for each celestial body. Valid *body* strings are:

```
'earth'        'pluto'
'jupiter'      'saturn'
'mars'         'sun'
'mercury'      'uranus'
'moon'         'venus'
'neptune'
```

`data = almanac(body,parameter)` returns the value of the requested parameter for the celestial body specified by *body*.

Valid *parameter* strings are: `'radius'` for the planetary radius, `'geoid'` for the two-element geoid vector, `'surfarea'` for the surface area, and `'volume'` for the planetary volume.

For the Earth, *parameter* can also be any valid predefined ellipsoid string. In this case, the two-element geoid vector for that ellipsoid model is returned. Valid ellipsoid definition strings for the Earth are:

`'everest'`	for the 1830 Everest ellipsoid
`'bessel'`	for the 1841 Bessel ellipsoid
`'airy'`	for the 1849 Airy ellipsoid
`'clarke66'`	for the 1866 Clarke ellipsoid
`'clarke80'`	for the 1880 Clarke ellipsoid
`'international'`	for the 1924 International ellipsoid

`'krasovsky'`	for the 1940 Krasovsky ellipsoid
`'wgs60'`	for the 1960 World Geodetic System ellipsoid
`'iau65'`	for the 1965 International Astronomical Union ellipsoid
`'wgs66'`	for the 1966 World Geodetic System ellipsoid
`'iau68'`	for the 1968 International Astronomical Union ellipsoid
`'wgs72'`	for the 1972 World Geodetic System ellipsoid
`'grs80'`	for the 1980 Geodetic Reference System ellipsoid

For the Earth, the *parameter* string `'geoid'` is equivalent to `'grs80'`.

`data = almanac(`*body*`,`*parameter*`,`*units*`)` specifies the units to be used for the output measurement, where *units* is any valid distance units string. Note that these are linear units, but the result for surface area is in square units, and for volume is in cubic units. The default units are `'kilometers'`.

`data = almanac(`*parameter*`,`*units*`,`*referencebody*`)` specifies the source of the information. This sets the assumptions about the shape of the celestial body used in the calculation of volumes and surface areas. A *referencebody* string of `'actual'` returns a tabulated value rather than one dependent upon a geoid model assumption. Other possible *referencebody* strings are `'sphere'` for a spherical assumption, and `'geoid'`, for the default geoid model. The default reference body is `'sphere'`.

For the Earth, any of the above-listed pre-defined ellipsoid definition strings can also be entered as a reference body.

For Mercury, Pluto, Venus, the Sun and the Moon, the eccentricity of the geoid model is zero, i.e. the `'geoid'` reference body is actually a sphere.

Examples

The radius of the Earth (treated as a sphere) in kilometers:

```
almanac('earth','radius')
ans =
   6371
```

The default geoid model for the Earth ([semimajor axis eccentricity]):

```
almanac('earth','geoid')
ans =
   1.0e+03 *
       6.3781    0.0001
```

Note that the radius returned for any geoid model reference body is the semimajor axis:

```
almanac('earth','radius','kilometers','geoid')
Warning: Semimajor axis returned for radius parameter
ans =
   6.3781e+03
```

Compare the tabulated values of the Earth's surface area with a spherical assumption and with the 1966 World Geodetic System ellipsoid model:

```
almanac('earth','surfarea','statutemiles','actual')
ans =
     1.969614809100133e+08

almanac('earth','surfarea','statutemiles','sphere')
ans =
     1.969477626875824e+08

almanac('earth','surfarea','statutemiles','wgs66')
ans =
     1.969486900374475e+08
```

Note that these values are so close that long notation is required to discriminate them.

Some lunar measurements:

```
almanac('moon','radius')
ans =
        1738

almanac('moon','surfarea')
ans =
   3.7959e+07

almanac('moon','volume')
ans =
   2.1991e+10
```

Remarks

Care should be taken when using angular arc length units for distance measurements. All planets have a radius of 1 radian, for example, and an area unit of *square degrees* indicates unit squares, 1 degree of arc length on a side, not 1-degree-by-1-degree quadrangles.

See Also

`distance`	Distance between points
`distdim`	Convert distance units
`geoid vector`	Data structure for geoid model

angl2str

Purpose

Convert angular values to strings

Syntax

```
str = angl2str(angin)
str = angl2str(angin,format)
str = angl2str(angin,format,units)
str = angl2str(angin,format,digits)
str = angl2str(angin,format,units,digits)
```

Description

The purpose of this command is to make angular-valued variables into strings suitable for map display.

`str = angl2str(angin)` converts the input vector of angles, `angin`, to a string matrix.

`str = angl2str(angin,format)` uses the *format* string to specify the notation to be used with the string matrix. The default, `'none'`, results in simple numerical representation (no indicator for positive angles, minus signs for negative angles); `'pm'` (for *plus-minus*) adds a '+' for positive angles; `'ns'` (for *north-south*) appends an 'S' for negative angles and an 'N' for positive angles; `'ew'` (for *east-west*) appends a 'W' for negative angles and an 'E' for positive angles.

`str = angl2str(angin,format,units)` uses the input *units* to define the angle units of the `angin` input. *units* is any valid angle string (`'degrees'` are the default). The *units* input also determines which unit symbol to suffix to the output strings.

`str = angl2str(angin,format,units,digits)` determines how many digits to display. `digits` is the power of 10 representing the last place of significance in the resulting output. For example, if `digits` = 2, the *hundreds* slot will be the last significant figure. In general, the 10^{digits} slot will be the last significant figure, rounded appropriately depending upon the value in the $10^{digits-1}$ slot. `digits` is –2 by default.

Examples

Create a string matrix to represent a series of values in *dms* units, using the north-south format:

```
a = -3:1.5:3;
str = angl2str(deg2dms(a),'ns','dms')
str =
3^{\circ} 00' 00.00" S
1^{\circ} 30' 00.00" S
0^{\circ} 00' 00.00"
1^{\circ} 30' 00.00" N
3^{\circ} 00' 00.00" N
```

These LaTeX strings will be displayed (using either `text` or `textm`) as:

```
3° 00' 00.00" S
1° 30' 00.00" S
0° 00' 00.00"
1° 30' 00.00" N
3° 00' 00.00" N
```

See Also

`angledim`	Convert angle units
`dist2str`	Convert distance values to strings
`time2str`	Convert time values to strings

angledim

Purpose

Convert angles between different units

Syntax

```
anglout = angledim(anglin,from,to)
```

Description

`anglout = angledim(anglin,from,to)` returns the value of the input angle `anglin`, which is in units specified by the valid angle units string *from*, in the desired units given by the valid angle units string *to*. Valid angle units strings are:

```
'degrees' for decimal degrees
'radians' for radians
'dms' for degrees-minutes-seconds
'dm' for degrees-minutes
```

Examples

Convert from degrees to radians:

```
angledim(23.45134,'degrees','radians')
ans =
    0.4093
```

What is the difference between *dms* and *dm*? (best displayed in *bank format*)

```
format bank
angledim(23.45134,'degrees','dms')
ans =
       2327.05
angledim(23.45134,'degrees','dm')
ans =
       2327.00
```

The *dm* answer is the *dms* answer correctly rounded to whole minutes (i.e., rounded based on 60 seconds per minute, not 100).

See Also

`angl2str`	Convert angle values to strings
`azimuth`	Angular relationship between points

`deg2dms`
`dms2rad`
`deg2rad`
	Direct angle conversion functions

`distdim`	Convert distance units
`timedim`	Convert time units

antipode

Purpose

Determine the antipodes of a geographic point

Syntax

```
[newlat,newlong] = antipode(lat,long)
[newlat,newlong] = antipode(lat,long,units)
```

Description

`[newlat,newlong] = antipode(lat,long)` returns the geographic coordinates of the points exactly opposite on the globe from the input points given by `lat` and `long`.

`[newlat,newlong] = antipode(lat,long,units)` specifies the standard angle units string, where *units* is any valid angle units string. The default value is `'degrees'`.

Examples

Given a point (43°N, 15°E), find its antipode:

```
[newlat,newlong] = antipode(43,15)
newlat =
         -43
newlong =
         -165
```

or (43°S, 165°W). Perhaps the most obvious antipodal points are the North and South Poles. The command `antipode` demonstrates this:

```
[newlat,newlong] = antipode(90,0,'degrees')
newlat =
         -90
newlong =
         180
```

Note that in this case, longitudes are irrelevant because all meridians converge at the poles.

Purpose

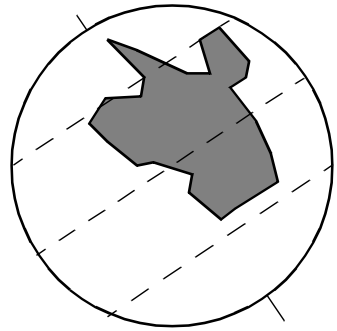
Calculate spherical surface area enclosed by a polygon

Syntax

```
area = areaint(lats,longs)
area = areaint(lats,longs,geoid)
area = areaint(lats,longs,units)
area = areaint(lats,longs,geoid,units)
```

Description

This command allows the measurement of areas enclosed by arbitrary polygons. This is a numerical estimate, using a line integral based on Green's Theorem. As such, it is limited by the accuracy and resolution of the input data.



Arbitrarily shaped polygons can be measured

`area = areaint(lats,longs)` returns the surface area enclosed by the polygon defined by the column vectors `lats` and `longs`. Multiple polygons can be delineated by `NaN`s. The output `area` is a fraction of the unit sphere's area of 4π, so the result ranges from 0 to 1.

`area = areaint(lats,longs,geoid)` allows the specification of the geoid model with the two-element geoid vector `geoid`. When a `geoid` is input, the resulting `area` is given in terms of the (squared) units of the `geoid`. For example, if the geoid `almanac('earth','geoid','kilometers')` is used, the resulting area will be in km^2. The default geoid is the unit sphere.

`area = areaint(lats,longs,geoid,units)` specifies the units of the inputs `lats` and `longs`, which are `'degrees'` by default.

Examples

Consider the area enclosed by a 30° lune, from pole to pole and bounded by the Prime Meridian and 30°E. You can use the command areaquad to get an exact solution:

```
area = areaquad(90,0,-90,30)
area =
    0.0833
```

This is 1/12 the spherical area. The more points used to define this polygon, the more integration steps areaint will take, improving the estimate. This first attempt takes a point every 30° of latitude:

```
lats = [-90:30:90,60:-30:-60]';
longs = [zeros(1,7),30*ones(1,5)]';
area = areaint(lats,longs)
area =
    0.0792
```

Now, a little finer, perhaps one point every 1° of latitude:

```
lats = [-90:1:90,89:-1:-89]';
longs = [zeros(1,181),30*ones(1,179)]';
area = areaint(lats,longs)
area =
    0.0833
```

Limitations

As noted above, this is a line integral estimation, only as good as the accuracy and the density of the polygon vertex data. However, given sufficient data, the areaint command is the best method for determining the areas of complex polygons, such as continents, cloud cover, and other natural or derived features. The calculations in this function employ a spherical Earth assumption. For nonspherical geoids, the latitude data is converted to the auxiliary authalic sphere.

See Also

almanac	Planetary data
areamat areaquad	Other area calculations

Purpose

Determine geographic area of matrix element

Syntax

```
[area,areavec] = areamat(map,maplegend)
[area,areavec] = areamat(map,maplegend,geoid)
[area,areavec] = areamat(map,maplegend,units)
[area,areavec] = areamat(map,maplegend,geoid,units)
```

Description

Given a regular matrix map that is a logical 0-1 matrix, the `areamat` function will return the area corresponding to the true, or 1, elements. The input matrix map can be a logical statement, such as `(topo>0)`, which is 1 everywhere that `topo` is greater than 0 meters, and zero everywhere else. This is an illustration of that matrix:

`[area,areavec] = areamat(map,maplegend)` returns the surface area corresponding to the entries equal to 1 in the regular matrix map, `map`, with a three-element map legend vector `maplegend`. The output `area` is a fraction of the unit sphere's area of 4π, so the result ranges from 0 to 1. Since the area of a given cell is the same within any row of the matrix, the second output, `areavec`, can be useful. It is a vector, having the length of a column of `map`, that provides the cell areas for each row, regardless of whether any element of that row is a 1.

`[area,areavec] = areamat(map,maplegend,geoid)` allows the specification of the geoid model with the two-element geoid vector `geoid`. When a `geoid` is input, the resulting `area` is given in terms of the (squared) units of the `geoid`. For example, if the geoid `almanac('earth','geoid','kilometers')` is used, the resulting area will be in km^2. The default geoid is the unit sphere.

`area = areaint(lats,longs,geoid,units)` specifies the units of the inputs of the map legend, which are `'degrees'` by default.

Examples

```
load topo
area = areamat((topo>127),topolegend)
area =
    0.2411
```

Approximately 24% of the Earth has an altitude greater than 127 meters. What is the surface area of this portion of the Earth in square kilometers if a spherical geoid is assumed? (use the `almanac` function with the sphere as its reference body).

```
earthgeoid = almanac('earth','geoid','km','sphere');
area = areamat((topo>127),topolegend,earthgeoid)
area =
   1.2299e+08
```

To illustrate the `areavec` output, consider a smaller map:

```
map = ones(9,18);
maplegend = [.05 90 0] % each cell 20x20 degrees

[area,areavec] = areamat(map,maplegend)
area =
    1.0000
areavec =
    0.0017
    0.0048
    0.0074
    0.0091
    0.0096
    0.0091
    0.0074
    0.0048
    0.0017
```

Each entry of `areavec` represents the portion of the unit sphere's total area a cell in that row of map would contribute. Since the column extends from pole to pole in this case, it is symmetric.

Remarks

This calculation is based on the areaquad command and is therefore limited only by the granularity of the cellular data resolution and the spherical Earth assumption. For nonspherical geoids, the latitude data is converted to the auxiliary authalic sphere.

See Also

almanac	Planetary data
areaint areaquad	Other area calculations

areaquad

Purpose

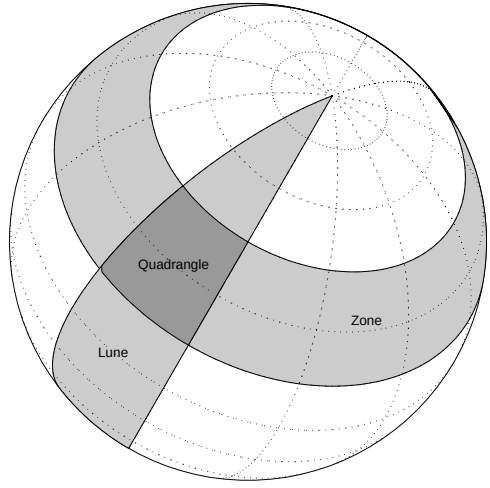
Compute area of a latitude-longitude quadrangle

Syntax

```
area = areaquad(lat1,lon1,lat2,lon2)
area = areaquad(lat1,lon1,lat2,lon2,geoid)
area = areaquad(lat1,lon1,lat2,lon2,units)
area = areaquad(lat1,lon1,lat2,lon2,geoid,units)
```

Description

A latitude-longitude quadrangle is a region bounded by two meridians and two parallels. In spherical geometry, it is the intersection of a *lune* (a section bounded by two meridians) and a *zone* (a section bounded by two parallels).



`area = areaquad(lat1,lon1,lat2,lon2)` returns the surface area bounded by the parallels `lat1` and `lat2` and the meridians `lon1` and `lon2`. The output `area` is a fraction of the unit sphere's area of 4π, so the result ranges from 0 to 1.

`area = areaquad(lat1,lon1,lat2,lon2,geoid)` allows the specification of the geoid model with the two-element geoid vector `geoid`. When a `geoid` is input, the resulting `area` is given in terms of the (squared) units of the `geoid`. For example, if the geoid `almanac('earth','geoid','kilometers')` is used, the resulting area will be in km^2. The default geoid is the unit sphere.

`area = areaquad(lat1,lon1,lat2,lon2,geoid,`*`units`*`)` specifies the units of the inputs, which are `'degrees'` by default.

Examples

What fraction of the Earth's surface lies between 30°N and 45°N, and also between 25°W and 60°E?

```
area = areaquad(30,-25,45,60)
area =
    0.0245
```

About 2.5%. What is the surface area of the Earth in square kilometers if a spherical geoid is assumed (use the `almanac` function with the sphere as its reference body)?

```
earthgeoid = almanac('earth','geoid','km','sphere');
area = areaquad(-90,-180,90,180,earthgeoid)
area =
   5.1006e+08
```

For comparison:

```
almanac('earth','surfarea','km')
ans =
   5.1006e+08
```

Remarks

This calculation is exact, being based on simple spherical geometry. For nonspherical geoids, the data is converted to the auxiliary authalic sphere.

See Also

`almanac`	Planetary data
`areaint` `areamat`	Other area calculations

aut2geod

Purpose

Convert from authalic to geodetic latitudes

Syntax

```
lat = aut2geod(lat0)
lat = aut2geod(lat0,geoid)
lat = aut2geod(lat,units)
lat = aut2geod(lat,geoid,units)
```

Background

Longitudes have the same meaning whether the Earth is treated as a sphere or as an ellipsoid. However, the definition of latitude in the case of an ellipsoid is more complicated. Geodetic, or geographic, latitudes on an ellipsoid can be transformed from latitudes on an auxiliary sphere with certain properties.

Geodetic latitude: (also called *geographic latitude*) the angle a normal line passing through a surface point makes with the plane of the Equator.

Authalic latitude: latitudes on an auxiliary sphere that is equal in area to the ellipsoid.

Description

`lat = aut2geod(lat0)` returns the authalic latitudes provided in `lat0` transformed to geodetic latitudes in `lat`.

`lat = aut2geod(lat0,geoid)` defines the elliptical model of the Earth, given by the geoid vector `geoid`, to which `lat0` is transformed. The default geoid is the same as the default of `almanac('earth','geoid')`, the 1980 Geodetic Reference System ellipsoid.

`lat = aut2geod(lat,geoid,units)` defines the angle units of the inputs and outputs, where *units* is any valid angle units string. The default is `'degrees'`.

Examples

```
lat = aut2geod([0 45 90])
lat =
         0   45.1283   90.0000
```

See Also

`almanac`	Planetary data
`geod2aut` `iso2geod`	Other auxiliary latitude functions

Purpose

Calculate eccentricity from semimajor and semiminor axes

Syntax

```
eccentricity = axes2ecc(semimajor,semiminor)
eccentricity = axes2ecc(axes)
```

Description

Eccentricity, the second element of the standard geoid vector in the Mapping Toolbox, can be determined given both the semimajor and semiminor axes.

`eccentricity = axes2ecc(semimajor,semiminor)` returns the eccentricity associated with the input axes.

`eccentricity = axes2ecc(axes)` allows the axes inputs to be packed into a single, two-column input of the form [semimajor, semiminor].

Examples

Using the axes for the default GRS 80 Earth model:

```
ecc = axes2ecc(6378.1370,6356.7523)
ecc =
   0.08181921804834
```

This is the eccentricity returned by `almanac('earth','geoid')`.

See Also

`almanac`	Planetary data
`ecc2n majaxis minaxis`	Related conversion functions

axesm

Purpose

Define map axes and set map properties

Syntax

```
axesm
axesm(handle)
axesm(PropertyName,PropertyValue,...)
axesm(ProjectionFile,PropertyName,PropertyValue,...)
```

Description

The axesm function creates a map axes object complete with a map data structure. Maps must be displayed in map axes. All standard MATLAB axes properties of map axes are controlled by the axes command, along with set and get. Map axes properties are defined on creation with axesm and may be queried and changed after creation of a map axes using getm and setm.

axesm, with no input arguments, initiates the map axes graphical user interface, which can be used to set map axes properties. This is detailed on the axesm, axesmui page in Chapter 3, "GUI Reference".

axesm(handle) makes the map axes with the given handle the current map axes.

axesm(*PropertyName*,PropertyValue,...) creates map axes with the specified property values. The MapProjection property must be the first input pair.

axesm(*ProjectionFile*,*PropertyName*,PropertyValue,...) allows omission of the MapProjection property name. The first input must be the identifying string of an available projection.

Axes Definition

Map axes are standard MATLAB axes with different default settings for some properties and with a map data structure in the UserData slot. The main differences are:

- Axes properties XGrid, YGrid, XTick, YTick are set to 'off'.
- The properties XColor, YColor, ZColor are set to the background color.
- The hold mode is on.

The map structure is assigned to the UserData property. Do not overwrite this entry if map axes are desired. The map structure contains the map axes properties which, in addition to the special standard axes settings described here, constitute a map axes. The map axes properties are described later.

Examples

Create map axes for a Mercator projection, with selected latitude limits:

```
axesm('MapProjection','mercator','FLatLimit',[-70 80])
```

In the above example, all properties not explicitly addressed in the call are set to either fixed or calculated defaults. The M-file `mercator.m` is a projection file, so the same result could have been achieved with the command:

```
axesm('mercator','FLatLimit',[-70 80])
```

A projection file will include default data for all properties. Any following property name/property value pairs are treated as overrides.

In either of the above examples, data displayed in the given map axes will be in a Mercator projection. Any data falling outside the prescribed frame limits would not be displayed.

Note: the names of projection files are case sensitive. The projection files included in the Mapping Toolbox use only lower-case letters and Arabic numerals.

Object Properties

Properties that control the map

`MapProjection` *`projection_name`* `{no default}`

Map projection – This property sets the projection, and hence all transformation calculations, for the map axes object. It is required in the creation of map axes. The projection name is a string corresponding to an M-file appropriate to the projection. It must be a member of the recognized projection set, which can be listed by typing `getm('MapProjection')` or `maps`. For more information on projections, see Chapter 4, "Map Projections," in the *Mapping Toolbox User's Guide.* Some projections set their own defaults for other properties, such as parallels and trim limits.

`Zone` `ZoneSpec | {[] or 31N}`

Zone for certain projections – This property specifies the zone for certain projections. A zone is a region on the globe that has a special set of projection parameters. In the Universal Transverse Mercator Projection, the world is divided into quadrangles which are generally 6 degrees wide and 8 degrees tall. The number in the zone designation refers to the longitude range, while the

letter refers to the latitude range. Most projections use the same parameters for the entire globe, and do not require a zone.

AngleUnits {degrees} | radians | dms

Angular unit of measure – This property controls the units of measure used for angles (including latitudes and longitudes) in the map axes. All input data are assumed to be in the given units; 'degrees' is the default. For a more detailed description of the angle unit options, see the *Mapping Toolbox User's Guide*.

Aspect {normal} | transverse

Display aspect – This property controls the orientation of the base projection of the map. When the aspect is 'normal' (the default), *north* in the base projection is up. In a transverse aspect, north is to the right. A cylindrical projection of the whole world would look like a *landscape* display under a 'normal' aspect, and like a *portrait* under a 'transverse' aspect. Note that this property is not the same as projection aspect, which is controlled by the Origin property vector discussed later.

FalseEasting scalar {0}

Coordinate shift for projections calculations – This property modifies the position of the map within the axes. The projected coordinates are shifted in the x-direction by amount of FalseEasting. The FalseEasting is in the same units as the projected coordinates, i.e. the units of first element of the Geoid map axes property. False eastings and northings are sometimes used to ensure non-negative values of the projected coordinates. For example, the Universal Transverse Mercator uses a false easting of 500,000 meters.

FalseNorthing scalar {0}

Coordinate shift for projections calculations – This property modifies the position of the map within the axes. The projected coordinates are shifted in the y-direction by amount of FalseNorthing. The FalseNorthing is in the same units as the projected coordinates, i.e. the units of first element of the Geoid map axes property. False eastings and northings are sometimes used to ensure non-negative values of the projected coordinates. For example, the Universal Transverse Mercator uses a false northing of 0 in the northern hemisphere, and 10,000,000 meters in the southern.

FixedOrient scalar {[]} (read-only)

Projection-based orientation – This read-only property fixes the orientation of certain projections (such as the Cassini and Wetch). When empty, which is true for most projections, the user can alter the orientation of the projection using the third element of the `Origin` property. When fixed, the fixed orientation is always used.

Geoid [semimajor_axis eccentricity]

Planet geoid definition – This property sets the geoid for calculating the projections of any displayed map objects. In the Mapping Toolbox, the geoid is approximated by a spheroid. The default geoid is a sphere with a radius of 1. This is represented as `[1 0]`. Any semimajor axis, in any distance units, can be entered; eccentricity lies between 0 and 1. For more information on the geoid definition, see the `geoid vector` reference page in this section.

MapLatLimit [south north] | [north south]

Latitude limits of the displayed map – This property sets the north and south latitude limits of the map data. This information is useful for two purposes. The default extents for the texture mapping commands `meshm`, `surfm`, `surfacem`, `surflm`, `pcolorm` and `imagem` are set for the map axes; if the latitude limits match the actual matrix map data limits, no graticule definitions are required when calling the above commands. Secondly, establishing map latitude limits sets the absolute limit on the extent of displayed meridians, regardless of the values of the meridian limits or the meridian exceptions. For non-azimuthal projections in the normal aspect, the map limits are truncated to the smaller of the map and frame limits. The default map latitude limits for most projections are at the poles, `[-90 90]`.

MapLonLimit [east west] | [west east]

Longitude limits of the displayed map – This property sets the east and west longitude limits of the map data. This information is useful for two purposes. The default extents for the texture mapping commands `meshm`, `surfm`, `surfacem`, `surflm`, `pcolorm` and `imagem` are set for the map axes; if the longitude limits match the actual matrix map data limits, no graticule definitions are required when calling the above commands. Secondly, establishing map longitude limits sets the absolute limit on the extent of displayed parallels, regardless of the values of the parallel limits or the parallel exceptions. For non-azimuthal projections in the normal aspect, the map limits

are truncated to the smaller of the map and frame limits. The default map longitude limits for most projections are at the International Date Line, `[-180 180]`.

`MapParallels` `[lat] | [lat1 lat2]`

Projection standard parallels – This property sets the standard parallels of projection. It can be an empty, one- or two-element vector, depending upon the projection. The elements are in the same units as the map axes `AngleUnits`. Many projections have specific, defining standard parallels. When a map axes object is based upon one of these projections, the parallels are set to the appropriate defaults. For conic projections, the default standard parallels are set to 15°N and 75°N, which will bias the projection towards the northern hemisphere.

For projections with one defined standard parallel, setting the parallels to an empty vector forces recalculation of the parallel to the middle of the map latitude limits. For projections requiring two standard parallels, setting the parallels to an empty vector forces recalculation of the parallels to one-sixth the distance from the latitude limits (e.g., if the map latitude limits correspond to the northern hemisphere `[0 90]`, the standard parallels for a conic projection will be set to `[15 75]`). For azimuthal projections, the `MapParallels` property will always contain an empty vector and cannot be altered.

See Chapter 4, "Map Projections," in the *Mapping Toolbox User's Guide* for more information on standard parallels.

`Parallels` `0, 1 or 2 (read-only, projection-dependent)`

Number of standard parallels – This read-only property contains the number of standard parallels associated with the projection. See Chapter 4, "Map Projections," in the *Mapping Toolbox User's Guide* for more information on standard parallels.

`Origin` `[latitude longitude orientation]`

Origin and orientation for projection calculations – This property sets the map origin for all projection calculations. The latitude, longitude, and orientation should be in the map axes `AngleUnits`. Latitude and longitude refer to the coordinates of the map origin; orientation refers to an angle of skewness or rotation about the axis running through the origin point and the center of the earth. The default origin is 0° latitude and a longitude centered between the

map longitude limits. If a scalar is entered, it is assumed to refer to the longitude; if a two-element vector is entered, the default orientation is 0°, a normal projection. If an empty origin vector is entered, the origin will be centered on the map longitude limits. For more information on the origin, see Chapter 4, "Map Projections," in the *Mapping Toolbox User's Guide*.

ScaleFactor `scalar {1}`

Scale factor for projection calculations – This property modifies the size of the map in projected coordinates. The geographic coordinates are transformed to cartesian coordinates by the map projection equations, and multiplied by the scale factor. Scale factors are sometimes used to minimize the scale distortion in a map projection. For example, the Universal Transverse Mercator uses a scale factor of 0.996 to shift the line of zero scale distortion to two lines on either side of the central meridian.

TrimLat `[south north] (read-only, projection-dependent)`

Latitude trimming for certain projections – This read-only property indicates the limits in latitude beyond which no plotting is attempted. This property is set by the projection and is usually used to avoid *blow-ups* to infinity. For example, the Mercator projection will trim all data outside the range `[-86 86]`. It is `[-90 90]` for most projections.

TrimLon `[west east] (read-only, projection-dependent)`

Longitude trimming for certain projections – This read-only property indicates the limits in longitude beyond which no plotting is attempted. This property is set by the projection and is usually used to avoid blow-ups to infinity. It is less commonly used than the latitude trimming and is `[-180 180]` for most projections.

Properties that control the frame

Frame `on | {off}`

Frame visibility – This property controls the visibility of the display frame box. When the frame is `'off'` (the default), the frame is not displayed. When the frame is `'on'`, an enclosing frame is visible. The frame is a patch which is plotted as the lowest layer of displayed map objects. Regardless of its display status, the frame always operates in terms of trimming map data.

FFill scalar plotting point density {100}

Frame plotting precision – This property sets the number of points to be used in plotting the frame for display. The default value is `100`, which for a rectangular frame would result in a plot with `100` points for each side, or a total of `400` points. The number of points required for a reasonable display varies with projection. Cylindrical projections such as the Miller require very few. Projections resulting in more complex frames, such as the Werner, look better with higher densities. The default value is generally sufficient.

FEdgeColor ColorSpec | {[0 0 0]}

Color of the displayed frame edge – This property specifies the color used for the displayed frame. You can specify a color using a vector of RGB values or one of MATLAB's predefined names. By default, the frame edge is displayed in black ([0 0 0]).

FFaceColor ColorSpec | {none}

Color of the displayed frame face – This property specifies the color used for the displayed frame face. You can specify a color using a vector of RGB values or one of MATLAB's predefined names. By default, the frame face is `'none'`, meaning no face color is filled in. Another useful color is `'cyan'` ([0 1 1]) which looks like water.

FLatLimit [south north] | [north south]

Latitude limits of the base projection frame – This property sets the north and south latitude limits of the map frame. Latitudes refer to the base projection, and are `[-90 90]` by default for most projections. The frame latitude limits determine where data will be trimmed in a north-south sense. Data lying outside the latitude limits will not be displayed. Also, these limits determine the latitude positions of the frame for its own display.

For non-azimuthal projections in the normal aspect, the frame limits are truncated to the smaller of the map and frame limits. Frame limits for non-azimuthal projections are specified in Cartesian coordinates with respect to the map origin specified in the `Origin` property. Frame latitude limits for azimuthal projections are specified by a –× and a radius in polar coordinates with respect to the map origin specified in the `Origin` property.

FLineWidth scalar {2}

Frame edge line width – This property sets the line width of the displayed frame edge. The value is a scalar representing points, which is 2 by default.

FLonLimit [east west] | [west east]

Longitude limits of the base projection frame – This property sets the east and west longitude limits of the map frame. Longitudes refer to the base projection, and are `[-180 180]` by default for most projections. The frame longitude limits determine where data will be trimmed in a east-west sense. Data lying outside the longitude limits will not be displayed. Also, these limits determine the longitude positions of the frame for its own display.

For non-azimuthal projections in the normal aspect, the frame limits are truncated to the smaller of the map and frame limits. Frame limits for non-azimuthal projections are specified in Cartesian coordinates with respect to the map origin specified in the `Origin` property. Frame longitude limits are ignored for azimuthal projections.

Properties that control the grid

Grid on | {off}

Grid visibility – This property controls the visibility of the display grid. When the grid is `'off'` (the default), the grid is not displayed. When the grid is `'on'`, meridians and parallels are visible. The grid is plotted as a set of line objects.

GAltitude scalar *z*-axis value {Inf}

Grid z-axis setting – This property sets the *z*-axis location for the grid when displayed. Its default value is infinity, which displays above all other map objects. However, this can be set to some other value for stacking objects above the grid, if desired.

GColor ColorSpec | {[0 0 0]}

Color of the displayed grid – This property specifies the color used for the displayed grid. You can specify a color using a vector of RGB values or one of MATLAB's predefined names. By default, the map grid is displayed in black (`[0 0 0]`).

GLineStyle LineStyle {:}

Grid line style – This property determines the style of line used when the grid is displayed. Any line style supported by the MATLAB `line` command can be specified. The default line style is a dotted line (i.e., `':'`).

GLineWidth scalar {0.5}

Grid line width – This property sets the line width of the displayed grid. The value is a scalar representing points, which is `0.5` by default.

MLineException vector of longitudes {[]}

Exceptions to grid meridian limits – This property allows specific meridians of the displayed grid to extend beyond the grid meridian limits to the poles. The value must be a vector of longitudes in the appropriate angle units. For longitudes so specified, grid lines will extend from pole to pole regardless of the existence of any grid meridian limits. This vector is empty by default.

MLineFill scalar plotting point density {100}

Grid meridian plotting precision – This property sets the number of points to be used in plotting the grid meridians. The default value is 100 points. The number of points required for a reasonable display varies with projection. Cylindrical projections such as the Miller require very few. Projections resulting in more complex shapes, such as the *Werner*, look better with higher densities. The default value is generally sufficient.

MLineLimit [north south] | [south north] {[]}

Grid meridian limits – This property establishes latitudes beyond which displayed grid meridians do not extend. By default, this property is empty, so the meridians extend to the poles. There are two exceptions to the meridian limits. No meridian will extend beyond the map latitude limits, and exceptions to the meridian limits for selected meridians are allowed (*see above*).

MLineLocation scalar interval or specific vector {30°}

Grid meridian interval or specific locations – This property establishes the interval between displayed grid meridians. When a scalar interval is entered in the map axes `MLineLocation`, meridians will be displayed, starting at 0° longitude and repeating every interval in both directions, which by default is 30°. Alternatively, a vector of longitudes can be entered, in which case a meridian will be displayed for each element of the vector.

PLineException vector of latitudes {[]}

Exceptions to grid parallel limits – This property allows specific parallels of the displayed grid to extend beyond the grid parallel limits to the International Date Line. The value must be a vector of latitudes in the appropriate angle units. For latitudes so specified, grid lines will extend from western to eastern map limit, regardless of the existence of any grid parallel limits. This vector is empty by default.

PLineFill scalar plotting point density {100}

Grid parallel plotting precision – This property sets the number of points to be used in plotting the grid parallels. The default value is `100`. The number of points required for a reasonable display varies with projection. Cylindrical projections such as the Miller require very few. Projections resulting in more complex shapes, such as the Bonne, look better with higher densities. The default value is generally sufficient.

PLineLimit [east west] | [west east] {[]}

Grid parallel limits – This property establishes longitudes beyond which displayed grid parallels do not extend. By default, this property is empty, so the parallels extend to the Date Line. There are two exceptions to the parallel limits. No parallel will extend beyond the map longitude limits, and exceptions to the parallel limits for selected parallels are allowed (*see above*).

PLineLocation scalar interval or specific vector {15°}

Grid parallel interval or specific locations – This property establishes the interval between displayed grid parallels. When an scalar interval is entered in the map axes `PLineLocation`, parallels will be displayed, starting at 0° latitude and repeating every interval in both directions, which by default is 15°. Alternatively, a vector of latitudes can be entered, in which case a parallel will be displayed for each element of the vector.

Properties that control grid labeling

FontAngle {normal} | italic | oblique

Select italic or normal font for all grid labels – This property selects the character slant for all displayed grid labels. `'normal'` specifies nonitalic font.`'italic'` and `'oblique'` specify italic font.

FontColor ColorSpec | {black}

Text color for all grid labels – This property sets the color of all displayed grid labels. ColorSpec is a three-element vector specifying an RGB triple, or a predefined MATLAB color string.

FontName courier | {helvetica} | symbol | times

Font family name for all grid labels – This property sets the font for all displayed grid labels. To display and print properly, FontName must be a font that your system supports.

FontSize scalar in units specified in FontUnits {9}

Font size – An integer specifying the font size to use for all displayed grid labels, in units specified by the FontUnits property. The default point size is 9.

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

Units used to interpret the FontSize property – When set to normalized, the Toolbox 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 bold | {normal}

Select bold or normal font – The character weight for all displayed grid labels.

LabelFormat {compass} | signed | none

Labelling format for grid – This property specifies the format of the grid labels. If 'compass' is employed (the default), meridian labels are suffixed with an 'E' for east and a 'W' for west, and parallel labels are suffixed with an 'N' for north and an 'S' for south. If 'signed' is used, meridian labels are prefixed with a '+' for east and a '–' for west and parallel labels are suffixed with a '+' for north and a '–' for south. If 'none' is selected, straight latitude and longitude numerical values are employed, so western meridian labels and southern parallel labels will have a '–', but no symbol will precede eastern and northern (positive) labels.

LabelRotation on | {off}

Label Rotation – This property determines whether the meridian and parallel labels are displayed without rotation (the default), or rotated to align to the graticule.

LabelUnits {degrees} | radians | dms | dm

Specify units for labels – Grid labels will be displayed in the desired angular units as specified in LabelUnits, regardless of the map axes AngleUnits. The default value, however, does match the AngleUnits property.

MeridianLabel on | {off}

Toggle display of meridian labels – This property specifies whether the meridian labels are visible or not.

MLabelLocation scalar interval or vector of longitudes

Specify meridians for labeling – Meridian labels need not coincide with the displayed meridian lines. Labels will be displayed at intervals if a scalar in the map axes MLabelLocation is entered, starting at the Prime Meridian and repeating at every interval in both directions. If a vector of longitudes is entered, labels will be displayed at those meridians. The default locations coincide with the displayed meridian lines, as specified in the MLineLocation property.

MLabelParallel {north} | south | equator | scalar latitude

Specify parallel for meridian label placement – This property specifies the latitude location of the displayed meridian labels. If a latitude is specified, all meridian labels will be displayed at that latitude. If 'north' is specified, the maximum of the MapLatLimit is used; if 'south' is specified, the minimum of the MapLatLimit is used. If 'equator' is specified, a latitude of 0° is used.

MLabelRound integer scalar {0}

Specify significant digits for meridian labels – This property specifies to which power of ten the displayed labels will be rounded. For example, if MLabelRound is –1, labels will be displayed down to the *tenths*. The default value of MLabelRound is 0, i.e., displayed labels have no decimal places, being rounded to the *ones* column (10^0).

ParallelLabel on | {off}

Toggle display of parallel labels – This property specifies whether the parallel labels are visible or not.

PLabelLocation scalar interval or vector of latitudes

Specify parallels for labeling – Parallel labels need not coincide with the displayed parallel lines. Labels will be displayed at intervals if a scalar in the

map axes `PLabelLocation` is entered, starting at the Equator and repeating at every interval in both directions. If a vector of latitudes is entered, labels will be displayed at those parallels. The default locations coincide with the displayed parallel lines, as specified in the `PLineLocation` property.

`PLabelMeridian` `east | {west} | prime | scalar longitude`

Specify meridian for parallel label placement – This property specifies the longitude location of the displayed parallel labels. If a longitude is specified, all parallel labels will be displayed at that longitude. If `'east'` is specified, the maximum of the `MapLonLimit` is used; if `'west'` is specified, the minimum of the `MapLonLimit` is used. If `'prime'` is specified, a longitude of 0° is used.

`PLabelRound` `integer scalar {0}`

Specify significant digits for parallel labels – This property specifies to which power of ten the displayed labels will be rounded. For example, if `PLabelRound` is –1, labels will be displayed down to the tenths. The default value of `PLabelRound` is 0, i.e., displayed labels have no decimal places, being rounded to the ones column (10^0).

See Also

`gcm`	Get current map structure
`getm`	Get map object properties
`setm`	Set map object properties

Purpose

Make scale the same between axes

Syntax

```
axesscale
axesscale(hbase)
axesscale(hbase,hother)
```

Description

`axesscale` resizes all axes in the current figure to have the same scale as the current axes (gca). In this context, scale means the relationship between axes x-and y-coordinates to figure and paper coordinates. When `axesscale` is used, a unit of length in x and y will print and display at the same size in all the affected axes. The `XLimMode` and `YLimMode` of the axes are set to `'manual'` to prevent autoscaling from changing the scale.

`axesscale(hbase)` uses the axes `hbase` as the reference axes, and rescales the other axes in the current figure.

`axesscale(hbase,hother)` uses the axes `hbase` as the base axes, and rescales only the axes in `hother`.

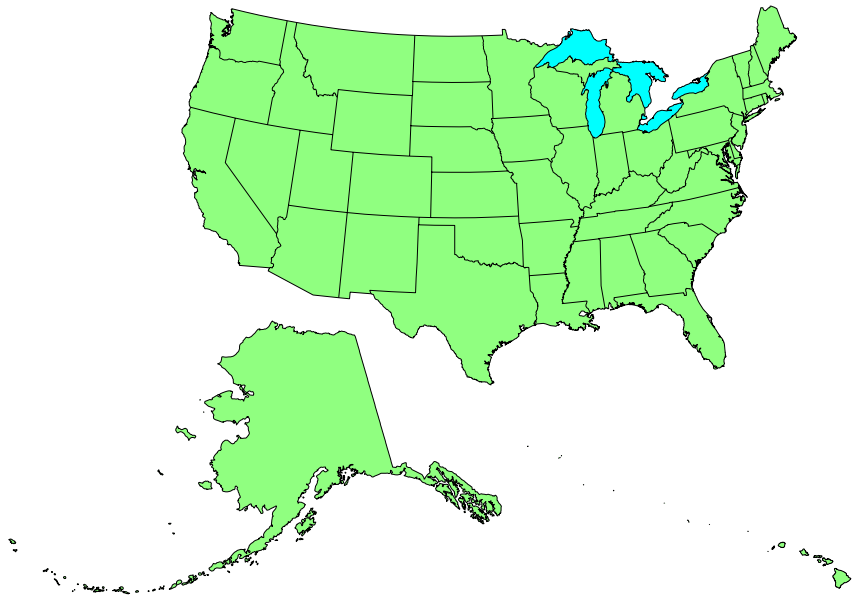
Examples

Display the conterminous United States, Alaska and Hawaii in separate axes.

```
hconus = usamap('conus'); tightmap

halaska= axes; usamap('alaskaonly'); tightmap;
framem off; mlabel off; plabel off

hhawaii = axes; usamap('hawaiionly'); tightmap;
framem off; mlabel off; plabel off

% Arrange the axes as desired
set(hconus,'Position',[0.1 0.25 0.85 0.6])
set(halaska,'Position',[0.019531 -0.020833 0.2 0.2])
set(hhawaii,'Position',[0.5 0 .2 .2])
```

Make the axes have the same scale.

```
axesscale(hconus) % resize alaska and hawaii axes
hidem([halaska hhawaii])
```



Limitations

The equivalence of scales holds only as long as no commands are issued that may change the scale of one of the axes. For example, changing the units of the geoid or the scale factor in one of the axes would change the scale.

Remarks

To ensure the same map scale between axes, use the same geoid and scale factors.

See Also

`paperscale`	Figure paper size for a given map scale

Purpose

Compute azimuth between two points on the globe

Syntax

```
az = azimuth(pt1,pt2)
az = azimuth(pt1,pt2,geoid)
az = azimuth(pt1,pt2,units)
az = azimuth(pt1,pt2,geoid,units)

az = azimuth(track,pt1,...)

az = azimuth(lat1,lon1,lat2,lon2)
az = azimuth(lat1,lon1,lat2,lon2,geoid)
az = azimuth(lat1,lon1,lat2,lon2,units)
az = azimuth(lat1,lon1,lat2,lon2,geoid,units)

az = azimuth(track,lat1,...)
```

Background

Azimuths are the bearings, or directions, between pairs of points. Azimuths are measured as angles, clockwise from true north. The North Pole has an azimuth of 0° from every other point on the globe.

Azimuth can be calculated in two manners. For great circles, the azimuth is the angle made between true north and the great circle passing through the two points *at the first point*. For rhumb lines, the azimuth is the *constant* angle made between true north and the entire rhumb line passing through the two points. For more information on this distinction, see the "Geographic Measurement" section of Chapter 1, "Mapping Fundamentals," in the *Mapping Toolbox User's Guide*.

Description

`az = azimuth(pt1,pt2)` calculates the great circle azimuths from `pt1` to `pt2`. These two-column matrices should be of the form `[latitude longitude]`.

`az = azimuth(lat1,lon1,lat2,lon2)` performs the same calculation for two pairs of latitude and longitude matrices.

`az = azimuth(pt1,pt2,geoid)` specifies the elliptical definition of the Earth to be used with the two-element `geoid` vector. The default geoid model is a unit sphere, which is sufficient for most applications.

`az = azimuth(pt1,pt2,units)` specifies the standard angle unit string. The default value is `'degrees'`.

`az = azimuth(track,pt1,...)` specifies whether great circle azimuths or rhumb line azimuths are desired. Great circle azimuths, the default, are indicated with the standard *track* string `'gc'`. Rhumb line azimuths are indicated with the standard *track* string `'rh'`.

Examples

Consider two points on the same parallel, for example, (10°N,10°E) and (10°N,40°E). The azimuth between these two points depends upon the *track* string selected. Using the `pt1,pt2` notation, the two cases result in:

```
az = azimuth('gc',[10,10],[10,40])
az =
   87.3360

az = azimuth('rh',[10,10],[10,40])
az =
    90
```

The great circle path begins on an azimuth north of east to take the shortest route to the second point; the rhumb line proceeds along the parallel, on a constant due east heading.

Rhumb lines and great circles coincide along meridians and the Equator. Consider two points on the same meridian, say (10°N,10°E) and (40°N,10°E); this time using the `lat1, lon1, lat2, lon2` notation:

```
az = azimuth(10,10,40,10) % great circle sense
az =
     0

az = azimuth('rh',10,10,40,10)
az =
     0
```

The azimuths are the same because the paths coincide.

See Also

`distance`	Distance between points
`reckon distance`	New point from an azimuth and

`track`
`track1`
`track2`
	Tracing paths on the globe

Purpose

Camera position in geographic coordinates

Syntax

```
camposm(lat,long,alt)
[x,y,z] = camposm(lat,long,alt)
```

Description

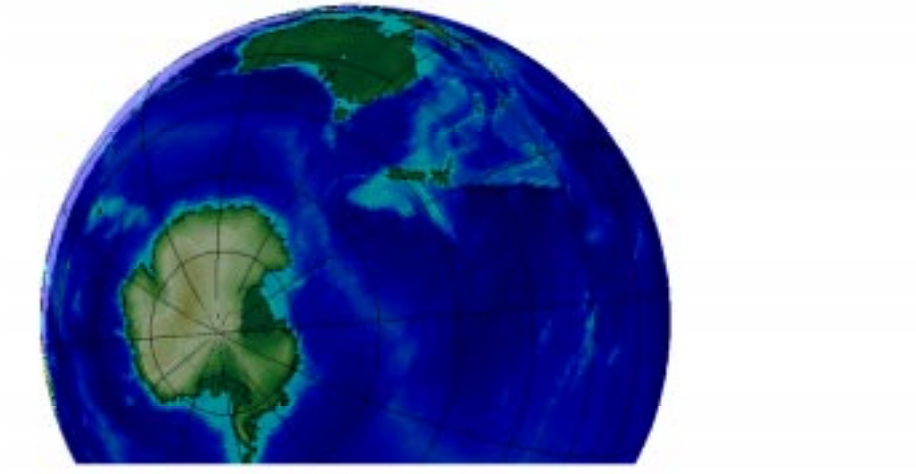
`camposm(lat,long,alt)` sets the axes `CameraPosition` property of the current map axes to the position specified in geographic coordinates. The inputs `lat` and `long` are assumed to be in the angle units of the current map axes.

`[x,y,z] = camposm(lat,long,alt)` returns the camera position in the projected Cartesian coordinate system.

Examples

Look at northern Australia from a point south of and one Earth radius above New Zealand:

```
figure; axesm('globe','galt',0); gridm('glinestyle','-')

load topo; meshm(topo,topolegend,size(topo)); demcmap(topo)
displaym(worldlo('POline'))
camlight; material(0.6*[ 1 1 1])

plat = -50; plon = 160; tlat = -10; tlon = 130;
camtargm(tlat,tlon,0); camposm(plat,plon,1); camupm(tlat,tlon)
set(gca,'CameraViewAngle',75)
```



See Also

`camtargm`	Camera target from geographic coordinates
`camupm`	Camera UpVector from geographic coordinates
`campos`	Camera position
`camva`	Camera view angle

Purpose

Camera target in geographic coordinates

Syntax

```
camtargm(lat,long,alt)
[x,y,z] = camtargm(lat,long,alt)
```

Description

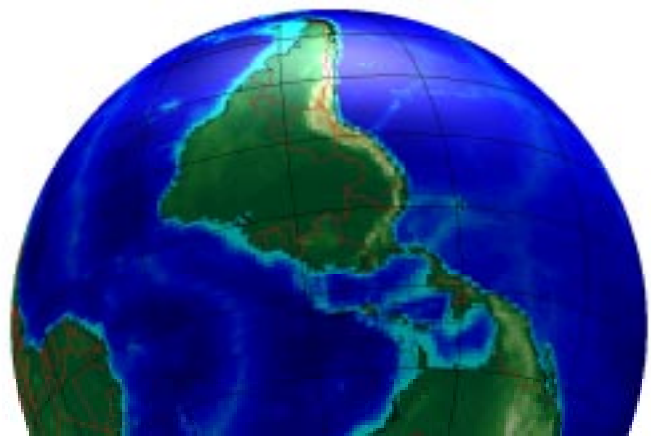
`camtargm(lat,long,alt)` sets the axes `CameraTarget` property of the current map axes to the position specified in geographic coordinates. The inputs `lat` and `long` are assumed to be in the angle units of the current map axes.

`[x,y,z] = camtargm(lat,long,alt)` returns the camera target in the projected Cartesian coordinate system.

Examples

Look down the spine of the Andes from a location 3 Earth radii above the surface:

```
figure; axesm('globe','galt',0); gridm('glinestyle','-')

load topo; meshm(topo,topolegend,size(topo)); demcmap(topo)
displaym(worldlo('POline'))
lightm(-80,-180); material(0.6*[ 1 1 1])

plat = 10; plon = -65; tlat = -30; tlon = -70;
camtargm(tlat,tlon,0); camposm(plat,plon,3);
camupm(tlat,tlon); camva(20)
set(gca,'CameraViewAngle',30)
```



See Also

`camposn`	Camera position from geographic coordinates
`camupm`	Camera UpVector from geographic coordinates
`camtarget`	Camera target
`camva`	Camera view angle

Purpose

Camera up vector in geographic coordinates

Syntax

```
camupm(lat,long)
[x,y,z] = camupm(lat,long)
```

Description

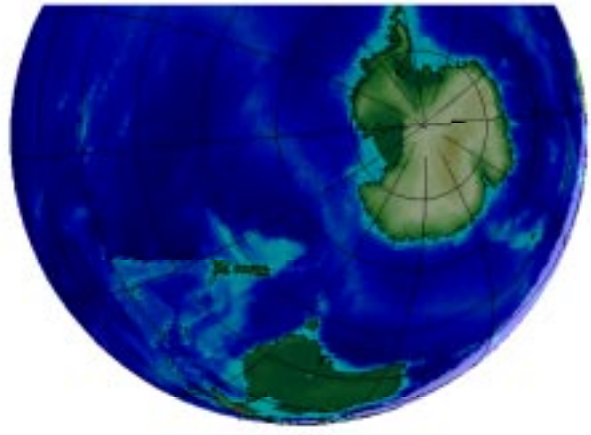
`camtargm(lat,long)` sets the axes `CameraUpVector` property of the current map axes to the position specified in geographic coordinates. The inputs `lat` and `long` are assumed to be in the angle units of the current map axes.

`[x,y,z] = camtargm(lat,long)` returns the camera position in the projected Cartesian coordinate system.

Examples

Look at northern Australia from a point south of and one Earth radius above New Zealand. Set `CameraUpVector` to the antipode of the camera target for that *down under* view:

```
figure; axesm('globe','galt',0); gridm('glinestyle','-')

load topo; meshm(topo,topolegend,size(topo)); demcmap(topo)
displaym(worldlo('POline'))
camlight; material(0.6*[ 1 1 1])

plat = -50; plon = 160; tlat = -10; tlon = 130;
[alat,alon] = antipode(tlat,tlon);
camtargm(tlat,tlon,0); camposm(plat,plon,1); camupm(alat,alon)
set(gca,'CameraViewAngle',80)
```



See Also

`camtargm`	Camera target from geographic coordinates
`camposm`	Camera position from geographic coordinates
`camup`	Camera up vector
`camva`	Camera view angle

Purpose

Return Greenwich coordinates for displayed map objects

Syntax

```
[lat,lon,alt] = cart2gr
[lat,lon,alt] = cart2grn(hndl)
[lat,lon,alt] = cart2grn(hndl,mstruct)
```

Description

When objects are projected and displayed on map axes, they are plotted in Cartesian coordinates appropriate for the selected projection. This command transforms those coordinates back into the Greenwich frame.

`[lat,lon,alt] = cart2grn` returns the latitude, longitude, and altitude data in Greenwich coordinates of the current map object, removing any clips or trims introduced during the display process from the output data.

`[lat,lon,alt] = cart2grn(hndl)` specifies the displayed map object desired with its handle `hndl`. The default handle is `gco`.

`[lat,lon,alt] = cart2grn(hndl,mstruct)` specifies the map structure associated with the object. The map structure of the current axes is the default.

See Also

`gcm`	Get current map structure
`mfwdtran`	Forward map projection transformation
`minvtran`	Inverse map projection transformation
`project`	Project existing graphics objects onto a map axes

cen2geod

Purpose

Convert from geocentric to geodetic latitudes

Syntax

```
lat = cen2geod(lat0)
lat = cen2geod(lat0,geoid)
lat = cen2geod(lat,units)
lat = cen2geod(lat,geoid,units)
```

Background

Longitudes have the same meaning whether the Earth is treated as a sphere or as an ellipsoid. However, the definition of latitude in the case of an ellipsoid is more complicated. Geodetic, or geographic, latitudes on an ellipsoid can be transformed from geocentric latitudes on the same ellipsoid.

Geodetic latitude: (also called *geographic latitude*) the angle a normal line passing through a surface point makes with the plane of the Equator.

Geocentric latitude: the angle a line from the center of the ellipsoid passing through a surface point makes with the plane of the Equator.

Description

`lat = cen2geod(lat0)` returns the geocentric latitudes provided in `lat0` transformed to geodetic latitudes in `lat`.

`lat = cen2geod(lat0,geoid)` defines the elliptical model of the Earth, given by the geoid vector `geoid`, for which `lat0` is transformed. The default geoid is the same as the default of `almanac('earth','geoid')`, the 1980 Geodetic Reference System ellipsoid.

`lat = cen2geod(lat,geoid,units)` defines the angle units of the inputs and outputs, where *units* is any valid angle units string. The default is `'degrees'`.

Examples

```
lat = cen2geod([0 45 90])
lat =
       0   45.1924   90.0000
```

See Also

`almanac`	Planetary data
`geod2cen` `iso2geod`	Other auxiliary latitude functions

Purpose

Replace entries with certain values throughout a matrix map

Syntax

```
mapout = changem(map,newcode)
mapout = changem(map,newcode,oldcode)
```

Description

`mapout = changem(map,newcode,oldcode)` returns a matrix map `mapout` identical to the input matrix `map`, except that each element of `map` with a value contained in the vector `oldcode` is replaced by the corresponding element of the vector `newcode`.

`oldcode` is 0 (scalar) by default, in which case `newcode` must be scalar. Otherwise, `newcode` and `oldcode` must be the same size.

Examples

Invent a map:

```
map = magic(3)
map =
     8     1     6
     3     5     7
     4     9     2
```

Replace instances of `8` or `9` with `0`'s:

```
mapout = changem(ans,[0 0],[9 8])
mapout =
     0     1     6
     3     5     7
     4     0     2
```

See Also

`maskm` Mask out specified matrix map entries

clabelm

Purpose

Label map contours

Syntax

```
h1 = clabelm(c,h)
h1 = clabelm(c,h,v)
h1 = clabelm(c,h,'manual')

h1 = clabelm(c)
h1 = clabelm(c,v)
h1 = clabelm(c,'manual')
```

Description

The `clabelm` function adds height labels to a two-dimensional contour plot. By default, `clabelm` labels all displayed contours and randomly selects label positions.

`c` is the contour matrix as described on the `contorm` reference page of this guide; `h` is the vector of handles for the displayed contours.

`h1 = clabelm(c,h)` rotates the labels and inserts them in line with the contour lines. The handles of the labels can be returned in `h1`.

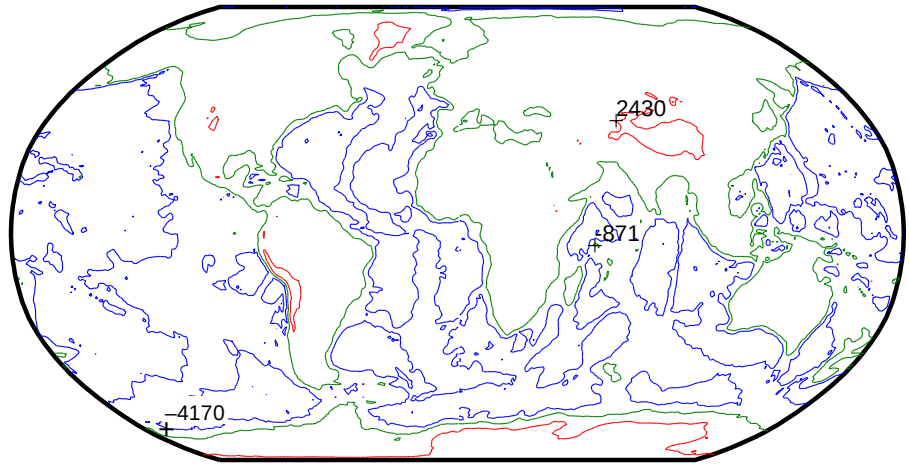
`h1 = clabelm(c,h,v)` creates in line labels only for those levels specified in the vector `v`.

`h1 = clabelm(c,h,'manual')` places contour labels at locations you select with a mouse. You press the left mouse button (the only 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 inserted in line with the contour lines.

`h1 = clabelm(c)`, `h1 = clabelm(c,v)`, and `h1 = clabelm(c,'manual')` operate as the above, except that instead of rotating the labels and placing them in line with the contours, the labels are upright, and a '+' indicates which contour line the label is annotating.

Examples

```
load topo
axesm robinson; framem
[c,h] = contorm(topo,topolegend,3);
clabelm(c)
```



See Also

`clegendm`	Display a legend for a contour plot of map data
`contorm`	Project a contour plot of map data
`contor3m`	Project a 3-D contour plot of map data
`clabel`	Contour plot elevation labels (see online *MATLAB Function Reference*)

clegendm

Purpose

Add legend labels to a map contour display

Syntax

```
clegendm(cs,h)
clegendm(cs,h,pos)
```

Description

This command will display a legend for a displayed map contour map.

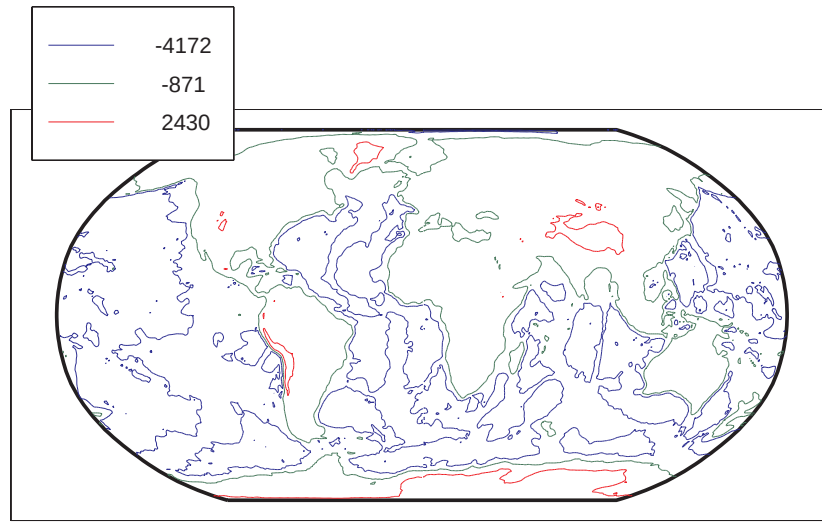
`clegendm(cs,h)` will display a legend for the contour map defined by the two-column contour definition matrix, `cs`, and the handle(s) `h`. Both of these inputs are produced as the outputs of either `contorm` or `contor3m`.

`clegendm(cs,h,pos)` allows you to specify the position of the legend in the display. The input `pos` can be any of the following integers, with the indicated result:

`0`	Automatic placement (this is the default)
`1`	Upper right-hand corner
`2`	Upper left-hand corner
`3`	Lower left-hand corner
`4`	Lower right-hand corner
`-1`	To the right of the plot

Examples

```
load topo
axesm robinson; framem
[cs,h] = contorm(topo,topolegend,3);
clegendm(cs,h,2)
```



See Also

`clabelm`	Label a contour plot of map data
`contorm`	Project a contour plot of map data
`contor3m`	Project a 3-D contour plot of map data
`contourc`	Low-level contour plot computation (see online *MATLAB Function Reference*)

clma

Purpose

Clear current map axes

Syntax

```
clma
clma all
clma purge
```

Description

`clma` deletes all displayed map objects from the current map axes but leaves the frame if it is displayed.

`clma all` deletes all displayed map objects, including the frame, but it leaves the map structure in the axes UserData property, thereby retaining the map axes.

`clma purge` clears all displayed map objects and clears the axes `UserData` slot, effectively changing the map axes to standard axes. This is equivalent to `cla reset`.

See Also

`cla`	Clear current Axes (see online *MATLAB Function Reference*)
`clmo`	Clear specified graphics objects
`handlem`	Return handles of displayed map objects
`hidem`	Hide specified graphics objects
`namem`	Determine names of valid graphics objects
`showm`	Show specified graphics objects
`tagm`	Assign name to graphics object tag property

Purpose

Clear specified graphics object

Syntax

```
clmo
clmo(handle)
clmo(object)
```

Description

`clmo` deletes all displayed graphics objects on the current axes.

`clmo(handle)` deletes those objects specified by their handles.

`clmo(object)` deletes those objects with names identical to the input string. This can be any string recognized by the `handlem` function, including entries in the `Tag` property of each object, or the object `Type` if the `Tag` property is empty.

See Also

`clma`	Clear current map
`handlem`	Return handles of displayed map objects
`hidem`	Hide specified graphics objects
`namem`	Determine names of valid graphics objects
`showm`	Show specified graphics objects
`tagm`	Assign name to graphics object tag property

cnf2geod

Purpose

Convert from conformal to geodetic latitudes

Syntax

```
lat = cnf2geod(lat0)
lat = cnf2geod(lat0,geoid)
lat = cnf2geod(lat,units)
lat = cnf2geod(lat,geoid,units)
```

Background

Longitudes have the same meaning whether the Earth is treated as a sphere or as an ellipsoid. However, the definition of latitude in the case of an ellipsoid is more complicated. Geodetic, or geographic, latitudes on an ellipsoid can be transformed from latitudes on an auxiliary sphere with certain properties.

Geodetic latitude: (also called *geographic latitude*) the angle a normal line passing through a surface point makes with the plane of the Equator.

Conformal latitude: latitudes on an auxiliary sphere that is conformal relative to the ellipsoid.

Description

`lat = cnf2geod(lat0)` returns the conformal latitudes provided in `lat0` transformed to geodetic latitudes, which are returned in `lat`.

`lat = cnf2geod(lat0,geoid)` defines the elliptical model of the Earth, given by the geoid vector `geoid`, to which `lat0` is transformed. The default geoid is the same as the default of `almanac('earth','geoid')`, the 1980 Geodetic Reference System ellipsoid.

`lat = cnf2geod(lat,geoid,units)` defines the angle units of the inputs and outputs, where *units* is any valid angle units string. The default is `'degrees'`.

Examples

```
lat = cnf2geod([0 45 90])
lat =
         0   45.1923   90.0000
```

See Also

`almanac`	Planetary data
`geod2cnf` `iso2geod`	Other auxiliary latitude functions

Purpose

Load coastline data

Syntax

```
ll = coast
```

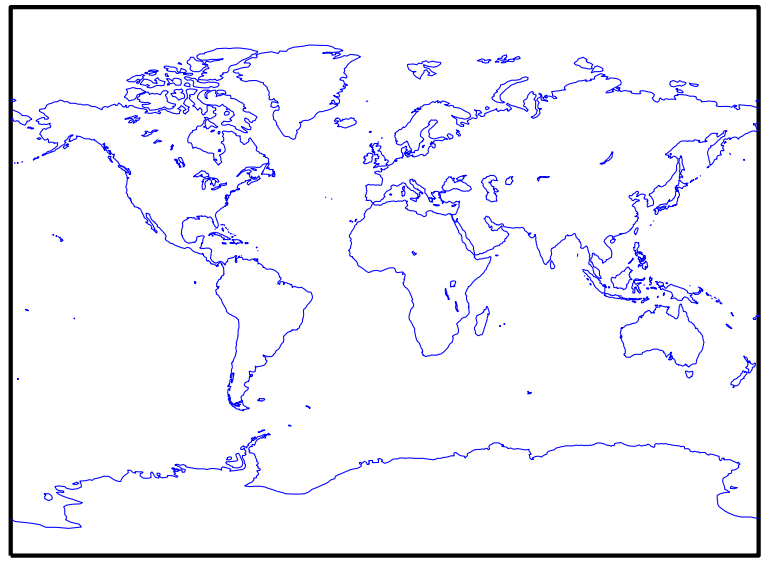
Description

`ll = coast` returns the world vector shoreline in the coast MAT-file as a two-column vector of latitudes and longitudes in degrees.

Examples

Add the coastline data to the map without loading it into the workspace.

```
axesm miller; framem
plotm(coast)
```



See Also

`coast.mat`	Coastline data file
`worldlo`	World atlas data function
`usalo`	Low-resolution United States data function
`usahi`	High-resolution United States data function

combntns

Purpose

Determine combinations of a set of values

Syntax

```
combos = combntns(set,subset)
```

Description

The `combntns` command provides the combinatorial subsets of a set of numbers. It is similar to the mathematical expression a *choose b*, except that instead of the number of such combinations, the actual combinations are returned. In combinatorial counting, the ordering of the values is not significant.

`combos = combntns(set,subset)` returns a matrix whose rows are the various combinations that can be taken of the elements of the vector `set` of length `subset`. Many combinatorial applications can make use of a vector `1:n` for the input set to return generalized, indexed combination subsets.

The numerical value of the mathematical statement *a choose b* is `size(combos,1)`.

Examples

How can the numbers 1 to 5 be taken in sets of three (i.e., what is *5 choose 3*)?

```
combos = combntns(1:5,3)
combos =
     1     2     3
     1     2     4
     1     2     5
     1     3     4
     1     3     5
     1     4     5
     2     3     4
     2     3     5
     2     4     5
     3     4     5

size(combos,1)  % "5 choose 3"
ans =
    10
```

Note that if a value is repeated in the input vector, each occurrence is treated as independent:

```
combos = combntns([2 2 5],2)
combos =
     2     2
     2     5
     2     5
```

Remarks

This is a recursive function.

Purpose

Project three-dimensional comet plot on map axes

Syntax

```
comet3m(lat,lon,z)
comet3m(lat,lon,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.

`comet3m(lat,lon,z)` traces a comet plot through the points specified by the input latitude, longitude, and altitude vectors.

`comet3m(lat,lon,z,p)` specifies a comet body of length `p*length(lat)`.The input `p` is `0.1` by default.

Examples

Create a 3-D comet plot of the coastlines data:

```
load coast
z = (1:length(lat))'/3000;
axesm miller
framem; gridm;
setm(gca,'galtitude',max(z)+.5)
view(3)
comet3m(lat,long,z,0.01)
```

See Also

`comet3`	3-D comet-like trajectories (see online *MATLAB Function Reference*)
`cometm`	Two-dimensional comet plot projected on map axes

cometm

Purpose

Project two-dimensional comet plot on map axes

Syntax

```
cometm(lat,lon)
cometm(lat,lon,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.

`cometm(lat,lon)` traces a comet plot through the points specified by the input latitude and longitude vectors.

`cometm(lat,lon,p)` specifies a comet body of length `p*length(lat)`. The input `p` is `0.1` by default.

Examples

Create a comet plot of the coast lines data:

```
load coast
axesm miller
framem
cometm(lat,long,0.01)
```

See Also

`comet`	Comet-like trajectories (see online *MATLAB Function Reference*)
`comet3`	Three-dimensional comet plot projected on map axes

Purpose

Project a contour plot of map data in 3-D space

Syntax

```
c = contor3m(lat,lon,map)
c = contor3m(lat,lon,map,LineType)
c = contor3m(lat,lon,map,PropertyName,PropertyValue,...)

c = contor3m(lat,lon,map,n,...)
c = contor3m(lat,lon,map,v,...)
[c,h] = contor3m(lat,lon,map,...)
```

Description

`contor3m(lat,lon,map)` produces a 3D contour plot of map data projected onto the current map axes. The input latitude and longitude vectors can be the size of map (as in a general matrix map), or can specify the corresponding row and column dimensions for the map

`contor3m(map,maplegend)` produces a contour plot of map data in a regular matrix map.

`contor3m(lat,lon,map,LineSpec)` uses any valid *LineSpec* string to draw the contour lines.

`contor3m(lat,lon,map,PropertyName,PropertyValue,...)` uses the line properties specified to draw the contours.

`contor3m(lat,lon,map,n,...)` draws n contour levels, where n is a scalar.

`contor3m(lat,lon,map,v,...)` draws contours at the levels specified by the input vector v.

`contor3m(map,maplegend,...)` takes any of the optional arguments described above.

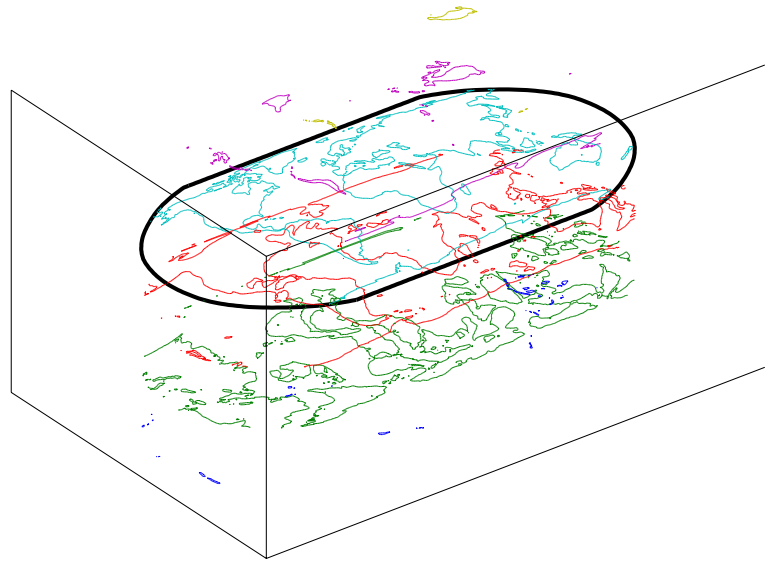
`c = contor3m(...)` returns a standard contour matrix, with the first row representing longitude data and the second row represents latitude data.

`[c,h] = contor3m(...)` returns the contour matrix and the handles to the contour lines drawn.

`contor3m`, without any inputs, will activate a GUI to project contour lines onto the current map axes.

Examples

```
load topo
axesm robinson; framem; view(3)
contor3m(topo,topolegend)
set(gca,'DataAspectRatio',[1 1 3000])
```



See Also

`clabelm`	Label a contour plot of map data
`clegendm`	Display a legend for a contour plot of map data
`contourc`	Contour computation (see online *MATLAB Function Reference*)
`contorm`	Project a contour plot of map data

Purpose

Project a contour plot of map data

Syntax

```
c = contorm(lat,lon,map)
c = contorm(lat,lon,map,LineType)
c = contorm(lat,lon,map,PropertyName,PropertyValue,...)

c = contorm(lat,lon,map,n,...)
c = contorm(lat,lon,map,v,...)
[c,h] = contorm(lat,lon,map,...)
```

Description

`contorm(lat,lon,map)` produces a contour plot of map data projected onto the current map axes. The input latitude and longitude vectors can be the size of map (as in a general matrix map), or can specify the corresponding row and column dimensions for the map.

`contorm(map,maplegend)` produces a contour plot of map data in a regular matrix map.

`contorm(lat,lon,map,LineSpec)` uses any valid LineSpec string to draw the contour lines.

`contorm(lat,lon,map,PropertyName,PropertyValue,...)` uses the line properties specified to draw the contours.

`contorm(lat,lon,map,n,...)` draws n contour levels, where n is a scalar.

`contorm(lat,lon,map,v,...)` draws contours at the levels specified by the input vector v.

`contorm(map,maplegend,...)` takes any of the optional arguments described above.

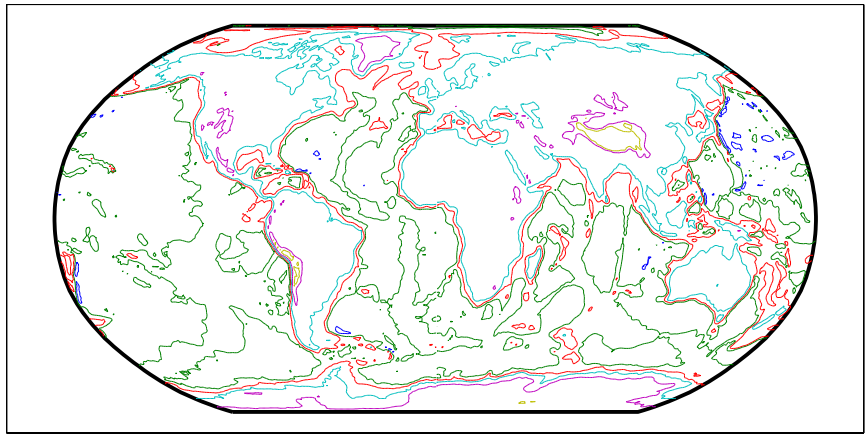
`c = contorm(...)` returns a standard contour matrix, with the first row representing longitude data and the second row represents latitude data.

`[c,h] = contorm(...)` returns the contour matrix and the handles to the contour lines drawn.

`contorm`, without any inputs, will activate a GUI to project contour lines onto the current map axes.

Examples

```
load topo
axesm robinson; framem
contorm(topo,topolegend)
```



See Also

`clabelm`	Label a contour plot of map data
`clegendm`	Display a legend for a contour plot of map data
`contourc`	Contour computation (see online *MATLAB Function Reference*)
`contor3m`	Project a 3-D contour plot of map data

Purpose

Project a filled contour map

Syntax

```
contourfm

contourfm(lat,lon,map)
contourfm(lat,lon,map,LineSpec)
contourfm(lat,lon,map,PropertyName,PropertyValue,...)
contourfm(lat,lon,map,n,...)
contourfm(lat,lon,map,v,...)

contourfm(map,maplegend)
contourfm(map,maplegend,...)

 c = contourfm(...)
[c,h] = contourfm(...)
```

Description

`contourfm(lat,lon,map)` produces a contour plot of map data projected onto the current map axes. The input latitude and longitude vectors can be the size of `map` (as in a general matrix map), or can specify the corresponding row and column dimensions for the map.

`contourfm(map,maplegend)` produces a contour plot of map data in a regular matrix map.

`contourfm(lat,lon,map,LineSpec)` uses any valid *LineSpec* string to draw the contour lines.

`contourfm(lat,lon,map,PropertyName,PropertyValue,...)` uses the line properties specified to draw the contours.

`contourfm(lat,lon,map,n,...)` draws `n` contour levels, where `n` is a scalar.

`contourfm(lat,lon,map,v,...)` draws contours at the levels specified by the input vector `v`.

`contourfm(map,maplegend,...)` takes any of the optional arguments described above.

`c = contourfm(...)` returns a standard contour matrix, with the first row representing longitude data and the second row represents latitude data.

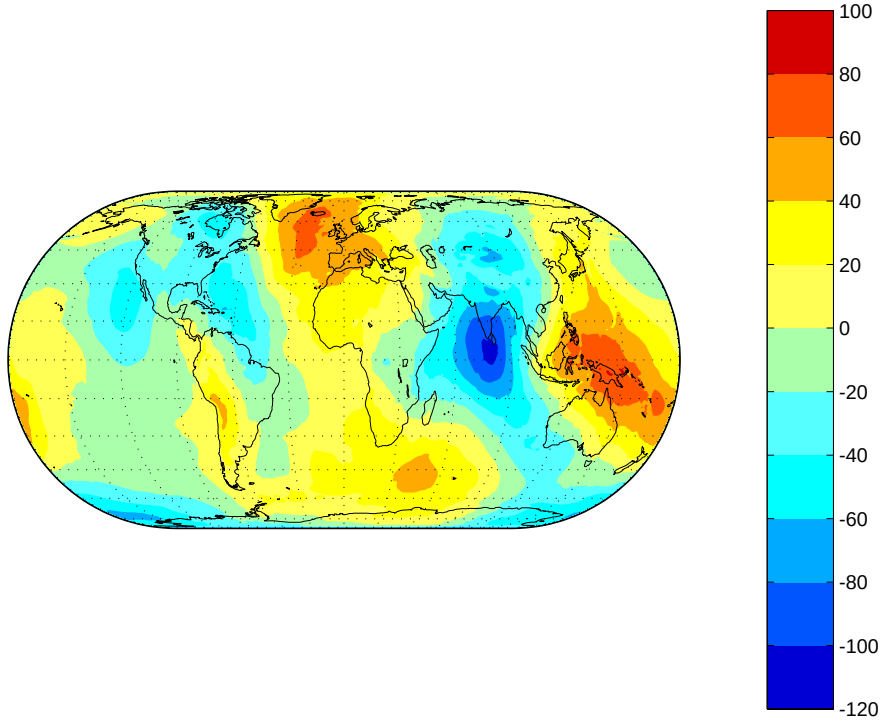
`[c,h] = contourfm(...)` returns the contour matrix and the handles to the contour lines drawn.

`contourfm`, without any inputs, will activate a GUI to project contour lines onto the current map axes.

Examples

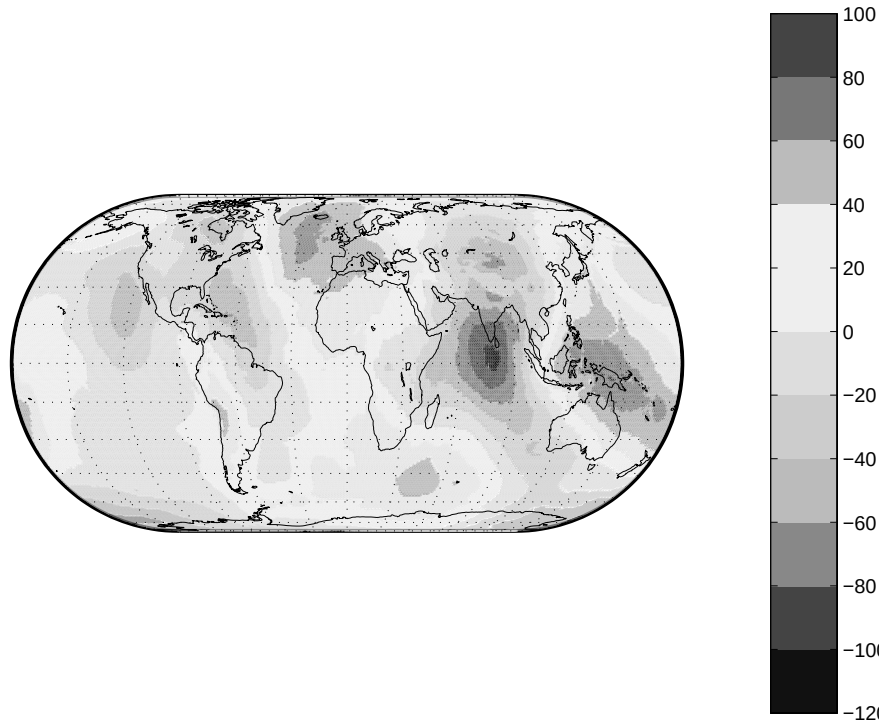
Plot the Earth's geoid with filled contours. The data is in meters.

```
load geoid
figure
axesm eckert4
framem;gridm
plotm(coast,'k')

caxis([-120 100]);colormap(jet(11));colorbar
contourfm(geoid,geoidlegend,-120:20:100);
```



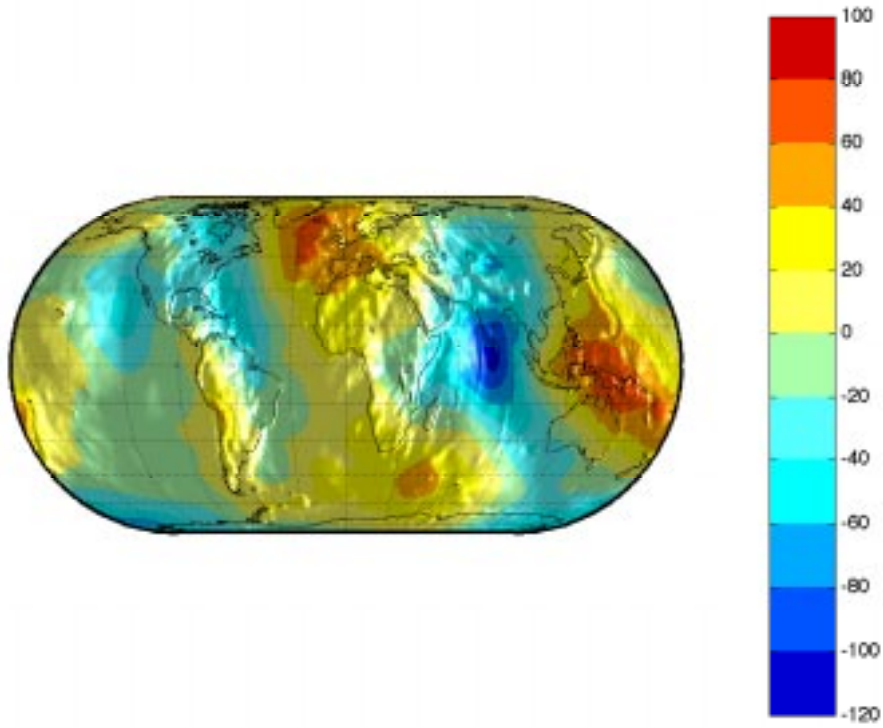
You can reproduce the filled contour display by using a surface instead of the patches created by `contourfm`.

```
figure
axesm eckert4
framem;gridm
plotm(coast,'k')

caxis([-120 100]);colormap(jet(11));colorbar
meshm(geoid,geoidlegend,size(geoid),'Facecolor','interp')
```



Surfaces also allow use of lighting to bring out the smaller variations in the data.

```
clmo surface
meshm(geoid,geoidlegend,size(geoid),geoid,'Facecolor','interp')
light;lighting phong; material(0.6*[ 1 1 1])
set(gca,'dataaspectratio',[ 1 1 200])
gridm reset
zdatam(handlem('line'),max(geoid(:)))
```



Limitations

`contourfm` may not fill properly with azimuthal projections.

Remarks

The patches are drawn at a range of z-levels < 0 to ensure proper display. Contours are displayed with no edge colors. To combine contour fill with contour lines, use both `contourfm` and `contorm`.

In most circumstances, contour plots made with surfaces are preferable to the filled patches created by `contourfm`. Surfaces render more quickly and take less time to project and re-project. The use of surfaces also allows surface lighting to create shaded 3-D maps.

See Also

`contorm`	Project a contour plot of data onto the current map axes
`contor3m`	Project a 3-D contour plot of data onto the current map axes
`clabelm`	Add contour labels to a map contour plot

country2mtx

Purpose

Construct a matrix map for a country in the `worldlo` database

Syntax

```
[map,maplegend] = country2mtx(countryname,scale)
[map,maplegend] = country2mtx(countryname,scale,latlim,lonlim)
[map,maplegend] = country2mtx(countryname,map1,maplegend1)
```

Description

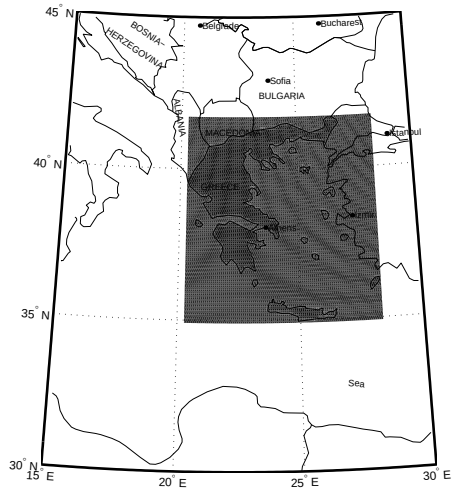
`[map,maplegend] = country2mtx(countryname,scale)` constructs a regular matrix map of a country in the `worldlo` `POpatch` database. The scale factor represents the number of matrix entries per degree of latitude and longitude (e.g., 10 entries per degree, 100 entries per degree). The scale input must be scalar. The returned matrix has values of 0 in the interior of the country, 1 on the border and 2 in the exterior.

`[map,maplegend] = country2mtx(countryname,scalelatlim,lonlim)` uses the two-element vector latitude and longitude limits to define the extent of the map. If omitted, the limits are computed automatically.

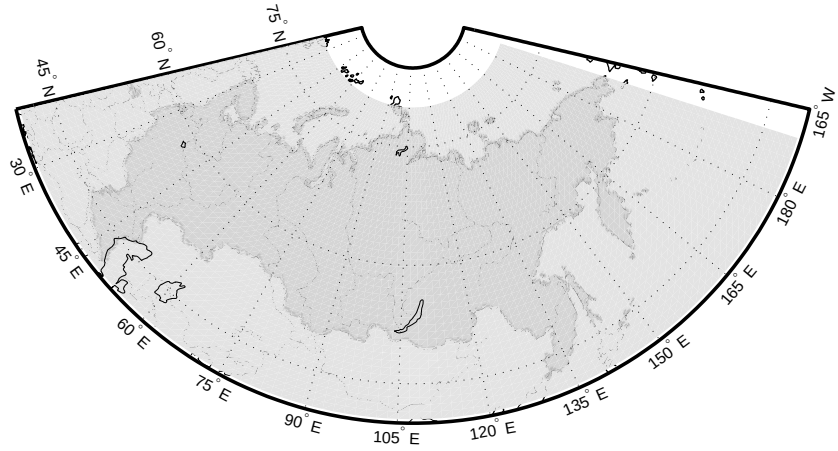
`[map,maplegend] = country2mtx(countryname,map1,maplegend1)` uses the limits of the provided map.

Examples

```
[map,maplegend] = country2mtx('greece',50);
worldmap('greece');
meshm(map,maplegend)
```



```
load russia
[map2,maplegend2] = country2mtx('russia',map,maplegend);
figure
worldmap('russia');
meshm(map2,maplegend2)
polcmap
```



Limitations

`country2mtx` may not fill properly if the vector data extends beyond a pole.

See Also

`vec2mtx`	Regular matrix map from vector data
`ltln2val`	Returns map code value associated with positions
`imbed`	Encodes data points into a regular matrix map
`encodem`	Fills in indexed maps with specified seeds
`interpm`	Interpolates vector data to a specified data separation

crossfix

Purpose

Determine cross fix positions from bearings and ranges

Syntax

```
[newlat,newlon] = crossfix(lat,long,az)
[newlat,newlon] = crossfix(lat,long,az,units)
[newlat,newlon] = crossfix(lat,long,az_range,case)
[newlat,newlon] = crossfix(lat,long,az_range,case,units)
[newlat,newlon] = crossfix(lat,long,az_range,case,drlat,drlong)
[newlat,newlon] = crossfix(lat,long,az_range,drlat,drlong,units)
[newlat,newlon] = crossfix(lat,long,az_range,case,...
                           drlat,drlong,units)
mat = crossfix(...)
```

Description

This function calculates the points of intersection between a set of objects taken in pairs. Given great circle azimuths and/or ranges from input points, the locations of the possible intersections are returned. This is different from the navigational function `navfix` in that `crossfix` uses great circle measurement, while `navfix` uses rhumb line azimuths and nautical mile distances.

`[newlat,newlon] = crossfix(lat,long,az)` returns the intersection points of all pairs of great circles passing through the points given by the column vectors `lat` and `long` that have azimuths `az` at those points. The outputs are two-column matrices `newlat` and `newlon` in which each row represents the two intersections of a possible pairing of the input great circles. If there are *n* input objects, there will be *n choose 2* pairings.

`[newlat,newlon] = crossfix(lat,long,az_range,case)` allows the input `az_range` to specify either azimuths or ranges. Where the vector `case` equals 1, the corresponding element of `az_range` is an azimuth; where `case` is 0, `az_range` is a range. The default value of `case` is a vector of ones (azimuths).

`[newlat,newlon] = crossfix(lat,long,az_range,case,drlat,drlong)` resolves the ambiguities when there is more than one intersection between two objects. The scalar valued `drlat` and `drlong` provide the location of an estimated (dead reckoned) position. The outputs `newlat` and `newlong` are column vectors in this case, returning only the intersection closest to the estimated point. When this option is employed, if any pair of objects fails to intersect, no output is returned and the warning `No Fix` is displayed.

`[newlat,newlon] = crossfix(lat,long,az,`*units*`),`
`[newlat,newlon] = crossfix(lat,long,az_range,case,`*units*`),`
`[newlat,newlon] = crossfix(lat,long,az_range,drlat,drlong,`*units*`),`
and `[newlat,newlon] =`
`crossfix(lat,long,az_range,case,drlat,drlong,`*units*`)` allow the specification of the angle units to be used for all angles and ranges, where *units* is any valid angle units string. The default value of *units* is `'degrees'`.

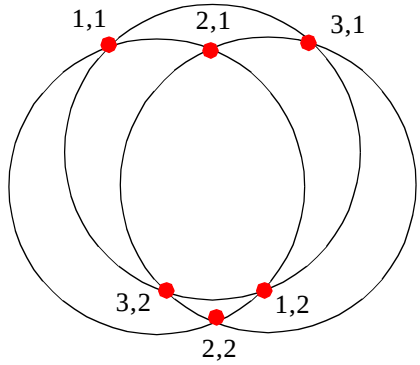
`mat = crossfix(...)` returns the output in a two- or four-column matrix `mat`.

Examples

Where do the small circles defined as all points 8° in distance from the points (0°,0°), (5°N,5°E), and (0°,10°E)" intersect?

```
[newlat,newlong] = crossfix([0 5 0]',[0 5 10]',[8 8 8]',[0 0 0]')
newlat =
    7.5594   -2.5744
    6.2529   -6.2529
    7.5594   -2.5744
newlong =
   -2.6260    7.5770
    5.0000    5.0000
   12.6260    2.4230
```

Here is an illustration to show why there are six intersections:



If a dead reckoning position is provided, say (0°,5°E), then one from each pair is returned (the closest):

```
[newlat,newlong] = crossfix([0 5 0]',[0 5 10]',...
                            [8 8 8]',[0 0 0]',0,5)
newlat =
   -2.5744
    6.2529
   -2.5744
newlong =
    7.5770
    5.0000
    2.4230
```

See Also

gcxgc gcxsc scxsc rhxrh	Other intersection functions
navfix	Mercator-based navigational fixing

Purpose

Control vertical exaggeration in a map display

Syntax

```
daspectm(zunits)
daspectm(zunits,vfac)
daspectm(zunits,vfac,lat,long)
daspectm(zunits,vfac,lat,long,az)
daspectm(zunits,vfac,lat,long,az,gunits)
daspectm(zunits,vfac,lat,long,az,gunits,radius)
```

Description

`daspectm(zunits)` sets the figure `'DataAspectRatio'` property so that the z-axis is in proportion to the x and y projected coordinates. This permits elevation data to be displayed without vertical distortion. The string *zunits* specifies the units of the elevation data, and can be any string recognized by `distdim`.

`daspectm(zunits,vfac)` sets the `'DataAspectRatio'` property so that the z-axis is vertically exaggerated by the factor `vfac`. If omitted, the default is no vertical exaggeration.

`daspectm(zunits,vfac,lat,long)` sets the aspect ratio based on the local map scale at the specified geographic location. If omitted, the default is the center of the map limits.

`daspectm(zunits,vfac,lat,long,az)` also specifies the direction along which the scale is computed. If omitted, 90 degrees (west) is assumed.

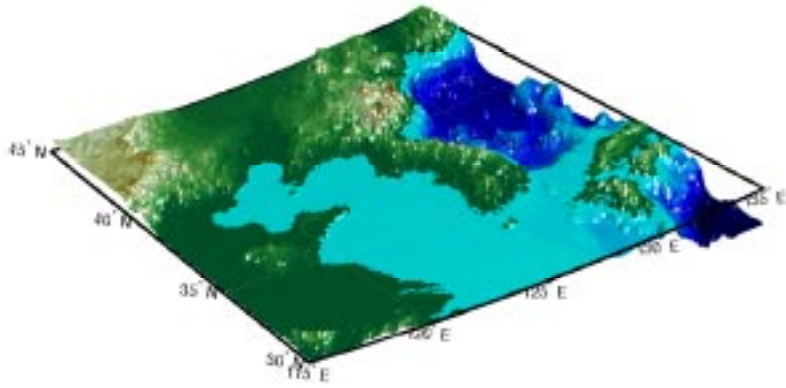
`daspectm(zunits,vfac,lat,long,az,gunits)` also specifies the units in which the geographic position and direction are given. If omitted, `'degrees'` is assumed.

`daspectm(zunits,vfac,lat,long,az,gunits,radius)` uses the last input to determine the radius of the sphere. If `radius` is a string, then it is evaluated as an `almanac` body to determine the spherical radius. If numerical, it is the radius of the desired sphere in `zunits`. If omitted, the default radius of the Earth is used.

Examples

Show the elevation map of the Korean peninsula with a vertical exaggeration factor of 30:

```
load korea
[latlim,lonlim] = limitm(map,maplegend);

worldmap(latlim,lonlim,'none')
meshm(map,maplegend,size(map),map)
demcmap(map)

view(3)
daspectm('m',30)
tightmap
camlight
```



Limitations

The relationship between the vertical and horizontal coordinates holds only as long as the geoid or scale factor properties of the map axes remain unchanged. If you change the scaling between geographic coordinates and projected axes coordinates, execute `daspectm` again.

See Also

`daspect`	Data aspect ratio
`paperscale`	Figure paper size for a given map scale

Purpose

Initialize a default map projection structure

Syntax

```
mstruct = defaultm
mstruct = defaultm(projection)
mstruct = defaultm(mstruct)
[mstruct,msg] = defaultm(...)
```

Description

The map projection structure contains all the information needed to project and display geographic data. It normally resides in the `UserData` property of a map axes, but it can also be used directly to project data without display.

`mstruct = defaultm` creates an empty map projection structure.

`mstruct = defaultm(`*`projection`*`)` initializes the map structure for the specified map projection. *`projection`* is any valid projection string, such as `'sinusoid'`.

`mstruct = defaultm(mstruct)` sets appropriate defaults based on existing parameter values in the map structure `mstruct`.

`[mstruct,msg] = defaultm(...)` returns the string `msg`, indicating any error encountered.

Examples

Create an empty map projection structure for a Mercator projection:

```
mstruct = defaultm('mercator')
mstruct =
     mapprojection: 'mercator'
              zone: []
        angleunits: 'degrees'
            aspect: 'normal'
      falseeasting: []
     falsenorthing: []
       fixedorient: []
             geoid: [1 0]
       maplatlimit: []
       maplonlimit: []
      mapparallels: 0
        nparallels: 1
            origin: []
       scalefactor: []
           trimlat: [-86 86]
           trimlon: [-180 180]
             frame: []
             ffill: 100
        fedgecolor: [0 0 0]
        ffacecolor: 'none'
         flatlimit: []
        flinewidth: 2
         flonlimit: []
              grid: []
         galtitude: Inf
            gcolor: [0 0 0]
        glinestyle: ':'
        glinewidth: 0.5000
    mlineexception: []
         mlinefill: 100
        mlinelimit: []
     mlinelocation: []
      mlinevisible: 'on'
    plineexception: []
         plinefill: 100
        plinelimit: []
```

```
      plinelocation: []
       plinevisible: 'on'
          fontangle: 'normal'
          fontcolor: [0 0 0]
           fontname: 'helvetica'
           fontsize: 9
          fontunits: 'points'
         fontweight: 'normal'
        labelformat: 'compass'
      labelrotation: 'off'
         labelunits: []
      meridianlabel: []
     mlabellocation: []
     mlabelparallel: []
        mlabelround: 0
      parallellabel: []
     plabellocation: []
     plabelmeridian: []
        plabelround: 0
```

Now change the map origin to `[0 90 0]`, and fill in default projection parameters accordingly:

```
mstruct.origin = [0 90 0];
mstruct = defaultm(mstruct)
mstruct =
     mapprojection: 'mercator'
              zone: []
        angleunits: 'degrees'
            aspect: 'normal'
       falseeasting: 0
      falsenorthing: 0
        fixedorient: []
              geoid: [1 0]
        maplatlimit: [-86 86]
        maplonlimit: [-180 180]
       mapparallels: 0
         nparallels: 1
             origin: [0 0 0]
        scalefactor: 1
            trimlat: [-86 86]
```

```
       trimlon: [-180 180]
         frame: 'off'
         ffill: 100
    fedgecolor: [0 0 0]
    ffacecolor: 'none'
     flatlimit: [-86 86]
    flinewidth: 2
     flonlimit: [-180 180]
          grid: 'off'
     galtitude: Inf
        gcolor: [0 0 0]
    glinestyle: ':'
    glinewidth: 0.50000000000000
mlineexception: []
     mlinefill: 100
    mlinelimit: []
 mlinelocation: 30
  mlinevisible: 'on'
plineexception: []
     plinefill: 100
    plinelimit: []
 plinelocation: 15
  plinevisible: 'on'
     fontangle: 'normal'
     fontcolor: [0 0 0]
      fontname: 'helvetica'
      fontsize: 9
     fontunits: 'points'
    fontweight: 'normal'
   labelformat: 'compass'
 labelrotation: 'off'
    labelunits: 'degrees'
 meridianlabel: 'off'
mlabellocation: 30
mlabelparallel: 86
   mlabelround: 0
 parallellabel: 'off'
plabellocation: 15
plabelmeridian: -180
   plabelround: 0
```

See Also

`axesm`	Define map axes and set map properties
`gcm`	Get current map data structure
`mfwdtran`	Map forward transformation
`minvtran`	Map inverse transformation
`setm`	Set and modify map properties

deg2dms, deg2dm

Purpose

Convert angle units from degrees to *dms* or *dm* format

Syntax

```
anglout = deg2dms(anglin)
angleout = deg2dm(anglin)
```

Description

`anglout = deg2dms(anglin)` converts angles input in degrees to the equivalent measure in the degrees-minutes-seconds (*dms*) format.

`angleout = deg2dm(anglin)` converts angles input in degrees to the equivalent measure in the degrees-minutes (*dm*) format. This is the *dms* format, properly rounded to just degrees and minutes.

Example

```
deg2dms(23.561)
ans =
       2333.40

deg2dm(23.561)
ans =
       2334.00
```

See Also

`angledim`	Convert angle units
`dms2mat`	Convert from *dms* to separated matrix components
`deg2rad` `dms2rad`	Other direct angle conversion functions
`dat2dms`	Convert from separated matrices input to *dms*

Purpose

Convert distance from degrees to kilometers, nautical miles, or statute miles

Syntax

```
distout = deg2km(distin)
distout = deg2km(distin,radius)

distout = deg2nm(distin)
distout = deg2nm(distin,radius)

distout = deg2sm(distin)
distout = deg2sm(distin,radius)
```

Description

`distout = deg2km(distin)` converts the input distance given in degrees to kilometers.

`distout = deg2nm(distin)` and `distout = deg2sm(distin)` work identically, except that the output units are nautical miles and statute miles, respectively.

`distout = deg2km(distin,radius)` specifies the radius of the sphere to use, since a degree of arc length covers less distance, for example, on Mars than it would on the Earth. You can enter the radius as a number in kilometers, as a call to the `almanac` function (e.g., `almanac('mars','radius','km')`), again in the appropriate units, or you can pass in a string planet name (e.g., `'mars'`), and the function will make the appropriate call to the `almanac` function. The radius of the Earth is the default.

For `distout = deg2nm(distin,radius)` and `distout = deg2sm(distin,radius)`, make sure your input radius is in the appropriate units, or just use the planet name string.

Examples

A degree of arc length is about 60 nautical miles:

```
deg2nm(1)
ans =
   60.0405
```

This is not true on Mercury, of course:

```
deg2nm(1,'mercury')
ans =
   22.9852
```

See Also

`deg2rad`	Convert degrees to radians
`distdim`	Convert distance units
`nm2km` `sm2deg`	Other direct distance conversion functions

Purpose

Convert angle (or distance) units from degrees to radians

Syntax

```
anglout = deg2rad(anglin)
```

Description

`anglout = deg2rad(anglin)` converts angles input in degrees to the equivalent measure in radians.

Remarks

This is both an angle conversion function and a distance conversion function, since arc length can be a measure of distance in either radians or degrees, provided that the radius is known.

Example

Show that there are 2¼ radians in a full circle:

```
2*pi - deg2rad(360)
ans =
     0
```

See Also

`angledim`	Convert angle units
`deg2dms` `dms2rad`	Other direct angle conversion functions
`distdim`	Convert distance units
`nm2km` `sm2deg`	Other direct distance conversion functions
`rad2deg`	Convert from radians to degrees

demcmap

Purpose

Create colormaps for digital elevation maps

Syntax

```
demcmap(map)
demcmap(map,ncolors)
demcmap(map,ncolors,cmapsea,cmapland)
demcmap(color,map,spec)
demcmap(color,map,spec,cmapsea,cmapland)
[cmap,caxis] = demcmap(...)
```

Description

`demcmap(map)` creates and assigns a colormap for elevation data. The colormap has the number of land and sea colors in the same proportions as the maximum elevations and depths in the matrix map. With no output arguments, the colormap is applied to the current figure and the coloraxis is set so that the interface between the land and sea is in the right place.

`demcmap(map,ncolors)` makes a colormap with a length of `ncolors`. The default value is 64.

`demcmap(map,ncolors,cmapsea,cmapland)` allows the default colors for sea and land to be replaced. The colors in the created colormap are interpolated from the RGB color matrix inputs, which can be of any length. Default colors for either land or sea can still be retained by providing an empty matrix in place of the color matrices. The current figure colormap can be specified by entering the string `'window'` in place of either RGB matrix.

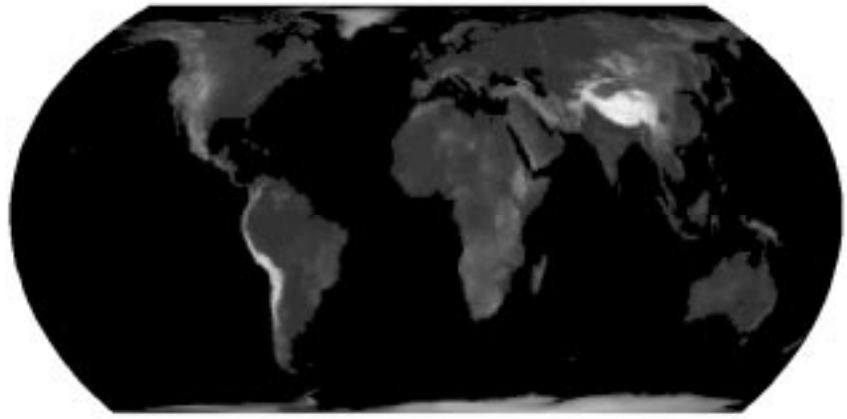
`demcmap(color,map,spec)` uses the *color* string to define a colormap. If the string is set to `'size'`, spec is the length of the colormap. If it is set to `'inc'`, spec is the size of the altitude range assigned to each color. If omitted, *color* is `'size'` by default.

`demcmap(color,map,spec,cmapsea,cmapland)` allows for both coloring options along with specified colors.

Examples

Display world topographical map using grayscale colors:

```
load topo
axesm hatano
meshm(topo,topolegend)
demcmap(topo,64,[0 0 0],[.2 .2 .2; 1 1 1])
```



See Also

`caxis`	Color axis scaling (see online *MATLAB Function Reference*)
`colormap`	Set and get the current colormap (see online *MATLAB Function Reference*)
`meshlsrm`	Project 3-D lighted shaded relief of regular matrix map
`meshm`	Display a regular matrix map warped to a projected graticule
`surflsrm`	Project 3-D lighted shaded relief of general matrix map
`surfm`	Display a matrix map warped to a projected graticule

departure

Purpose

Compute departure between longitudes at specified latitudes

Syntax

```
dist = departure(long1,long2,lat)
dist = departure(long1,long2,lat,units)
dist = departure(long1,long2,lat,geoid)
dist = departure(long1,long2,lat,geoid,units)
```

Description

Departure is the distance along a parallel between two points. Whereas a degree of latitude is always the same distance, a degree of longitude is different in length at different latitudes. In practice, this distance is usually given in nautical miles.

`dist = departure(long1,long2,lat)` returns the departure between two longitudes at a given latitude in degrees. Departure is dimensionless; the shorter of the two direction is taken from the first longitude to the second. The distance is given in degrees of arc length.

`dist = departure(long1,long2,lat,units)` specifies the valid angle units string to apply to the latitude, longitudes, and output distance.

`dist = departure(long1,long2,lat,geoid)` specifies the elliptical definition of the Earth to be used with the two-element `geoid` vector. The default geoid model is a unit sphere, which is sufficient for most applications. When a geoid model is input, the resulting distance is given in terms of the distance units in the `geoid` vector, regardless of the angle units used.

Examples

On a spherical Earth, the departure is proportional to the cosine of the latitude:

```
distance = departure(0,10,0)
distance =
    10

distance = departure(0,10,60)
distance =
     5
```

When a geoid is used, the result is more complicated. The distance at 60° is not exactly twice the 0° value:

```
distance = departure(0,10,0,almanac('earth','geoid','nm'))
distance =
  601.0772

distance = departure(0,10,60,almanac('earth','geoid','nm'))
distance =
  299.7819
```

See Also

`distance`	Distance between two points
`stdm`	Standard deviation for geographic data

displaym

Purpose

Project data in a geographic data structure

Syntax

```
displaym(mstruct)
h = displaym(mstruct)
```

Description

`displaym(mstruct)` will project the data contained in the input structure onto the current axes. The current axes must have a valid map definition. The input mstruct must be a valid Mapping Toolbox Geographic Data Structure.

`displaym(mstruct,'object')` will display vector data from entries in the Mapping Toolbox Geographic Data structure whose tags begin with the 'object' string. The output vectors use NaNs to separate the individual entries in the map structure. Matches of the tag string must be vector data (lines and patches) to be included in the output. The search is not case-sensitive.

`displaym(mstruct,objects)` where objects is a character array or a cell array or strings, allows more than one object to be the basis for the search. Character array objects will have trailing spaces stripped before matching.

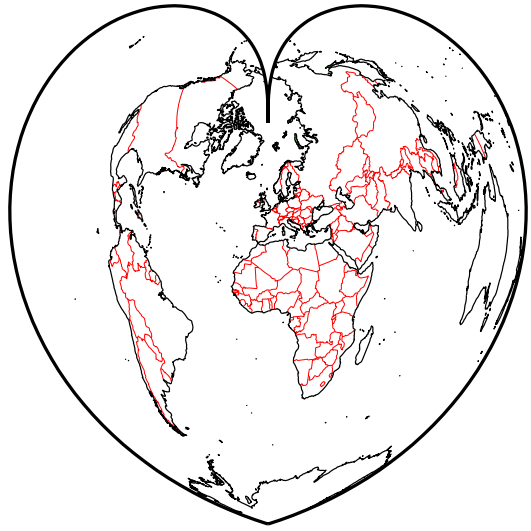
`[lat,lon] = displaym(mstruct,objects,'exact')` requires an exact match to extract data.

`h = displaym(mstruct,...)` returns the handles to the objects projected.

Examples

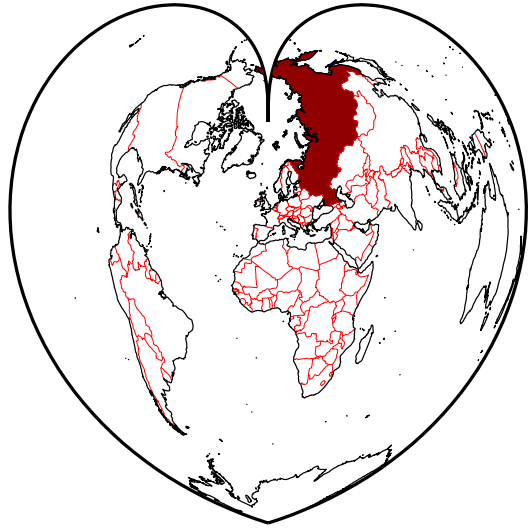
Display political and coast lines contained in structure format:

```
axesm werner; framem
displaym(POline)
```



Highlight Russia by displaying a patch map of the country:

```
displaym(POpatch,'russia')
```



Remarks

A Mapping Toolbox geographic data structure is a MATLAB structure that may contain line, patch, text, regular matrix map, general matrix map, and light objects.

Object properties used in the display are taken from the `otherproperty` field of the structure. If a line or patch object's `otherproperty` field is empty, `displaym` uses default colors. A patch is assigned an index into the current colormap based on the structure's `tag` field. Lines are assigned colors from the current color order according to their tags.

See Also

`extractm`	Extract vector data from geographic data structures
`geographic data structure`	Specially formatted structure for map data
`mlayers`	GUI for manipulating layers of a geographic data structure

Purpose

Convert distance values to strings

Syntax

```
str = dist2str(distin)
str = dist2str(distin,format)
str = dist2str(distin,format,units)
str = dist2str(distin,format,digits)
str = dist2str(distin,format,units,digits)
```

Description

The purpose of this command is to make distance-valued variables into strings suitable for map display.

`str = dist2str(distin)` converts the input vector of distances, `distin`, to a string matrix.

`str = dist2str(distin,format)` uses the *format* string to specify the notation to be used for the string matrix. The default, `'none'`, results in simple numerical representation (no indicator for positive distances, minus signs for negative distances); `'pm'` (for *plus-minus*) prefixes a '+' for positive distances.

`str = dist2str(distin,format,units)` uses the input *units* to define the units in which the input distances are supplied. *units* is any valid distance string (`'kilometers'` are the default). *units* also determines which unit symbol to suffix to the strings.

`str = dist2str(distin,format,units,digits)` determines how many digits to display. `digits` is the power of 10 representing the last place of significance in the resulting output. For example, if `digits` = 2, the *hundreds* slot of will be its last significant figure. In general, the 10^{digits} slot will be the last significant figure, rounded appropriately depending upon the value in the $10^{digits-1}$ slot. `digits` is –2 by default.

Examples

Create a vector of values and convert to strings:

```
d = [-3.7 2.95 87]
str = dist2str(d,'none','km')
str =
-3.70 km
 2.95 km
87.00 km
```

Now change the units, add '+'s to positive values, and truncate to the tenths (10^{-1}) slot:

```
str = dist2str(d,'pm','sm',-1)
str =
 -3.7 mi
 +3.0 mi
+87.0 mi
```

See Also

`angl2str`	Convert angle values to strings
`distdim`	Convert distance units
`time2str`	Convert time values to strings

Purpose

Compute distance between two points on the globe

Syntax

```
dist = distance(pt1,pt2)
dist = distance(pt1,pt2,geoid)
dist = distance(pt1,pt2,units)
dist = distance(pt1,pt2,geoid,units)

dist = distance(track,pt1,...)

dist = distance(lat1,lon1,lat2,lon2)
dist = distance(lat1,lon1,lat2,lon2,geoid)
dist = distance(lat1,lon1,lat2,lon2,units)
dist = distance(lat1,lon1,lat2,lon2,geoid,units)

dist = distance(track,lat1,...)
```

Background

Distance between two points can be calculated in two manners. For great circles, the distance is the shortest surface distance between two points. For rhumb lines, the distance is measured along the rhumb line passing through the two points, which is not, in general, the shortest surface distance between them. For more information on this distinction, see the "Geographic Measurement" section of Chapter 1, "Mapping Fundamentals," in the *Mapping Toolbox User's Guide*.

Description

`dist = distance(pt1,pt2)` calculates the great circle distance from `pt1` to `pt2`. These two-column matrices should be of the form `[latitude longitude]`. The resulting distance is returned in terms of angle units of arc length (degrees by default).

`dist = distance(lat1,lon1,lat2,lon2)` performs the same calculation for two pairs of latitude and longitude matrices.

`dist = distance(pt1,pt2,geoid)` specifies the elliptical definition of the Earth to be used with the two-element `geoid` vector. The default geoid model is a unit sphere, which is sufficient for most applications. When a geoid is input, the resulting distance is given in terms of the distance units used in the `geoid` vector.

`dist = distance(pt1,pt2,`*`units`*`)` specifies the standard angle unit string. The default value is `'degrees'`. These units are also the distance units of the result (e.g., degrees of arc length) unless a `geoid` vector is specified.

`dist = distance(`*`track`*`,pt1,...)` specifies whether great circle distances or rhumb line distances are desired. Great circle distances, the default, are indicated with the standard *track* string `'gc'`. Rhumb line distances are indicated with the standard *track* string `'rh'`.

Examples

Imagine a trip from Norfolk, Virginia (37°N,76°W), to Cape St. Vincent, Portugal (37°N,9°W), just outside the Straits of Gibraltar. The distance between these two points depends upon the *track* string selected. Using the `pt1,pt2` notation, the two cases result in:

```
dist = distance('gc',[37,-76],[37,-9])
dist =
   52.3094

dist = distance('rh',[37,-76],[37,-9])
dist =
    53.5086
```

The difference between these two tracks is 1.992 degrees, or about 72 nautical miles. This represents about 2% of the total trip distance. The trade-off is that at the cost of those 72 miles, the entire trip can be made on a course of 090°, due east, while in order to follow the great circle path, the course must be changed continuously.

When a great circle and rhumb line coincide, the distances are the same. Using two points on the same meridian, this time in the `lat1,lon1,lat2,lon2` notation:

```
dist = distance(37,-76,67,-76) % great circle sense
dist =
     30.0000

dist = distance('rh',37,-76,67,-76)
dist =
     30.0000
```

The distances are the same, about 1800 nautical miles (there are about 60 nautical miles in a degree of arc length).

See Also

`azimuth`	Azimuth between two points on the globe
`distdim`	Convert distance units
`reckon`	New point with an azimuth and distance

`track`
`track1`
`track2`
	Trace paths on the globe

distortcalc

Purpose

Calculate distortion parameters for a map projection

Syntax

```
areascale = distortcalc(lat,long)
areascale = distortcalc(mstruct,lat,long)
[areascale,angdef,maxscale,minscale,merscale,parscale] =
  distortcalc(...)
```

Background

Map projections inevitably introduce distortions in the shape and size of objects as they are transformed from three-dimensional spherical coordinates to two-dimensional cartesian coordinates. The amount and type of distortion varies between projections, over the projection, and with the selection of projection parameters such as standard parallels. This function allows a quantitative evaluation of distortion parameters.

Description

`areascale = distortcalc(lat,long)` computes the area distortion for the current map projection at the specified geographic location. An area scale of 1 indicates no scale distortion. Latitude and longitude may be scalars, vectors or matrices in the angle units of the defined map projection.

`areascale = distortcalc(mstruct,lat,long)` uses the projection defined in the map structure `mstruct`.

`[areascale,angdef,maxscale,minscale,merscale,parscale] = distortcalc(...)` computes the area scale, maximum angular deformation of right angles (in the angle units of the defined projection), the particular maximum and minimum scale distortions in any direction, and the particular scale along the meridian and parallel. `distortcalc` may also be called with fewer output arguments, in the order shown.

Examples

At the equator, the Mercator projection is free of both area and angular distortion:

```
axesm mercator
[areascale,angdef] = distortcalc(0,0)
areascale =
      0.99825
angdef =
      0.10009
```

At 60 degrees north, objects are shown at 400% of their true area. The projection is conformal, so angular distortion is still zero.

```
[areascale,angdef] = distortcalc(60,0)
areascale =
       3.9965
angdef =
     0.050518
```

Remarks

This function uses a finite difference technique. The geographic coordinates are perturbed slightly in different directions and projected. A small amount of error is introduced by numerical computation of derivatives and the variation of map distortion parameters.

See Also

`mdistort`	Contours of constant distortion on a map
`tissot`	Graphic despiction of distortion characteristics

distdim

Purpose

Convert distances between different units

Syntax

```
distout = distdim(distin,from,to)
distout = distdim(distin,from,to,radius)
```

Description

`distout = distdim(distin,from,to)` returns the value of the input distance `distin`, which is in units specified by the valid distance units string *from*, in the desired units given by the valid distance units string *to*. Valid distance units strings are:

```
'kilometers' or 'km' for kilometers
'meters' or 'm' for meters
'nauticalmiles' or 'nm' for nautical miles
'statutemiles' or 'sm' for statute miles
'feet' or 'ft' for feet
'degrees' or 'deg' for degrees (arc length)
'radians' or 'rad' for radians (arc length)
```

`distout = distdim(distin,from,to,radius)` specifies the radius of a sphere to use when one of *from* or *to* is an unit string associated with arc length (radians or degrees). A degree of arc length covers more kilometers, for example, on Jupiter than it would on the Earth. You can enter the radius as a number (the radius of the sphere in the non arc length units), as a call to the `almanac` function (e.g., `almanac('jupiter','radius','units')`), again in the appropriate units, or as a string planet name (e.g., `'mars'`), and the function will make the appropriate call to the `almanac` function. The radius of the Earth is the default.

Remarks

Distance is expressed in one of two general forms: as a linear measure in some unit (kilometers, miles, etc.) or as angular arc length (degrees or radians). While the use of linear units is generally understood, angular arc length is not necessarily as clear. The conversion from angular units to linear units for the arc along any circle is the angle in radians multiplied by the radius of the circle. On the sphere, this means that radians of latitude are directly translatable to kilometers, say, by multiplying by the radius of the Earth in kilometers (about 6371 km). However, the linear distance associated with radians of longitude changes with latitude; the radius in question is then not the radius of the Earth, but the (chord) radius of the small circle defining that parallel. In the Mapping Toolbox, the angle in radians or degrees associated with any distance is the arc length of a great circle passing through the points of interest.

Therefore, the radius in question always refers to the radius of the relevant sphere, consistent with the distance command.

Examples

Convert 100 kilometers to nautical miles:

```
distkm = 100
distkm =
   100
distnm = distdim(distkm,'kilometers','nauticalmiles')
distnm =
   53.9957
```

A degree of arc length is about 60 nautical miles:

```
distnm = distdim(1,'deg','nm')
distnm =
   60.0405
```

This is not accidental. It is the original definition of the nautical mile. Naturally, this assumption does not hold on other planets:

```
distnm = distdim(1,'deg','nm','mars')
distnm =
   31.9474
```

See Also

almanac	Planetary data
angledim	Convert angle units
deg2km sm2nm nm2rad	Direct distance conversion functions
dist2str	Convert distances to display strings
distance	Distance between points
timedim	Convert time units

dms2deg, dms2rad

Purpose

Convert angle units from *dms* format to degrees or radians

Syntax

```
anglout = dms2deg(anglin)
angleout = dms2rad(anglin)
```

Description

`anglout = dms2deg(anglin)` converts angles input in degrees-minutes-seconds (*dms*) format to the equivalent measure in decimal degrees.

`anglout = dms2rad(anglin)` converts angles input in degrees-minutes-seconds (*dms*) format to the equivalent measure in radians.

Remarks

The inputs can be in degrees-minutes (*dm*) format, since numerically they look like *dms* format in which seconds are always zero.

Example

```
dms2deg(430.00)
ans =
          4.50
```

See Also

`angledim`	Convert angle units
`deg2rad` `dms2rad`	Other direct angle conversion functions
`dms2dm`	Round *dms* format angles to *dm* format
`dms2mat`	Convert from *dms* to separated matrix components
`mat2dms`	Convert from separated matrices to *dms* format

Purpose

Convert the elements of *dms* format to distinct matrix elements

Syntax

```
[d,m,s] = dms2mat(anglin)
[d,m,s] = dms2mat(anglin,n)
matout = dms2mat(anglin,n)
```

Description

`[d,m,s] = dms2mat(anglin)` takes angles input in *dms* inputs and splits their components into three outputs, one each for degrees, minutes and seconds.

`[d,m,s] = dms2mat(anglin,n)` specifies the power of 10, `n`, to which the resulting seconds output should be rounded (i.e., if a result is 12.567 seconds, and `n` = -2, the resulting seconds output would be 12.57). The default value of `n` is -5.

`matout = dms2mat(anglin,n)` returns a three-column matrix, `matout`, in which the columns represent degrees, minutes, and seconds, respectively. In this case, `anglin` must be a vector.

Examples

```
anglin = [12547.34; 54323.17];

[d,m,s] = dms2mat(anglin)
d =
   125
   543
m =
    47
    23
s =
    34
    17

matout = dms2mat(anglin)
matout =
   125    47    34
   543    23    17
```

See Also

`mat2dms` Convert from separated matrices to *dms* format

dms2dm

Purpose

Round from *dms* format to *dm* format

Syntax

```
anglout = dms2dm(anglin)
```

Description

`anglout = dms2deg(anglin)` rounds angles input in degrees-minutes-seconds (*dms*) format to the appropriate value in degrees-minutes (*dm*) format. This special handling is needed because there are 60, and not 100, seconds in a minute.

Example

Round 4°45'29” and 4°45'31" to *dm* format:

```
dms2dm(445.29)
ans =
        445.00

dms2dm(445.31)
ans =
        446.00
```

See Also

`angledim`	Convert angle units
`deg2rad` `dms2rad`	Other direct angle conversion functions
`dms2dm`	Round *dms* format angles to *dm* format
`dms2mat`	Convert from *dms* to separated matrix components
`mat2dms`	Convert from separated matrices input to *dms*

Purpose

Compute dead reckoning positions for a track

Syntax

```
[drlat,drlong,drtime] = dreckon(waypoints,time,speed)
[drlat,drlong,drtime] = dreckon(waypoints,time,speed,spdtimes)
```

Background

This is a navigational function. As such, it assumes that all latitudes and longitudes are in degrees, all distances are in nautical miles, all times are in hours, and all speeds are in knots, i.e., nautical miles per hour.

Dead reckoning is an estimation of position at various times based on courses, speeds, and times elapsed from the last certain position, or fix. In navigational practice, a dead reckoning position, or DR, must be plotted at every course change, every speed change, and at every hour, on the hour. Navigators also DR at other times that are not relevant to this function.

Often in practice, when two events occur that require DRs within a very short time, only one DR will be generated. This function mimics that practice by setting a tolerance of 3 minutes (0.05 hours). No two DRs will fall closer than that.

Refer to the "Navigation" section of Chapter 5 in the *Mapping Toolbox User's Guide* for further information.

Description

`[drlat,drlong,drtime] = dreckon(waypoints,time,speed)` returns the positions and times of required dead reckoning (DR) points for the input track which starts at the input time. The track should be in navigational track format (two-columns, latitude then longitude, in order of traversal. These waypoints are the starting and ending points of each leg of the track. There is one fewer track leg than waypoints, as the last point included is the end of the track. In navigation, the first waypoint would be a navigational fix, taken at `time`. The `speed` input can be a scalar, in which case a constant speed is used throughout, or it can be a vector in which one speed is given for each track leg (i.e., speed changes coincide with course changes).

`[drlat,drlong,drtime] = dreckon(waypoints,time,speed,spdtimes)` allows speed changes to occur independent of course changes. The elements of the `speed` vector must have a one-to-one correspondence with the elements of the `spdtimes` vector. This latter variable consists of the time interval after `time` at which each speed order *ends*. For example, if `time` is 6.75, and the first element of `spdtimes` is 1.35, then the first `speed` element is in effect from 6.75 to 8.1 hours. When this syntax is used, the last output DR is the *earlier* of the final `spdtimes` time or the final `waypoints` point.

Examples

Assume that a navigator gets a fix at noon, 1200Z, which is (10.3°N, 34.67°W). He's in a hurry to make a 1330Z rendezvous with another ship at (9.9°N, 34.5°W), so he plans on a speed of 25 knots. After the rendezvous, both ships head for (0°, 37°W). The engineer wants to take an engine off line for maintenance at 1430Z, so at that time, speed must be reduced to 15 knots. At 1530Z, the maintenance will be done. Determine the DR points up to the end of the maintenance.

```
waypoints = [10.1 -34.6; 9.9 -34.5; 0 -37]
waypoints =
   10.1000  -34.6000       % Fix at noon
    9.9000  -34.5000       % Rendezvous point
         0  -37.0000       % Ultimate destination
speed = [25; 15];
spdtimes = [2.5; 3.5];     % Elapsed times after fix
noon = 12;
[drlat,drlong,drtime] = dreckon(waypoints,noon,speed,spdtimes);
[drlat,drlong,drtime]
ans =
    9.8999  -34.4999   12.5354  % Course change at waypoint
    9.7121  -34.5478   13.0000  % On the hour
    9.3080  -34.6508   14.0000  % On the hour
    9.1060  -34.7022   14.5000  % Speed change to 15 kts
    8.9847  -34.7330   15.0000  % On the hour
    8.8635  -34.7639   15.5000  % Stop at final spdtime, last
                                 % waypoint has not been reached
```

See Also

`legs`	Find courses and distances between waypoints
`navfix`	Mercator-based navigational fixing
`track`	Connect navigational waypoints with track segments

Purpose

Wraps longitudes to values east of a meridian

Syntax

```
ang = eastof(angin,meridian)
ang = eastof(angin,meridian,units)
```

Description

`ang = eastof(angin,meridian)` transforms input angles into equivalent angles east of the specified meridian.

`ang = eastof(angin,meridian,units)` uses the units defined by the input string *units*. If omitted, default units of 'degrees' are assumed.

Example

```
eastof(1,360)
ans =
   361
```

Remarks

This function can be used to prepare vector data for use with regular matrix maps. Regular matrix maps use geographic locations that are strictly east of the left edge of the map.

See Also

`westof`	Wraps longitudes to values west of a meridian
`zero22pi`	Truncates angles into the 0 deg to 360 deg range
`npi2pi`	Truncates angles into the –180 deg to 180 deg range
`smoothlong`	Removes discontinuities for longitude data
`angledim`	Converts angles from one unit system to another

ecc2flat

Purpose

Convert from eccentricity to flattening representation of the ellipsoid

Syntax

```
flattening = ecc2flat(eccentricity)
```

Description

Flattening and eccentricity are two methods of defining an ellipsoid.

`flattening = ecc2flat(eccentricity)` returns the equivalent flattening for the input eccentricities. If the input, `eccentricity`, is a two-column vector, only the second column is used. This allows the standard two-element geoid vectors to be used as rows of the input, since the second element of these vectors is the eccentricity. In all other cases, all columns of the input are used.

Example

```
flattening = ecc2flat(almanac('earth','geoid'))
flattening =
    0.0034
```

See Also

`almanac`	Planetary data
`ecc2n` `majaxis`	Other ellipsoid functions
`flat2ecc`	Convert from flattening to eccentricity

Purpose

Convert from eccentricity to the *n* representation of the ellipsoid

Syntax

```
n = ecc2n(eccentricity)
```

Description

Eccentricity and the parameter *n* are two methods of defining an ellipsoid. The definition of *n* is:

(semimajor axis – semiminor axis)/(semimajor axis + semiminor axis).

`n = ecc2n(eccentricity)` returns the equivalent *n* for the input eccentricities. If the input, `eccentricity`, is a two-column vector, only the second column is used. This allows the standard two-element geoid vectors to be used as rows of the input, since the second element of these vectors is the eccentricity. In all other cases, all columns of the input are used.

Example

```
n = ecc2n(almanac('earth','geoid'))
n =
   0.00167922039463
```

See Also

`almanac`	Planetary data
`ecc2flat` `majaxis`	Other ellipsoid functions
`n2ecc`	Convert from *n* to eccentricity

ellipse1

Purpose

Geographic ellipse defined by its center, semimajor axes, eccentricity and azimuth

Syntax

```
[lat,lon] = ellipse1(lat0,lon0,ellipse)
[lat,lon] = ellipse1(lat0,lon0,ellipse,offset)
[lat,lon] = ellipse1(lat0,lon0,ellipse,offset,az)
[lat,lon] = ellipse1(lat0,lon0,ellipse,offset,az,geoid)
[lat,lon] = ellipse1(lat0,lon0,ellipse,offset,units),
[lat,lon] = ellipse1(lat0,lon0,ellipse,offset,az,units)
[lat,lon] = ellipse1(lat0,lon0,ellipse,offset,az,geoid,units)
[lat,lon] = ellipse1(lat0,lon0,ellipse,offset,az,geoid,units,npts)
[lat,lon] = ellipse1(track,...)
mat = ellipse1(...)
```

Description

`[lat,lon] = ellipse1(lat0,lon0,ellipse)` computes ellipses with a center at the point `lat0`, `lon0`. The ellipse is defined by the third input which of the form `[semimajor-axis, eccentricity]`. The `lat0`, `lon0` inputs can be scalar or column vectors. The eccentricity input can be a 2-element row vector or a 2-column matrix. The ellipse input must have the same number of rows as the input `lat0` and `lon0`. The input semimajor axis is in degrees of arc length on a sphere. All ellipses are oriented so that their semimajor axis lies due north.

`[lat,lon] = ellipse1(lat0,lon0,ellipse,offset)` computes the ellipses where the semimajor axis is rotated from due north by an azimuth offset. The `offset` angle is measure clockwise from due north. If `offset=[]`, then no offset is assumed.

`[lat,lon] = ellipse1(lat0,lon0,ellipse,offset,az)` uses the input `az` to define the ellipse arcs computed. The arc azimuths are measured clockwise from due north. If `az` is a column vector, then the arc length is computed from due north. If `az` is a two column matrix, then the ellipse arcs are computed starting at the azimuth in the first column and ending at the azimuth in the second column. If `az=[]`, then a complete ellipse is computed.

`[lat,lon] = ellipse1(lat0,lon0,ellipse,offset,az,geoid)` computes the ellipse on the ellipsoid defined by the input `geoid` vector, of the form `[semimajor-axis, eccentricity]`. If omitted, the unit sphere, `geoid=[1 0]`, is assumed. When a geoid is supplied, the input semimajor axis must be in the same units as the geoid semimajor axes. In this calling form, the units of the ellipse semimajor axis are not assumed to be in degrees.

`[lat,lon] = ellipse1(lat0,lon0,ellipse,offset,`*`units`*`)`, `[lat,lon] = ellipse1(lat0,lon0,ellipse,offset,az,`*`units`*`)`, and `[lat,lon] = ellipse1(lat0,lon0,ellipse,offset,az,geoid,`*`units`*`)` are all valid calling forms, which use the input *units* to define the angle units of the inputs and outputs. If omitted, 'degrees' are assumed.

`[lat,lon] = ellipse1(lat0,lon0,ellipse,offset,az,geoid,`*`units`*`,npts)` uses the input `npts` to determine the number of points per ellipse computed. The input `npts` is a scalar, and if omitted, `npts=100`.

`[lat,lon] = ellipse1(`*`track`*`,...)` uses the *track* string to define either a great circle or rhumb line distances from the ellipse center. If *track* = 'gc', then great circle distances are computed. If *track* = 'rh', then rhumb line distances are computed. If omitted, 'gc' is assumed.

`mat = ellipse1(...)` returns a single output argument where `mat=[lat lon]`. This is useful if only one ellipse is computed.

Example

Create and plot the small ellipse centered at (0°,0°), with a semimajor axis of 10° and a semiminor axis of 5°.

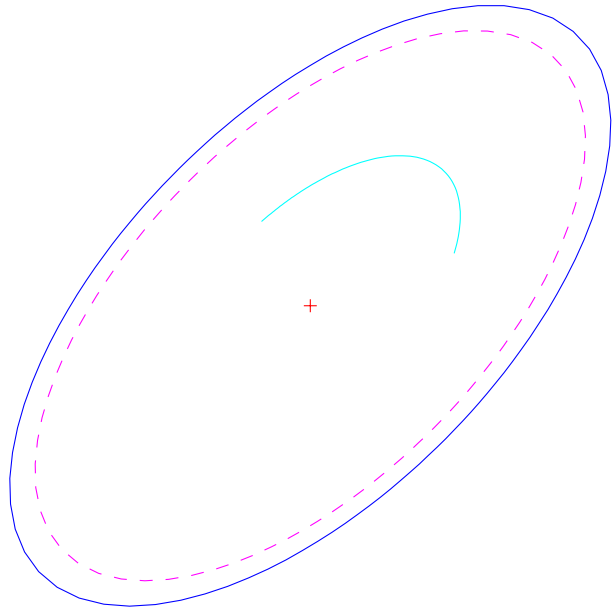
```
axesm mercator
ecc = axes2ecc(10,5);
plotm(0,0,'r+')
[elat,elon] = ellipse1(0,0,[10 ecc],45);
plotm(elat,elon)
```

If the desired radius is known in some non-angular distance unit, use the radius returned by the almanac function as the geoid input to set the range units (use an empty azimuth entry to specify a full ellipse).

```
earthradius = almanac('earth','radius','nm');
[elat,elon] = ellipse1(0,0,[550 ecc],45,[],earthradius);
plotm(elat,elon,'m--')
```

For just an arc of the ellipse, enter an azimuth range:

```
[elat,elon] = ellipse1(0,0,[5 ecc],45,[-30 70]);
plotm(elat,elon,'c-')
```



Remarks

This function extends the concept of the small circle, which is the locus of all points at an equal surface distance, to a 'small ellipse'. The small ellipse is constructed by computing the locus of points for which the distance from the center point varies as the parametric description of the ellipse.

Multiple circles can be defined from a single starting point by providing scalar lat0, lon0 inputs and a 2 column matrix for the ellipse definitions.

See Also

`scircle1`	Small circle defined by its center, range and azimuth
`track1`	Track lines defined by starting point, azimuth and range
`axes2ecc`	Computes eccentricity given semimajor, semiminor axes

Purpose

Fill in regions of indexed matrix maps with specified values

Syntax

```
newmap = encodem(map,seedmat)
newmap = encodem(map,seedmat,stopvals)
```

Description

This command *fills in* regions of matrix maps with desired values. If a *boundary* exists, the new value will replace all entries in all four directions until the boundary is reached. The boundary is made up of selected stopping values and the edges of the matrix. The new value will try to flood the region exhaustively, stopping only when no new spaces can be reached by moving up, down, left, or right without hitting a stopping value.

`newmap = encodem(map,seedmat)` fills in regions of the input matrix map, `map`, with desired new values. The boundary consists of the edges of the matrix and any entries with the value "1." The *seeds*, or starting points, and the values associated with them, are specified by the three-column matrix `seedmat`, the rows of which have the form `[row column value]`.

`newmap = encodem(map,seedmat,stopvals)` allows you to specify a vector, `stopvals`, of stopping values. Any value that is an element of `stopvals` will act as a boundary.

Examples

For this imaginary map, fill in the upper right region with 7s and the lower left region with 3s:

```
map = eye(4)
map =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1

newmap = encodem(map,[4,1,3; 1,4,7])
newmap =
     1     7     7     7
     3     1     7     7
     3     3     1     7
     3     3     3     1
```

See Also

`getseeds`	Interactively create seed matrix
`imbedm`	Encode data points into a regular matrix map
`maskm`	Replace entries of a matrix map specified by a mask matrix

Purpose

Show map precision

Syntax

```
epsm
epsm(units)
```

Description

`epsm` is the limit of map angular precision. It is useful in avoiding trigonometric singularities, among other things.

`epsm(units)` returns the same angle in units corresponding to any valid angle units string. The default is `'degrees'`.

Examples

The value of `epsm` is 10^{-6} degrees. To put this in perspective, in terms of an angular arc length, the distance is:

```
epsmkm = deg2km(epsm)
epsmkm =
   1.1119e-04      % kilometers
```

This is about 11 centimeters, a very small distance on a global scale.

See Also

`roundn` Significant figures

eqa2grn

Purpose

Convert from equal area to Greenwich coordinates

Syntax

```
[lat,lon] = eqa2grn(x,y)
[lat,lon] = eqa2grn(x,y,origin)
[lat,lon] = eqa2grn(x,y,origin,geoid)
[lat,lon] = eqa2grn(x,y,origin,units)
[lat,lon] = eqa2grn(x,y,origin,geoid,units)
mat = eqa2grn(x,y,origin...)
```

Description

This command converts data from equal-area *x*-*y*-coordinates to Greenwich (latitude-longitude) coordinates. The opposite conversion can be performed with `grn2eqa`.

`[lat,lon] = eqa2grn(x,y)` converts the equal-area coordinate points `x` and `y` to the Greenwich coordinates `lat` and `lon`.

`[lat,lon] = eqa2grn(x,y,origin)` specifies the location in the Greenwich system of the *x*-*y* origin (0,0). The two-element vector `origin` must be of the form `[latitude longitude]`. The default places the origin at the Greenwich coordinates (0°,0°).

`[lat,lon] = eqa2grn(x,y,origin,geoid)` specifies the two-element geoid vector describing the ellipsoidal model of the figure of the Earth. The `geoid` is spherical by default.

`[lat,lon] = eqa2grn(x,y,origin,`*`units`*`)` specifies the units for the outputs, where *units* is any valid angle units string. The default value is `'degrees'`.

`mat = eqa2grn(x,y,origin...)` packs the outputs into a single variable.

Examples

```
[lat,lon] = eqa2grn(.5,.5)
lat =
   30.0000
lon =
   28.6479
```

See Also

`grn2eqa`	Convert Greenwich coordinates to equal-area coordinates
`hista`	Equal area histogram

Purpose

Extract vector data from geographic data structures

Syntax

```
[lat,lon] = extractm(gstruct,object)
[lat,lon] = extractm(gstruct,objects)
[lat,lon] = extractm(gstruct,objects,'exact')
[lat,lon,indx] = extractm(...)
mat = extractm(...)
```

Description

`[lat,lon] = extractm(gstruct,object)` will extract vector data from those entries in the Mapping Toolbox geographic data structure that have tags beginning with the *object* string. The output vectors use `NaN`s to separate the entries in the map structure. Matches of the tag string must be vector data (lines and patches) to be included in the output.

`[lat,lon] = extractm(gstruct,objects)`, where `objects` is a character array, allows more than one object to be the basis for the search.

`[lat,lon] = extractm(gstruct,objects,'exact')` requires an exact match to extract data.

`[lat,lon,indx] = extractm(gstruct)` extracts all vector data from the input map structure.

`[lat,lon,indx] = extractm(...)` also returns the vector `indx` identifying the entries in the structure which meet the selection criteria.

`mat = extractm(...)` returns the vector data in a single, two-column matrix, in which the first column contains latitudes and the second column, longitudes.

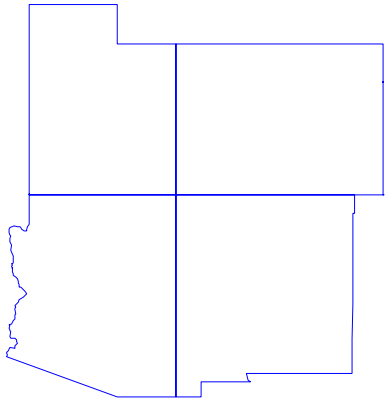
Examples

Extract the District of Columbia from the low resolution U.S. vector data:

```
load usalo
extractm(state,'district of columb')
ans =
   38.9000  -77.0700
   38.9000  -77.0500
   38.9000  -77.0700
   38.8700  -77.0200
   38.8000  -77.0200
   38.7800  -77.0300
   38.9000  -76.9000
   39.0000  -77.0300
   38.9500  -77.1200
   38.9000  -77.0700
```

Extract the states that meet at the Four Corners and plot the extracted data:

```
states4 = strvcat('colo','new mex','ariz','utah');
[lat,long,indx] = extractm(state,states4);

axesm mercator
patchm(lat,long,'g')
```



Remarks

A Mapping Toolbox geographic data structure is a MATLAB structure that may contain line, patch, text, regular matrix map, general matrix map, and light objects.

See Also

`displaym`	Project data in a geographic data structure
`geographic data structure`	Specially formatted structure for map data
`mlayers`	GUI for manipulating layers of a map

fill3m

Purpose

Project 3-D patch objects onto the current map axes

Syntax

```
h = fill3m(lat,lon,z,cdata)
h = fill3m(lat,lon,z,PropertyName,PropertyValue,...)
```

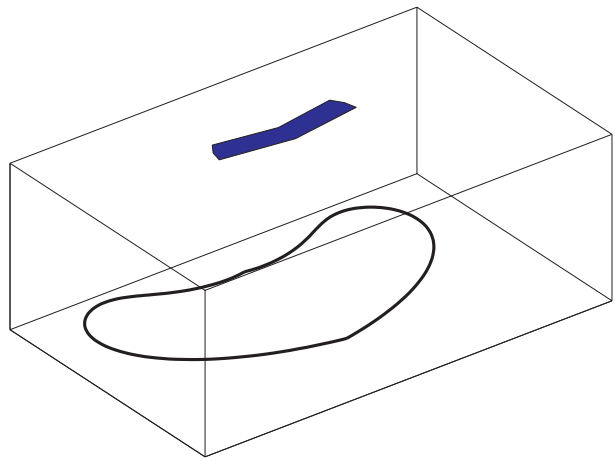
Description

`h = fill3m(lat,lon,z,cdata)` projects and displays any patch object with vertices defined by vectors `lat` and `lon` to the current map axes. The scalar `z` indicates the altitude plane at which the patch will be displayed. The input `cdata` defines the patch face color. The patch handle or handles, `h`, can be returned.

`h = fill3m(lat,lon,z,PropertyName,PropertyValue,...)` allows any property/value pair supported by `patch` to be assigned to the `fill3m` object.

Examples

```
lat = [30 15 0 0 0 15 30 30]';
lon = [-60 -60 -60 0 60 60 60 0]';
axesm bonne; framem
view(3)
fill3m(lat,lon,2,'b')
```



See Also

`fillm`	Project 2-D patch objects onto the current map axes
`patchesm`	Project multiple patch objects more rapidly
`patchm`	Project and display patch objects on the current map axes

fillm

Purpose

Project 2-D patch objects onto the current map axes

Syntax

```
h = fillm(lat,lon,cdata)
h = fillm(lat,lon,'PropertyName',PropertyValue,...)
```

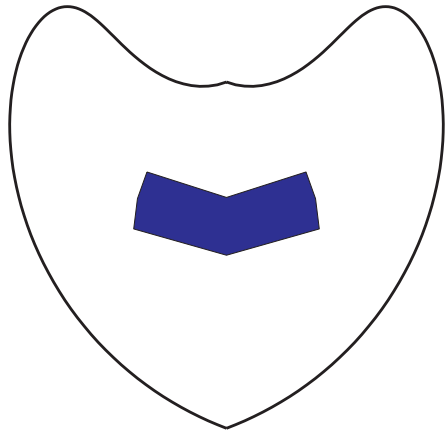
Description

`h = fillm(lat,lon,cdata)` projects and displays any patch object with vertices defined by the vectors `lat` and `lon` to the current map axes. The input `cdata` defines the patch face color. The patch handle or handles, `h`, can be returned.

`h = fillm(lat,lon,'PropertyName',PropertyValue,...)` allows any property/value pair supported by `patch` to be assigned to the `fillm` object.

Examples

```
lat = [30 15 0 0 0 15 30 30]';
lon = [-60 -60 -60 0 60 60 60 0]';
axesm bonne; framem
fillm(lat,lon,'b')
```



See Also

`fill3m`	Project 3-D patch objects onto the current map axes
`patchesm`	Project multiple patch objects more rapidly
`patchm`	Project and display patch objects on the current map axes

Purpose

Filter geographic datasets

Syntax

```
[newlat,newlong] = filterm(lat,long,map,maplegend,allowed)
```

Description

`[newlat,newlong] = filterm(lat,long,map,maplegend,allowed)` filters geographic data based upon the corresponding entries of a regular matrix map, `map`, with a three-element map legend vector `maplegend`. The data locations to be filtered is input in the vectors `lat` and `lon`. For those locations corresponding to entries of `map` equal to one of the values contained in the vector `allowed`, an output location is returned in `newlat` and `newlon`. Those locations not corresponding to such entries of `map` are not returned in the outputs.

Examples

Filter a random set of 100 geographic points. Use the `topo` map for starters:

```
load topo
```

Then, generate 100 random points:

```
lat = -90+180*rand(100,1);
long = -180+360*rand(100,1);
```

Make a land map, which is "`1`" where `topo>0` elevation:

```
land = topo>0;
[newlat,newlong] = filterm(lat,long,land,topolegend,1);
size(newlat)
ans =
    15     1
```

15 of the 100 random points fell on *land*.

See Also

`hista`	Spatial equal area histogram
`histr`	Spatial equirectangular histogram

findm

Purpose

Find latitude and longitude coordinates for nonzero map entries

Syntax

```
[lat,lon] = findm(map,maplegend)
[lat,lon,val] = findm(map,maplegend)
[lat,lon,val] = findm(latin,lonin,map)
mat = findm(...)
```

Description

This function works in two modes: with a regular matrix restriction and without.

`[lat,lon] = findm(map,maplegend)` returns latitude and longitude vectors `lat` and `lon`, which provide the locations of all nonzero entries of the regular matrix `map`, `map`, with three-element map legend vector `maplegend`.

`[lat,lon,val] = findm(map,maplegend)` also returns the values, `val`, of the matrix `map` corresponding to the `lat` and `lon` locations.

`[lat,lon,val] = findm(latin,lonin,map)` removes the *regular* matrix restriction. Two matrices, `latin` and `lonin`, the same size as `map`, must provide cell-by-cell latitude and longitude coordinates matched with the corresponding entries of `map`.

`mat = findm(...)` returns a single output `mat` of the form `[lat,lon]`.

Examples

The entered map can also be the result of a logical statement. Where is elevation greater than 5500 meters?

```
load topo
mat = findm((topo>5500),topolegend)
mat =
   34.5000   79.5000
   34.5000   80.5000
   30.5000   84.5000
   28.5000   86.5000
```

These points are in the Himalayas.

See Also

`find` Find indices and values of nonzero elements (see online *MATLAB Function Reference*)

Purpose

Convert from flattening to eccentricity representation of the ellipsoid

Syntax

```
eccentricity = flat2ecc(flattening)
```

Description

Flattening and eccentricity are two methods of defining an ellipsoid.

`eccentricity = flat2ecc(flattening)` returns the equivalent eccentricity for the input flattening. If the input, `flattening`, is a two-column vector, only the second column is used. This allows two-element vectors to be used as rows of the input, since the form [semimajor-axis, flattening] is a complete representation of an ellipsoid (but is not the standard form for geoid vectors in the Mapping Toolbox). In all other cases, all columns of the input are used.

Example

```
e = flat2ecc(0.003353)
e =
   0.08182149712026
```

This eccentricity is the default value for the Earth.

See Also

`almanac`	Planetary data
`ecc2flat`	Convert from eccentricity to flattening
`ecc2n` `majaxis`	Other ellipsoid functions

framem

Purpose

Toggle and control the display of the map frame

Syntax

```
framem
framem('on')
framem('off')
framem('reset')
framem(linespec)
framem(PropertyName,PropertyValue,...)
```

Description

`framem` toggles the visibility of the map frame by setting the map axes property `Frame` to `'on'` or `'off'`. The default setting for map axes is `'off'`.

`framem('on')` sets the map axes property `Frame` to `'on'`.

`framem('off')` sets the map axes property `Frame` to `'off'`.

When called with the string argument `'off'`, the map axes property `Frame` property is set to `'off'`.

`framem('reset')` resets the entire frame using the current properties. This is essentially a *refresh* option.

`framem(linespec)` sets the map axes `FEdgeColor` property to the color component of any *linespec* string recognized by the MATLAB `line` command.

`feamem(PropertyName,PropertyValue,...)` sets the appropriate map axes properties to the desired values. These property names and values are described on the `axesm` reference page of this guide.

Remarks

Map frame properties can be also created or altered using the `axesm` or `setm` commands.

See Also

`axesm`	Define map axes and set map properties
`setm`	Set map properties

Purpose

Convert great circles to small circle notation

Syntax

```
[centerlat,centerlong,radius] = gc2sc(lat,long,az)
[centerlat,centerlong,radius] = gc2sc(lat,long,az,units)
```

Description

Great circles are a sub-category of small circles, having a radius of 90°. Because of the computational circumstances under which these objects often arise, however, two different notations are convenient.

Great circle notation consists of a point on the great circle and the azimuth at that point along which the great circle proceeds.

Small circle notation consists of a center point and a radius in units of angular arc length.

`[centerlat,centerlong,radius] = gc2sc(lat,long,az)` returns the *small circle notation* for great circles entered in *great circle notation*.

`[centerlat,centerlong,radius] = gc2sc(lat,long,az,units)` specifies the standard angle unit string. The default value is `'degrees'`.

Examples

Given a great circle passing through (25°S,70°W) on an azimuth of 45°, how can it be represented in small circle notation?

```
[newlat,newlong,range] = gc2sc(-25,-70,45)
newlat =
  -39.8557
newlong =
   42.9098
range =
    90
```

A great circle always bisects the sphere. As a demonstration of this statement, consider the Equator, which passes through any point with a latitude of 0° and proceeds on an azimuth of 90° or 270°. In small circle notation, this is:

```
[newlat,newlong,range] = gc2sc(0,-70,270)
newlat =
    90
newlong =
 -145.9638
range =
    90
```

Not surprisingly, the small circle is centered on the North Pole. As always, at the poles, the longitude is arbitrary, due to the convergence of the meridians.

Remarks

Note that the center coordinates returned by this command always lead to one of two possibilities. Since the great circle bisects the sphere, the antipode of the returned point is also a center with a radius of 90°. In the above example, the South Pole would also be a suitable center for the Equator in small circle notation.

See Also

`antipode`	Find antipodal points
`distdim`	Distance unit conversions

`gcxgc`
`gcxsc`
`rhxrh`
`crossfix`
	Intersection functions

Purpose

Get current map structure

Syntax

```
mapstruct = gcm
mapstruct = gcm(hndl)
```

Description

`mapstruct = gcm` returns the map axes *map structure*, which contains the settings for all the current map axes properties.

`mapstruct = gcm(hndl)` specifies the map axes by axes handle.

Examples

Establish a map axes with default values, then look at the structure:

```
axesm mercator
mapstruct = gcm
mapstruct =
     mapprojection: 'mercator'
              zone: []
        angleunits: 'degrees'
            aspect: 'normal'
       fixedorient: []
             geoid: [1 0]
       maplatlimit: [-86 86]
       maplonlimit: [-180 180]
      mapparallels: 0
        nparallels: 1
            origin: [0 0 0]
     falsenorthing: 0
      falseeasting: 0
       scalefactor: 1
           trimlat: [-86 86]
           trimlon: [-180 180]
             frame: 'off'
             ffill: 100
        fedgecolor: [0 0 0]
        ffacecolor: 'none'
         flatlimit: [-86 86]
        flinewidth: 2
         flonlimit: [-180 180]
              grid: 'off'
         galtitude: Inf
            gcolor: [0 0 0]
        glinestyle: ':'
        glinewidth: 0.50000000000000
    mlineexception: []
         mlinefill: 100
        mlinelimit: []
     mlinelocation: 30
      mlinevisible: 'on'
    plineexception: []
         plinefill: 100
```

```
      plinelimit: []
   plinelocation: 15
    plinevisible: 'on'
       fontangle: 'normal'
       fontcolor: [0 0 0]
        fontname: 'helvetica'
        fontsize: 9
       fontunits: 'points'
      fontweight: 'normal'
     labelformat: 'compass'
      labelunits: 'degrees'
   labelrotation: 'off'
   meridianlabel: 'off'
  mlabellocation: 30
  mlabelparallel: 86
     mlabelround: 0
   parallellabel: 'off'
  plabellocation: 15
  plabelmeridian: -180
     plabelround: 0
```

Remarks

Map structure properties are created with the `axesm` command. They can be queried with the `getm` command and modified with the `setm` command.

See Also

`axesm`	Create map axes object
`getm`	Query map axes map structure
`setm`	Modify map axes map structure

gcpmap

Purpose

Get current mouse point from the map

Syntax

```
pt = gcpmap
pt = gcpmap(hndl)
```

Description

`pt = gcpmap` returns the current point of the current map axes in the form `[latitude longitude z-altitude]`.

`pt = gcpmap(hndl)` specifies the map axes in question by its handle.

Remarks

`gcpmap` works much like the standard MATLAB `get(gca,'CurrentPoint')`, except that the returned matrix is in `[lat lon z]`, not `[x y z]`.

See Also

`inputm` Mouse selection of position

Purpose

Find equally spaced waypoints along a great circle

Syntax

```
[lat,lon] = gcwaypts(lat1,lon1,lat2,lon2)
[lat,lon] = gcwaypts(lat1,lon1,lat2,lon2,nlegs)
pts = gcwaypts(lat1,lon1,lat2,lon2...)
```

Background

This is a navigational function. As such, it assumes that all latitudes and longitudes are in degrees.

In navigational practice, great circle paths are often approximated by rhumb line segments. This is done to come reasonably close to the shortest distance between points without requiring course changes too frequently. The `gcwaypts` command provides an easy means of finding waypoints along a great circle path that can serve as end points for rhumb line segments (track legs).

Description

`[lat,lon] = gcwaypts(lat1,lon1,lat2,lon2)` returns the coordinates of equally spaced points along a great circle path connecting two endpoints, (`lat1`, `lon1`) and (`lat2`, `lon2`).

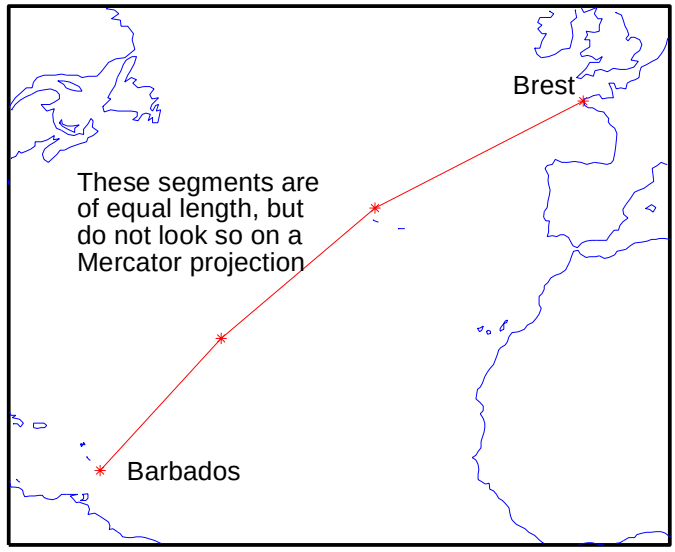
`[lat,lon] = gcwaypts(lat1,lon1,lat2,lon2,nlegs)` specifies the number of equal-length track legs to calculate. `nlegs+1` output points are returned, since a final endpoint is required. The default number of legs is 10.

`pts = gcwaypts(lat1,lon1,lat2,lon2...)` packs the outputs, which are otherwise two-column vectors, into a two-column matrix of the form `[latitude longitude]`. This format for successive waypoints along a navigational track is called *navigational track format* in this guide. See the `navigational track format` reference page in this section for more information.

Examples

Imagine you own a sailing yacht and are planning a voyage from North Point, Barbados (13.33° N,59.62°W), to Brest, France (48.33°N,4.83°W). To divide the track into three equal-length segments:

```
[l,g] = gcwaypts(13.33,-59.62,48.33,-4.83,3)
l =
   13.3300
   27.3316
   39.6250
   48.3300
g =
  -59.6200
  -45.8919
  -28.4459
   -4.8300
```



See Also

`dreckon`	Dead reckon points for a track
`legs`	Courses and distances between waypoints

`navfix`	Mercator-based navigational fixing
`navigational track format`	Successive waypoints along a track
`track`	Connect waypoints

gcxgc

Purpose

Provide intersection coordinates for pairs of great circles

Syntax

```
[newlat,newlong] = gcxgc(lat1,long1,az1,lat2,long2,az2)
[newlat,newlong] = gcxgc(lat1,long1,az1,lat2,long2,az2,units)
```

Description

For any pair of great circles, there are two possible intersection conditions: the circles are identical or they intersect exactly twice on the sphere.

Great circle notation consists of a point on the great circle and the azimuth at that point along which the great circle proceeds.

`[newlat,newlong] = gcxgc(lat1,long1,az1,lat2,long2,az2)` returns the two intersection points of pairs of great circles input in *great circle notation*.When the two great circles are identical (which is not, in general, apparent by inspection), two `NaN`s are returned instead and a warning is displayed. For multiple pairings, the inputs must be column vectors.

`[newlat,newlong] = gcxgc(lat1,long1,az1,lat2,long2,az2,units)`specifies the standard angle unit string. The default value is `'degrees'`.

Examples

Given a great circle passing through (10°N,13°E) and proceeding on an azimuth of 10°, where does it intersect with a great circle passing through (0°, 20°E), on an azimuth of –23° (i.e. 337°)?

```
[newlat,newlong] = gcxgc(10,13,10,0,20,-23)
newlat =
   14.3105  -14.3105
newlong =
   13.7838  -166.2162
```

Note that the two intersection points are always antipodes of each other. As a simple example, consider the intersection points of two meridians, which are just great circles with azimuths of 0° or 180°:

```
[newlat,newlong] = gcxgc(10,13,0,0,20,180)
newlat =
   -90    90
newlong =
   -174.4504   12.5094
```

The two meridians intersect at the North and South Poles, which is exactly correct.

See Also

`antipode`	Find antipodal points
`gc2sc`	Convert great circle to small circle notation

`scxsc`
`gcxsc`
`rhxrh crossfix`
	Other intersection functions

gcxsc

Purpose

Provide intersection coordinates for great circles paired with small circles

Syntax

```
[newlat,newlong] = gcxsc(gclat,gclong,gcaz,sclat,sclong,scrange)
[newlat,newlong] = gcxsc(gclat,gclong,gcaz,...
                        sclat,sclong,scrange,units)
```

Description

For a pairing of a great circle with a small circle, there are four possible intersection conditions: the circles are identical (possible because great circles are a subset of small circles), they do not intersect, they are tangent to each other (the small circle interior to the great circle) and hence they intersect once, or they intersect twice.

Great circle notation consists of a point on the great circle and the azimuth at that point along which the great circle proceeds.

Small circle notation consists of a center point and a radius in units of angular arc length.

`[newlat,newlong] = gcxsc(gclat,gclong,gcaz,sclat,sclong,scrange)` returns the points of intersection of a great circle in *great circle notation* followed by a small circle in *small circle notation*. For multiple pairings, the inputs must be column vectors. The results are two-column matrices with the coordinates of the intersection points. If the circles do not intersect, or are identical, two `NaN`s are returned and a warning is displayed. If the two circles are tangent, the single intersection point is repeated twice.

`[newlat,newlong] = gcxsc(...,units)` specifies the standard angle unit string. The default value is `'degrees'`.

Examples

Given a great circle passing through (43°N,0°) and proceeding on an azimuth of 10°, where does it intersect with a small circle centered at (47°N,3°E) with an arc length radius of 12°?

```
[newlat,newlong] = gcxsc(43,0,10,47,3,12)
newlat =
   35.5068   58.9143
newlong =
   -1.6159    5.4039
```

See Also

`gc2sc`	Convert great circle to small circle notation

`gcxgc`
`scxsc`
`rhxrh`
`crossfix`
	Other intersection functions

geod2aut

Purpose

Convert from geodetic to authalic latitudes

Syntax

```
lat = geod2aut(lat0)
lat = geod2aut(lat0,geoid)
lat = geod2aut(lat,units)
lat = geod2aut(lat,geoid,units)
```

Background

Longitudes have the same meaning whether the Earth is treated as a sphere or as an ellipsoid. However, the definition of latitude in the case of an ellipsoid is more complicated. Geodetic, or geographic, latitudes on an ellipsoid can be transformed to latitudes on an auxiliary sphere with certain properties.

Geodetic latitude: (also called *geographic latitude*) the angle a normal line passing through a surface point makes with the plane of the Equator.

Authalic latitude: latitudes on an auxiliary sphere that is equal in area to the ellipsoid.

Description

`lat = geod2aut(lat0)` returns the geodetic latitudes provided in `lat0` transformed to authalic latitudes, which are returned in `lat`.

`lat = geod2aut(lat0,geoid)` defines the elliptical model of the Earth, given by the geoid vector `geoid`, from which `lat0` is transformed. The default geoid is the same as the default of `almanac('earth','geoid')`, the 1980 Geodetic Reference System ellipsoid.

`lat = geod2aut(lat,geoid,units)` defines the angle units of the inputs and outputs, where *units* is any valid angle units string. The default is `'degrees'`.

Examples

```
lat = geod2aut([0 45 90])
lat =
         0   44.8717   90.0000
```

See Also

`almanac`	Planetary data
`geod2cen` `aut2geod`	Other auxiliary latitude functions

Purpose

Convert from geodetic to geocentric latitudes

Syntax

```
lat = geod2cen(lat0)
lat = geod2cen(lat0,geoid)
lat = geod2cen(lat,units)
lat = geod2cen(lat,geoid,units)
```

Background

Longitudes have the same meaning whether the Earth is treated as a sphere or as an ellipsoid. However, the definition of latitude in the case of an ellipsoid is more complicated. Geodetic, or geographic, latitudes on an ellipsoid can be transformed to geocentric latitudes on the same ellipsoid.

Geodetic latitude: (also called *geographic latitude*) the angle a normal line passing through a surface point makes with the plane of the Equator.

Geocentric latitude: the angle a line from the center of the ellipsoid passing through a surface point makes with the plane of the Equator.

Description

`lat = geod2cen(lat0)` returns the geodetic latitudes provided in `lat0` transformed to geocentric latitudes, which are returned in `lat`.

`lat = geod2cen(lat0,geoid)` defines the elliptical model of the Earth, given by the geoid vector `geoid`, from which `lat0` is transformed. The default geoid is the same as the default of `almanac('earth','geoid')`, the 1980 Geodetic Reference System ellipsoid.

`lat = geod2cen(lat,geoid,units)` defines the angle units of the inputs and outputs, where *units* is any valid angle units string. The default is `'degrees'`.

Examples

```
lat = geod2cen([0 45 90])
lat =
         0   44.8076   90.0000
```

See Also

`almanac`	Planetary data
`geod2aut` `cen2geod`	Other auxiliary latitude functions

geod2cnf

Purpose

Convert from geodetic to conformal latitudes

Syntax

```
lat = geod2cnf(lat0)
lat = geod2cnf(lat0,geoid)
lat = geod2cnf(lat,units)
lat = geod2cnf(lat,geoid,units)
```

Background

Longitudes have the same meaning whether the Earth is treated as a sphere or as an ellipsoid. However, the definition of latitude in the case of an ellipsoid is more complicated. Geodetic, or geographic, latitudes on an ellipsoid can be transformed to latitudes on an auxiliary sphere with certain properties.

Geodetic latitude: (also called *geographic latitude*) the angle a normal line passing through a surface point makes with the plane of the Equator.

Conformal latitude: latitudes on an auxiliary sphere that is conformal relative to the ellipsoid.

Description

`lat = geod2cnf(lat0)` returns the geodetic latitudes provided in `lat0` transformed to conformal latitudes, which are returned in `lat`.

`lat = geod2cnf(lat0,geoid)` defines the elliptical model of the Earth, given by the geoid vector `geoid`, from which `lat0` is transformed. The default geoid is the same as the default of `almanac('earth','geoid')`, the 1980 Geodetic Reference System ellipsoid.

`lat = geod2cnf(lat,geoid,units)` defines the angle units of the inputs and outputs, where *units* is any valid angle units string. The default is `'degrees'`.

Examples

```
lat = geod2cnf([0 45 90])
lat =
         0   44.8077   90.0000
```

See Also

`almanac`	Planetary data
`geod2aut` `cnf2geod`	Other auxiliary latitude functions

Purpose

Convert from geodetic to isometric latitudes

Syntax

```
lat = geod2iso(lat0)
lat = geod2iso(lat0,geoid)
lat = geod2iso(lat,units)
lat = geod2iso(lat,geoid,units)
```

Background

Longitudes have the same meaning whether the Earth is treated as a sphere or as an ellipsoid. However, the definition of latitude in the case of an ellipsoid is more complicated. Geodetic, or geographic, latitudes on an ellipsoid can be transformed to latitudes on an auxiliary sphere with certain properties.

Geodetic latitude: (also called *geographic latitude*) the angle a normal line passing through a surface point makes with the plane of the Equator.

Isometric latitude: latitudes correctly spaced from the Equator for an ellipsoidal Mercator projection.

Description

`lat = geod2iso(lat0)` returns the geodetic latitudes provided in `lat0` transformed to isometric latitudes, which are returned in `lat`.

`lat = geod2iso(lat0,geoid)` defines the elliptical model of the Earth, given by the geoid vector `geoid`, from which `lat0` is transformed. The default geoid is the same as the default of `almanac('earth','geoid')`, the 1980 Geodetic Reference System ellipsoid.

`lat = geod2iso(lat,geoid,units)` defines the angle units of the inputs and outputs, where *units* is any valid angle units string. The default is `'degrees'`.

Examples

```
lat = geod2iso([0 45])
lat =
          0   50.2275
```

See Also

`almanac`	Planetary data
`geod2aut` `iso2geod`	Other auxiliary latitude functions

geod2par

Purpose

Convert from geodetic to parametric latitudes

Syntax

```
lat = geod2par(lat0)
lat = geod2par(lat0,geoid)
lat = geod2par(lat,units)
lat = geod2par(lat,geoid,units)
```

Background

Longitudes have the same meaning whether the Earth is treated as a sphere or as an ellipsoid. However, the definition of latitude in the case of an ellipsoid is more complicated. Geodetic, or geographic, latitudes on an ellipsoid can be transformed to latitudes on an auxiliary sphere with certain properties.

Geodetic latitude: (also called *geographic latitude*) the angle a normal line passing through a surface point makes with the plane of the Equator.

Parametric latitude: latitudes on the set of spheres with parallel radii identical to the ellipsoidal parallel radii at every latitude.

Description

`lat = geod2par(lat0)` returns the geodetic latitudes provided in `lat0` transformed to parametric latitudes, which are returned in `lat`.

`lat = geod2par(lat0,geoid)` defines the elliptical model of the Earth, given by the geoid vector `geoid`, from which `lat0` is transformed. The default geoid is the same as the default of `almanac('earth','geoid')`, the 1980 Geodetic Reference System ellipsoid.

`lat = geod2par(lat,geoid,units)` defines the angle units of the inputs and outputs, where *units* is any valid angle units string. The default is `'degrees'`.

Examples

```
lat = geod2par([0 45 90])
lat =
         0   44.9038   90.0000
```

See Also

`almanac`	Planetary data
`geod2aut` `par2geod`	Other auxiliary latitude functions

Purpose

Convert from geodetic to rectifying latitudes

Syntax

```
lat = geod2rec(lat0)
lat = geod2rec(lat0,geoid)
lat = geod2rec(lat,units)
lat = geod2rec(lat,geoid,units)
```

Background

Longitudes have the same meaning whether the Earth is treated as a sphere or as an ellipsoid. However, the definition of latitude in the case of an ellipsoid is more complicated. Geodetic, or geographic, latitudes on an ellipsoid can be transformed to latitudes on an auxiliary sphere with certain properties.

Geodetic latitude: (also called *geographic latitude*) the angle a normal line passing through a surface point makes with the plane of the Equator.

Rectifying latitude: latitudes on an auxiliary sphere giving correct distances along meridians relative to the ellipsoid.

Description

`lat = geod2rec(lat0)` returns the geodetic latitudes provided in `lat0` transformed to rectifying latitudes, which are returned in `lat`.

`lat = geod2rec(lat0,geoid)` defines the elliptical model of the Earth, given by the geoid vector `geoid`, from which `lat0` is transformed. The default geoid is the same as the default of `almanac('earth','geoid')`, the 1980 Geodetic Reference System ellipsoid.

`lat = geod2rec(lat,geoid,units)` defines the angle units of the inputs and outputs, where *units* is any valid angle units string. The default is `'degrees'`.

Examples

```
lat = geod2rec([0 45 90])
lat =
         0   44.8557   90.0000
```

See Also

`almanac`	Planetary data
`geod2aut` `rec2geod`	Other auxiliary latitude functions

geographic data structure

Purpose

Define specially formatted structure containing data for map objects

Description

Data for map objects can be stored in a specially formatted structure called a *geographic data structure*, which allows for easy display, manipulation, and extraction of map data. The Mapping Toolbox provides the `displaym`, `extractm`, and `mlayers` tools specifically designed to be used with such structures.

A geographic data structure can be defined for six different types of map objects: lines, patches, regular surfaces, general surfaces, text, and light objects. A specific set of fields is required for each different type of map object. Each of these required fields must exist for each element in the structure. However, fields that are not essential for displaying the object can be left empty. For example, the `otherproperty` field must exist, but can be an empty cell array. Listed below are the six types of map objects along with their required fields.

`line`	`type, tag, lat, long, altitude, otherproperty`
`patch`	`type, tag, lat, long, altitude, otherproperty`
`regular`	`type, tag, map, maplegend, meshgrat, altitude, otherproperty`
`surface`	`type, tag, map, lat, long, altitude, otherproperty`
`text`	`type, tag, lat, long, altitude, string, otherproperty`
`light`	`type, tag, lat, long, altitude, otherproperty`

The `type` field must be one of the specified map object types: `'line'`, `'patch'`, `'regular'`, `'surface'`, `'text'` or `'light'`.

The `tag` field must be a string different from the `type` field usually containing the name or kind of map object.

The `lat`, `long`, and `altitude` fields can be scalar values, vectors, or matrices, as appropriate for the map object type.

The `map` field is a matrix map. If `map` is a regular matrix map, `maplegend` is its corresponding matrix map legend, and `meshgrat` is a two-element vector specifying the graticule mesh size. If `map` is a general matrix map, `lat` and `long` are the matrices of latitude and longitude coordinates.

The otherproperty field is a cell array containing any additional display properties appropriate for the map object. Cell array entries can be a line specification string, such as 'r+', or property-value pairs, such as 'color','red'. If the otherproperty field is left as an empty cell array, default colors are used in the display of lines and patches based on the tag field.

Examples

The layermtx workspace contains five different geographic data structures:

```
load layermtx
whos
  Name             Size         Bytes  Class
  cities           1x10         15956  struct array
  citymarker       1x10          7048  struct array
  lights           1x2           1798  struct array
  matrixmaps       1x2         141414  struct array
  topostruct       1x1         519342  struct array
```

The citymarker structure contains ten elements, each consisting of the required fields for a line object:

```
citymarker
citymarker =
1x10 struct array with fields:
    lat
    long
    type
    tag
    altitude
    otherproperty
```

The first element contains data for a Cape Town city marker. Note that the altitude field can be left empty:

```
citymarker(1)
ans =
              lat: -34
             long: 18.30000000000000
             type: 'line'
              tag: 'Cape Town'
         altitude: []
    otherproperty: {1x1 cell}
```

The otherproperty field indicates that the city marker for Cape Town will be a blue asterisk:

```
citymarker(1).otherproperty
ans =
    'b*'
```

See Also

displaym	Project data in a geographic data structure
extractm	Extract vector data from a geographic data structure
mlayers	GUI for manipulating map layers

Purpose

Define geoid model

Description

The geoid vector is a one- or two-element vector that describes a spheroid approximating the geoid. Its normal form is a two-element vector of the form:

```
[semimajor-axis eccentricity]
```

Eccentricity can range from 0 to 1. When only one element is provided, a spherical (0) eccentricity is assumed. When a geoid input is optional, the lack of an input results in a spherical Earth assumption, which is sufficient for most applications. The semimajor axis is entered in terms of some unit of distance measure. While this could be anything, bear in mind that some functions will calculate output based upon the semimajor axis units.

Examples

When used, the default geoid for the Earth is the 1980 Geodetic Reference System ellipsoid:

```
almanac('earth','geoid','kilometers')
ans =
   1.0e+03 *
   6.37813700000000   0.00008181919104
```

Notice how close this is to a spherical geoid definition:

```
almanac('earth','sphere','kilometers')
ans =
        6371           0
```

See Also

`almanac` Planetary data

getm

Purpose

Get map object properties

Syntax

```
mat = getm(h)
mat = getm(h,MapPropertyName)
getm('MapProjection')
getm('axes')
getm('units')
```

Description

`mat = getm(h)` returns the map structure of the map axes specified by its handle. If the handle of a child of the map axes is specified, only its properties are returned.

`mat = getm(h,MapPropertyName)` returns the specified property value.

`getm('MapProjection')` lists all available projections.

`getm('axes')` lists the map axes properties by property name.

`getm('units')` lists the available units.

Examples

Create a default map axes, and query a property value:

```
axesm('mercator','AngleUnits','degrees')
getm(gca,'MapParallels')
ans =
     0
```

See Also

`axesm`	Define map axes and set map properties
`setm`	Set map properties

Purpose

Interactively assign seeds for matrix map encoding

Syntax

```
[row,col,val] = getseeds(map,maplegend,nseeds)
[row,col,val] = getseeds(map,maplegend,nseeds,seedval)
rcvmat = getseeds(...)
```

Description

This command allows you to interactively create the seed matrix values used by the encodem command to fill in regions of matrix maps.

`[row,col,val] = getseeds(map,maplegend,nseeds)` prompts the user for a number, `nseeds`, of mouse-input locations on the current map axes. After the locations are selected, the user is prompted for a value to associate with each location. The outputs are the row and column, `row` and `col`, of the input regular matrix map, `map`, with its associated map legend vector `maplegend`, corresponding to the input locations. The third output, `val`, returns the selected value for each location.

`[row,col,val] = getseeds(map,maplegend,nseeds,seedval)` predefines the values of the locations. If `seedval` is a scalar, the same value is assigned to all points. If it is a vector with a length of `nseeds`, each entry corresponds to a particular location.

`seedmat = getseeds(...)` packs the outputs into a single, three-column matrix, `seedmat`, which is a suitable input for the `encodem` command. The form of this matrix is `[lat lon val]`.

Examples

Demonstrate this for yourself by typing the following and interactively selecting points:

```
load topo
axesm('gortho','grid','on')
seedmat = getseeds(topo,topolegend,3)
```

When you have selected three points, you will be prompted for their values. The regular matrix map need not be displayed to execute `getseeds` on it.

See Also

`encodem` Fill in regions of indexed matrix maps with specified values

grepfields

Purpose

Identify matching fields in fixed record length files

Syntax

```
grepfields(filename,searchstring)
grepfields(filename,searchstring,casesens)
grepfields(filename,searchstring,casesens,startcol)
grepfields(filename,searchstring,casesens,startfield,fields)
grepfields(filename,searchstring,casesens,startfield,fields,
   machineformat)
indx = grepfields(...)
```

Description

`grepfields(filename,searchstring)` displays lines in the file that begin with the search string. The file must have fixed length records with line endings.

`grepfields(filename,searchstring,casesens)`, with `casesens` 'matchcase' specifIes a case-sensitive search. If omitted or 'none', the search string will match regardless of the case.

`grepfields(filename,searchstring,casesens,startcol)` searches starting with the specified column. `startcol` is an integer between 1 and the bytes-per-record in the file. In this calling form, the file is regarded as a text file with line endings.

`grepfields(filename,searchstring,casesens,startfield,fields)` searches within the specified field. `startfield` is an integer between 1 and the number of fields-per-record. The format of file is described by the `fields` structure. See `readfields` for recognized fields structure entries. In this calling form, the file can be binary and lack line endings. The search is within `startfield`, which must be a character field.

`grepfields(filename,searchstring,casesens,startfield,fields,machineformat)` opens the file with the specified machine format. `machineformat` must be recognized by `fopen`.

`indx = grepfields(...)` returns the record numbers of matched records instead of displaying them on-screen.

Example

Write a binary file and read it

```
fid = fopen('testbin','wb');
for i = 1:3
   fwrite(fid,['character' num2str(i) ],'char');
   fwrite(fid,i,'int8');
   fwrite(fid,[i i],'int16');
   fwrite(fid,i,'integer*4');
   fwrite(fid,i,'real*8');
end
fclose(fid);

fs(1).length = 10;fs(1).type = 'char';fs(1).name = 'field 1';
fs(2).length = 1;fs(2).type = 'int8';fs(2).name = 'field 2';
fs(3).length = 2;fs(3).type = 'int16';fs(3).name = 'field 3';
fs(4).length = 1;fs(4).type = 'integer*4';fs(4).name = 'field 4';
fs(5).length = 1;fs(5).type = 'float64';fs(5).name = 'field 5';
```

Find the record matching the string 'character2'. The record contains binary data, which cannot be properly displayed.

```
grepfields('testbin','character2','none',1,fs)
character2? ? ?   ?@

indx = grepfields('testbin','character2','none',1,fs)
indx =
     2
```

Read the formatted file containing the following:

```
-------------------------------------------------------

character data 1  1  2  3 1e6 10D6

character data 2 11 22 33 2e6 20D6

character data 3111222333 3e6 30D6
```

```
fs(1).length = 16;fs(1).type = 'char';fs(1).name = 'field 1';
fs(2).length = 3;fs(2).type = '%3d';fs(2).name = 'field 2';
fs(3).length = 1;fs(3).type = '%4g';fs(3).name = 'field 3';
fs(4).length = 1;fs(4).type = '%5D';fs(4).name = 'field 4';
fs(5).length = 1;fs(5).type = 'char';fs(5).name = '';
```

Find the records which match at the beginning of the line.

```
grepfields('testfile1','character')
character data 1  1  2  3 1e6 10D6
character data 2 11 22 33 2e6 20D6
character data 3111222333 3e6 30D6

grepfields('testfile1','character data 2')
character data 2 11 22 33 2e6 20D6
```

Find the record which match, starting the search in column 11.

```
grepfields('testfile1','data 2','none',11)
character data 2 11 22 33 2e6 20D6
```

Search record number 1.

```
grepfields('testfile1','character data 2','none',1,fs)
character data 2 11 22 33 2e6 20D6
```

Limitations

Searches are limited to field containing character data.

Remarks

See `readfields` for a complete discussion of the format and contents of the `fields` argument.

See Also

`readfields` Read fields or records from a fixed format file

Purpose

Toggle and control the display of the map grid

Syntax

```
gridm
gridm('on')
gridm('off')
gridm('reset')
gridm(linestyle)
gridm(PropertyName,PropertyValue,...)
```

Description

`gridm` toggles the visibility of the map grid by setting the map axes property `Grid` to `'on'` or `'off'`. The default setting for map axes is `'off'`.

`gridm('on')` sets the map axes `Grid` property to `'on'`.

`gridm('off')` sets the map axes `Grid` property to `'off'`.

`gridm('reset')` resets the entire grid using the current properties. This is essentially a refresh option.

`gridm(linestyle)` sets the map axes `GridLineStyle` property to any line style string recognized by the MATLAB `line` command.

`gridm(PropertyName,PropertyValue,...)` sets the appropriate map axes properties to the desired values. These property names and values are described on the `axesm` reference page of this guide.

Remarks

Map grid properties can be also created or altered using the `axesm` or `setm` commands.

See Also

`axesm`	Define map axes and set map properties
`setm`	Set map properties

grn2eqa

Purpose

Convert from Greenwich to equal area coordinates

Syntax

```
[x,y] = grn2eqa(lat,lon)
[x,y] = grn2eqa(lat,lon,origin)
[x,y] = grn2eqa(lat,lon,origin,geoid)
[x,y] = grn2eqa(lat,lon,origin,units)
[x,y] = grn2eqa(lat,lon,origin,geoid,units)

mat = grn2eqa(lat,lon,origin...)
```

Description

This command converts data from Greenwich (latitude-longitude) coordinates to equal-area *x-y* coordinates. The opposite conversion can be performed with `eqa2grn`.

`[x,y] = grn2eqa(lat,lon)` converts the Greenwich coordinates `lat` and `lon` to the equal-area coordinate points `x` and `y`.

`[x,y] = grn2eqa(lat,lon,origin)` specifies the location in the Greenwich system of the *x-y* origin (0,0). The two-element vector `origin` must be of the form `[latitude, longitude]`. The default places the origin at the Greenwich coordinates (0°,0°).

`[x,y] = grn2eqa(lat,lon,origin,geoid)` specifies the two-element geoid vector describing the ellipsoidal model of the figure of the Earth. The `geoid` is spherical by default.

`[x,y] = grn2eqa(lat,lon,origin,units)` specifies the units for the inputs, where *units* is any valid angle units string. The default value is `'degrees'`.

`mat = grn2eqa(lat,lon,origin...)` packs the outputs into a single variable.

Examples

```
lats = [56 34]; longs = [-140 23];
[x,y] = grn2eqa(lats,longs)
x =
   -2.4435    0.4014
y =
    0.8290    0.5592
```

See Also

`eqa2grn`	Convert from equal area to Greenwich coordinates
`hista`	Equal area histogram

gtextm

Purpose

Place text on map using mouse

Syntax

```
h = gtextm(string)
h = gtextm(string,PropertyName,PropertyValue,...)
```

Description

`h = gtextm(string)` places the text object string at the position selected by mouse input. When this function is called, the current map axes are brought up and the cursor is activated for mouse-click position entry. The text object's handle is returned.

`h = gtextm(string,PropertyName,PropertyValue,...)` allows the specification of any properties supported by the MATLAB `text` command.

Example

Create map axes:

```
axesm('sinusoid','FEdgeColor','red')
gtextm('hello world','FontWeight','bold')
```

Click inside the frame, and the text will appear.

See Also

`axesm`	Create map axes object
`textm`	Project text objects

Purpose

Get handles of graphics objects

Syntax

```
handlem
handlem('taglist')
handlem('prompt')
handlem('object',axesh)
handlem('object',axesh,'searchmethod')
h = handlem(object)
h = handlem(handles)
```

Description

`handlem` or `handlem('taglist')` displays a dialog box for selecting the objects for which handles are desired. For more information, see Chapter 6, "GUI Tools," in the *Mapping Toolbox User's Guide*.

`h = handlem('prompt')` displays another dialog box, which allows greater control of object selection.

`h = handlem(object)` returns the handles of those objects specified by the input string. The options for the *object* string are:

`'all'`	for all children of the current axes
`'clabel'`	for contour labels on the current map axes
`'contour'`	for contour lines on the current map axes
`'frame'`	for the map frame
`'grid'`	for the map grid lines
`'hidden'`	for hidden objects on the current axes
`'image'`	for image objects on the current axes
`'light'`	for light objects on the current axes
`'line'`	for line objects on the current axes
`'map'`	for all objects on the map, excluding the frame (default)
`'meridian'`	for longitude grid lines
`'mlabel'`	for longitude labels

'parallel'	for latitude grid lines
'patch'	for patch objects on the current axes
'plabel'	for latitude labels
'surface'	for surface objects on the current axes
'text'	for text objects on the current axes
'tissot'	for tissot indicatrices on the current map axes
'visible'	for visible objects on the current axes

Or any user-defined object tag string.

A prefix of 'all' can be applied to strings defining a Handle Graphics Object type ('allimage', 'allline', 'allsurface', 'allpatch', 'alltext') to determine all object handles that meet the type criteria. Without the 'all' prefix, those objects named by the user with the tagm command are not included (e.g., a line with the tag 'route' would not be included for a object string 'line', but would be for 'allline').

handlem('*object*',axesh) searches within the axes specified by the input handle axesh.

handlem('*object*',axesh,'*searchmethod*') controls the method used to match the 'str' input. If omitted, 'exact' is assumed. Search method 'strmatch' searches for matches at the beginning of the tag, similar to the MATLAB STRMATCH function. Search method 'findstr' searches within the tag, similar to the MATLAB FINDSTR function.

h = handlem(handles) returns those elements of an input vector of handles that are still valid.

See Also

clma	Clear current map
clmo	Clear specified graphics objects
hidem	Hide specified graphics objects
namem	Determine names of valid graphics objects

`showm`	Show specified graphics objects
`tagm`	Assign a name to graphics object tag property

hidem

Purpose

Hide specified graphic object

Syntax

```
hidem
hidem(handle)
hidem(object)
```

Description

`hidem` brings up a dialog box for selecting the objects to hide (set their `Visible` property to `'off'`).

`hidem(handle)` hides the objects specified by a vector of handles.

`hidem(object)` hides those objects specified by the object string, which can be any string recognized by the `handlem` command.

See Also

`clma`	Clear current map
`clmo`	Clear specified graphics objects
`handlem`	Get handle of displayed map objects
`namem`	Determine names of valid graphics objects
`showm`	Show specified graphics objects
`tagm`	Assign name to graphics object tag property

Purpose

Create spatial equal area histogram

Syntax

```
[lat,lon,num] = hista(lats,lons)
[lat,lon,num] = hista(lats,lons,binarea)
[lat,lon,num] = hista(lats,lons,binarea,geoid)
[lat,lon,num] = hista(lats,lons,binarea,units)
[lat,lon,num] = hista(lats,lons,binarea,geoid,units)
```

Description

`[lat,lon,num] = hista(lats,lons)` returns the center coordinates of equal-area bins and the number of observations falling in each based on the geographically distributed input data.

`[lat,lon,num] = hista(lats,lons,binarea)` specifies the equal area bin size, in square kilometers. It is 100 km^2 by default.

`[lat,lon,num] = hista(lats,lons,binarea,geoid)` specifies the elliptical definition of the Earth to be used with the two-element geoid vector. The default geoid model is a spherical Earth, which is sufficient for most applications.

`[lat,lon,num] = hista(lats,lons,binarea,units)` specifies the standard angle unit string. The default value is `'degrees'`.

Examples

Create random data:

```
lats = rand(4)
lats =
    0.4451    0.8462    0.8381    0.8318
    0.9318    0.5252    0.0196    0.5028
    0.4660    0.2026    0.6813    0.7095
    0.4186    0.6721    0.3795    0.4289

longs = rand(4)
longs =
    0.3046    0.3028    0.3784    0.4966
    0.1897    0.5417    0.8600    0.8998
    0.1934    0.1509    0.8537    0.8216
    0.6822    0.6979    0.5936    0.6449
```

Bin the data in 50-by-50 km cells (2500 sq km):

```
[lat,lon,num] = hista(lats,longs,2500);
[lat lon num]
ans =
    0.2574    0.3757    4.0000
    0.7070    0.3757    5.0000
   -0.1923    0.8253    1.0000
    0.2573    0.8253    2.0000
    0.7070    0.8254    4.0000
```

See Also

`eqa2grn` `grn2eqa`	Greenwich/equal are a conversion
`histr`	Spatial equirectangular histogram

Purpose

Create spatial equirectangular histogram

Syntax

```
[lat,lon,num,wnum] = histr(lats,lons)
[lat,lon,num,wnum] = histr(lats,lons,units)
[lat,lon,num,wnum] = histr(lats,lons,bindensty)
[lat,lon,num,wnum] = histr(lats,lons,bindensty,units)
```

Description

This command sorts geographic data into equirectangular bins for histogram purposes. Equirectangular in this context means that each bin has the same angular measurement on each side (e.g., 1°-by-1°). Consequently, the result is not an equal area histogram. The `hista` command provides that capability. However, the results of `histr` can be weighted by their area bias to, in some sense, correct for this.

`[lat,lon,num,wnum] = histr(lats,lons)` returns the center coordinates of equal-rectangular bins and the number of observations, `num`, falling in each based on the geographically distributed input data. Additionally, an area-weighted observation value, `wnum`, is returned. `wnum` is the bin's `num` divided by its normalized area. The largest bin has the same `num` and `wnum`; a smaller bins has a larger `wnum` than `num`.

`[lat,lon,num,wnum] = histr(lats,lons,`*`units`*`)` specifies the standard angle unit string. The default value is `'degrees'`.

`[lat,lon,num,wnum] = histr(lats,lons,bindensty)` sets the number of bins per angular unit. For example, if *`units`* are `'degrees'`, a `bindensty` of 10 would be 10 bins per degree of latitude or longitude, resulting in 100 bins per *square* degree. The default is one cell per angular unit.

Examples

Create random data:

```
lats = rand(4)
lats =
    0.4451    0.8462    0.8381    0.8318
    0.9318    0.5252    0.0196    0.5028
    0.4660    0.2026    0.6813    0.7095
    0.4186    0.6721    0.3795    0.4289

longs = rand(4)
longs =
    0.3046    0.3028    0.3784    0.4966
    0.1897    0.5417    0.8600    0.8998
    0.1934    0.1509    0.8537    0.8216
    0.6822    0.6979    0.5936    0.6449
```

Bin the data in 0.5-by-0.5 degree cells (2 bins per degree):

```
[lat,lon,num,wnum] = histr(lats,longs,2);
[lat,lon,num,wnum]
ans =
    0.2500    0.2500    3.0000    3.0000
    0.7500    0.2500    4.0000    4.0003
    0.2500    0.7500    4.0000    4.0000
    0.7500    0.7500    5.0000    5.0004
```

The bins centered at 0.75°N are slightly smaller in area than the others. `wnum` reflects the relative count per normalized unit area.

See Also

`filterm`	Geographic filter for data sets
`hista`	Spatial equal area histogram

Purpose

Round from *hms* format to *hm* format

Syntax

```
timeout = hms2hm(timein)
```

Description

`timeout = hms2hm(timein)` rounds times input in hours-minutes-seconds (*hms*) format to the appropriate value in hours-minutes (*hm*) format. This special handling is needed because there are 60, not 100, seconds in a minute.

Example

Round 12:34:29 and 12:34:31 to *hm* format:

```
timeout = hms2hm([1234.29 1234.31])
timeout =
        1234        1235
```

See Also

`hms2hr` `sec2hr`	Other direct time conversion functions
`hms2mat`	Convert from *hms* to separated matrix components
`mat2hms`	Convert from separated matrices to *hms* format
`timedim`	Convert time units

hms2hr, hms2sec

Purpose

Convert time units from *hms* format to hours or seconds

Syntax

```
timeout = hms2hr(timein)
timeout = hms2sec(timein)
```

Description

`timeout = hms2hr(timein)` converts times input in hours-minutes-seconds (*hms*) format to the equivalent measure in decimal hours.

`timeout = hms2sec(timein)` converts times input in hours-minutes-seconds (*hms*) format to the equivalent measure in seconds.

Remarks

The inputs can be in hours-minutes (*hm*) format, since numerically they look like *hms* format in which seconds are always zero.

Example

```
hms2hr(1230)
ans =
   12.5000

hms2sec(100.10)
ans =
        3610
```

See Also

`hms2hm`	Round *hms* format to *hm* format
`hms2mat`	Convert from *hms* to separated matrix components
`sec2hr` `hr2hms`	Other direct time conversion functions
`mat2hms`	Convert from separated matrices to *hms* format
`timedim`	Convert time units

Purpose

Convert the elements of *hms* format to distinct matrix elements

Syntax

```
[h,m,s] = hms2mat(timein)
[h,m,s] = hms2mat(timein,n)
matout = hms2mat(timein,n)
```

Description

`[h,m,s] = hms2mat(timein)` takes times in *hms* format and splits their components into three outputs, one each for hours, minutes, and seconds.

`[h,m,s] = hms2mat(timein,n)` specifies the power of 10, `n`, to which the resulting seconds output should be rounded (i.e., if a result is 12.567 seconds, and `n` = –2, the resulting seconds output would be 12.57). The default value of `n` is –5.

`matout = hms2mat(timein,n)` returns a three-column matrix, `matout`, in which the columns represent hours, minutes, and seconds, respectively. In this case, `timein` must be a vector.

Examples

```
[h,m,s] = hms2mat(1234.567)
h =
    12
m =
    34
s =
   56.7000

matout = hms2mat(1234.567)
matout =
   12.0000   34.0000   56.7000
```

See Also

`mat2hms` Convert from separated matrices to *hms* format

hr2hms, hr2hm

Purpose

Convert time units from hours to *hms* or *hm*

Syntax

```
timeout = hr2hms(timein)
timeout = hr2hm(timein)
```

Description

`timeout = hr2hms(timein)` converts times input in hours to the equivalent measure in the hours-minutes-seconds (*hms*) format.

`timeout = hr2hm(timein)` converts times input in hours to the equivalent measure in the hours-minutes (*hm*) format. This is the *hms* format, properly rounded to just hours and minutes.

Example

```
hr2hms(12.51)
ans =
       1230.36

hr2hm(12.51)
ans =
       1231.00
```

See Also

`hms2mat`	Convert from *hms* to separated matrix components
`sec2hr` `hr2hms`	Other direct time conversion functions
`mat2hms`	Convert from separated matrices to *hms* format
`timedim`	Convert time units

Purpose

Convert time from hours to seconds

Syntax

```
timeout = hr2sec(timein)
```

Description

`timeout = hr2sec(timein)` converts times input in hours to the equivalent measure in seconds.

Example

```
hr2sec(1)
ans =
        3600
```

See Also

`sec2hr` `hr2hms`	Other direct time conversion functions
`timedim`	Convert time units

imagem

Purpose

Display regular matrix map as an image

Syntax

```
h = imagem(map,maplegend)
h = imagem(map,maplegend,PropertyName,PropertyValue,...)
```

Description

This command displays regular matrix maps as images and displays latitude and longitude axes. Because it is a matrix image, the resulting projection is equirectangular (Platte Carre in particular) and hence is not necessarily ideal as a final product. However, `imagem` is useful for quickly viewing matrix maps for content.

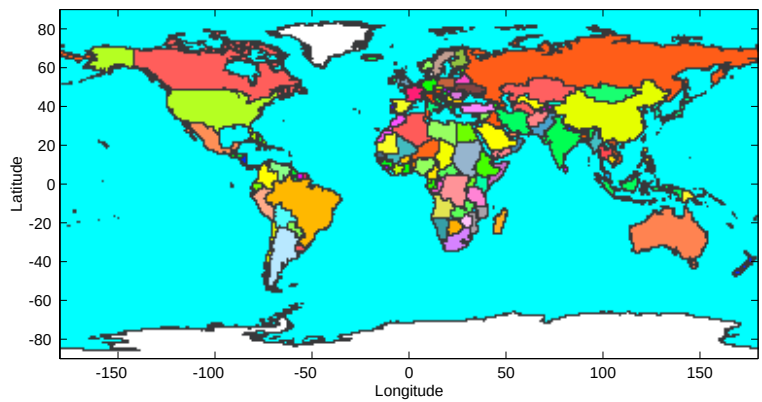
Note that `imagem` uses standard MATLAB axes, *not* map axes.

`h = imagem(map,maplegend)` displays the regular matrix map, `map`, as an image, as well as latitude and longitude scales. The map legend vector `maplegend` is also required. The object handle `h` can be returned.

`h = imagem(map,maplegend,PropertyName,PropertyValue,...)` allows the specification of property/value pairs to control the image object properties. Any property supported by the standard MATLAB command `image` except `XData` and `YData` can be altered. For a complete description of image properties, see `image` in the online *MATLAB Function Reference*.

Examples

```
load worldmtx
imagem(map,maplegend)
colormap(clrmap)
```



See Also

`pcolorm`	Project a matrix map in the `z = 0` plane

imbedm

Purpose

Encode data points into regular matrix map

Syntax

```
newmap = imbedm(lat,lon,value,map,maplegend)
newmap = imbedm(lat,lon,value,map,maplegend,units)
[newmap,badindx] = imbedm(lat,lon,value,map,maplegend,units)
```

Description

`newmap = imbedm(lat,long,value,map,maplegend)` resets certain entries of a regular matrix map, `map`. The entries to be reset correspond to the locations defined by the latitude and longitude position vectors `lat` and `lon`. The entries are reset to the same number if `value` is a scalar, or to individually specified numbers if `value` is a vector the same size as `lat` and `lon`. If any points lie outside the input `map`, a warning is displayed.

`newmap = imbedm(lat,lon,value,map,maplegend,units)` specifies the units of the vectors `lat` and `lon`, where *`units`* is any valid angle units string ('degrees' by default).

`[newmap,badindx] = imbedm(lat,lon,value,map,maplegend,units)` returns the indices of `lat` and `lon` corresponding to points outside the `map` in the variable `badindx`.

Examples

Create a simple map and imbed new values in it:

```
map = ones(3,6)
map =
     1     1     1     1     1     1
     1     1     1     1     1     1
     1     1     1     1     1     1

maplegend = [1/60 90 -180]
maplegend =
    0.0167   90.0000 -180.0000

newmap = imbedm([23 -23], [45 -45],[5 5],map,maplegend)
newmap =
     1     1     1     1     1     1
     1     1     5     5     1     1
     1     1     1     1     1     1
```

See Also

`ltln2val`	Latitude and longitude to matrix entry value
`setpostn`	Latitude and longitude to row and column

inputm

Purpose

Select latitude and longitude coordinates using mouse

Syntax

```
[lat,long] = inputm
[lat,long] = inputm(npts)
[lat,long] = inputm(npts,hndl)
pts = inputm(npts)
```

Description

`[lat,long] = inputm` returns the latitude-longitude point of a mouse-selected point of the current map axes.

`[lat,long] = inputm(npts)` specifies the number of points, `npts`, to be mouse-selected.

`[lat,long] = inputm(npts,hndl)` specifies the handle of the map axes on which mouse selection is desired.

`pts = inputm(npts)` returns the points in a single, two-column output.

Remarks

`inputm` works much like the standard MATLAB `ginput`, except that the returned values are latitudes and longitudes extracted from the projection, rather than axes *x-y* coordinates.

See Also

`gcpmap`	Map current point
`ginput`	Request user input (see online *MATLAB Function Reference*)

Purpose

Linearly interpolate latitude and longitude data to a given resolution

Syntax

```
[latout,lonout] = interpm(lat,lon,maxdiff)
[latout,lonout] = interpm(lat,lon,maxdiff,method)
[latout,lonout] = interpm(lat,lon,maxdiff,method,units)
```

Description

`[latout,lonout] = interpm(lat,lon,maxdiff)` *fills in* any gaps in latitude (`lat`) or longitude (`lon`) data vectors that are greater than a defined tolerance `maxdiff` apart in either dimension. The angle units of the three inputs need not be specified, but they must be identical. `latout` and `lonout` are the new latitude and longitude data vectors, in which any gaps larger than `maxdiff` in the original vectors have been filled with additional points. The default method of interpolation used by `interpm` is linear.

`[latout,lonout] = interpm(lat,lon,maxdiff,method)` interpolates between vector data coordinate points using a specified interpolation method. Valid interpolation method strings are `'gc'` for great circle,`'rh'` for rhumb line, and `'lin'` for linear interpolation.

`[latout,lonout] = interpm(lat,lon,maxdiff,method,units)` specifies the units used, where *units* is any valid angle units string. The default is `'degrees'`.

Examples

```
lat = [1 2 4 5]; lon = [7 8 9 11];
[latout,lonout] = interpm(lat,lon,1);
[latout lonout]
ans =
    1.0000    7.0000
    2.0000    8.0000
    3.0000    8.5000
    4.0000    9.0000
    4.5000   10.0000
    5.0000   11.0000
```

See Also

`intrplat`	Interpolated latitudes for given longitudes
`intrplon`	Interpolated longitudes for given latitudes

intrplat

Purpose

Interpolate latitude for a given longitude

Syntax

```
newlat = intrplat(long,lat,newlong)
newlat = intrplat(long,lat,newlong,method)
newlat = intrplat(long,lat,newlong,method,units)
```

Description

The command `intrplat` is a geographic data analogy of the standard MATLAB command `interp1`.

`newlat = intrplat(long,lat,newlong)` returns an interpolated latitude, `newlat`, corresponding to a longitude `newlong`. `long` must be a monotonic vector of longitude values. The actual entries must be monotonic, i.e., the longitude vector `[350 357 3 10]` is not allowed even though the geographic *direction* is unchanged (use `[350 357 363 370]` instead). `lat` is a vector of the latitude values paired with each entry in `long`.

`newlat = intrplat(long,lat,newlong,method)` specifies the method of interpolation employed. The default value of the *method* string is `'linear'`, which results in linear, or Cartesian, interpolation between the numerical values entered. This is really just a pass-through to the MATLAB `interp1` command. Similarly, `'spline'` and `'cubic'` perform cubic spline and cubic interpolation, respectively. The `'rh'` method returns interpolated points that lie on rhumb lines between input data. Similarly, the `'gc'` method returns interpolated points that lie on great circles between input data.

`newlat = intrplat(long,lat,newlong,method,units)` specifies the units used, where *units* is any valid angle units string. The default is `'degrees'`.

Examples

Compare the results of the various methods:

```
lats = [25 45]; longs = [30 60];
newlat = intrplat(longs,lats,45,'linear')
newlat =
    35

newlat = intrplat(longs,lats,45,'rh')
newlat =
   35.6213

newlat = intrplat(longs,lats,45,'gc')
newlat =
   37.1991
```

Remarks

There are separate commands for interpolating latitudes and longitudes, for although the cases are identical when using those methods supported by `interp1`, when latitudes and longitudes are treated like the spherical angles they are (using `'rh'` or `'gc'`), the results are different. Compare the above example to the example in under `intrplon`, which reverses the values of latitude and longitude.

See Also

`interpm`	Linear interpolation of latitude and longitude
`intrplon`	Interpolated longitudes for given latitudes

intrplon

Purpose

Interpolate longitude for a given latitude

Syntax

```
newlon = intrplon(lat,lon,newlat)
newlon = intrplon(lat,lon,newlat,method)
newlon = intrplon(lat,lon,newlat,method,units)
```

Description

The command `intrplon` is a geographic data analogy of the MATLAB command `interp1`.

`newlon = intrplon(lat,lon,newlat)` returns an interpolated longitude, `newlon`, corresponding to a latitude `newlat`. `lat` must be a monotonic vector of longitude values. `lon` is a vector of the longitude values paired with each entry in `lat`.

`newlon = intrplon(lat,lon,newlat,method)` specifies the method of interpolation employed. The default value of the *method* string is `'linear'`, which results in linear, or cartesian, interpolation between the numerical values entered. This is really just a pass-through to the MATLAB `interp1` command. Similarly, `'spline'` and `'cubic'` perform cubic spline and cubic interpolation, respectively. The `'rh'` method returns interpolated points that lie on rhumb lines between input data. Similarly, the `'gc'` method returns interpolated points that lie on great circles between input data.

`newlon = intrplon(lat,lon,newlat,method,units)` specifies the units used, where *units* is any valid angle units string. The default is `'degrees'`.

Examples

Compare the results of the various methods:

```
lon = [25 45]; lat = [30 60];
newlon = intrplon(lat,long,45,'linear')
newlon =
    35

newlon = intrplon(lat,long,45,'rh')
newlon =
   33.6515

newlon = intrplon(lat,long,45,'gc')
newlon =
   32.0526
```

Remarks

There are separate commands for interpolating latitudes and longitudes, for although the cases are identical when using those methods supported by `interp1`, when latitudes and longitudes are treated like the spherical angles they are (using `'rh'` or `'gc'`), the results are different. Compare the previous example to the example under `intrplat`, which reverses the values of latitude and longitude.

See Also

`interpm`	Linear interpolation of latitude and longitude
`intrplat`	Interpolated longitudes for given longitude

ismap

Purpose

Test whether axes have a map definition

Syntax

```
mflag = ismap
mflag = ismap(hndl)
[mflag,msg] = ismap(hndl)
```

Description

This command tests an axes object to determine whether it is a map axes.

`mflag = ismap` returns a 1 if the current axes is a map axes, and 0 otherwise.

`mflag = ismap(hndl)` specifies the handle of the axes to be tested.

`[mflag,msg] = ismap(hndl)` returns a string message if the axes is not a map axes, specifying why not.

See Also

`gcm`	Get current map data structure
`ismapped`	Test whether object is projected on map axes

Purpose

Test whether object is projected on map axes

Syntax

```
mflag = ismapped
mflag = ismapped(hndl)
[mflag,msg] = ismapped(hndl)
```

Description

This command tests an object to determine whether it is projected on map axes.

`mflag = ismapped` returns a 1 if the current object is projected on a map axes, and 0 otherwise.

`mflag = ismapped(hndl)` specifies the handle of the object to be tested.

`[mflag,msg] = ismapped(hndl)` returns a string message if the axes is not projected on a map axes, specifying why not.

See Also

`gcm`	Get current map data structure
`ismap`	Test whether axes have a map definition

iso2geod

Purpose

Convert from isometric to geodetic latitudes

Syntax

```
lat = iso2geod(lat0)
lat = iso2geod(lat0,geoid)
lat = iso2geod(lat,units)
lat = iso2geod(lat,geoid,units)
```

Background

Longitudes have the same meaning whether the Earth is treated as a sphere or as an ellipsoid. However, the definition of latitude in the case of an ellipsoid is more complicated. Geodetic, or geographic, latitudes on an ellipsoid can be transformed from latitudes on an auxiliary sphere with certain properties.

Geodetic latitude: (also called *geographic latitude*) the angle a normal line passing through a surface point makes with the plane of the Equator.

Isometric latitude: latitudes correctly spaced from the Equator for an ellipsoidal Mercator projection.

Description

`lat = iso2geod(lat0)` returns the isometric latitudes provided in `lat0` transformed to geodetic latitudes, which are returned in `lat`.

`lat = iso2geod(lat0,geoid)` defines the elliptical model of the Earth, given by the geoid vector `geoid`, to which `lat0` is transformed. The default geoid is the same as the default of `almanac('earth','geoid')`, the 1980 Geodetic Reference System ellipsoid.

`lat = iso2geod(lat,geoid,units)` defines the angle units of the inputs and outputs, where *units* is any valid angle units string. The default is `'degrees'`.

Examples

```
lat = iso2geod([0 45 90])
lat =
         0   41.1704   66.6535
```

See Also

`almanac` Planetary data

`geod2aut` Other auxiliary latitude functions
`par2geod`

Purpose

Convert distance from kilometers to other units

Syntax

```
distout = km2deg(distin)
distout = km2deg(distin,radius)

distout = km2nm(distin)

distout = km2rad(distin)
distout = km2rad(distin,radius)

distout = km2sm(distin)
```

Description

`distout = km2deg(distin)` converts the input distance given in kilometers to degrees. `distout = km2nm(distin)`, `distout = km2rad(distin)`, and `distout = km2sm(distin)` perform analogously, converting to nautical miles, radians, and statute miles, respectively.

`distout = km2deg(distin,radius)` and `distout = km2rad(distin,radius)` specify the radius of the sphere to use, since a degree (or radian) of arc length covers less distance, for example, on Mars than it does on the Earth. You can enter the radius as a number in kilometers, as a call to the `almanac` function (e.g., `almanac('mars','radius','km')`), or you can pass in a string planet name (e.g., `'mars'`), and the function will make the appropriate call to the `almanac` function. The radius of the Earth is the default.

Examples

How many miles is a *10k Run*?

```
distout = km2sm(10)
distout =
    6.2139
```

See Also

`distdim`	Convert distance units
`nm2km` `sm2deg`	Other direct distance conversion functions

legs

Purpose

Determine courses and distances between navigational track waypoints

Syntax

```
[course,dist] = legs(lat,lon)
[course,dist] = legs(lat,lon,method)
[course,dist] = legs(pts)
[course,dist] = legs(pts,method)
mat = legs(lat,...)
```

Description

This is a navigation function. All angles are in degrees, and all distances are in nautical miles. Track legs are the courses and distances traveled between navigational waypoints.

`[course,dist] = legs(lat,lon)` returns the azimuths (`course`) and distances (`dist`) between navigational waypoints, which are specified by the column vectors `lat` and `lon`.

`[course,dist] = legs(lat,lon,`*`method`*`)` specifies the logic for the leg characteristics. If the string *`method`* is `'rh'` (the default), `course` and `dist` are calculated in a rhumb line sense. If *`method`* is `'gc'`, great circle calculations are used.

`[course,dist] = legs(pts)` allows the user to input the waypoints in a single, two-column matrix `pts`.

`mat = legs(lat,...)` packs up the outputs into a single, two-column matrix, `mat`.

Examples

Imagine an airplane taking off from Logan International Airport in Boston (42.3°N,71°W) and travelling to LAX in Los Angeles (34°N,118°W). The pilot wishes to file a flight plan that takes the plane over O'Hare Airport in Chicago (42°N,88°W) for a navigational update, while maintaining a constant heading on each of the two legs of the trip.

What are those headings and how long are the legs?

```
lat = [42.3; 42; 34]; long = [-71; -88; -118];
[course,dist] = legs(lat,long,'rh')
course =
  268.6365
  251.2724
dist =
   1.0e+03 *
    0.7564
    1.4950
```

Upon takeoff, the plane should proceed on a heading of about 269° for 756 nautical miles, then alter course to 251° for another 1495 miles.

How much farther is it travelling by not following a great circle path between waypoints? Using rhumb lines, it is travelling:

```
totalrh = sum(dist)
totalrh =
   2.2514e+03
```

For a great circle route:

```
[coursegc,distgc] = legs(lat,long,'gc'); totalgc = sum(distgc)
totalgc =
   2.2436e+03
```

The great circle path is less than one-half of one percent shorter.

See Also

`dreckon`	Dead reckon points for a track
`gcwaypts`	Find waypoints along a great circle
`navfix`	Mercator-based navigational fixing
`track`	Connect waypoints

lightm

Purpose

Project light objects on the current map axes

Syntax

```
h = lightm(lat,lon)
h = lightm(lat,lon,PropertyName,PropertyValue,...)

h = lightm(lat,lon,alt)
h = lightm(lat,lon,alt,PropertyName,PropertyValue,...)
```

Description

`h = lightm(lat,lon)` projects a light object at the coordinates `lat` and `lon`. The handle, `h`, of the object can be returned.

`h = lightm(lat,lon,PropertyName,PropertyValue,...)` allows the specification of any property/value pair supported by the standard MATLAB `light` command.

`h = lightm(lat,lon,alt)` allows the specification of an altitude, `alt`, for the light object. When omitted, the default is an infinite light source altitude.

Examples

```
load topo
axesm globe; view(120,30)
meshm(topo,topolegend); demcmap(topo)
lightm(0,90,'color','yellow')
material([.5 .5 1]); lighting phong
```

See Also

`light`	Create light (see online *MATLAB Function Reference*)

limitm

Purpose

Determine latitude and longitude limits of a regular matrix map

Syntax

```
[latlimits,lonlimits] = limitm(map,maplegend)
limvec = limitm(map,maplegend)
```

Description

`[latlimits,lonlimits] = limitm(map,maplegend)` returns two-element limit vectors, `latlimits` and `lonlimits`, describing the extremes of the input regular matrix `map` with a legend vector `maplegend`.

`latlimits` and `lonlimits` are of the form `[south-limit north-limit]` and `[west-limit east-limit]`, respectively. All elements are in degrees, since this command deals only with regular matrix maps.

`limvec = limitm(map,maplegend)` returns a single, four-element output vector of the form: `[south-limit north-limit west-limit east-limit]`.

Examples

Using a familiar matrix map:

```
load topo
[latlimits,lonlimits] = limitm(topo,topolegend)
latlimits =
   -90    90
lonlimits =
     0   360
```

Which is what we expect, since `topo` covers the whole globe.

See Also

`map legend vector`	Data structure for describing regular matrix maps

`nanm`
`onem`
`spzerom`
`zerom`
	Create special matrix maps

`sizem`	Row and column dimensions needed for matrix map

Purpose

Project line objects onto current map axes

Syntax

```
h = linem(lat,lon)
h = linem(lat,lon,linetype)
h = linem(lat,lon,PropertyName,PropertyValue,...)

h = linem(lat,lon,z)
h = linem(lat,lon,z,linetype)
h = linem(lat,lon,z,PropertyName,PropertyValue,...)
```

Description

`linem` is the mapping equivalent of the MATLAB `line` function. It is a low-level graphics function for displaying line objects in map projections. Ordinarily, it is not used directly. Use `plotm` or `plot3m` instead.

`h = linem(lat,lon)` displays projected line objects on the current map axes. `lat` and `lon` are the latitude and longitude coordinates, respectively, of the line object to be projected. Note that this ordering is conceptually reversed from the MATLAB `line` command, because the *vertical* (*y*) coordinate comes first. However, the ordering latitude, then longitude, is standard geographic usage. `lat` and `lon` must be the same size and in the `AngleUnits` of the map axes. The object handle for the displayed line can be returned in `h`.

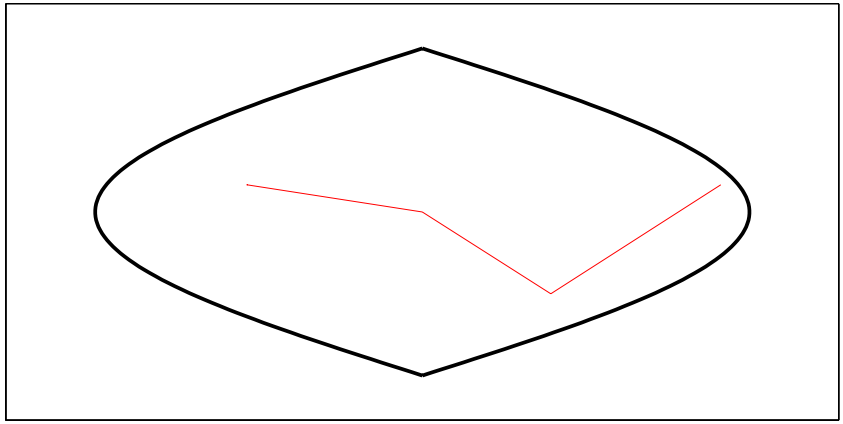
`h = linem(lat,lon,linetype)` allows the specification of the line style, where *linetype* is any string recognized by the MATLAB `line` command.

`h = linem(lat,lon,PropertyName,PropertyValue,...)` allows the specification of any number of property/value pairs for any properties recognized by the MATLAB `line` command except for `XData`, `YData`, and `ZData`.

`h = linem(lat,lon,z)` displays a line object in three dimensions, where `z` is the same size as `lat` and `lon` and contains the desired altitude data. `z` is independent of `AngleUnits`. If omitted, all points are assigned a *z*-value of 0 by default.

Examples

```
axesm sinusoid; framem
linem([15; 0; -45; 15],[-100; 0; 100; 170],'r-')
```



See Also

`line`	Create line (see online MATLAB Funtion Reference)
`plot3m`	Project lines onto current map axes in 3-D space
`plotm`	Project 2-D lines onto current map axes

Purpose

Determine matrix map entries or interpolated values associated with latitude-longitude points

Syntax

```
value = ltln2val(map,maplegend,lat,lon)
value = ltln2val(map,maplegend,lat,lon,method)
```

Description

`value = ltln2val(map,maplegend,lat,lon)` returns the values of the regular matrix map, `map`, corresponding to the locations specified by the vectors `lat` and `lon`.

`value = ltln2val(map,maplegend,lat,lon,`*`method`*`)` specifies the method for determining the returned value. The entry values of a matrix map are actually vertex values. The default *`method`* is `'nearest'`, which returns the unaltered value of the cell containing the coordinates `lat` and `lon`. Using a *`method`* of `'linear'` or `'cubic'` results in values that are linearly and cubically interpolated between vertices, respectively.

Examples

Find the elevations in `topo` associated with three European cities – Milan, Bern, and Prague (`topo` elevations are in meters):

```
load topo
```

The city locations, [Milan Bern Prague]:

```
lats = [45.45; 46.95; 50.1];
longs = [9.2; 7.4; 14.45];

elevations = ltln2val(topo,topolegend,lats,longs)
elevations =
    94
   902
   545
```

See Also

`findm` Coordinates of nonzero map entries

majaxis

Purpose

Calculate semimajor axis from semiminor axis and eccentricity

Syntax

```
semimajor = majaxis(semiminor,eccentricity)
semimajor = majaxis([semiminor,eccentricity])
```

Description

The semimajor axis, the first element of the standard geoid vector in the Mapping Toolbox, can be determined given both the semiminor axis and the eccentricity.

`semimajor = majaxis(semiminor,eccentricity)` returns the semimajor axis length corresponding to the input semiminor axis and eccentricity.

`semimajor = majaxis([semiminor eccentricity])` allows the inputs to be packed into a single, two-column input of the form `[semiminor eccentricity]`.

Examples

Using the default values for the Earth:

```
semimajor = majaxis(6356.7523,0.0818192)
semimajor =
   6.3781e+03
```

This is the default semimajor axis.

See Also

`almanac`	Planetary data
`axes2ecc` `minaxis`	Related conversion functions

Purpose

Make an object a mapped object

Syntax

```
makemapped(h)
```

Background

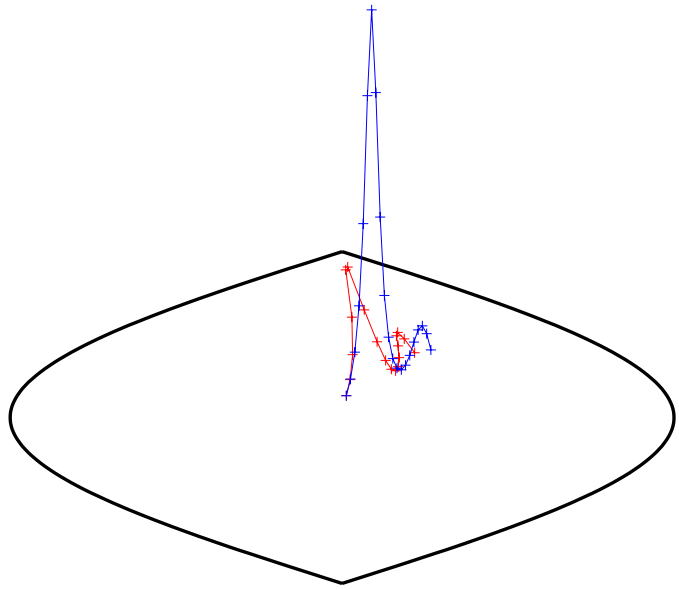
The Mapping Toolbox identifies displayed objects that have been projected from geographic coordinates by a special structure in that object's `UserData` property. Objects created using standard MATLAB display commands lack this structure, and retain the same Cartesian coordinates, regardless of the projection. This function adds the structure that makes an object mapped. The coordinates are unchanged in the process, but will change if the map projection is modified.

Description

`makemapped(h)` adds a Mapping Toolbox structure to the displayed objects associated with `h`. `h` can be a handle, vector of handles, or any name string recognized by `handlem`. The objects are then considered to be geographic data. Objects extending outside the map frame should first be trimmed to the map frame using `trimcart`.

Examples

```
axesm('miller','geoid',[25 0])
framem
plot(humps,'b+-')

h = plot(humps,'r+-');
trimcart(h)
makemapped(h)

setm(gca,'MapProjection','sinusoid')
```



Remarks

Objects should first be trimmed to the map frame using `trimcart`. This avoids problems in taking inverse map projections with out-of-range data.

See Also

`trimcart`	Trims graphic objects to the map frame
`handlem`	Returns the graphics handle for identified objects
`cart2grn`	Transforms from projected cartesian coordinates to Greenwich frame

Purpose

Define resolution and location of regular matrix maps

Description

The map legend is a three element vector that identifies the size of each matrix entry and the coordinates on the globe of the northwestern corner of the regular matrix map (i.e., the upper-left corner of the matrix). The three entries are given in degrees (regular matrix maps must be in degrees). The specific format of the structure is:

```
maplegend = [cells/deg northmost-latitude westmost-longitude]
```

Examples

The topo workspace contains a map legend in the variable `topolegend`:

```
load topo
topolegend
topolegend =
     1    90     0
```

This map legend can be interpreted as:

1. Each matrix entry represents one degree of latitude and one degree of longitude.
2. The northern edge of the map is at 90°N (the North Pole).
3. The western edge of the map is at 0° (the Prime Meridian).

See Also

`limitm`	Latitude and longitude limits of a regular matrix map
`sizem`	Row and column dimensions needed for a regular matrix map

maps

Purpose

List projection names or projection codes

Syntax

```
maps
strmat = maps('namelist')
strmat = maps('idlist')
stdstr = maps(id_string)
```

Description

`maps` displays in the command window a table describing all projections available for use.

`strmat = maps('namelist')` returns the English names for the available projections as a matrix of strings.

`strmat = maps('idlist')` returns the standard projection identification strings for the available projections as a matrix of strings.

`stdstr = maps(id_string)` returns the specific standard projection identification string associated with a unique truncation abbreviation.

Examples

To show the first five entries of the projections name list:

```
str1 = maps('namelist');
str1(1:5,:)
ans =
Balthasart Cylindrical
Behrmann Cylindrical
Bolshoi Sovietskii Atlas Mira
Braun Perspective Cylindrical
Cassini Cylindrical
```

The corresponding shorthand names are:

```
str2 = maps('idlist');
str2(1:5,:)
ans =
balthsrt
behrmann
bsam
braun
cassini
```

These are the strings used, for example, when setting the axesm property MapProjection.

The functions setm and axesm recognizes unique abbreviations (truncations) of these strings. The maps function can be used to convert such an abbreviation to the standard id string:

```
stdstr = maps('merc')
stdstr =
mercator
```

When the function name alone is used:

```
maps
MapTools Projections
CLASS            NAME                              ID STRING
Cylindrical      Balthasart Cylindrical            balthsrt
Cylindrical      Behrmann Cylindrical              behrmann
Cylindrical      Bolshoi Sovietskii Atlas Mira*    bsam
Cylindrical      Braun Perspective Cylindrical*    braun
Cylindrical      Cassini Cylindrical               cassini
Cylindrical      Central Cylindrical*              ccylin
Cylindrical      Equal Area Cylindrical            eqacylin
Cylindrical      Equidistant Cylindrical           eqdcylin
Cylindrical      Gall Isographic                   giso
```

The actual result contains all defined projections.

See Also

axesm	Create map axes
setm	Modify the properties of a displayed map

maptriml

Purpose

Trim line vector map to a specified region

Syntax

```
[lat,lon] = maptriml(lat0,lon0,latlim,lonlim)
```

Description

`[lat,lon] = maptriml(lat0,lon0,latlim,lonlim)` returns *filtered* `NaN`-delimited vector map data sets from which all points lying outside the desired latitude and longitude limits have been discarded. These limits are specified by the two-element vectors `latlim` and `lonlim`, which are two-element vectors of the form `[south-limit north-limit]` and `[west-limit east-limit]`, respectively.

Examples

A simple example:

```
lat0 = [1:10,9:-1:0]; lon0 = -30:-11;
[lat,lon] = maptriml(lat0,lon0,[3 7],[-29 -12]);
[lat lon]
ans =
   NaN   NaN
     3   -28
     4   -27
     5   -26
     6   -25
     7   -24
   NaN   NaN
     7   -18
     6   -17
     5   -16
     4   -15
     3   -14
   NaN   NaN
```

Notice that trimmed line segment ends have `NaN`s inserted at trim points.

See Also

`maptrimp`	Trim patch map data to specified region
`maptrims`	Trim surface matrix map data to specified region

Purpose

Trim patch map data to a specified region

Syntax

```
[lat,lon] = maptrimp(lat0,lon0,latlim,lonlim)
```

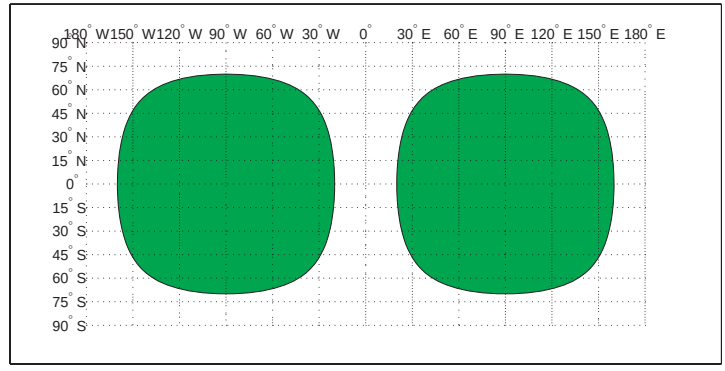
Description

`[lat,lon] = maptrimp(lat0,lon0,latlim,lonlim)` returns *adjusted* patch map data sets. The patches must be input in `NaN`-delimited vectors `lat0` and `lon0`. For any patch face completely outside the specified limits, the entire patch face is discarded. When parts of a patch are outside the limits, those data points are moved to lie on the limits. These limits are specified by the two-element vectors `latlim` and `lonlim`, which have the form `[south-limit north-limit]` and `[west-limit east-limit]`, respectively.

Examples

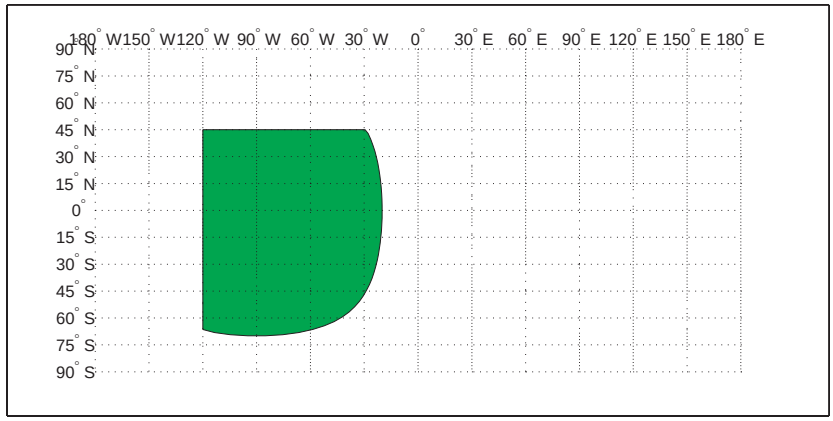
Make two patches using the scircle1 command, and display them:

```
[lat0,lon0] = scircle1([0 0]',[-90 90]',[70 70]');
axesm('pcarree','Grid','on',...
      'MeridianLabel','on','ParallelLabel','on')
h = fillm(lat0,lon0,'green');
```



Now, trim the patch data to lie between 80°S and 45°N latitude, and 120°W and 0° longitude. The patch data is in two columns coming out of scircle1, so first you must turn it into NaN-delimited vectors.

```
lat0 = [lat0; NaN NaN];
lon0 = [lon0; NaN NaN];
[lat,lon] = maptrimp(lat0(:),lon0(:),[45 -80],[-120 0]);
clmo(h)
fillm(lat,lon,'green')
```



Notice that the patch face to the east, lying completely outside the allowed area, was removed. The western patch was trimmed to the required area.

See Also

`maptriml`	Trim line vector map data to specified region
`maptrims`	Trim surface matrix map data to specified region

Purpose

Trim surface regular matrix map data to a specified region

Syntax

```
[submap,sublegend] = maptrims(map,maplegend,latlim,lonlim)
[submap,sublegend] = maptrims(map,maplegend,latlim,lonlim,scale)
```

Description

This command selects a portion of a larger matrix map defined by a latitude-longitude quadrangle.

`[submap,sublegend] = maptrims(map,maplegend,latlim,lonlim)` returns the subset of the input regular matrix `map` between the latitude and longitude limits, in degrees, defined by the two-element vectors `latlim` and `lonlim`. `maplegend` is the map legend vector of the input matrix map; `sublegend` is the map legend vector of the output matrix map.

`[submap,sublegend] = maptrims(map,maplegend,latlim,lonlim,scale)` is a means of further reducing the size of the output matrix. The cells-per-degree scale of the original matrix is given by the first element of `maplegend`. The desired cells-per-degree scale in the output map is given by `scale`, which must equally divide `maplegend(1)`. For example, if `maplegend(1)` were 20 (cells per degree), then `scale` could be 1, 2, 4, 5, 10 or 20.

The reduced matrix is created using `resizem` with a `'nearest'` interpolation method.

Examples

```
load topo
[submap,sublegend] = maptrims(topo,topolegend,...
                              [80.5 85],[165 170.5])
submap =
       -2826       -2810       -2802       -2793       -2790
       -2915       -2913       -2905       -2884       -2844
       -3192       -3186       -3165       -3122       -3078
       -3399       -3324       -3273       -3214       -3167

sublegend =
     1    85   165
```

See Also

`map legend vector`	Data structure for describing regular matrix maps
`maptriml`	Trim line vector map data to specified region
`maptrimp`	Trim patch map data to specified region
`resizem`	Resize a matrix map

Purpose

Replace entries of a matrix map specified by a mask matrix

Syntax

```
mapout = maskm(map,mask)
mapout = maskm(map,mask,newcode)
```

Description

`mapout = maskm(map,mask,newcode)` returns a matrix map `mapout` identical to the matrix map, `map`, except that every element corresponding to a 1 in `mask` is replaced with the scalar value `newcode`. `mask` is a matrix of 1s and 0s, the same size as `map`. The `mask` can be the result of a logical operation on the variable `map` (e.g., `mask = (map>100)`). When no `newcode` is supplied, `NaN`s are inserted.

Examples

Invent a map:

```
map = magic(3)
map =
     8     1     6
     3     5     7
     4     9     2
```

Use a logical test on the map as a mask. Replace with the default value (`NaN`):

```
mapout = maskm(map,(map>5))
mapout =
   NaN     1   NaN
     3     5   NaN
     4   NaN     2
```

See Also

`changem` Replace certain values in a matrix map

mat2dms

Purpose

Convert distinct matrix elements to *dms* format

Syntax

```
anglout = mat2dms(d,m,s)
anglout = mat2dms(d,m,s,n)
anglout = mat2dms([d,m,s],n)
```

Description

`anglout = mat2dms(d,m,s)` takes angles separated into three inputs, one each for degrees, minutes, and seconds, and converts them to single *dms* values.

`anglout = mat2dms(d,m,s,n)` specifies the power of 10, `n`, to which the input seconds should be rounded before they are converted (i.e., if a result is 12.567 seconds, and `n` = –2, the resulting seconds output would be 12.57). The default value of `n` is –2.

`anglout = mat2dms([d,m,s],n)` allows the inputs to be packed into a three-column matrix in which the columns represent degrees, minutes, and seconds, respectively.

Examples

```
anglout = mat2dms(23,45,17.5)
anglout =
                  2345.175
```

See Also

`dms2mat` Convert from *dms* to separated matrices

Purpose

Convert distinct matrix elements to *hms* format.

Syntax

```
timeout = mat2hms(h,m,s)
timeout = mat2hms(h,m,s,n)
timeout = mat2hms([h,m,s],n)
```

Description

`timeout = mat2hms(h,m,s)` takes times separated into three inputs, one each for hours, minutes, and seconds, and converts them to single *hms* values.

`timeout = mat2hms(h,m,s,n)` specifies the power of 10, n, to which the input seconds should be rounded before they are converted (i.e., if a result is 12.567 seconds, and n = –2, the resulting seconds output would be 12.57). The default value of n is –2.

`timeout = mat2hms([h,m,s],n)` allows the inputs to be packed into a three-column matrix in which the columns represent hours, minutes, and seconds, respectively.

Examples

```
timeout = mat2hms([13 35],[34 18],[29.8 17.0])
timeout =

                    1334.298                    3518.17
```

See Also

`hms2mat` Convert from *hms* to separated matrices

mdistort

Purpose

Displays contours of constant distortion on a map

Syntax

```
mdistort
mdistort off
mdistort parameter
mdistort(parameter,levels)
mdistort(parameter,levels,gsize)
[h,ht] = mdistort(...)
```

Background

Map projections inevitably introduce distortions in the shape and size of objects as they are transformed from three-dimensional spherical coordinates to two-dimensional Cartesian coordinates. The amount and type of distortion varies between projections, over the projection, and with the selection of projection parameters such as standard parallels. This function provides a quantitative graphical display of distortion parameters.

Description

`mdistort`, with no input arguments, toggles the display of contours of projection-induced distortion on the current map axes. The magnitude of the distortion is reported in percent.

`mdistort off` removes the contours.

`mdistort(parameter)` or `mdistort parameter` displays contours of distortion for the specified parameter. Recognized *parameter* strings are 'area', 'angles' for the maximum angular distortion of right angles, 'scale' or 'maxscale' for the maximum scale, 'minscale' for the minimum scale, 'parscale' for scale along the parallels, 'merscale' for scale along the meridians, and 'scaleratio' for the ratio of maximum and minimum scale. If omitted, the 'maxscale' parameter is displayed. All parameters are displayed as percent distortion, except angles, which are displayed in degrees.

`mdistort(parameter,levels)` specifies the levels for which the contours are drawn. `levels` is a vector of values as used by `contour`. If empty, the default levels are used.

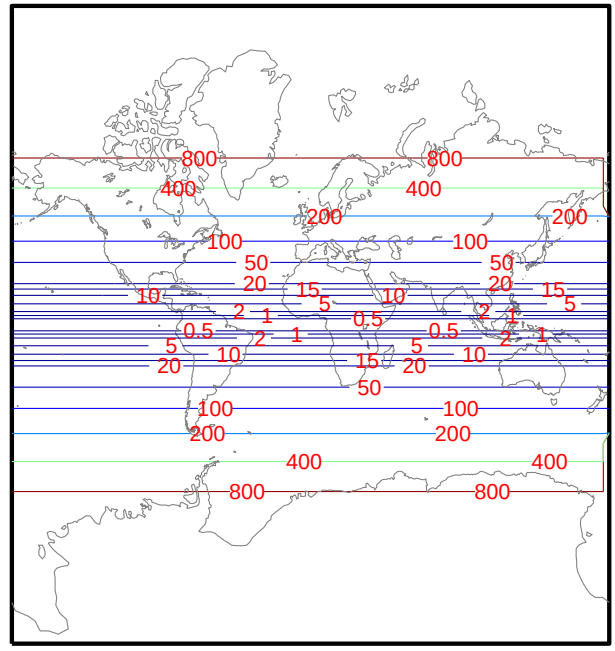
`mdistort(parameter,levels,gsize)` controls the size of the underlying graticule matrix used to compute the contours. `gsize` is a two-element vector containing the number of rows and columns. If omitted, the default Mapping Toolbox graticule size of `[50 100]` is assumed.

`[h,ht] = mdistort(...)` returns the handles to the line and text objects.

Examples

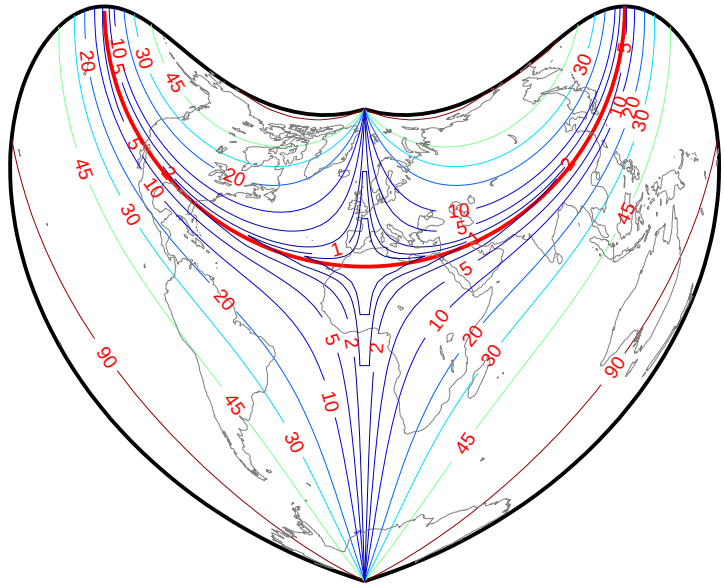
Note the extreme area distortion of the Mercator projection. This makes it ill-suited for global displays.

```
figure
axesm mercator
framem;plotm(coast,'color',.5*[1 1 1])

mdistort area
```



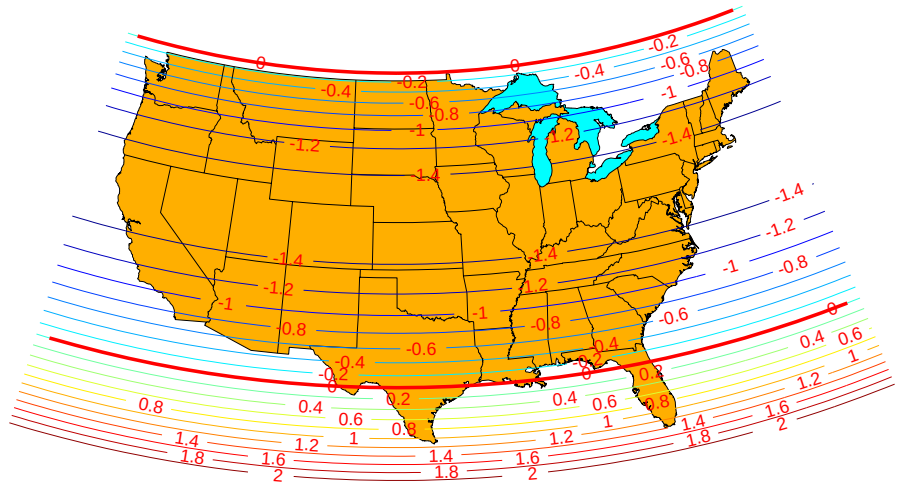
The lines of zero distortion for the Bonne projection follow the central meridian and the standard parallel.

```
figure
axesm bonne
framem;plotm(coast,'color',.5*[1 1 1])

mdistort angles
parallelui
```



An equidistant conic projection with properly chosen parallels can map the conterminous United States with less than 1.5% distortion.

```
figure
usamap conus
mdistort('parscale', -2:.2:2)
parallelui
```



Remarks

`mdistort` can help in the placement of standard parallels for projections. Standard parallels are generally placed to minimize distortion over the region of interest. The default parallel locations may not be appropriate for maps of smaller regions. By using `mdistort` and `parallelui`, you can immediately see how the movement of parallels reduces distortion.

See Also

`tissot`	Tissot indicatrices projected onto a map axes
`distortcalc`	Calculates distortion parameters for a map projection
`vfwdtran`	Transforms vector azimuths to a projection space angle

meanm

Purpose

Compute mean for geographic data

Syntax

```
[latmean,lonmean] = meanm(lat,lon)
[latmean,lonmean] = meanm(lat,lon,units)
[latmean,lonmean] = meanm(lat,lon,geoid)
[latmean,lonmean] = meanm(lat,lon,geoid,units)
geomeans = meanm(...)
```

Background

Finding the mean position of geographic points is more complicated than simply averaging the latitudes and longitudes. `meanm` determines mean position through 3-dimensional vector addition. See the section entitled "Geostatistics" in Chapter 5, "Mapping Applications," of the *Mapping Toolbox User's Guide*.

Description

`[latmean,lonmean] = meanm(lat,lon)` returns row vectors of the geographic mean positions of the columns of the input latitude and longitude points.

`[latmean,lonmean] = meanm(lat,lon,units)` indicates the angular units of the data. When the standard angle string *units* is omitted, `'degrees'` is assumed.

`[latmean,lonmean] = meanm(lat,lon,geoid)` specifies the elliptical definition of the Earth to be used with the two-element `geoid` vector. The default geoid model is a spherical Earth, which is sufficient for most applications.

If a single output argument is used, then `geomeans = [latmean,longmean]`. This is particularly useful if the original `lat` and `lon` inputs are column vectors.

Examples

Create random latitude and longitude matrices:

```
lat = rand(3)
lat =
    0.9501    0.4860    0.4565
    0.2311    0.8913    0.0185
    0.6068    0.7621    0.8214

lon = rand(3)
lon =
    0.4447    0.9218    0.4057
    0.6154    0.7382    0.9355
    0.7919    0.1763    0.9169

[latmean,lonmean] = meanm(lat,lon,'radians')
latmean =
    0.6004    0.7395    0.4448
lonmean =
    0.6347    0.6324    0.7478
```

See Also

`filterm`	Geographic filter for datasets
`hista`	Spatial equal area histogram
`histr`	Spatial equirectangular histogram
`stdist`	Standard distances for geographic data
`stdm`	Standard deviation for geographic data

meshgrat

Purpose

Construct a map graticule mesh for surface object display

Syntax

```
[latgrat,longrat] = meshgrat(map,maplegend)
[latgrat,longrat] = meshgrat(map,maplegend,npts)
[latgrat,longrat] = meshgrat(lat,lon)
[latgrat,longrat] = meshgrat(lat,lon,units)
[latgrat,longrat] = meshgrat(latlim,lonlim,npts)
[latgrat,longrat] = meshgrat(latlim,lonlim,npts,units)
```

Description

The graticule mesh is a grid of points that are projected on a map axes and to which surface map objects are warped. The fineness, or resolution, of this grid determines the quality of the projection and the speed of plotting. There is no hard and fast rule for sufficient graticule resolution, but in general, cylindrical projections need very few graticules in the longitudinal direction, while complex curve-generating projections require more.

`[latgrat,longrat] = meshgrat(map,maplegend)` constructs a graticule for the regular matrix map, `map`, with the associated map legend vector `maplegend`. The default graticule size is equal to the size of the map matrix.

`[latgrat,longrat] = meshgrat(map,maplegend,npts)` returns a graticule mesh of size `npts`. The input `npts` is a two-element vector of the form `[latitude-points longitude-points]`. If `npts` is set to an empty matrix, then the graticule returned is the Mapping Toolbox default graticule size `[50 100]`.

`[latgrat,longrat] = meshgrat(lat,lon)` can be used for matrix maps that are not regular in spacing (e.g., row one represents 1°, row two represents 1.34°) but are regular in orientation (rows are north-south, columns are east-west). The inputs `lat` and `lon` are vectors describing the latitudes and longitudes on a row-by-row and column-by-column basis for the matrix map to be displayed. Regardless of the variable spacing of the matrix, the graticule will be evenly spaced. In this form, `meshgrat` is similar to the MATLAB function `meshgrid`.

`[latgrat,longrat] = meshgrat(latlim,lonlim,npts)` returns a graticule mesh, of size `npts`. The input vectors `latlim` and `lonlim` are two-element vectors specifying the graticule latitude and longitude limits. The input `npts` is a two-element vector of the form `[latitude-points longitude-points]`. If `npts` is set to an empty matrix, then the graticule returned is the Mapping Toolbox default graticule size `[50 100]`.

`[latgrat,longrat] = meshgrat(lat,lon,`*`units`*`)` and `[latgrat,longrat] = meshgrat(latlim,lonlim,npts,`*`units`*`)` use the input *units* to specify the angle units of the input and output parameters. If omitted, `'degrees'` are assumed.

Examples

Make a (coarse) graticule for the entire world:

```
latlim = [-90 90]; longlim = [-180 180];
[latgrat,longrat] = meshgrat(latlim,longlim,[3 6])
latgrat =
  -90.0000  -90.0000  -90.0000  -90.0000  -90.0000  -90.0000
         0         0         0         0         0         0
   90.0000   90.0000   90.0000   90.0000   90.0000   90.0000
longrat =
 -180.0000 -108.0000  -36.0000   36.0000  108.0000  180.0000
 -180.0000 -108.0000  -36.0000   36.0000  108.0000  180.0000
 -180.0000 -108.0000  -36.0000   36.0000  108.0000  180.0000
```

These paired coordinates are the graticule vertices, which are projected according to the requirements of the desired map projection. Then a surface object like the `topo` map can be warped to the grid.

See Also

`meshm`	Regular matrix map warped to projected graticule mesh
`pcolorm`	Project a matrix map in the z = 0 plane
`surfacem`	Matrix map warped to projected graticule mesh
`surfm`	Matrix map projected a map axes

meshlsrm

Purpose

Project 3-D lighted shaded relief of a general matrix map

Syntax

```
meshlsrm(map,maplegend)
meshlsrm(map,maplegend,[azim elev])
meshlsrm(map,maplegend,[azim elev],cmap)
meshlsrm(map,maplegend,[azim elev],cmap,clim)
h = meshlsrm(...)
```

Description

`meshlsrm(map,maplegend)` displays the regular matrix map colored according to elevation and surface slopes. The current axes must have a valid map projection definition.

`meshlsrm(map,maplegend,[azim elev])` displays the regular matrix map with the light coming from the specified azimuth and elevation. Lighting is applied before the data is projected. Angles are in degrees, with the azimuth measured clockwise from North and elevation up from the zero plane of the surface. By default, the direction of the light source is East (90° azimuth) at an elevation of 45°.

`meshlsrm(map,maplegend,[azim elev],cmap)` displays the regular matrix map using the provided colormap. The number of grayscales is chosen to keep the size of the shaded colormap below 256. By default, the colormap is constructed from 16 colors and 16 grays. If the vector of azimuth and elevation is empty, the default locations are used.

`meshlsrm(map,maplegend,[azim elev],cmap,clim)` uses the provided color axis limits, which by default, are computed from the data.

`h = meshlsrm(...)` returns the handle to the surface drawn.

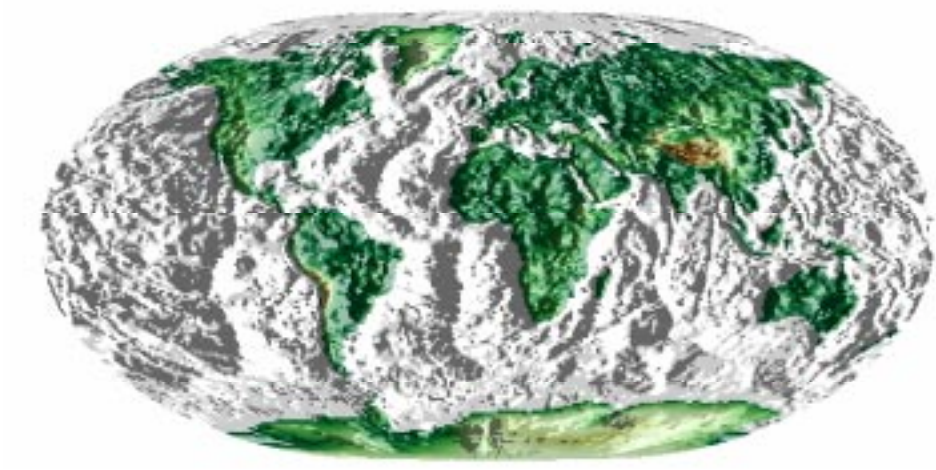
Remarks

This function effectively multiplies two colormaps, one with color based on elevation, the other with a grayscale based on the slope of the surface, to create a new colormap. This produces an effect similar to using a light on a surface, but with all of the visible colors actually in the colormap. Lighting calculations are performed on the unprojected data.

Examples

Create a new colormap using demcmap, with white colors for the sea and default colors for land. Use this colormap for a lighted shaded relief map of the world.

```
load topo
[cmap,clim] = demcmap(topo,[],[1 1 1],[]);
axesm loximuth
meshlsrm(topo,topolegend,[],cmap,clim)
```



See Also

meshm	Regular matrix map warped to projected graticule mesh
pcolorm	Project a matrix map in the z = 0 plane
shaderel	Construct cdata and colormap for colored shaded relief
surfacem	Matrix map warped to projected graticule mesh
surflm	Display a lighted matrix map warped to a projected graticule
surfm	Matrix map projected a map axes
surflsrm	Project a 3-D lighted shaded relief of a general matrix map

meshm

Purpose

Warp regular matrix map to projected graticule mesh

Syntax

```
h = meshm(map,maplegend)
h = meshm(map,maplegend,PropertyName,PropertyValue,...)
h = meshm(map,maplegend,npts)
h = meshm(map,maplegend,npts,alt)
h = meshm(map,maplegend,npts,PropertyName,PropertyValue,...)
h = meshm(map,maplegend,npts,alt,PropertyName,PropertyValue,...)
```

Description

This command warps a regular matrix map to a graticule mesh, which is itself projected according to the `MapProjection` property of the current map axes. The fineness, or resolution, of this grid determines the quality of the projection and the speed of plotting it. There is no hard and fast rule for sufficient graticule resolution, but in general, cylindrical projections need very few graticules points in the longitudinal direction, while complex curve-generating projections require more.

`h = meshm(map,maplegend)` projects the regular matrix map onto the current map axes. The handle, `h`, of the displayed surface can be returned.

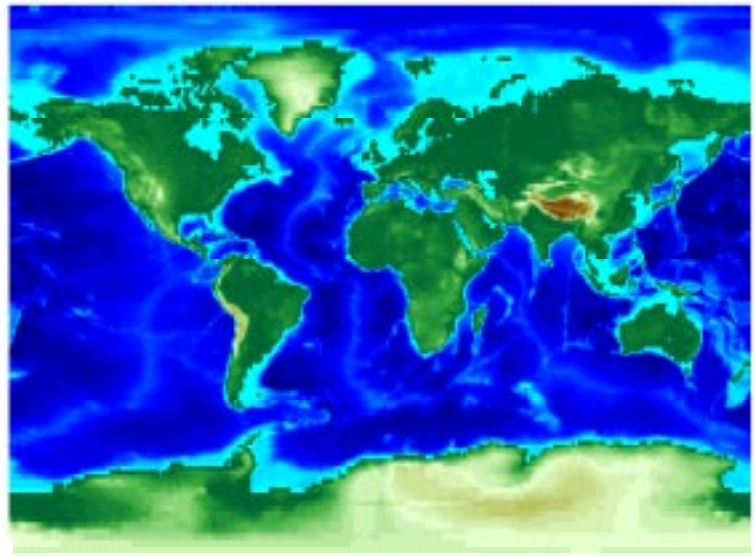
`h = meshm(map,maplegend,npts)` specifies the resolution of the graticule grid. The input `npts` is of the form `[latitude-points longitude-points]`. The default value of `npts` is `[50 100]` (the graticule has 50 vertices in the latitude direction and 100 vertices in the longitude direction).

`h = meshm(map,maplegend,npts,alt)` sets the *z*-axis altitude of the graticule mesh. `alt` can be a scalar, in which case the map is plotted on a `z = alt` plane, or `alt` can be a matrix of `size(alt) = npts`, in which case the graticule mesh is plotted in 3-D.

`h = meshm(map,maplegend,PropertyName,PropertyValue,...)` allows the input of property/value pairs to control the surface object properties. Any property supported by the standard MATLAB command `surface` except `XData`, `YData`, and `ZData` can be altered in this manner.

Examples

```
load topo
axesm miller
meshm(topo,topolegend,[90 180])
demcmap(topo)
```



See Also

`meshgrat`	Construct map graticule grid
`pcolorm`	Project a matrix map in the z = 0 plane
`surfacem`	Matrix map warped to projected graticule mesh
`surfm`	Matrix map projected a map axes

mfwdtran

Purpose

Transform unprojected Greenwich data to a projected Cartesian coordinate system

Syntax

```
[x,y] = mfwdtran(lat,lon)
[x,y,z] = mfwdtran(lat,lon,alt)
[x,y,z,struct] = mfwdtran(lat,lon,alt,object)
[...] = mfwdtran(mstruct,...)
```

Description

`[x,y] = mfwdtran(lat,lon)` transforms unprojected Greenwich data to the projected Cartesian coordinate frame using the map projection defined for the current axes. No clipping or trimming of data is performed with this calling form.

`[x,y,z] = mfwdtran(lat,lon,alt)` transforms the three-dimensional data to the projected Cartesian coordinate frame using the map projection defined for the current axes. If `alt = []` or `alt` is omitted, the default `alt = 0` is used.

`[x,y,z,struct] = mfwdtran(lat,lon,alt,`*`object`*`)` clips and trims the data during the transformation process. Allowable *`object`* strings are `'surface'`, `'line'`, `'patch'`, `'light'`, `'text'`, and `'none'`. `'none'` will result in no clipping or trimming of the input data. The output `struct` is a structure containing information about the clips and trims associated with the transformed object. This structure is also found in the displayed object's `UserData` property.

`[...] = mfwdtran(mstruct,...)` requires a valid map projection structure as the first argument. This structure is used to define the map projection calculations performed. No map axes need be displayed when using this calling form.

Examples

The following latitude and longitude data for the District of Columbia is obtained from the `usalo` workspace:

```
load usalo
lat = state(51).lat;
lon = state(51).long;
[lat lon]
```

```
ans =
   38.9000  -77.0700
   38.9000  -77.0500
   38.9000  -77.0700
   38.8700  -77.0200
   38.8000  -77.0200
   38.7800  -77.0300
   38.9000  -76.9000
   39.0000  -77.0300
   38.9500  -77.1200
   38.9000  -77.0700
```

Before projecting the data, it is necessary to define projection parameters. This can be done with the `axesm` command, or with the `defaultm` command:

```
mstruct = defaultm('mercator');
mstruct.origin = [38.89 -77.04 0];
mstruct = defaultm(mstruct);
```

Now that the projection parameters have been set, transform the District of Columbia data into the Cartesian frame using the Mercator projection:

```
[x,y] = mfwdtran(mstruct,lat,lon);
[x y]
ans =
   -0.0004    0.0002
   -0.0001    0.0002
   -0.0004    0.0002
    0.0003   -0.0003
    0.0003   -0.0016
    0.0001   -0.0019
    0.0019    0.0002
    0.0001    0.0019
   -0.0011    0.0010
   -0.0004    0.0002
```

See Also

`defaultm`	Initialize a default map data structure
`gcm`	Get current map data structure
`minvtran`	Map inverse transformation
`vfwdtran`	Vector forward transformation
`vinvtran`	Vector inverse transformation

Purpose

Calculate semiminor axis from semimajor axis and eccentricity

Syntax

```
semiminor = minaxis(semimajor,eccentricity)
semiminor = minaxis([semimajor,eccentricity])
```

Description

The semiminor axis can be determined given both the semimajor axis and the eccentricity, the two elements of standard geoid vector in the Mapping Toolbox.

`semiminor = minaxis(semimajor,eccentricity)` returns the semiminor axis length corresponding to the input semimajor axis and eccentricity.

`semiminor = minaxis([semimajor,eccentricity])` allows the inputs to be packed into a single, two-column input of the form [semimajor, eccentricity].

Examples

Using the default values for the Earth:

```
semiminor = minaxis(almanac('earth','geoid'))
semiminor =
   6.3568e+03
```

See Also

`almanac`	Planetary data
`axes2ecc` `majaxis`	Related conversion functions

minvtran

Purpose

Transform projected Cartesian data to an unprojected Greenwich coordinate system

Syntax

```
[lat,lon] = minvtran(x,y)
[lat,lon,alt] = minvtran(x,y,z)
[lat,lon,alt] = minvtran(x,y,z,object,struct)
[...] = minvtran(mstruct,...)
```

Description

`[lat,lon] = minvtran(x,y)` transforms projected Cartesian data to an unprojected Greenwich coordinate frame using the map projection defined for the current axes. No data clips or trims are removed with this calling form.

`[lat,lon,alt] = minvtran(x,y,z)` transforms the three-dimensional data to the unprojected Greenwich coordinate frame using the map projection defined for the current axes. If `z = []` or `z` is omitted, the default `z = 0` is used.

`[lat,lon,alt] = minvtran(x,y,z,object,struct)` removes all clips and trims from the input data. Allowable *object* strings are `'surface'`, `'line'`, `'patch'`, `'light'`, `'text'`, and `'none'`. `'none'` will result in no removal of any clips or trims of the input data. The output `struct` is a structure containing information about the clips and trims associated with the transformed object, and is created by the function `mfwdtran`.

`[...] = minvtran(mstruct,...)` requires a valid map projection structure as the first argument. This structure is used to define the map projection calculations performed. No map axes need be displayed when using this calling form.

Examples

Before using any transformation commands, it is necessary to create a map projection structure. You can do this with `axesm` or the `defaultm` command:

```
mstruct = defaultm('mercator');
mstruct.origin = [38.89 -77.04 0];
mstruct = defaultm(mstruct);
```

The following latitude and longitude data for the District of Columbia is obtained from the usalo workspace:

```
load usalo
lat = state(51).lat;
lon = state(51).long;
[lat lon]
ans =
   38.9000  -77.0700
   38.9000  -77.0500
   38.9000  -77.0700
   38.8700  -77.0200
   38.8000  -77.0200
   38.7800  -77.0300
   38.9000  -76.9000
   39.0000  -77.0300
   38.9500  -77.1200
   38.9000  -77.0700
```

This data can be projected into Cartesian coordinates of the Mercator projection using the mfwdtran command:

```
[x,y] = mfwdtran(mstruct,lat,lon);
[x y]
ans =
   -0.0004    0.0002
   -0.0001    0.0002
   -0.0004    0.0002
    0.0003   -0.0003
    0.0003   -0.0016
    0.0001   -0.0019
    0.0019    0.0002
    0.0001    0.0019
   -0.0011    0.0010
   -0.0004    0.0002
```

To transform the projected *x-y* data back into the unprojected Greenwich frame, use the `minvtran` command:

```
[lat2,lon2] = minvtran(mstruct,x,y);
[lat2 lon2]
ans =
   38.9000  -77.0700
   38.9000  -77.0500
   38.9000  -77.0700
   38.8700  -77.0200
   38.8000  -77.0200
   38.7800  -77.0300
   38.9000  -76.9000
   39.0000  -77.0300
   38.9500  -77.1200
   38.9000  -77.0700
```

See Also

`axesm`	Map axes definition and property setting
`defaultm`	Initialize a default map data structure
`gcm`	Get current map data structure
`mfwdtran`	Map forward transformation
`vfwdtran`	Vector forward transformation
`vinvtran`	Vector inverse transformation

Purpose

Project meridian labels on a map axes

Syntax

```
mlabel
mlabel('on')
mlabel('off')
mlabel('reset')
mlabel(parallel)
h = mlabel(MapAxesPropertyName,PropertyValue,...)
```

Description

`mlabel` will toggle the visibility of meridian labelling on the current map axes.

`mlabel('on')` will set the visibility of meridian labels to `'on'`.

`mlabel('off')` will set the visibility of meridian labels to `'off'`.

`mlabel('reset')` will reset the displayed meridian labels using the currently defined meridian label properties.

`mlabel(parallel)` sets the value of the `MLabelParallel` property of the map axes to the value of `parallel`. This determines the parallel upon which the labels are placed (see `axesm`). The options for `parallel` are a scalar latitude, or the strings `'north'`, `'south'`, or `'equator'`.

`mlabel(`*`MapAxesPropertyName`*`,PropertyValue,...)` allows paired map axes property names and property values to be passed in. For a complete description of map axes properties, see the `axesm` reference page in this guide.

Meridian label handles can be returned in `h` if desired.

See Also

`axesm`	Define map axes and set map properties
`plabel`	Parallel labels projected on a map axes
`setm`	Set map properties

n2ecc

Purpose

Convert from the *n* to the eccentricity representation of the ellipsoid

Syntax

```
eccentricity = n2ecc(n)
```

Description

Eccentricity and the parameter *n* are two methods of defining an ellipsoid. The definition of *n* is:

(semimajor axis – semiminor axis)/(semimajor axis + semiminor axis).

`eccentricity = n2ecc(n)` returns the equivalent eccentricities for the input *n* parameters. If the input, *n*, is a two-column vector, only the second column is used. This allows two-element vectors to be used as rows of the input, since the form [semimajor-axis, n] is a complete representation of an ellipsoid (but is not the standard form for geoid vectors in the Mapping Toolbox). In all other cases, all columns of the input are used.

Example

```
ecc = n2ecc(0.00167922039463)
ecc =
   0.08181919104285
```

This eccentricity is the default value for the Earth.

See Also

`almanac`	Planetary data
`ecc2flat` `majaxis`	Similar conversion functions
`ecc2n`	Convert from eccentricity to *n*

Purpose

Determine the names of valid graphics objects

Syntax

```
objects = namem
objects = namem(handles)
[objects,message] = namem(...)
```

Description

`objects = namem` returns the object name for all objects on the current axes. The object name is defined as its tag, if the object `Tag` property is supplied. Otherwise, it is the object `Type`. Duplicate object names are removed from the output string matrix.

`objects = namem(handles)` returns the object names for the objects specified by the input `handles`.

`[objects,message] = namem(...)` returns a string `message` indicating any error encountered.

The names returned are either set at object creation or defined by the user with the `tagm` command.

See Also

`clma`	Clear current map
`clmo`	Clear specified graphics objects
`handlem`	Get handles of displayed graphics objects
`hidem`	Hide specified graphics objects
`showm`	Show specified graphics objects
`tagm`	Assign a name to graphics object tag property

nanclip

Purpose

Convert pen-down delimited data to NaN-delimited data

Syntax

```
dataout = nanclip(datain)
dataout = nanclip(datain,pendowncmd)
```

Description

Pen-down delimited data is a matrix with a first column consisting of pen commands. At the beginning of each segment in the data, this first column has an entry corresponding to a pen-down command. Other entries indicate that the segment is continuing. NaN-delimited data consists of columns of data, each segment of which ends in a NaN in every data column. Since there is no pen command column, the NaN-delimited format can represent the same data in one fewer columns; the remaining columns have more entries, one for each NaN (i.e., for each segment).

`dataout = nanclip(datain,pendowncmd)` returns the pen-down delimited data in the matrix `datain` as NaN-delimited data in `dataout`. When the first column of `datain` equals `pendowncmd`, a segment is started and a NaN is inserted in all columns of `dataout`. The default `pendowncmd` is `–1`.

Examples

```
datain = [-1 45 67; 0 23 54; 0 28 97; -1 47 89; 0 56 12]
datain =
    -1    45    67          % Begin first segment
     0    23    54
     0    28    97
    -1    47    89          % Begin second segment
     0    56    12
dataout = nanclip(datain)
dataout =
    45    67
    23    54
    28    97
   NaN   NaN                % End first segment
    47    89
    56    12
   NaN   NaN                % End second segment
```

See Also

`spcread` Read space-delimited data

Purpose

Create matrix maps of NaNs

Syntax

```
map = nanm(latlim,lonlim,scale)
[map,maplegend] = nanm(latlim,lonlim,scale)
```

Description

`map = nanm(latlim,lonlim,scale)` returns a regular matrix map consisting entirely of NaNs. The two-element vectors `latlim` and `lonlim` define the latitude and longitude limits of the geographic region. They should be of the form [north south] and [east west], respectively. The number of rows and columns per angle unit is set by the scalar value `scale`.

`[map,maplegend] = nanm(latlim,lonlim,scale)` returns the three-element map legend vector for the returned `map`.

Examples

```
[map,maplegend] = nanm([46,51],[-79,-75],1)
map =
   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN
maplegend =
     1    51   -79
```

See Also

`limitm`	Matrix map limits
`map legend vector`	Data structure for describing regular matrix maps
`onem`	Create a matrix map of ones
`sizem`	Row and column dimensions needed for map
`spzerom`	Create a sparse matrix map of zeros
`zerom`	Create a full matrix map of zeros

navfix

Purpose

Determine Mercator-based navigational fix

Syntax

```
[latfix,lonfix] = navfix(lat,long,az)
[latfix,lonfix] = navfix(lat,long,range,casetype)
[latfix,lonfix] = navfix(lat,long,az_range,casetype)
[latfix,lonfix] = navfix(lat,long,az_range,casetype,drlat,drlon)
```

Background

This is a navigational function. As such, it assumes that all latitudes and longitudes are in degrees and all distances are in nautical miles. In navigation, piloting is the practice of fixing one's position based on the observed bearing and ranges *to* fixed landmarks (points of land, lighthouses, smokestacks, etc.) *from* the navigator's vessel. In conformance with navigational practice, bearings are treated as rhumb lines and ranges are treated as the radii of circles on a Mercator projection.

In practice, at least three azimuths (bearings) and/or ranges are required for a usable fix. The resulting intersections are unlikely to exactly coincide. Please refer to the section entitled "Navigation" in Chapter 5 of the *Mapping Toolbox User's Guide* for a more complete description of the use of this function.

Description

`[latfix,lonfix] = navfix(lat,long,az)` returns the intersection points of rhumb lines drawn parallel to the observed bearings, `az`, of the landmarks located at the points `lat` and `long` and passing through these points. One bearing is required for each landmark. Each possible pairing of the *n* landmarks will generate one intersection, so the total number of resulting intersection points is the combinatorial *n choose 2*. The calculation time therefore grows rapidly with *n*.

`[latfix,lonfix] = navfix(lat,long,range,casetype)` returns the intersection points of Mercator projection circles with radii defined by `range`, centered on the landmarks located at the points `lat` and `long`. One range value is required for each landmark. Each possible pairing of the *n* landmarks will generate up to two intersections (circles can intersect twice), so the total number of resulting intersection points is the combinatorial *2 times (n choose 2)*. The calculation time therefore grows rapidly with *n*. In this case, the variable `casetype` is a vector of zeroes the same size as the variable `range`.

`[latfix,lonfix] = navfix(lat,long,az_range,casetype)` combines ranges and bearings. For each element of `casetype` equal to 1, the corresponding element of `az_range` represents an azimuth to the associated landmark. Where `casetype` is a 0, `az_range` is a range.

`[latfix,lonfix] = navfix(lat,long,az_range,casetype,drlat,drlon)` returns for each possible pairing of landmarks only that intersection which lies closest to the dead reckoning position indicated by `drlat` and `drlon`. When this syntax is used, all included landmarks' bearing lines or range arc must intersect. If any possible pairing fails, the warning `No Fix` is displayed.

Note: The outputs of this function are matrices providing the locations of the intersections for all possible pairings of the *n* entered lines of bearing and range arcs. These matrices will therefore have *n-choose-2* rows. In order to allow for two intersections per combination, these matrices have two columns. Whenever there are fewer than two intersections for that combination, one or two `NaN`s are returned in that row.

When a dead reckoning position is included, these matrices will be column vectors.

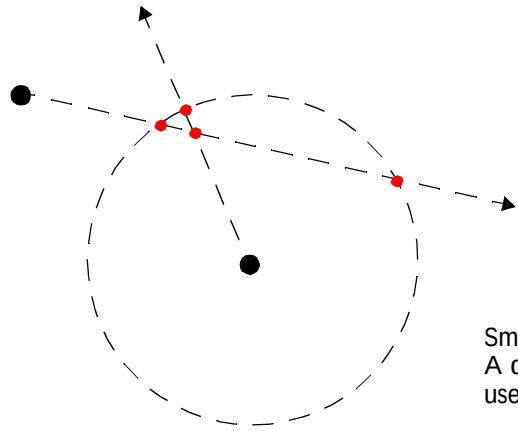
Examples

For a fully illustrated example of the application of this function, please refer to the "Navigation" section in Chapter 5 of the *Mapping Toolbox User's Guide*.

Imagine you have two landmarks, at (15°N,30.4°W) and (14.8°N,30.1°W). You have a visual bearing to the first of 280° and to the second of 160°. Additionally, you have a range to the second of 12 nm. Find the intersection points:

```
[latfix,lonfix] = navfix([15 14.8 14.8],[-30.4 -30.1 -30.1],...
                         [280 160 12],[1 1 0])
latfix =
   14.9591       NaN
   14.9680   14.9208
   14.9879       NaN
lonfix =
  -30.1599       NaN
  -30.2121  -29.9352
  -30.1708       NaN
```

Here is an illustration of the geometry:



Small dots are the intersection points. A dead reckoning position could be used to eliminate the inconsistent point.

Limitations

Traditional plotting and the `navfix` command are limited to relatively short distances. Visual bearings are in fact great circle azimuths, not rhumb lines, and range arcs are actually arcs of small circles, not of the planar circles plotted on the chart. However, the mechanical ease of the process and the practical limits of visual bearing- and navigational radar-ranges (~ 30 nm) make this limitation moot in practice. The error contributed due to these assumptions is minuscule at that scale.

See Also

`crossfix`	Great circle fixing

`gcxgc`
`gcxsc`
`scxsc`
`rhxrh`
	Other intersection functions

Purpose

Describe waypoints of a navigational track

Description

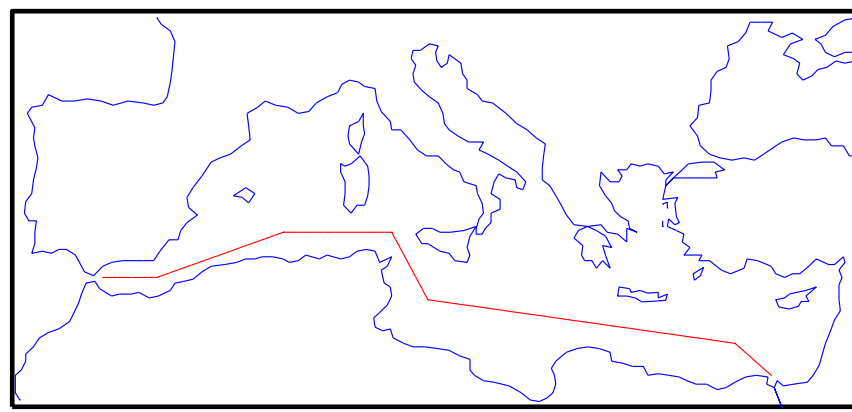
Track waypoints put into latitude and longitude column-vectors are said to be in *navigational track format*. *Waypoints* are points through which a track passes, usually corresponding to course or speed changes. A navigational track is made up of the line segments connecting these waypoints, which are called *legs*. In navigational track format, *n* legs are described using *n+1* waypoints.

For Mapping Toolbox navigation functions, angle units are always in degrees. See the section entitled “Conventions for Navigational Functions” in Chapter 5 of the *Mapping Toolbox User's Guide* for more information on unit formats for navigation functions.

Examples

The waypoints of the following track, in navigational track format, are:

```
waypoints = [36 -5; 36 -2; 38 5; 38 11; 35 13; 33 30; 31.5 32]
waypoints =
   36.0000   -5.0000
   36.0000   -2.0000
   38.0000    5.0000
   38.0000   11.0000
   35.0000   13.0000
   33.0000   30.0000
   31.5000   32.0000
```



See Also

`dreckon`	Compute dead reckoning positions for a track
`gcwaypts`	Find equally-spaced waypoints along a great circle
`legs`	Find courses and distances between waypoints
`navfix`	Mercator-based navigational fixing
`track`	Connect navigational waypoints with track segments

Purpose

Transform regular matrix map to new coordinate system based on a new origin

Syntax

```
[map,lat,lon] = neworig(map0,maplegend,origin)
[map,lat,lon] = neworig(map0,maplegend,origin,direction)
[map,lat,lon] = neworig(map0,maplegend,origin,direction,units)
```

Description

This command will transform a regular matrix map into a new matrix in an altered coordinate system. An analytical use of the new matrix can be realized in conjunction with the `newpole` command. If a selected point is made the *north pole* of the new system, then when a new matrix is created with `neworig`, each row of the new matrix is constant distance from the selected point, and each column is constant azimuth from that point.

`[map,lat,lon] = neworig(map0,maplegend,origin)` returns the data in the original regular matrix map, `map0`, with its three-element map legend vector `maplegend`, reallocated to the cells of the new (same-sized) matrix `map`. This transformation is governed by the input `origin`. This is a three- (or two-) element vector of the form `[latitude longitude orientation]`. The latitude and longitude are the coordinates of the point in the original system that is the center of the output system. The orientation is the azimuth from the new origin point to the original North Pole in the new system. If `origin` has only two elements, the orientation is assumed to be 0°. This `origin` vector might be the output of `putpole` or `newpole`. The outputs `lat` and `lon` are matrices the size of `map` which give a cell-by-cell registration of `map` to the coordinates of the original (`map0`) system in latitude and longitude, respectively.

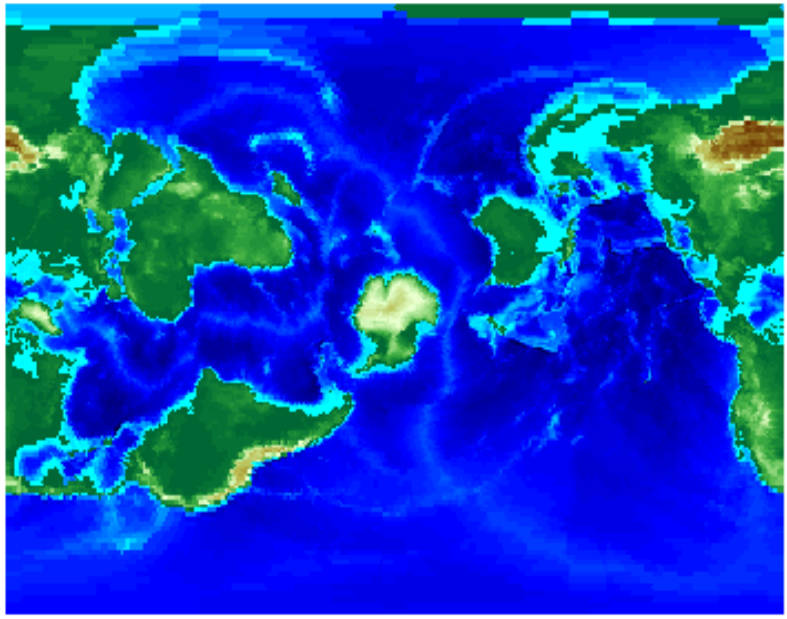
`[map,lat,lon] = neworig(map0,maplegend,origin,direction)` allows the specification of the operation. If the string *direction* is `'forward'` (the default), the transformation occurs as described above. If the *direction* is `'inverse'`, then the output `map` is the *original* system from which a transformed matrix `map0` was derived, via the input `origin`. Note that if the matrix `map1` is transformed forward to `map2`, and `map2` is transformed inversely to `map3`, `map3` will look very much like `map1`, but the two matrices will not be identical. This is because neworig is in fact projecting the values of the cells twice, rather than *undoing* the first transformation, and matrix data has granularity to it.

`[map,lat,lon] = neworig(map0,maplegend,origin,direction,units)` allows the specification of the angular units of the origin vector, where *units* is any valid angle units string. The default is `'degrees'`.

Examples

This is the topo map transformed to put Sri Lanka at the North Pole:

```
load topo
origin = newpole(7,80)
origin =
   83.0000 -100.0000         0
[map,lat,lon] = neworig(topo,topolegend,origin);

axesm miller
surfm(map,[30 30])
demcmap(topo)
```



See Also

`newpole`	Select point to place at North Pole
`org2pol`	Pole of transformed coordinate system
`putpole`	Origin of transformed coordinate system
`rotatem`	Transform vector data to new coordinate system based on a new origin

newpole

Purpose

Compute origin of a transformed coordinate system based on a new pole

Syntax

```
origin = newpole(polelat,polelon)
origin = newpole(polelat,polelon,units)
```

Description

When developing transverse or oblique projections, you need transformed coordinate systems. One way to define these systems is to establish which point in the original (untransformed) system will become the new (transformed) *north pole*.

`origin = newpole(polelat,polelon)` provides the `origin` vector for a transformed coordinate system based upon moving the point (`polelat`, `polelon`) to become the north pole singularity in the new system. The origin is a three-element vector of the form `[latitude longitude orientation]`, where the latitude and longitude are the coordinates the new center (origin) had in the untransformed system, and the orientation is the azimuth of the true North Pole from the new origin point. For the `newpole` calculation, this orientation is constrained to be always 0°.

`origin = newpole(polelat,polelon,units)` specifies the units of the inputs and output, where *units* is any valid angle units string. The default is `'degrees'`.

Examples

Let's take a point and make it the new north pole:

```
origin = newpole(60,180)
origin =
   30.0000         0         0
```

This makes sense: as a point 30° beyond the true North Pole on the original origin's meridian is pulled up to become the *pole*, the point originally 30° above the origin is pulled down into the origin spot.

See Also

`neworig`	Transform regular matrix map to new coordinate system
`org2pol`	Pole of transformed coordinate system
`putpole`	Origin of transformed coordinate system

Purpose

Convert distance from nautical miles to other units

Syntax

```
distout = nm2deg(distin)
distout = nm2deg(distin,radius)

distout = nm2km(distin)

distout = nm2rad(distin)
distout = nm2rad(distin,radius)

distout = nm2sm(distin)
```

Description

`distout = nm2deg(distin)` converts the input distance given in nautical miles to degrees. In addition, `distout = nm2km(distin)`, `distout = nm2rad(distin)` and `distout = nm2sm(distin)` perform analogously, converting to kilometers, radians, and statute miles, respectively.

`distout = nm2deg(distin,radius)`and `distout = nm2rad(distin,radius)` specify the radius of the sphere to use, since a degree (or radian) of arc length covers less distance, for example, on Mars than it does on the Earth. You can enter the radius as a number in nautical miles, as a call to the `almanac` function (e.g., `almanac('mars','radius','nm')`), or you can pass in a string planet name (e.g., `'mars'`), and the function will make the appropriate call to the `almanac` function. The radius of the Earth is the default.

Examples

How fast is 30 knots (nautical miles per hour) in kph?

```
distout = nm2km(30)
distout =
   55.5600
```

See Also

`distdim`	Convert distances between different units
`km2sm` `sm2deg`	Other direct distance conversion functions

npi2pi

Purpose

Convert normalize angles to lie between –π and π

Syntax

```
anglout = npi2pi(anglin)
anglout = npi2pi(anglin,units)
anglout = npi2pi(anglin,units,approach)
```

Description

`anglout = npi2pi(anglin)` *wraps* the input angle `anglin` to lie on the range –180 to 180 (e.g., 270° is renamed –90°).

`anglout = npi2pi(anglin,units)` specifies the angle units with any valid angle units string *units*. The default is `'degrees'`.

`anglout = npi2pi(anglin,units,approach)` specifies the approach logic for this wrapping. The *approach* string `'exact'` calculates a mathematically precise wrap. `'inward'` and `'outward'` calculate more quickly by shifting the values by an *epsilon* either toward or away from the origin and performing a trigonometric wrap. The trigonometric wrap is inexact to allow for the fact that different computer math processors might give different (although trigonometrically identical) results (180° or –180°, for example). The offset prevents this.

Examples

```
npi2pi(315)
ans =
   -45
npi2pi(181)
ans =
  -179
```

See Also

`zero22pi` Normalize angles to lie between 0 and 2π

Purpose

Create matrix maps of ones

Syntax

```
map = onem(latlim,lonlim,scale)
[map,maplegend] = onem(latlim,lonlim,scale)
```

Description

`map = onem(latlim,lonlim,scale)` returns a regular matrix map consisting entirely of ones. The two-element vectors `latlim` and `lonlim` define the latitude and longitude limits of the geographic region. They should be of the form `[south north]` and `[west east]`, respectively. The number of rows and columns per angle unit is set by the scalar value `scale`.

`[map,maplegend] = onem(latlim,lonlim,scale)` returns the three-element map legend vector for the returned `map`.

Examples

```
[map,maplegend] = onem([46,51],[-79,-75],1)
map =
     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1
maplegend =
     1    51   -79
```

See Also

`limitm`	Matrix map limits
`map legend vector`	Data structure for regular matrix maps
`nanm`	Create a matrix map of `NaN`s
`sizem`	Row and column dimensions needed for map
`spzerom`	Create a sparse matrix map of zeros
`zerom`	Create a full matrix map of zeros

org2pol

Purpose

Compute pole of a transformed coordinate system based on a new origin

Syntax

```
pole = org2pol(origin)
pole = org2pol(origin,units)
```

Description

When developing transverse or oblique projections, transformed coordinate systems are required. One way to define these systems is to establish the point at which, in terms of the original (untransformed) system, the (transformed) true North Pole will lie.

`pole = org2pol(origin)` returns the location of the North Pole in terms of the coordinate system after transformation based on the input `origin`. The `origin` is a three-element vector of the form `[latitude longitude orientation]`, where `latitude` and `longitude` are the coordinates that the new center (origin) had in the untransformed system, and `orientation` is the azimuth of the true North Pole from the new origin point in the transformed system. The output `pole` is a three-element vector of the form `[latitude longitude meridian]`, which gives the latitude and longitude point in terms of the original untransformed system of the new location of the true North Pole. The meridian is the longitude from the original system upon which the new system is centered.

`pole = org2pol(origin,units)` allows the specification of the angular units of the origin vector, where *units* is any valid angle units string. The default is `'degrees'`.

Examples

Say we want to make (30°N,0°) the new origin. Where will the North Pole end up in terms of the original coordinate system?

```
pole = org2pol([30 0 0])
pole =
   60.0000         0         0
```

This makes sense: pull a point 30° down to the origin, and the North Pole gets pulled down 30°. A little less obvious example is the following:

```
pole = org2pol([5 40 30])
pole =
   59.6245   80.0750   40.0000
```

See Also

`neworig`	Transform regular matrix map to new coordinate system
`putpole`	Origin of transformed coordinate system based on a new pole

Purpose

Figure paper size for a given map scale

Syntax

```
paperscale(paperdist,punits,surfdist,sunits)
paperscale(paperdist,punits,surfdist,sunits,lat,long)
paperscale(paperdist,punits,surfdist,sunits,lat,long,az)
paperscale(paperdist,punits,surfdist,sunits,lat,long,az,gunits)
paperscale(paperdist,punits,surfdist,sunits,lat,long,az,
   gunits,radius)
paperscale(scale,...)
[paperXdim,paperYdim] = paperscale(...)
```

Background

Maps are usually printed at a size that allows an easy comparison of distances measured on paper to distances on the Earth. The relationship of geographic distance and paper distance is termed *scale*. It is usually expressed as a ratio, such as 1 to 100,000 or 1:100,000 or 1 cm = 1 km.

Description

`paperscale(paperdist,punits,surfdist,sunits)` sets the figure paper position to print the map in the current axes at the desired scale. The scale is described by the geographic distance which corresponds to a paper distance. For example, a scale of 1 inch = 10 kilometer is specified as `degrees(1,'inch',10,'km')`. See below for an alternate method of specifying the map scale. The surface distance units string *sunits* can be any string recognized by `distdim`. The paper units string can be any dimensional units string recognized for the figure `PaperUnits` property.

`paperscale(paperdist,punits,surfdist,sunits,lat,long)` sets the paper position so that the scale is correct at the specified geographic location. If omitted, the default is the center of the map limits.

`paperscale(paperdist,punits,surfdist,sunits,lat,long,az)` also specifies the direction along which the scale is correct. If omitted, 90 degrees (east) is assumed.

`paperscale(paperdist,punits,surfdist,sunits,lat,long,az,gunits)` also specifies the units in which the geographic position and direction are given. If omitted, `'degrees'` is assumed.

`paperscale(paperdist,`*`punits`*`,surfdist,`*`sunits`*`,lat,long,az,`*`gunits`*`,radius)` uses the last input to determine the radius of the sphere. If `radius` is a string, then it is evaluated as an `almanac` body to determine the spherical radius. If numerical, it is the radius of the desired sphere in the same units as the surface distance. If omitted, the default radius of the Earth is used.

`paperscale(scale,...)`, where the numeric scale replaces the two property/value pairs, specifies the scale as a ratio between distance on the sphere and on paper. This is commonly notated on maps as 1:scale (e.g. 1:100 000, or 1:1 000 000). For example, `paperscale(100000)` or `paperscale(100000,lat,long)`.

`[paperXdim,paperYdim] = paperscale(...)` returns the computed paper dimensions. The dimensions are in the paper units specified. For the scale calling form, the returned dimensions are in centimeters.

Examples

The small circle measures 10 cm across when printed.

```
axesm mercator
[lat,lon] = scircle1(0,0,km2deg(5));
plotm(lat,lon)
[x,y] = paperscale(1,'centimeter',1,'km'); [x y]
ans =
       13.154       12.509

set(gca,'pos', [ 0 0 1 1])
[x,y] = paperscale(1,'centimeter',1,'km'); [x y]
ans =
       10.195       10.195
```

Limitations

The relationship between the paper and geographic coordinates holds only as long as there are no changes to the display that affect the axes limits or the relationship between geographic coordinates and projected coordinates. Changes of this type include the geoid or scale factor properties of the map axes, or adding elements to the display that cause MATLAB to modify the axes autoscaling. To be sure that the scale is correct, execute `paperscale` just before printing.

See Also

`pagedlg`	Page position dialog box
`axesscale`	Resize axes for equivalent scale
`daspectm`	Figure DataAspectRatio property for a map

par2geod

Purpose

Convert from parametric to geodetic latitudes

Syntax

```
lat = par2geod(lat0)
lat = par2geod(lat0,geoid)
lat = par2geod(lat,units)
lat = par2geod(lat,geoid,units)
```

Background

Longitudes have the same meaning whether the Earth is treated as a sphere or as an ellipsoid. However, the definition of latitude in the case of an ellipsoid is more complicated. Geodetic, or geographic, latitudes on an ellipsoid can be transformed from latitudes on an auxiliary sphere with certain properties.

Geodetic latitude: (also called *geographic latitude*) the angle a normal line passing through a surface point makes with the plane of the Equator.

Parametric latitude: latitudes on the set of spheres with parallel radii identical to the ellipsoidal parallel radii at every latitude.

Description

`lat = par2geod(lat0)` returns the parametric latitudes provided in `lat0` transformed to geodetic latitudes, which are returned in `lat`.

`lat = par2geod(lat0,geoid)` defines the elliptical model of the Earth, given by the geoid vector `geoid`, to which `lat0` is transformed. The default geoid is the same as the default of `almanac('earth','geoid')`, the 1980 Geodetic Reference System ellipsoid.

`lat = par2geod(lat,geoid,units)` defines the angle units of the inputs and outputs, where *units* is any valid angle units string. The default is `'degrees'`.

Examples

```
lat = par2geod([0 45 90])
lat =
         0   45.0962   90.0000
```

See Also

`almanac`	Planetary data
`geod2par` `iso2geod`	Other auxiliary latitude functions

Purpose

Project patches onto the current map axes as separate objects

Syntax

```
h = patchesm(lat,lon,cdata)
h = patchesm(lat,lon,PropertyName,PropertyValue,...)
h = patchesm(lat,lon,cdata,PropertyName,PropertyValue,...)
h = patchesm(lat,lon,z,cdata)
h = patchesm(lat,lon,z,PropertyName,PropertyValue,...)
h = patchesm(lat,lon,z,cdata,PropertyName,PropertyValue,...)
```

Description

This command is very similar to the `patchm` command. The significant difference is that in `patchesm`, separate patches (delineated by `NaN`s in the inputs `lat` and `lon`) are separated and plotted as distinct patch objects on the current map axes. The advantage to this is that less memory is required. The disadvantage is that multifaced objects cannot be treated as a single object. For example, the archipelago of the Philippines cannot be treated and handled as a single handle graphics object.

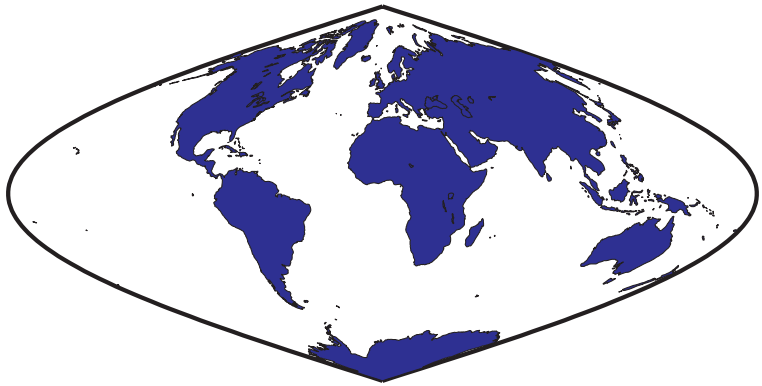
`h = patchesm(lat,lon,cdata)` projects and displays patch (polygon) objects defined by their vertices given in `lat` and `lon` on the current map axes. `lat` and `lon` must be vectors. The color data, `cdata`, can be any color data designation supported by the standard MATLAB `patch` command. The object handle or handles, `h`, can be returned.

`h = patchesm(lat,lon,PropertyName,PropertyValue,...)` allows any property/value pair supported by `patch` to be assigned to the `patchesm` objects.

`h = patchesm(lat,lon,z,cdata)` allows the assignment of an altitude, `z`, to each patch object. The default altitude is `z = 0`.

Examples

```
load coast
axesm sinusoid; framem
h = patchesm(lat,long,'b');
```



```
length(h)
ans =
   238
```

See Also

`patchm`	Project patch objects on the current map axes
`fill3m`	Project 3-D patch objects onto the current map axes
`fillm`	Project 2-D patch objects onto the current map axes

Purpose

Project patch objects onto the current map axes

Syntax

```
h = patchm(lat,lon,cdata)
h = patchm(lat,lon,PropertyName,PropertyValue,...)
h = patchm(lat,lon,cdata,PropertyName,PropertyValue,...)
h = patchm(lat,lon,z,cdata)
h = patchm(lat,lon,z,PropertyName,PropertyValue,...)
h = patchm(lat,lon,z,cdata,PropertyName,PropertyValue,...)
```

Description

This Mapping Toolbox command is very similar to the standard MATLAB `patch` command. Like its analog, and unlike higher level functions such as `fillm` and `fill3m`, `patchm` will add patch objects to the current map axes regardless of hold state. Except for `XData`, `YData`, and `ZData`, all line properties and styles available through `patch` are supported by `patchm`.

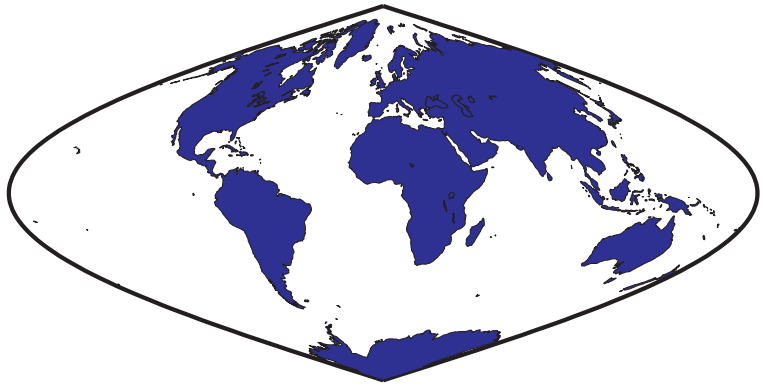
`h = patchm(lat,lon,cdata)` projects and displays patch (polygon) objects defined by their vertices given in `lat` and `lon` on the current map axes. `lat` and `lon` must be vectors. The color data, `cdata`, can be any color data designation supported by the standard MATLAB `patch` command. The object handle or handles, `h`, can be returned.

`h = patchm(lat,lon,PropertyName,PropertyValue,...)` allows any property/value pair supported by `patch` to be assigned to the `patchm` object.

`h = patchm(lat,lon,z,cdata)` allows the assignment of an altitude, `z`, to each patch object. The default altitude is `z = 0`.

Examples

```
load coast
axesm sinusoid; framem
h = patchm(lat,long,'b');
```



```
length(h)
ans =
     1
```

See Also

patchesm	Project patches as separate objects
fill3m	Project 3-D patch objects onto the current map axes
fillm	Project 2-D patch objects onto the current map axes

Purpose

Project matrix map in the `z = 0` plane

Syntax

```
h = pcolorm(map)
h = pcolorm(map,npts)
h = pcolorm(lat,lon,map)
h = pcolorm(lat,lon,map,PropertyName,PropertyValue,...)
```

Description

This command warps a matrix map to a graticule mesh, which itself is projected according to the map axes property `MapProjection`. The fineness, or resolution of this grid determines the quality of the projection and the speed of plotting it. There is no hard and fast rule for sufficient graticule resolution, but in general, cylindrical projections need very few graticules points in the longitudinal direction, while complex curve-generating projections require more.

`h = pcolorm(map)` projects the matrix map, `map`, on a graticule grid the size of `map` between the latitude and longitude limits of the current map axes. The handle `h` of the displayed surface can be returned.

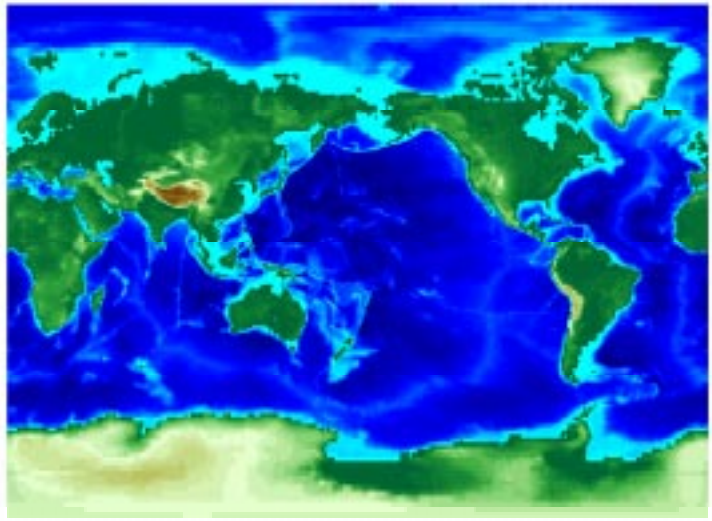
`h = pcolorm(map,npts)` results in a graticule grid defined by `npts`, which is a two-element vector of the form `[latitude-points longitude-points]`. The default `npts` is `[50 100]`.

`h = pcolorm(lat,lon,map)` allows three other methods of defining the graticule grid. If `lat` and `lon` are matrices, they represent the actual graticule vertices as might be returned by `meshgrat`. If vectors, they are the representative coordinates of the rows and columns, respectively, of such a grid. If they are two-element vectors, they are treated as latitude and longitude limits, and a graticule mesh the size of the default `npts` is calculated.

`h = pcolorm(lat,lon,map,PropertyName,PropertyValue,...)` allows the input of property/value pairs to control the surface object properties. Any property supported by the standard MATLAB command `surface` except `XData`, `YData`, and `ZData` can be altered in this manner.

Examples

```
load topo
axesm miller
pcolorm(topo,[30 30])
demcmap(topo)
```



See Also

`meshgrat`	Construct map graticule grid
`meshm`	Regular matrix map warped to projected graticule mesh
`surfacem`	Matrix map warped to projected graticule mesh
`surfm`	Matrix map projected a map axes

Purpose

Project parallel labels on a map axes

Syntax

```
plabel
plabel('on')
plabel('off')
plabel('reset')
plabel(meridian)
h = plabel(MapAxesPropertyName,PropertyValue,...)
```

Description

`plabel` will toggle the visibility of parallel labelling on the current map axes.

`plabel('on')` will set the visibility of parallel labels to `'on'`.

`plabel('off')` will set the visibility of parallel labels to `'off'`.

`plabel('reset')` will reset the displayed parallel labels using the currently defined parallel label properties.

`plabel(meridian)` sets the value of the `PLabelMeridian` property of the map axes to the value meridian. This determines the meridian upon which the labels are placed (see `axesm`). The options for `meridian` are a scalar longitude, or the strings `'east'`, `'west'`, or `'prime'`.

`plabel(MapAxesPropertyName,PropertyValue,...)` allows paired map axes property names and property values to be passed in. For a complete description of map axes properties, see the `axesm` reference page in this guide.

Parallel label handles can be returned in `h` if desired.

See Also

`axesm`	Define map axes and set map properties
`setm`	Set map properties
`mlabel`	Meridian labels projected on a map axes

plot3m

Purpose

Project line objects onto current map axes in 3-D space

Syntax

```
h = plot3m(lat,lon,z)
h = plot3m(lat,lon,z,linetype)
h = plot3m(lat,lon,z,PropertyName,PropertyValue,...)
```

Description

`plot3m` is the mapping equivalent of the MATLAB `plot3` function.

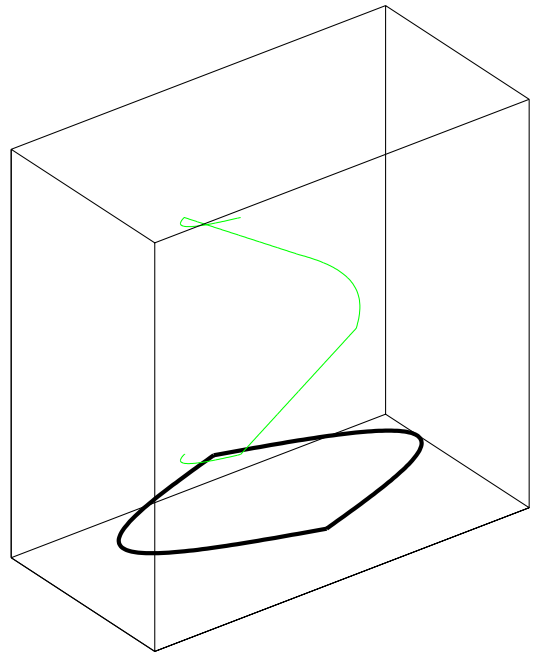
`h = plot3m(lat,lon,z)` displays projected line objects on the current map axes. `lat` and `lon` are the latitude and longitude coordinates, respectively, of the line object to be projected. Note that this ordering is conceptually reversed from the MATLAB `line` command, because the *vertical* (*y*) coordinate comes first. However, the ordering latitude, then longitude, is standard geographic usage. `lat` and `lon` must be the same size, and in the `AngleUnits` of the map axes. `z` is the altitude data associated with each point in `lat` and `lon`. The object handle for the displayed line can be returned in `h`.

`h = plot3m(lat,lon,linetype)` allows the specification of the line style, where *linetype* is any string recognized by the MATLAB `line` command.

`h = plot3m(lat,lon,PropertyName,PropertyValue,...)` allows the specification of any number of property/value pairs for any properties recognized by the MATLAB `line` command except for `XData`, `YData`, and `ZData`.

Examples

```
axesm sinusoid; framem; view(3)
[lats,longs] = interpm([45 -45 -45 45 45 -45]',...
                       [-100 -100 100 100 -100 -100]',1);
z = (1:671)'/100;
plot3m(lats,longs,z,'g')
```



See Also

`linem`	Project line objects onto current map axes.
`plot3`	Plot lines and points in 3-D space (see online *MATLAB Function Reference*)
`plotm`	Project lines onto current map axes in 2-D space

plotm

Purpose

Project 2-D lines onto current map axes

Syntax

```
h = plotm(lat,lon)
h = plotm(lat,lon,linetype)
h = plotm(lat,lon,PropertyName,PropertyValue,...)
h = plotm([lat lon],...)
```

Description

`plotm` is the mapping equivalent of the MATLAB `plot` function.

`h = plotm(lat,lon)` displays projected line objects on the current map axes. `lat` and `lon` are the latitude and longitude coordinates, respectively, of the line object to be projected. Note that this ordering is conceptually reversed from the MATLAB `line` command, because the *vertical* (*y*) coordinate comes first. However, the ordering latitude, then longitude, is standard geographic usage. `lat` and `lon` must be the same size, and in the `AngleUnits` of the map axes. The object handle for the displayed line can be returned in `h`.

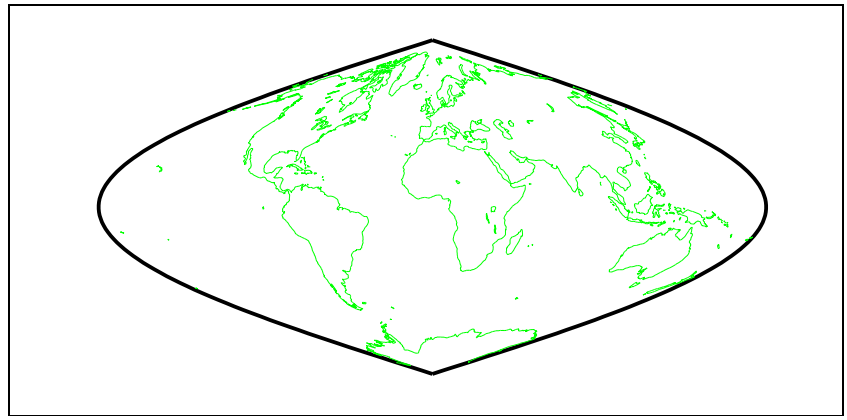
`h = plotm(lat,lon,linetype)` allows the specification of the line style, where *linetype* is any string recognized by the MATLAB `line` command.

`h = plotm(lat,lon,PropertyName,PropertyValue,...)` allows the specification of any number of property/value pairs for any properties recognized by the MATLAB `line` command except for `XData`, `YData`, and `ZData`.

`h = plotm([lat lon],...)` allows the coordinates to be packed into a single two-column matrix.

Examples

```
load coast
axesm sinusoid; framem
plotm(lat,long,'g')
```



See Also

`linem`	Project line objects onto current map axes.
`plot`	Linear plot (see online *MATLAB Function Reference*)
`plot3m`	Project lines onto current map axes in 3-D space

polcmap

Purpose

Colormap for political maps

Syntax

```
polcmap
polcmap(ncolors)
polcmap(ncolors,maxsat)
polcmap(ncolors,huelimits,saturationlimits,valuelimits)
cmap = polcmap(...)
```

Description

`polcmap` applies a random muted colormap to the current figure. The size of the colormap is the same as the existing colormap.

`polcmap(ncolors)` creates a colormap with the specified number of colors

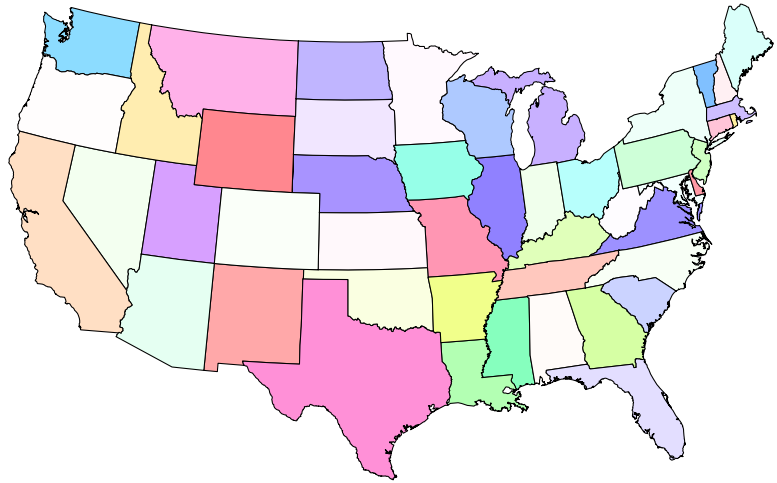
`polcmap(ncolors,maxsat)` controls the maximum saturation of the colors. Larger maximum saturation values produce brighter, more saturated colors. If omitted, the default is 0.5.

`polcmap(ncolors,huelimits,saturationlimits,valuelimits)` controls the colors. Hue, saturation and value are randomly selected values within the limit vectors. These are two-element vector of the form `[min max]`. Valid values range from 0 to 1. As the hue varies from 0 to 1, the resulting color varies from red, through yellow, green, cyan, blue and magenta, back to red. When the saturation is 0, the colors are unsaturated; they are simply shades of gray. When the saturation is 1, the colors are fully saturated; they contain no white component. As the value varies from 0 to 1, the brightness increases.

`cmap = polcmap(...)` returns the colormap without applying it to the figure.

Examples

```
usamap('conus','none')
framem off; gridm off; mlabel off; plabel off

load usalo
displaym(state)
polcmap
```



See Also

`demcmap`	Colormaps for Digital Elevation Maps
`colormap`	Color look-up table

previewmap

Purpose

View map at printed size

Syntax

```
previewmap
```

Background

The appearance of a map on screen may differ from the final printed output. This results from the difference in the size and shape of the figure window and the area the figure occupies on the printed page. A map that appears readable on screen may be cluttered when the printed output is smaller. Likewise, the relative position of multiple axes may appear different when printed. This functions resizes the figure to the printed size

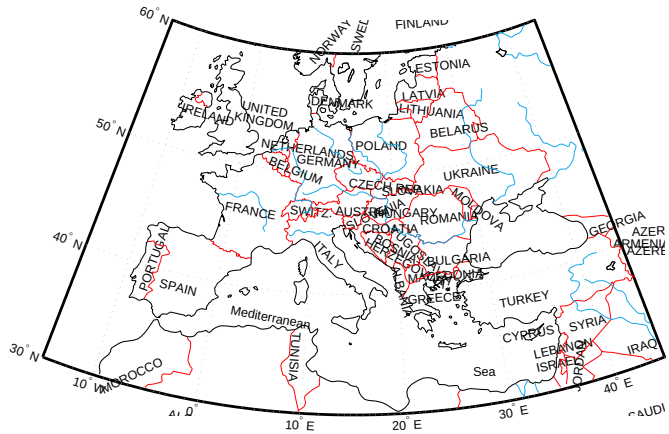
Description

`previewmap` changes the size of the current figure to match the printed output. If the resulting figure size exceeds the screen size, the figure will be enlarged as much as possible.

Examples

Is the text small enough to avoid overlapping in a map of Europe?

```
worldmap europe
h = displaym(worldlo('POtext'));
trimcart(h)
rotatetext(h)

orient landscape
tightmap
hidem(gca)
previewmap
```



Limitations

The figure cannot be made larger than the screen.

See Also

`pagedlg`	Page position dialog box
`paperscale`	Figure paper size for a given map scale
`axesscale`	Resize axes for equivalent scale

project

Purpose

Project a displayed graphics object on a map axes

Syntax

```
project(h)
project(h,'xy')
project(h,'yx')
```

Description

`project(h)` takes unprojected objects with handles `h` that are displayed on map axes and projects them. For example, `project` will take a line created on a map axes with the `plot` command and project it as though it had been created with the `plotm` command. This can be useful if a standard MATLAB command was accidentally executed. The map structure of the existing map axes will determine the specifics of the projection. If `h` is the handle of the map axes, then all the children of `h` will be projected. This should not be attempted if any children of `h` have already been projected!

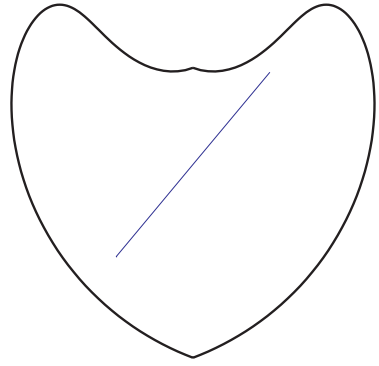
`project(h,'xy')` specifies that the `XData` of the unprojected objects correspond to longitudes and the `YData` to latitudes. This is the default assumption.

`project(h,'yx')` specifies that the `XData` of the unprojected objects correspond to latitudes and the `YData` to longitudes.

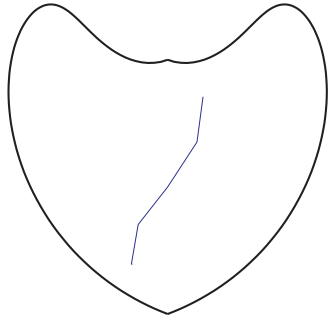
Examples

Create an axes, plot a line, then project it:

```
axesm('bonne','AngleUnits','radians');framem;
h = plot([-1 -.5 0 .5 1],[-1 -.5  0 .5 1]);
```



```
project(h)
```



The line is straight in *x-y* space, but when converted to a projected map object, it bends with the projection.

See Also

`linem`	Project line objects
`patchm`	Project patch objects
`surfacem`	Project matrix map
`textm`	Project text objects

putpole

Purpose

Compute origin of transformed coordinate system

Syntax

```
origin = putpole(pole)
origin = putpole(pole,units)
```

Description

When developing transverse or oblique projections, you need transformed coordinate systems. One way to define these systems is to establish which point in the original (untransformed) system will become the new (transformed) origin.

`origin = putpole(pole)` returns an `origin` vector required to transform a coordinate system in such a way as to put the true North Pole at a point specified by the three- (or two-) element vector `pole`. This vector is of the form `[latitude longitude meridian]` specifying the coordinates in the original system at which the true North Pole is to be placed in the transformed system. The meridian is the longitude upon which the new system is to be center, which is the new pole longitude if omitted. The output is a three-element vector of the form `[latitude longitude orientation]`, where the latitude and longitude are the coordinates in the untransformed system of the new origin, and orientation is the azimuth of the true North Pole in the transformed system.

`origin = putpole(pole,units)` allows the specification of the angular units of the origin vector, where *units* is any valid angle units string. The default is `'degrees'`.

Examples

Pull the north pole down the 0° meridian by 30° to 60°N. What is the resulting origin vector?

```
origin = putpole([60 0])
origin =
   30.0000         0         0
```

This makes sense: when the pole slid down 30°, the point that was 30° north of the origin slid down to become the origin. A less obvious transformation:

```
origin = putpole([60 80 0])   % constrain to original central
                              % meridian
origin =
    4.9809         0   29.6217
origin = putpole([60 80 40])  % constrain to arbitrary meridian
origin =
    4.9809   40.0000   29.6217
```

See Also

`neworig`	Transform regular matrix map to new coordinate system
`org2pol`	Pole of a transformed coordinate system based on a new origin

quiver3m

Purpose

Project three-dimensional quiver plot on map axes

Syntax

```
h = quiver3m(lat,lon,alt,u,v,w)
h = quiver3m(lat,lon,alt,u,v,w,linespec)
h = quiver3m(lat,lon,alt,u,v,w,linespec,'filled')
h = quiver3m(lat,lon,alt,u,v,w,scale)
h = quiver3m(lat,lon,alt,u,v,w,linespec,scale)
h = quiver3m(lat,lon,alt,u,v,w,linespec,scale,'filled')
```

Description

`h = quiver3m(lat,lon,alt,u,v,w)` displays *velocity* vectors with components `(u,v,w)` at the geographic points `(lat,lon)` and altitude `alt` on a displayed map axes. The inputs `u`, `v`, and `w` determine the direction of the vectors in latitude, longitude, and altitude, respectively. The function automatically determines the length of these vectors to make them as long as possible without overlap. The object handles of the displayed vectors can be returned in `h`.

`h = quiver3m(lat,lon,alt,u,v,w,linespec)` allows the control of the line specification of the displayed vectors with a *linespec* string recognized by the MATLAB `line` command. If symbols are indicated in *linespec*, they are plotted at the start points of the vectors, i.e., the input points `(lat,lon,alt)`.

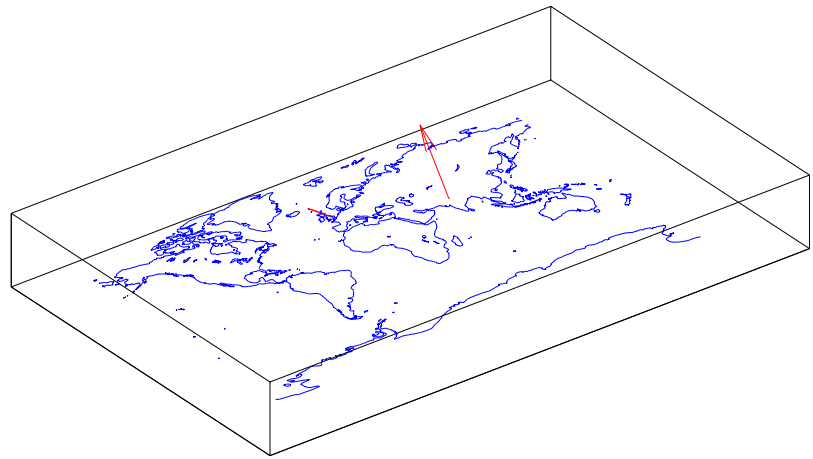
`h = quiver3m(lat,lon,alt,u,v,w,linespec,'filled')` results in any symbols specified by *linespec* being filled in.

`h = quiver3m(lat,lon,alt,u,v,w,scale)` alters the automatically calculated vector lengths by multiplying them by the scalar value `scale`. For example, if `scale` is 2, the displayed vectors are twice as long as they would be if `scale` were 1 (the default). When scale is set to 0, the automatic scaling is suppressed and the length of the vectors is determined by the inputs. In this case, the vectors are plotted from `(lat,lon,alt)` to `(lat+u,lon+v,alt+w)`.

Examples

Plot 3-D quiver vectors from London (51.5°N,0°) and New Delhi (29N,77.5°E), both at an altitude of 0. Suppress the automatic scaling. Terminate both vectors at an altitude of 1; the London vector should terminate 100° southward and 70° eastward, while the New Delhi vector should terminate 50° northward and 10° eastward.

```
load coast
axesm miller; view(3)
plotm(lat,long)
lat0 = [51.5,29]; lon0 = [0 77.5]; alt = [0 0];
u = [-40 50]; v = [-70 10]; w = [1 1];
quiver3m(lat0,lon0,alt,u,v,w,'m')
```



See Also

`quiverm`	Two-dimensional quiver plot projected on map axes
`quiver3`	Three-dimensional velocity plot (see online *MATLAB Function Reference*)

quiverm

Purpose

Project two-dimensional quiver plot on map axes

Syntax

```
h = quiverm(lat,lon,u,v)
h = quiverm(lat,lon,u,v,linespec)
h = quiverm(lat,lon,u,v,scale)
h = quiverm(lat,lon,u,v,linespec,'filled')
h = quiverm(lat,lon,u,v,linespec,scale)
h = quiverm(lat,lon,u,v,linespec,scale,'filled')
```

Description

`h = quiverm(lat,lon,u,v)` displays *velocity* vectors with components `(u,v)` at the geographic points `(lat,lon)` on displayed map axes. All four inputs should be in the `AngleUnits` of the map axes. The inputs `u` and `v` determine the direction of the vectors in latitude and longitude, respectively. The function automatically determines the length of these vectors to make them as long as possible without overlap. The object handles of the displayed vectors can be returned in `h`.

`h = quiverm(lat,lon,u,v,linespec)` allows the control of the line specification of the displayed vectors with a *linespec* string recognized by the MATLAB `line` command. If symbols are indicated in *linespec*, they are plotted at the start points of the vectors, i.e., the input points `(lat,lon)`.

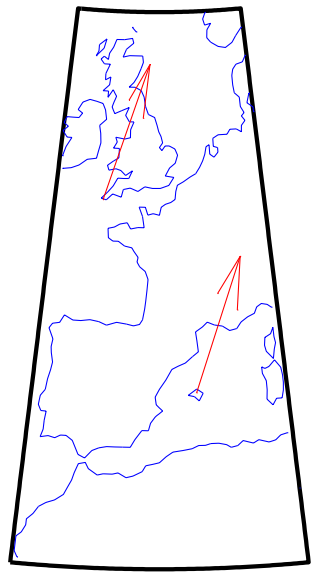
`h = quiverm(lat,lon,u,v,linespec,'filled')` results in any symbols specified by *linespec* being filled in.

`h = quiverm(lat,lon,u,v,scale)` alters the automatically calculated vector lengths by multiplying them by the scalar value `scale`. For example, if `scale` is 2, the displayed vectors are twice as long as they would be if `scale` were 1 (the default). When scale is set to 0, the automatic scaling is suppressed, and the length of the vectors is determined by the inputs. In this case, the vectors are plotted from `(lat,lon)` to `(lat+u,lon+v)`.

Examples

Plot quiver vectors from Land's End (50°N,5.4°W) and Majorca (39.7°N,2.9°E) in a direction corresponding to +5° latitude and +3° longitude. Use automatic scaling.

```
load coast
axesm('eqacon','MapLatLimit',[30 60],'MapLonLimit',[-10 10])
framem; plotm(lat,long)
lat0 = [50 39.7]; lon0 = [-5.4 2.9];
u = [5 5]; v = [3 3];
quiverm(lat0,lon0,u,v,'r')
```



See Also

`quiver3m`	Three-dimensional quiver plot projected on map axes
`quiver`	Quiver or velocity plot (see online *MATLAB Function Reference*)

rad2deg

Purpose

Convert angle (or distance) units from radians to degrees

Syntax

```
anglout = rad2deg(anglin)
```

Description

`anglout = rad2deg(anglin)` converts angles input in radians to the equivalent measure in degrees.

Remarks

This is both an angle conversion function and a distance conversion function, since arc length can be a measure of distance in either radians or degrees (provided the radius is known).

Example

There are 180° in π radians:

```
anglout = rad2deg(pi)
anglout =
   180
```

See Also

`angledim`	Convert angle units
`deg2dms` `dms2rad`	Other direct angle conversion functions
`deg2rad`	Convert degrees to radians
`distdim`	Convert distances between different units
`nm2km` `sm2deg`	Other direct distance conversion functions

Purpose

Convert angle units from radians to *dms* or *dm*

Syntax

```
anglout = rad2dms(anglin)
angleout = rad2dm(anglin)
```

Description

`anglout = rad2dms(anglin)` converts angles input in radians to the equivalent measure in the degrees-minutes-seconds (*dms*) format.

`angleout = rad2dm(anglin)` converts angles input in radians to the equivalent measure in the degrees-minutes (*dm*) format. This is the *dms* format, properly rounded to just degrees and minutes.

Example

```
rad2dms(1)
ans =
       5717.45

rad2dm(1)
ans =
       5718.00
```

See Also

`angledim`	Convert angle units
`deg2rad` `dms2rad`	Other direct angle conversion functions
`dms2mat`	Convert from *dms* to separated matrix components
`mat2dms`	Convert from separated matrices input to *dms*

rad2km, rad2nm, rad2sm

Purpose

Convert distance from radians to kilometers, nautical miles, or statute miles

Syntax

```
distout = rad2km(distin)
distout = deg2km(distin,radius)

distout = rad2nm(distin)
distout = rad2nm(distin,radius)

distout = rad2sm(distin)
distout = rad2sm(distin,radius)
```

Description

`distout = rad2km(distin)` converts the input distance given in radians to kilometers.

`distout = rad2nm(distin)` and `distout = rad2sm(distin)` work identically, except that the output units are nautical miles and statute miles, respectively.

`distout = rad2km(distin,radius)` specifies the radius of the sphere to use, since a radian of arc length covers less distance, for example, on Mars than it would on the Earth. You can enter the radius as a number in kilometers, as a call to the `almanac` function (e.g., `almanac('mars','radius','km')`), again in the appropriate units, or you can pass in a string planet name (e.g., `'mars'`), and the function will make the appropriate call to the `almanac` function. The radius of the Earth is the default.

For `distout = rad2nm(distin,radius)` and `distout = rad2sm(distin,radius)` make sure your input radius is in the appropriate units, or just use the planet name string.

Examples

How long is a trip around the Equator in statute miles?

```
distout = rad2sm(2*pi)
distout =
   2.4874e+04
```

How about on Jupiter?

```
distout = rad2sm(2*pi,'jupiter')
distout =
   2.7284e+05
```

See Also

`distdim`	Convert distances between different units
`nm2km` `sm2deg`	Other direct distance conversion functions
`rad2deg`	Convert radians to degrees

rc2yx

Purpose

Compute row and column indices for x- and y-coordinates

Syntax

```
[ygrat,xgrat] = rc2yx(rowgrat,colgrat,y1,x1,yperrow,xpercol)
```

Description

`[ygrat,xgrat] = rc2yx(rowgrat,colgrat,y1,x1,yperrow,xpercol)` computes y- and x-coordinates from row and column indices. `rowgrat` and `colgrat` are matrices of the same size containing integer row and column indices for a grid with the upper left corner at the Cartesian coordinates `x1` and `y1`. `yperrow` and `xpercol` are the y- and x-increments per row and column. The outputs `ygrat` and `xgrat` are the Cartesian coordinates corresponding to matrix locations `rowgrat` and `colgrat`.

Examples

```
[rowgrat,colgrat] = meshgrat(1:3,1:3)
rowgrat =
     1     1     1
     2     2     2
     3     3     3
colgrat =
     1     2     3
     1     2     3
     1     2     3

x1 = -1000; y1 = 1000; yperrow = -500; xpercol = 500;

[ygrat,xgrat] = rc2yx(rowgrat,colgrat,y1,x1,yperrow,xpercol)
ygrat =
        1000        1000        1000
         500         500         500
           0           0           0
xgrat =
       -1000        -500           0
       -1000        -500           0
       -1000        -500           0
```

Remarks

This function is used in the process of extracting projected matrix data from a file. The process generally consists of the following steps: 1) define the map projection as described in the data source documentation, 2) map desired geographic limits into projected coordinates using `mfwdtran`, 3) use `yx2rc` to determine associated row and column limits in the stored matrix, 4) extract sub-matrix using `readmtx`, 5) convert graticule of row and column indices to projected x and y coordinates using `rc2yx`, 6) convert x and y to geographic graticule using `minvtran`.

See Also

`yx2rc`	x- and y-coordinates to row and column indices
`readmtx`	Read a matrix stored in a file

rcurve

Purpose

Calculate radii of curvature on an ellipsoid

Syntax

```
r = rcurve(geoid,lat)
r = rcurve(geoid,lat,units)
r = rcurve('parallel',geoid,lat,units)
r = rcurve('meridian',geoid,lat,units)
r = rcurve('transverse',geoid,lat,units)
```

Description

`r = rcurve(geoid,lat)` or `r = rcurve('parallel',geoid,lat)` returns the parallel radius of curvature at the latitude `lat` for a given elliptical definition, where `geoid` is a two-element geoid vector. This is the radius of the small circle encompassing the ellipsoid at the given latitude. The radius is a distance in units consistent with the semimajor axis, the first element of `geoid`.

`r = rcurve(geoid,lat,units)` specifies the units of the input `lat`, where *units* is any valid angle units string. The default is `'degrees'`.

`r = rcurve('meridian',geoid,lat,units)` returns the meridianal radius, which is the radius of curvature at the latitude `lat` for the ellipse described by a meridian on the ellipsoid.

`r = rcurve('transverse',geoid,lat,units)` returns the transverse radius, which is the radius of a curve described by the intersection of the ellipsoid with a plane normal to the surface of the ellipsoid at the latitude `lat`.

Examples

The radii of curvature of the default geoid at 45°, in kilometers:

```
r = rcurve('transverse',almanac('earth','geoid','km'),45,...
           'degrees')
r =
   6.3888e+03

r = rcurve('meridian',almanac('earth','geoid','km'),45,...
           'degrees')
r =
   6.3674e+03

r = rcurve('parallel',almanac('earth','geoid','km'),45,...
           'degrees')
r =
   4.5024e+03
```

See Also

`geod2aut` `iso2geod`	Ellipsoid definition functions notation
`rsphere`	Radii of auxiliary spheres

readfields

Purpose

Read fields or records from a fixed format file.

Syntax

```
struc = readfields(fname,fstruc)
struc = readfields(fname,fstruc,recordIDs)
struc = readfields(fname,fstruc,recordIDs,mformat)
struc = readfields(fname,fstruc,recordIDs,mformat,fid)
struc = readfields(fname,fstruc,recordIDs,mformat,fid,'sparse')
```

Background

Map data is often provided as binary or ASCII files with a fixed format. Writing your own functions to read the data into MATLAB can be difficult and time-consuming, particularly for binary files. This function allows you to read the data by simply specifying the format of the file.

Description

`struc = readfields(fname,fstruc)` reads all the records from a fixed format file. *fname* is a string containing the name of the file. If it is empty, the file is selected interactively. `fstruc` is structure defining the format of the file. The contents of `fstruc` are described below. The result is returned in a structure.

`struc = readfields(fname,fstruc,recordIDs)` reads only the records specified in the vector `recordIDs`. For example, `recordIDs = [1 2 3 4]`. All of the fields in the selected records are read.

`struc = readfields(fname,fstruc,fieldIDs)` reads only the fields specified in the cell array `fieldIDs`. For example, `fieldIDs = {1 2 4}`. The selected fields are read from all of the records. `fieldIDs` may be used in place of `recordIDs` in all calling forms.

`struc = readfields(fname,fstruc,recordIDs,mformat)` opens the file with the specified machine format. *mformat* must be recognized by `fopen`.

`struc = readfields(fname,fstruc,recordIDs,mformat,fid)` reads from a file that is already open. `fid` is the file identifier returned by `fopen`. The records are read starting from the current location in the file.

`struc = readfields(fname,fstruc,recordIDs,mformat,fid,'sparse')` disables error messages when the number of elements read does not agree with the stated format of the file. This is useful for formatted files with empty fields. Use `fid = []` for files that are not already open. This option is only compatible with reading selected records.

Examples

Write a binary file and read it.

```
fid = fopen('testbin','wb');
for i = 1:3
   fwrite(fid,['character' num2str(i) ],'char');
   fwrite(fid,i,'int8');
   fwrite(fid,[i i],'int16');
   fwrite(fid,i,'integer*4');
   fwrite(fid,i,'real*8');
end
fclose(fid);

fs(1).length = 10;fs(1).type = 'char';fs(1).name = 'field 1';
fs(2).length = 1;fs(2).type = 'int8';fs(2).name = 'field 2';
fs(3).length = 2;fs(3).type = 'int16';fs(3).name = 'field 3';
fs(4).length = 1;fs(4).type = 'integer*4';fs(4).name = 'field 4';
fs(5).length = 1;fs(5).type = 'float64'; fs(5).name = 'field 5';

s = readfields('testbin',fs);

s(1)
ans =
    field1: 'character1'
    field2: 1
    field3: [1 1]
    field4: 1
    field5: 1
```

Limitations

Formatted numbers must stay within the width specified for them. Files must have a size that is an integer multiple of the computed record length. This is potentially a problem for formatted files on DOS platforms that use a carriage return/line-feed line ending everywhere except the last record. File sizes are not checked when an open file is provided.

Remarks

The format of the file is described in the input argument `fstruc`. `fstruc` is a structure with one entry for every field in the file. `fstruc` has three required fields: `length`, `name` and `type`. For fields containing data binary data of the type that would be read by `fread`, `length` is the number of elements to be read, `name` is a string containing the fieldname under which the read data will be stored in the output structure, and `type` is a format string recognized by `fread`. Repetition modifiers such as `'40*char'` are *not* supported. Fields with empty field names are omitted from the output.

The following `fstruc` definition is for a file with a 40 character field, a field containing two integers, and a field with a single-precision floating point number.

```
fstruc(1).length = 40;
fstruc(1).name = 'character Field'; % spaced will be suppressed
filestruc(1).type = 'char';

fstruc(2).length = 2;
fstruc(2).name = 'integer Field';   % spaced will be suppressed
fstruc(2).type = 'int16';

fstruc(3).length = 1;
fstruc(3).name = 'float Field';     % spaced will be suppressed
fstruc(3).type = 'real*4';
```

The type can also be a `fscanf` and `sscanf`-style format string of the form `'%nX'`, where `n` is the number of characters within which the formatted data is found, and `X`is the conversion character such as `'g'` or `'d'`. For formatted fields, the length entry in `fstruc` is the number of elements, each of which have the width specified in the type string. Fortran-style double precision output such as `'0.0D00'` may be read using a type string such as `'%nD'`, where `n` is the number of characters per element. This is an extension to the C-style format strings accepted by `sscanf`. Users unfamiliar with C should note that `'%d'` is preferred over `'%i'` for formatted integers. MATLAB follows C in interpreting `'%i'` integers with leading zeros as octal. Line ending characters in ASCII files must also be counted in the `fstruc` specification. Note that the number of line ending characters differs across platforms.

A field specification for a formatted field with two integers each 6 characters wide would be of the form:

```
fstruc(4).length = 2;
fstruc(4).name = 'Elevation Units';
fstruc(4).type = '%6d'
```

To summarize, `length` is number of elements for binary numbers, the number of characters for strings, and the number of elements for formatted data.

Fields can be omitted from all output by providing an empty string for the `fstruc` name field.

See Also

`grepfields`	Search within fields of a fixed format file
`readmtx`	Read a matrix stored in a file
`textread`	Read formatted text files
`spcread`	Read columns of data from an ASCII text file
`dlmread`	Read ASCII delimited file

readmtx

Purpose

Read a matrix stored in a file

Syntax

```
mtx = readmtx(fname,nrows,ncols,precision)
readmtx(fname,nrows,ncols,precision,readrows,readcols)
readmtx(fname,nrows,ncols,precision,readrows,readcols,mformat)
readmtx(fname,nrows,ncols,precision,readrows,readcols,mformat,
  nheadbytes)
readmtx(fname,nrows,ncols,precision,readrows,readcols,mformat,
  nheadbytes,nRowHeadBytes)
readmtx(fname,nrows,ncols,precision,readrows,readcols,mformat,
  nheadbytes,nRowHeadBytes,nRowTrailBytes)
readmtx(fname,nrows,ncols,precision,readrows,readcols,mformat,
  nheadbytes,nRowHeadBytes,nRowTrailBytes,nFileTrailBytes)
readmtx(fname,nrows,ncols,precision,readrows,readcols,mformat,
  nheadbytes,nRowHeadBytes,nRowTrailBytes,nFileTrailBytes,
  recordlen)
```

Background

Map data is often provided as binary or ASCII files with a fixed format. Writing your own functions to read the data into MATLAB can be difficult and time-consuming, particularly for binary files. This function allows you to read the data by simply specifying the format of the file.

Description

`mtx = readmtx(`*`fname`*`,nrows,ncols,`*`precision`*`)` reads a matrix stored in a file. The file contains only a matrix of numbers with the dimensions `nrows` by `ncols` stored with the specified *`precision`*. Recognized *`precision`* strings are described below.

`mtx = readmtx(`*`fname`*`,nrows,ncols,`*`precision`*`,readrows,readcols)` reads a subset of the matrix. `readrows` and `readcols` specify which rows and columns are to be read. They can be vectors containing the row or column numbers, or two-element vectors of the form `[start end]`, which are expanded using the colon operator to `start:end`. To read just two rows or columns, without expansion by the colon operator, provide the indices as a column matrix.

`mtx = readmtx(`*`fname`*`,nrows,ncols,`*`precision`*`,readrows,readcols,`*`mformat`*`)` specifies the machine format used to write the file. *`mformat`* can be any string recognized by `fopen`. This option is used to automatically swap bytes for file written on platforms with a different byte ordering.

`mtx = readmtx(`*`fname`*`,nrows,ncols,`*`precision`*`,readrows,readcols,`*`mformat`*`, nheadbytes)` skips the file header, whose length is specified in bytes.

`mtx = readmtx(`*`fname`*`,nrows,ncols,`*`precision`*`,readrows,readcols,`*`mformat`*`, nheadbytes,nRowHeadBytes)` also skips a header which precedes every row of the matrix. The length of the header is specified in bytes.

`mtx = readmtx(`*`fname`*`,nrows,ncols,`*`precision`*`,readrows,readcols,`*`mformat`*`, nheadbytes,nRowHeadBytes,nRowTrailBytes)` also skips a trailer which follows every row of the matrix. The length of the trailer is specified in bytes.

`mtx = readmtx(`*`fname`*`,nrows,ncols,`*`precision`*`,readrows,readcols,`*`mformat`*`, nheadbytes,nRowHeadBytes,nRowTrailBytes,nFileTrailBytes)` accounts for the length of data following the matrix. The sizes of the components of the matrix are used to compute an expected file size, which is compared to the actual file size.

`mtx = readmtx(`*`fname`*`,nrows,ncols,`*`precision`*`,readrows,readcols,`*`mformat`*`, nheadbytes,nRowHeadBytes,nRowTrailBytes,nFileTrailBytes,recordlen)` overrides the record length calculated from the precision and number of columns, and instead uses the record length given in bytes. This is used for formatted data with extra spaces or line breaks in the matrix.

Examples

Write and read a binary matrix file:

```
fid = fopen('binmat','w');
fwrite(fid,1:100,'int16');
fclose(fid);

mtx = readmtx('binmat',10,10,'int16')
mtx =
     1     2     3     4     5     6     7     8     9    10
    11    12    13    14    15    16    17    18    19    20
    21    22    23    24    25    26    27    28    29    30
    31    32    33    34    35    36    37    38    39    40
    41    42    43    44    45    46    47    48    49    50
    51    52    53    54    55    56    57    58    59    60
    61    62    63    64    65    66    67    68    69    70
    71    72    73    74    75    76    77    78    79    80
    81    82    83    84    85    86    87    88    89    90
    91    92    93    94    95    96    97    98    99   100

mtx = readmtx('binmat',10,10,'int16',[2 5],3:2:9)
mtx =
    13    15    17    19
    23    25    27    29
    33    35    37    39
    43    45    47    49
```

Limitations

Every row of the matrix must have the same number of elements.

Remarks

This function reads files that have a general format consisting of a header, a matrix and a trailer. Each row of the matrix may have a certain number of bytes of extraneous information preceding or following the matrix data.

Both binary and formatted data files can be read. If the file is binary, the precision argument is a format string recognized by `fread`. Repetition modifiers such as `'40*char'` are *not* supported. If the file is formatted, precision is a `fscanf` and `sscanf`-style format string of the form `'%nX'`, where `n` is the number of characters within which the formatted data is found, and `X` is the conversion character such as `'g'` or `'d'`. Fortran-style double precision output such as `'0.0D00'` may be read using a precision string such as `'%nD'`, where `n` is the number of characters per element. This is an extension to the C-style format strings accepted by `sscanf`. Users unfamiliar with C should note that `'%d'` is preferred over `'%i'` for formatted integers. MATLAB follows C in interpreting `'%i'` integers with leading zeros as octal. Formatted files with line endings need to provide the number of trailing bytes per row, which may be 1 for platforms with carriage returns *or* line-feed (Macintosh, UNIX), or 2 for platforms with carriage returns *and* line-feeds (DOS).

See Also

`readfields`	Reads fields or records from a fixed format file
`textread`	Read formatted text files
`spcread`	Read columns of data from an ascii text file
`dlmread`	Read ASCII delimited file

rec2geod

Purpose

Convert from rectifying to geodetic latitudes

Syntax

```
lat = rec2geod(lat0)
lat = rec2geod(lat0,geoid)
lat = rec2geod(lat,units)
lat = rec2geod(lat,geoid,units)
```

Background

Longitudes have the same meaning whether the Earth is treated as a sphere or as an ellipsoid. However, the definition of latitude in the case of an ellipsoid is more complicated. Geodetic, or geographic, latitudes on an ellipsoid can be transformed from latitudes on an auxiliary sphere with certain properties.

Geodetic latitude: (also called *geographic latitude*) the angle a normal line passing through a surface point makes with the plane of the Equator.

Rectifying latitude: latitudes on an auxiliary sphere giving correct distances along meridians relative to the ellipsoid.

Description

`lat = rec2geod(lat0)` returns the rectifying latitudes provided in `lat0` transformed to geodetic latitudes, which are returned in `lat`.

`lat = rec2geod(lat0,geoid)` defines the elliptical model of the Earth, given by the geoid vector `geoid`, to which `lat0` is transformed. The default geoid is the same as the default of `almanac('earth','geoid')`, the 1980 Geodetic Reference System ellipsoid.

`lat = rec2geod(lat,geoid,units)` defines the angle units of the inputs and outputs, where *units* is any valid angle units string. The default is `'degrees'`.

Examples

```
lat = rec2geod([0 45 90])
lat =
         0   45.1443   90.0000
```

See Also

`almanac`	Planetary data
`geod2rec` `iso2geod`	Other auxiliary latitude functions

Purpose

Determine new position from starting point, azimuth, and range

Syntax

```
[newlat,newlon] = reckon(lat,lon,rng,az)
[newlat,newlon] = reckon(lat,lon,rng,az,units)
[newlat,newlon] = reckon(lat,lon,rng,az,geoid)
[newlat,newlon] = reckon(lat,lon,rng,az,geoid,units)
[newlat,newlon] = reckon(track,lat,...)
```

Description

`[newlat,newlon] = reckon(lat,lon,rng,az)` returns a new geographic position in `newlat` and `newlon` at a given range `rng` and azimuth `az` from starting points in `lat` and `lon`. There must be a one-for-one correspondence between the elements of all four inputs.

`[newlat,newlon] = reckon(lat,lon,rng,az,units)` specifies the angular units of the inputs and outputs, where *`units`* is any valid angle units string. The default value is `'degrees'`. Note that the range input, `rng`, must be in these angular units as well.

`[newlat,newlon] = reckon(lat,lon,rng,az,geoid,units)` specifies an elliptical definition of the Earth, where `geoid` is a two-element geoid vector. The default geoid is a spherical Earth, which is sufficient for most applications. If a geoid vector is input, the range input, `rng`, is in the units of the semimajor axis, that is, the first element of `geoid`.

`[newlat,newlon] = reckon(track,lat,...)` specifies the sense of the reckoning. If the string *`track`* is `'gc'` (the default), the new positions are reckoned by following great circle paths. Alternatively, track can be the string `'rh'`, in which case the new positions are reckoned by following rhumb lines.

Examples

What are the coordinates of the point 600 nautical miles northwest of London, UK (51.5°N,0°), in a great circle sense?

```
dist = nm2deg(600)  % convert nm distance to degrees
dist =
    9.9932

reckon(51.5,0,dist,315)  % northwest is 315 degrees
pt1 =
   57.8999  -13.3507
```

Now, where would a plane taking off from London and traveling on a constant northwesterly course for 600 nautical miles end up?

```
reckon('rh',51.5,0,dist,315)
pt2 =
   58.5663  -12.3699
```

How far apart are these points (distance in great circle sense)?

```
separation = distance('gc',pt1,pt2)
separation =
    0.8430

nmsep = deg2nm(separation)  % convert answer to nautical miles
nmsep =
   50.6152
```

Over 50 nautical miles separate the two points.

See Also

`azimuth`	Azimuth between two points on the globe
`distdim`, `distance`	Distance between two points on the globe
`km2deg`	Convert distance units
`dreckon`	Navigational dead reckoning
`track` `track1` `track2`	Tracing paths on the globe

Purpose

Reduce the number of points in vector data

Syntax

```
[latout,lonout] = reducem(latin,lonin)
[latout,lonout] = reducem(latin,lonin,tol)
[latout,lonout,cerr] = reducem(...)
[latout,lonout,cerr,tol] = reducem(...)
```

Description

`[latout,lonout] = reducem(latin,lonin)` reduces the number of points in vector map data. In this case the tolerance is computed automatically.

`[latout,lonout] = reducem(latin,lonin,tol)` uses the provided tolerance. The units of the tolerance are degrees of arc on the surface of a sphere.

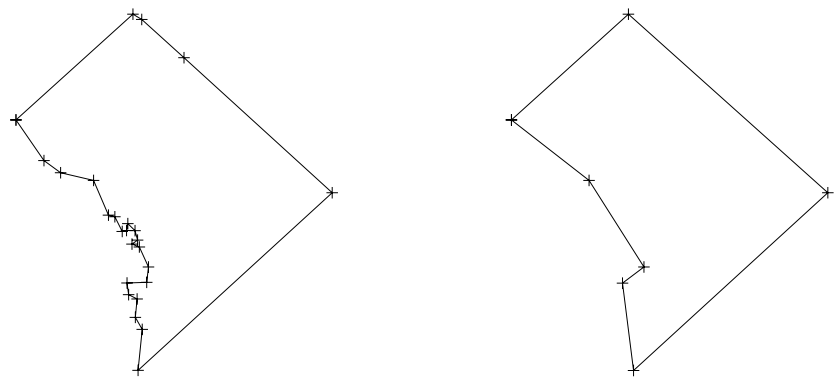
`[latout,lonout,cerr] = reducem(...)` in addition returns a measure of the error introduced by the simplification. The output `cerr` is the difference in the arc length of the original and reduced data, normalized by the original length.

`[latout,lonout,cerr,tol] = reducem(...)` also returns the tolerance used in the reduction, which is useful when the tolerance is computed automatically.

Examples

Compare the original and reduced outlines of the District of Columbia from the usahi atlas data:

```
load usahi
[lat,long] = extractm(stateline,'district');
[latout,lonout,cerr] = reducem(lat,long);
```



Remarks

Vector data is reduced using the Douglas-Peucker line simplification algorithm. This method recursively subdivides a polygon until a run of points can be replaced by a straight line segment, with no point in that run deviating from the straight line by more than the tolerance. The distances used to decide on which runs of points to eliminate are computed in a Plate Carree projection.

Reduced geographic data may not always be appropriate for display. If all intermediate points in a dataset are reduced, then lines appearing straight in one projection will be incorrectly displayed as straight lines in others.

See Also

`interpm`	Interpolates vector data to a specified data separation
`resizem`	Resizes a matrix map

Purpose

Resize a matrix map

Syntax

```
map = resizem(map0,m)
map = resizem(map0,[r c])
map = resizem(map0,m,method)
map = resizem(map0,[r c],method)
[map,maplegend] = resizem(map0,m,maplegend0)
[map,maplegend] = resizem(map0,m,maplegend0,method)
```

Description

`map = resizem(map0,m)` resizes an original matrix map, `map0` by a resizing factor `m`. For example, if `m` is 0.5, the number of rows and the number of columns will be cut in half. The result is the resized map, `map`.

`map = resizem(map0,[r c])` resizes `map0` so that the output map, `map` has `r` rows and `c` columns.

`map = resizem(map0,m,method)` specifies the method of interpolation. The string *method* `'nearest'` results in nearest neighbor interpolation, the default. `'cubic'` results in bicubic interpolation, and `'linear'` results in bilinear interpolation.

`[map,maplegend] = resizem(map0,m,maplegend0)`resizes a regular matrix map with a map legend vector `maplegend0` and returns a regular matrix map and its map legend vector, `maplegend`.

This case requires a resizing factor, `m`, rather than the `[r c]` vector, as map legends only have meaning for regular matrix maps (i.e., rows represent the same angular dimension as columns).

When the map size is being reduced, `resizem` lowpass filters the map before interpolating to avoid aliasing. By default, this filter is designed using FIR1, but can be specified using:

```
resizem(...,method,h) The default filter is 11-by-11
resizem(...,method,n) uses an n-by-n filter
resizem(...,method,0) turns off the filtering
```

Unless a filter `h` is specified, `resizem` will not filter when `'nearest'` is used. These filters are associated with the MATLAB Image Processing Toolbox.

Example

Double the size of a map:

```
map = [1 2; 3 4]
map =
      1  2
      3  4

newmap = resizem(map,2)
newmap =
      1  1  2  2
      1  1  2  2
      3  3  4  4
      3  3  4  4
```

Purpose

Re-stack objects within the axes

Syntax

```
restack(h,position)
```

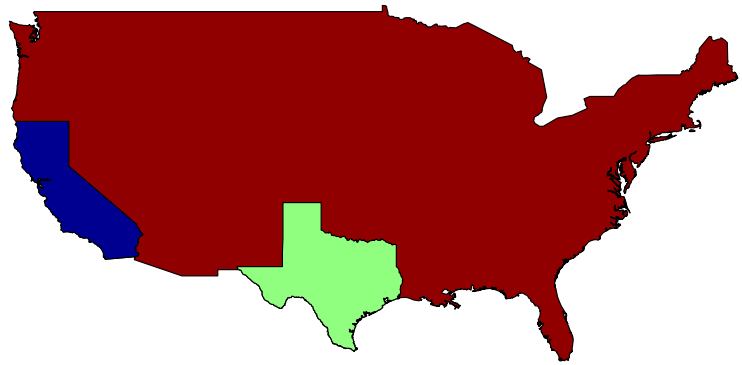
Description

`restack(h,position)` changes the stacking position of the object `h` within the axes. `h` can be a handle, a vector of handles to graphics objects, or a name string recognized by `handlem`. Recognized *position* strings are 'top', 'bottom', 'bot', 'up' or 'down'.

Examples

Re-stack the patch objects of California and Texas to lie above the U.S.

```
axesm miller
load usalo

h = displaym(state,'california');
displaym(state,'texas');
displaym(conus)

restack(h,'top')
restack('Texas','top')
```



Remarks

This function is the command-line equivalent of the stacking buttons in the `mobjects` graphical user interface. The stacking order is the order of the children of the axes.

See Also

`mobjects`	GUI for manipulating objects displayed on an axes

Purpose

Provide intersection coordinates for pairs of rhumb lines

Syntax

```
[newlat,newlon] = rhxrh(lat1,lon1,az1,lat2,lon2,az2)
[newlat,newlon] = rhxrh(lat1,lon1,az1,lat2,lon2,az2,units)
```

Description

For any pair of rhumb lines, there are three possible intersection conditions: the lines are identical, they intersect once, or they do not intersect at all (except at the poles, where all non-equatorial rhumb lines meet – this is not considered an intersection). `rhxrh` does not allow multiple rhumb line intersections, although it is possible to construct cases in which such a condition occurs. See the discussion of *Limitations* below.

Rhumb line notation consists of a point on the line and the constant azimuth of the line.

`[newlat,newlong] = rhxrh(lat1,lon1,az1,lat2,lon2,az2)` returns in `newlat` and `newlon` the location of the intersection point for each pair of rhumb lines input in *rhumb line notation*. For example, the first line in the pair passes through the point (`lat1`,`lon1`) and has a constant azimuth of `az1`.When the two rhumb lines are identical or do not intersect (conditions that are not, in general, apparent by inspection), two `NaN`s are returned instead and a warning is displayed. The inputs must be column vectors.

`[newlat,newlon] = rhxrh(lat1,lon1,az1,lat2,lon2,az2,units)` specifies the units used, where *units* is any valid units string. The default units are `'degrees'`.

Examples

Given a starting point at (10°N,56°W), a plane maintains a constant heading of 35°. Another plane starts at (0°,10°W) and proceeds at a constant heading of 310° (–50°). Where would their two paths cross each other?

```
[newlat,newlong] = rhxrh(10,-56,35,0,-10,310)
newlat =
   26.9774
newlong =
  -43.4088
```

Limitations

Rhumb lines are specifically helpful in navigation because they represent lines of constant heading, whereas great circles have, in general, continuously changing heading. In fact, the Mercator projection was originally designed so that rhumb lines plot as straight lines, which facilitates both manual plotting with a straightedge and numerical calculations using a Cartesian planar representation. When a rhumb line proceeds off the left or right *edge* of this representation at some latitude, it reappears on the other edge at the same latitude and continues on the same slope. For rhumb lines where this occurs – for example, one with a heading of 85° – it is easy to imagine another rhumb line, say one with a heading of 0°, repeatedly intersecting the first. The real-world uses of rhumb lines make this merely an intellectual exercise, however, for in practice it is always clear which *crossing* line segment is relevant. The command `rhxrh` returns at most one intersection, selecting in each case that line segment containing the input starting point for its computation.

See Also

`gcxgc`
`gcxsc`
`scxsc`
`crossfix`
	Other intersection functions

`navfix`	Navigational fixing

Purpose

Use workspace variables to construct a cell array for input to the mlayers tool

Syntax

```
rootlayr
```

Description

rootlayr allows the mlayers tool to be used with workspace variables. It constructs a cell array that contains all the structure variables in the current workspace. This cell array is returned in the variable ans, which can then be an input to mlayers. If there is an existing variable named ans, it is overwritten.

The recommended calling procedure is `rootlayr;mlayers(ans);`

Examples

rootlayr will create a cell array named ans, consisting of the three structure variables in the following workspace.

```
whos
  Name          Size         Bytes  Class
  borders       1x1          38390  struct array
  lats       2345x1          18760  double array
  lons       2345x1          18760  double array
  nation        1x1          70224  struct array
  states        1x51        254970  struct array

rootlayr
ans
  ans =
    [1x1  struct]    'borders'
    [1x1  struct]    'nation'
    [1x51 struct]    'states'
```

The command mlayers(ans) can now be used to activate the mlayers tool for the structures contained in ans.

See Also

geographic data structure	Specially formatted structure for map data
mlayers	Interactive geographic data structure manipulation (see Chapter 3, “GUI Reference”)

rotatem

Purpose

Transform vector data to a new coordinate system based on a new origin

Syntax

```
[lat1,lon1] = rotatem(lat,lon,origin,'forward')
[lat1,lon1] = rotatem(lat,lon,origin,'inverse')
[lat1,lon1] = rotatem(lat,lon,origin,'forward',units)
[lat1,lon1] = rotatem(lat,lon,origin,'inverse',units)
```

Description

This command will transform vector map data to a new coordinate system.

An analytical use of the new data can be realized in conjunction with the `newpole` command. If a selected point is made the *north pole* of the new system, then when new vector data is created with `rotatem`, the distance of every data point from this new north pole is its new colatitude (90° minus latitude). The absolute difference in the great circle azimuths between every pair of points from their new *pole* is the same as the difference in their new longitudes.

`[lat1,lon1] = rotatem(lat,lon,origin,'forward')` transforms latitude and longitude data (`lat` and `lon`) to their new coordinates (`lat1` and `lon1`) in a coordinate system resulting from Euler angle rotations as specified by `origin`. The input `origin` is a three- (or two-) element vector having the form `[latitude longitude orientation]`. The latitude and longitude are the coordinates of the point in the original system, which is the center of the output system. The orientation is the azimuth from the new origin point to the original North Pole in the new system. If `origin` has only two elements, the orientation is assumed to be 0°. This `origin` vector might be the output of `putpole` or `newpole`.

`[lat1,lon1] = rotatem(lat,lon,origin,'inverse')` transforms latitude and longitude data (`lat` and `lon`) in a coordinate system *that has been transformed* by Euler angle rotations specified by `origin` to their coordinates (`lat1` and `lon1`) in the coordinate system *from which they were originally transformed*. In a sense, this *undoes* the `'forward'` process. Be warned, however, that if data is rotated forward and then inverted, the final data may not be identical to the original. This is due to round off and *data collapse* at the original and intermediate singularities (the poles).

`[lat1,lon1] = rotatem(lat,lon,origin,'forward',units)` specifies the angle units of the data, where *units* is any recognized angle units string. The default is `'radians'`. Note that this default is different from that of most functions.

Examples

What would the coordinates of Rio de Janeiro (23°S,43°W) be in a coordinate system in which New York (41°N,74°W) was made the north pole? Use the `newpole` function to get the origin vector associated with putting New York at the Pole:

```
nylat = 41; nylon = -74;
riolat = -23; riolon = -43;
origin = newpole(nylat,nylon);
[riolat1,riolon1] = rotatem(riolat,riolon,origin,...
                            'forward','degrees')
riolat1 =
   19.8247
riolon1 =
 -149.7375
```

What does this mean? For one thing, the colatitude of Rio in this new system is its distance from New York. Compare the distance between the original points and the new colatitude:

```
dist = distance(nylat,nylon,riolat,riolon)
dist =
   70.1753

90-riolat1
ans =
   70.1753
```

See Also

`neworig`	Transform regular matrix map to new coordinate system based on a new origin
`newpole`	Select point to place at north pole
`org2pol`	Pole of transformed coordinate system
`putpole`	Origin of transformed coordinate system

rotatetext

Purpose

Rotate text to the projected graticule

Syntax

```
rotatetext
rotatetext(objects)
rotatetext(objects,direction)
```

Description

`rotatetext` rotates displayed text objects to account for the curvature of the graticule. The objects are selected interactively from a graphical user interface.

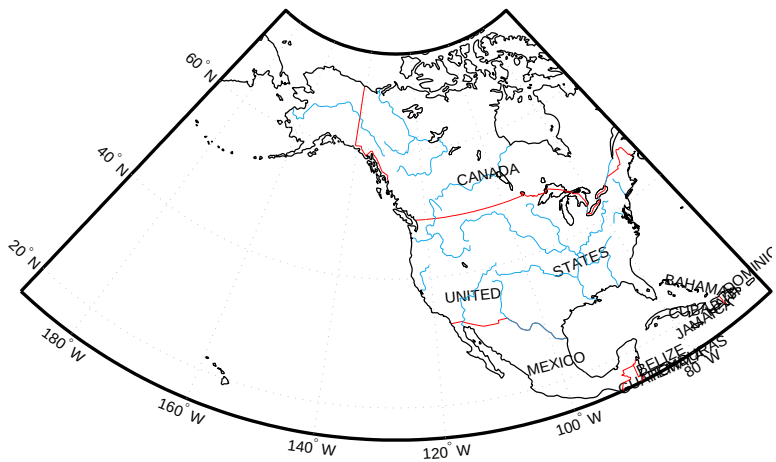
`rotatetext(objects)` rotates the selected objects. `objects` may be a name string recognized by `handlem`, or a vector of handles to displayed text objects.

`rotatetext(objects,'inverse')` removes the rotation added by an earlier use of `rotatetext`. If omitted, '`forward`' is assumed.

Examples

Add text to a map, and rotate the text to the graticule.

```
worldmap('usa','lineonly')
h = displaym(worldlo('POtext'));
trimcart(h)
rotatetext(h)
```



Remarks

Meridian and parallel labels can be rotated automatically by setting the map axes `LabelRotation` property to 'on'.

See Also

`vfwdtran`	Transforms vector azimuths to a projection space angle
`vinvtran`	Transforms vector azimuths from a projection space angle

roundn

Purpose

Round numbers at specified powers of 10

Syntax

```
outnum = roundn(innum)
outnum = roundn(innum,n)
```

Description

`outnum = roundn(innum)` rounds the elements of `innum` to the nearest one-hundredth.

`outnum = roundn(innum,n)` specifies the power of 10 to which the elements of `innum` are rounded. For example, if `n = 2`, round to the nearest hundred (10^2).

Examples

Using generated numbers, round them to significant tenths, ones, and tens figures (note that your original numbers could differ):

```
fullfig = 1000*magic(2)/7
fullfig =
   142.8571  428.5714
   571.4286  285.7143

tenths = roundn(fullfig,-1)
tenths =
   142.9000  428.6000
   571.4000  285.7000

units = roundn(fullfig,0)
units =
   143   429
   571   286

tens = roundn(fullfig,1)
tens =
   140   430
   570   290
```

See Also

`epsm` Map precision

Purpose

Compute auxiliary sphere radii

Syntax

```
r = rsphere('biaxial',geoid)
r = rsphere('biaxial',geoid,method)

r = rsphere('triaxial',geoid)
r = rsphere('triaxial',geoid,method)

r = rsphere('eqavol',geoid)
r = rsphere('authalic',geoid)
r = rsphere('rectifying',geoid)

r = rsphere('curve',geoid)
r = rsphere('curve',geoid,lat)
r = rsphere('curve',geoid,method)
r = rsphere('curve',geoid,lat,method)
r = rsphere('curve',geoid,lat,method,units)

r = rsphere('euler',lat1,lon1,lat2,lon2,geoid)
r = rsphere('euler',lat1,lon1,lat2,lon2,geoid,units)
```

Description

This command calculates the radii of auxiliary spheres for the ellipsoid. An auxiliary sphere is a sphere that shares certain desired characteristics with the ellipsoid.

`r = rsphere('biaxial',geoid)` calculates the radius of an biaxial auxiliary sphere for the ellipsoid specified by the two-element geoid vector `geoid`. The output, `r`, is the radius of this sphere in units consistent with the semimajor axis, that is, the first element of `geoid`. The biaxial radius is calculated by averaging the semimajor and semiminor axes of the ellipsoid, giving each equal weight.

`r = rsphere('biaxial',geoid,`*`method`*`)` specifies the averaging method. If the string *method* is `'mean'` (the default), an arithmetic mean is used. If *method* is `'norm'`, a geometric mean is used.

`r = rsphere('triaxial',geoid)` results in a triaxial radius, which is calculated by averaging the ellipsoidal axes while giving double weight to the semimajor axis to reflect its role in two of the ellipsoid's three dimensions.

`r = rsphere('eqavol',geoid)` returns the radius of a sphere with a volume equal to that of the ellipsoid.

`r = rsphere('authalic',geoid)` returns the radius of a sphere with a surface area equal to that of the ellipsoid.

`r = rsphere('rectifying',geoid)` returns the radius of a sphere with meridional distances equal to those of the ellipsoid.

`r = rsphere('curve',geoid,lat,`*`method`*`,`*`units`*`)` returns a radius that is the result of averaging the meridianal and transverse radii of curvature at the specified latitude, `lat`. The units of the input `lat` can be specified by the valid angle units string *units*. The default units are `'degrees'`, the default averaging method is `'mean'`, and the default latitude is 45°.

`r = rsphere('euler',lat1,lon1,lat2,lon2,geoid)` calculates a radius using Euler's Theorem. This calculation requires the specification of an arc, which is defined by its endpoints, (`lat1`,`lon1`) and (`lat2`,`lon2`).

Examples

Different criteria result in different spheres:

```
r = rsphere('biaxial',almanac('earth','geoid','km'))
r =
   6.3674e+03

r = rsphere('triaxial',almanac('earth','geoid','km'))
r =
   6.3710e+03

r = rsphere('curve',almanac('earth','geoid','km'))
r =
   6.3781e+03
```

See Also

`rcurve` Radii of curvature for the ellipsoid

Purpose

Add graphic scale

Syntax

```
scaleruler on
scaleruler off
scaleruler(property,value,...)
```

Background

Cartographers often add graphic elements to the map to indicate its scale. Perhaps the most commonly used is the graphic scale, a ruler-like object that shows distances on the ground at the correct size for the projection.

Description

`scaleruler` toggles the display of a graphic scale. If no graphic scale is currently displayed in the current map axes, one will be added. If any graphic scales are currently displayed, they will be removed.

`scaleruler on` adds a graphic scale to the current map axes. Multiple graphic scales can be added to the same map axes.

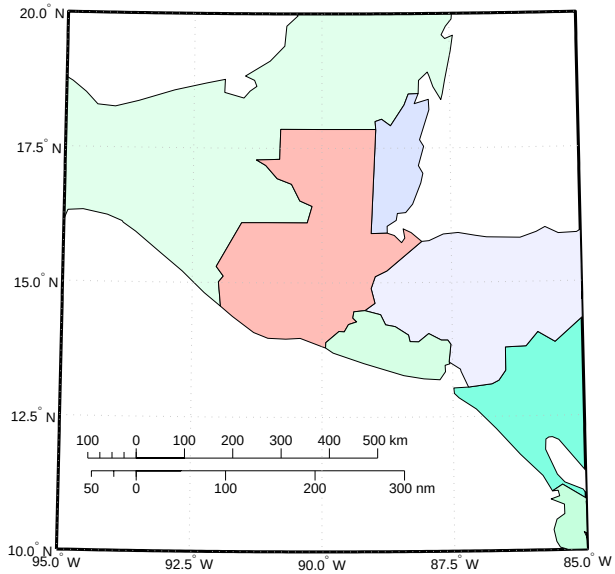
`scaleruler off` removes any currently displayed graphic scales.

`scaleruler(property,value,...)` adds a graphic scale and sets the properties to the values specified.

Examples

Create a map, add a graphic scale with the default settings, and shift it up a bit. Add a second scale showing nautical miles, and change the tick marks and direction.

```
worldmap('guatemala','patchonly')

scaleruler
setm(handlem('scaleruler1'),'YLoc',.205)

scaleruler('units','nm')
setm(handlem('scaleruler2'),...
     'MajorTick',0:100:300,'MinorTick',0:25:50,...
     'TickDir','down','MajorTickLength',km2nm(20),...
     'MinorTickLength',km2nm(12.5))
```



Remarks

Graphic scale objects are controlled using `setm` and `getm`. A list of graphic scale properties is displayed by the command `setm(h)`, where `h` is the handle to a graphic scale object. The current values for a displayed graphic scale object can be retrieved using `getm`. The properties of a displayed graphic scale object can be modified using `setm`.

Modifying the properties of the graphic scale results in the replacement of the original object. For this reason, handles to the graphic scale object will change. Use `handlem('scaleruler')` to get a list of the current handles to all graphic scale objects. Use `handlem('scalerulerN')`, where `N` is an integer, to get the handle to a particular graphic scale. Use `namem` to see the names of existing graphic scale objects. The name of a graphic scale object is also stored in the read-only '`Children`' property, which is accessed using `getm`.

Use `scaleruler off`, `clmo scaleruler`, or `clmo scalerulerN` to remove the scale rulers. The handles to the components of the graphic scale are hidden, so using `delete` will remove only the baseline. In that event, the remaining elements of the graphic scale can be removed with the command

`delete(findall(gca,'Tag','scalerulerN'))`, where `N` is the corresponding integer.

The graphic scale object can be repositioned by dragging the baseline with the mouse. The position can also be changed by modifying the `'XPos'` and `'YPos'` properties using `setm`.

Object Properties

Properties that control appearance

`Color` `ColorSpec {no default}`

Color of the displayed graphic scale — This property controls the color of the graphic scale lines and text. You can specify a color using a vector of RGB values or one of MATLAB's predefined names. By default, the graphic scale is displayed in black (`[0 0 0]`).

`FontAngle` `{normal} | italic | oblique`

Angle of the graphic scale label text — This property controls the appearance of the graphic scale text components. Use any font angle string recognized by MATLAB.

`FontName` `courier | {helvetica} | symbol | times`

Font family name for all graphic scale labels — This property sets the font for all displayed graphic scale labels. To display and print properly `FontName` must be a font that your system supports.

`FontSize` `scalar in units specified in FontUnits {9}`

Font size — A property specifying the font size to use for all displayed graphic scale labels, in units specified by the `FontUnits` property. The default point size is 9.

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

Units used to interpret the FontSize property — When set to normalized, the Toolbox interprets the value of `FontSize` as a fraction of the height of the axes. For example, a normalized `FontSize` of 0.16 the text characters to a font whose height is one—tenth of the axes' height. The default units of points are equal to 1/72 of an inch.

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

Select bold or normal font — the character weight for all displayed graphic scale labels.

Label `string`

Label text for the graphic scale — this property contains a string used to label the graphic scale. The text is displayed centered on the scale. The label is often used to indicate the scale of the map, for example “1:50,000,000.”

LineWidth `scalar {0.5}`

Graphic scale line width — this property sets the line width of the displayed scale. The value is a scalar representing points, which is 0.5 by default.

MajorTick `vector`

Graphic scale major tick locations — this property sets the major tick locations for the graphic scale. The default values are chosen to give a reasonably sized scale. You can specify the locations of the tick marks by providing a vector of locations. These are usually equally spaced values as generated by `start:step:end`. The values are distances in the units of the `Units` property.

MajorTickLabel `Cell array of strings`

Graphic scale major tick labels — this property sets the text labels associated with the major tick locations. By default, the labels are identical to the major tick locations. These can be overridden by providing a cell array of strings. There must be as many strings as tick locations.

MajorTickLength `scalar`

Length of the major tick lines — this property controls the length of the major tick lines. The length is a distance in the units of the `Units` property.

MinorTick `vector`

Graphic scale minor tick locations — this property sets the minor tick locations for the graphic scale. The default values are chosen to give a reasonably sized scale. You can specify the locations of the tick marks by providing a vector of locations. These are usually equally spaced values as generated by `start:step:end`. The values are distances in the units of the `Units` property.

MinorTickLabel strings

Graphic scale minor tick labels — this property sets the text label associated with the minor tick locations. By default, the label is identical to last minor tick location. This can be overridden by providing a string label.

MinorTickLength scalar

Length of the minor tick lines — this property controls the length of the minor tick lines. The length is a distance in the units of the Units property.

RulerStyle {ruler} | lines | patches

Style of the graphic scale — This property selects between three different kinds of graphic scale displays. The default ruler style looks like the MATLAB x-axis. The lines style has three horizontal lines across the tick marks. This type of graphic scale is often used on maps from the U. S. Geological Survey. The patches style has alternating black and white rectangles in place of lines and tic marks.

TickDir {up} | down

Direction of the tick marks and text — This property controls the direction in which the lick marks and text labels are drawn. In the default up direction, the tick marks and text labels are placed above the baseline, which is placed at location given in the XLoc property. In the down position, the tick marks and labels are drawn below the baseline.

TickMode {auto} | manual

Tick locations mode — This property controls whether the tick locations and labels are computed automatically, or user-specified. Explicitly setting the tick labels or locations results in a 'manual' tick mode. Setting any of the tick labels or locations to an empty matrix resets the tick mode back to 'auto'. Setting the tick mode to 'auto' clears any explicitly specified tick locations and labels, which are then replaced by default values.

XLoc scalar

X-location of the graphic scale — this property controls the horizontal location of the graphic scale within the axes. The location is specified in the axes Cartesian projected coordinates. Use showaxes to make the Cartesian grid labels visible. The graphic scale can also be moved by dragging the baseline with the mouse.

YLoc scalar

Y-location of the graphic scale — this property controls the vertical location of the graphic scale within the axes. The location is specified in the axes Cartesian projected coordinates. Use showaxes to make the Cartesian grid labels visible. The graphic scale can also be moved by dragging the baseline with the mouse.

Properties that control scaling

Azimuth scalar

Azimuth of scale computation — The scale of a map varies within the projection, with geographic location and azimuth. This property controls the azimuth along which the scaling between geographic and projected coordinates is computed. The azimuth is given in the current angle units of the map axes. The default azimuth is 0.

Lat scalar

Latitude of scale computation — The scale of a map varies within the projection, with geographic location and azimuth. This property controls the geographic location at which the scaling between geographic and projected coordinates is computed. The latitude is given in the current angle units of the map axes. The default location is the center of the displayed map.

Long scalar

Longitude of scale computation — The scale of a map varies within the projection, with geographic location and azimuth. This property controls the geographic location at which the scaling between geographic and projected coordinates is computed. The longitude is given in the current angle units of the map axes. The default location is the center of the displayed map.

Radius almanac body or scalar

Planetary radius — The radius property controls the scaling between angular and surface distances. If radius is a string, then it is evaluated as an almanac body to determine the spherical radius. If numerical, it is the radius of the desired sphere in the same units as the Units property. The default is 'earth'.

Units (valid distance unit strings)

Surface distance units — This property defines the distance units displayed in the graphic scale. Units can be any distance unit string recognized by `distdim`. The distance string is also used in the last graphic scale text label.

Other Properties

Children (read-only)

Name string of graphic scale elements — This property contains the tag string assigned to the graphic elements that comprise the graphic scale. All elements of the graphic scale have hidden handles except the baseline. You do not normally need to access the elements directly.

See Also

`distance`	Calculates distances between points on a geoid
`surfdist`	Interactive tool for distance, azimuth and reckoning calculations
`axesscale`	Resize axes for equivalent scale
`paperscale`	Figure paper size for a given map scale
`distortcalc`	Calculates distortion parameters for a map projection
`mdistort`	Displays contours of constant distortion on a map

scatterm

Purpose

Construct a thematic map with proportional symbols

Syntax

```
scatterm
scatterm(lat,lon)
scatterm(lat,lon,s)
scatterm(lat,lon,s,c)
scatterm(...,m)
scatterm(...,'filled')
```

Description

`scatterm(lat,lon,s,c)` displays colored circles at the locations specified by the vectors `lat` and `lon` (which must be the same size). The area of each marker is determined by the values in the vector `s` (in points^2) and the colors of each marker are based on the values in `c`. `s` can be a scalar, in which case all the markers are drawn the same size, or a vector the same length as `lat` and `lon`.

When `c` is a vector the same length as `lat` and `lon`, the values in `c` are linearly mapped to the colors in the current colormap. When `c` is a `length(lat)`-by-3 matrix, the values in `c` specify the colors of the markers as RGB values. `c` can also be a color string.

`scatterm(lat,lon)` draws the markers in the default size and color.

`scatterm(lat,lon,s)` draws the markers with a single color.

`scatterm(...,m)` uses the marker *m* instead of 'o'.

`scatterm(...,'filled')` fills the markers.

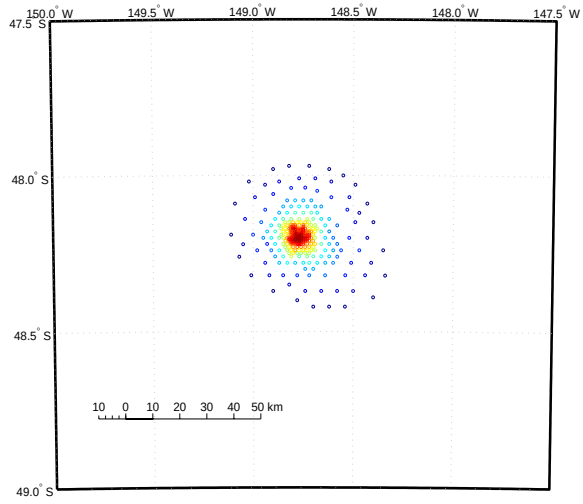
`scatterm`, without any inputs, will activate a GUI to project point data onto the current map axes.

`h = scatterm(...)` returns handles of patches created.

Examples

Plot the seamount data provided with MATLAB as symbols with the color proportional to the height.

```
load seamount
worldmap([-49 -47.5],[-150 -147.5],'none')
scatterm(y,x,5,z)
scaleruler
```



See Also

`stem3m` Project stem plot on map axes

scircle1

Purpose

Compute coordinates of a small circle path from center, radius, and arc limits

Syntax

```
pts = scircle1(lat,lon,rng)
[latc,lonc] = scircle1(lat,lon,rng)
[latc,lonc] = scircle1(lat,lon,rng,az)
[latc,lonc] = scircle1(lat,lon,rng,units)
[latc,lonc] = scircle1(lat,lon,rng,az,units)
[latc,lonc] = scircle1(lat,lon,rng,az,geoid)
[latc,lonc] = scircle1(lat,lon,rng,az,geoid,units)
[latc,lonc] = scircle1(lat,lon,rng,az,geoid,units,npts)
[latc,lonc] = scircle1(track,lat,lon,rng...)
```

Background

A small circle is the locus of all points an equal surface distance from a given center. For true small circles, this distance is always calculated in a great circle sense; however, the `scircle1` command allows a locus to be calculated using distances in a rhumb line sense as well. An example of a small circle is *all points exactly 100 miles from the Washington Monument*. Parallels on the globe are all small circles. Great circles are a subset of small circles, specifically those with a radius of 90° or its angular equivalent, so all meridians on the globe are small circles as well.

Small circle notation consists of a center point and a radius in units of angular arc length.

Description

`[latc,lonc] = scircle1(lat,lon,rng)` returns the coordinates of points along small circles centered at the points provided in `lat` and `lon` with radii given in `rng`. These radii must in this case be given in the same angle units as the center points (`'degrees'`). The coordinates for multiple small circles are stored in separate columns of `latc` and `lonc`.

`[latc,lonc] = scircle1(lat,lon,rng,az)` specifies the arc section of the small circle for which points are returned. The input `az` is a one- or two-column vector. When `az` has a single column, points are returned for the arc segment from 0° azimuth clockwise to the positive entries in `az` (counterclockwise for negative entries). When `az` has two columns, the returned points correspond to arc segments from the first-column entry clockwise to the second-column entry. When `az` is empty or not provided, points for the entire small circle are returned.

`[latc,lonc] = scircle1(lat,lon,rng,az,`*`units`*`)` specifies the units for the inputs and outputs, where *units* is any valid angle units string. The default value is `'degrees'`.

`[latc,lonc] = scircle1(lat,lon,rng,az,geoid,`*`units`*`)` specifies the elliptical definition of the Earth to be used with the two-element `geoid` vector. The default geoid model is the sphere, which is sufficient for most applications. When a geoid is input, the range inputs in `rng` must be in the units of the geoid semimajor axis, rather than in the angle units specified by *units*.

`[latc,lonc] = scircle1(lat,lon,rng,az,geoid,`*`units`*`,npts)` specifies the number of output points, `npts`, returned per small circle. The default value of `npts` is 100.

`[latc,lonc] = scircle1(track,lat,lon,rng...)` specifies the logic with which ranges are calculated. If the string track is `'gc'` (the default), great circle distance is used. It track is `'rh'`, rhumb line distance is used.

`pts = scircle1(lat,lon,rng)` returns the points in a two-column output `pts`.

Examples

Create and plot a small circle centered at (0°,0°) with a radius of 10°:

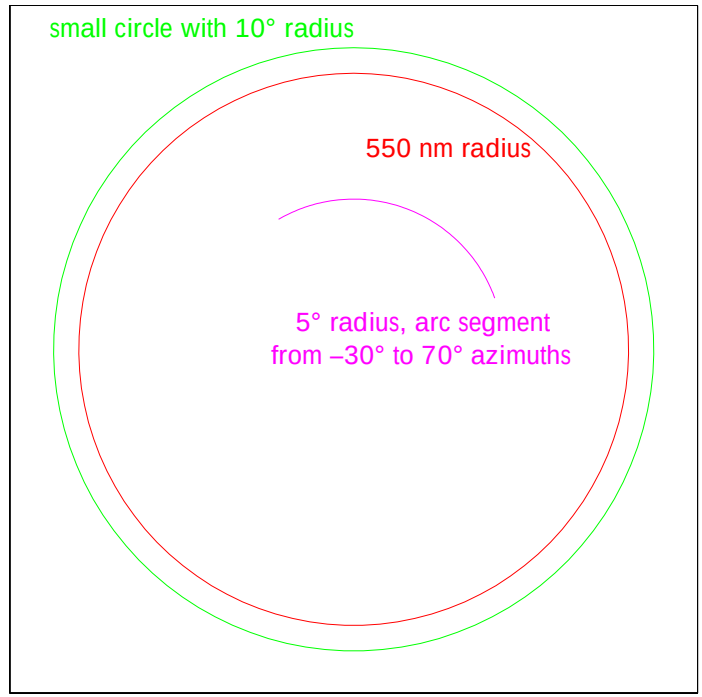
```
axesm('mercator','MapLatLimit',[30 -30],'MapLonLimit',[-30 30]);
[latc,longc] = scircle1(0,0,10);
plotm(latc,longc,'g')
```

If the desired radius is known in some nonangular distance unit, use the radius returned by the `almanac` function as the geoid to set the range units (use an empty azimuth entry to indicate a full circle):

```
earthradius = almanac('earth','radius','nm');
[latc,longc] = scircle1(0,0,550,[],earthradius);
plotm(latc,longc,'r')
```

For just an arc of the circle, enter an azimuth range:

```
[latc,longc] = scircle1(0,0,5,[-30 70]);
plotm(latc,longc,'m')
```



See Also

`scircle2`	Small circle from center and perimeter point
`track`	Connect waypoints with track segments
`track1` `track2`	Great circles and rhumb lines

Purpose

Compute coordinates of a small circle path from center and perimeter point

Syntax

```
pts = scircle2(lat1,lon1,lat2,lon2)
[latc,lonc] = scircle2(lat1,lon1,lat2,lon2)
[latc,lonc] = scircle2(lat1,lon1,lat2,lon2,units)
[latc,lonc] = scircle2(lat1,lon1,lat2,lon2,geoid)
[latc,lonc] = scircle2(lat1,lon1,lat2,lon2,geoid,units)
[latc,lonc] = scircle2(lat1,lon1,lat2,lon2,geoid,units,npts)
[latc,lonc] = scircle2(track,lat1,,lon1,lat2,lon2...)
```

Background

A small circle is the locus of all points an equal surface distance from a given center. For true small circles, this distance is always calculated in a great circle sense; however, the `scircle2` command allows a locus to be calculated using distances in a rhumb line sense as well. An example of a small circle is *all points exactly 100 miles from the Washington Monument*.

Description

`[latc,lonc] = scircle2(lat1,lon1,lat2,lon2)` returns the coordinates of points along small circles centered at the points provided in `lat1` and `lon1`, which pass through the points provided in `lat2` and `lon2`. The coordinates of multiple small circles are stored in separate columns of `latc` and `lonc`.

`[latc,lonc] = scircle2(lat1,lon1,lat2,lon2,units)` specifies the units for the inputs and outputs, where *units* is any valid angle units string. The default value is `'degrees'`.

`[latc,lonc] = scircle2(lat1,lon1,lat2,lon2,geoid)` specifies the elliptical definition of the Earth to be used with the two-element `geoid` vector. The default geoid model is the sphere, which is sufficient for most applications.

`[latc,lonc] = scircle2(lat1,lon1,lat2,lon2,geoid,units,npts)` specifies the number of output points, `npts`, returned per small circle. The default value of `npts` is 100.

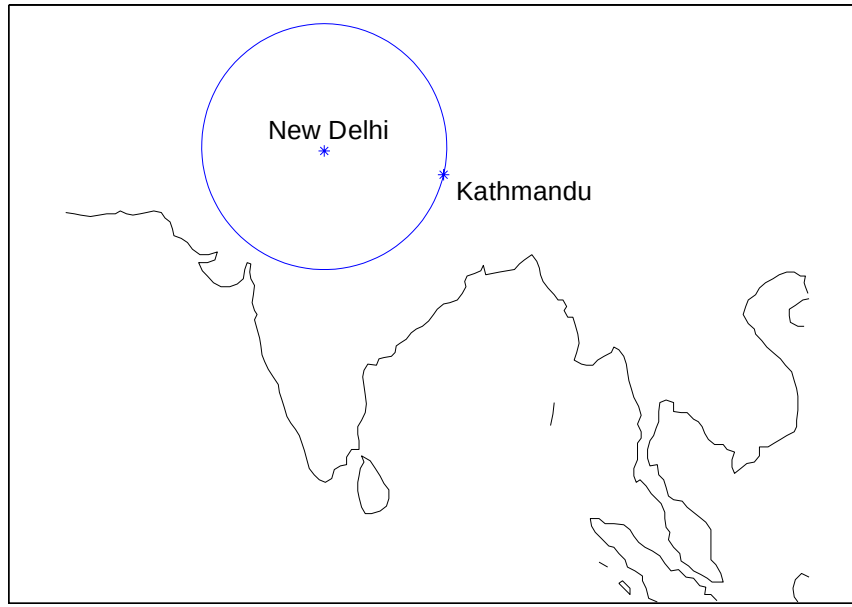
`[latc,lonc] = scircle2(track,lat1,lon1,lat2,lon2...)` specifies the logic with which ranges are calculated. If the string track is `'gc'` (the default), great circle distance is used. It track is `'rh'`, rhumb line distance is used.

`pts = scircle2(lat1,lon1,lat2,lon2)` returns the points in a two-column output `pts`.

Examples

Plot the locus of all points the same distance from New Delhi as Kathmandu:

```
axesm('mercator','MapLatlimit',[0 40],'MapLonLimit',[60 110]);
load coast
plotm(lat,long,'k');    % For reference
lat1 = 29; lon1 = 77.5;    % New Delhi
lat2 = 27.6; lon2 = 85.5;  % Kathmandu
plotm([lat1 lat2],[lon1 lon2],'b*')   % Plot the cities
[latc,lonc] = scircle2(lat1,lon1,lat2,lon2);
plotm(latc,lonc,'b')
```



See Also

`scircle1`	Small circle from center and radius
`track`	Connect waypoints with track segments
`track1` `track2`	Great circles and rhumb lines

Purpose

Display small circle defined via mouse clicks

Syntax

```
h = scircleg(ncirc)
h = scircleg(ncirc,npts)
h = scircleg(ncirc,linestyle)
h = scircleg(ncirc,npts,linestyle)
h = scircleg(ncirc,PropertyName,PropertyValue,...)
h = scircleg(ncirc,npts,PropertyName,PropertyValue,...)
[lat,lon] = scircleg(ncirc,npts,PropertyName,PropertyValue,...)

h = scircleg(track,ncirc,...)
```

Background

A small circle is the locus of all points an equal surface distance from a given center. For true small circles, this distance is always calculated in a great circle sense; however, the `scircleg` command allows a locus to be calculated using distances in a rhumb line sense as well.

Description

This function is used to define small circles for display using mouse clicks. For each circle, two clicks are required: one to mark the center of the circle and one to mark any point on the circle itself, thereby defining the radius.

`h = scircleg(ncirc)` brings forward the current map axes and waits for the user to make (2 x `ncirc`) mouse clicks. The output `h` is a vector of handles for the `ncirc` small circles, which are then displayed.

`h = scircleg(ncirc,npts)` specifies the number of plotting points to be used for each small circle. `npts` is 100 by default.

`h = scircleg(ncirc,linestyle)` specifies the line style for the displayed small circles, where *linestyle* is any line style string recognized by the standard MATLAB function `line`.

`h = scircleg(ncirc,PropertyName,PropertyValue,...)` allows property/value pairs to be set, where *PropertyName* and `PropertyValue` are recognized by the `line` command.

`[lat,lon] = scircleg(ncirc,npts,...)` returns the coordinates of the plotted points rather than the handles of the small circles. Successive circles are stored in separate columns of `lat` and `lon`.

`h = scircleg(track,ncirc,...)` specifies the logic with which ranges are calculated. If the string track is `'gc'` (the default), great circle distance is used. It track is `'rh'`, rhumb line distance is used.

See Also

`scircle1`	Small circle from center, azimuth, and radius
`scircle2`	Small circle from center and perimeter point

Purpose

Provide intersection coordinates for pairs of small circles

Syntax

```
[newlat,newlon] = scxsc(lat1,lon1,range1,lat2,lon2,range2)
[newlat,newlon] = scxsc(lat1,lon1,range1,lat2,lon2,range2,units)
```

Description

For any pair of small circles, there are four possible intersection conditions: the circles are identical, they do not intersect, they are tangent to each other and hence they intersect once, or they intersect twice.

Small circle notation consists of a center point and a radius in units of angular arc length.

`[newlat,newlon] = scxsc(lat1,lon1,range1,lat2,lon2,range2)` returns in `newlat` and `newlon` the locations of the points of intersection of two small circles in *small circle notation*. For example, the first small circle in a pair would be centered on the point (`lat1`,`lon1`) with a radius of `range1` (in angle units). The inputs must be column vectors. If the circles do not intersect, or are identical, two `NaN`s are returned and a warning is displayed. If the two circles are tangent, the single intersection point is repeated twice.

`[newlat,newlon]=scxsc(lat1,lon1,range1,lat2,lon2,range2,units)` specifies the angle units used for all inputs, where *units* is any valid angle units string. The default units are `'degrees'`.

Examples

Given a small circle centered at (10°S,170°W) with a radius of 20° (~1200 nautical miles), where does it intersect with a small circle centered at (3°N, 179°E), with a radius of 15° (~900 nautical miles)?

```
[newlat,newlong] = scxsc(-10,-170,20,3,179,15)
newlat =
   -8.8368    9.8526
newlong =
   169.7578 -167.5637
```

Note that in this example, the two small circles cross the Dateline.

Remarks

Great circles are a subset of small circles – a great circle is just a small circle with a radius of 90°. This provides two methods of notation for defining great circles. *Great circle notation* consists of a point on the circle and an azimuth at that point. *Small circle notation* for a great circle consists of a center point and a radius of 90° (or its equivalent in radians or *dms* units).

See Also

`gc2sc`	Convert great circle to small circle notation

`gcxgc`
`gcxsc`
`rhxrh`
`crossfix`
	Other intersection functions

Purpose

Convert time units from seconds to *hms* or *hm*

Syntax

```
timeout = sec2hms(timein)
timeout = sec2hm(timein)
```

Description

`timeout = sec2hms(timein)` converts times input in seconds to the equivalent measure in the hours-minutes-seconds (“hms”) format.

`timeout = sec2hm(timein)` converts times input in seconds to the equivalent measure in the hours-minutes (“hm”) format. This is the “hms” format, properly rounded to just hours and minutes.

Example

```
sec2hms(3661)
ans =
        101.01

sec2hm(3661)
ans =
        101.00
```

See Also

`hms2mat`	Convert from *hms* to separated matrix components
`mat2hms`	Convert from separated matrices input to *hms*
`sec2hm` `hr2sec`	Other direct time conversion functions
`timedim`	Convert times between different units

sec2hr

Purpose

Convert time from seconds to hours

Syntax

```
timeout = sec2hr(timein)
```

Description

`timeout = sec2hr(timein)` converts times input in seconds to the equivalent measure in hours.

Example

```
sec2hr(1000)
ans =
    0.2778
```

See Also

`hr2sec` `hms2sec`	Other direct time conversion functions
`timedim`	Convert times between different units

Purpose

Get latitudes and longitudes from matrix row and column coordinates

Syntax

```
[lat,lon] = setltln(map,maplegend,row,col)
mat = setltln(map,maplegend,row,col)
[lat,lon,badindx] = setltln(map,maplegend,row,col)
```

Description

`[lat,lon] = setltln(map,maplegend,row,col)` returns the latitude and longitudes associated with the input row and column coordinates of the input `map` with a map legend vector of `maplegend`.

`mat = setltln(map,maplegend,row,col)` returns the coordinates in a single, two-column matrix of the form `[latitude longitude]`.

`[lat,lon,badindx] = setltln(map,maplegend,row,col)` returns the indices of the elements of the `row` and `col` vectors that lie outside the input map. The outputs `lat` and `lon` always ignore these points; the third output accounts for them.

Examples

Find the coordinates of row 45, column 65 of `topo`:

```
load topo
[lat,lon,badindx] = setltln(topo,topolegend,45,65)
lat =
  -45.5000
lon =
   64.5000
badindx =
     []      % Empty since the point is valid
```

See Also

`ltln2val`	Latitude and longitude to matrix entry value
`setpostn`	Latitude and longitude to row and column

setm

Purpose

Modify the properties of a displayed map

Syntax

```
setm(axishndl,PropertyName,PropertyValue,...)
setm(texthndl,'MapPosition',position)
setm(surfhndl,'Graticule',lat,lon,alt)
setm(surfhndl,'MeshGrat',npts,alt)
```

Description

`setm(axishndl,PropertyName,PropertyValue,...)` sets the properties of the map axes specified by its handle to the given values. The map properties must be recognized by `axesm`.

`setm(texthndl,'MapPosition',position)` alters the position of the projected text object specified by its handle to the `[latitude longitude]` or the `[latitude longitude altitude]` specified by the `position` vector.

`setm(surfhndl,'Graticule',lat,lon,alt)` alters the graticule of the projected surface object specified by its handle. The graticule is specified by the latitude and longitude matrices, specifying locations of the graticule vertices. The altitude can be specified by a scalar, or by a matrix providing a value for each vertex.

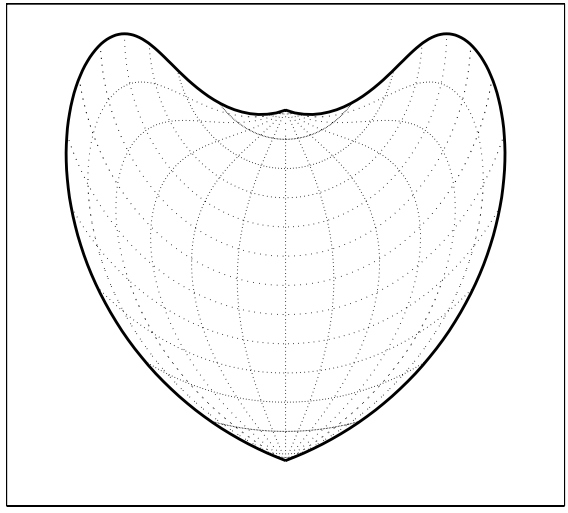
`setm(surfhndl,'MeshGrat',npts,alt)` alters the mesh graticule of projected surface objects displayed using the `meshm` command. In this case, the two-element vector `npts` specifies the graticule size in the manner described under `meshm`. The altitude can be a scalar or a matrix with a size corresponding to `npts`.

Examples

Display a map axes and alter it:

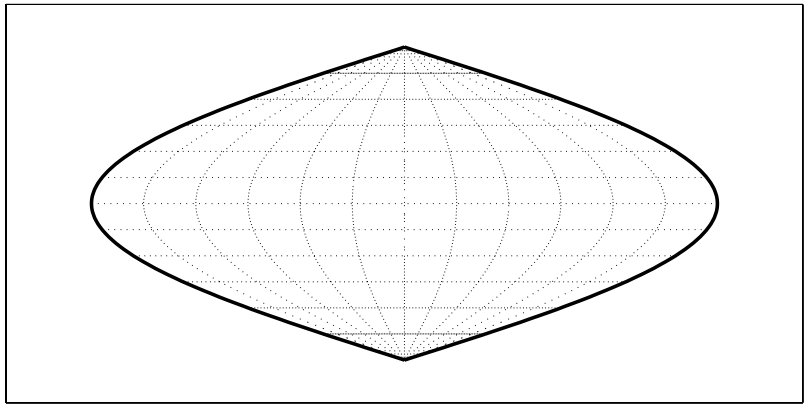
```
axesm('bonne','Frame','on','Grid','on')
```

The standard Bonne projection has a standard parallel at 30°N.



Setting this standard parallel to 0° results in a Sinusoidal projection:

```
setm(gca,'MapParallels',0)
```



See Also

`axesm`	Create and set map axes properties
`getm`	Get map object properties

Purpose

Convert latitude and longitude coordinates to matrix map row and column

Syntax

```
[row,col] = setpostn(map,maplegend,lat,long)
indx = setpostn(map,maplegend,lat,long)
[row,col,badindx] = setpostn(map,maplegend,lat,long)
```

Description

`[row,col] = setpostn(map,maplegend,lat,long)` returns the row and column indices of the input regular matrix `map` with a map legend vector of `maplegend` for the points specified by the vectors `lat` and `long`. All angles are in degrees.

`indx = setpostn(map,maplegend,lat,long)` returns the single-value indices of `map(:)`.

`[row,col,badindx] = setpostn(map,maplegend,lat,long)` also returns the indices of `lat` and `long` corresponding to points outside of `map`. These points are always ignored in `row` and `col`.

Examples

What are the matrix coordinates in topo of Denver, Colorado, at (39.7°N,105°W)?

```
load topo
[row,col] = setpostn(topo,topolegend,39.7,105)
row =
   130
col =
   105
```

See Also

`ltln2val`	Latitude and longitude to matrix entry value
`setltln`	Row and column to latitude and longitude

shaderel

Purpose

Construct cdata and colormap for colored shaded relief

Syntax

```
[cindx,cimap,clim] = shaderel(X,Y,Z,cmap)
[cindx,cimap,clim] = shaderel(X,Y,Z,cmap,[azim elev])
[cindx,cimap,clim] = shaderel(X,Y,Z,cmap,[azim elev],cmapl)
[cindx,cimap,clim] = shaderel(X,Y,Z,cmap,[azim elev],cmapl,clim)
```

Description

`[cindx,cimap,clim] = shaderel(X,Y,Z,cmap)` constructs the colormap and color indices to allow a surface to be displayed in colored shaded relief. The colors are proportional to the magnitude of `Z`, but modified by shades of gray based on the surface normals to simulate surface lighting. This representation allows both large and small-scale differences to be seen. `X`, `Y`, and `Z` define the surface. `cmap` is the colormap used to create the new shaded colormap `cimap`. `cindx` is a matrix of color indices to `cimap`, based on the elevation and surface normal of the `Z` matrix element. `clim` contains the color axis limits.

`[cindx,cimap,clim] = shaderel(X,Y,Z,cmap,[azim elev])` places the light at the specified azimuth and elevation. By default, the direction of the light is East (90° azimuth) at an elevation of 45°.

`[cindx,cimap,clim] = shaderel(X,Y,Z,cmap,[azim elev],cmapl)` chooses the number of grays to give a `cimap` of length `cmapl`. By default, the number of grayscales is chosen to keep the shaded colormap below 256. If the vector of azimuth and elevation is empty, the default locations are used.

`[cindx,cimap,clim] = shaderel(X,Y,Z,cmap,[azim elev],cmapl,clim)` uses the color limits to index `Z` into `cmap`.

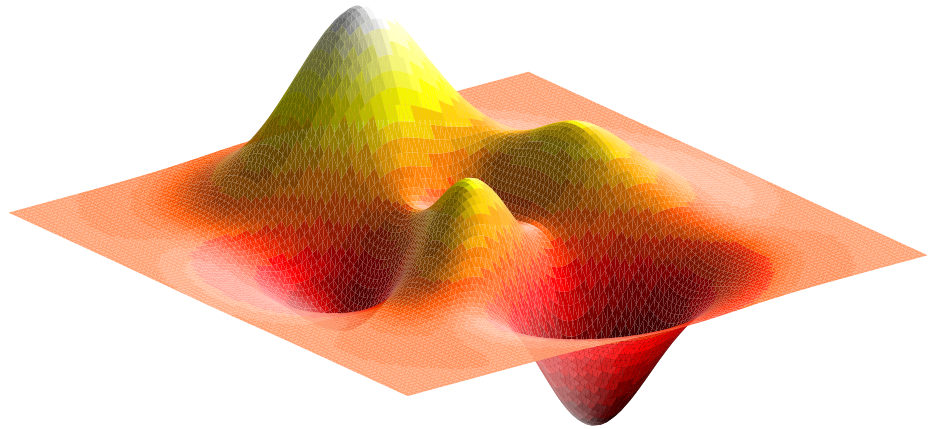
Remarks

This function effectively multiplies two colormaps, one with color based on elevation, the other with a grayscale based on the slope of the surface, to create a new color map. This produces an effect similar to using a light on a surface, but with all of the visible colors actually in the colormap. Lighting calculations are performed on the unprojected data.

Examples

Display the peaks surface with a shaded colormap:

```
[X,Y,Z] = peaks(100);
cmap = hot(16);
[cindx,cimap,clim] = shaderel(X,Y,Z,cmap);
surf(X,Y,Z,cindx); colormap(cimap); caxis(clim)
shading flat
```



See Also

`caxis`	Pseudocolor axis scaling
`colormap`	Color lookup table
`light`	Create a Light object see online *MATLAB Function Reference*)
`meshlsrm`	Project 3-D lighted shaded relief of a regular matrix map
`surf`	3-D shaded surface plot (see online *MATLAB Function Reference*)
`surflsrm`	Project 3-D lighted shaded relief of a general matrix map

showaxes

Purpose

Toggle display of Cartesian MATLAB axes

Syntax

```
showaxes
showaxes('on')
showaxes('off')
showaxes('hide')
showaxes('show')
showaxes('reset')
showaxes(color)
showaxes(color)
```

Description

`showaxes` toggles the visibility of the axes between the `'on'` and `'off'` conditions.

`showaxes('on')` sets the color of the axes (the `XColor` and `YColor` properties) to black.

`showaxes('off')` sets the color of the axes (the `XColor` and `YColor` properties) to the background color, effectively making them invisible. This is the default condition for map axes.

`showaxes('hide')` sets the `Visible` property of the axes to `'on'`.

`showaxes('show')` sets the `Visible` property of the axes to `'off'`.

`showaxes('reset')` sets the axes properties to the default map axes settings.

`showaxes(`*`color`*`)` sets the color of the axes (the `XColor` and `YColor` properties) to the color specified by any valid color string.

`showaxes(color)` sets the color of the axes (the `XColor` and `YColor` properties) to the color specified by the input RGB triple.

See Also

`axesm` Map axes creation

Purpose

Show specified graphics objects

Syntax

```
showm
showm(handle)
showm(object)
```

Description

`showm` brings up a dialog box for selecting the objects to show (set their `Visible` property to `'on'`).

`showm(handle)` shows the objects specified by a vector of handles.

`showm(object)` shows those objects specified by the object string, which can be any string recognized by the `handlem` command.

See Also

`clma`	Clear current map
`clmo`	Clear specified graphics objects
`handlem`	Get handles of displayed map objects
`hidem`	Hide specified graphics objects
`namem`	Determine names of valid graphics objects
`tagm`	Assign a name to graphics object tag property

sizem

Purpose

Determine row and column dimensions needed for a regular matrix map

Syntax

```
[r,c] = sizem(latlim,lonlim,scale)
rc = sizem(latlim,lonlim,scale)
[r,c,maplegend] = sizem(latlim,lonlim,scale)
```

Description

`[r,c] = sizem(latlim,lonlim,scale)` returns the required size for a regular matrix map lying between the latitude and longitude limits specified by the two-element input vectors `latlim` and `lonlim`, which are of the form `[south-limit north-limit]` and `[west-limit and east-limit]`, respectively. The `scale` is the desired "cells-per-degree" measure of the desired matrix map.

`rc = sizem(latlim,lonlim,scale)` returns the size of the matrix in one two-element vector.

`[r,c,maplegend] = sizem(latlim,lonlim,scale)` also returns the map legend vector for the desired regular matrix map.

Examples

How large a matrix would be required for a map of the world at a scale of 25 matrix cells per degree? (That's 25x25=625 cells per "square" degree.)

```
[r,c] = sizem([90,-90],[-180,180],25)
r =
        4500
c =
        9000
```

Bear in mind for memory purposes – 9000x4500=4.05x10^7 entries!

See Also

`findm`	Coordinates of nonzero map entries
`limitm`	Matrix map limits
`map legend vector`	Data structure for describing regular matrix maps

`nanm`
`onem`
`spzerom`
`zerom`
	Create special maps

Purpose

Convert distance from statute miles to other units

Syntax

```
distout = sm2deg(distin)
distout = sm2deg(distin,radius)

distout = sm2km(distin)

distout = sm2rad(distin)
distout = sm2rad(distin,radius)

distout = sm2nm(distin)
```

Description

`distout = sm2deg(distin)` converts the input distance given in statute miles to degrees. `distout = sm2km(distin)`, `distout = sm2rad(distin)`, and `distout = sm2nm(distin)` perform analogously, converting to kilometers, radians, and nautical miles, respectively.

`distout = sm2deg(distin,radius)` and `distout = sm2rad(distin,radius)` specify the radius of the sphere to use, since a degree (or radian) of arc length covers less distance, for example, on Mars than it does on the Earth. You can enter the radius as a number in statute miles, as a call to the `almanac` function (e.g., `almanac('mars','radius','sm')`), or you can pass in a string planet name (e.g., `'mars'`), and the function will make the appropriate call to the `almanac` function. The radius of the Earth is the default.

Examples

In track, is the quarter mile exactly the same as the 400-meter?

```
distout = sm2km(0.25)
distout =
    0.4023
```

No, it's 2.3 meters longer.

See Also

`distdim`	Convert distances between different units
`km2sm` `nm2deg`	Other direct distance conversion functions

smoothlong

Purpose

Remove discontinuities in longitudes

Syntax

```
ang = smoothlong(angin)
ang = smoothlong(angin,units)
```

Description

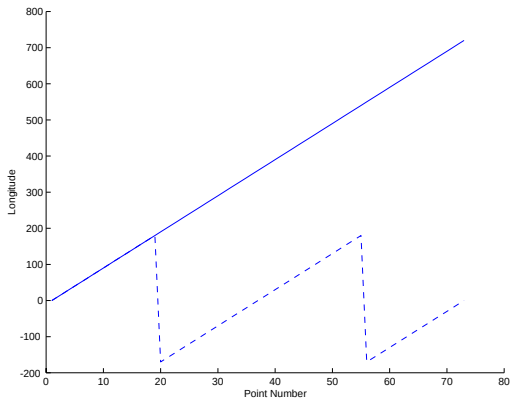
`ang = smoothlong(angin)` removes discontinuities in longitude data. The resulting angles may cover more than one revolution.

`ang = smoothlong(angin,`*`units`*`)` removes

`mat = spcread(cols)` uses the units defined by the input string *units*. If omitted, default units of 'degrees' are assumed.

Examples

```
long = npi2pi(0:10:720);
long2 = smoothlong(long);
figure;hold on
plot(long,'--'); plot(long2)
xlabel 'Point Number'; ylabel Longitude
```



Remarks

This function can remove large jumps in longitude that might otherwise result in spurious data when interpolating with `interpm`.

See Also

`npi2pi`	Truncates angles into the –180 deg to 180 deg range
`zero22pi`	Truncates angles into the 0 deg to 360 deg range
`interpm`	Interpolates vector data to a specified data separation

spcread

Purpose

Read an ASCII file of space-delimited data columns

Syntax

```
mat = spcread
mat = spcread(filename)
mat = spcread(cols)
mat = spcread(filename,cols)
```

Description

`mat = spcread` reads an ascii file of space delimited data in two columns and returns the data in a matrix `mat`. The file is selected by dialog box.

`mat = spcread(filename)` specifies the file from which to read by its name, given as the string *filename*.

`mat = spcread(cols)` specifies the number of columns of space delimited data in the file with the integer `cols`. The default value of `cols` is 2.

Remarks

The functionality of this command is similar to the standard MATLAB command `dlmread`. `spcread`, however, is much faster at reading large data sets of the type common for geographic purposes.

See Also

`nanclip` Convert pen-down delimited data to `NaN`-delimited data

Purpose

Create a sparse matrix map of zeros

Syntax

```
map = spzerom(latlim,lonlim,scale)
[map,maplegend] = spzerom(latlim,lonlim,scale)
```

Description

`map = spzerom(latlim,lonlim,scale)` returns a sparse regular matrix map consisting entirely of zeros. The two-element vectors `latlim` and `lonlim` define the latitude and longitude limits of the geographic region. They should be of the form `[south north]` and `[west east]`, respectively. The number of rows and columns per angle unit is set by the scalar `scale`.

`[map,maplegend] = spzerom(latlim,lonlim,scale)` returns the three-element map legend vector for the returned `map`.

Examples

```
[map,maplegend] = spzerom([46,51],[-79,-75],1)
map =
   All zero sparse: 5-by-4
maplegend =
     1    51   -79
```

See Also

`limitm`	Matrix map limits
`map legend vector`	Data structure for describing regular matrix maps
`nanm`	Create a matrix map of `NaN`s
`onem`	Create a matrix map of ones
`sizem`	Rows and columns needed for map
`zerom`	Create a full matrix map of zeros

stdist

Purpose

Compute standard distance of geographic data

Syntax

```
dist = stdist(lat,lon)
dist = stdist(lat,lon,units)
dist = stdist(lat,lon,geoid)
dist = stdist(lat,lon,geoid,units)
dist = stdist(lat,lon,geoid,units,method)
```

Background

The command `stdm` provides independent standard deviations in latitude and longitude of data points. `stdist` provides a means of examining data scatter that does not separate these components. The result is a *standard distance*, which can be interpreted as a measure of the scatter in great circle distance of the data points from the centroid as returned by `meanm`.

Description

`dist = stdist(lat,lon)` returns a row vector of the latitude and longitude geographic standard distance for the data points specified by the columns of `lat` and `lon`.

`dist = stdist(lat,lon,units)` indicates the angular units of the data. When the standard angle string *units* is omitted, `'degrees'` is assumed. Output measurements are in terms of these *units* (as arc length distance).

`dist = stdist(lat,lon,geoid)`specifies the elliptical definition of the Earth to be used with the two-element `geoid` vector. The default geoid model is a spherical Earth, which is sufficient for most applications. Output measurements are in terms of the distance units of the `geoid` vector.

`dist = stdist(lat,lon,geoid,units,method)` specifies the method of calculating the standard distance of the data. The default, `'linear'`, is simply the average great circle distance of the data points from the centroid. Using `'quadratic'` results in the square root of the average of the squared distances, and `'cubic'` results in the cube root of the average of the cubed distances.

The output distance can be thought of as the radius of a circle centered on the geographic mean position, which gives a measure of the spread of the data.

Examples

Generate latitude and longitude matrices using the `worldo` dataset (compare with example for `stdm`):

```
load worldlo
lat=[PPpoint(2:5).lat]
lat =
    5.3386   24.5808    5.4619   36.8862

lon=[PPpoint(2:5).long]
lon =
   -3.7602   54.9619   -0.3450   35.5141

dist= stdist(lat,lon)
dist =
   26.5282
```

See Also

`meanm`	Mean of geographic data
`stdm`	Standard deviation of geographic data

stdm

Purpose

Compute standard deviation for geographic data

Syntax

```
[latdev,londev] = stdm(lat,lon)
[latdev,londev] = stdm(lat,lon,geoid)
[latdev,londev] = stdm(lat,lon,units)
[latdev,londev] = stdm(lat,lon,geoid,units)
geodevs = stdm(...)
```

Background

Determining the deviations of geographic data in latitude and longitude is more complicated than simple sum-of-squares deviations from the data averages. For latitude deviation, a straightforward angular standard deviation calculation is performed from the *geographic mean* as calculated by `meanm`. For longitudes, a similar calculation is performed based on data *departure* rather than on angular deviation. See the section entitled "Geostatistics" in Chapter 5, "Mapping Applications," of the *Mapping Toolbox User's Guide*.

Description

`[latdev,londev] = stdm(lat,lon)` returns row vectors of the latitude and longitude geographic standard deviations for the data points specified by the columns of `lat` and `lon`.

`[latdev,londev] = stdm(lat,lon,units)` indicates the angular units of the data. When the standard angle string *units* is omitted, `'degrees'` is assumed. Output measurements are in terms of these *units* (as arc length distance).

`[latdev,londev] = stdm(lat,lon,geoid)` specifies the elliptical definition of the Earth to be used with the two-element `geoid` vector. The default geoid model is a spherical Earth, which is sufficient for most applications. Output measurements are in terms of the distance units of the `geoid` vector.

If a single output argument is used, then `geodevs = [latdev longdev]`. This is particularly useful if the original `lat` and `lon` inputs are column vectors.

Examples

Create latitude and longitude matrices using the `worldo` dataset (compare with example for `stdist`):

```
load worldlo
lat=[PPpoint(2:5).lat]
lat =
    5.3386   24.5808    5.4619   36.8862

lon=[PPpoint(2:5).long]
lon =
   -3.7602   54.9619   -0.3450   35.5141

[latstd,lonstd]=stdm(lat,lon)
latstd =
    8.9962
lonstd =
  159.1153
```

See Also

`departure`	Departure of longitudes at specific latitudes
`filterm`	Geographic filter for data sets
`hista`	Spatial equal area histogram
`histr`	Spatial equirectangular histogram
`meanm`	Mean for geographic data
`stdist`	Standard distances for geographic data

stem3m

Purpose

Project a stem plot map on the current map axes

Syntax

```
h = stem3m(lat,lon,z)
h = stem3m(lat,lon,z,LineType)
h = stem3m(lat,lon,z,PropertyName,PropertyValue,...)
```

Description

A stem plot displays data as lines extending normal to the *xy*-plane, in this case, on a map.

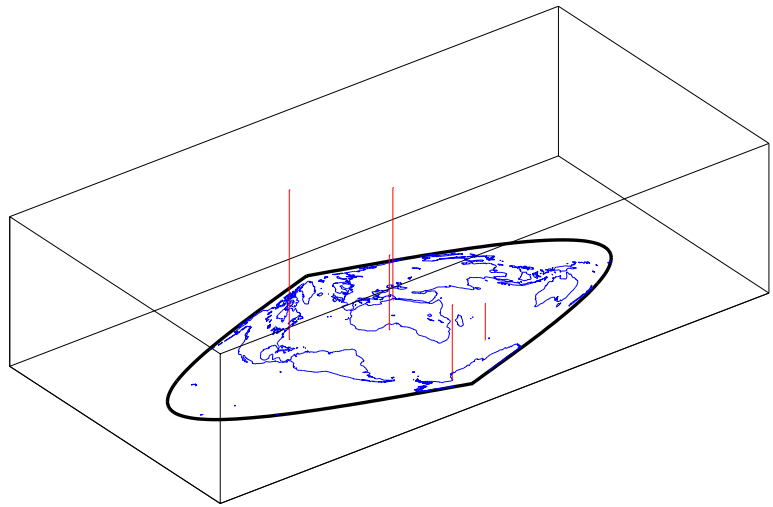
`h = stem3m(lat,lon,z)` displays a stem plot on the current map axes. Stems are located at the points (`lat`,`lon`) and extend from an altitude of 0 to the values of `z`. The coordinate inputs should be in the same `AngleUnits` as the map axes. It is important to note that the selection of *z*-values will greatly affect the 3-D look of the plot. Regardless of `AngleUnits`, the *x* and *y* limits of the map axes are at most $-\pi$ to $+\pi$ and $-\pi/2$ to $+\pi/2$, respectively. This means that for most purposes, appropriate z values would be on the order of 1 to 3, not 10 to 30. The axes `DataAspectRatio` property can be used to adjust the appearance of the graphic. The handles of the displayed stem lines can be returned in `h`.

`h = stem3m(lat,lon,z,LineType)` allows the style of the stem plot's lines to be specified with any string *LineType* recognized by the MATLAB `line` command.

`h = stem3m(lat,lon,z,PropertyName,PropertyValue,...)` allows any property/value pair recognized by the MATLAB line command to be specified for the stems.

Examples

```
load coast
axesm sinusoid; view(3)
h = framem; set(h,'zdata',zeros(size(lat)))
plotm(lat,long)
ptlat = [0 30 30 -50 -78]';
ptlon = [0 30 -70 65 -35]';
ptz = [1 1.5 2 .5 1]';
stem3m(ptlat,ptlon,ptz,'r-')
```



See Also

scatterm — Thematic map with proportional symbols

surfacem

Purpose

Warp matrix map to a projected graticule mesh

Syntax

```
h = surfacem(map)
h = surfacem(map,npts)
h = surfacem(lat,lon,map)
h = surfacem(lat,lon,map,PropertyName,PropertyValue,...)
h = surfacem(lat,lon,map,alt)
h = surfacem(lat,lon,map,alt,PropertyName,PropertyValue,...)
```

Description

Unlike `meshm` and `surfm`, this command adds surfaces to the current axes, regardless of hold state. This command warps a matrix map to a graticule mesh, which itself is projected according to the map axes `MapProjection` property. The fineness, or resolution of this grid determines the quality of the projection and the speed of plotting it. There is no hard and fast rule for sufficient graticule resolution, but in general, cylindrical projections need very few graticules points in the longitudinal direction, while complex curve-generating projections require more.

`h = surfacem(map)` projects the matrix map `map` on a graticule grid the size of `map` between the latitude and longitude limits of the current map axes. The handle `h` of the displayed surface can be returned.

`h = surfacem(map,npts)` results in a graticule grid defined by npts, which is a two element vector of the form `[latitude-points longitude-points]`. The default `npts` is `[50 100]` (the graticule has 50 vertices in the latitude direction and 100 vertices in the longitude direction).

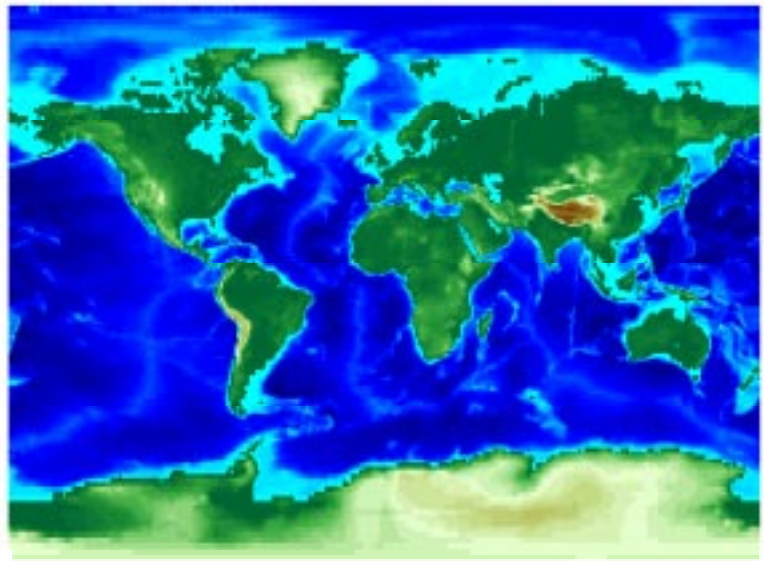
`h = surfacem(lat,lon,map)` allows greater graticule control through the inputs `lat` and `lon`. If matrices, they are the graticule vertex coordinates as might be returned by `meshgrat`. If vectors, they are the representative coordinates of the rows and columns, respectively, of such a grid. If they are two-element vectors, they are treated as latitude and longitude limits, and a graticule mesh the size of the default `npts` is calculated internally.

`h = surfacem(lat,lon,map,alt)` sets the *z*-axis altitude of the graticule mesh. `alt` must be the same size as `lat`. If no `alt` is supplied, the mesh is plotted at z=0, unless `lat` is the same size as `map`, in which case, `zdata=map`, and a 3-D topographical map results.

`h = surfacem(lat,lon,map,`*`PropertyName`*`,PropertyValue,...)` allows the input of property/value pairs to control the surface object properties. Any property supported by the standard MATLAB command `surface` except `XData`, `YData`, and `ZData` can be altered in this manner.

Examples

```
load topo
axesm miller
surfacem(topo,[30 30])
demcmap(topo)
```



See Also

`meshgrat`	Construct map graticule grid
`meshm`	Regular matrix map warped to projected graticule mesh
`pcolorm`	Projected matrix map in the z=0 plane
`surfm`	Matrix map projected on a map axes

surflm

Purpose

Project three-dimensional shaded surface with lighting on a map axes

Syntax

```
h = surflm(map)
h = surflm(map,s)
h = surflm(lat,lon,map)
h = surflm(lat,lon,map,s)
h = surflm(lat,lon,map,s,k)
```

Description

`surflm` is like `surfm` except that it shades the monochrome map surface with a light source, and the only allowed graticule is the size of the map matrix.

`h = surflm(map)` displays the matrix map `map` projected to a graticule grid the size of `map` in accordance with the current map axes `MapProjection` property. It is displayed with a default light source. The handle `h` of the displayed surface object can be returned.

`h = surflm(map,s)` specifies the direction of the light source. `s` is a two- or three-element vector that specifies the direction from the surface map to the light source. `s=[sx sy sz]` or `s=[azimuth elevation]`. The default `s` is 45° counterclockwise from the current view direction.

`h = surflm(lat,lon,map)` allows you to specify your graticule. `lat` and `lon` can be vectors with elements corresponding to `map` rows and columns, respectively, or they can be matrices the size of `map`. The resulting graticule will be the size of `map`.

`h = surflm(lat,lon,map,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]`.

Examples

To see this, type the following. The graticule is the size of topo (180 x 360) and is rendered in 3-D, so it might take a while. It is also memory intensive:

```
load topo
axesm miller
surflm(topo)
```

See Also

`surfm` Matrix map projected on map axes

Purpose

Project 3-D lighted shaded relief of a general matrix map

Syntax

```
surflsrm(lat,long,map)
surflsrm(lat,long,map,[azim elev])
surflsrm(lat,long,map,[azim elev],cmap)
surflsrm(lat,long,map,[azim elev],cmap,clim)
h = surflsrm(...)
```

Description

`surfsrlm(lat,long,map)` displays the general matrix map colored according to elevation and surface slopes. The current axes must have a valid map projection definition.

`surfsrlm(lat,long,map,[azim elev])` displays the general matrix map with the light coming from the specified azimuth and elevation. Lighting is applied before the data is projected. Angles are in degrees, with the azimuth measured clockwise from North, and elevation up from the zero plane of the surface. By default, the direction of the light source is east (90° azimuth) at an elevation of 45°.

`surfsrlm(lat,long,map,[azim elev],cmap)` displays the general matrix map using the provided colormap. The number of grayscales is chosen to keep the size of the shaded colormap below 256. By default, the colormap is constructed from 16 colors and 16 grays. If the vector of azimuth and elevation is empty, the default locations are used.

`surfsrlm(lat,long,map,[azim elev],cmap,clim)` uses the provided color axis limits, which by default, are automatically computed from the data.

`h = surfsrlm(...)` returns the handle to the surface drawn.

Remarks

This function effectively multiplies two colormaps, one with color based on elevation, the other with a grayscale based on the slope of the surface, to create a new colormap. This produces an effect similar to using a light on a surface, but with all of the visible colors actually in the colormap. Lighting calculations are performed on the unprojected data.

Examples

Create a new colormap using demcmap with white colors for the sea and default colors for land. Use this colormap for the lighted shaded relief map of the Middle East region:

```
load mapmtx
[cmap,clim] = demcmap(map1,[],[1 1 1],[]);
axesm loximuth
surflsrm(lt1,lg1,map1,[],cmap,clim)
```

See Also

`meshlsrm`	Project 3-D lighted shaded relief of a regular matrix map
`meshm`	Display a regular matrix map warped to a projected graticule
`pcolorm`	Projected matrix map in the z = 0 plane
`shaderel`	Construct cdata and colormap for colored shaded relief
`surfacem`	Display a matrix map warped to a projected graticule
`surflm`	Display a lighted matrix map warped to a projected graticule
`surfm`	Display a matrix map warped to a projected graticule

surfm

Purpose

Project matrix map on a map axes

Syntax

```
h = surfm(map)
h = surfm(map,npts)
h = surfm(lat,lon,map)
h = surfm(lat,lon,map,alt)
h = surfm(lat,lon,map,PropertyName,PropertyValue,...)
h = surfm(lat,lon,map,alt,PropertyName,PropertyValue,...)
```

Description

This command warps a matrix map to a graticule mesh, which itself is projected according to the map axes property `MapProjection`. The fineness, or resolution of this grid determines the quality of the projection and the speed of plotting it. There is no hard and fast rule for sufficient graticule resolution, but in general, cylindrical projections need very few graticules points in the longitudinal direction, while complex curve-generating projections require more.

`h = surfm(map)` projects the matrix map `map` on a graticule grid the size of `map` between the latitude and longitude limits of the current map axes. The handle `h` of the displayed surface can be returned.

`h = surfm(map,npts)` results in a graticule grid defined by npts, which is a two element vector of the form `[latitude-points longitude-points]`.

`h = surfm(lat,lon,map)` allows three other methods of defining the graticule grid. If `lat` and `lon` are matrices, they represent the actual graticule vertices as might be returned by `meshgrat`. If vectors, they are the representative coordinates of the rows and columns, respectively, of such a grid. If they are two-element vectors, they are treated as latitude and longitude limits, and a graticule mesh of `size(map)` is calculated.

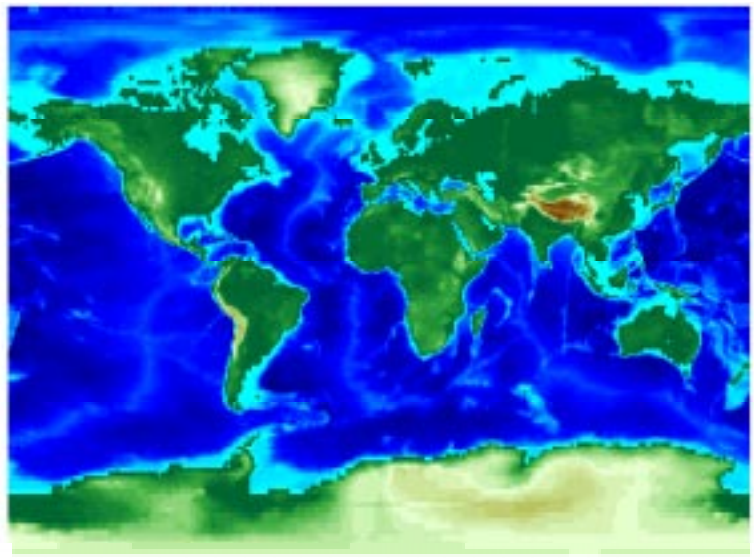
`h = surfm(lat,lon,map,alt)` sets the *z*-axis altitude of the graticule mesh. `alt` must be the same size as `lat`. If no `alt` is supplied, the mesh is plotted at z=0, unless `lat` is the same size as `map`, in which case `zdata=map`, and a 3-D topological map results. Since the default graticule is the size of `map`, the default condition for `surfm` is to create the topographic map.

`h = surfm(lat,lon,map,PropertyName,PropertyValue,...)` allows the input of property/value pairs to control the surface object properties. Any property

supported by the standard MATLAB command surface except XData, YData, and ZData can be altered in this manner.

Examples

```
load topo
axesm miller
[meshlat,meshlon] = meshgrat(topo,topolegend,[90 180]);
surfm(meshlat,meshlon,topo)
demcmap(topo)
```



See Also

meshgrat	Construct map graticule grid
meshm	Regular matrix map warped to projected graticule mesh
pcolorm	Projected matrix map in the z = 0 plane
surfacem	Matrix map warped to projected graticule mesh

tagm

Purpose

Assign a name to a graphics object in its tag property

Syntax

```
tagm(hndl,tagstr)
```

Description

`tagm(hndl,tagstr)` sets the `Tag` property of each object designated in the vector of handles `hndl` to the associated string (row) of the matrix of strings `tagstr`.

This property is recognizable to the `namem` and `handlem` functions.

Examples

Normally, a plotted line has a name of `'line'`:

```
axesm miller
lats = [3 2 1 1 2 3]; longs = [7 8 9 7 8 9];
h=plotm(lats,longs);

untagged = namem(h)
untagged =
line
```

The `tagm` command can rename it:

```
tagm(h,'testpath');
tagged = namem(h)
tagged =
testpath
```

See Also

`clma`	Clear current map
`clmo`	Clear specified graphics objects
`handlem`	Get handles of displayed graphics objects
`hidem`	Hide specified graphics objects
`namem`	Determine names of valid graphics objects
`showm`	Show specified graphics objects

Purpose

Project text annotation on map axes

Syntax

```
h = textm(lat,lon,string)
h = textm(lat,lon,string,PropertyName,PropertyValue,...)
h = textm(lat,lon,z,string)
h = textm(lat,lon,z,string,PropertyName,PropertyValue,...)
```

Description

`h = textm(lat,lon,string)` displays the strings (rows) of the matrix of strings *string* at the geographic locations specified by the vectors `lat` and `lon`. The handles `h` of the displayed strings can be returned.

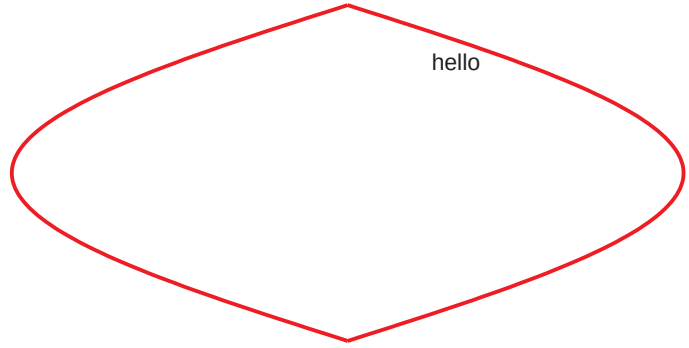
`h = textm(lat,lon,z,string)` displays the strings at the *z*-axis altitudes `z`. The default altitude is 0.

`h = textm(lat,lon,z,string,PropertyName,PropertyValue,...)` sets the text object properties. All properties supported by the MATLAB `text` command are supported by `textm`.

Example

The feature of `textm` that distinguishes it from the standard MATLAB text command is that the `text` object is projected appropriately. Type the following:

```
axesm sinusoid
framem('FEdgeColor','red')
textm(60,90,'hello')
```



```
figure; axesm miller
framem('FEdgeColor','red')
textm(60,90,'hello')
```

The string `'hello'` is placed at the same geographic point, but it appears to have moved relative to the axes because of the different projections. If the projection is changed using the `setm` command, the text will move as necessary. Use `text` to fix text objects in the axes independent of projection.

See Also

`axesm`	Create map axes object
`text`	Create Text object in current Axes (see online *MATLAB Function Reference*)

Purpose

Removes white-space around a map

Syntax

```
tightmap
```

Description

`tightmap` sets the MATLAB axis limits to be tight around the map in the current axes. This eliminates or reduces the white border between the map frame and the axes box. Use `axis auto` to undo `tightmap`.

Examples

Display a map of Africa. Notice the white space between the map frame and the edge of the axes box.

```
axesm('miller','maplatlim',[-40 40],'maplonlim',[-20 60])
framem; gridm; mlabel; plabel
plotm(coast)
```

Now use `tightmap` to reduce the "wasted space":

```
tightmap
```

Limitations

The axis limits are fixed. If a change in the projection parameters changes the size or position of the map display within the projected coordinate system, execute `tightmap` again.

See Also

`panzoom`	Pan and zoom on a 2D plot
`zoom`	Zoom in and out on a 2-D plot
`paperscale`	Figure paper size for a given map scale
`axesscale`	Resize axes for equivalent scale
`previewmap`	View map at printed size

time2str

Purpose

Convert time to a clock string

Syntax

```
str = time2str(timein)
str = time2str(timein,clock)
str = time2str(timein,clock,format)
str = time2str(timein,clock,format,units)
str = time2str(timein,clock,format,digits)
str = time2str(timein,clock,format,units,digits)
```

Description

The purpose of this command is to make time-valued variables into strings suitable for map display.

`str = time2str(timein)` converts a numerical vector of times to a string matrix. The output string matrix is useful for the display of times.

`str = time2str(timein,clock)` uses the specified *clock* input to construct the string matrix. Allowable *clock* strings are `'24'` (default) for a 24-hour clock, `'12'` for a 12-hour clock, and `'nav'` for a navigational hour clock.

`str = time2str(timein,clock,format)` uses the specified *format* input to construct the string matrix. Allowable for *format* strings are `'hms'`, for hours, minutes, and seconds, and `'hm'` (default), for hours and minutes.

`str = time2str(timein,clock,format,units)` defines the units in which the input times are supplied. Any valid time *units* string can be entered. If omitted, `'hours'` is assumed.

`str = time2str(timein,clock,format,digits)` indicates the power of ten to be included for the seconds (if *format* `= 'hms'`) or minutes (if *format* `= 'hm'`). The default value is 0, so nothing is returned to the right of the decimal (10^0 is the ones column). For example, if `digits = -2`, seconds would be returned down to the hundredths column.

Examples

13 hours, 56 minutes, 44 seconds in *hms* format is `1356.44`.

```
time = 1356.44;
str = time2str(time,'12','hms','hms')
str =
1:56:44 PM
```

For *hm* format, appropriate rounding occurs:

```
str = time2str(time,'12','hm','hms')
str =
1:57 PM
```

The 24-hour and navigational representations:

```
str = time2str(time,'24','hms','hms')
str =
13:56:44
str = time2str(time,'nav','hms','hms')
str =
1356'''
```

Navigational times are four digits; if seconds are included, they are rounded to the nearest 15 seconds, which are represented by tick marks (`0`=none, `15`=', `30`=", `45`="').

Consider the *hms* format time 1356.4456 for rounding purposes:

```
str = time2str(1356.4456,'12','hms','hms',-2)  % hundredths
str =
1:56:44.56 PM
str = time2str(1356.4456,'12','hms','hms',-1)  % tenths
str =
1:56:44.6 PM
```

See Also

`hr2hms` `hr2sec`	Other direct time conversion functions
`timedim`	Convert times between different units

timedim

Purpose

Convert times between different units

Syntax

```
timeout = timedim(anglin,from,to)
```

Description

`timeout = timedim(timein,from,to)` returns the value of the input time `timein`, which is in units specified by the valid time units string *from*, in the desired units given by the valid time units string *to*. Valid time units strings are:

```
'hours' or 'hr'        for decimal hours
'seconds' or 'sec'     for seconds
'hms'                  for hours-minutes-seconds
'hm'                   for hours-minutes
```

Examples

Convert from hours to seconds:

```
timedim(2.56,'hours','seconds')
ans =
        9216
```

What is the difference between *hms* and *hm*? (best displayed in *bank format*)

```
format bank
timedim(2.56,'hours','hms')
ans =
        233.36

timedim(2.56,'hours','hm')
ans =
        234.00
```

The *hm* answer is the *hms* answer correctly rounded to whole minutes (i.e., rounded based on 60 seconds per minute, not 100).

See Also

`angledim`	Convert angle units
`distdim`	Convert distance units
`hr2hms` `hr2sec`	Other direct time conversion functions
`time2str`	Convert time to selected display string

Purpose

Determine time zone based on longitude

Syntax

```
[zd,zltr,zone] = timezone(long)
[zd,zltr,zone] = timezone(long,units)
```

Description

`[zd,zltr,zone] = timezone(long)` returns an integer zone description `zd`, an alphabetical string zone indicator `zltr`, and a string `zone`, with the complete zone description and alphabetical zone indicator corresponding to the input longitude `long`.

`[zd,zltr,zone] = timezone(long,units)` specifies the angular units with a standard angle *units* string. The default value is `'degrees'`.

Examples

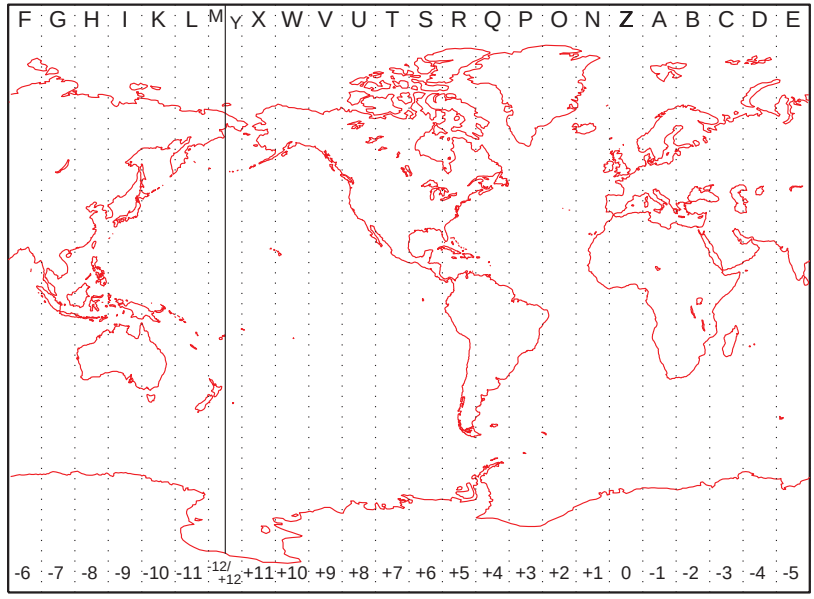
Given that it is locally 1330 (1:30 p.m.) at a longitude of 75°W, determine GMT:

```
[zd,zltr,zone] = timezone(-75,'degrees')
zd =
     5
zltr =
R
zone =
+5 R
```

Greenwich Mean Time (GMT) is 1330 plus five hours, or 1830 (6:30 p.m.).

Background

Time is determined by the position of the Sun relative to the Prime Meridian, the zero longitude line running through Greenwich, England. When this meridian lies directly below the Sun, it is noon, GMT. For local times elsewhere, the Earth is divided into 15° longitude bands, each centered on a central meridian. When a central meridian lies directly below the Sun, Local Mean Time (LMT) in that zone is noon. The zone description is an integer, which, when added to LMT, gives GMT. For notational convenience, each zone is also given an alphabetical indicator. The indicator at Greenwich is *Z*, so GMT is often called *ZULU Time*.



Note that there are actually 25 time zones, because the zone centered on the International Date Line (180° E/W) is split into two: “+12 Y” and “–12 M”.

Limitations

National and local governments set their own time zone boundaries for political or geographic convenience. The `timezone` command does not account for statutory deviations from the meridian-based system.

Purpose

Project Tissot indicatrices onto map axes

Syntax

```
h = tissot
h = tissot(spec)
h = tissot(spec,linestyle)
h = tissot(spec,PropertyName,PropertyValue,...)
h = tissot(spec,linestyle,PropertyName,PropertyValue,...)
h = tissot(linestyle)
h = tissot(PropertyName,PropertyValue,...)
h = tissot(linestyle,PropertyName,PropertyValue,...)
```

Background

Tissot indicatrices are plotting symbols, which are useful for understanding the various distortions of a given map projection. The indicatrices are circles of identical true radius on the Earth's surface. When plotted on a map projection, they indicate whether the projection has certain features. If the plotted indicatrices all enclose the same area, the projection is equal area (for example, a Sinusoidal projection would have this feature). If they all remain circular, then conformality is indicated (a Mercator projection has this property). Distortions in meridianal or parallel distance are exhibited by flattened or stretched indicatrices. Many projections will show very even, circular indicatrices in some regions, often near the center, and wildly distorted indicatrices in others, such as near the edges. The Tissot diagram is therefore very useful in analyzing the appropriateness of a projection to a given purpose or region. Chapter 2, "Projections Reference" of this guide includes Tissot diagrams for every projection on a global scale.

Description

The general layout of the Tissot diagram is defined by the specification vector spec.

```
spec = [Radius]
spec = [Latint,Longint]
spec = [Latint,Longint,Radius]
spec = [Latint,Longint,Radius,Points]
```

`Radius` is the small circle radius of each indicatrix circle. If entered, it should be in the same units as the map axes `Geoid`. The default radius is 1/10th the radius of the sphere.

`Latint` is the latitude interval between indicatrix circle centers. If entered it should be in the map axes `AngleUnits`. The default value is one circle every 30° of latitude (i.e. 0°,+/–30°, etc.).

`Longint` is the longitude interval between indicatrix circle centers. If entered it should be in the map axes `AngleUnits`. The default value is one circle every 30° of latitude (i.e. 0°,+/–30°, etc.).

`Points` is the number of plotting points per circle. The default is 100 points.

`h = tissot` plots the default Tissot diagram, as described above on the current map axes and returns handles for the displayed indicatrices.

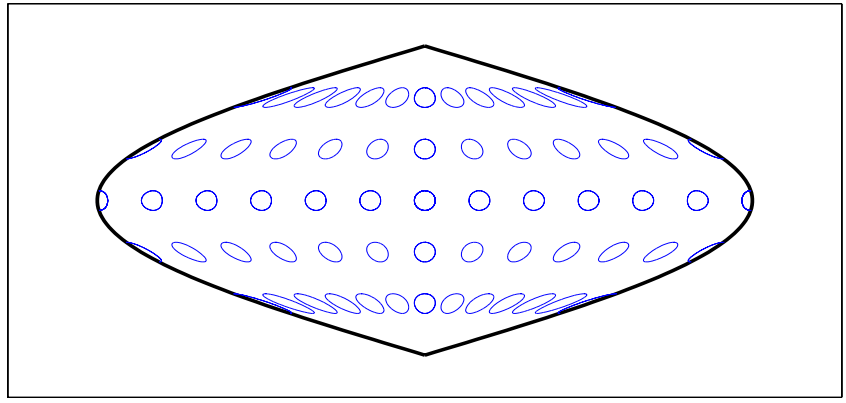
`h = tissot(spec)` allows you to specify plotting parameters of the displayed Tissot diagram as described above.

`h = tissot(spec,`*`linestyle`*`)` specifies any *linestyle* string recognized by the standard MATLAB function `line` to set the line style of the Tissot indicatrices.

`h = tissot(spec,`*`PropertyName`*`,PropertyValue,...)` allows the specification of any property and value recognized by the `line` command.

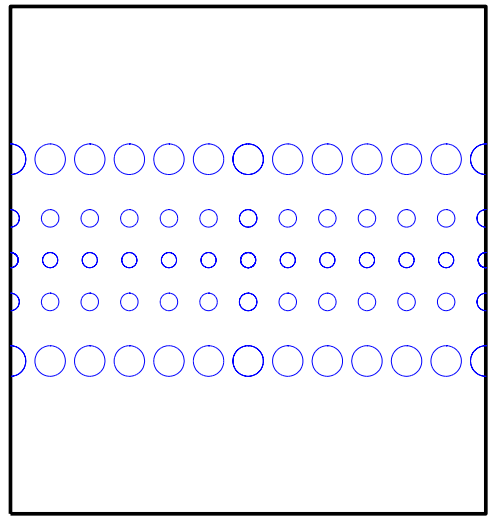
Examples

```
axesm sinusoid; framem
tissot
```



The Sinusoidal projection is equal area.

```
setm(gca,'MapProjection','Mercator')
```



The Mercator projection is conformal.

See Also

`mdistort`	Display contours of constant distortion on a map
`distortcalc`	Calculate distortion parameters for a map projection
`projections`	See Chapter 2, “Projections Reference”

track

Purpose

Connect navigational waypoints with track segments

Syntax

```
[lattrk,lontrk] = track(waypts)
[lattrk,lontrk] = track(waypts,units)
[lattrk,lontrk] = track(lat,lon)
[lattrk,lontrk] = track(lat,lon,units)
[lattrk,lontrk] = track(lat,lon,geoid)
[lattrk,lontrk] = track(lat,lon,geoid,units)
[lattrk,lontrk] = track(lat,lon,geoid,units,npts)
[lattrk,lontrk] = track(method,lat,...)
trkpts = track(lat,lon...)
```

Description

`[lattrk,lontrk] = track(waypts)` returns points in `lattrk` and `lontrk` along a track between the waypoints provided in navigational track format in the two-column matrix `waypts`. The outputs are column vectors in which successive segments are delineated with `NaN`s.

`[lattrk,lontrk] = track(waypts,units)` specifies the units of the inputs and outputs, where *units* is any valid angle unit string. The default is `'degrees'`.

`[lattrk,lontrk] = track(lat,lon)` allows the user to input the waypoints in two vectors, `lat` and `lon`.

`[lattrk,lontrk] = track(lat,lon,geoid)` specifies the elliptical definition of the Earth with a two-element geoid model vector `geoid`. The default geoid is a spherical Earth, which is sufficient for most applications.

`[lattrk,lontrk] = track(lat,lon,geoid,units,npts)` establishes how many intermediate points are to be calculated for every track segment. By default, `npts` is 30.

`[lattrk,lontrk] = track(method,lat,...)` establishes which logic is to be used to determine the intermediate points along the track between waypoints. Because this is a navigationally motivated function, the default method is `'rh'`, which results in the a rhumb line logic. Great circle logic can be specified with `'gc'`.

`trkpts = track(lat,lon...)` compresses the output into one two-column matrix, `trkpts`, in which the first column represents latitudes and the second column, longitudes.

Examples

The `track` command is useful for generating data in order to display tracks. Lieutenant Sextant is the navigator of the USS Neversail. He has been charged with plotting a track to take Neversail from the Straits of Gibraltar to Port Said, Egypt, the northern end of the Suez Canal. He has picked appropriate waypoints and now would like to display the track for his captain's approval.

First, display a chart of the Mediterranean Sea:

```
load coast
axesm('mercator','MapLatLimit',[30 47],'MapLonLimit',[-10 37])
plotm(lat,long,'b')
```

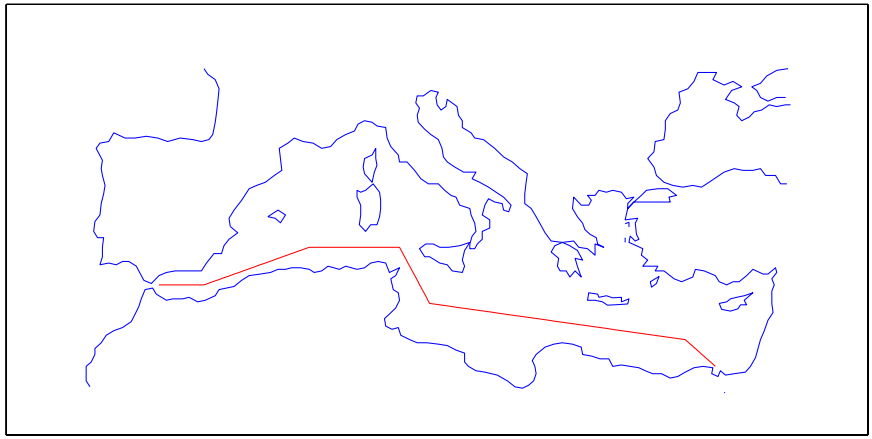
These are the waypoints Lt. Sextant has selected:

```
waypoints = [36,-5; 36,-2; 38,5; 38,11; 35,13; 33,30; 31.5,32]
waypoints =
   36.0000   -5.0000
   36.0000   -2.0000
   38.0000    5.0000
   38.0000   11.0000
   35.0000   13.0000
   33.0000   30.0000
   31.5000   32.0000
```

Now display the track:

```
[lttrk,lntrk] = track('rh',waypoints,'degrees');
plotm(lttrk,lntrk,'r')
```

With a display this clear, the captain gladly approves the plan.



See Also

`dreckon`	Dead reckon points for a track
`gcwaypts`	Find waypoints along a great circle
`legs`	Courses and distances between waypoints
`navfix`	Mercator-based navigational fixing

Purpose

Compute great circle or rhumb line track defined by point, azimuth, and range

Syntax

```
[lattrk,lontrk] = track1(lat,lon,az)
[lattrk,lontrk] = track1(track,lat,lon,az...)
[lattrk,lontrk] = track1(track,lat,lon,az,units)
[lattrk,lontrk] = track1(track,lat,lon,az,rng)
[lattrk,lontrk] = track1(track,lat,lon,az,rng,units)
[lattrk,lontrk] = track1(track,lat,lon,az,rng,geoid)
[lattrk,lontrk] = track1(track,lat,lon,az,rng,geoid,units)
[lattrk,lontrk] = track1(track,lat,lon,az,rng,geoid,units,npts)
pts = track1(lat,lon,az)
```

Background

A path along the surface of the Earth connecting two points is a *track*. Two types of track lines are of interest geographically, great circles and rhumb lines. Great circles represent the shortest possible path between two points. Rhumb lines are paths with constant angular headings. They are not, in general, the shortest path between two points.

Full great circles bisect the Earth; the *ends* of the track meet to form a complete circle. Rhumb lines with true east or west azimuths are parallels; the “ends” also meet to form a complete circle. All other rhumb lines terminate at the poles; their “ends” do not meet.

Description

`[lattrk,lontrk] = track1(lat,lon,az)` returns, in `lattrk` and `lontrk`, points along a complete (great circle) track passing through the point specified by `lat` and `lon` with an initial azimuth at that point of `az`. When the inputs are column vectors, the successive tracks are stored in separate columns of `lattrk` and `lontrk`.

`[lattrk,lontrk] = track1(track,lat,lon,az)` allows the specification of the track logic to be employed. A string *track* of `'gc'` is the default, resulting in a great circle track. A *track* of `'rh'` results in a complete rhumb line track.

`[lattrk,lontrk] = track1(track,lat,lon,az,units)` specifies the units of the inputs and outputs, where *units* is any valid angle unit string. The default is `'degrees'`.

`[lattrk,lontrk] = track1(`*`track`*`,lat,lon,az,rng)` specifies the range of the track. `rng` is a one- or two-column matrix. If `rng` has one column, the track will extend from the point `(lat,lon)` at an azimuth of `az` for a distance `rng` if `rng` is positive, or at an azimuth `az+180°` (or its angular equivalent) for a distance of `abs(rng)` if `rng` is negative. If `rng` has two columns, the end points are defined as above. In this case, the segment extends from the point associated with the first column of `rng` to the point associated with the second column. `rng` is in *`units`* (unless a `geoid` is input). When no `rng` is provided, or `rng` is empty, a *complete* track is returned.

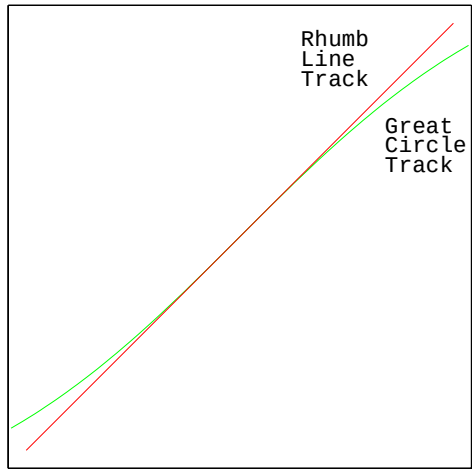
`[lattrk,lontrk] = track1(track,lat,lon,az,rng,geoid,`*`units`*`)` specifies the elliptical definition of the Earth with a two-element geoid model vector `geoid`. The default geoid is a spherical Earth, which is sufficient for most applications. If used, the units of the semimajor axis of the geoid vector define the units for the `rng` input, overriding *`units`* for this purpose.

`[lattrk,lontrk] = track1(`*`track`*`,lat,lon,az,rng,geoid,`*`units`*`,npts)` specifies the number of points, npts, per output track. `npts` is 100 by default.

`pts = track1(lat,lon,az,...)` combines the outputs into a single, two-column matrix `pts`.

Examples

```
axesm('mercator','MapLatLimit',[-60 60],'MapLonLimit',[-60 60])
[lattrkgc,lontrkgc] = track1(0,0,45,[-55 55]);
plotm(lattrkgc,lontrkgc,'g')
[lattrkrh,lontrkrh] = track1('rh',0,0,45,[-55 55]);
plotm(lattrkrh,lontrkrh,'r')
```



See Also

azimuth	Azimuth between two points on the globe
distance	Distance between two points on the globe
reckon	New point with an azimuth and distance
scircle1 scircle2	Small circle coordinates
track	Connect waypoints with track segments
track2	Great circle or rhumb line defined by two points

track2

Purpose

Compute great circle or rhumb line track defined by two points

Syntax

```
[lattrk,lontrk] = track2(lat1,lon1,lat2,lon2)
[lattrk,lontrk] = track2(track,lat1,lon1,lat2,lon2)
[lattrk,lontrk] = track2(track,lat1,lon1,lat2,lon2,units)
[lattrk,lontrk] = track2(track,lat1,lon1,lat2,lon2,geoid)
[lattrk,lontrk] = track2(track,lat1,lon1,lat2,lon2,geoid,units)
[lattrk,lontrk] = track2(lat1,lon1,lat2,lon2,geoid,units,npts)
pts = track2(lat1,lon1,lat2,lon2)
```

Background

A path along the surface of the Earth connecting two points is a *track*. Two types of track lines are of interest geographically, great circles and rhumb lines. Great circles represent the shortest possible path between two points. Rhumb lines are paths with constant angular headings. They are not, in general, the shortest path between two points.

Description

`[lattrk,lontrk] = track2(lat1,lon1,lat2,lon2)` returns, in `lattrk` and `lontrk`, points along a (great circle) track between the points specified by `lat1` with `lon1` and `lat2` and `lon2`. When the inputs are column vectors, the successive tracks are stored in separate columns of `lattrk` and `lontrk`.

`[lattrk,lontrk] = track2(track,lat1,lon1,lat2,lon2)` allows the specification of the track logic to be employed. A string *track* of `'gc'` is the default, resulting in a great circle track. A *track* of `'rh'` results in a rhumb line track.

`[lattrk,lontrk] = track2(track,lat1,lon1,lat2,lon2,units)` specifies the units of the inputs and outputs, where *units* is any valid angle unit string. The default is `'degrees'`.

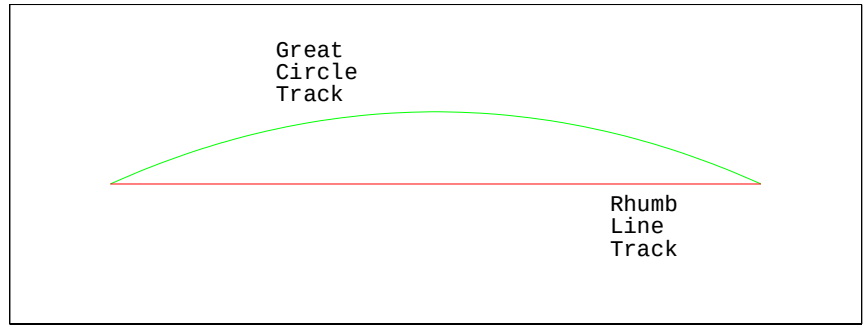
`[lattrk,lontrk] = track2(track,lat1,lon1,lat2,lon2,geoid,units)` specifies the elliptical definition of the Earth with a two-element geoid model vector `geoid`. The default geoid is a spherical Earth, which is sufficient for most applications.

`[lattrk,lontrk] = track2(lat1,lon1,lat2,lon2,geoid,units,npts)` specifies the number of points, `npts`, per output track. `npts` is 100 by default.

`pts = track2(lat1,lon1,lat2,lon2,...)` combines the outputs into a single, two-column matrix `pts`.

Example

```
axesm('mercator','MapLatLimit',[30 50],'MapLonLimit',[-40 40])
[lattrkgc,lontrkgc] = track2(40,-35,40,35);
[lattrkrh,lontrkrh] = track2('rh',40,-35,40,35);
plotm(lattrkgc,lontrkgc,'g')
plotm(lattrkrh,lontrkrh,'r')
```



See Also

azimuth	Azimuth between two points on the globe
distance	Distance between two points on the globe
reckon	New point with an azimuth and distance
scircle1 scircle2	Small circle coordinates
track	Connect waypoints with track segments
track1	Great circle or rhumb line from point and azimuth

trackg

Purpose

Display great circle or rhumb line track defined via mouse input

Syntax

```
h = trackg(ntrax)
h = trackg(ntrax,npts)
h = trackg(ntrax,linestyle)
h = trackg(ntrax,npts,linestyle)
h = trackg(ntrax,PropertyName,PropertyValue,...)
h = trackg(ntrax,npts,PropertyName,PropertyValue,...)
[lat,lon] = trackg(ntrax,npts,PropertyName,PropertyValue,...)
h = trackg(track,ntrax,...)
```

Description

This function is used to define great circles or rhumb lines for display using mouse clicks. For each track, two clicks are required, one for each endpoint of the desired track segment.

`h = trackg(ntrax)` brings forward the current map axes and waits for the user to make (2 x `ntrax`) mouse clicks. The output `h` is a vector of handles for the `ntrax` track segments, which are then displayed.

`h = trackg(ntrax,npts)` specifies the number of plotting points to be used for each track segment. `npts` is 100 by default.

`h = trackg(ntrax,linestyle)` specifies the line style for the displayed track segments, where *linestyle* is any line style string recognized by the standard MATLAB function `line`.

`h = trackg(ntrax,PropertyName,PropertyValue,...)` allows property/value pairs to be set, where *PropertyName* and `PropertyValue` are recognized by the `line` command.

`[lat,lon] = trackg(ntrax,npts,...)` returns the coordinates of the plotted points rather than the handles of the track segments. Successive segments are stored in separate columns of `lat` and `lon`.

`h = trackg(track,ntrax,...)` specifies the logic with which tracks are calculated. If the string *track* is `'gc'` (the default), great circle path is used. If *track* is `'rh'`, rhumb line logic is used.

See Also

`track1`	Great circle or rhumb line from point, azimuth, and range
`track2`	Great circle or rhumb line from two points

Purpose

Trim graphic objects to the map frame

Syntax

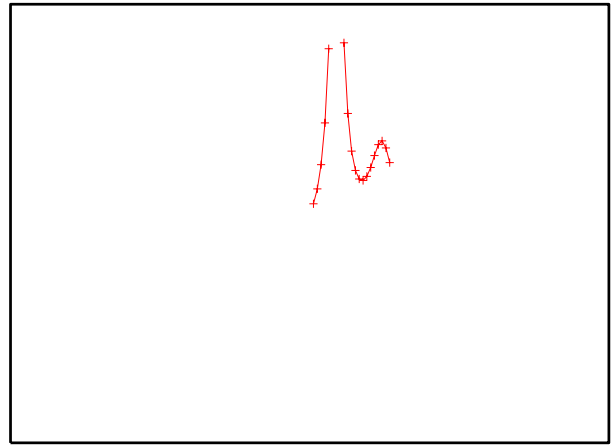
```
trimcart(h)
```

Description

`trimcart(h)` clips the graphic objects to the map frame. `h` can be a handle or a vector of handles to graphics objects. `h` can also be any object name recognized by `handlem`. `trimcart` clips lines, surfaces, and text objects.

Examples

```
str = unitstr('sm','distances')
axesm('miller','geoid',[25 0])
framem
h = plot(humps,'r+-');
trimcart(h)
```



Limitations

`trimcart` does not trim patch objects.

See Also

`handlem`	Graphics handle for identified objects
`makemapped`	Make an object mapped

unitstr

Purpose

Test for valid unit strings or abbreviations

Syntax

```
unitstr
str = unitstr(str0,measstr)
```

Description

`unitstr` lists all valid unit strings and all abbreviations that are not simply truncations of the original strings (e.g., `'km'` for `'kilometers'`).

`str = unitstr(str0,measstr)` returns the valid, standard string *str* corresponding to the recognized abbreviation *str0*. The type of string sought is specified by *measstr*, which can be `'distances'`, `'angles'`, or `'time'`.

Examples

This command recognizes and standardizes certain abbreviations:

```
str = unitstr('sm','distances')
str =
statutemiles
```

And any unique truncation:

```
str = unitstr('hou','time')
str =
hours
```

See Also

`angledim`	Angle unit conversion
`distdim`	Distance unit conversion
`timedim`	Time unit conversion

Purpose

Return data from the USAHI atlas data file

Syntax

```
usahi
s = usahi(request)
```

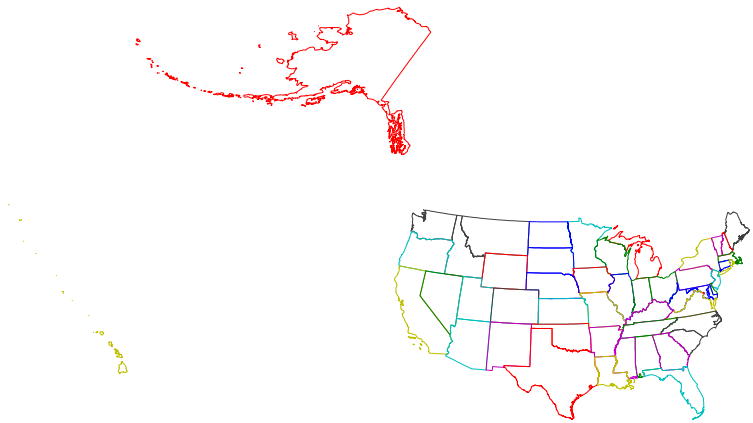
Description

`usahi` types a list of the atlas data variables in the USAHI MAT-file to the screen.

`s = usahi(request)` returns the requested variable. Valid requests are `'stateline'`,`'statepatch'`, and `'statetext'`. This command can be used as an argument to other commands, such as `displaym(usahi('statepatch'))`.

Examples

```
axesm('bonne','origin',-100)
displaym(usahi('stateline'))
```



See Also

`coast`	Coastline data
`worldlo`	World atlas data
`usalo`	Low-resolution atlas data for the United States
`usahi.mat`	MAT-file containing high-resolution atlas data for the United States

usalo

Purpose

Return data from the USALO atlas data file

Syntax

```
usalo
s = usahlo(request)
```

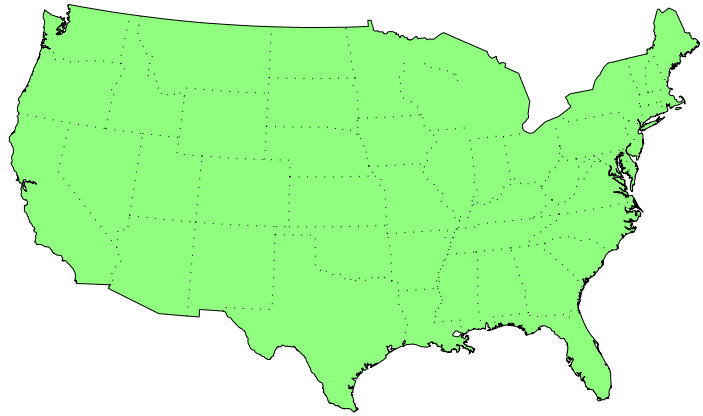
Description

`usalo` types a list of requests for the usalo MAT-file to the screen.

`s = usalo(request)` returns the requested data. Valid requests for two-column vectors are `'conusvec'`, `'statebvec'` and `'gtlakevec'`. Valid requests for geographic data structures are `'conus'`, `'state'`, `'stateborder'` and `'greatlakes'`. This command can be used as an argument to other commands, such as `displaym(usalo('conus'))`.

Examples

```
axesm('bonne','origin',-100)
displaym(usalo('conus'))
displaym(usalo('stateborder'))
```



See Also

`coast`	Coastline data
`worldlo`	World atlas data
`usahi`	High-resolution atlas data for the United States
`usalo.mat`	MAT-file containing low-resolution atlas data for the United States

usamap

Purpose

Create a map of the United States of America

Syntax

```
usamap
usamap all
usamap allequal
usamap conus
usamap state
usamap stateonly
usamap('state','type')
usamap(latlim,lonlim)
usamap(latlim,lonlim,'type')
usamap(map,maplegend)
usamap(map,maplegend,'type')
h = usamap(...)
```

Description

`usamap` creates a map of all or some of the United States of America. The state or states are selected interactively from a list. A map axes and map is created in the current axes. The axis limits are set tight around the map frame.

`h = usamap('all')` or `usamap all` creates a standard map of the U. S. The conterminous states, Alaska and Hawaii are each displayed as insets in different axes using projection parameters suggested by the U. S. Geological Survey. The handles for the three map axes are returned in `h`. `h(1)` contains the conterminous states, `h(2)` Alaska, and `h(3)` Hawaii.

`usamap allequal` creates the map with Alaska and Hawaii at the same scale as the conterminous states.

`usamap conus` maps only the conterminous states.

`usamap` *`state`* maps the requested state. Example: `usamap vermont`. *`state`* may also be a padded string matrix or a cell array of strings containing multiple state names.

`usamap` *`stateonly`* maps only that state. Example: `usamap vermontonly`. If any of the country names in a *`state`* string matrix or cell array of strings end with 'only', only the requested countries are displayed.

`usamap(`*`state`*`,`*`type`*`)` controls the atlas data displayed. Type '`line`' or '`patch`' creates a map with atlas data of those types and text annotation of the state

names. Type 'lineonly' or 'patchonly' suppresses text annotation. Type 'none' suppresses all atlas data. If omitted, type 'patch' is assumed.

`usamap(latlim,lonlim)` creates a map of the states covering the provided latitude and longitude limits. The limits are two-element vectors in units of degrees.

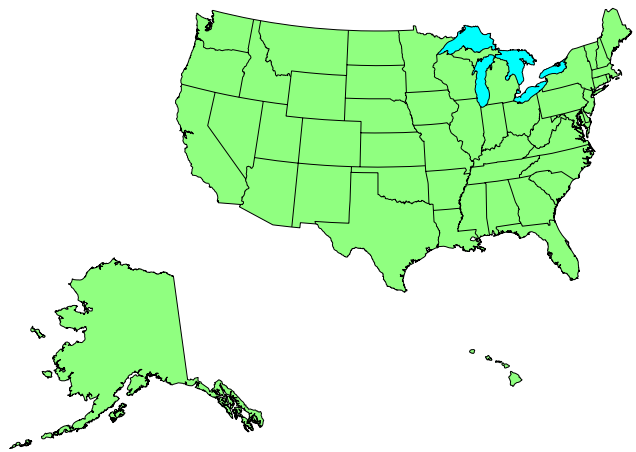
`usamap(latlim,lonlim,`*`type`*`)` is also a valid calling form.

`usamap(lmap,maplegend)` and `usamap(lmap,maplegend,'type')` use the supplied regular matrix map to define the geographic limits. The matrix map is displayed using `MESHM` unless type is `'none'`, `'lineonly'`, `'patch'` or `'patchonly'`. Use type `'meshonly'` to display only the matrix map

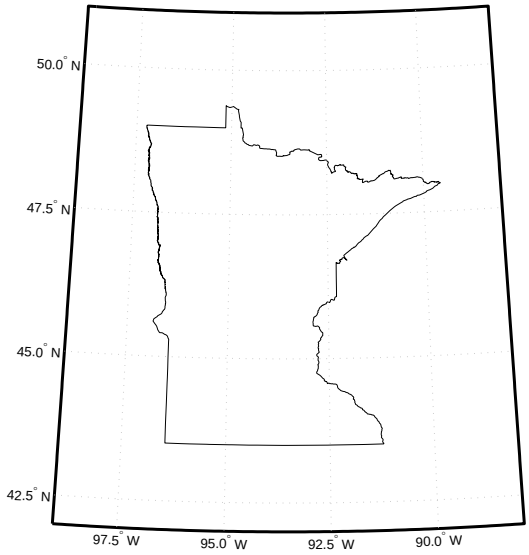
`h = usamap(...)` returns the handle or handles of the axes containing the map.

Examples

```
usamap allequal
```



```
usamap('minnesotaonly','line')
```



Remarks

usamap uses `tightmap` to set the axis limits tight around the map. If you change the projection, or just want more white space around the map frame, use `axis auto`.

`axes(h(n))`, where `n = 1,2` or `3`, makes the desired axes current.

`set(h,'Visible','on')` makes the axes visible.

`set(h,'ButtonDownFcn','selectmoveresize')` allows interactive repositioning of the axes. `set(h,'ButtonDownFcn','uimaptbx')` restores the Mapping Toolbox interfaces.

`axesscale(h(1))` resizes the axes containing Alaska and Hawaii to the same scale as the conterminous states.

See Also

`worldmap`	Maps a country or region using `worldlo` atlas data
`usalo`	Returns data from the `usalo` atlas data file
`selectmoveresize`	Interactively select, move, resize, or copy objects
`axesscale`	Resize axes for equivalent scale
`paperscale`	Figure paper size for a given map scale

utmzone

Purpose

Define Universal Transverse Mercator projection zone

Syntax

```
zone = utmzone

zone = utmzone(lat,long)
zone = utmzone(mat)

[latlim,lonlim] = utmzone(zone)
lim = utmzone(zone)

[zone,msg] = utmzone(...)
[latlim,lonlim,msg] = utmzone(...)
```

Background

The Universal Transverse Mercator (UTM) system of projections tiles the world into quadrangles called zones. This function can be used to identify which zone is used for a geographic area, and conversely, what geographic limits apply to a UTM zone.

Description

`zone = utmzone` selects a Universal Transverse Mercator (UTM) zone with a graphical user interface. The zone designation is returned as a string.

`zone = utmzone(lat,long)` returns the UTM zone containing the geographic coordinates. If `lat` and `long` are vectors, the zone containing the geographic mean of the data set is returned. The geographic coordinates must be in units of degrees.

`zone = utmzone(mat)`, where `mat` is or the form `[lat long]`.

`[latlim,lonlim] = utmzone(zone)`, where *zone* is a valid UTM zone designation, returns the geographic limits of the zone. Valid UTM zones designations are numbers, or numbers followed by a single letter. Example: `'31'` or `'31N'`. The returned limits are in units of degrees.

`lim = utmzone(zone)` returns the limits in a single vector output.

`[zone,msg] = utmzone(...)` and `[latlim,lonlim,msg] = utmzone(...)` returns a message if there is an error. `msg` is empty when there are no errors.

Examples

```
[latlim,lonlim] = utmzone('12F')
latlim =
   -56   -48
lonlim =
  -114  -108

utmzone(latlim,lonlim)
ans =
12F
```

Limitations

The UTM zone system is based on a regular division of the globe, with the exception of a few zones in northern Europe. utmzone does not account for these deviations.

See Also

utmzoneui	Universal Transverse Mercator projection zone graphical user interface
utmgeoid	Universal Transverse Mercator suggested ellipsoids

utmgeoid

Purpose

Recommend geoids for Universal Transverse Mercator projection zone

Syntax

```
geoid = utmgeoid
geoid = utmgeoid(zone)
[geoid,geoidstr] = utmgeoid(...)
```

Background

The Universal Transverse Mercator (UTM) system of projections tiles the world into quadrangles called zones. Each zone has different projection parameters and commonly used ellipsoidal models of the Earth. This function returns a list of ellipsoid models commonly used in a zone.

Description

`geoid = utmgeoid`, without any arguments, opens the `utmzoneui` interface for selecting a UTM zone. This zone is then used to return the recommended ellipsoid definition(s) for that particular zone.

`geoid = utmgeoid(`*`zone`*`)` uses the input *zone* to return the recommended ellipsoid definition(s).

`[geoid,geoidstr] = utmgeoid(...)` returns the geoid string used by the `almanac` function.

Examples

```
zone = utmzone(0,100) % degrees
zone =
47N

[geoid,names] = utmgeoid(zone)
geoid =
       6377.3     0.081473
       6377.4     0.081697
names =
everest
bessel
```

See Also

`utmzone`	Universal Transverse Mercator projection zones
`utmzoneui`	Universal Transverse Mercator projection zone graphical user interface

Purpose

Regular matrix map from vector data

Syntax

```
[map,maplegend] = vec2mtx(lat,lon,scale)
[map,maplegend] = vec2mtx(lat,lon,scale,latlim,lonlim)
[map,maplegend] = vec2mtx(lat,lon,map1,maplegend1)
[...] = vec2mtx(...,'filled')
```

Description

`[map,maplegend] = vec2mtx(lat,lon,scale)` creates a regular matrix map from vector data. The returned map has values of one corresponding to the vector data, and zeros otherwise. `lat` and `lon` are vectors of equal length containing geographic locations in units of degrees. The scale factor represents the number of matrix entries per single unit of latitude and longitude (e.g., 10 entries per degree, 100 entries per degree). The `scale` input must be scalar.

`[map,maplegend] = vec2mtx(lat,lon,scale,latlim,lonlim)` uses the two element vector latitude and longitude limits to define the extent of the map. If omitted, the limits are computed automatically.

`[map,maplegend] = vec2mtx(lat,lon,map1,maplegend1)` uses the provided `map` and `maplegend` to define the extent of the map. If omitted, the limits are computed automatically.

`[...] = vec2mtx(...,'filled')` also fills the area outside the border. The interior then has values of 0, the border 1 and the exterior 2. `lat` and `lon` should contain data that closes on itself

Examples

```
[lat,long] = extractm(worldlo('POpatch'),'Russia');
[map,maplegend] = vec2mtx(lat,long,2,'filled');
[latlim,lonlim] = limitm(map,maplegend);
worldmap(latlim,lonlim,'none'); framem off
meshm(map,maplegend)
colormap(flag(3))
```

Limitations

The `vec2mtx` function will not fill properly if the vector data extends beyond a pole.

See Also

`country2mtx`	Construct a matrix map for a country in the `worldlo` database
`ltln2val`	Returns map code value associated with positions
`imbedm`	Encodes data points into a regular matrix map
`encodem`	Fills in indexed maps with specified seeds
`interpm`	Interpolates vector data to a specified data separation

Purpose

Transform vector azimuths to a projection space angle

Syntax

```
th = vfwdtran(lat,lon,az)
th = vfwdtran(mstruct,lat,lon,az)
[th,len] = vfwdtran(...)
```

Background

The direction of north is easy to define on the three-dimensional sphere, but more difficult on a two-dimensional map. For cylindrical projections in the normal aspect, north is always in the positive y-direction. For conic projections, north may be to the left or right of the y-axis. This function transforms any azimuth angle on the sphere to the corresponding angle in the projected paper coordinates.

Description

`th = vfwdtran(lat,lon,az)` transforms the azimuth angle at specified latitude and longitude points on the sphere into the projection space. The map projection currently displayed is used to define the projection space. The input angles must be in the same units as specified by the current map projection. The inputs can be scalars or matrices of the equal size. The angle in the projection space is defined positive counter-clockwise from the x-axis.

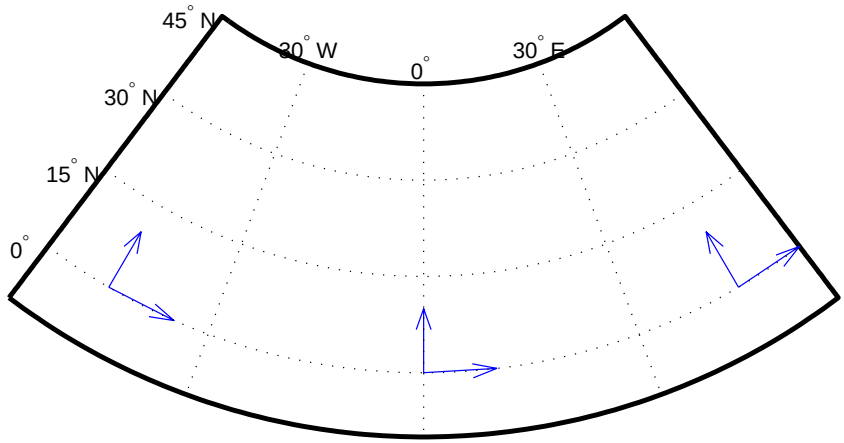
`th = vfwdtran(mstruct,lat,lon,az)` uses the map projection defined by the input `mstruct` to compute the map projection.

`[th,len] = vfwdtran(...)` also returns the vector length in the projected coordinate system. A value of 1 indicates no scale distortion.

Examples

Sample calculations:

```
axesm('eqdconic','maplatlim',[-10 45],'maplonlim',[-55 55])
gridm; framem; mlabel; plabel
quiverm([0 0 0],[-45 0 45],[0 0 0],[10 10 10],0)
quiverm([0 0 0],[-45 0 45],[10 10 10],[0 0 0],0)
```




```
vfwdtran([0 0 0],[-45 0 45],[0 0 0])
ans =
       59.614           90       120.39

vfwdtran([0 0 0],[-45 0 45],[90 90 90])
ans =
      -30.385    0.0001931       30.386
```

Limitations

This transformation is limited to the region specified by the frame limits in the current map definition.

Remarks

The geographic azimuth angle is measured clockwise from north. The projection space angle is measured counter-clockwise from the x axis.

This function uses a finite difference technique. The geographic coordinates are perturbed slightly in different directions and projected. A small amount of error is introduced by numerical computation of derivatives and the variation of map distortion parameters.

See Also

`vinvtran`	Transforms azimuths from a projection space angle
`mfwdtran`	Process the map forward transformations
`minvtran`	Process the map inverse transformations
`defaultm`	Initializes the default map data structure

vinvtran

Purpose

Transform azimuths from a projection space angle

Syntax

```
az = vinvtran(x,y,th)
az = vinvtran(struct,x,y,th)
[az,len] = vinvtran(...)
```

Background

While vectors along the y-axis always point to north in a cylindrical projection in the normal aspect, they may point east or west of north on conics, azimuthals, and other projections. This function computes the geographic azimuth for angles in the projected space

Description

`az = vinvtran(x,y,th)` transforms an angle in the projection space at the point specified by x and y into an azimuth angle in Greenwich coordinates. The map projection currently displayed is used to define the projection space. The input angles must be in the same units as specified by the current map projection. The inputs can be scalars or matrices of the equal size. The angle in the projection space angle `th` is defined positive counter-clockwise from the x axis.

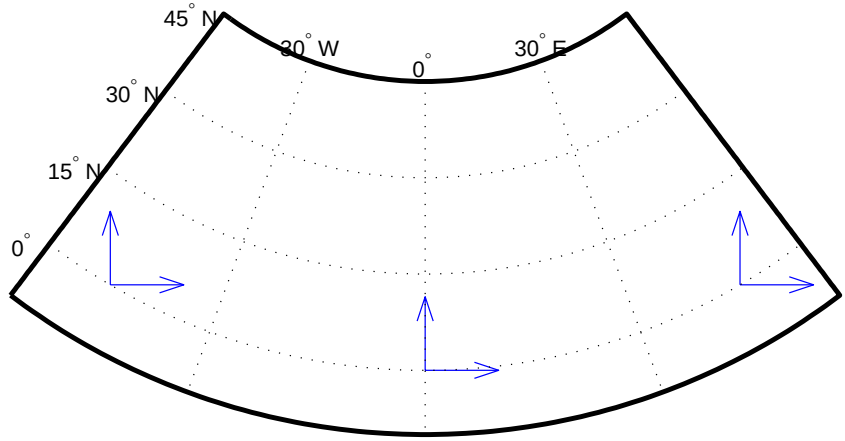
`az = vinvtran(mstruct,x,y,th)` uses the map projection defined by the input struct to compute the map projection.

`[az,len] = vfwdtran(...)` also returns the vector length in the Greenwich coordinate system. A value of 1 indicates no scale distortion for that angle.

Examples

Sample calculations:

```
axesm('eqdconic','maplatlim',[-10 45],'maplonlim',[-55 55])
gridm; framem; mlabel; plabel

[x,y] = mfwdtran([0 0 0],[-45 0 45]);
quiver(x,y,[ .2 .2 .2],[0 0 0],0)
quiver(x,y,[0 0 0],[ .2 .2 .2],0)
```




```
vinvtran(x,y,[ 0 0 0])
ans =
       57.345       90.338       124.98

vinvtran(x,y,[ 90 90 90])
ans =
       331.99            0       28.008
```

Limitations

This transformation is limited to the region specified by the frame limits in the current map definition.

Remarks

The geographic azimuth angle is measured clockwise from north. The projection space angles is measured counter-clockwise from the x axis.

This function uses a finite difference technique. The geographic coordinates are perturbed slightly in different directions and projected. A small amount of error is introduced by numerical computation of derivatives and the variation of map distortion parameters.

See Also

`vfwdtran`	Transforms vector azimuths to a projection space angle
`mfwdtran`	Process the map forward transformations
`minvtran`	Process the map inverse transformations
`defaultm`	Initializes the default map data structure

Purpose

Wraps longitudes to values west of a meridian

Syntax

```
ang = westof(angin,meridian)
ang = westof(angin,meridian,units)
```

Description

`ang = westof(angin,meridian)` transforms input angles into equivalent angles west of the specified meridian.

`ang = westof(angin,meridian,units)` uses the units defined by the input string *units*. If omitted, default units of 'degrees' are assumed.

Examples

```
westof(20,0)
ans =
-340

westof(20,-360)
ans =
-700
```

See Also

eastof	Wraps longitudes to values east of a meridian
zero22pi	Truncates angles into the 0 deg to 360 deg range
npi2pi	Truncates angles into the –180 deg to 180 deg range
smoothlong	Removes discontinuities in longitude data
angledim	Converts angles from one unit system to another

worldlo

Purpose

Returns data from the `worldlo` atlas data file

Syntax

```
worldlo
s = worldlo(request)
```

Description

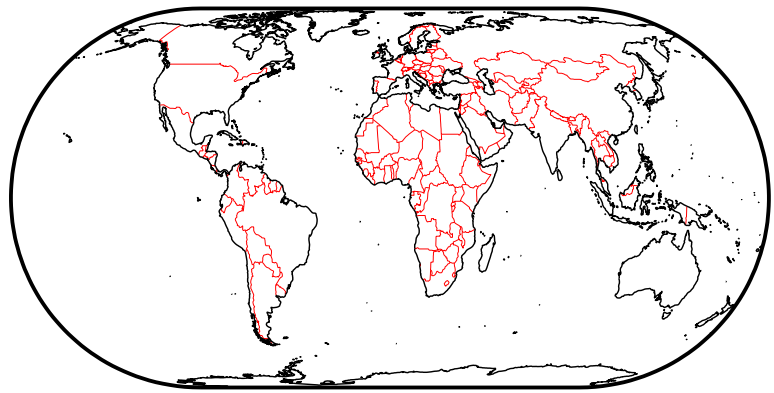
`worldlo` types a list of the atlas data variables in the `worldlo` MAT-file to the screen.

`s = worldlo(request)` returns the requested variable. Valid requests are `'POline'`, `'POpatch'`, `'POtext'`, `'PPpoint'`, `'PPtext'`, `'DNline'`, `'DNpatch'`, `'oceanmask'` and `'gazette'`. This command can be used as an argument to other commands, such as `displaym(worldlo('POline'))`.

Examples

Display the world line data without loading it into the workspace.

```
axesm eckert4
framem
displaym(worldlo('POline'))
```



See Also

`coast`	Coastline data
`usahi`	Returns data from the `usahi` atlas data file
`usalo`	Returns data from the `usalo` atlas data file
`worldlo.mat`	MAT-file containing low-resolution atlas data for the world
`oceanlo.mat`	MAT-file containing an ocean mask for the `worldlo` atlas data

worldmap

Purpose

Maps a country or region using the `worldlo` atlas data

Syntax

```
worldmap
worldmap region
worldmap regiononly
worldmap(region,type)
worldmap(latlim,lonlim)
worldmap(latlim,lonlim,'type')
worldmap(map,maplegend)
worldmap(map,maplegend,'type')
h = worldmap(...)
```

Description

`worldmap` maps a region or country using the `worldlo` atlas data. The region is selected interactively from a list. The map is created in the current axes.

`worldmap region` or `worldmap(region)` makes a map of the specified region. Recognized *region* strings are 'Africa', 'Antarctica', 'Asia', 'Europe', 'North America', 'North Pole', 'Pacific', 'South America', 'South Pole', 'World' and the names of the countries in the `worldlo` atlas data. *region* may also be a padded string matrix or a cell array of strings containing multiple country names.

`worldmap regiononly` adds only the specified country to the map. For example, `worldmap 'italyonly'`. If any of the country names in *region* string matrix or cell array of strings end with 'only', only the requested countries are displayed.

`worldmap(region,type)` controls the atlas data displayed. Type 'line' or 'patch' creates a map with atlas data of those types. If the size of the map permits, point and text annotation of countries and cities is included. Type 'lineonly' or 'patchonly' suppresses the point and text annotation. Type 'none' suppresses all atlas data. If omitted, type 'line' is assumed.

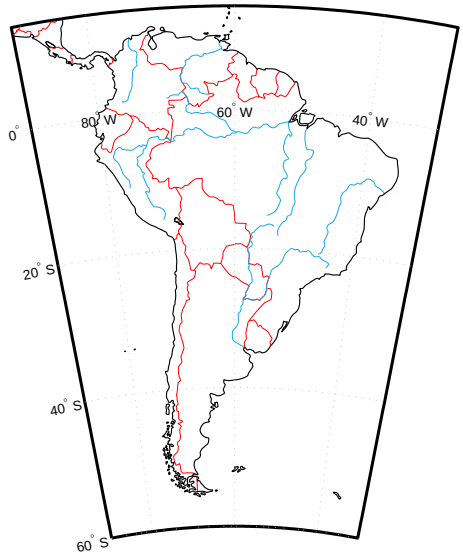
`worldmap(latlim,lonlim)` and `worldmap(latlim,lonlim,type)` use the supplied geographic limits to define the map. The geographic limits are two-element vectors of the form `[start end]`, in angular units of degrees.

`worldmap(map,maplegend)` and `worldmap(map,maplegend,'type')` use the supplied regular matrix map to define the geographic limits. The matrix map is displayed using `MESHM` unless type is `'none'`, `'lineonly'`, `'patch'` or `'patchonly'`. Use type `'meshonly'` to display only the matrix map.

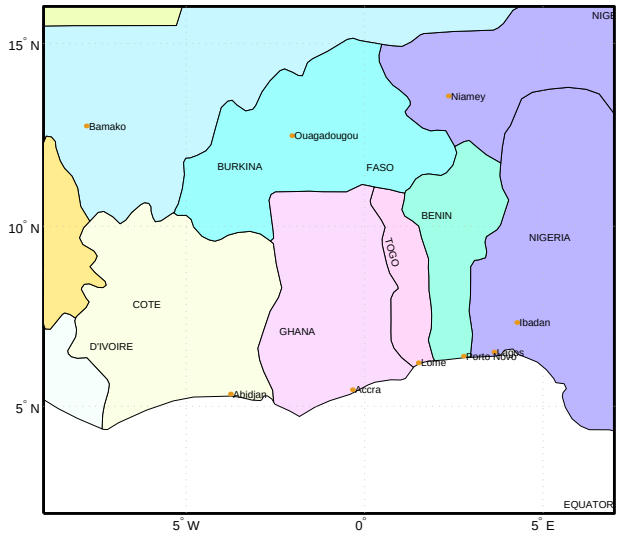
`h = worldmap(...)` returns the handle of the map axes.

Examples

```
worldmap 'south america'
```




```
worldmap([2 16],[-9 7],'patch')
```



Remarks

`worldmap` uses `tightmap` set the axis limits tight around the map. If you change the projection or just want more white space around the map frame, use `axis auto`.

Use `hidem(gca)` to remove the axes box. `set(gca,'Color','w')` makes the figure background white.

See Also

`usamap`	Creates a map of the United States of America
`gridm`	Toggle and control the display of the map grid
`mlabel`	Meridian labels projected onto a map axes
`plabel`	Parallel labels projected onto a map axes
`framem`	Toggle and control the display of the map frame
`worldlo`	Returns data from the `worldlo` atlas data file

Purpose

Convert x- and y-coordinates to row and column indices

Syntax

```
[rowgrat,colgrat] = yx2rc(ygrat,xgrat,y1,x1,yperrow,xpercol)
```

Description

`[rowgrat,colgrat] = yx2rc(ygrat,xgrat,y1,x1,yperrow,xpercol)` computes row and column indices from y- and x-coordinates. `ygrat` and `xgrat` are the Cartesian coordinates for a grid with the upper left corner at coordinates `x1` and `y1`. `yperrow` and `xpercol` are the y- and x-increment per row and column. The outputs `rowgrat` and `colgrat` correspond to matrix locations `ygrat` and `xgrat`.

Examples

```
[ygrat,xgrat] = meshgrat(1000:-500:0,-1000:500:0)
ygrat =
        1000        1000        1000
         500         500         500
           0           0           0
xgrat =
       -1000        -500           0
       -1000        -500           0
       -1000        -500           0

x1 = -1000; y1 = 1000; yperrow = -500; xpercol = 500;

[rowgrat,colgrat] = yx2rc(ygrat,xgrat,y1,x1,yperrow,xpercol)
rowgrat =
     1     1     1
     2     2     2
     3     3     3
colgrat =
     1     2     3
     1     2     3
     1     2     3
```

Remarks

This function is used for one step in the process of extracting projected matrix data from a file. The process generally consists of the following steps: 1) define the map projection as described in the data source documentation, 2) map desired geographic limits into projected coordinates using `mfwdtran`, 3) use `yx2rc` to determine associated row and column limits in the stored matrix, 4) extract sub-matrix using `readmtx`, 5) convert graticule of row and column indices to projected x and y coordinates using `rc2yx`, 6) convert x and y to geographic graticule using `minvtran`.

See Also

`rc2yx`	Row and column indices to x- and y-coordinates
`readmtx`	Read a matrix stored in a file

Purpose

Adjust the *z*-plane of specified graphics objects

Syntax

```
zdatam(hndl)
zdatam(object)
zdatam(hndl,zdata)
zdatam(object,zdata)
```

Description

This command adjusts the *z*-plane position of selected graphics objects. It accomplishes this by setting the objects' `ZData` properties to the appropriate values.

`zdatam(hndl)` sets the *z*-level of all objects specified by the vector of handles to 0.

`zdatam(object)` sets the *z*-level of all objects identified by the string *object* to 0. The string can be any string recognized by the `handlem` function.

`zdatam(hndl,zdata)` sets the *z*-level of all specified objects to the value of a scalar `zdata`, or sets each object at its own level if `zdata` is a vector the same size as `hndl`. When `hndl` is a scalar, `zdata` can also be a matrix with the same size as the object designated by `hndl`.

`zdatam(object,zdata)` sets the *z*-level of the designated object to a scalar `zdata`, or to match a `zdata` matrix the same size as the object.

See Also

`handlem`	Handles of graphics objects
`setm`	Set map properties

zero22pi

Purpose

Convert normalize angles to lie between 0 and 2π

Syntax

```
anglout = zero22pi(anglin)
anglout = zero22pi(anglin,units)
anglout = zero22pi(anglin,units,approach)
```

Description

`anglout = zero22pi(anglin)` *wraps* the input angle `anglin` to lie on the range 0 to 2π (e.g., 450° is renamed 90°).

`anglout = zero22pi(anglin,units)` specifies the angle units with any valid angle units string *units*. The default is `'degrees`.

`anglout = zero22pi(anglin,units,approach)` specifies the approach logic for this wrapping. The *approach* string `'exact'` calculates a mathematically precise wrap. `'inward'` and `'outward'` calculate more quickly by shifting the values by an *epsilon* either toward or away from the origin and performing a trigonometric wrap. The trigonometric wrap is inexact to allow for the fact that different computer math processors might give different (although trigonometrically identical) results (180° or –180°, for example). The offset prevents this.

Examples

```
zero22pi(567.5)
ans =
                    207.5

zero22pi(-567.5)
ans =
                    152.5
```

See Also

`npi2pi` Normalize angles to lie between –π and π

Purpose

Create a matrix maps of zeros

Syntax

```
map = zerom(latlim,lonlim,scale)
[map,maplegend] = zerom(latlim,lonlim,scale)
```

Description

`map = zerom(latlim,lonlim,scale)` returns a full regular matrix map consisting entirely of zeros. The two-element vectors `latlim` and `lonlim` define the latitude and longitude limits of the geographic region. They should be of the form `[south north]` and `[west east]`, respectively. The number of rows and columns per angle unit is set by the scalar `scale`.

`[map,maplegend] = zerom(latlim,lonlim,scale)` returns the three-element map legend vector for the returned `map`.

Examples

```
[map,maplegend] = zerom([46,51],[-79,-75],1)
map =
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
maplegend =
     1    51   -79
```

See Also

`limitm`	Matrix map limits
`map legend vector`	Data structure for describing regular matrix maps
`nanm`	Create a matrix map of `NaN`s
`onem`	Create a matrix map of ones
`sizem`	Rows and columns needed for map
`spzerom`	Create a sparse matrix map of zeros

2

Projections Reference

How to Use the Projections Reference Pages

The Projections Reference pages are organized in alphabetical order by the name of the map projection. The entries in this chapter contain the following:

Classification	Classifies the projection by the geometric or mathematical means of construction.
Syntax	Provides the name of the projection M-file used to specify a particular map projection.
Graticule	Describes the appearance of meridians, parallels, poles, and map symmetry.
Features	Describes the properties of the projection and identifies map distortion.
Parallels	Describes the standard parallels of projection.
Remarks	Describes the history of the projection and relationships to other projections.
Limitations	Describes any restrictions on using the projection.

Aitoff Projection

Classification Modified Azimuthal

Syntax `aitoff`

Graticule Meridians: Central meridian is a straight line half the length of the Equator. Other meridians are complex curves, equally spaced along the Equator, and concave towards the central meridian.

Parallels: Equator is straight. Other parallels are complex curves, equally spaced along the central meridian, and concave towards the nearest pole.

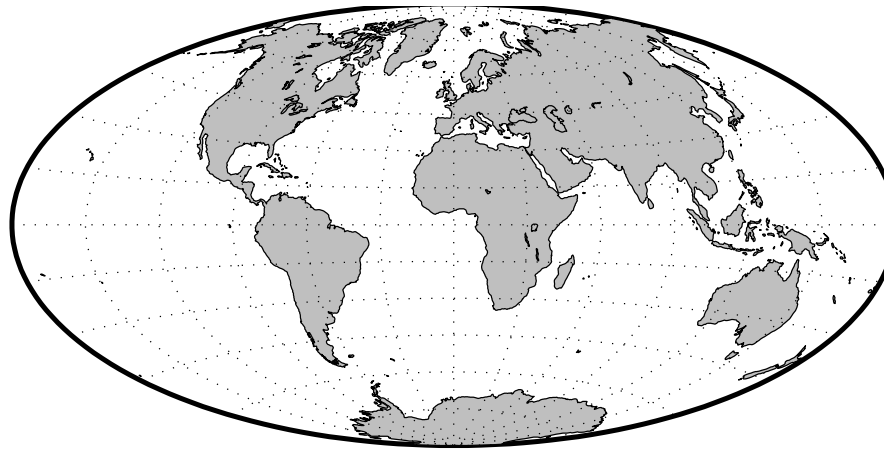
Poles: Points.

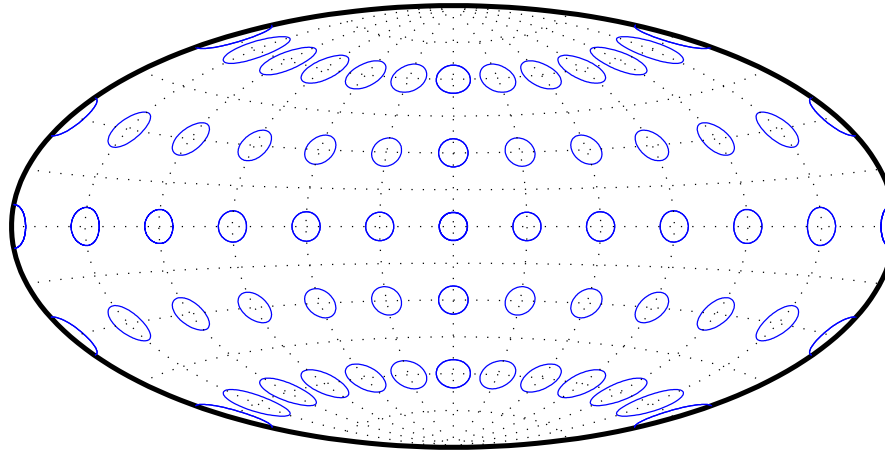
Symmetry: About the Equator and central meridian.

Features This projection is neither conformal nor equal area. The only point free of distortion is the center point. Distortion of shape and area are moderate throughout. This projection has less angular distortion on the outer meridians near the poles than pseudoazimuthal projections

Parallels There is no standard parallel for this projection.

Remarks This projection was created by David Aitoff in 1889. It is a modification of the Equidistant Azimuthal projection. The Aitoff projection inspired the similar Hammer projection, which is equal area.





Albers Equal-Area Conic Projection

Classification Conic

Syntax `eqaconic`

Graticule Meridians: Equally spaced straight lines converging to a common point, usually beyond the pole. The angles between the meridians are less than the true angles.

Parallels: Unequally spaced concentric circular arcs centered on the point of convergence. Spacing of parallels decreases away from the central latitudes.

Poles: Normally circular arcs, enclosing the same angle as the displayed parallels.

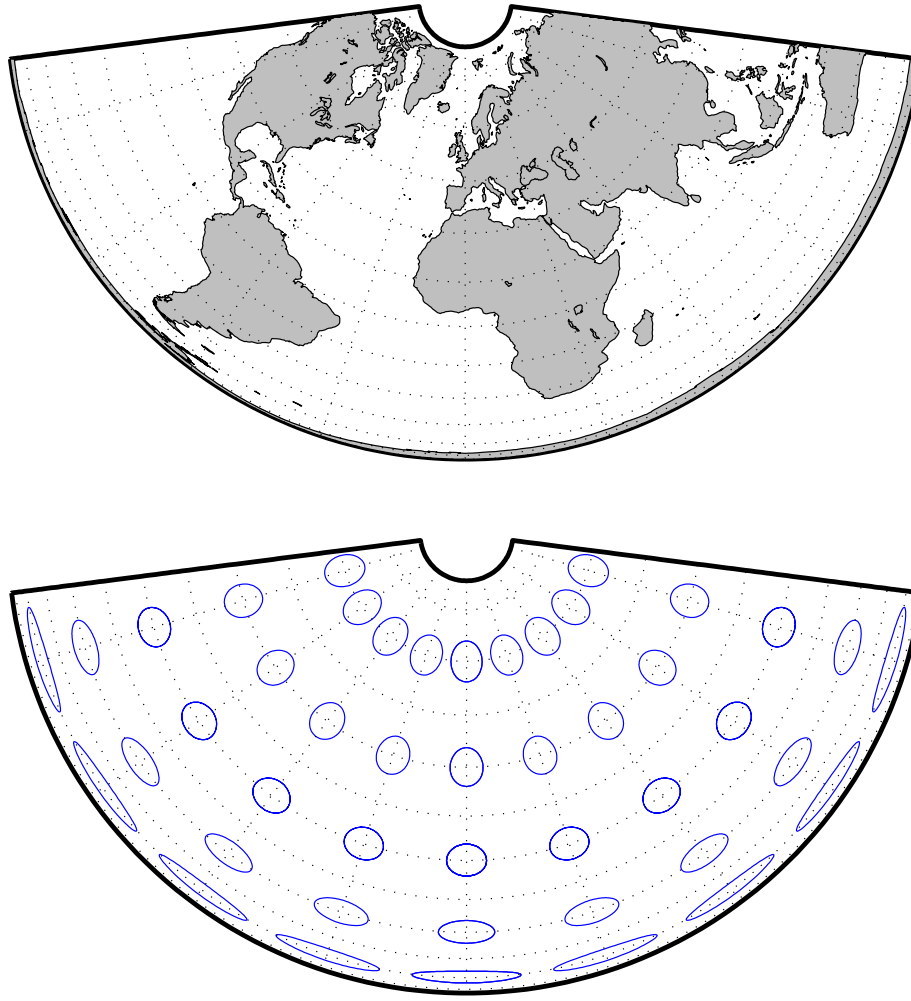
Symmetry: About any meridian.

Features This is an equal-area projection. Scale is true along the one or two selected standard parallels. Scale is constant along any parallel; the scale factor of a meridian at any given point is the reciprocal of that along the parallel to preserve equal-area. This projection is free of distortion along the standard parallels. Distortion is constant along any other parallel. This projection is neither conformal nor equidistant.

Parallels The cone of projection has interesting limiting forms. If a pole is selected as a single standard parallel, the cone is a plane and a Lambert Azimuthal Equal-Area projection results. If two parallels are chosen, not symmetric about the Equator, then a Lambert Equal-Area Conic projection results. If a pole is selected as one of the standard parallels, then the projected pole is a point, otherwise the projected pole is an arc. If the Equator is chosen as a single parallel, the cone becomes a cylinder and a Lambert Equal-Area Cylindrical projection is the result. Finally, if two parallels equidistant from the Equator are chosen as the standard parallels, a Behrmann or other equal-area cylindrical projection is the result. Suggested parallels for maps of the conterminous U.S. are [29.5 45.5]. The default parallels are [15 75].

Remarks This projection was presented by Heinrich Christian Albers in 1805.

Limitations Longitude data greater than 135° east or west of the central meridian is trimmed.



Apianus II Projection

Classification Pseudocylindrical

Syntax `apianus`

Graticule Meridians: Equally spaced elliptical curves converging at the poles.

Parallels: Equally spaced straight lines.

Poles: Points.

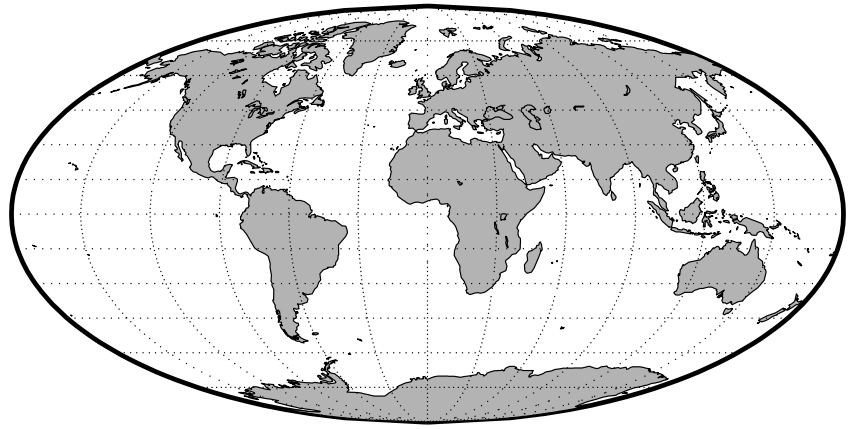
Symmetry: About the Equator and central meridian.

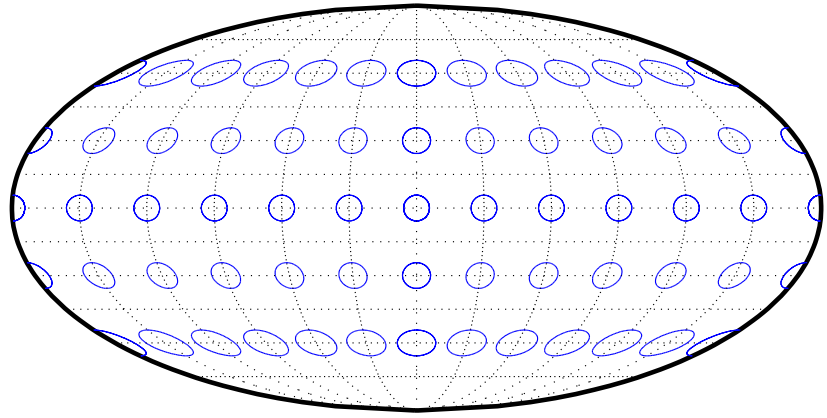
Features Scale is constant along any parallel or pair of parallels equidistant from the Equator, as well as along the central meridian. The Equator is free of angular distortion. This projection is not equal-area, equidistant, or conformal.

Parallels There is no standard parallel for this projection.

Remarks This projection was first described in 1524 by Peter Apian (or Bienewitz).

Limitations This projection is available for the spherical geoid only.





Balthasart Cylindrical Projection

Classification Cylindrical

Syntax `balthsrt`

Graticule Meridians: Equally spaced straight parallel lines.

Parallels: Unequally spaced straight parallel lines, perpendicular to the meridians. Spacing is closest near the poles.

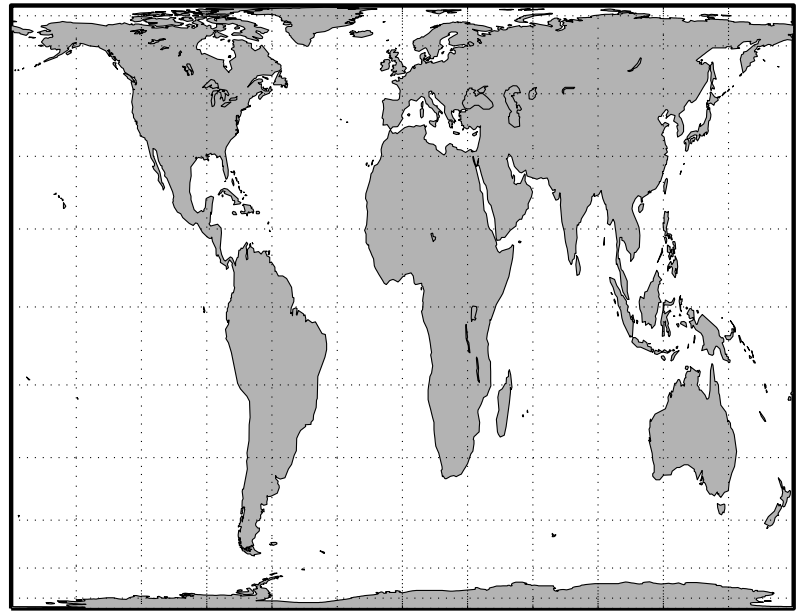
Poles: Straight lines equal in length to the Equator.

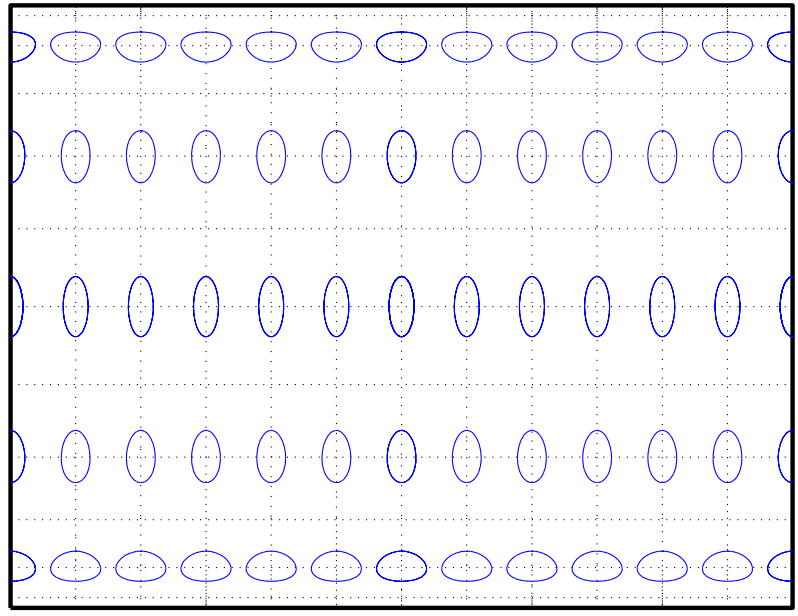
Symmetry: About any meridian or the Equator.

Features This is an orthographic projection onto a cylinder secant at the 50° parallels. It is equal-area, but distortion of shape increases with distance from the standard parallels. Scale is true along the standard parallels and constant between two parallels equidistant from the Equator. This projection is not equidistant.

Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, the standard parallel is by definition fixed at 50°.

Remarks The Balthasart Cylindrical projection was presented in 1935 and is a special form of the Equal-Area Cylindrical projection secant at 50°N and S.





Behrmann Cylindrical Projection

Classification Cylindrical

Syntax `behrmann`

Graticule Meridians: Equally spaced straight parallel lines 0.42 as long as the Equator.

Parallels: Unequally spaced straight parallel lines, perpendicular to the meridians. Spacing is closest near the poles.

Poles: Straight lines equal in length to the Equator.

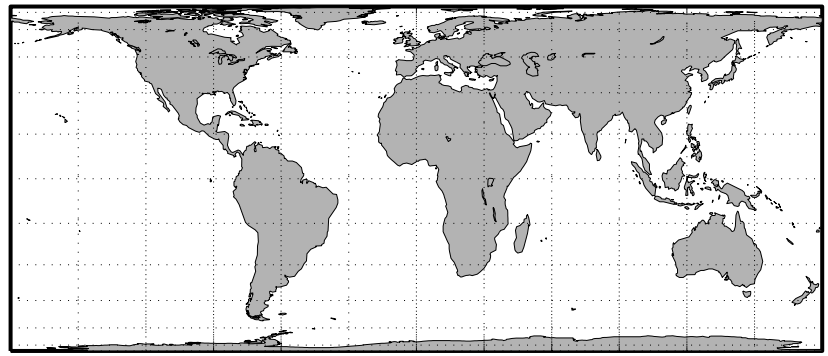
Symmetry: About any meridian or the Equator.

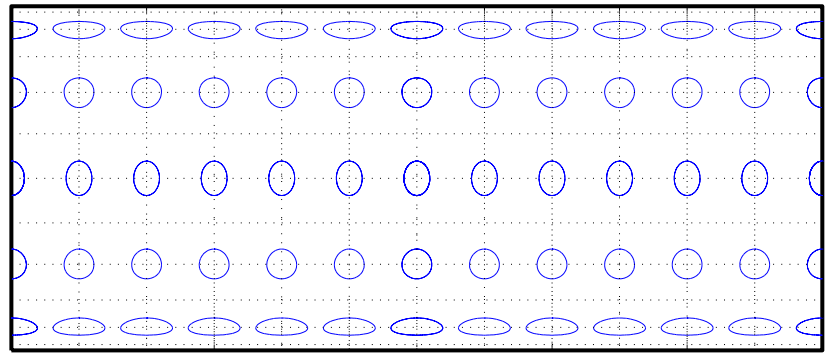
Features This is an orthographic projection onto a cylinder secant at the 30° parallels. It is equal-area, but distortion of shape increases with distance from the standard parallels. Scale is true along the standard parallels and constant between two parallels equidistant from the Equator. This projection is not equidistant.

Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, the standard parallel is by definition fixed at 30°.

Remarks This projection is named for Walter Behrmann, who presented it in 1910 and is a special form of the Equal-Area Cylindrical projection secant at 30°N and S.

Behrmann Cylindrical Projection





Bolshoi Sovietskii Atlas Mira Projection

Classification Cylindrical

Syntax `bsam`

Graticule Meridians: Equally spaced straight parallel lines.

Parallels: Unequally spaced straight parallel lines, perpendicular to the meridians. Spacing increases toward the poles.

Poles: Straight lines equal in length to the Equator.

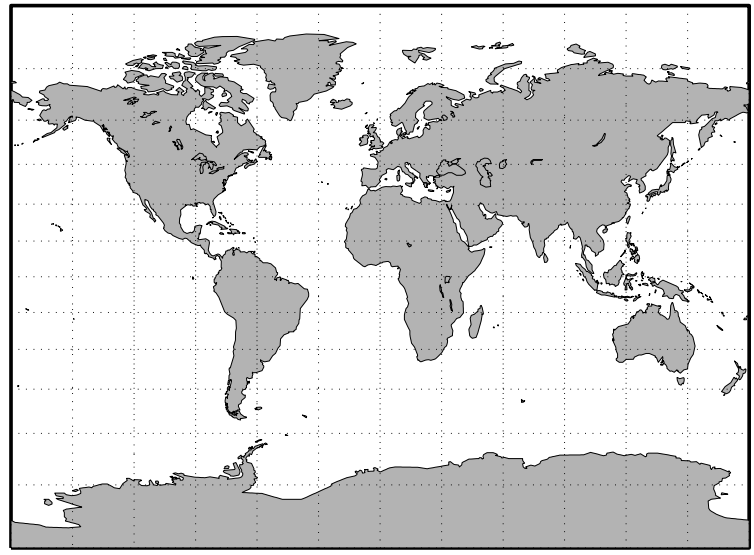
Symmetry: About any meridian or the Equator.

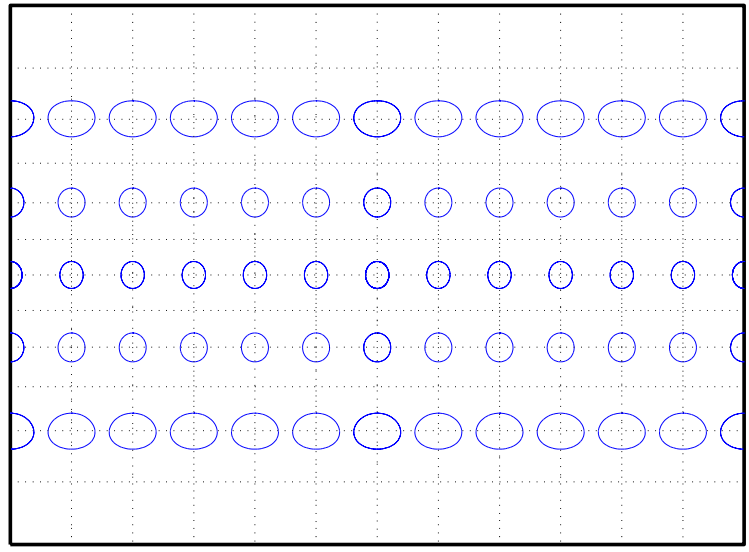
Features This is a perspective projection from a point on the Equator opposite a given meridian onto a cylinder secant at the 30° parallels. It is not equal-area, equidistant, or conformal. Scale is true along the standard parallels and constant between two parallels equidistant from the Equator. There is no distortion along the standard parallels, but it increases moderately away from these parallels, becoming severe at the poles.

Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, the standard parallel is by definition fixed at 30°.

Remarks This projection was first described in 1937, when it was used for maps in the *Bolshoi Sovietskii Atlas Mira* (Great Soviet World Atlas). It is commonly abbreviated as the BSAM projection. It is a special form of the Braun Perspective Cylindrical projection secant at 30°N and S.

Limitations This projection is available for the spherical geoid only.





Bonne Projection

Classification Pseudoconic

Syntax bonne

Graticule Central Meridian: A straight line.

Meridians: Complex curves connecting points equally spaced along each parallel and concave toward the central meridian.

Parallels: Concentric circular arcs spaced at true distances along the central meridian.

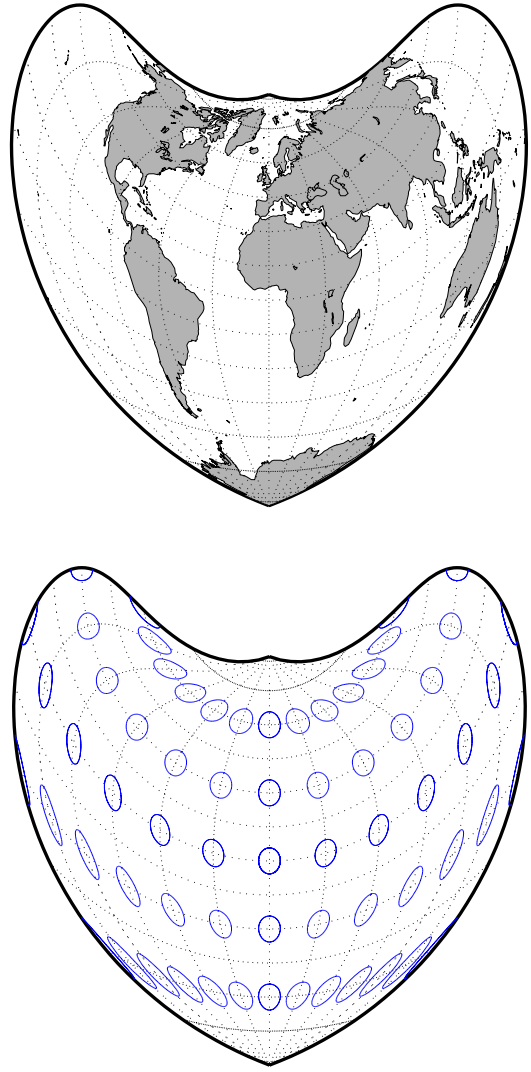
Poles: Points.

Symmetry: About the central meridian.

Features This is an equal-area projection. The curvature of the standard parallel is identical to that on a cone tangent at that latitude. The central meridian and the central parallel are free of distortion. This projection is not conformal.

Parallels This projection has one standard parallel, which is 30°N by default. It has two interesting limiting forms. If a pole is employed as the standard parallel, a Werner projection results; if the Equator is used, a Sinusoidal projection results.

Remarks This projection dates in a rudimentary form back to Claudius Ptolemy (about A.D. 100). It was further developed by Bernardus Sylvanus in 1511. It derives its name from its considerable use by Rigobert Bonne, especially in 1752.



Braun Perspective Cylindrical Projection

Classification Cylindrical

Syntax `braun`

Graticule Meridians: Equally spaced straight parallel lines.

Parallels: Unequally spaced straight parallel lines, perpendicular to the meridians. Spacing increases toward the poles.

Poles: Straight lines equal in length to the Equator.

Symmetry: About any meridian or the Equator.

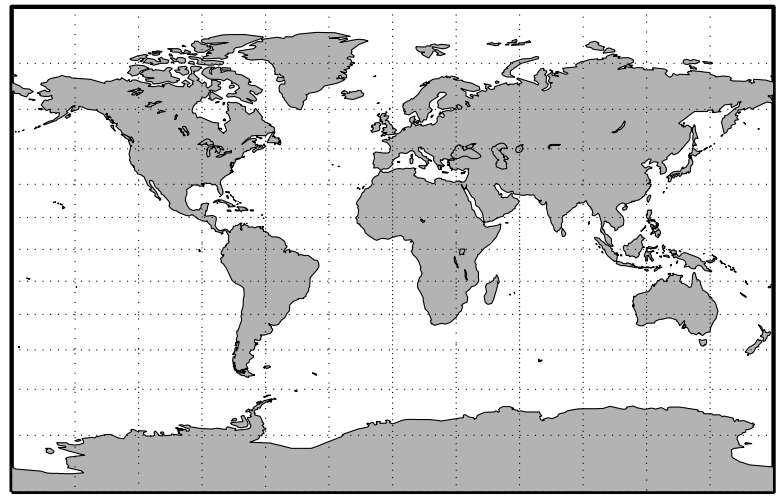
Features This is an perspective projection from a point on the Equator opposite a given meridian onto a cylinder secant at standard parallels. It is not equal-area, equidistant, or conformal. Scale is true along the standard parallels and constant between two parallels equidistant from the Equator. There is no distortion along the standard parallels, but it increases moderately away from these parallels, becoming severe at the poles.

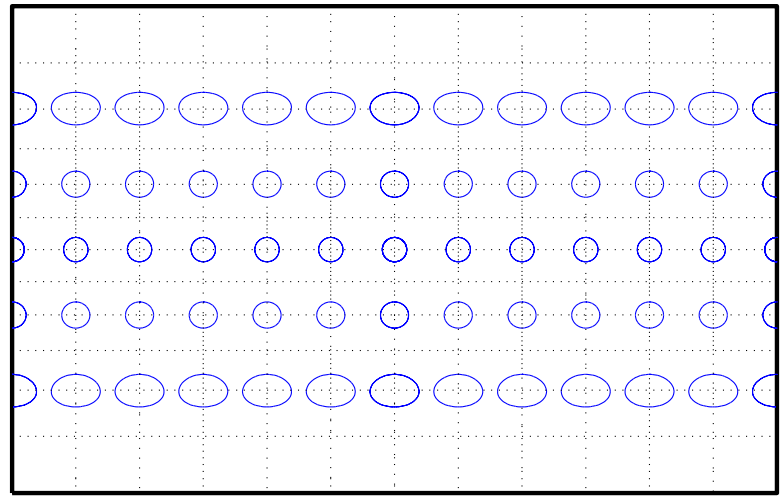
Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, any latitude may be chosen; the default is arbitrarily set to 0°.

Remarks This projection was first described by Braun in 1867. It is less well known than the specific forms of it called the Gall Stereographic and the *Bolshoi Sovietskii Atlas Mira* projections.

Limitations This projection is available for the spherical geoid only.

Braun Perspective Cylindrical Projection





Breusing Harmonic Mean Projection

Classification Azimuthal

Syntax `breusing`

Graticule The graticule described is for the polar aspect.

Meridians: Equally spaced straight lines intersecting at the central pole.

Parallels: Unequally spaced circles centered on the central pole. The opposite hemisphere cannot be shown. Spacing increases (slightly) away from the central pole.

Poles: The central pole is a point, while the opposite pole cannot be shown.

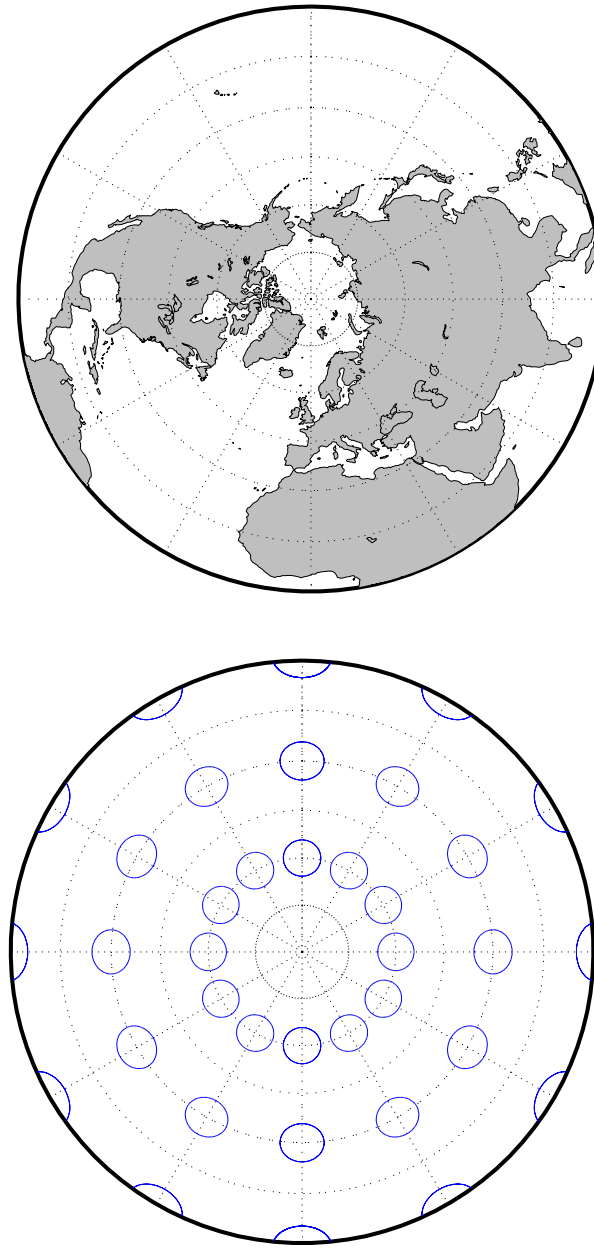
Symmetry: About any meridian.

Features This is a harmonic mean between a Stereographic and Lambert Equal-Area Azimuthal projection. It is not equal-area, equidistant, or conformal. There is no point at which scale is accurate in all directions. The primary feature of this projection is that it is minimum error – distortion is moderate throughout.

Parallels There are no standard parallels for azimuthal projections.

Remarks F. A. Arthur Breusing developed a geometric mean version of this projection in 1892. A. E. Young modified this to the harmonic mean version presented here in 1920. This projection is virtually indistinguishable from the Airy Minimum Error Azimuthal projection, presented by George Airy in 1861.

Limitations This projection is available for the spherical geoid only.



Briesemeister Projection

Classification Modified Azimuthal

Syntax `bries`

Graticule Meridians: Central meridian is straight. Other meridians are complex curves.

Parallels: Complex curves.

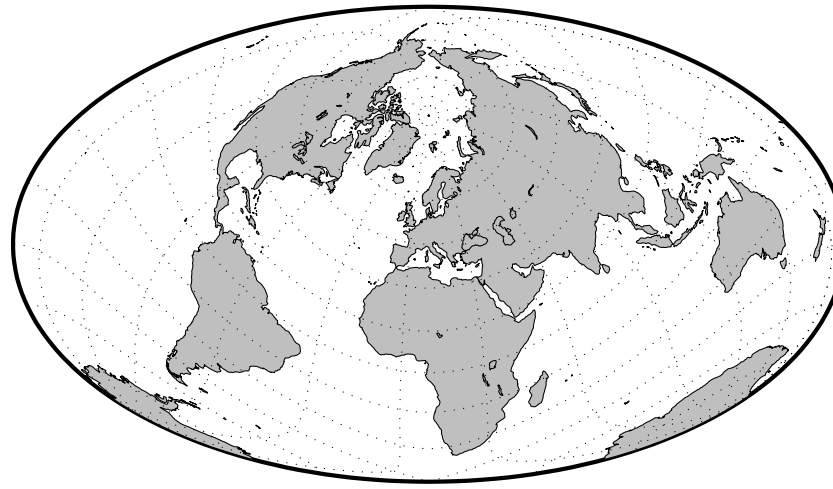
Poles: Points.

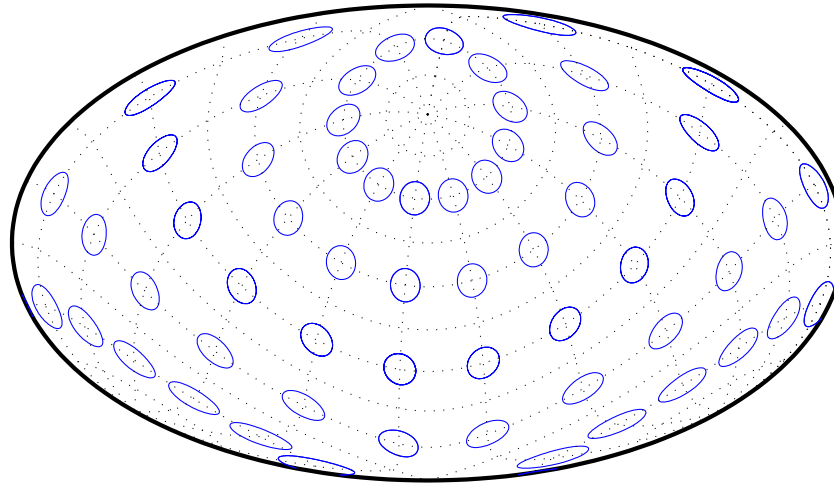
Symmetry: About the central meridian.

Features This equal-area projection groups the continents about the center of the projection. The only point free of distortion is the center point. Distortion of shape and area are moderate throughout.

Parallels There is no standard parallel for this projection.

Remarks This projection was presented by William Briesemeister in 1953. It is an oblique Hammer projection with an axis ratio of 1.75 to 1, instead of 2 to 1.





Cassini Cylindrical Projection

Classification Cylindrical

Syntax `cassini`

Graticule Central Meridian: Straight line (includes meridian opposite the central meridian in one continuous line).

Other Meridians: Straight lines if 90° from central meridian, complex curves concave toward the central meridian otherwise.

Parallels: Complex curves concave toward the nearest pole.

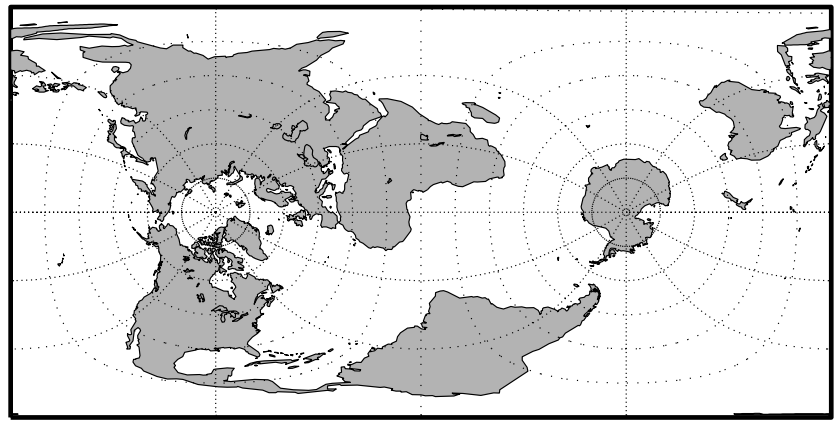
Poles: Points along the central meridian.

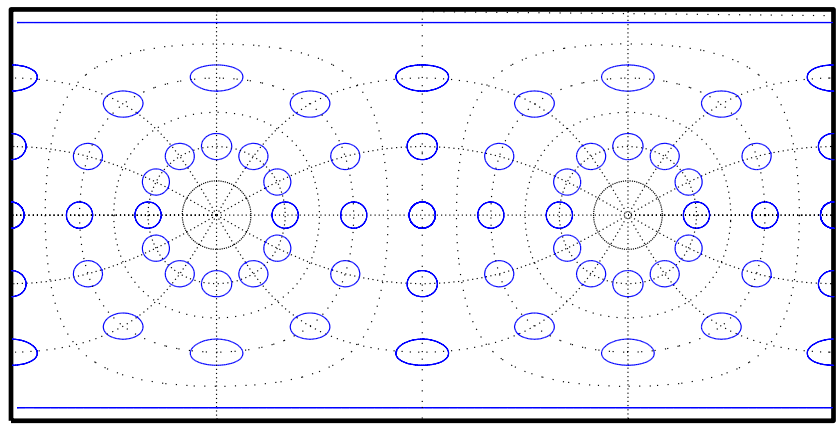
Symmetry: About any straight meridian or the Equator.

Features This is a projection onto a cylinder tangent at the central meridian. Distortion of both shape and area are functions of distance from the central meridian. Scale is true along the central meridian and along any straight line perpendicular to the central meridian (i.e., it is equidistant).

Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, the standard parallel *of the base projection* is by definition fixed at 0°.

Remarks This projection is the transverse aspect of the Plate Carrée projection, developed by César François Cassini de Thury (1714-84). It is still used for the topographic mapping of a few countries.





Central Cylindrical Projection

Classification Cylindrical

Syntax `ccylin`

Graticule Meridians: Equally spaced straight parallel lines.

Parallels: Unequally spaced straight parallel lines, perpendicular to the meridians. Spacing increases toward the poles, more rapidly than that of the Mercator projection.

Poles: Cannot be shown.

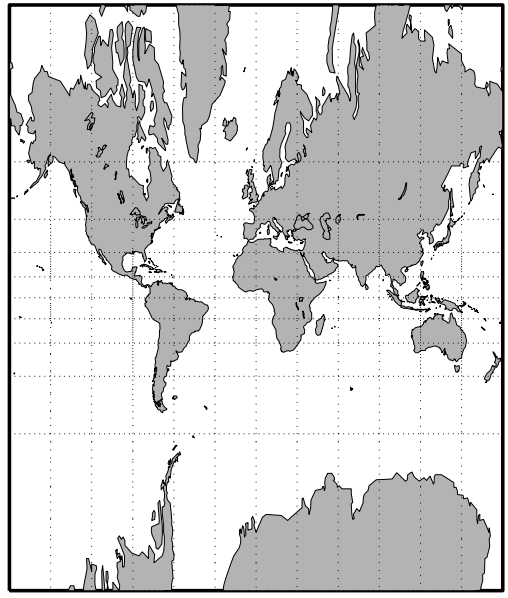
Symmetry: About any meridian or the Equator.

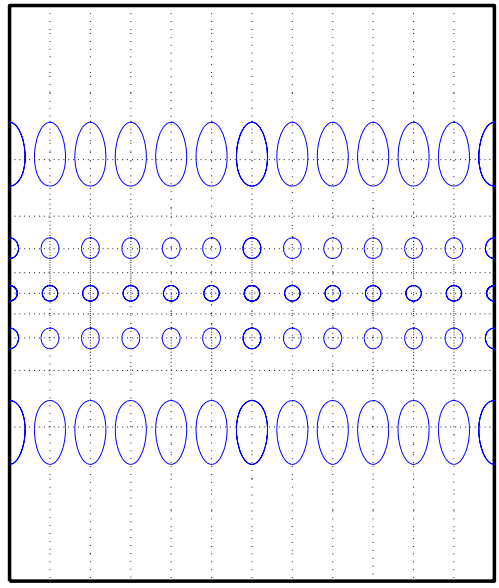
Features This is a perspective projection from the center of the Earth onto a cylinder tangent at the Equator. It is not equal-area, equidistant, or conformal. Scale is true along the Equator and constant between two parallels equidistant from the Equator. Scale becomes infinite at the poles. There is no distortion along the Equator, but it increases rapidly away from the Equator.

Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, the standard parallel is by definition fixed at 0°.

Remarks The origin of this projection is unknown; it has little use beyond the educational aspects of its method of projection and as a comparison to the Mercator projection, which is not perspective. The transverse aspect of the Central Cylindrical is called the Wetch projection.

Limitations This projection is available for the spherical geoid only. Data at latitudes greater than 75° is trimmed to prevent large values from dominating the display.





Collignon Projection

Classification Pseudocylindrical

Syntax `collig`

Graticule Meridians: Equally spaced straight lines converging at the North Pole.

Parallels: Unequally spaced straight parallel lines, farthest apart near the North Pole, closest near the South Pole

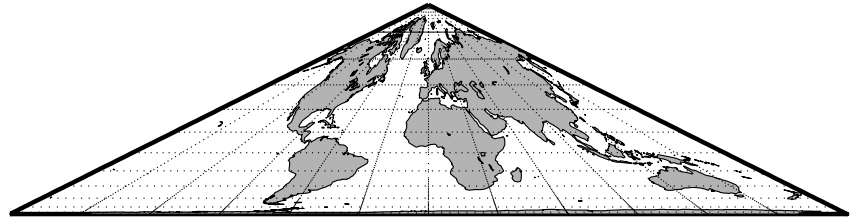
Poles: North Pole is a point, South Pole is a line 1.41 as long as the Equator.

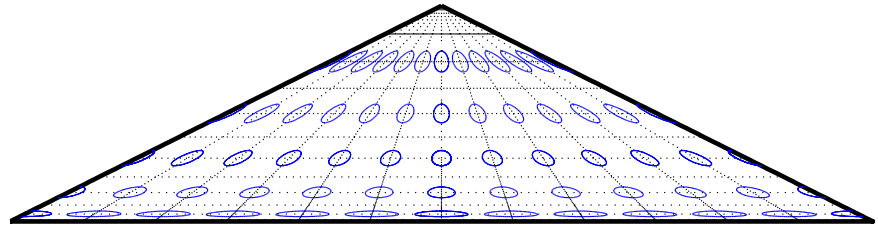
Symmetry: About the central meridian.

Features This is a novelty projection showing a straight-line, equal-area graticule. Scale is true along the 15°51'N parallel, constant along any parallel, and *different* for any pair of parallels. Distortion is severe in many regions, and is only absent at 15°51'N on the central meridian. This projection is not conformal or equidistant.

Parallels This projection has one standard parallel, which is by definition fixed at 15°51'.

Remarks This projection was presented by Édouard Collignon in 1865.





Craster Parabolic Projection

Classification Pseudocylindrical

Syntax `craster`

Graticule Central Meridian: Straight line half as long as the Equator.

Other Meridians: Equally spaced parabolas intersecting at the poles and concave toward the central meridian.

Parallels: Unequally spaced straight parallel lines, perpendicular to the central meridian. Spacing changes very gradually and is greatest near the Equator.

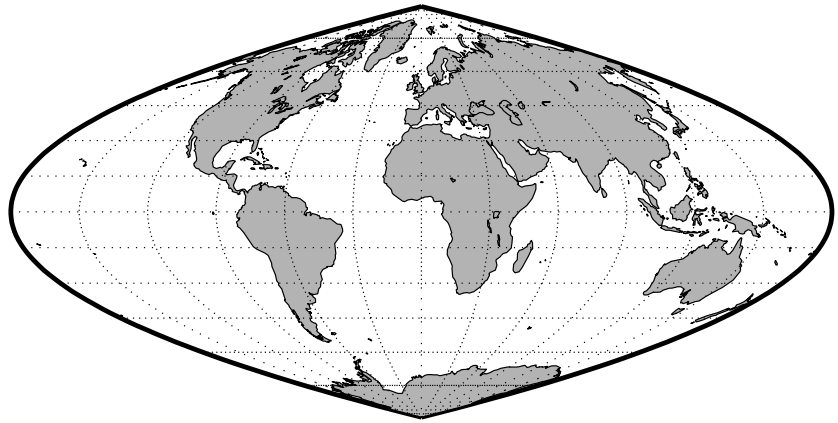
Poles: Points.

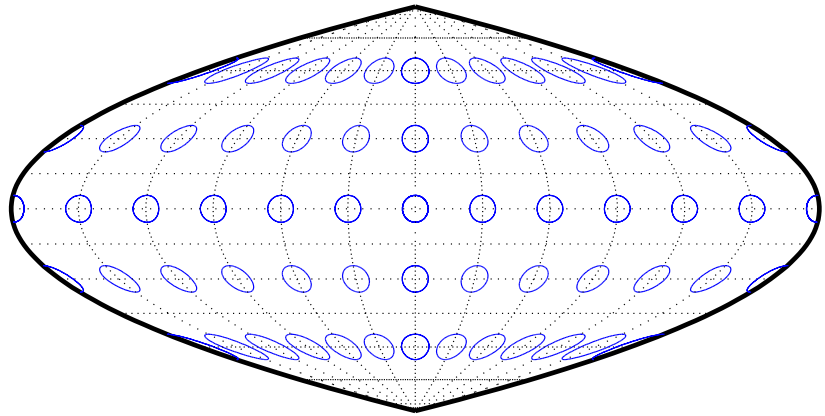
Symmetry: About the central meridian or the Equator.

Features This is an equal-area projection. Scale is true along the 36°46' parallels and is constant along any parallel and between any pair of parallels equidistant from the Equator. Distortion is severe near the outer meridians at high latitudes, but less so than the Sinusoidal projection. This projection is free of distortion only at the two points where the central meridian intersects the 36°46' parallels. This projection is not conformal or equidistant.

Parallels For this projection, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. The standard parallel is by definition fixed at 36°46'.

Remarks This projection was developed by John Evelyn Edmund Craster in 1929; it was further developed by Charles H. Deetz and O.S. Adams in 1934. It was presented independently in 1934 by Putnins as his P_4 projection.





Eckert I Projection

Classification Pseudocylindrical

Syntax `eckert1`

Graticule Central Meridian: Straight line half as long as the Equator.

Other Meridians: Equally spaced straight converging lines broken at the Equator.

Parallels: Equally spaced straight parallel lines, perpendicular to the central meridian.

Poles: Lines half as long as the Equator.

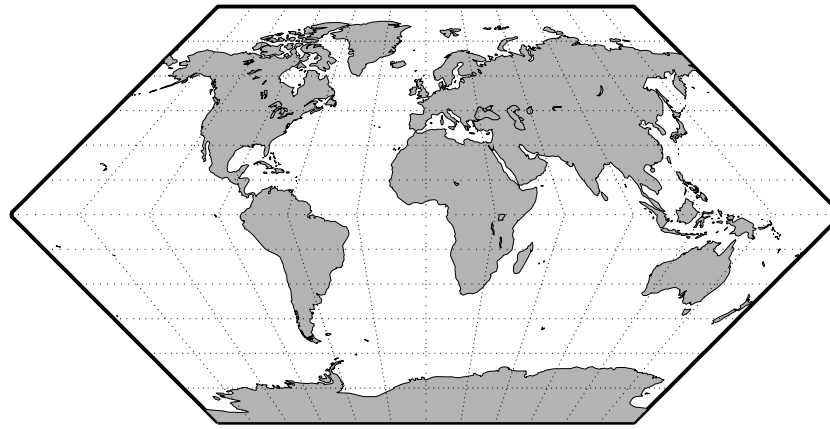
Symmetry: About the central meridian or the Equator.

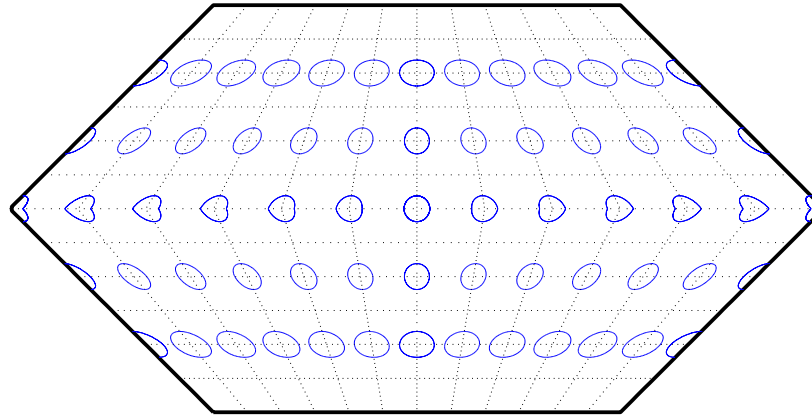
Features Scale is true along the 47°10' parallels and is constant along any parallel, between any pair of parallels equidistant from the Equator, and along any given meridian. It is not free of distortion at any point, and the break at the Equator introduces excessive distortion there; regardless of the appearance here, the Tissot indicatrices are of indeterminate shape along the Equator. This novelty projection is not equal-area or conformal.

Parallels For this projection, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. The standard parallel is by definition fixed at 47°10'.

Remarks This projection was presented by Max Eckert in 1906.

Limitations This projection is available for the spherical geoid only.





Eckert II Projection

Classification Pseudocylindrical

Syntax `eckert2`

Graticule Central Meridian: Straight line half as long as the Equator.

Other Meridians: Equally spaced straight converging lines broken at the Equator.

Parallels: Unequally spaced straight parallel lines, perpendicular to the central meridian. Spacing is widest near the Equator.

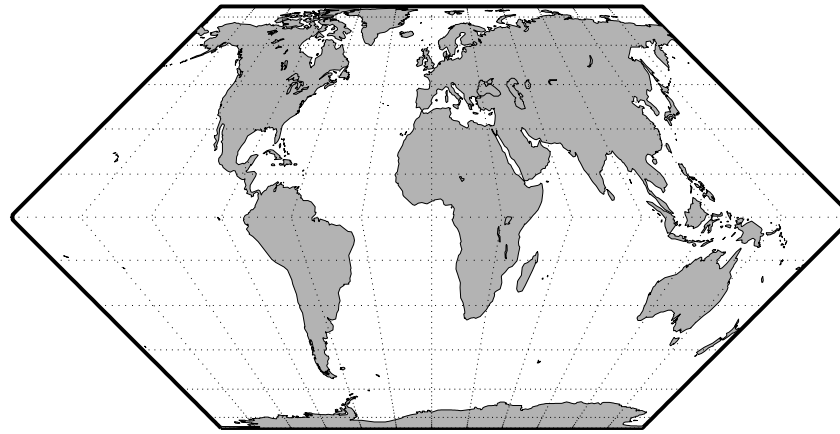
Poles: Lines half as long as the Equator.

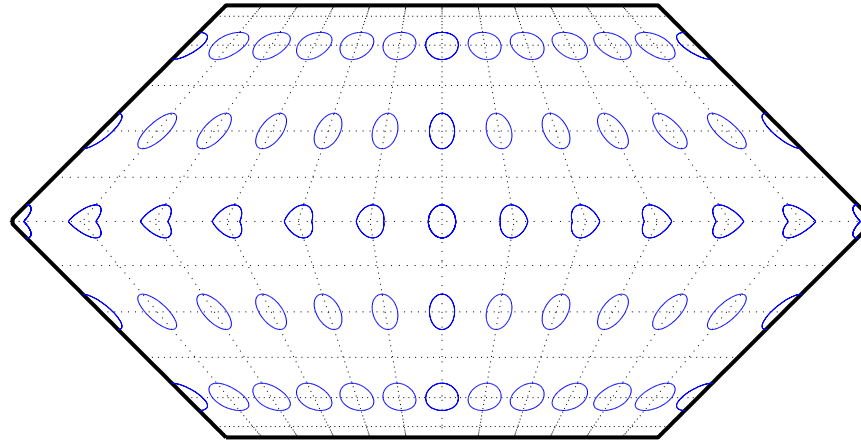
Symmetry: About the central meridian or the Equator.

Features This is an equal-area projection. Scale is true along the 55°10' parallels and is constant along any parallel and between any pair of parallels equidistant from the Equator. It is not free of distortion at any point except at 55°10'N and S along the central meridian; the break at the Equator introduces excessive distortion there. Regardless of the appearance here, the Tissot indicatrices are of indeterminate shape along the Equator. This novelty projection is not conformal or equidistant.

Parallels For this projection, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. The standard parallel is by definition fixed at 55°10'.

Remarks This projection was presented by Max Eckert in 1906.





Eckert III Projection

Classification Pseudocylindrical

Syntax `eckert3`

Graticule Central Meridian: Straight line half as long as the Equator.

Other Meridians: Equally spaced semiellipses concave toward the central meridian. The outer meridians, 180° east and west of the central meridian, are semicircles.

Parallels: Equally spaced straight parallel lines, perpendicular to the central meridian.

Poles: Lines half as long as the Equator.

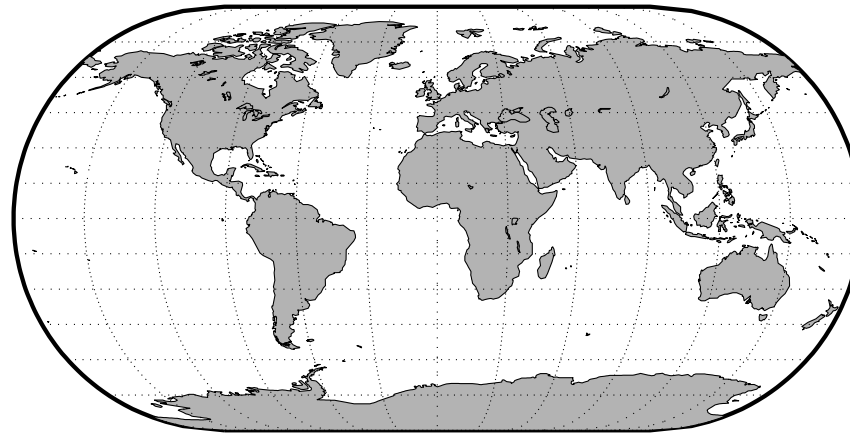
Symmetry: About the central meridian or the Equator.

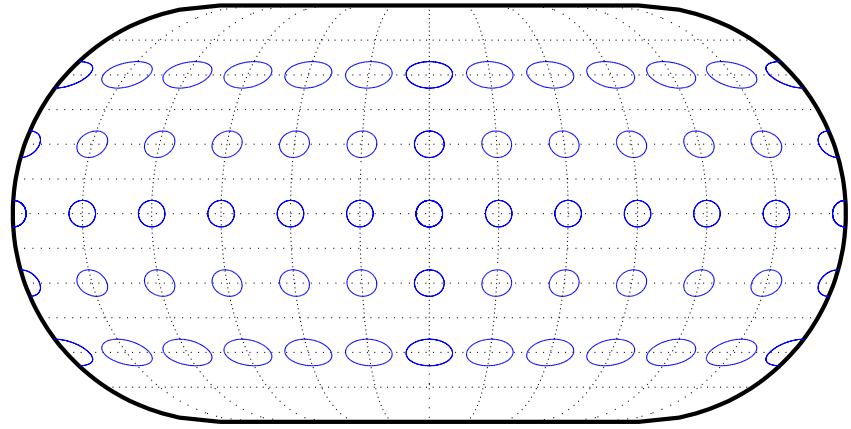
Features Scale is true along the 35°58' parallels and is constant along any parallel and between any pair of parallels equidistant from the Equator. No point is free of all scale distortion, but the Equator is free of angular distortion. This projection is not equal-area, conformal, or equidistant.

Parallels For this projection, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. The standard parallel is by definition fixed at 35°58'.

Remarks This projection was presented by Max Eckert in 1906.

Limitations This projection is available for the spherical geoid only.





Eckert IV Projection

Classification Pseudocylindrical

Syntax `eckert4`

Graticule Central Meridian: Straight line half as long as the Equator.

Other Meridians: Equally spaced semiellipses concave toward the central meridian. The outer meridians, 180° east and west of the central meridian, are semicircles.

Parallels: Unequally spaced straight parallel lines, perpendicular to the central meridian. Spacing is greatest toward the Equator.

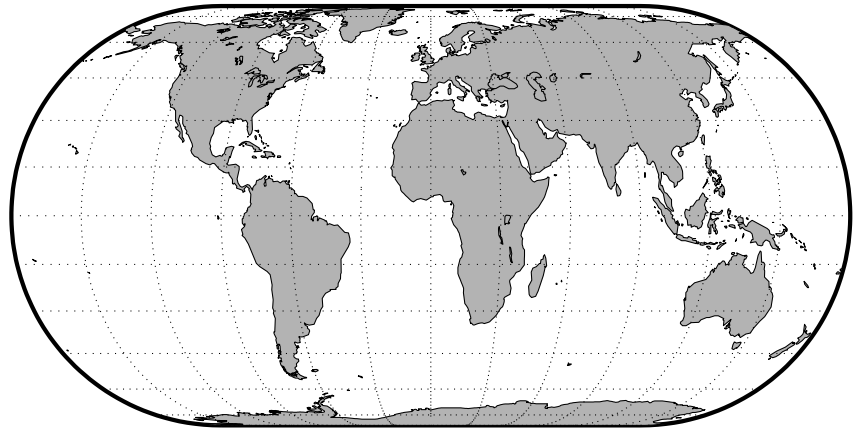
Poles: Lines half as long as the Equator.

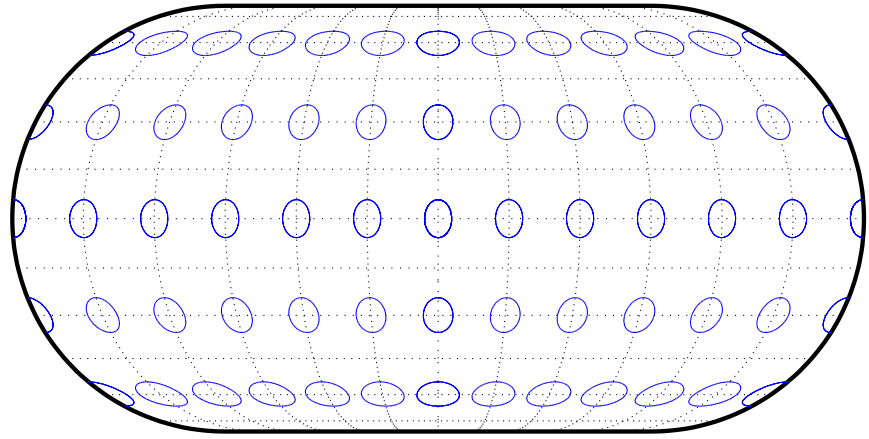
Symmetry: About the central meridian or the Equator.

Features This is an equal-area projection. Scale is true along the 40°30' parallels and is constant along any parallel and between any pair of parallels equidistant from the Equator. It is free of distortion only at the two points where the 40°30' parallels intersect the central meridian. This projection is not conformal or equidistant.

Parallels For this projection, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. The standard parallel is by definition fixed at 40°30'.

Remarks This projection was presented by Max Eckert in 1906.





Eckert V Projection

Classification Pseudocylindrical

Syntax `eckert5`

Graticule Central Meridian: Straight line half as long as the Equator.

Other Meridians: Equally spaced sinusoidal curves concave toward the central meridian.

Parallels: Equally spaced straight parallel lines, perpendicular to the central meridian.

Poles: Lines half as long as the Equator.

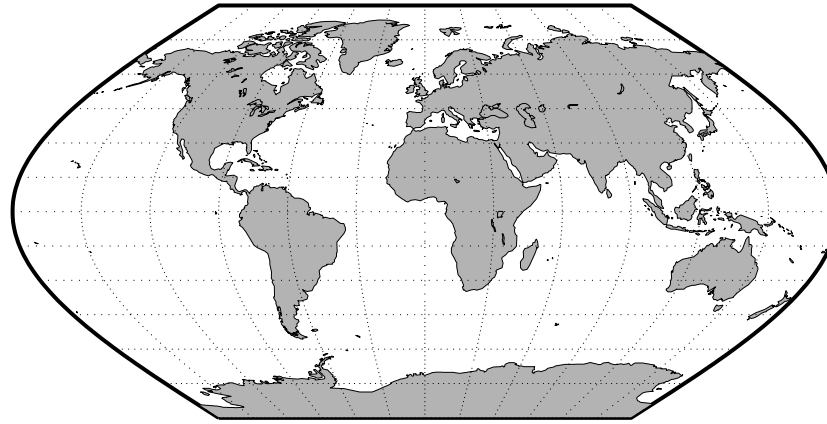
Symmetry: About the central meridian or the Equator.

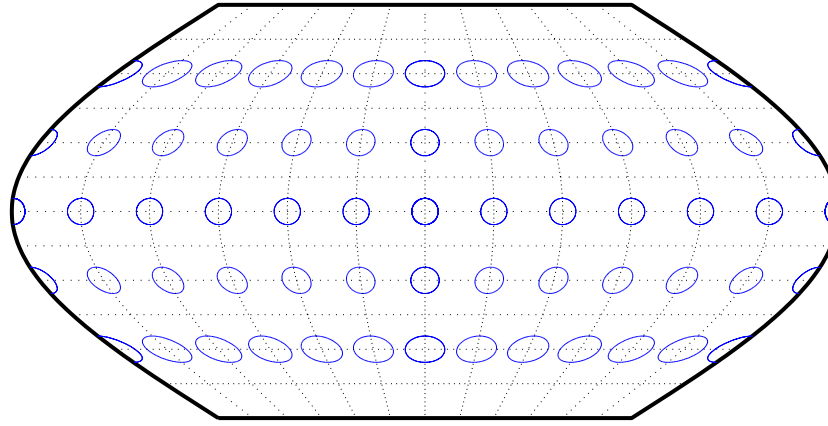
Features This projection is an arithmetic average of the *x* and *y* coordinates of the Sinusoidal and Plate Carrée projections. Scale is true along latitudes 37°55'N and S, and is constant along any parallel and between any pair of parallels equidistant from the Equator. There is no point free of all distortion, but the Equator is free of angular distortion. This projection is not equal-area, conformal, or equidistant.

Parallels This projection has one standard parallel, which is by definition fixed at 0°.

Remarks This projection was presented by Max Eckert in 1906.

Limitations This projection is available for the spherical geoid only.





Eckert VI Projection

Classification Pseudocylindrical

Syntax `eckert6`

Graticule Central Meridian: Straight line half as long as the Equator.

Other Meridians: Equally spaced sinusoidal curves concave toward the central meridian.

Parallels: Unequally spaced straight parallel lines, perpendicular to the central meridian. Spacing is greatest toward the Equator.

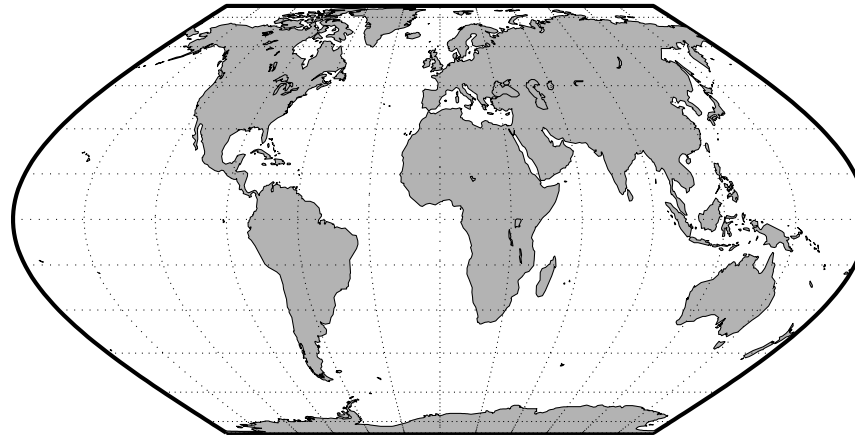
Poles: Lines half as long as the Equator.

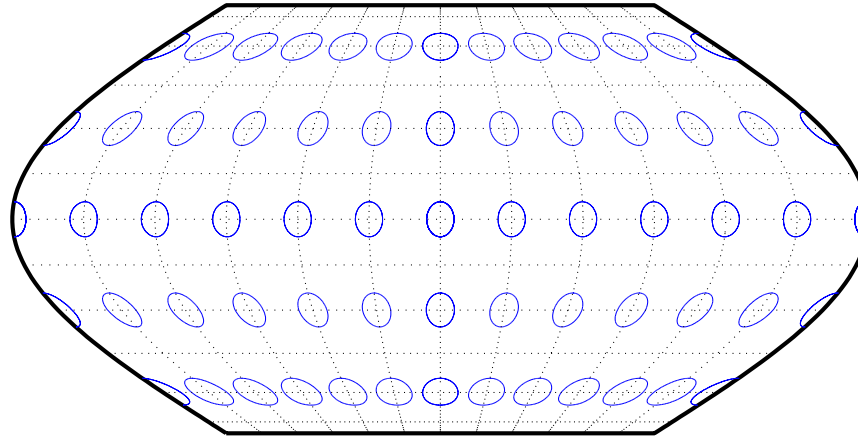
Symmetry: About the central meridian or the Equator.

Features This is an equal-area projection. Scale is true along the 49°16' parallels and is constant along any parallel and between any pair of parallels equidistant from the Equator. It is free of distortion only at the two points where the 49°16' parallels intersect the central meridian. This projection is not conformal or equidistant.

Parallels For this projection, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. The standard parallel is by definition fixed at 49°16'.

Remarks This projection was presented by Max Eckert in 1906.





Equal-Area Cylindrical Projection

Classification Cylindrical

Syntax `eqacylin`

Graticule Meridians: Equally spaced straight parallel lines.

Parallels: Unequally spaced straight parallel lines, perpendicular to the meridians. Spacing is closest near the poles.

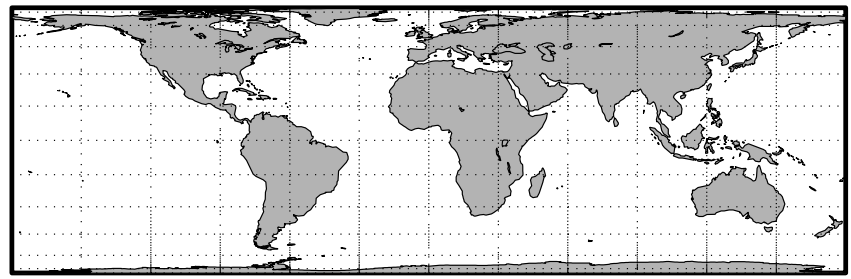
Poles: Straight lines equal in length to the Equator.

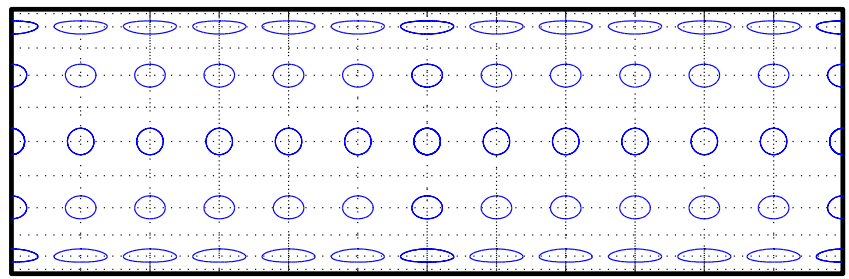
Symmetry: About any meridian or the Equator.

Features This is an orthographic projection onto a cylinder secant at the standard parallels. It is equal-area, but distortion of shape increases with distance from the standard parallels. Scale is true along the standard parallels and constant between two parallels equidistant from the Equator. This projection is not equidistant.

Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, any latitude may be chosen; the default is arbitrarily set to 0° (the Lambert variation).

Remarks This projection was proposed by Johann Heinrich Lambert (1772), a prolific cartographer who proposed seven different important projections. The form of this projection tangent at the Equator is often called the Lambert Equal-Area Cylindrical projection. That and other special forms of this projection are included separately in this guide, including the Gall Orthographic, the Behrmann Cylindrical, the Balthasart Cylindrical, and the Trystan Edwards Cylindrical projections.





Equidistant Azimuthal Projection

Classification Azimuthal

Syntax `eqdazim`

Graticule The graticule described is for the polar aspect.

Meridians: Equally spaced straight lines intersecting at a central pole. The angles between them are the true angles.

Parallels: Equally spaced circles, centered on the central pole. The entire Earth may be shown.

Poles: Central pole is a point. The opposite pole is a bounding circle with a radius twice that of the Equator.

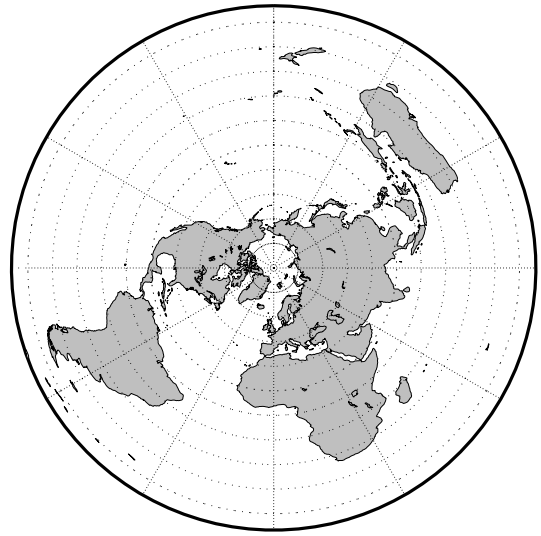
Symmetry: About any meridian.

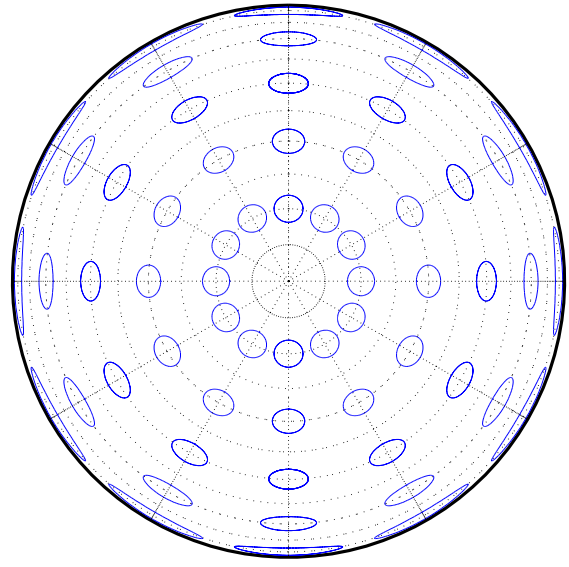
Features This is an equidistant projection. It is neither equal-area nor conformal. In the polar aspect, scale is true along any meridian. The projection is distortion free only at the center point. Distortion is moderate for the inner hemisphere, but it becomes extreme in the outer hemisphere.

Parallels There are no standard parallels for azimuthal projections.

Remarks This projection may have been first used by the ancient Egyptians for star charts. Several cartographers used it during the sixteenth century, including Guillaume Postel, who used it in 1581. Other names for this projection include Postel and Zenithal Equidistant.

Limitations This projection is available for the spherical geoid only.





Equidistant Conic Projection

Classification Conic

Syntax `eqdconic`

Graticule Meridians: Equally spaced straight lines converging to a common point, usually beyond the pole. The angles between the meridians are less than the true angles.

Parallels: Equally spaced concentric circular arcs centered on the point of meridanal convergence.

Poles: Normally circular arcs, enclosing the same angle as the displayed parallels.

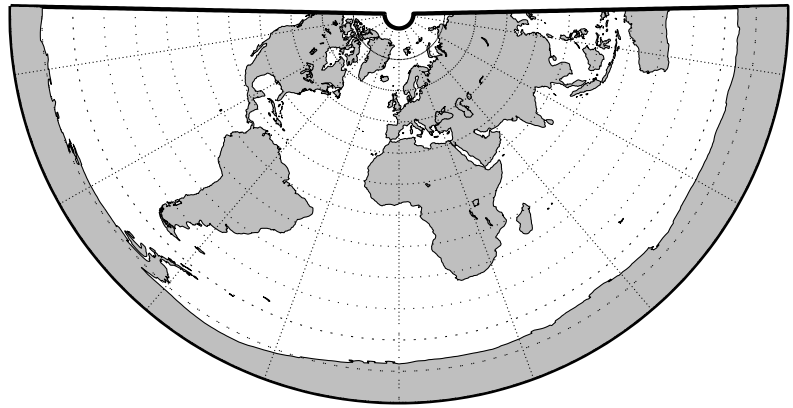
Symmetry: About any meridian.

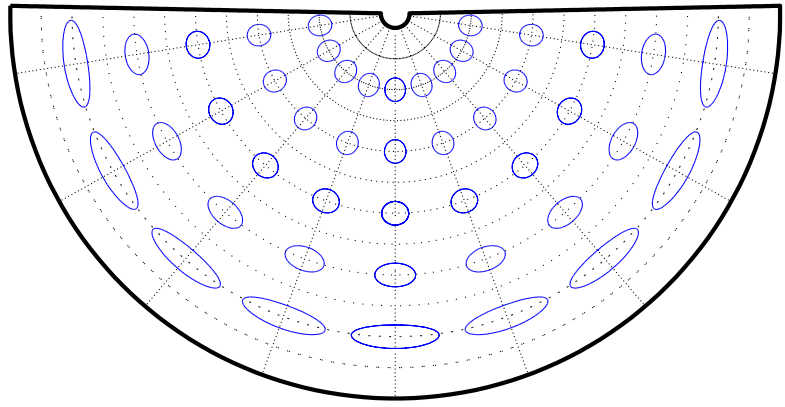
Features Scale is true along each meridian and the one or two selected standard parallels. Scale is constant along any parallel. This projection is free of distortion along the two standard parallels. Distortion is constant along any other parallel. This projection provides a compromise in distortion between conformal and equal-area conic projections, of which it is neither.

Parallels The cone of projection has interesting limiting forms. If a pole is selected as a single standard parallel, the cone is a plane, and an Equidistant Azimuthal projection results. If two parallels are chosen, not symmetric about the Equator, then an Equidistant Conic projection results. If a pole is selected as one of the standard parallels, then the projected pole is a point, otherwise the projected pole is an arc. If the Equator is so chosen, the cone becomes a cylinder and a Plate Carrée projection results. If two parallels equidistant from the Equator are chosen as the standard parallels, an Equidistant Cylindrical projection results. The default parallels are [15 75].

Remarks In a rudimentary form, this projection dates back to Claudius Ptolemy, about A.D. 100. Improvements were developed by Johannes Ruysch in 1508, Gerardus Mercator in the late 16th century, and Nicolas de l'Isle in 1745. It is also known as the Simple Conic or Conic projection.

Limitations Longitude data greater than 135° east or west of the central meridian is trimmed.





Equidistant Cylindrical Projection

Classification Cylindrical

Syntax `eqdcylin`

Graticule Meridians: Equally spaced straight parallel lines more than half as long as the Equator.

Parallels: Equally spaced straight parallel lines, perpendicular to and having wider spacing than the meridians.

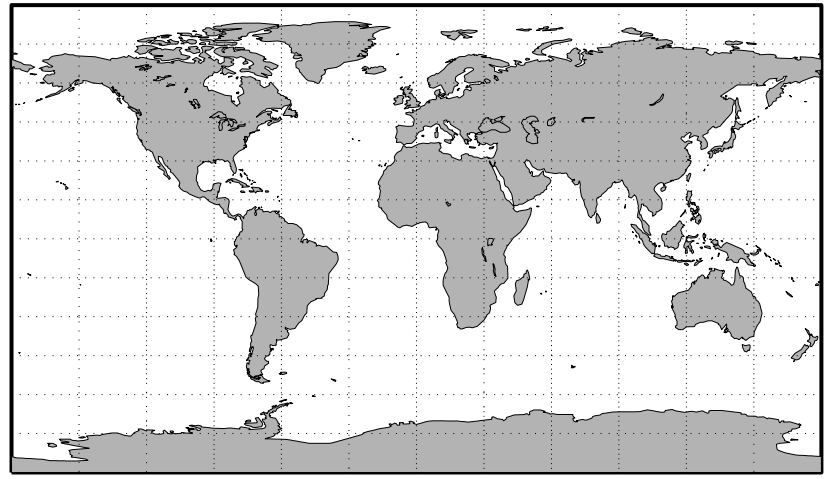
Poles: Straight lines equal in length to the Equator.

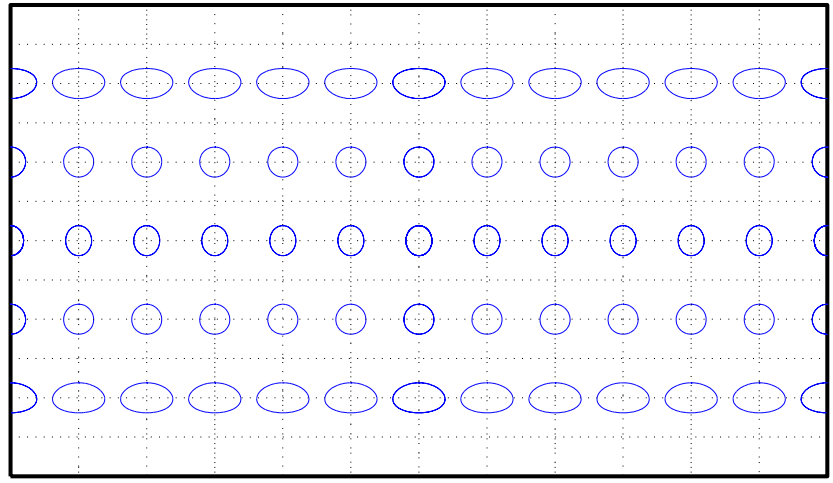
Symmetry: About any meridian or the Equator.

Features This is a projection onto a cylinder secant at the standard parallels. Distortion of both shape and area increase with distance from the standard parallels. Scale is true along all meridians (i.e., it is equidistant) and the standard parallels and is constant along any parallel and along the parallel of opposite sign.

Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, any latitude can be chosen; the default is arbitrarily set to 30°.

Remarks This projection was first used by Marinus of Tyre about A.D. 100. Special forms of this projection are the Plate Carrée, with a standard parallel at 0°, and the Gall Isographic, with standard parallels at 45°N and S. Other names for this projection include Equirectangular, Rectangular, Projection of Marinus, *La Carte Parallélogrammatique*, and *Die Rechteckige Plattkarte*.





Fournier Projection

Classification Pseudocylindrical

Syntax `fournier`

Graticule Meridians: Equally spaced elliptical curves converging at the poles.

Parallels: Straight lines.

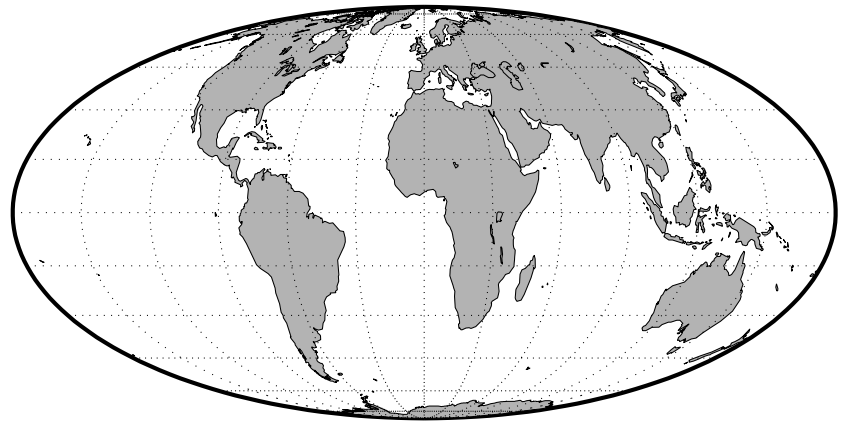
Poles: Points.

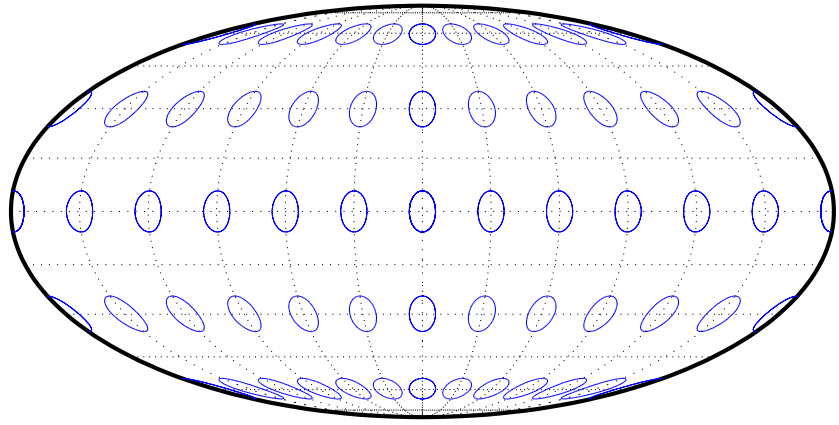
Symmetry: About the Equator and central meridian.

Features This projection is equal-area. Scale is constant along any parallel or pair of parallels equidistant from the Equator. This projection is neither equidistant nor conformal.

Parallels There is no standard parallel for this projection.

Remarks This projection was first described in 1643 by Georges Fournier. This is actually his second projection, the Fournier II.





Gall Isographic Projection

Purpose Cylindrical

Syntax `giso`

Graticule Meridians: Equally spaced straight parallel lines more than half as long as the Equator.

Parallels: Equally spaced straight parallel lines, perpendicular to and having wider spacing than the meridians.

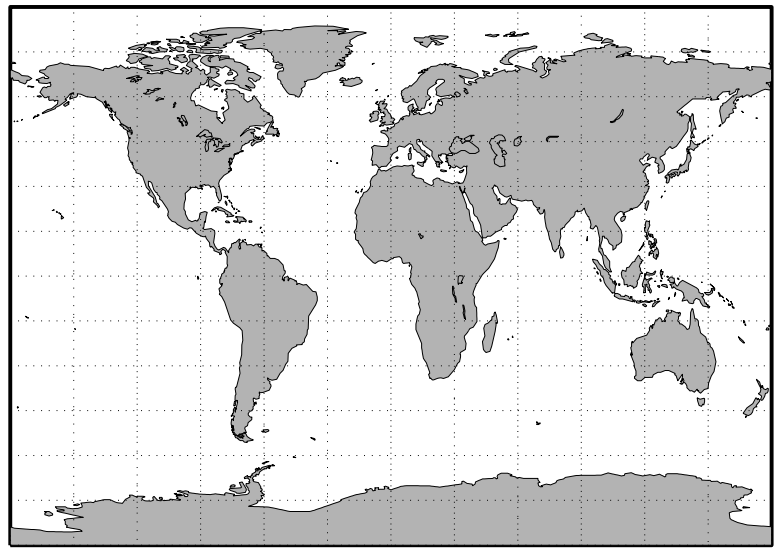
Poles: Straight lines equal in length to the Equator.

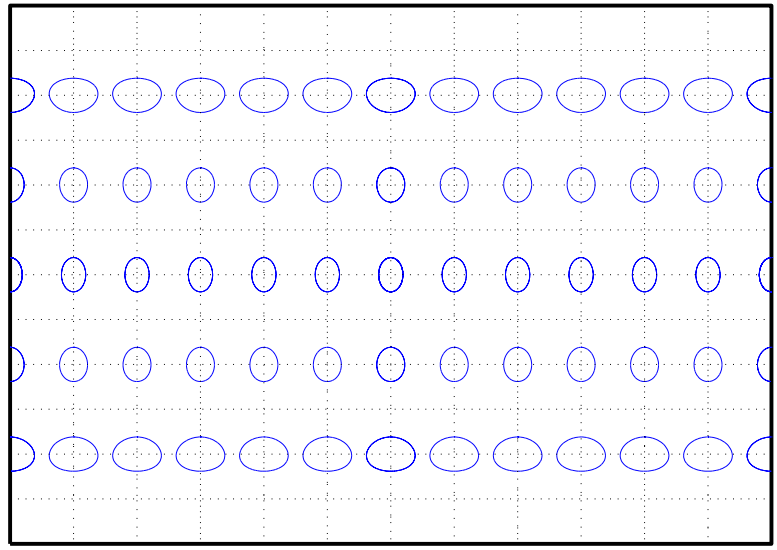
Symmetry: About any meridian or the Equator.

Features This is a projection onto a cylinder secant at the 45° parallels. Distortion of both shape and area increase with distance from the standard parallels. Scale is true along all meridians (i.e., it is equidistant) and the two standard parallels, and is constant along any parallel and along the parallel of opposite sign.

Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, the standard parallel is by definition fixed at 45°.

Remarks This projection is a specific case of the Equidistant Cylindrical projection, with standard parallels at 45°N and S.





Gall Orthographic Projection

Classification Cylindrical

Syntax `gortho`

Graticule Meridians: Equally spaced straight parallel lines.

Parallels: Unequally spaced straight parallel lines, perpendicular to the meridians. Spacing is closest near the poles.

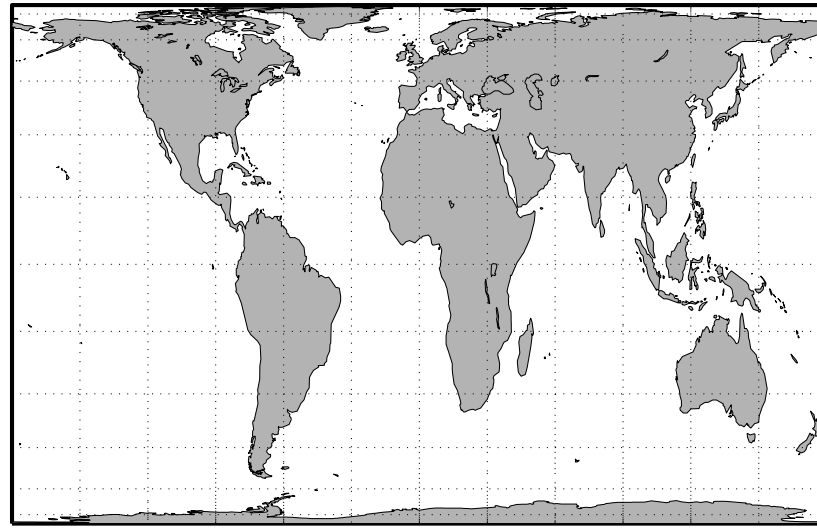
Poles: Straight lines equal in length to the Equator.

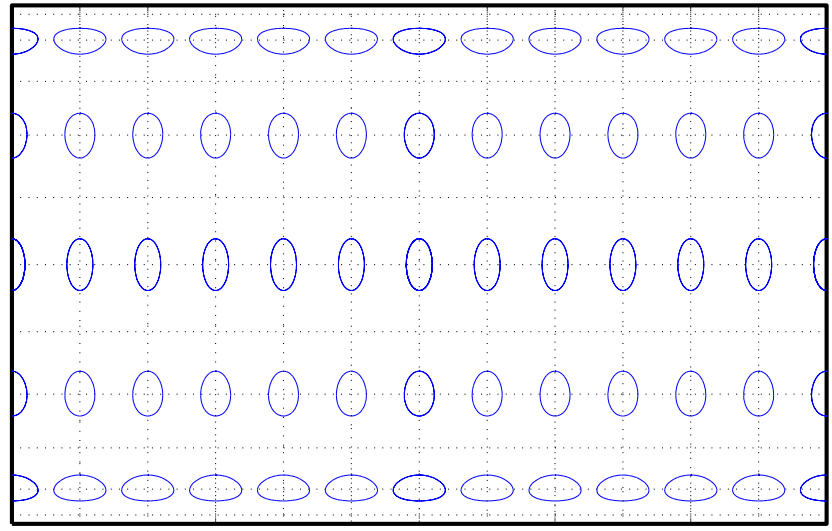
Symmetry: About any meridian or the Equator.

Features This is an orthographic projection onto a cylinder secant at the 45° parallels. It is equal-area, but distortion of shape increases with distance from the standard parallels. Scale is true along the standard parallels and constant between two parallels equidistant from the Equator. This projection is not equidistant.

Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, the standard parallel is by definition fixed at 45°.

Remarks This projection is named for James Gall, who originated it in 1855 and is a special form of the Equal-Area Cylindrical projection secant at 45°N and S. This projection is also known as the Peters projection.





Gall Stereographic Projection

Classification Cylindrical

Syntax `gstereo`

Graticule Meridians: Equally spaced straight parallel lines 0.77 as long as the Equator.

Parallels: Unequally spaced straight parallel lines, perpendicular to the meridians. Spacing increases toward the poles.

Poles: Straight lines equal in length to the Equator.

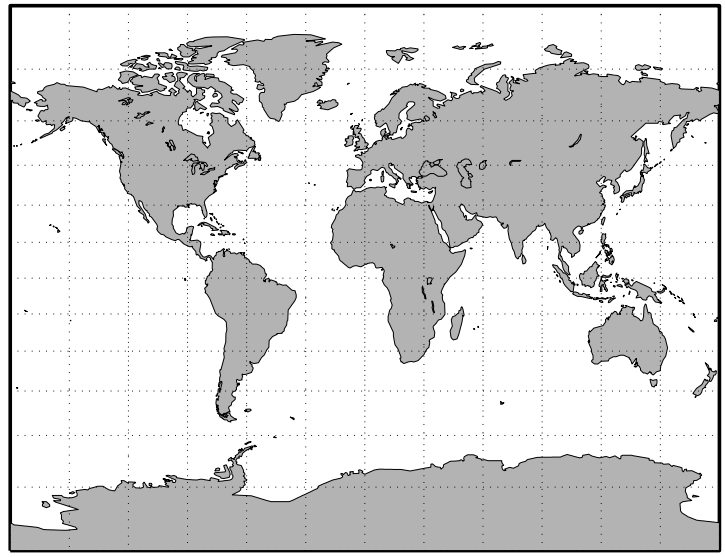
Symmetry: About any meridian or the Equator.

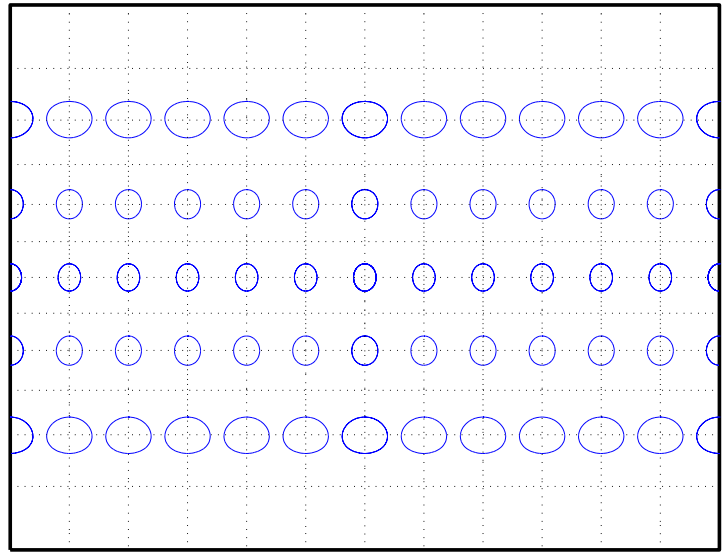
Features This is a perspective projection from a point on the Equator opposite a given meridian onto a cylinder secant at the 45° parallels. It is not equal-area, equidistant, or conformal. Scale is true along the standard parallels and constant between two parallels equidistant from the Equator. There is no distortion along the standard parallels, but it increases moderately away from these parallels, becoming severe at the poles.

Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, the standard parallel is by definition fixed at 45°.

Remarks This projection was presented by James Gall in 1855. It is also known simply as the Gall projection. It is a special form of the Braun Perspective Cylindrical projection secant at 45°N and S.

Limitations This projection is available for the spherical geoid only.





Globe

Classification Spherical

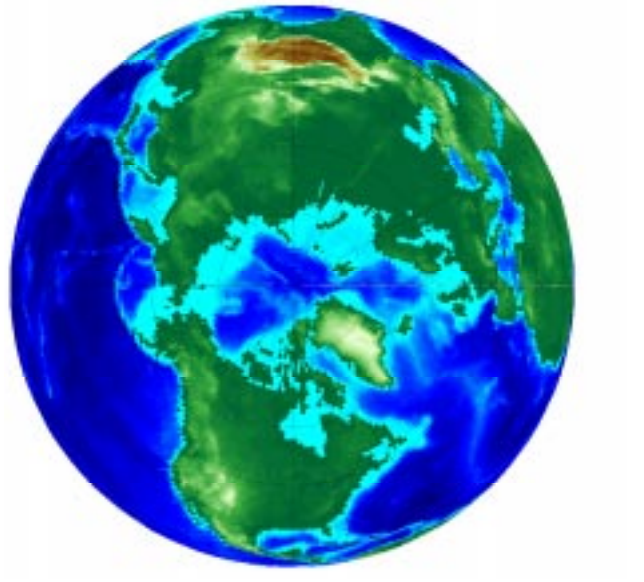
Syntax `globe`

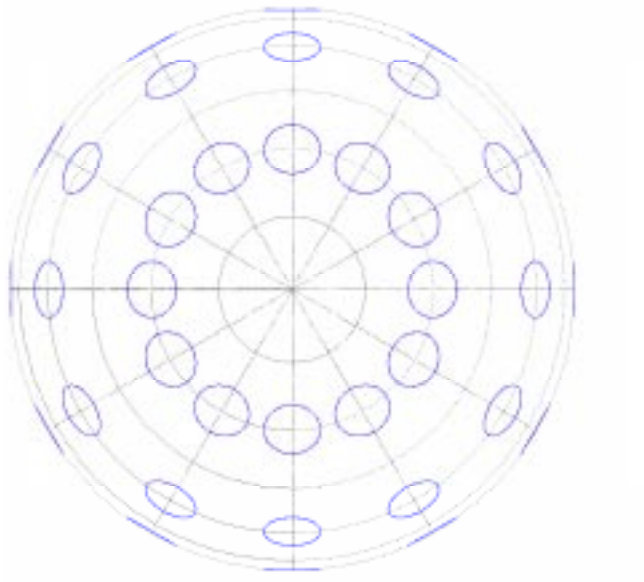
Graticule This *projection* is constructed by calculating a three-dimensional frame and displaying the map objects on the surface of this frame.

Features In the three-dimensional sense, this projection is true in scale, equal-area, conformal, minimum error, and equidistant everywhere. When displayed, however, this projection looks like an Orthographic azimuthal projection, provided that the MATLAB Axes `Projection` property is set to `'orthographic'`.

Parallels The globe requires no standard parallels.

Remarks This is the only three-dimensional representation provided for display. Unless some other display purpose requires three dimensions, the Orthographic projection's display is equivalent.





Gnomonic Projection

Classification Azimuthal

Syntax `gnomonic`

Graticule The graticule described is for a polar aspect.

Meridians: Equally spaced straight lines intersecting at the central pole. The angles displayed are the true angles between meridians.

Parallels: Unequally spaced circles centered on the central pole. Spacing increases rapidly away from this pole. The Equator and the opposite hemisphere cannot be shown

Pole: The central pole is a point; the other pole is not shown.

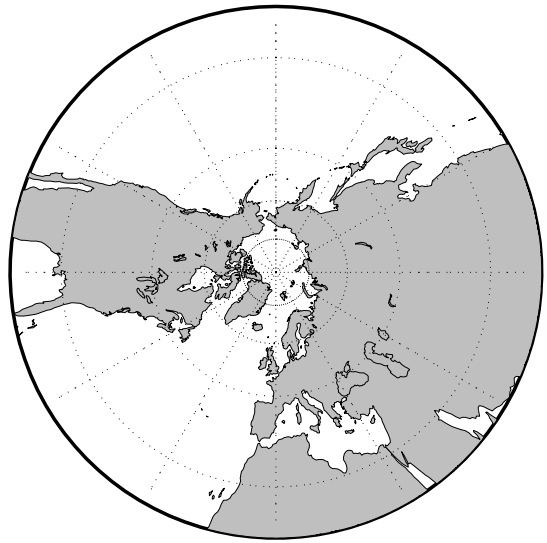
Symmetry: About any meridian.

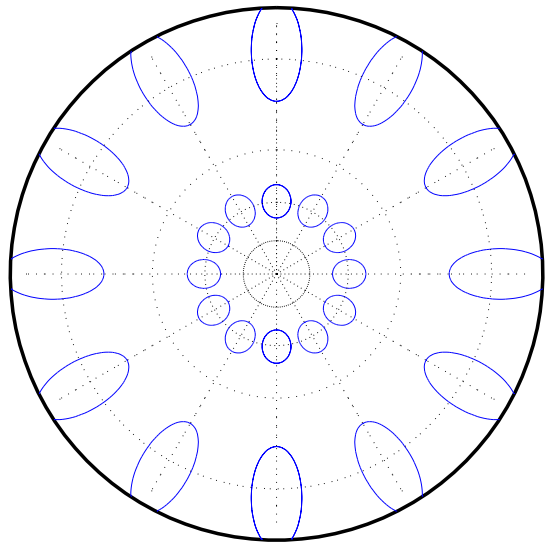
Features This is a perspective projection from the center of the globe on a plane tangent at the center point, which is a pole in the common polar aspect, but can be any point. Less than one hemisphere can be shown with this projection, regardless of its center point. The significant property of this projection is that all great circles are straight lines. This is useful in navigation, as a great circle is the shortest path between two points on the globe. Only the center point enjoys true scale and zero distortion. This projection is neither conformal nor equal-area.

Parallels There are no standard parallels for azimuthal projections.

Remarks This projection may have been first developed by Thales around 580 B.C. Its name is derived from the gnomon, the face of a sundial, since the meridians radiate like hour markings. This projection is also known as a Gnomic or Central projection.

Limitations This projection is available for the spherical geoid only. Data greater than 65° distant from the center point is trimmed.





Goode Homolosine Projection

Classification Pseudocylindrical

Syntax goode

Graticule Central Meridian: Straight line 0.44 as long as the Equator.

Other Meridians: Equally spaced sinusoidal curves between the 40°44'11.8" parallels and elliptical arcs elsewhere, all concave toward the central meridian. The result is a slight, visible bend in the meridians at 40°44'11.8" N and S.

Parallels: Straight parallel lines, perpendicular to the central meridian. Equally spaced between the 40°44'11.8" parallels, with gradually decreasing spacing outside these parallels.

Poles: Points.

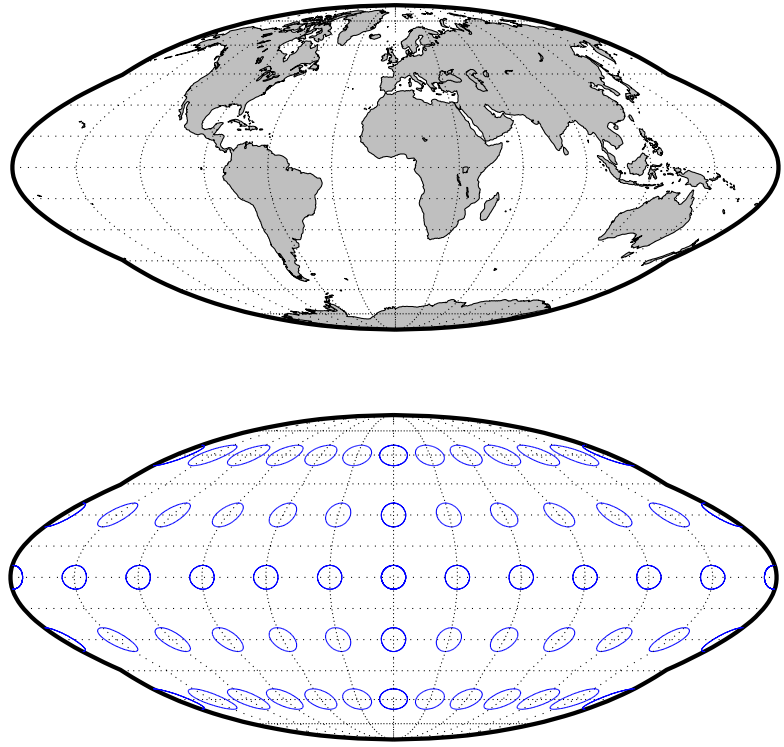
Symmetry: About the central meridian or the Equator.

Features This is an equal-area projection. Scale is true along all parallels and the central meridian between 40°44'11.8" N and S, and is constant along any parallel and between any pair of parallels equidistant from the Equator for all latitudes. Its distortion is identical to that of the Sinusoidal projection between 40°44'11.8" N and S, and to that of the Mollweide projection elsewhere. This projection is not conformal or equidistant.

Parallels This projection has one standard parallel, which is by definition fixed at 0°.

Remarks This projection was developed by J. Paul Goode in 1916. It is sometimes called simply the Homolosine projection, and it is usually used in an interrupted form. It is a merging of the Sinusoidal and Mollweide projections.

Limitations This projection is available in an uninterrupted form only.



Hammer Projection

Classification Modified Azimuthal

Syntax `hammer`

Graticule Meridians: Central meridian is a straight line half the length of the Equator. Other meridians are complex curves, equally spaced along the Equator, and concave towards the central meridian.

Parallels: Equator is straight. Other parallels are complex curves, equally spaced along the central meridian, and concave towards the nearest pole.

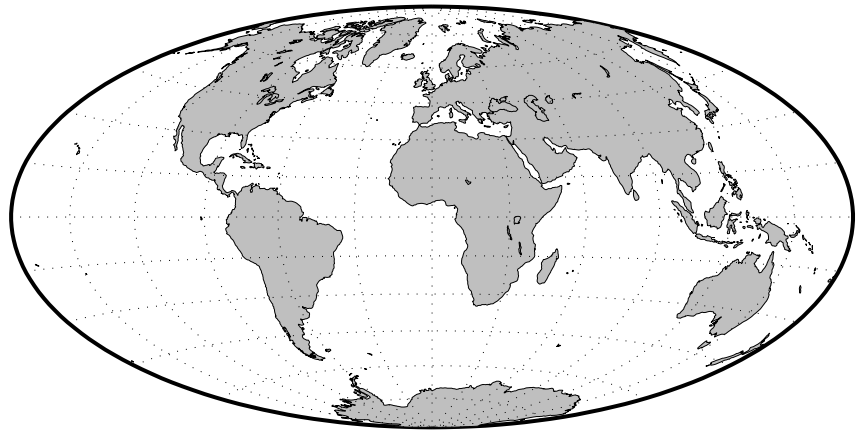
Poles: Points.

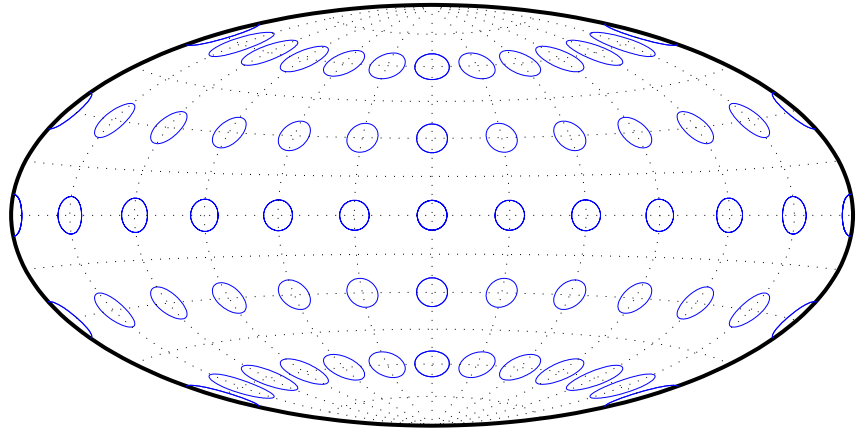
Symmetry: About the Equator and central meridian.

Features This projection is equal-area. The only point free of distortion is the center point. Distortion of shape is moderate throughout. This projection has less angular distortion on the outer meridians near the poles than pseudoazimuthal projections

Parallels There is no standard parallel for this projection.

Remarks This projection was presented by H. H. Ernst von Hammer in 1892. It is a modification of the Lambert Azimuthal Equal Area projection. Inspired by Aitoff projection, it is also known as the Hammer-Aitoff. It in turn inspired the Briesemeister, a modified oblique Hammer projection. John Bartholomew's Nordic projection is an oblique Hammer centered on 45 degrees north and the Greenwich meridian. The Hammer projection is used in whole-world maps and astronomical maps in galactic coordinates.





Hatano Asymmetrical Equal-Area Projection

Classification Pseudocylindrical

Syntax `hatano`

Graticule Central Meridian: Straight line 0.48 as long as the Equator.

Other Meridians: Equally spaced elliptical arcs concave toward the central meridian. The eccentricity of each ellipse changes at the Equator.

Parallels: Unequally spaced straight parallel lines, perpendicular to the central meridian. Spacing is not symmetrical about the Equator.

Poles: The North Pole is a line two-thirds the length of the Equator; the South Pole is a line three-fourths the length of the Equator.

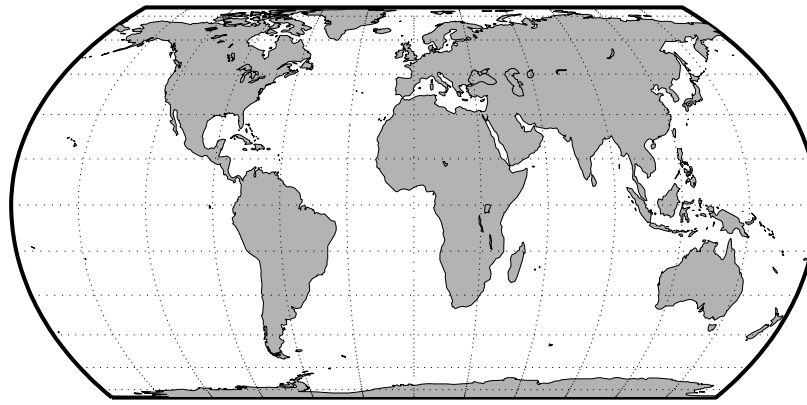
Symmetry: About the central meridian but *not* the Equator.

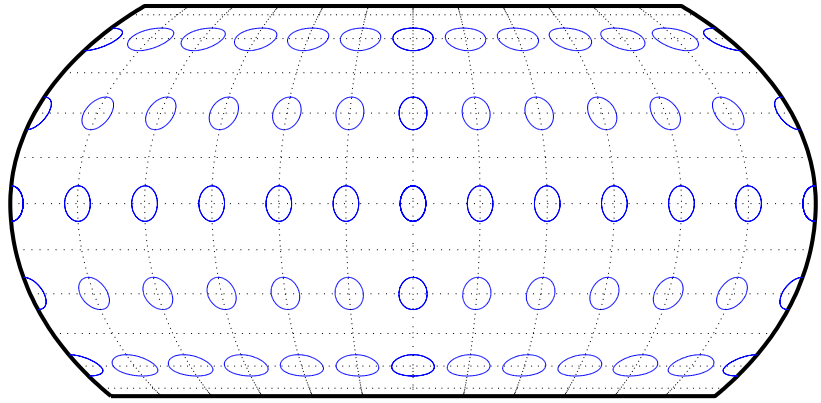
Features This is an equal-area projection. Scale is true along 40°42'N and 38°27'S, and is constant along any parallel but generally *not* between pairs of parallels equidistant from the Equator. It is free of distortion only along the central meridian at 40°42'N and 38°27'S. This projection is not conformal or equidistant.

Parallels Because of the asymmetrical nature of this projection, two standard parallels must be specified. The standard parallels are by definition fixed at 40°42'N and 38°27'S.

Remarks This projection was presented by Masataka Hatano in 1972.

Hatano Asymmetrical Equal-Area Projection





Kavraisky V Projection

Classification Pseudocylindrical

Syntax `kavrsky5`

Graticule Meridians: Complex curves converging at the poles. A sine function is used for *y*, but the meridians are not sine curves.

Parallels: Unequally spaced straight lines.

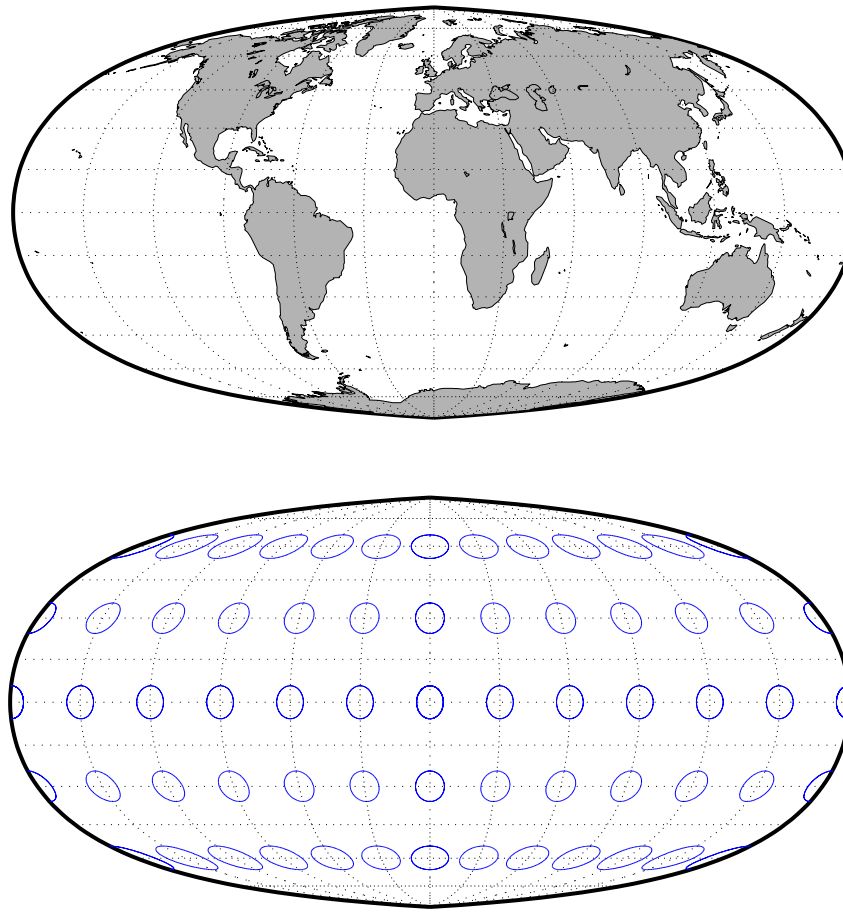
Poles: Points.

Symmetry: About the Equator and the central meridian.

Features This is an equal-area projection. Scale is true along the fixed standard parallels at 35°, and 0.9 true along the Equator. This projection is neither conformal nor equidistant.

Parallels The fixed standard parallels are at 35°.

Remarks This projection was described by V. V. Kavraisky in 1933.



Kavraisky VI Projection

Classification Pseudocylindrical

Syntax `kavrsky6`

Graticule Central Meridian: Straight line half the length of the Equator.

Meridians: Sine curves (60° segments).

Parallels: Unequally spaced straight lines.

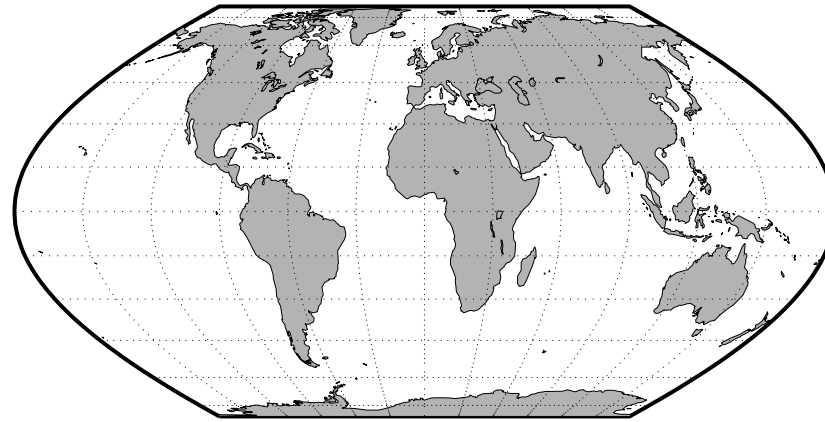
Poles: Straight lines half the length of the Equator.

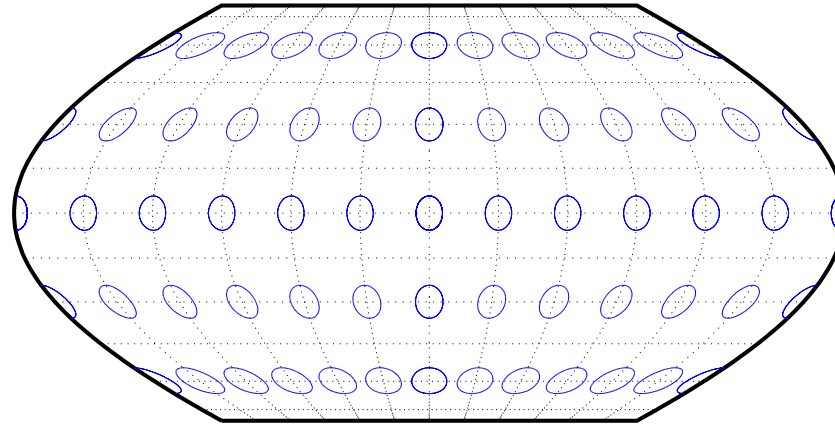
Symmetry: About the Equator and the central meridian.

Features This is an equal-area projection. Scale is constant along any parallel or pair of equidistant parallels. This projection is neither conformal nor equidistant.

Parallels There are no standard parallels for this projection.

Remarks This projection was described by V. V. Kavraisky in 1936. It is also called the Wagner I, for Karlheinz Wagner, who described it in 1932.





Lambert Azimuthal Equal-Area Projection

Classification Azimuthal

Syntax `eqaazim`

Graticule The graticule described is for a polar aspect.

Meridians: Equally spaced straight lines intersecting at the central pole. The angles displayed are the true angles between meridians.

Parallels: Unequally spaced circles centered on the central pole. The entire Earth can be shown. Spacing decreases away from the central pole.

Pole: The central pole is a point; the other pole is a bounding circle with 1.41 the radius of the Equator.

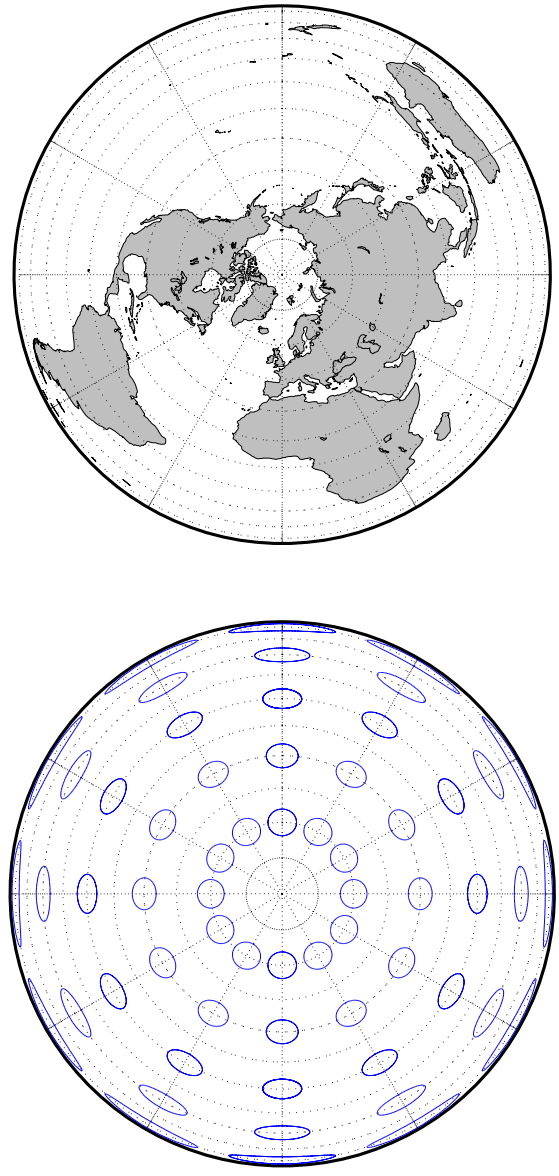
Symmetry: About any meridian.

Features This nonperspective projection is equal-area. Only the center point is free of distortion, but distortion is moderate within 90° of this point. Scale is true only at the center point, increasing tangentially and decreasing radially with distance from the center point. This projection is neither conformal nor equidistant.

Parallels There are no standard parallels for azimuthal projections.

Remarks This projection was presented by Johann Heinrich Lambert in 1772. It is also known as the Zenithal Equal-Area and the Zenithal Equivalent projection, and the Lorgna projection in its polar aspect.

Limitations Data greater than 160° distant from the center point is trimmed.



Lambert Conformal Conic Projection

Classification Conic

Syntax `lambert`

Graticule Meridians: Equally spaced straight lines converging at one of the poles. The angles between the meridians are less than the true angles.

Parallels: Unequally spaced concentric circular arcs centered on the pole of convergence. Spacing of parallels increases away from the central latitudes.

Poles: The pole nearest a standard parallel is a point, the other cannot be shown.

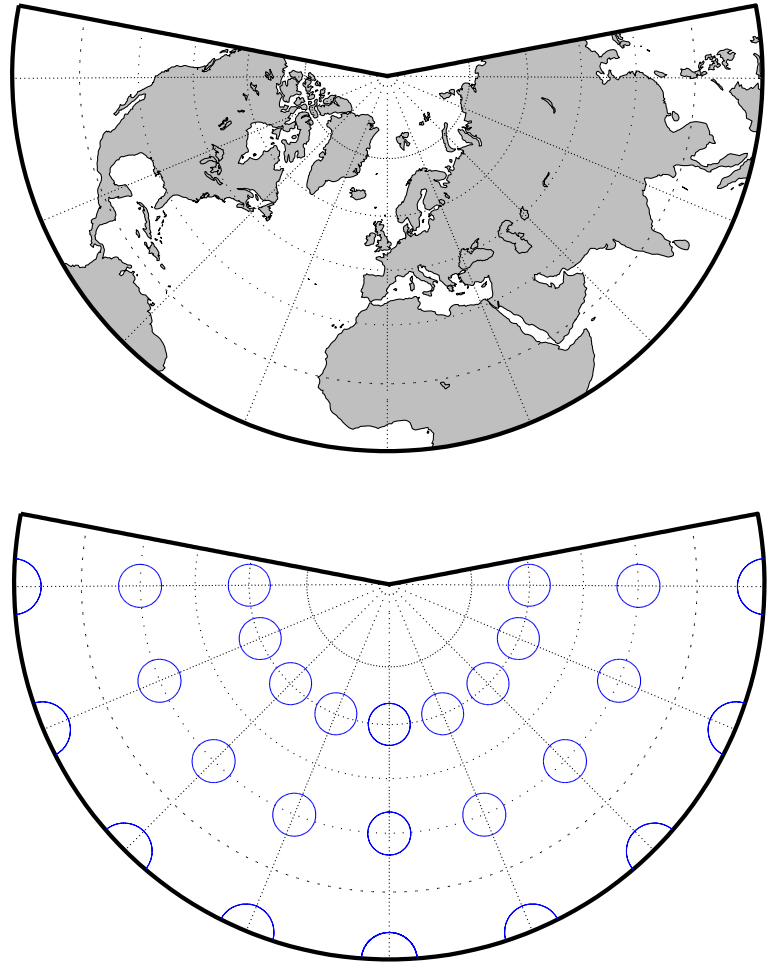
Symmetry: About any meridian.

Features Scale is true along the one or two selected standard parallels. Scale is constant along any parallel and is the same in every direction at any point. This projection is free of distortion along the standard parallels. Distortion is constant along any other parallel. This projection is conformal everywhere but the poles; it is neither equal-area nor equidistant.

Parallels The cone of projection has interesting limiting forms. If a pole is selected as a single standard parallel, the cone is a plane, and a Stereographic Azimuthal projection results. If two parallels are chosen, not symmetric about the Equator, then a Lambert Conformal Conic projection results. If a pole is selected as one of the standard parallels, then the projected pole is a point, otherwise the projected pole is an arc. If the Equator or two parallels equidistant from the Equator are chosen as the standard parallels, the cone becomes a cylinder, and a Mercator projection results. The default parallels are [15 75].

Remarks This projection was presented by Johann Heinrich Lambert in 1772 and is also known as a Conical Orthomorphic projection.

Limitations Longitude data greater than 135° east or west of the central meridian is trimmed. The default map limits are [0 90] to avoid extreme area distortion.



Lambert Equal-Area Cylindrical Projection

Classification Cylindrical

Syntax `lambcyln`

Graticule Meridians: Equally spaced straight parallel lines 0.32 as long as the Equator.

Parallels: Unequally spaced straight parallel lines, perpendicular to the meridians. Spacing is closest near the poles.

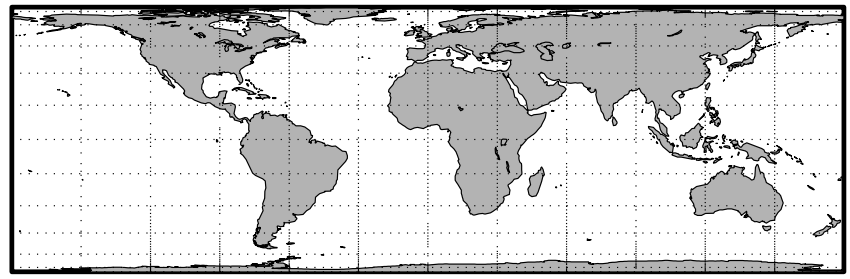
Poles: Straight lines equal in length to the Equator.

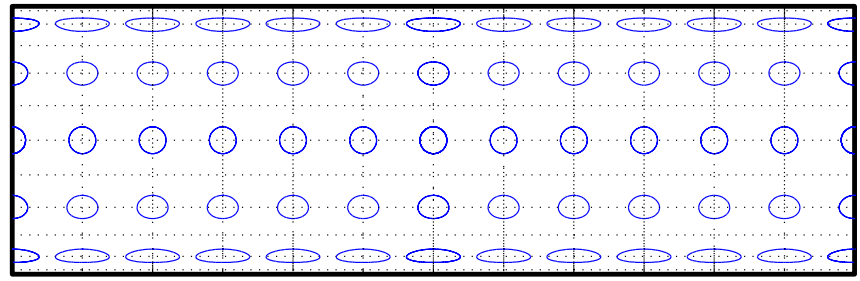
Symmetry: About any meridian or the Equator.

Features This is an orthographic projection onto a cylinder tangent at the Equator. It is equal-area, but distortion of shape increases with distance from the Equator. Scale is true along the Equator and constant between two parallels equidistant from the Equator. This projection is not equidistant.

Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, the standard parallel is by definition fixed at 0°.

Remarks This projection is named for Johann Heinrich Lambert and is a special form of the Equal-Area Cylindrical projection tangent at the Equator.





Loximuthal Projection

Classification Pseudocylindrical

Syntax `loximuth`

Graticule Central Meridian: Straight line at least half as long as the Equator. Actual length depends on the choice of central latitude. Length is 0.5 when the central latitude is the Equator, for example, and 0.65 for central latitudes of 40°.

Other Meridians: Complex curves intersecting at the poles and concave toward the central meridian.

Parallels: Equally spaced straight parallel lines, perpendicular to the central meridian.

Poles: Points.

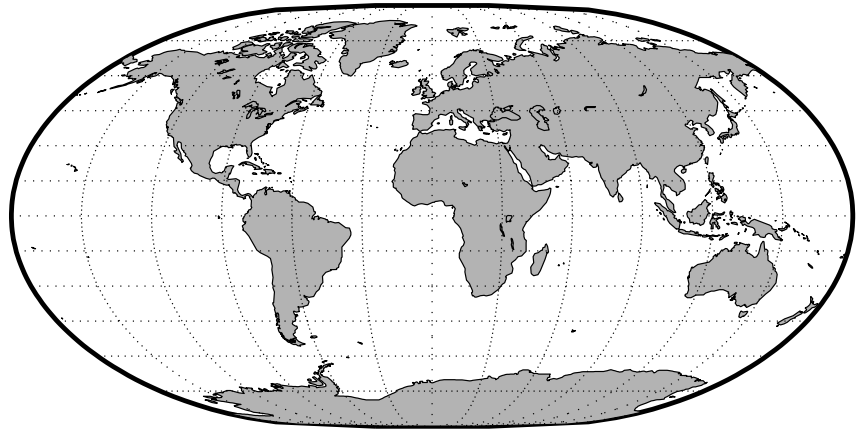
Symmetry: About the central meridian. Symmetry about the Equator only when it is the central latitude.

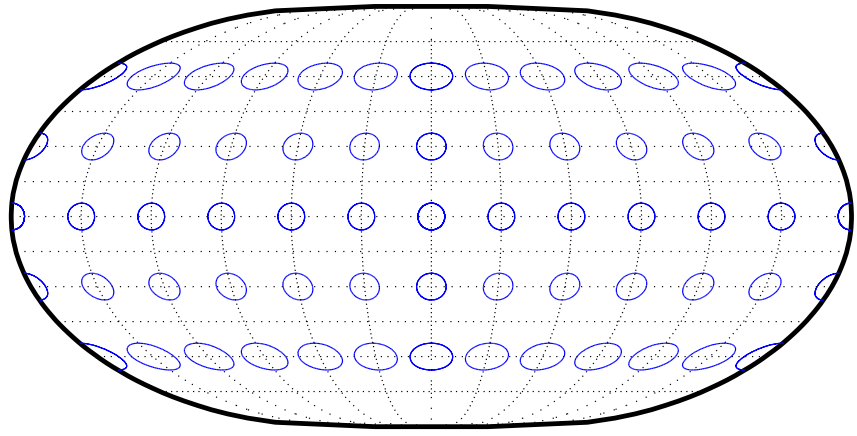
Features This projection has the special property that from the central point (the intersection of the central latitude with the central meridian), rhumb lines (loxodromes) are shown as straight, true to scale, and correct in azimuth from the center. This differs from the Mercator projection, in that rhumb lines are here shown in true scale and that unlike the Mercator, this projection does not maintain true azimuth for all points along the rhumb lines. Scale is true along the central meridian and is constant along any parallel, but not, generally, between parallels. It is free of distortion only at the central point and can be severely distorted in places. However, this projection is designed for its specific special property, in which distortion is not a concern.

Parallels For this projection, only one standard parallel is specified: the central latitude described above. Specification of this central latitude defines the center of the Loximuthal projection. The default value is 0°.

Remarks This projection was presented by Karl Siemon in 1935 and independently by Waldo R. Tobler in 1966. The Bordone Oval projection of 1520 was very similar to the Equator-centered Loximuthal.

Limitations This projection is available for the spherical geoid only.





McBryde-Thomas Flat-Polar Parabolic Projection

Classification Pseudocylindrical

Syntax `flatplrp`

Graticule Central Meridian: Straight line 0.48 as long as the Equator.

Other Meridians: Equally spaced parabolic curves concave toward the central meridian.

Parallels: Unequally spaced straight parallel lines, perpendicular to the central meridian. Spacing is greatest near the Equator.

Poles: Lines one-third as long as the Equator.

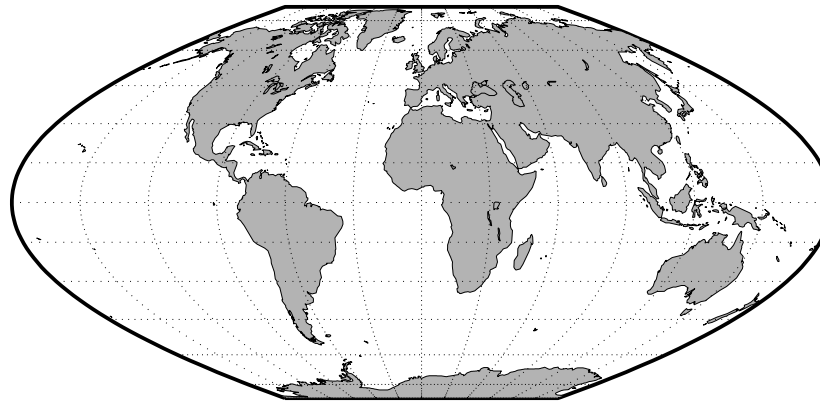
Symmetry: About the central meridian or the Equator.

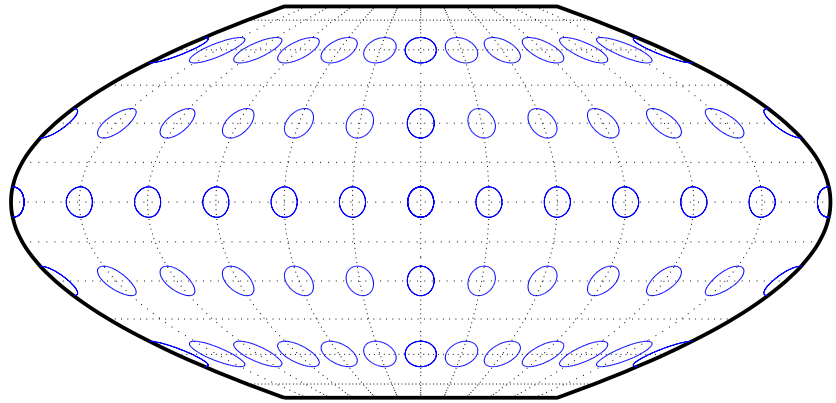
Features This is an equal-area projection. Scale is true along the 45°30' parallels and is constant along any parallel and between any pair of parallels equidistant from the Equator. Distortion is severe near the outer meridians at high latitudes, but less so than on the pointed-polar projections. It is free of distortion only at the two points where the central meridian intersects the 45°30' parallels. This projection is not conformal or equidistant.

Parallels For this projection, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. The standard parallel is by definition fixed at 45°30'.

Remarks This projection was presented by F. Webster McBryde and Paul D. Thomas in 1949.

McBryde-Thomas Flat-Polar Parabolic Projection





McBryde-Thomas Flat-Polar Quartic Projection

Classification Pseudocylindrical

Syntax `flatplrq`

Graticule Central Meridian: Straight line 0.45 as long as the Equator.

Other Meridians: Equally spaced quartic (fourth-order equation) curves concave toward the central meridian.

Parallels: Unequally spaced straight parallel lines, perpendicular to the central meridian. Spacing is greatest near the Equator.

Poles: Lines one-third as long as the Equator.

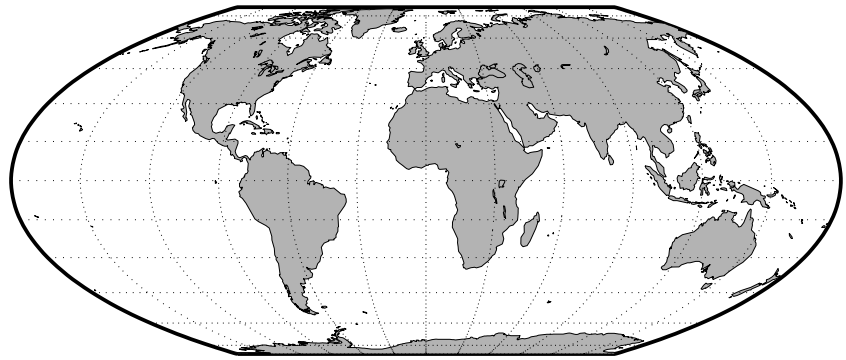
Symmetry: About the central meridian or the Equator.

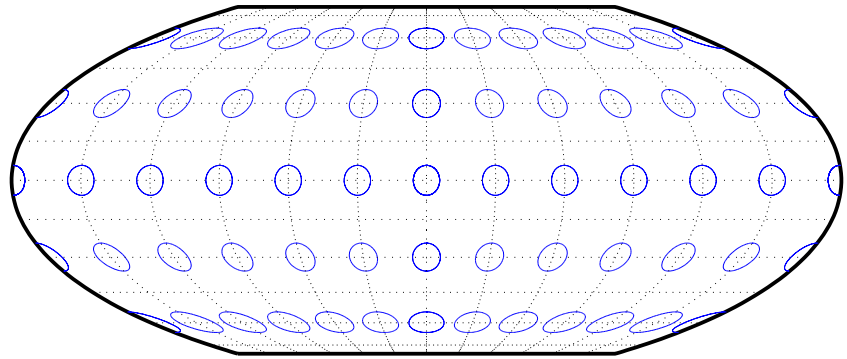
Features This is an equal-area projection. Scale is true along the 33°45' parallels and is constant along any parallel and between any pair of parallels equidistant from the Equator. Distortion is severe near the outer meridians at high latitudes, but less so than on the pointed-polar projections. It is free of distortion only at the two points where the central meridian intersects the 33°45' parallels. This projection is not conformal or equidistant.

Parallels For this projection, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. The standard parallel is by definition fixed at 33°45'.

Remarks This projection was presented by F. Webster McBryde and Paul D. Thomas in 1949, and is also known simply as the Flat-Polar Quartic projection.

McBryde-Thomas Flat-Polar Quartic Projection





McBryde-Thomas Flat-Polar Sinusoidal Projection

Classification Pseudocylindrical

Syntax `flatplrs`

Graticule Central Meridian: Straight line half as long as the Equator.

Other Meridians: Equally spaced sinusoidal curves intersecting at the poles and concave toward the central meridian.

Parallels: Unequally spaced straight parallel lines, perpendicular to the central meridian. Spacing is widest near the Equator.

Poles: Lines one-third as long as the Equator.

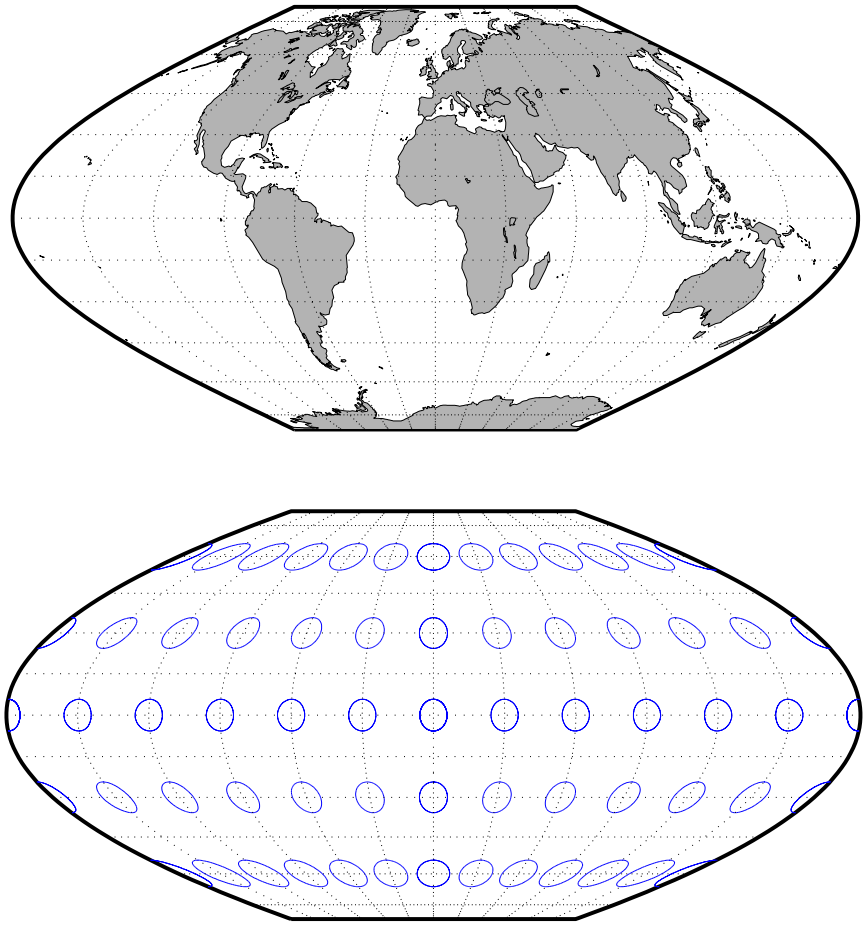
Symmetry: About the central meridian or the Equator.

Features This projection is equal-area. Scale is true along the 55°51' parallels and is constant along any parallel and between any pair of parallels equidistant from the Equator. It is free of distortion only at the two points where the central meridian intersects the 55°51' parallels. This projection is not conformal or equidistant.

Parallels For this projection, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. The standard parallel is by definition fixed at 55°51'.

Remarks This projection was presented by F. Webster McBryde and Paul D. Thomas in 1949.

McBryde-Thomas Flat-Polar Sinusoidal Projection



Mercator Projection

Classification Cylindrical

Syntax `mercator`

Graticule Meridians: Equally spaced straight parallel lines.

Parallels: Unequally spaced straight parallel lines, perpendicular to the meridians. Spacing increases toward the poles.

Poles: Cannot be shown.

Symmetry: About any meridian or the Equator.

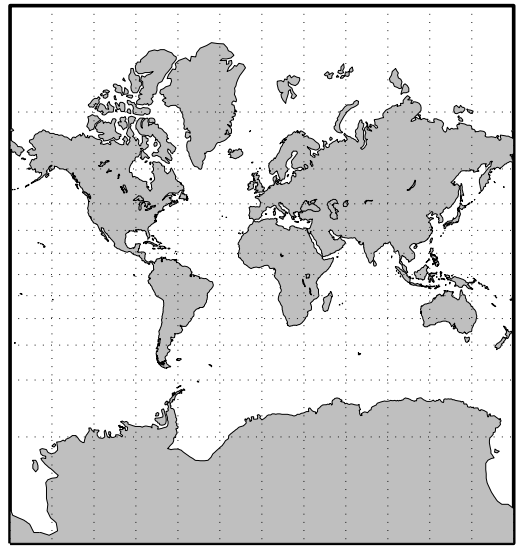
Features This is a projection with parallel spacing calculated to maintain conformality. It is not equal-area, equidistant, or perspective. Scale is true along the standard parallels and constant between two parallels equidistant from the Equator. It is also constant in all directions near any given point. Scale becomes infinite at the poles. The appearance of the Mercator projection is unaffected by the selection of standard parallels; they serve only to define the latitude of true scale.

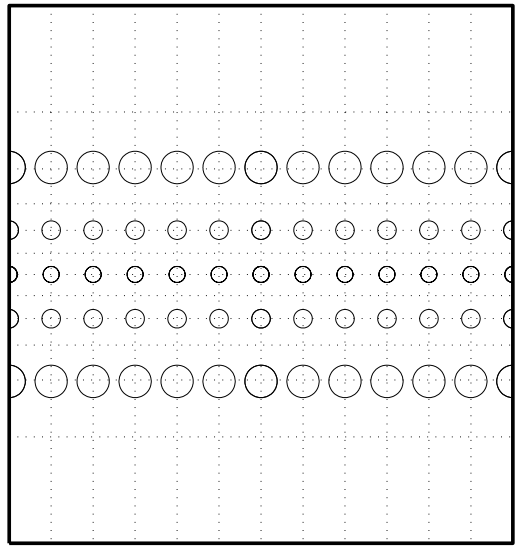
The Mercator, which may be the most famous of all projections, has the special feature that all rhumb lines, or loxodromes (lines that make equal angles with all meridians, i.e., lines of constant heading), are straight lines. This makes it an excellent projection for navigational purposes. However, the extreme area distortion makes it unsuitable for general maps of large areas.

Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, any latitude less than 86° may be chosen; the default is arbitrarily set to 0°.

Remarks The Mercator projection is named for Gerardus Mercator, who presented it *for navigation* in 1569. It is now known to have been used for the Tunhuang star chart as early as 940 by Ch'ien Lo-Chih. It was first used in Europe by Erhard Etzlaub in 1511. It is also, but rarely, called the Wright projection, after Edward Wright, who developed the mathematics behind the projection in 1599.

Limitations Data at latitudes greater than 86° is trimmed to prevent large *y*-values from dominating the display.





Miller Cylindrical Projection

Classification Cylindrical

Syntax `miller`

Graticule Meridians: Equally spaced straight parallel lines 0.73 as long as the Equator.

Parallels: Unequally spaced straight parallel lines, perpendicular to the meridians. Spacing increases toward the poles, less rapidly than that of the Mercator projection.

Poles: Straight lines equal in length to the Equator.

Symmetry: About any meridian or the Equator.

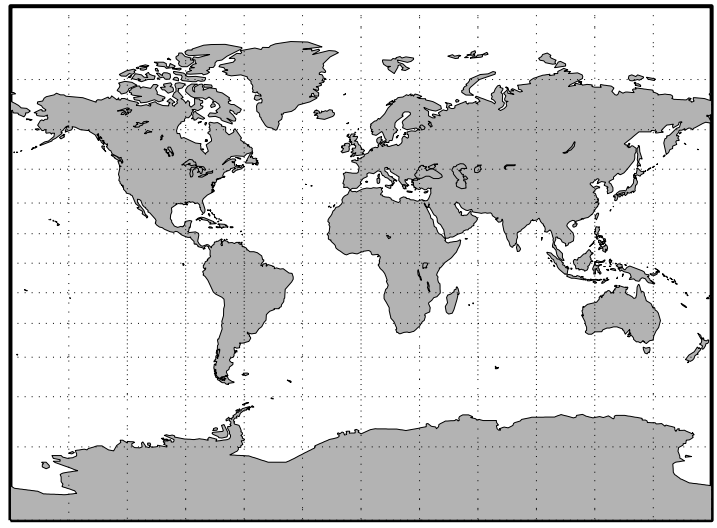
Features This is a projection with parallel spacing calculated to maintain a look similar to the Mercator projection while reducing the distortion near the poles and allowing the poles to be displayed. It is not equal-area, equidistant, conformal, or perspective. Scale is true along the Equator and constant between two parallels equidistant from the Equator. There is no distortion near the Equator, and it increases moderately away from the Equator, but it becomes severe at the poles.

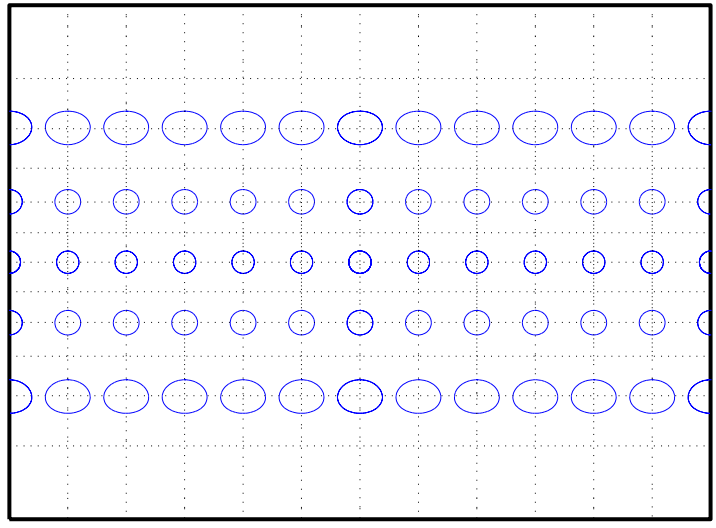
The Miller Cylindrical projection is derived from the Mercator projection; parallels are spaced from the Equator by calculating the distance on the Mercator for a parallel at 80% of the true latitude and dividing the result by 0.8. The result is that the two projections are almost identical near the Equator.

Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, the standard parallel is by definition fixed at 0°.

Remarks This projection was presented by Osborn Maitland Miller of the American Geographical Society in 1942. It is often used in place of the Mercator projection for atlas maps of the world, for which it is somewhat more appropriate.

Limitations This projection is available for the spherical geoid only.





Mollweide Projection

Classification Pseudocylindrical

Syntax `mollweid`

Graticule Central Meridian: Straight line half as long as the Equator.

Other Meridians: Meridians 90° east and west of the central meridian form a circle. The others are equally spaced semiellipses intersecting at the poles and concave toward the central meridian.

Parallels: Unequally spaced straight parallel lines, perpendicular to the central meridian. Spacing is greatest toward the Equator, but the spacing changes gradually.

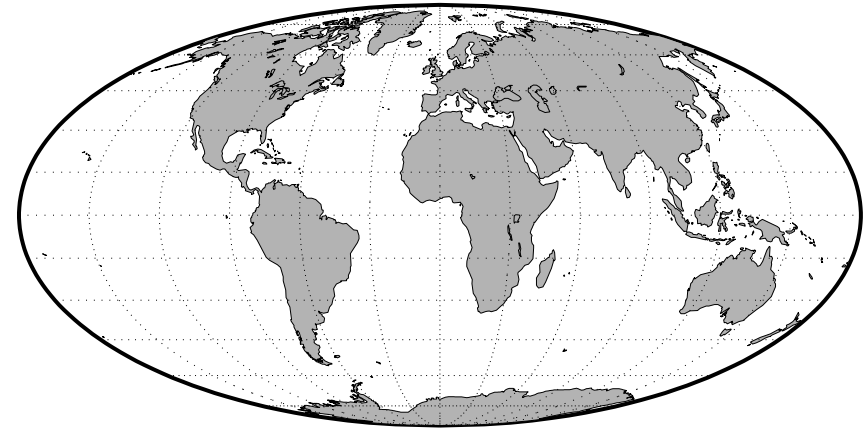
Poles: Points.

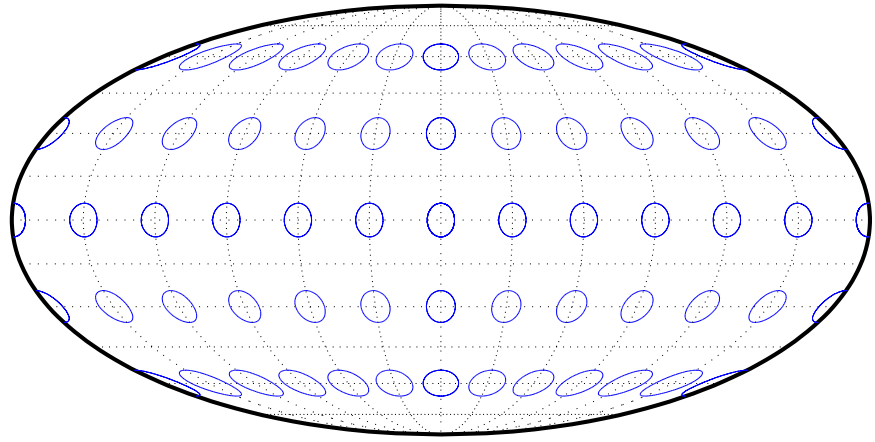
Symmetry: About the central meridian or the Equator.

Features This is an equal-area projection. Scale is true along the 40°44' parallels and is constant along any parallel and between any pair of parallels equidistant from the Equator. It is free of distortion only at the two points where the 40°44' parallels intersect the central meridian. This projection is not conformal or equidistant.

Parallels For this projection, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. The standard parallel is by definition fixed at 40°44'.

Remarks This projection was presented by Carl B. Mollweide in 1805. Its other names include the Homolographic, the Homalographic, the Babinet, and the Elliptical projections. It is occasionally used for thematic world maps, and it is combined with the Sinusoidal to produce the Goode Homolosine projection.





Murdoch I Conic Projection

Classification Conic

Syntax `murdoch1`

Graticule Meridians: Equally spaced straight lines converging at one of the poles.

Parallels: Equally spaced concentric circular arcs.

Poles: Arcs, one of which might become a point in the limit.

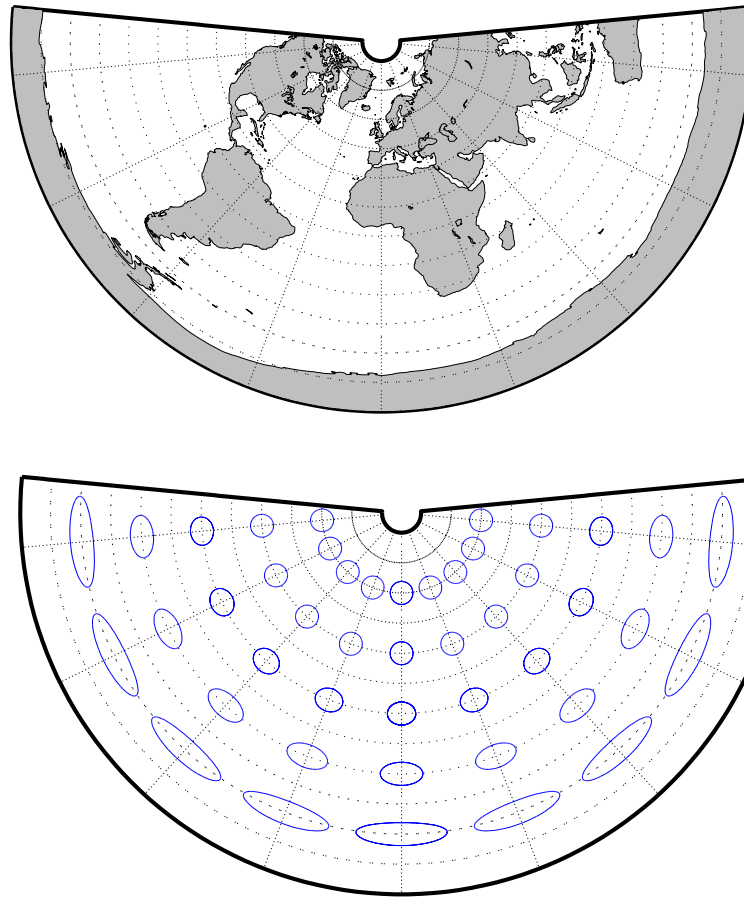
Symmetry: About any meridian.

Features This is an equidistant projection that is nearly minimum-error. Scale is true along any meridian and is constant along any parallel. Scale is also true along two standard parallels. These must be calculated, however (see below). The total area of the mapped area is correct, but it is not equal-area everywhere.

Parallels The parallels for this projection are not standard parallels, but rather limiting parallels. The special feature of this map, correct total area, holds between these parallels. The default parallels are [15 75].

Remarks Described by Patrick Murdoch in 1758.

Limitations This projection is available for the spherical geoid only. Longitude data greater than 135° east or west of the central meridian is trimmed.



Murdoch III Minimum Error Conic Projection

Classification Conic

Syntax `murdoch3`

Graticule Meridians: Equally spaced straight lines converging at one of the poles.

Parallels: Equally spaced concentric circular arcs.

Poles: Arcs, one of which might become a point in the limit.

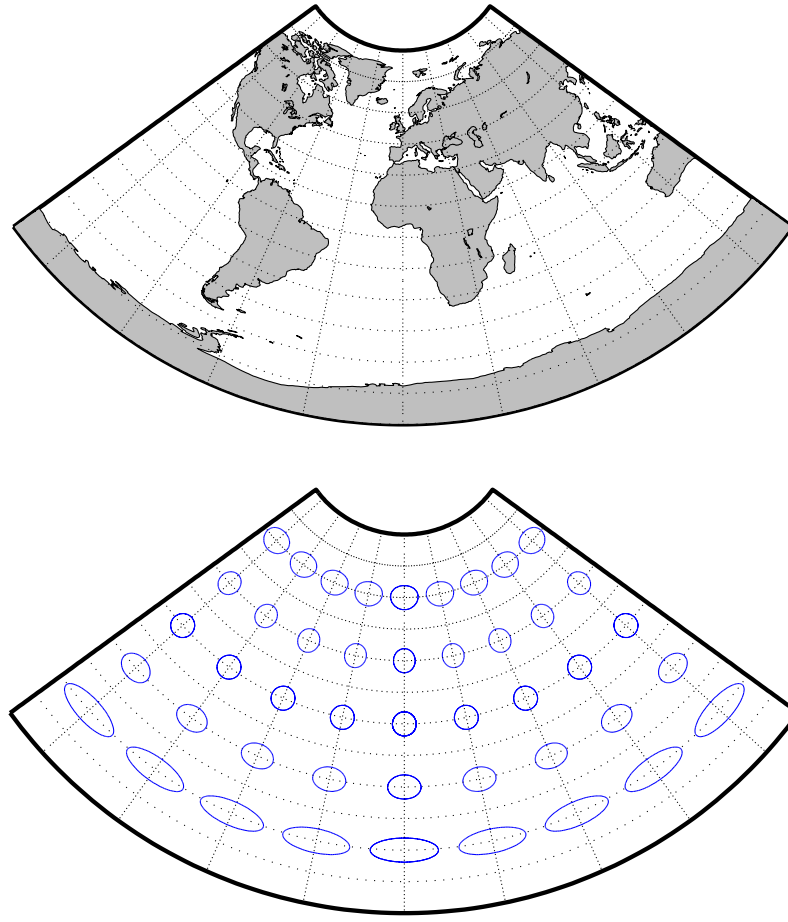
Symmetry: About any meridian.

Features This is an equidistant projection that is minimum-error. Scale is true along any meridian and is constant along any parallel. Scale is also true along two standard parallels. These must be calculated, however (see below). The total area of the mapped area is correct, but it is not equal-area everywhere.

Parallels The parallels for this projection are not standard parallels, but rather limiting parallels. The special feature of this map, correct total area, holds between these parallels. The default parallels are [15 75].

Remarks Described by Patrick Murdoch in 1758, with errors corrected by Everett in 1904.

Limitations This projection is available for the spherical geoid only. Longitude data greater than 135° east or west of the central meridian is trimmed.



Orthographic Projection

Classification Azimuthal

Syntax `ortho`

Graticule The graticule described is for a polar aspect.

Meridians: Equally spaced straight lines intersecting at the central pole. The angles displayed are the true angles between meridians.

Parallels: Unequally spaced circles centered on the central pole. Spacing decreases away from this pole. The opposite hemisphere cannot be shown.

Pole: The central pole is a point; the other pole is not shown.

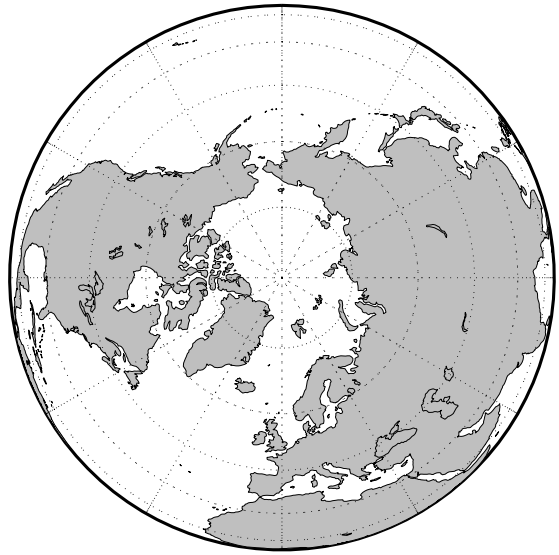
Symmetry: About any meridian.

Features This is a perspective projection on a plane tangent at the center point from an infinite distance (that is, orthogonally). The center point is a pole in the common polar aspect, but can be any point. This projection has two significant properties. It looks like a globe, providing views of the Earth resembling those seen from outer space. Additionally, all great and small circles are either straight lines or elliptical arcs on this projection. Scale is true only at the center point and is constant in the circumferential direction along any circle having the center point as its center. Distortion increases rapidly away from the center point, the only place that is distortion-free. This projection is neither conformal nor equal-area.

Parallels There are no standard parallels for azimuthal projections.

Remarks This projection appears to have been developed by the Egyptians and Greeks by the second century B.C.

Limitations This projection is available for the spherical geoid only. Data greater than 89° distant from the center point is trimmed.



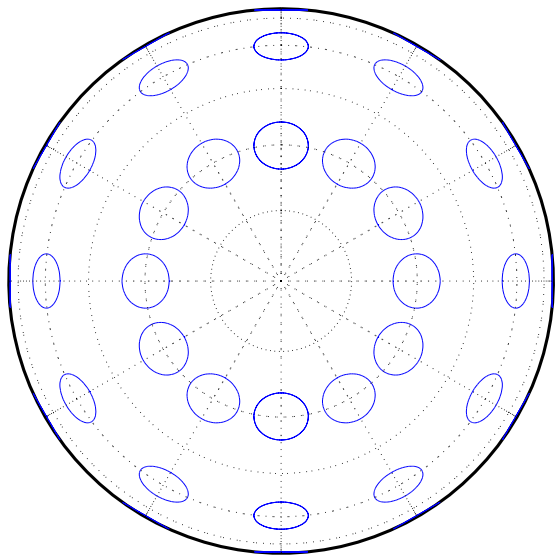


Plate Carrée Projection

Classification Cylindrical

Syntax `pcarree`

Graticule Meridians: Equally spaced straight parallel lines half as long as the Equator.

Parallels: Equally spaced straight parallel lines, perpendicular to and having the same spacing as the meridians.

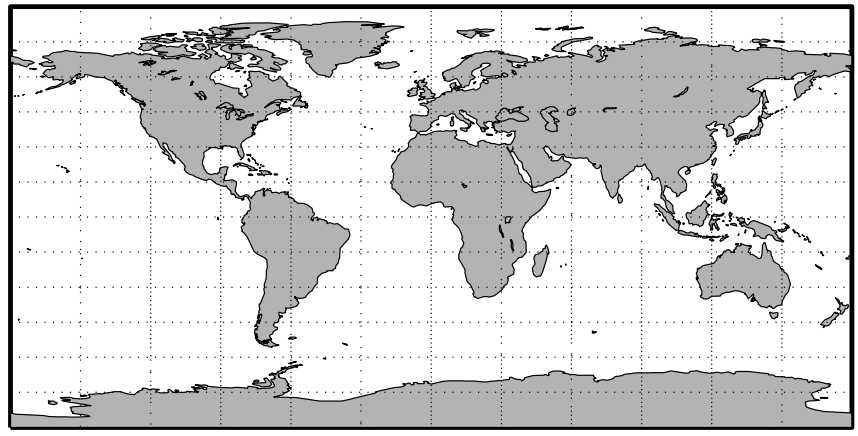
Poles: Straight lines equal in length to the Equator.

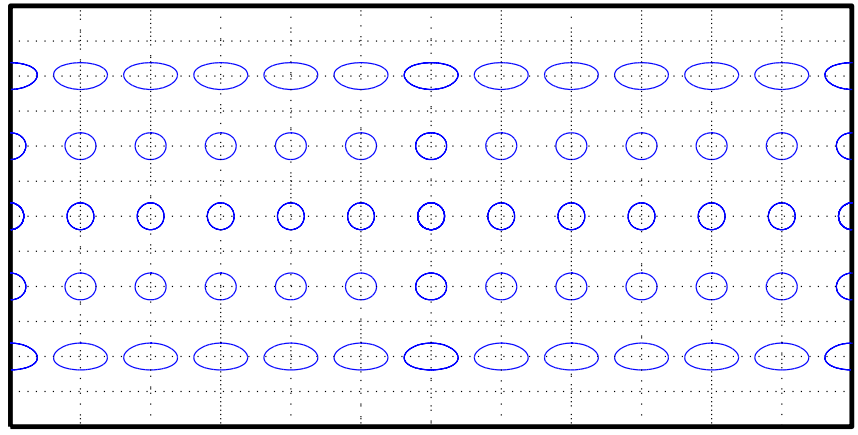
Symmetry: About any meridian or the Equator.

Features This is a projection onto a cylinder tangent at the Equator. Distortion of both shape and area increases with distance from the Equator. Scale is true along all meridians (i.e., it is equidistant) and the Equator and is constant along any parallel and along the parallel of opposite sign.

Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, the standard parallel is by definition fixed at 0°.

Remarks This projection, like the more general Equidistant Cylindrical, is credited to Marinus of Tyre, thought to have invented it about A.D. 100. It may, in fact, have been originated by Erastosthenes, who lived approximately 275-195 B.C. The Plate Carrée has the most simply constructed graticule of any projection. It was used frequently in the 15th and 16th centuries and is quite common today in very simple computer mapping programs. It is the simplest and limiting form of the Equidistant Cylindrical projection. Another name for this projection is the Simple Cylindrical. Its transverse aspect is the Cassini projection.





Polyconic Projection

Classification Polyconic

Syntax `polycon`

Graticule Central Meridian: A straight line.

Meridians: Complex curves spaced equally along the Equator and each parallel, and concave toward the central meridian.

Parallels: The Equator is a straight line. All other parallels are nonconcentric circular arcs spaced at true distances along the central meridian.

Poles: Normally circular arcs, enclosing the same angle as the displayed parallels.

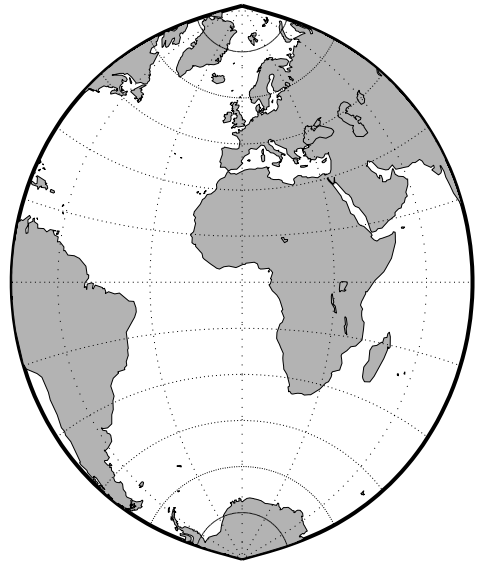
Symmetry: About the Equator or the central meridian.

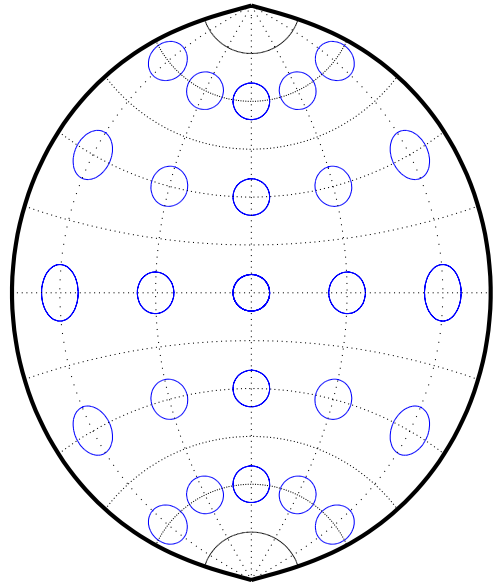
Features For this projection, each parallel has a curvature identical to its curvature on a cone tangent at that latitude. Since each parallel has its own cone, this is a "polyconic" projection. Scale is true along the central meridian and along each parallel. This projection is free of distortion only along the central meridian; distortion can be severe at extreme longitudes. This projection is neither conformal nor equal-area.

Parallels By definition, this projection has no standard parallels, since every parallel is a *standard parallel*.

Remarks This projection was apparently originated about 1820 by Ferdinand Rudolph Hassler. It is also known as the American Polyconic and the Ordinary Polyconic projection.

Limitations Longitude data greater than 75° east or west of the central meridian is trimmed.





Putnins P5 Projection

Classification Pseudocylindrical

Syntax `putnins5`

Graticule Central Meridian: Straight line half as long as the Equator.

Other Meridians: Equally spaced portions of hyperbolas intersecting at the poles and concave toward the central meridian.

Parallels: Equally spaced straight parallel lines, perpendicular to the central meridian.

Poles: Points.

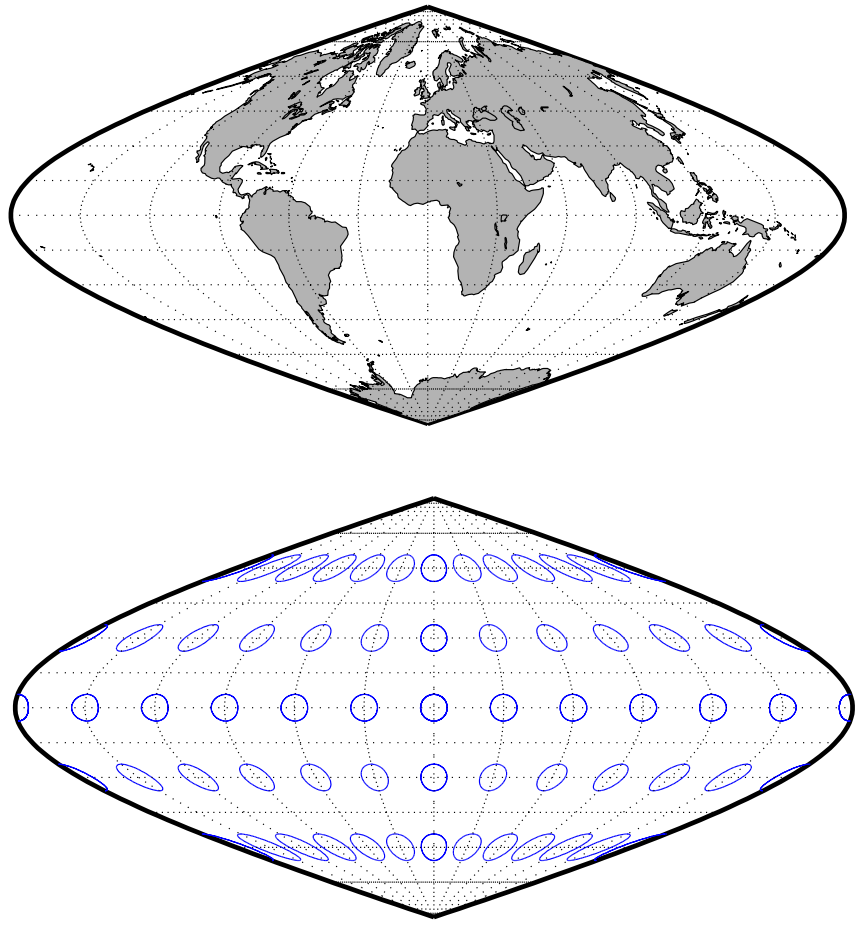
Symmetry: About the central meridian or the Equator.

Features Scale is true along the 21°14' parallels and is constant along any parallel, between any pair of parallels equidistant from the Equator, and along the central meridian. It is not free of distortion at any point. This projection is not equal-area, conformal, or equidistant.

Parallels For this projection, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. The standard parallel is by definition fixed at 21°14'.

Remarks This projection was presented by Reinholds V. Putnins in 1934.

Limitations This projection is available for the spherical geoid only.



Quartic Authalic Projection

Classification Pseudocylindrical

Syntax `quartic`

Graticule Central Meridian: Straight line 0.45 as long as the Equator.

Other Meridians: Equally spaced quartic (fourth-order equation) curves concave toward the central meridian.

Parallels: Unequally spaced straight parallel lines, perpendicular to the central meridian. Spacing changes gradually and is greatest near the Equator.

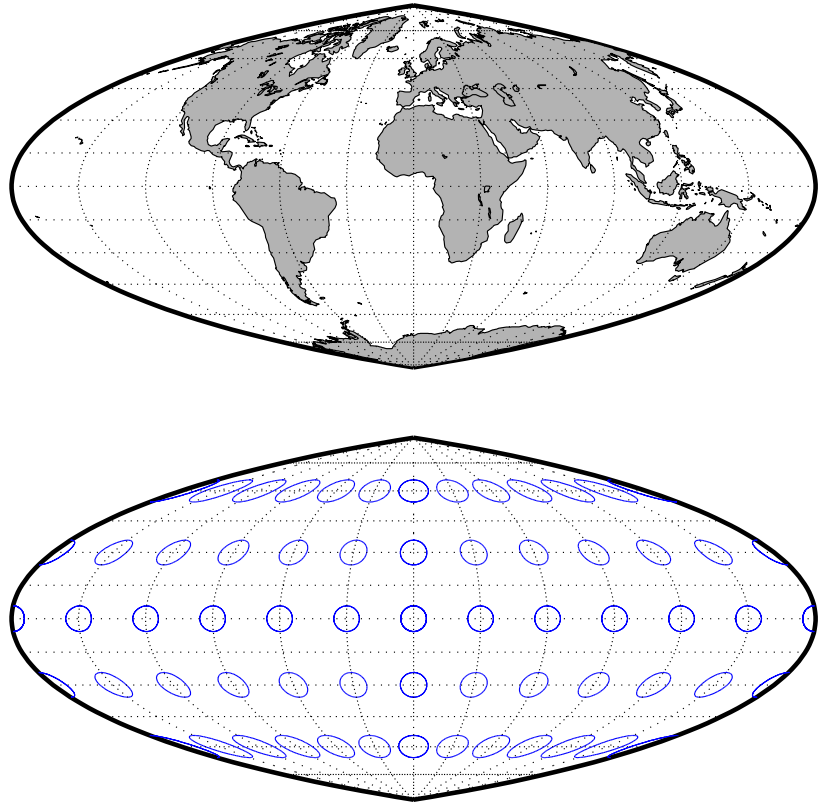
Poles: Points.

Symmetry: About the central meridian or the Equator.

Features This is an equal-area projection. Scale is true along the Equator and is constant along any parallel and between any pair of parallels equidistant from the Equator. Distortion is severe near the outer meridians at high latitudes, but less so than on the Sinusoidal projection. It is free of distortion along the Equator. This projection is not conformal or equidistant.

Parallels This projection has one standard parallel, which is by definition fixed at 0°.

Remarks This projection was presented by Karl Siemon in 1937 and independently by Oscar Sherman Adams in 1945.



Robinson Projection

Classification Pseudocylindrical

Syntax `robinson`

Graticule Central Meridian: Straight line 0.51 as long as the Equator.

Other Meridians: Equally spaced, resemble elliptical arcs and are concave toward the central meridian.

Parallels: Straight parallel lines, perpendicular to the central meridian. Spacing is equal between the 38° parallels, decreasing outside these limits.

Poles: Lines 0.53 as long as the Equator.

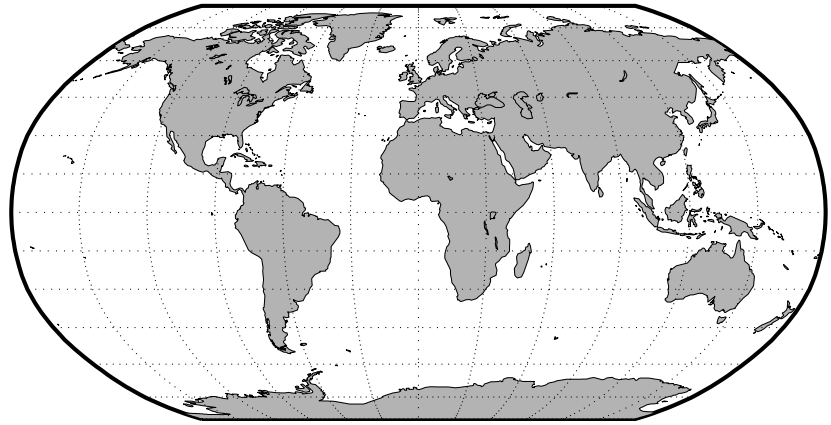
Symmetry: About the central meridian or the Equator.

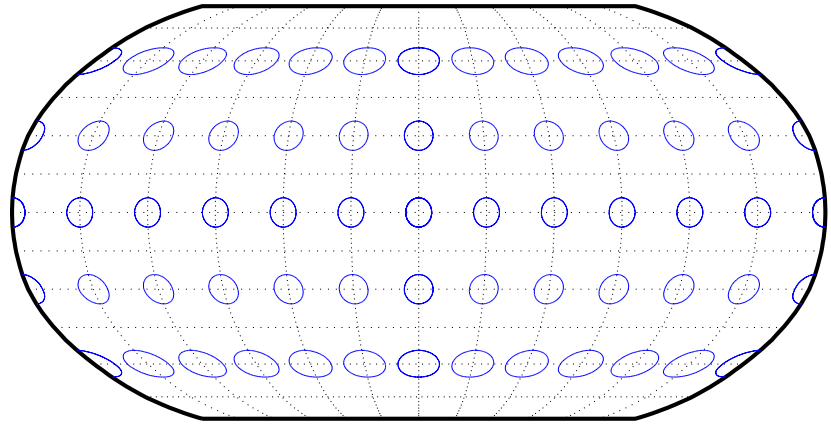
Features Scale is true along the 38° parallels and is constant along any parallel or between any pair of parallels equidistant from the Equator. It is not free of distortion at any point, but distortion is very low within about 45° of the center and along the Equator. This projection is not equal-area, conformal, or equidistant; however, it is considered to *look right* for world maps, and hence is widely used by Rand McNally, the National Geographic Society, and others. This feature is achieved through the use of tabular coordinates rather than mathematical formulae for the graticules.

Parallels For this projection, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. The standard parallel is by definition fixed at 38°.

Remarks This projection was presented by Arthur H. Robinson in 1963, and is also called the Orthophanic projection, which means *right appearing.*

Limitations This projection is available for the spherical geoid only.





Sinusoidal Projection

Classification Pseudocylindrical

Syntax `sinusoid`

Graticule Central Meridian: Straight line half as long as the Equator.

Other Meridians: Equally spaced sinusoidal curves intersecting at the poles and concave toward the central meridian.

Parallels: Equally spaced straight parallel lines, perpendicular to the central meridian.

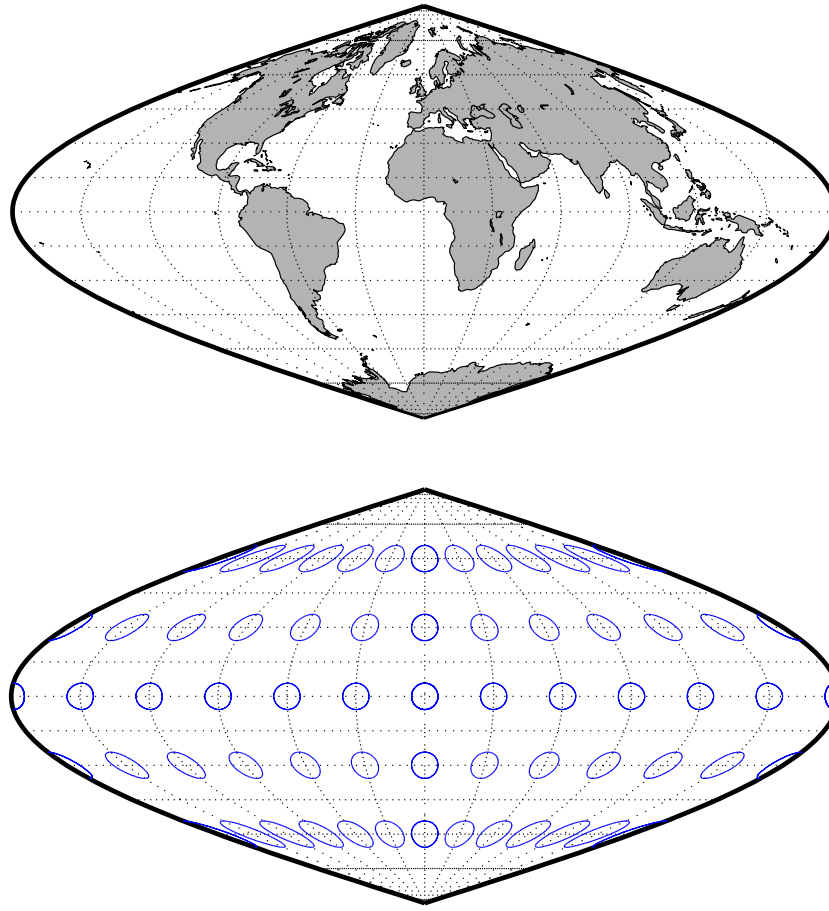
Poles: Points.

Symmetry: About the central meridian or the Equator.

Features This projection is equal-area. Scale is true along every parallel and along the central meridian. There is no distortion along the Equator or along the central meridian, but it becomes severe near the outer meridians at high latitudes.

Parallels This projection has one standard parallel, which is by definition fixed at 0°.

Remarks This projection was developed in the 16th century. It was used by Jean Cossin in 1570 and by Jodocus Hondius in Mercator atlases of the early 17th century. It is the oldest pseudocylindrical projection currently in use, and is sometimes called the Sanson-Flamsteed or the Mercator Equal-Area projection.



Stereographic Projection

Classification Azimuthal

Syntax `stereo`

Graticule The graticule described is for a polar aspect.

Meridians: Equally spaced straight lines intersecting at the central pole. The angles displayed are the true angles between meridians.

Parallels: Unequally spaced circles centered on the central pole. Spacing increases gradually away from this pole. The opposite hemisphere cannot be shown

Pole: The central pole is a point; the other pole is not shown.

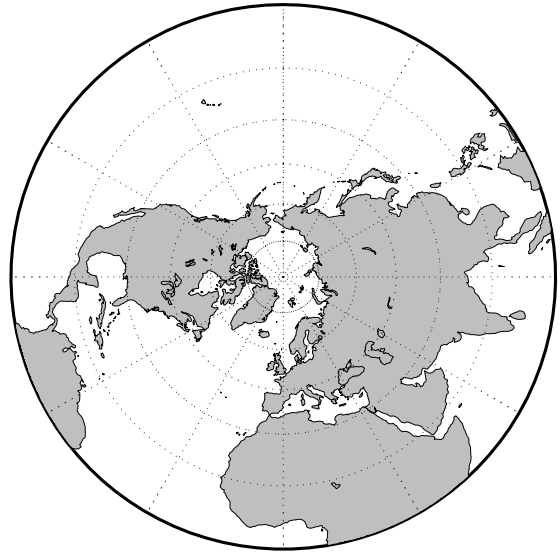
Symmetry: About any meridian.

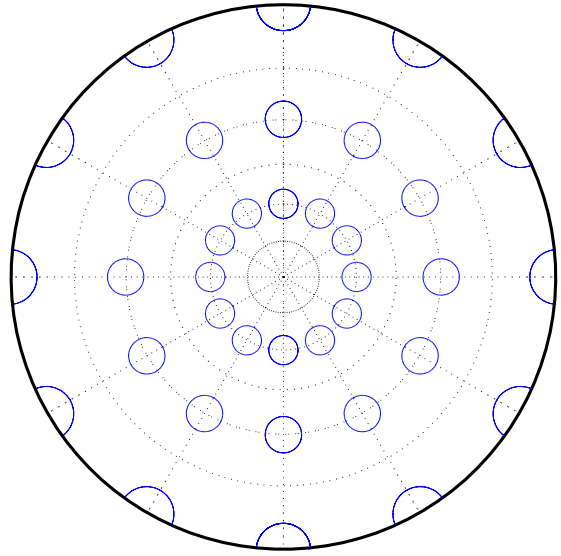
Features This is a perspective projection on a plane tangent at the center point from the point antipodal to the center point. The center point is a pole in the common polar aspect, but can be any point. This projection has two significant properties. It is conformal, being free from angular distortion. Additionally, all great and small circles are either straight lines or circular arcs on this projection. Scale is true only at the center point and is constant along any circle having the center point as its center. This projection is not equal-area.

Parallels There are no standard parallels for azimuthal projections.

Remarks The polar aspect of this projection appears to have been developed by the Egyptians and Greeks by the second century B.C.

Limitations Data greater than 90° distant from the center point is trimmed.





Tissot Modified Sinusoidal Projection

Classification Pseudocylindrical

Syntax `modsine`

Graticule Meridians: Sine curves converging at the Poles.

Parallels: Equally spaced straight lines.

Poles: Points.

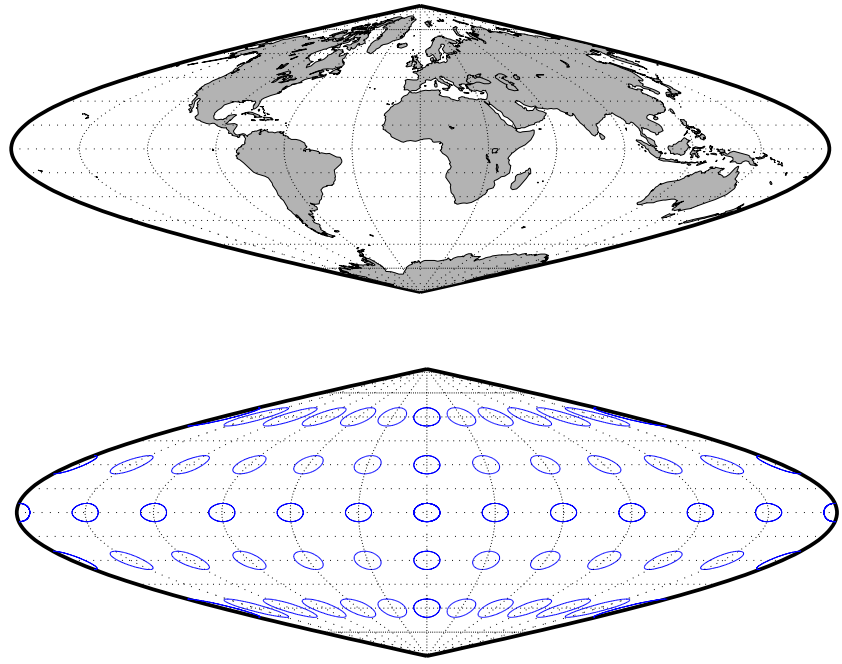
Symmetry: About the Equator and the central meridian

Features This is an equal-area projection. Scale is constant along any parallel or any pair of equidistant parallels, and along the central meridian. It is not equidistant or conformal.

Parallels There are no standard parallels for this projection

Remarks This projection was first described by N. A. Tissot in 1881

Limitations This projection is available for the spherical geoid only.



Trystan Edwards Cylindrical Projection

Classification Cylindrical

Syntax `trystan`

Graticule Meridians: Equally spaced straight parallel lines.

Parallels: Unequally spaced straight parallel lines, perpendicular to the meridians. Spacing is closest near the poles.

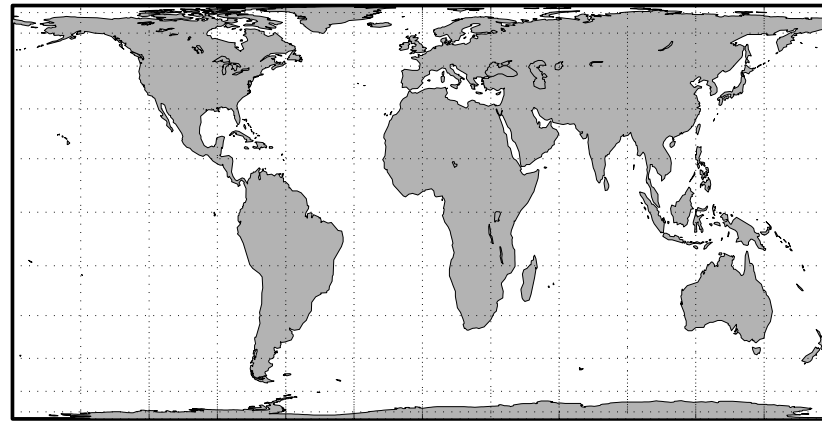
Poles: Straight lines equal in length to the Equator.

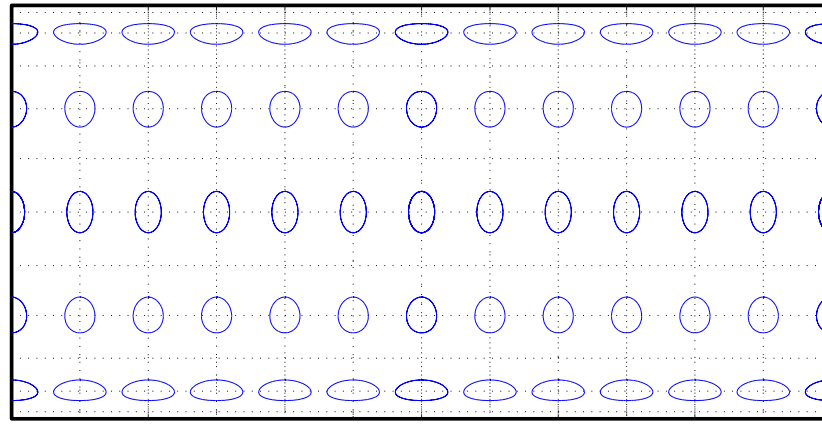
Symmetry: About any meridian or the Equator.

Features This is an orthographic projection onto a cylinder secant at the 37°24' parallels. It is equal-area, but distortion of shape increases with distance from the standard parallels. Scale is true along the standard parallels and constant between two parallels equidistant from the Equator. This projection is not equidistant.

Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, the standard parallel is by definition fixed at 37°24'.

Remarks This projection is named for Trystan Edwards, who presented it in 1953. It is a special form of the Equal-Area Cylindrical projection secant at 37°24'N and S.





Universal Polar Stereographic Projection

Classification Azimuthal

Syntax ups

Graticule The graticule described is for the southern zone.

Meridians: Equally spaced straight lines centered on the South Pole. The angles displayed are the true angles between meridians.

Parallels: Unequally spaced circles centered on the South Pole. Spacing increases gradually away from the circle of true scale along latitude 87 degrees, 7 minutes N. The opposite pole cannot be shown.

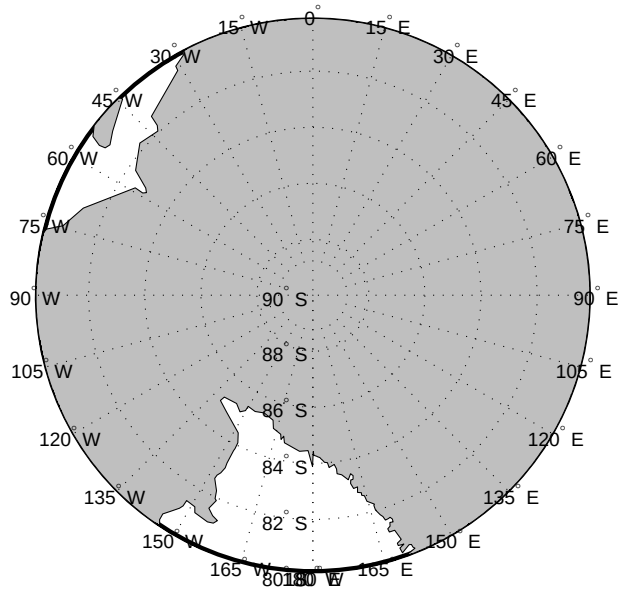
Poles: The South Pole is a point. The North Pole is not shown.

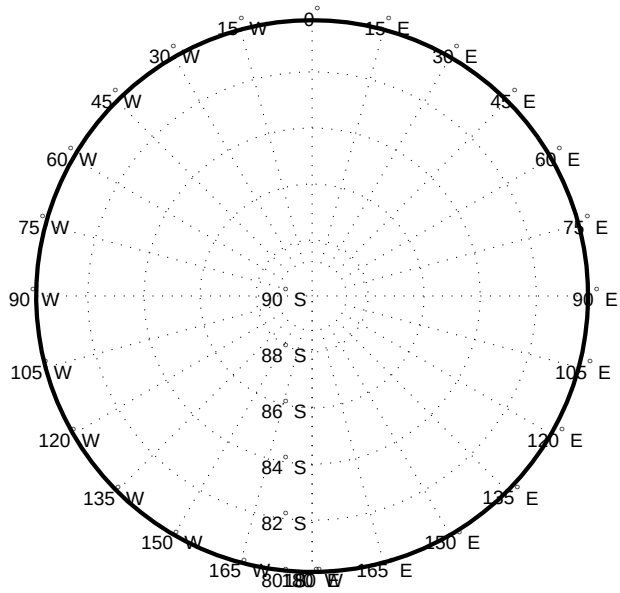
Symmetry: About any meridian.

Features This is a perspective projection on a plane tangent to either the North or South Pole. It is conformal, being free from angular distortion. Additionally, all great and small circles are either straight lines or circular arcs on this projection. Scale is true along latitudes 87 degrees, 7 minutes N or S, and is constant along any other parallel. This projection is not equal area.

Parallels The parallels 87 degrees, 7 minutes N and S are lines of true scale by virtue of the scale factor. There are no standard parallels for azimuthal projections.

Remarks This projection is a special case of the stereographic projection in the polar aspect. It is used as part of the Universal Transverse Mercator (UTM) system to extend coverage to the poles. This projection has two zones: 'North' for latitudes 84° N to 90° N, and 'South' for latitudes 80° S to 90° S. The defaults for this projection are: scale factor is 0.994, false easting and northing are 2,000,000 meters. The international ellipsoid in units of meters is used as the geoid model.


0°
15° W
15° E
30° W
30° E
45° W
45° E
60° W
60° E
75° W
75° E
90° W
90° E
105° W
105° E
120° W
120° E
135° W
135° E
150° W
150° E
165° W
165° E
90° S
88° S
86° S
84° S
82° S


0°
15° W
15° E
30° W
30° E
45° W
45° E
60° W
60° E
75° W
75° E
90° W
90° E
105° W
105° E
120° W
120° E
135° W
135° E
150° W
150° E
165° W
165° E
90° S
88° S
86° S
84° S
82° S

Universal Transverse Mercator Projection

Classification Cylindrical

Syntax `utm`

Graticule Meridians: Complex curves concave towards the central meridian.

Parallels: Complex curves concave towards the nearest pole.

Poles: Not shown.

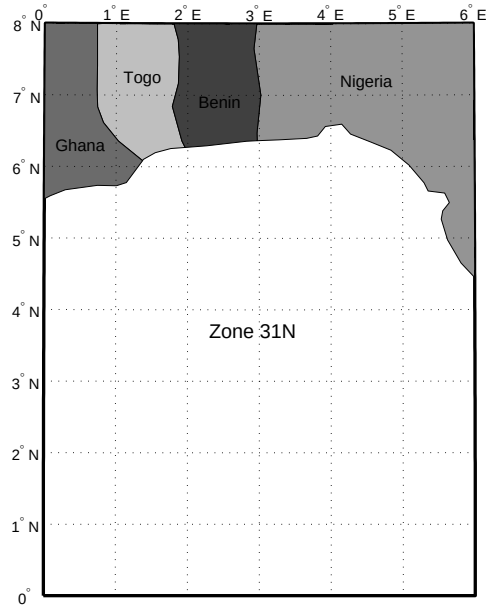
Symmetry: About the central meridian or the Equator.

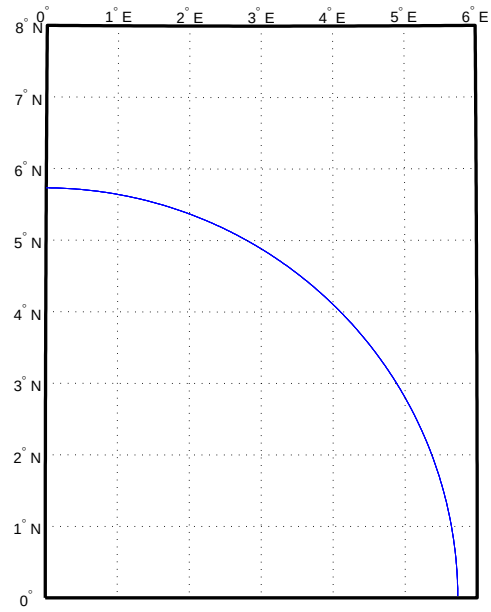
Features This is a conformal projection with parameters chosen to minimize distortion over a defined set of small areas. It is not equal area, equidistant, or perspective. Scale is true along two straight lines on the map approximately 180 kilometers east and west of the central meridian, and is constant along other straight lines equidistant from the central meridian. Scale is less than true between, and greater than true outside the lines of true scale.

Parallels There are no standard parallels for this projection. There are two lines of zero distortion by virtue of the scale factor.

Remarks The UTM system divides the world between 80° S and 84° degrees N into a set of quadrangles called zones. Zones generally cover 6 degrees of longitude and 8 degrees of latitude. Each zone has a set of defined projection parameters, including central meridian, false eastings and northings and the reference ellipsoid. The projection equations are the Gauss-Krüger versions of the Transverse Mercator. The projected coordinates form a grid system, in which a location is specified by the zone, easting and northing.

The UTM system was introduced in the 1940's by the U.S. Army. It is widely used in topographic and military mapping.


0°
1° E
2° E
3° E
4° E
5° E
6° E
8° N
7° N
6° N
5° N
4° N
3° N
2° N
1° N
0°
Togo
Benin
Nigeria
Ghana
Zone 31N


0°
1° E
2° E
3° E
4° E
5° E
6° E
8° N
7° N
6° N
5° N
4° N
3° N
2° N
1° N
0°

Van der Grinten I Projection

Classification Polyconic

Syntax `vgrint1`

Graticule Central Meridian: A straight line.

Meridians: Circular curves spaced equally along the equator and concave toward the central meridian.

Parallels: The Equator is a straight line. All other parallels are circular arcs concave toward the nearest pole.

Poles: Points.

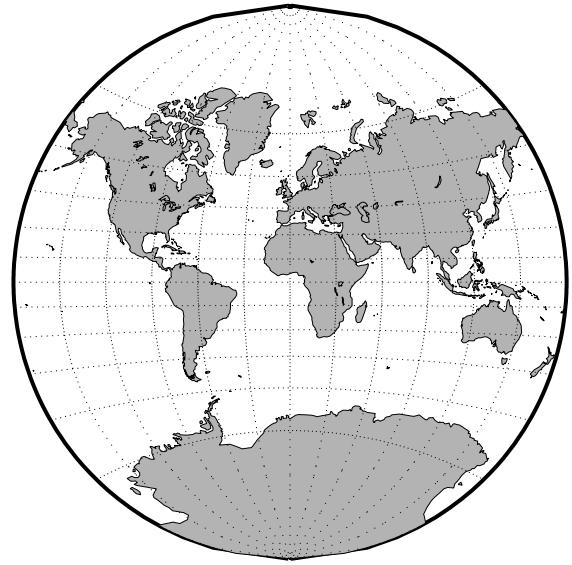
Symmetry: About the Equator or the central meridian.

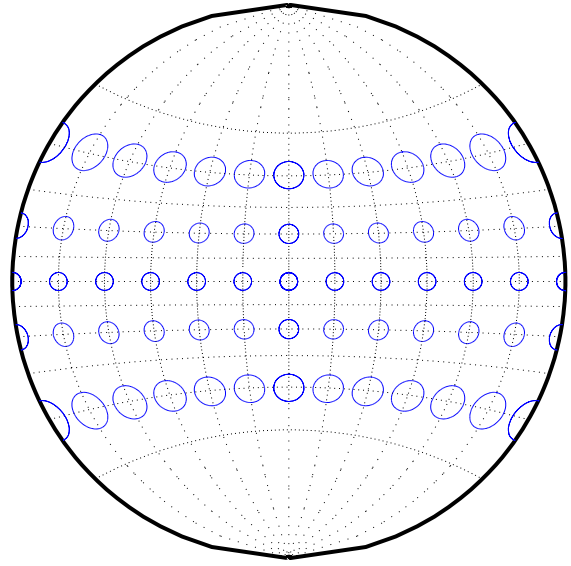
Features In this projection, the world is enclosed in a circle. Scale is true along the Equator and increases rapidly away from the Equator. Area distortion is extreme near the poles. This projection is neither conformal nor equal-area.

Parallels There are no standard parallels for this projection.

Remarks This projection was presented by Alphons J. Van der Grinten in 1898. He obtained a U.S. patent for it in 1904. It is also known simply as the Van der Grinten projection (without the “I”).

Limitations This projection is available for the spherical geoid only.





Vertical Perspective Azimuthal Projection

Classification Azimuthal

Syntax `vperspec`

Graticule The graticule described is for a polar aspect.

Meridians: Equally spaced straight lines intersecting at the central pole. The angles displayed are true angles between meridians.

Parallels: Unequally spaced circles centered on the central pole. Spacing decreases away from this pole. The opposite hemisphere cannot be shown, nor can distant parts of the closer hemisphere. The limit of visibility depends on the observation altitude.

Poles: The central pole is a point. The other pole is not shown.

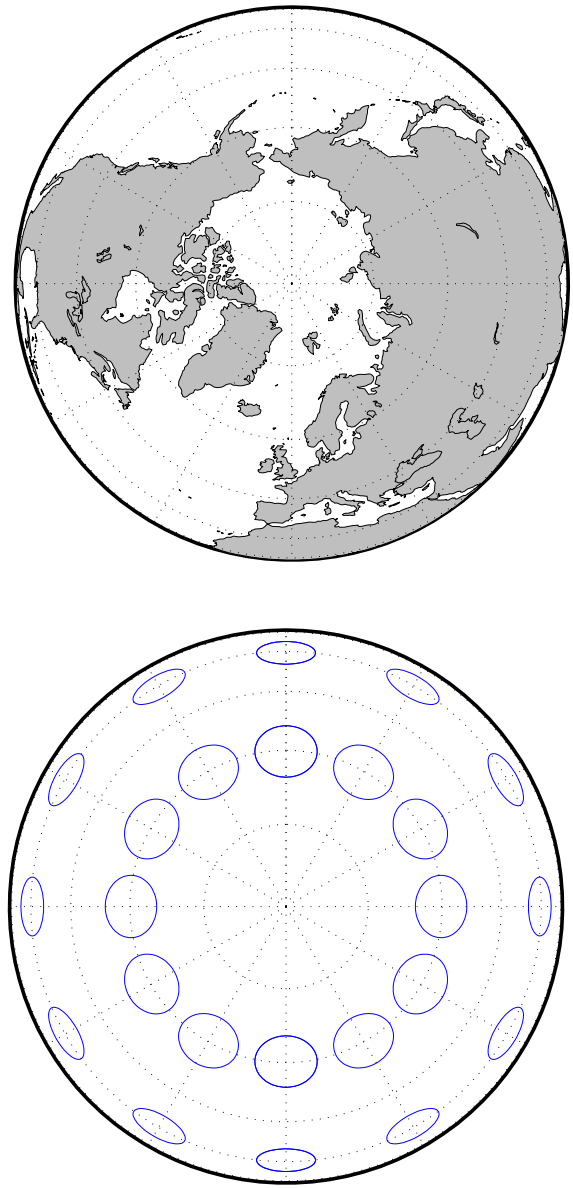
Symmetry: About any meridian.

Features This is a perspective projection on a plane tangent at the center point from a finite distance. Scale is true only at the center point, and is constant in the circumferential direction along any circle having the center point as its center. Distortion increases rapidly away from the center point, the only point which is distortion free. This projection is neither conformal nor equal area.

Parallels The standard parallel contains the observation altitude above the surface in the same units as the geoid semi-major axis.

Remarks This projection provides views of the globe resembling those seen from a spacecraft in orbit. The Orthographic projection is a limiting form with the observer at an infinite distance.

Limitations This projection is available for the spherical geoid only. Data more distant than the limit of visibility is trimmed.



Wagner IV Projection

Classification Pseudocylindrical

Syntax `wagner4`

Graticule Central Meridian: Straight line half as long as the Equator.

Other Meridians: Equally spaced portions of ellipses concave toward the central meridian. The meridians 103°55' east and west of the central meridian are circular arcs.

Parallels: Unequally spaced straight parallel lines, perpendicular to the central meridian. Spacing is greatest toward the Equator.

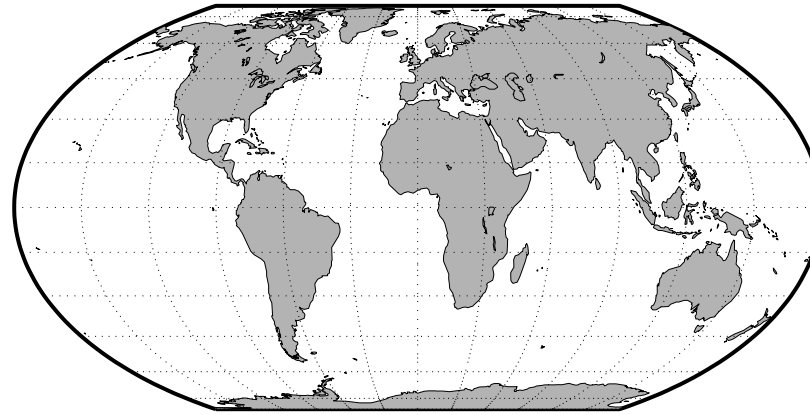
Poles: Lines half as long as the Equator.

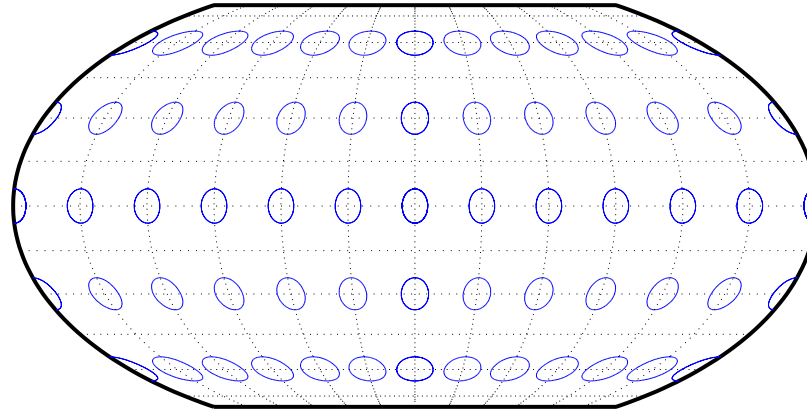
Symmetry: About the central meridian or the Equator.

Features This is an equal-area projection. Scale is true along the 42°59' parallels and is constant along any parallel and between any pair of parallels equidistant from the Equator. Distortion is not as extreme near the outer meridians at high latitudes as for pointed-polar pseudocylindrical projections, but there is considerable distortion throughout the polar regions. It is free of distortion only at the two points where the 42°59' parallels intersect the central meridian. This projection is not conformal or equidistant.

Parallels For this projection, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. The standard parallel is by definition fixed at 42°59'.

Remarks This projection was presented by Karlheinz Wagner in 1932.





Werner Projection

Classification Pseudoconic

Syntax `werner`

Graticule Central Meridian: A straight line.

Meridians: Complex curves connecting points equally spaced along each parallel and concave toward the central meridian.

Parallels: Concentric circular arcs spaced at true distances along the central meridian, centered on one of the poles.

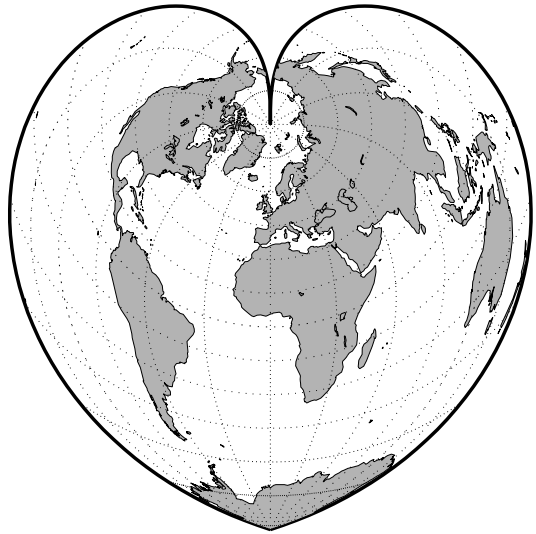
Poles: Points.

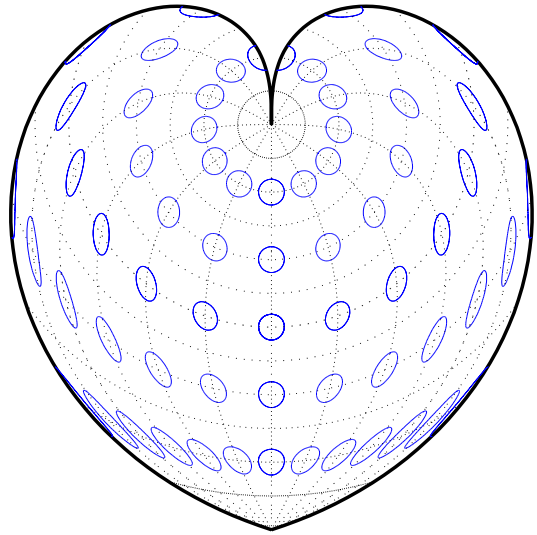
Symmetry: About the central meridian.

Features This is an equal-area projection. It is a Bonne projection with one of the poles as its standard parallel. The central meridian is free of distortion. This projection is not conformal. Its heart shape gives it the additional descriptor *cordiform*.

Parallels The standard parallel for this projection is set to 90° N.

Remarks This projection was developed by Johannes Stabius (Stab) about 1500 and was promoted by Johannes Werner in 1514. It is also called the Stab-Werner projection.





Wetch Cylindrical Projection

Classification Cylindrical

Syntax `wetch`

Graticule Central Meridian: Straight line (includes meridian opposite the central meridian in one continuous line).

Other Meridians: Straight lines if 90° from central meridian, complex curves concave toward the central meridian otherwise.

Parallels: Complex curves concave toward the nearest pole.

Poles: Points along the central meridian.

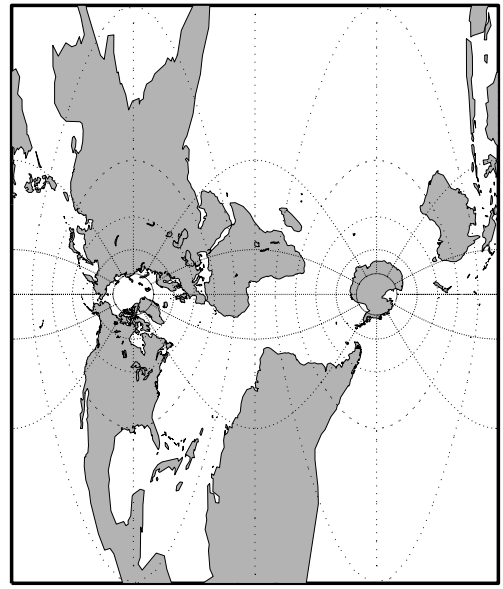
Symmetry: About any straight meridian or the Equator.

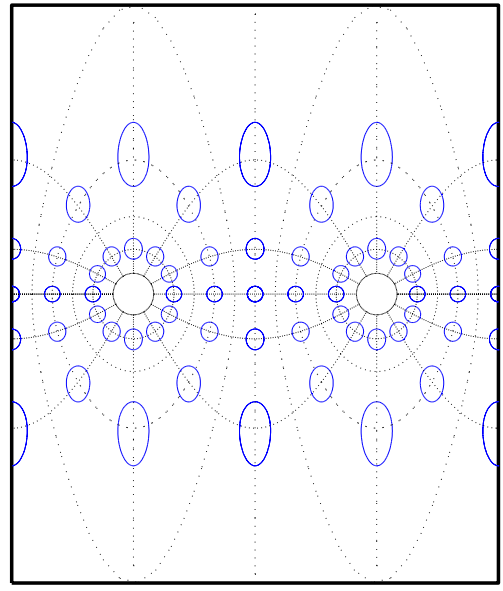
Features This is a perspective projection from the center of the Earth onto a cylinder tangent to the central meridian. It is not equal-area, equidistant, or conformal. Scale is true along the central meridian and constant between two points equidistant in *x* and *y* from the central meridian. There is no distortion along the central meridian, but it increases rapidly away from the central meridian in the *y*-direction.

Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, which is the transverse aspect of the Central Cylindrical, the standard parallel *of the base projection* is by definition fixed at 0°.

Remarks This is the transverse aspect of the Central Cylindrical projection discussed by J. Wetch in the early 19th century.

Limitations This projection is available for the spherical geoid only. To prevent large *y*-values from dominating the display, data at *y*-values that would correspond to latitudes of greater than 75° in the normal aspect of the Central Cylindrical projection is trimmed.





Wiechel Projection

Classification Pseudoazimuthal

Syntax `wiechel`

Graticule The graticule described is for a polar aspect.

Meridians: Equally spaced semicircles from pole to pole, concave toward the west.

Parallels: Concentric circles.

Pole: The central pole is a point; the other pole is a bounding circle.

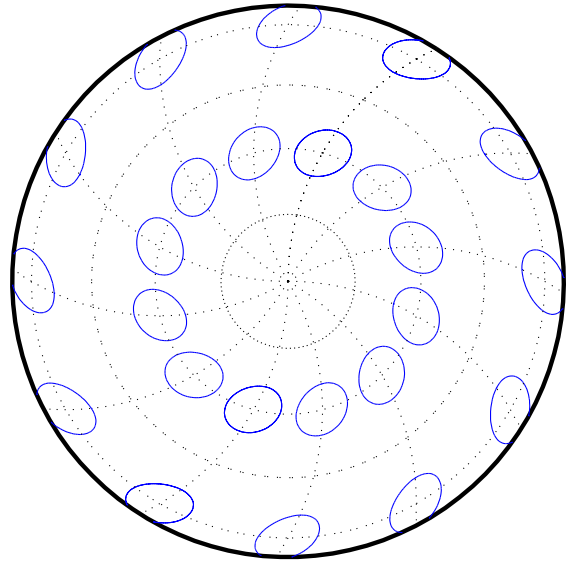
Symmetry: Radially about the center point.

Features This equal-area projection is a novelty map, usually centered at a pole, showing semicircular meridians in a pinwheel arrangement. Scale is correct along the meridians. This projection is not conformal.

Parallels There are no standard parallels for azimuthal projections.

Remarks This projection was presented by H. Wiechel in 1879.

Limitations Data greater than 65° distant from the center point is trimmed.



Winkel I Projection

Classification Pseudocylindrical

Syntax `winkel`

Graticule Central Meridian: Straight line at least half as long as the Equator.

Other Meridians: Equally spaced sinusoidal curves concave toward the central meridian.

Parallels: Equally spaced straight parallel lines, perpendicular to the central meridian.

Poles: Lines at least half as long as the Equator.

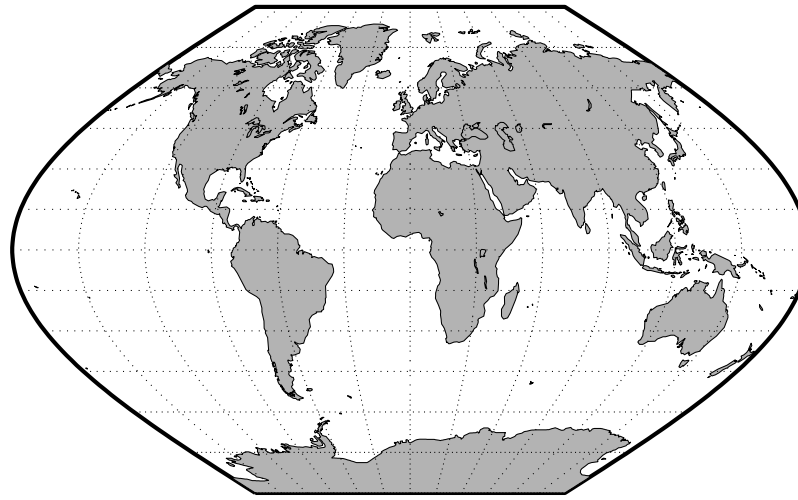
Symmetry: About the central meridian or the Equator.

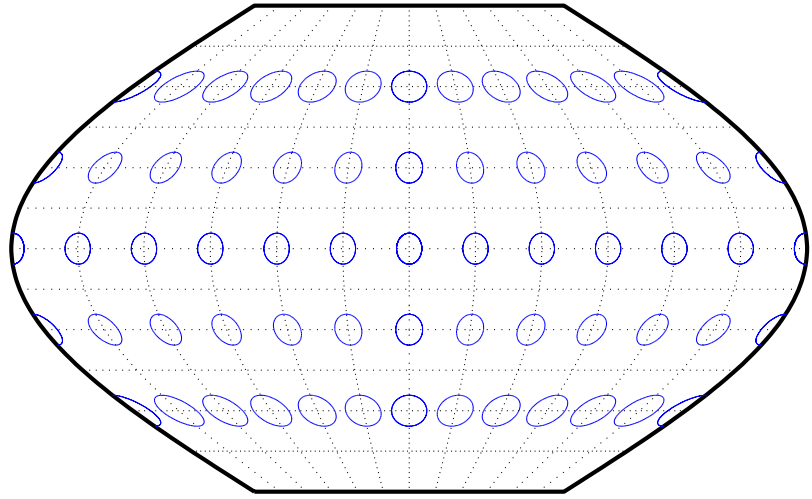
Features This projection is an arimethical average of the *x* and *y* coordinates of the Sinusoidal and Equidistant Cylindrical projections. Scale is true along the standard parallels and is constant along any parallel and between any pair of parallels equidistant from the Equator. There is no point free of distortion. This projection is not equal-area, conformal, or equidistant.

Parallels For this projection, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. Any latitude may be chosen; the default is set to 50°28'.

Remarks This projection was developed by Oswald Winkel in 1914. Its limiting form is the Eckert V when a standard parallel of 0° is chosen.

Limitations This projection is available for the spherical geoid only.





Winkel I Projection

3

GUI Reference

How to Use the GUI Reference Pages

The GUI Reference pages are organized alphabetically by the name of the function or tool. Most of the GUI tools in the Mapping Toolbox are activated by command-line functions without any input arguments. Users should consult the corresponding pages in Chapter 1, “Mapping Reference” as well. The entries in this chapter contain the following:

Purpose	Provides a short, concise description of the tool.
Activation	Provides methods of activation from the MATLAB command window, by mouse interaction with the map display, and/or from within `maptool`.
Description	Describes what the tool does, how each activation method works, and any rules or restrictions that apply.
Controls	Describes how to use the interface associated with the tool, including dialog boxes, menu bars, etc.
Examples	Provides examples of how the tool can be used.
See Also	Refers users to other related command, functions, or tools.

Graphical User Interface Tables

Map Definition Tools

Function	Description
`axesm` `axesmui`	Define map axes and display property setting.
`originui`	Allow interactive modification of map origin.
`parallelui`	Tool for interactively modifying map parallels.

Mapping Tools

Function	Description
`maptool`	Create figure window with menu-activated mapping tools.
`mlayers`	Manipulate layers of a map.
`mobjects`	Manipulate mapped object sets.
`qrydata`	Allow interactive queries of map data.

Object Projection Tools

Function	Description

`linem`
`plotm`
`plot3m`
	Project line objects onto map axes.

`fillm`
`fill3m`
`patchm`
	Project patch objects onto map axes.

`patchesm`	Project patches as individual objects onto map axes.
`meshm`	Warp regular matrix map onto projected graticule mesh.

`pcolorm`
`surface`
`surfm`
	Warp general matrix map onto projected graticule mesh.

`lightm`	Project light objects onto map axes.
`surflm`	Project 3-D shaded surface with lighting onto map axes.
`meshlsrm` `surflsrm`	Project 3-D lighted shaded surface onto map axes.
`textm`	Project text objects onto map axes.

Display Manipulation Tools

Function	Description
`clrmenu`	Add colormap menu to figure.
`panzoom`	Pan and zoom on 2-D map display.

Thematic Map Tools

Function	Description
`cometm` `comet3m`	Project comet plot onto map axes.
`contorm` `contor3m`	Project contour plot onto map axes.
`quiverm`	Project 2-D quiver plot onto map axes.
`quiver3m`	Project 3-D quiver plot onto map axes.
`stem3m`	Project stem plot onto map axes.
`scatterm`	Project proportional symbol map onto map axes.

Object Property Tools

Function	Description
`clmo`	Clear specified map objects from map axes.
`colorui`	Set custom RGB color triples.
`handlem`	Return handles of specified map objects.
`hidem`	Hide specified map objects.
`showm`	Show specified map objects.
`tagm`	Edit tag of specified map objects.
`zdatam`	Edit z-plane of specified map objects.
`property editors`	Edit properties of specified map objects.

Track Tools

Function	Description
`scirclui`	Display small circles on map axes.
`surfdist`	Calculate distance, azimuth, and reckoning.
`trackui`	Display great circle and rhumb line tracks on map axes.

Map Data Construction Tools

Function	Description
`colorm`	Create colormaps for regular matrix map.
`limitm`	Compute latitude and longitude limits of regular matrix map.
`maptrim`	Allow interactive trimming of map data.
`seedm`	Encode regular matrix map.

Purpose

Define map axes and modify map projection and display properties

Activation

Command Line	Maptool	Map Display

`axesm`
`axesm(h)`
`axesmui`
`axesmui(h)`
`c = axesmui(...)`
	Display⇒Projection	extend-click on map display

Description

`axesm` activates a **Projection Control** dialog box, which allows map projection definition and property modification. If no map is currently defined, `axesm` creates a map axes with the Robinson projection as the default.

`axesm(h)` activates the **Projection Control** box for the axes specified by the handle `h`.

`axesmui` activates the **Projection Control** box for the current map axes.

`axesmui(h)` activates the **Projection Control** box for the map axes specified by the handle `h`.

`c` is an optional output argument that indicates whether the **Projection Control** dialog box was closed by the cancel button. `c = 1` if the cancel button is pushed. Otherwise, `c = 0`.

Extend-clicking on a map display brings up the **Projection Control** dialog box for that map axes.

Controls

The **Map Projection** pull-down menu is used to select a map projection. The projections are listed by type, and each is preceded by a four-letter type indicator:

```
Cyln = Cylindrical
Pcyl = Pseudocylindrical
Coni = Conic
Poly = Polyconic
Pcon = Pseudoconic
Azim = Azimuthal
Mazi = Modified Azimuthal
Pazi = Pseudoazimuthal
```

The **Zone** button and edit box are used to specify the UTM or UPS zone. For non-UTM and UPS projections, the two are disabled.

The **Geoid** edit boxes and pull-down menu are used to specify the geoid. Units must be in meters for the UTM and UPS projections, since this is the standard unit for the two projections. For non-UTM and UPS projections, the geoid unit can be anything, bearing in mind that the resulting projected data will be in the same units as the geoid.

The **Angle Units** pull-down menu is used to specify the angle units used on the map projection. All angle entries corresponding to the current map projection must be entered in these units. Current angle entries are automatically updated when new angle units are selected.

The **Map Limits** edit boxes are used to specify the extent of the map data in geographic coordinates. The **Latitude** edit boxes contain the southern and northern limits of the map. The **Longitude** edit boxes contain the western and eastern limits of the map. The map limits establish the extent of the meridian and parallel grid lines, regardless of the display settings (see grid settings). Map limits are always in Greenwich coordinates, regardless of the map origin and orientation setting. In the normal aspect, the map display is trimmed to the minimum of the map and frame limits.

The **Frame Limits** edit boxes are used to specify the location of the map frame, measured from the center of the map projection in the base coordinate system. The **Latitude** edit boxes contain the southern and northern frame edge locations. The **Longitude** edit boxes contain the western and eastern frame edge locations. Displayed map data are trimmed at the frame limits. For azimuthal map projections, the latitude limits should be set to `-inf` and the desired trim distance from the map origin. In the normal aspect, the map display is trimmed to the minimum of the map and frame limits.

The **Map Origin** edit boxes are used to specify the origin and aspect angle of the map projection. The **Lat** and **Long** boxes specify the map origin in Greenwich coordinates. This is the point that is placed in the center of the projection. If either box is left blank, 0 degrees is used. The **Orientation** box specifies the azimuth angle of the North Pole relative to the map origin. Azimuth is measured clockwise from the top of the projection. If the **Orientation** box is disabled, then the selected map projection requires a fixed orientation. See the section entitled “The Origin Vector” in Chapter 4 of the *Mapping Toolbox User's Guide* for a complete description of the map origin.

The **Cartesian Origin** edit boxes are used to specify the x-y offset, along with a desired scale factor of the map projection. The **False E and N** boxes specify the false easting and northing in Cartesian coordinates. These must be in the same units as the geoid. The **Scalefactor** box specifies the scale factor used in the map projection calculations.

The **Parallels** edit boxes specify the standard parallels of the selected map projection. A particular map projection may have one or two standard parallels. If the edit boxes are disabled, then the selected projection has no standard parallels or the standard parallels are fixed.

The **Aspect** pull-down menu is used to select a normal or transverse display aspect. When the aspect is normal, *north* (on the base projection) is up, and the map is displayed in a *portrait* setting. In a transverse aspect, north (in the base projection) is to the right, and the map is displayed in a *landscape* setting. This property does not control the map projection aspect. The projection aspect is determined by the map `Origin` property).

The **Frame** button brings up the **Map Frame Properties** dialog box, which allows the map frame settings to be modified.

The **Grid** button brings up the **Map Grid Properties** dialog box, which allows the map grid settings to be modified.

The **Labels** button brings up the **Map Label Properties** dialog box, which allows the parallel and meridian label settings to be modified.

The **Fill in** button is used to compute projection and display settings based on any currently specified map parameters. Only settings that are left blank are affected when this button is pushed.

The **Reset** button is used to reset the default projection properties and display settings of the current map. Default display settings include frame, grid, and label properties set to `'off'`.

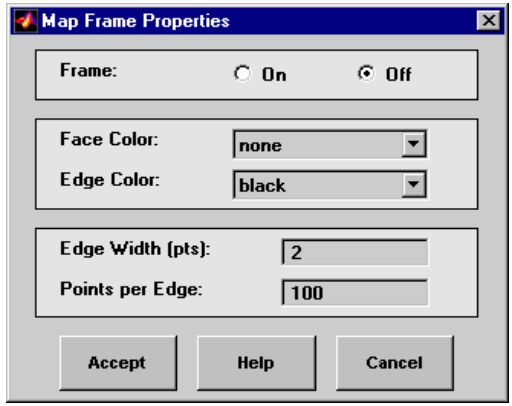
The **Apply** button is used to apply the projection and display settings to the current map, which results in the map being reprojected.

The **Help** button is used to bring up online help text for each control on the **Projection Control** dialog box.

The **Cancel** button disregards any modified projection or display settings and closes the **Projection Control** dialog box.

Map Frame Properties Dialog Box

This dialog box allows modification of the map frame settings. It is accessed via the **Frame** button on the **Projection Control** dialog box.



The **Frame** selection buttons determine whether the map frame is visible.

The **Face Color** pull-down menu is used to select the background color of the map frame. Selecting `none` results in a transparent frame background, i.e., the same as the axes color. Selecting `custom` allows a custom RGB triple to be defined for the background color.

The **Edge Color** pull-down menu is used to select the color of the frame edge. Selecting `none` hides the frame edge. Selecting `custom` allows a custom RGB triple to be defined for the edge color.

The **Edge Width** edit box is used to enter the line width of the frame edge, in points.

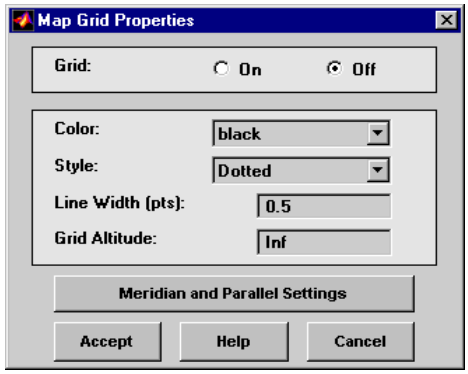
The **Points per Edge** edit box is used to enter the number of points used to display each edge of the map frame.

The **Accept** button accepts any modifications made to the map frame properties and returns to the **Projection Control** dialog box. Changes are applied to the current map only when the **Apply** button on the **Projection Control** dialog box is pushed.

The **Cancel** button disregards any modifications to the map frame properties and returns to the **Projection Control** dialog box.

Map Grid Properties Dialog Box

This dialog box allows modification of the map frame settings. It is accessed via the **Grid** button on the **Projection Control** dialog box.



The **Grid** selection buttons determine whether the map grid is visible.

The **Color** pull-down menu is used to select the color of the map grid lines. Selecting `custom` allows a custom RGB triple to be defined for the grid line color.

The **Style** pull-down menu is used to select the line style of the map grid lines.

The **Line Width** edit box is used to enter the width of the map grid lines, in points.

The **Grid Altitude** edit box is used to enter *z*-axis location of the map grid. This property can be used to place some mapped objects above or below the map grid. The default map grid altitude is `inf`, which places the grid above all other mapped objects.

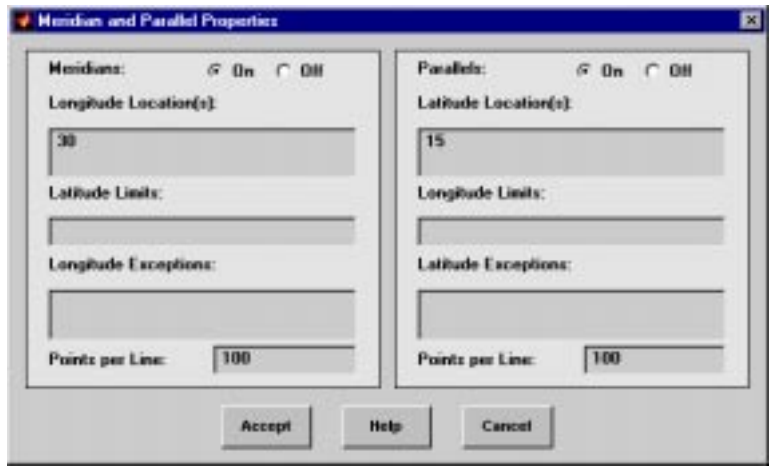
The **Meridian and Parallel Settings** button brings up the **Meridian and Parallel Properties** dialog box, which allows the properties of the meridian and parallel grid lines to be modified.

The **Accept** button accepts any modifications made to the map grid properties and returns to the **Projection Control** dialog box. Changes are applied to the current map only when the **Apply** button on the **Projection Control** dialog box is pushed.

The **Cancel** button disregards any modifications to the map grid properties and returns to the **Projection Control** dialog box.

Meridian and Parallel Properties Dialog Box

This dialog box is used to modify the settings for meridian and parallel grid lines. It is accessed via the **Meridian and Parallel Settings** button on the **Map Grid Properties** dialog box.



The **Meridians** selection buttons determine whether the meridian grid lines are visible when the map grid is turned on.

The **Longitude Location(s)** edit box is used to specify which meridians are to bedisplayed if the meridian lines are turned on. If a scalar interval value is entered, meridian lines are displayed at that interval, starting from the Prime Meridian and proceeding in east and west directions. If a vector of values is entered, meridian lines are displayed at locations given by each element of the vector.

The **Latitude Limits** edit box is used to specify the latitude limits beyond which meridian lines do not extend. If this property is left empty, all meridian lines extend to the map latitude limits (specified by the Latitude Map Limits entry on the **Projection Control** dialog box). This entry must be a two-element vector enclosed in brackets.

The **Longitude Exceptions** edit box is used to enter specific meridians of the displayed grid that are to extend beyond the latitude limits, to the map limits. This entry is a vector of longitude values.

The **Parallels** selection buttons determine whether the parallel grid lines are visible when the map grid is turned on.

The **Latitude Location(s)** edit box is used to specify which parallels are to be displayed if the parallel lines are turned on. If a scalar interval value is entered, parallel lines are displayed at that interval, starting from the Equator and proceeding in north and south directions. If a vector of values is entered, parallel lines are displayed at locations given by each element of the vector.

The **Longitude Limits** edit box is used to specify the longitude limits beyond which parallel lines do not extend. If this property is left empty, all parallel lines extend to the map longitude limits (specified by the Longitude Map Limits entry on the **Projection Control** dialog box). This entry must be a two-element vector enclosed in brackets.

The **Latitude Exceptions** edit box is used to enter specific parallels of the displayed grid that are to extend beyond the longitude limits, to the map limits. This entry is a vector of latitude values.

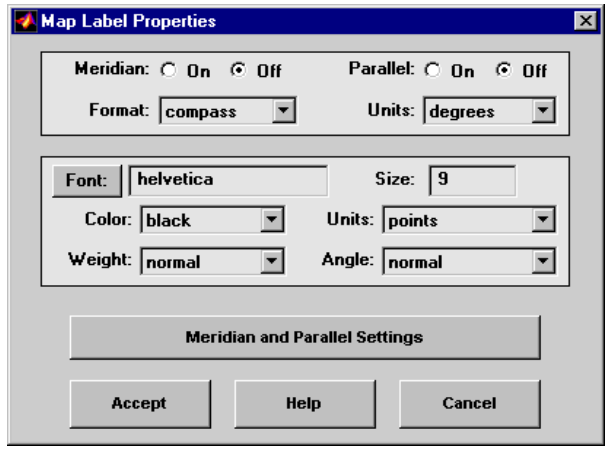
The **Points per Line** edit boxes are used to enter the number of points used to plot each meridian and each parallel grid line. The default value is 100 points.

The **Accept** button accepts any modifications that have been made to the meridian and parallel grid line properties and return to the **Map Grid Properties** dialog box. Changes are applied to the current map only when the **Apply** button on the **Projection Control** dialog box is pushed.

The **Cancel** button disregards any modifications to the meridian and parallel grid lines and returns to the **Map Grid Properties** dialog box.

Map Label Properties Dialog Box

This dialog box is used to modify the settings of the meridian and parallel labels. It is accessed via the **Label** button on the **Projection Control** dialog box.



The **Meridian** and **Parallel** selection buttons determine whether the meridian and parallel labels are visible.

The **Format** pull-down menu is used to specify the format of the grid labels. If `compass` is selected, meridian labels are appended with `E` for east and `W` for west, and parallel labels are appended with `N` for north and `S` for south. If `signed` is chosen, meridian labels are prefixed with + for east and – for west, and parallel labels are prefixed with + for north and – for south. If `none` is selected, western meridian labels and southern parallel labels are prefixed by –, but no symbol precedes eastern meridian labels and northern parallel labels.

The label **Units** pull-down menu is used to specify the angle units used to display the parallel and meridian labels. These units, used for display purposes only, need not be the same as the angle units of the map projection.

The **Font** edit box is used to specify the character font used to display the parallel and meridian labels. If the font specified does not exist on the computer, the default of `Helvetica` is used. Pressing the **Font** button previews the selected font.

The font **Size** edit box is used to enter an integer value that specifies the font size of the parallel and meridian labels. This value must be in the units specified by the font **Units** pull-down menu.

The font **Color** pull-down menu is used to select the color of the parallel and meridian labels. Selecting `custom` allows a custom RGB triple to be defined for the labels.

The font **Weight** pull-down menu is used to specify the character weight of the parallel and meridian labels.

The font **Units** pull-down menu is used to specify the units used to interpret the font size entry. When set to `normalized`, the value entered in the **Size** edit box is interpreted as a fraction of the height of the axes. For example, a normalized font size of 0.1 sets the label text to a height of one tenth of the axes height.

The font **Angle** pull-down menu is used to select the character slant of the parallel and meridian labels. `normal` specifies non-italic font. `italic` and `oblique` specify italic font.

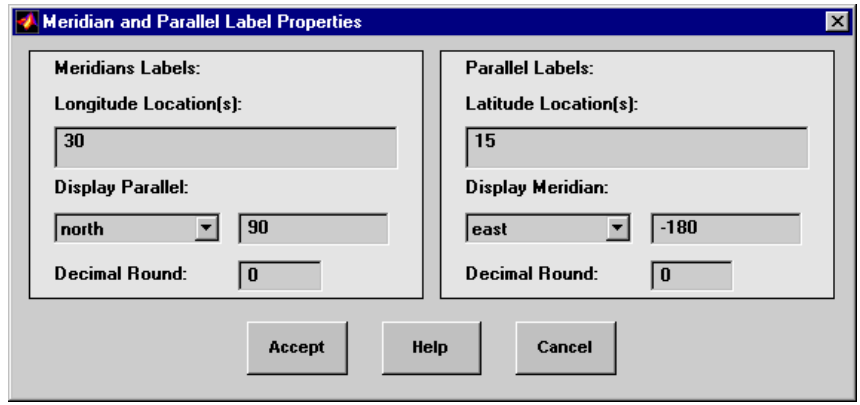
The **Meridian and Parallel Settings** button brings up the **Meridian and Parallel Label Properties** dialog box, which allows modification of properties specific to the meridian and parallel grid labels.

The **Accept** button accepts any modifications that have been made to the map label properties and returns to the **Projection Control** dialog box. Changes are applied to the current map only when the **Apply** button on the **Projection Control** dialog box is pushed.

The **Cancel** button disregards any modifications to the map labels and returns to the **Projection Control** dialog box.

Meridian and Parallel Label Properties Dialog Box

This dialog box is used to modify properties specific to the meridian and parallel grid labels. It is accessed via the **Meridian and Parallel Settings** button on the **Map Label Properties** dialog box.



The **Longitude Location(s)** edit box is used to specify which meridians are to be labeled. Meridian labels need not coincide with displayed meridian grid lines. If a scalar interval value is entered, labels are displayed at that interval, starting from the Prime Meridian and proceeding in east and west directions. If a vector of values is entered, labels are displayed at longitude locations given by each element of the vector.

The **Display Parallel** pull-down menu and edit box are used to specify the latitude location of the meridian labels. If a scalar latitude value is provided in the edit box, the meridian labels are placed at that parallel. Alternatively, the pull-down menu can be used to select a latitude location. If `north` is chosen, meridian labels are placed at the maximum map latitude limit. If `south` is chosen, meridian labels are placed at the minimum map latitude limit.

The **Latitude Location(s)** edit box is used to specify which parallels are to be labeled. Parallel labels need not coincide with displayed parallel grid lines. If a scalar interval value is entered, labels are displayed at that interval, starting from the Equator and proceeding in north and south directions. If a vector of values is entered, labels are displayed at latitude locations given by each element of the vector.

The **Display Meridian** pull-down menu and edit box are used to specify the longitude location of the parallel labels. If a scalar longitude value is provided in the edit box, the parallel labels are placed at that meridian. Alternatively, the pull-down menu can be used to specify a longitude location. If `east` is

chosen, parallel labels are placed at the maximum map longitude limit. If `west` is chosen, parallel labels are placed at the minimum map longitude limit.

The **Decimal Round** edit boxes are used to specify the power of ten to which the meridian and parallel labels are rounded. For example, a value of -1 results in labels displayed to the tenths decimal place.

The **Accept** button accepts any modifications that have been made to the meridian and parallel label properties and return to the **Map Label Properties** dialog box. Changes are applied to the current map only when the **Apply** button on the **Projection Control** dialog box is pushed.

The **Cancel** button disregards any modifications to the meridian and parallel labels and returns to the **Map Label Properties** dialog box.

The **Map Geoid** edit box is used to specify the geoid definition for the current map axes. The geoid is defined by a two-element vector of the form `[semimajor-axis eccentricity]`. Eccentricity must be a value between 0 and 1, but not equal to 1. A nonzero eccentricity represents an ellipsoid. The default geoid is a sphere with radius 1, represented as `[1 0]`. If a scalar entry is provided, it is assumed to be the radius of a sphere.

The **Accept** button accepts any modifications that have been made to the map geoid and return to the **Projection Control** dialog box. Changes are applied to the current map only when the **Apply** button on the **Projection Control** dialog box is pushed.

The **Cancel** button disregards any modifications to the map geoid and returns to the **Projection Control** dialog box.

See Also

`axesm`

Purpose

Clear mapped objects

Activation

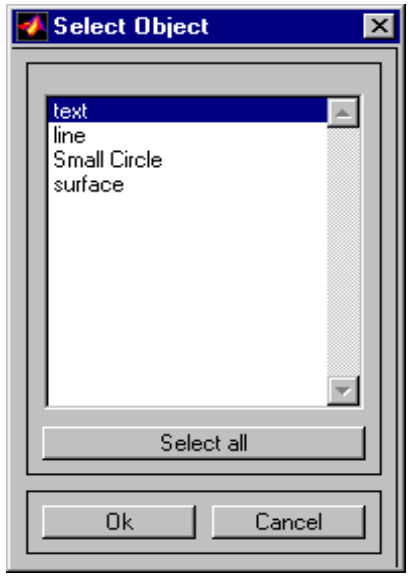
Command Line	Maptool
`clmo`	**Tools⇒Delete⇒Object**

Description

`clmo` brings up a **Select Object** dialog box for selecting mapped objects to delete.

Controls

The scroll box is used to select the desired objects from the list of mapped objects.



Pushing the **Select all** button highlights all objects in the scroll box for selection. Pushing the **Ok** button deletes the selected objects from the map. Pushing the **Cancel** button aborts the operation.

See Also

`clmo`

clrmenu

Purpose

Add a colormap menu to a figure

Activation

Command Line
`clrmenu` `clrmenu(h)`

Description

`clrmenu` adds a colormap menu to the current figure.

`clrmenu(h)` adds a colormap menu to the figure specified by the handle `h`.

Controls

The following choices are included on the colormap menu:

Gray, **Hsv**, **Hot**, **Pink**, **Cool**, **Bone**, **Jet**, **Copper**, **Spring**, **Summer**, **Autumn**, **Winter**, **Flag**, and **Prism** generate colormaps.

Rand is a random colormap.

Brighten increases the brightness.

Darken decreases the brightness.

Flipud inverts the order of the colormap entries.

Fliplr interchanges the red and blue components.

Permute permutes the colormap: red –> blue, blue –> green, green –> red.

Spin spins the colormap.

Define allows a workspace variable to be specified for the colormap.

Digital Elevation activates the DEM Color Map Input dialog box associated with the `demcmap` tool. This tool is used to create a colormap for a digital elevation map.

Remember stores the current colormap.

Restore reverts to the stored colormap (initially, the stored colormap is the colormap in use when `clrmenu` is invoked).

Refresh redraws the current figure window.

See Also

`colormap`

Purpose

Create colormaps for an indexed regular matrix map

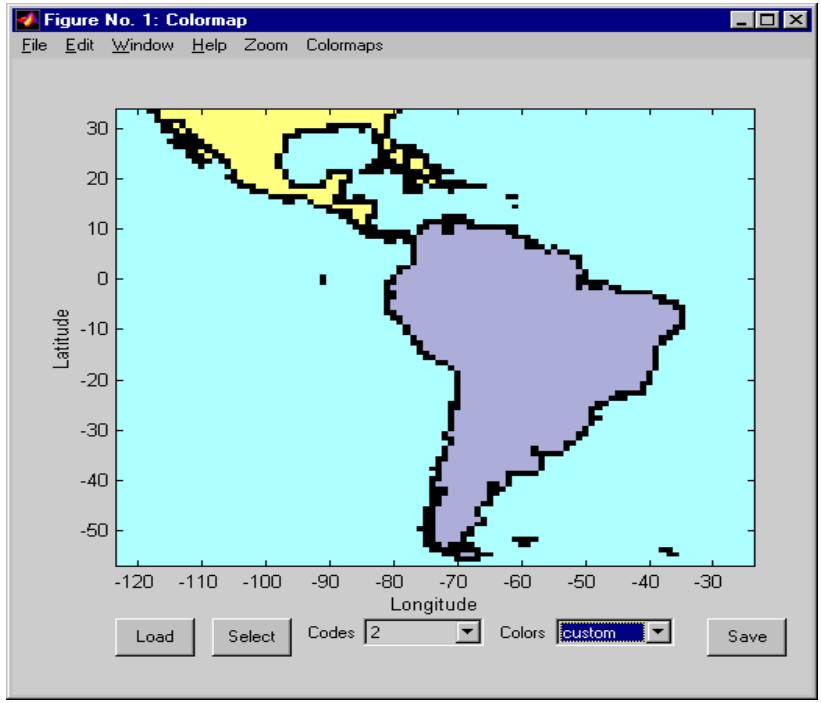
Activation

Command Line
`colorm(map,maplegend)`

Description

`colorm(map,maplegend)` displays the matrix map in a new figure window and allows a colormap to be edited and saved to a new variable. `map` and `maplegend` are the matrix map and the matrix map legend vector of the surface. `map` must have positive index values into the colormap.

Controls



The `colorm` tool displays the surface map data in a new figure window with the current colormap. **Zoom** and **Colormaps** menus are activated for that figure.

The **Zoom On/Off** menu toggles the `panzoom` box on and off. The box can be moved by clicking on the new location or by dragging the box to the new location. The box size can be increased or decreased by dragging a corner of the box. Pressing the `Return` key or double-clicking in the center of the box zooms in.

The **Colormaps** menu provided a variety of colormap options that can be applied to the map. See `clrmenu` in this guide for a description of the **Colormaps** menu options.

The **Load** button activates a dialog box, used to specify a colormap variable to be applied to the displayed surface map. This colormap can then be edited and saved.

The **Select** button activates the mouse cursor and allows a point on the map to be selected. The value of that point then appears in the **Codes** pull-down menu. The color of the selected point appears in the **Color** pull-down menu and can then be edited.

The **Codes** pull-down menu is used to select a particular value in the matrix map. The color associated with that value then appears in the **Color** pull-down menu and can be edited.

The **Color** pull-down menu is used to select a particular color to assign to the value currently displayed in the Codes pull-down menu. A custom color can be defined by selecting the `custom` option. This brings up a custom color interface with which an RGB triple can be selected.

The **Save** button is used to save the modified colormap to the workspace. A dialog box appears in which the colormap variable name is entered.

See Also

`encodem` `getseeds` `maptrim` `panzoom` `seedm`

Purpose

Project animated 2-D and 3-D comet plots on the current map axes

Activation

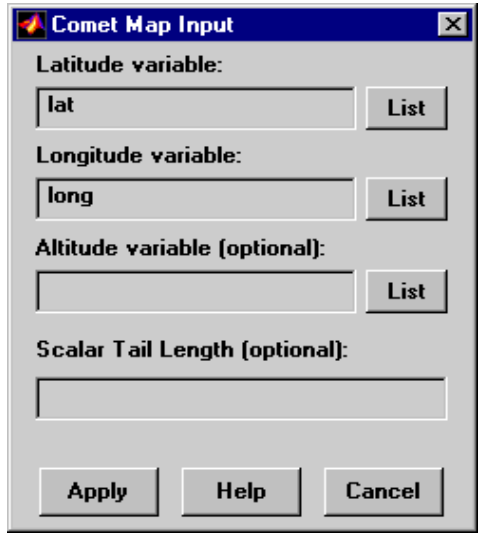
Command Line	Maptool
`cometm` `comet3m`	**Map⇒Comet**

Description

`cometm` and `comet3m` activate a **Comet Map Input** dialog box for projecting comet plots onto the current map axes.

If no map axes are current, a **No Map Axes** dialog box appears. Choose **Yes** to activate the **Projection Control** dialog box for defining map axes properties. Upon creation of the map axes, the **Comet Map Input** dialog box appears.

Controls



The **Latitude variable** edit box is used to specify the workspace variable containing the latitude data for the comet plot.

The **Longitude variable** edit box is used to specify the workspace variable containing the longitude data for the comet plot.

The **Altitude variable** edit box is used to specify the workspace variable containing the altitude data for the comet plot.

Pressing the **List** button produces a list of all current workspace variables, from which the latitude, longitude, and altitude variables can be selected.

The **Scalar Tail Length** edit box is used to enter a scalar value between 0 and 1 for the length of the comet tail. The default value is 0.1.

Pressing the **Apply** button accepts the input data and projects the comet plot onto the current map axes.

Pressing the **Cancel** button disregards any input data and closes the **Comet Map Input** dialog box.

See Also

`cometm  comet3m`

Purpose

Project 2-D and 3-D contour plots onto the current map axes

Activation

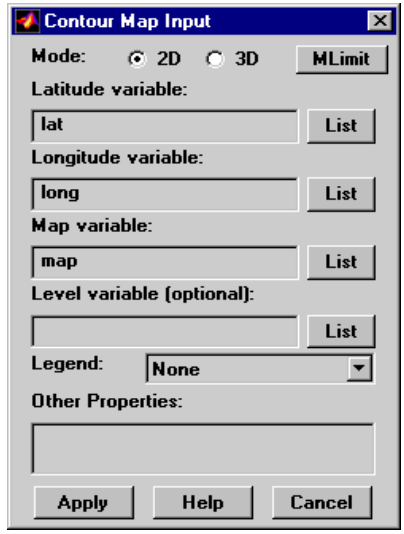
Command Line	Maptool
`contorm` `contor3m`	**Map⇒Contours**

Description

`contorm` and `contor3m` activate a **Contour Map Input** dialog box to project contour lines onto the current map axes.

If no map axes are current, a **No Map Axes** dialog box appears. Choose **Yes** to activate the **Projection Control** dialog box for defining map axes properties. Upon creation of the map axes, the **Contour Map Input** dialog box appears.

Controls



The **Mode** selection buttons are used to indicate a two- or three-dimensional contour plot.

The **MLimit** button brings up a **Map Limit Input** dialog box that computes the limits of a regular matrix map and stores them as variables that can be used as the latitude and longitude inputs for the contour plot. This enables the

creation of contour plots for regular matrix maps. See `limitm` in this guide for more information about the **Map Limit Input** dialog box.

The **Latitude variable** edit box is used to specify the workspace variable containing the latitude vector or matrix for the contour plot. If a vector, it should be monotonically increasing and describe the latitude of each row of the matrix map. If a matrix, it should be the size of the map matrix and give the latitude associated with each map matrix element.

The **Longitude variable** edit box is used to specify the workspace variable containing the longitude vector or matrix for the contour plot. If a vector, it should be monotonically increasing and describe the longitude of each column of the matrix map. If a matrix, it should be the size of the map matrix and give the longitude associated with each map matrix element.

The **Map variable** edit box is used to specify the workspace variable containing the matrix map.

The **Level variable** edit box is used to specify the workspace variable containing the values of the contours to be plotted. A vector of contour level values, enclosed in brackets, can be entered instead of a variable name. If omitted, the contour values are chosen automatically.

Pressing the **List** button produces a list of all current workspace variables, from which the latitude, longitude, map, and level variables can be selected.

The **Legend** pull-down menu is used to select the type of contour labeling or legend to be added to the plot. If the `Plot Legend` option is selected, any existing legend is deleted.

The **Other Properties** edit box is used to specify additional properties of the contour lines, such as `'Color','b'`. String entries must be enclosed in quotes. Linespec strings, such as `'c-'`, are also valid entries.

Pressing the **Apply** button accepts the input data and projects the contour plot onto the current map axes.

Pressing the **Cancel** button disregards any input data and closes the **Contour Map Input** dialog box.

See Also

`contor  contor3m`

Purpose

Create and assign a colormap to a digital elevation matrix map

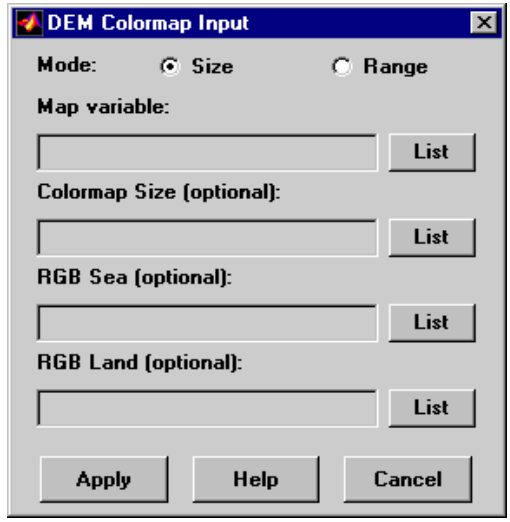
Activation

Command Line	Maptool
`demcmap`	**Colormaps⇒Digital Elevation**

Description

demcmap activates the **DEM Color Map Input** dialog box, which accepts inputs used to create a colormap for a digital elevation matrix map, and then applies the colormap to the current figure. The number of land and sea colors in the colormap is appropriate for the maximum elevations and depths of the matrix map.

Controls



The **Mode** selection buttons are used to specify whether the length of the colormap is specified or whether the altitude range increment assigned to each color is specified.

The **Map variable** edit box is used to specify the matrix map containing the elevation data.

The **Color Map Size** edit box is used in Size mode. This entry defines the length of the colormap. If omitted, a default length of 64 is used. This entry must be a scalar value.

The **Altitude Range** edit box is used in Range mode. This entry defines the altitude range increment assigned to each color. If omitted, a default increment of 100 is used. This entry must be a scalar value.

The **RGB Sea** edit box is used to define colors for data with negative values. The actual sea colors of the generated colormap are interpolated from this matrix. This entry can be a matrix of any length (n by 3). The colormap matrix of the current figure can be used by entering the string `'window'` in this box. The `demcmap` function provides default sea colors, which are used if this entry is left blank.

The **RGB Land** edit box is used to define colors for data with positive values. The actual land colors of the generated colormap are interpolated from this matrix. This entry can be a matrix of any length (n by 3). The colormap matrix of the current figure can be used by entering the string `'window'` in this box. The `demcmap` function provides default sea colors, which are used if this entry is left blank.

Pressing the **Apply** button accepts the input data, creates the colormap, and assigns it to the current figure.

Pressing the **Cancel** button disregards any input data and closes the **DEM Color Map Input** dialog box.

See Also

`demcmap`

Purpose

Project patch objects on the current map axes

Activation

Command Line	Maptool

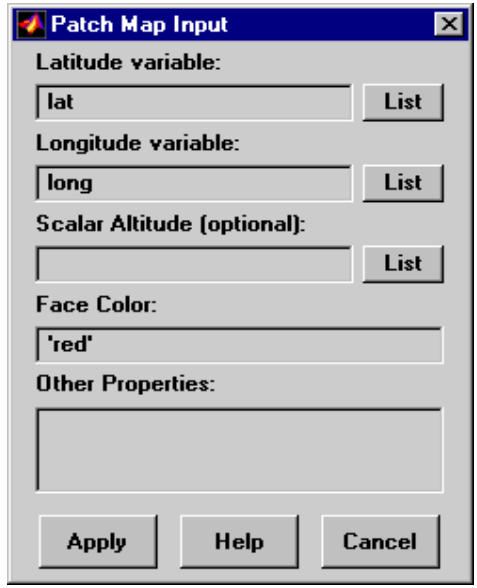
`fillm`
`fill3m`
`patchm`
`patchesm`
	Map⇒Patch

Description

`fillm`, `fill3m`, `patchm`, and `patchesm` all activate a **Patch Map Input** dialog box that accepts input data to project a patch object onto the current map axes.

If no map axes are current, a **No Map Axes** dialog box appears. Choose **Yes** to activate the **Projection Control** dialog box for defining map axes properties. Upon creation of the map axes, the **Patch Map Input** dialog box appears.

Controls



The **Latitude variable** edit box is used to specify the workspace variable containing the latitude data of the patch object to be projected.

The **Longitude variable** edit box is used to specify the workspace variable containing the longitude data of the patch object to be projected.

The **Scalar Altitude** edit box is used to specify a scalar value or scalar workspace variable that determines the plane in which the mapped patch object is to be displayed.

Pressing the **List** button produces a list of all current workspace variables, from which the latitude, longitude, and altitude variables can be selected.

The **Face Color** edit box is used to specify the color of the patch face. A valid color string, enclosed in quotes, or an RGB triple enclosed in brackets, can be entered. A workspace variable can also be entered, provided it is a color string or an RGB triple.

The **Other Properties** edit box is used to specify additional properties of the patch object to be projected, such as `'EdgeColor','none'`. String entries must be enclosed in quotes.

Pressing the **Apply** button accepts the input data and projects the patch object onto the current map axes.

Pressing the **Cancel** button disregards any input data and closes the **Patch Map Input** dialog box.

See Also

`fillm` `fill3m` `patchm` `patchesm`

Purpose

Return handles of mapped objects

Activation

Command Line:h = handlem
`h = handlem('prompt')`

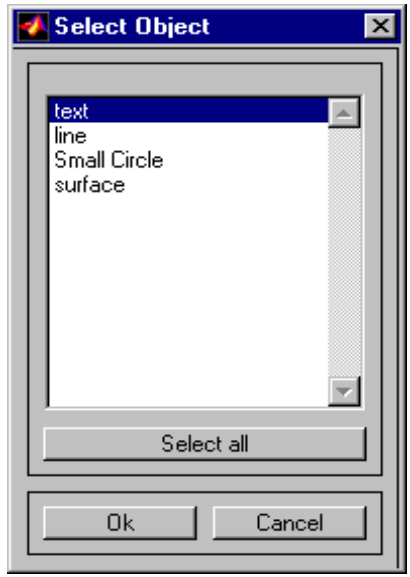
Description

`h = handlem` brings up a **Select Object** dialog box, which lists all currently displayed objects. Objects can be selected and their handles returned.

`h = handlem('prompt')` brings up a **Specify Object** dialog box, which allows greater control of object selection.

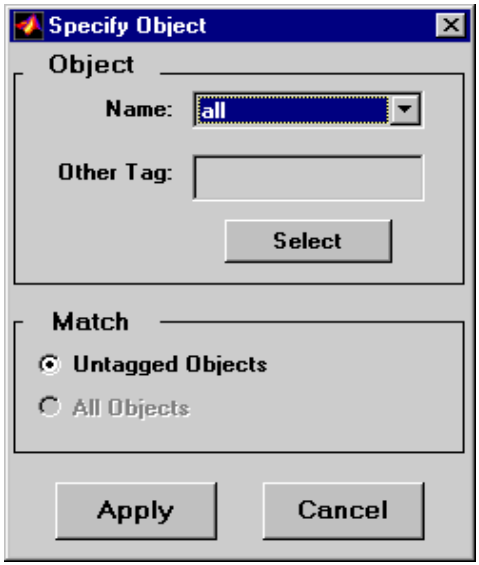
Controls

Select Object Dialog Box



The scroll box is used to select the desired objects from the list of mapped objects. Pushing the **Select all** button highlights all objects in the scroll box for selection. Pushing the **Ok** button returns the object handles in the variable `h`. Pushing the **Cancel** button aborts the operation.

Specify Object Dialog Box



The **Object** Controls are used to select an object type or tag. The **Name** pull-down menu is used to select from a list of predefined object strings. The **Other Tag** edit box is used to specify an object tag not listed in the **Name** pull-down menu. Pushing the **Select** button brings up the **Select Object** dialog box, which shows only the currently displayed objects for selection.

The **Match** Controls are used when a Handle Graphics object type (image, line, surface, patch, or text) is specified. The **Untagged Objects** selection button is used to return the handles of only those objects with empty tag properties. The **All Objects** selection button is used to return all object handles of the specified type, regardless of whether they are tagged.

Pushing the **Apply** button returns the handles of the specified objects. Pushing the **Cancel** button aborts the operation.

See Also

`handlem`

Purpose

Hide mapped objects

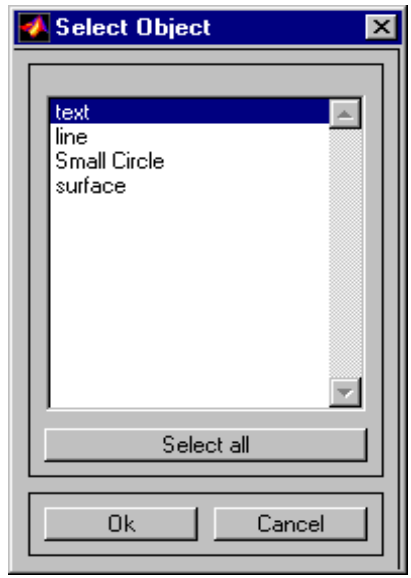
Activation

Command Line	Maptool
`hidem`	**Tools⇒Hide⇒Object**

Description

`hidem` brings up a **Select Object** dialog box for selecting mapped objects to hide (`Visible` property set to `'off'`).

Controls



The scroll box is used to select the desired objects from the list of mapped objects. Pushing the **Select all** button highlights all objects in the scroll box for selection. Pushing the **Ok** button changes the `Visible` property of the selected objects to `'off'`. Pushing the **Cancel** button aborts the operation without changing any properties of the selected objects.

See Also

`hidem`

lightm

Purpose

Project light objects on the current map axes

Activation

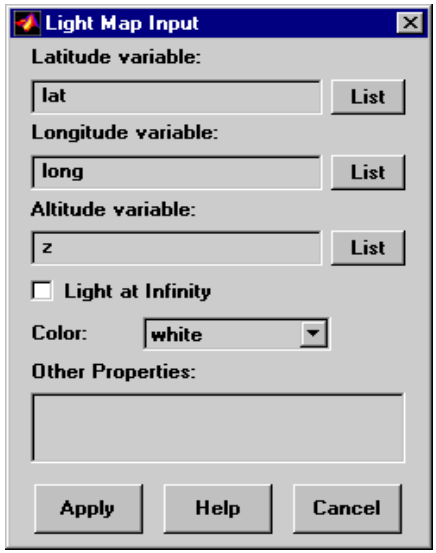
Command Line	Maptool
`lightm`	**Map⇒Light**

Description

`lightm` activates a **Light Map Input** dialog box for projecting a light object onto the current map axes.

If no map axes are current, a **No Map Axes** dialog box appears. Choose **Yes** to activate the **Projection Control** dialog box for defining map axes properties. Upon creation of the map axes, the **Light Map Input** dialog box appears.

Controls



The **Latitude variable** edit box is used to specify the workspace variable containing the latitude data of the light object to be projected.

The **Longitude variable** edit box is used to specify the workspace variable containing the longitude data of the light object to be projected.

The **Altitude variable** edit box is used to specify the workspace variable containing the altitude data of the light object to be projected. A scalar value can be entered to indicate at which height above the map the light object is to be displayed. This entry has no effect if an infinite light source is specified by the **Light at Infinity** check box.

Pressing the **List** button produces a list of all current workspace variables, from which the latitude, longitude, and altitude variables can be selected.

The **Light At Infinity** check box is used to specify a parallel or divergent light source. If the box is checked, the light source is at infinity, in which case the light rays are parallel. If the box is not checked, the altitude of the light source is specified by the altitude variable, and the light rays diverge in all directions. If this box is checked, the altitude variable has no effect.

The **Color** pull-down menu is used to specify the color of the light coming from the light object. Selecting `custom` allows a custom RGB triple to be defined.

The **Other Properties** edit box is used to specify additional properties of the light object to be projected, such as `'Tag','Blue Light'`. String entries must be enclosed in quotes.

Pressing the **Apply** button accepts the input data and projects the lighted object onto the current map axes.

Pressing the **Cancel** button disregards any input data and closes the **Light Map Input** dialog box.

See Also

`lightm`

limitm

Purpose

Compute latitude and longitude limits for a regular matrix map

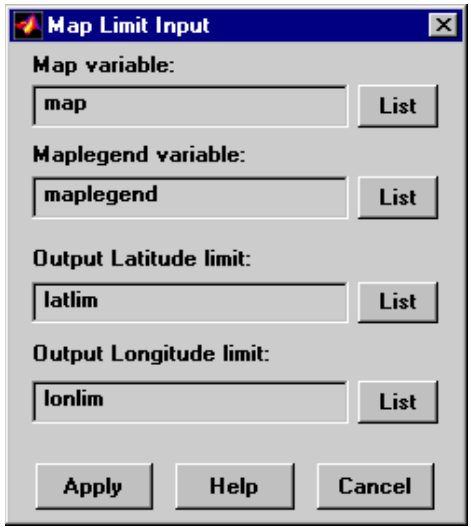
Activation

Command Line	Maptool
`limitm`	**Map⇒Contours**

Description

`limitm` activates the **Map Limit Input** dialog box, which allows the limits of a regular matrix map to be computed. These limits are then stored in the workspace as variables.

Controls



The **Map variable** edit box is used to specify the workspace variable containing the regular matrix map.

The **Maplegend variable** is used to specify the workspace variable containing the matrix map legend. A three-element map legend vector, enclosed in brackets, can be entered instead of a variable name.

The **Output Latitude limit** edit box is used to specify the name of the variable that stores the computed latitude limits of the matrix map. If this variable name already exists in the workspace, it is overwritten.

The **Output Longitude limit** edit box is used to specify the name of the variable that stores the computed longitude limits of the matrix map. If this variable already exists in the workspace, it is overwritten.

Pressing the **List** button produces a list of all current workspace variables, from which the map, maplegend, output latitude, and output longitude variables can be selected.

Pressing the **Apply** button calculates the limits of the matrix map and stores the results in the specified output variables.

Pressing the **Cancel** button disregards any input data and closes the **Map Limit Input** dialog box.

See Also

`limitm`

linem, plotm, plot3m

Purpose

Project 2-D and 3-D line objects on the current map axes

Activation

Command Line	Maptool

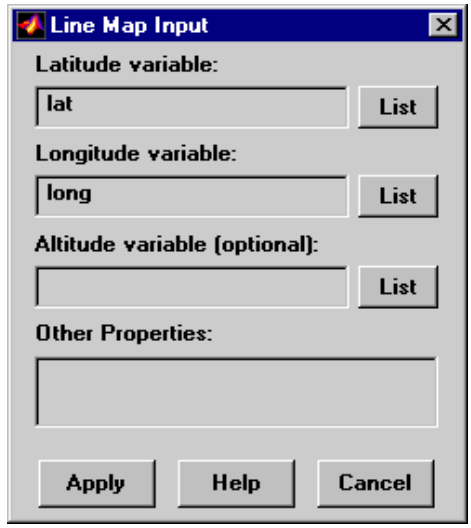
`linem`
`plotm`
`plot3m`
	Map⇒Lines

Description

`linem`, `plotm` and `plot3m` activate a **Line Map Input** dialog box that accepts input data to project a line object onto the current map axes.

If no map axes are current, a **No Map Axes** dialog box appears. Choose **Yes** to activate the **Projection Control** dialog box for defining map axes properties. Upon creation of the map axes, the **Line Map Input** dialog box appears.

Controls



The **Latitude variable** edit box is used to specify the workspace variable containing the latitude data of the line object to be projected.

The **Longitude variable** edit box is used to specify the workspace variable containing the longitude data of the line object to be projected.

The **Altitude variable** edit box is used to specify the workspace variable containing the altitude data of the line object to be projected. A scalar value can be entered to indicate the plane in which to display the object.

Pressing the **List** button produces a list of all current workspace variables, from which the latitude, longitude, and altitude variables can be selected.

The **Other Properties** edit box is used to specify additional properties of the line object to be projected, such as `'LineWidth',2`. String entries must be enclosed in quotes. Linespec strings, such as `'b:'`, are also valid.

Pressing the **Apply** button accepts the input data and projects the line object onto the current map axes.

Pressing the **Cancel** button disregards any input data and closes the **Line Map Input** dialog box.

See Also

`linem` `plotm` `plot3m`

maptool

Purpose

Create a figure window with a map axes and associated mapping tools

Activation

Command Line

`maptool`
`maptool(`*`PropertyName`*`,PropertyValue)`
`maptool(`*`ProjectionFile`*`,...)`
`h = maptool(...)`

Description

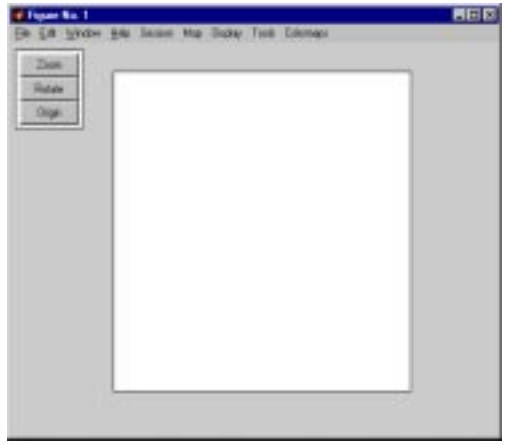
`maptool` creates a figure window with a map axes and activates the **Projection Control** dialog box for defining map projection and display properties. The figure window features a special menu bar that provides access to most of the Mapping Toolbox GUIs.

`maptool(`*`PropertyName`*`,PropertyValue,...)` creates a figure window with a map axes defined by the supplied map properties. The `MapProjection` property must be the first input pair. `maptool` supports the same map properties as `axesm`.

`maptool(`*`ProjectionFile`*`,`*`PropertyName`*`,PropertyValue,...)` allows for the omission of the `MapProjection` property name. *`ProjectionFile`* must be the identifying string of an available map projection.

`h = maptool(...)` returns a two-element vector containing the handle of the `maptool` figure window and the handle of the map axes.

Controls



Session Menu

The **Load** option is used to load workspace data. Select from the workspace names provided, or use the **Specify Workspace** option to enter a different workspace.

The **Layers** option is used to load a map layers workspace and activate the `mlayers` tool. Select from the workspace names provided, or use the **Other** option to enter a different workspace. Choosing **Workspace** loads all structure variables in the current workspace.

The **Renderer** option is used to set the renderer for the maptool figure window. The **Figure Renderer** dialog box is activated when this option is selected.

The **Variables** option is used to view the current workspace variables.

The **Command** option brings up the **Workspace Commands** dialog box for entering commands to operate on the current workspace.

The **Clear** option is used to clear variables and functions from memory.

Map Menu

The **Lines** option activates the **Line Map Input** dialog box for projecting two- and three-dimensional line objects onto the map axes.

The **Patches** option activates the **Patch Map Input** dialog box for projecting patch objects onto the map axes.

The **Regular Surfaces** option activates the **Mesh Map Input** dialog box for projecting a regular matrix map onto a graticule projected onto the map axes.

The **General Surfaces** option activates the **Surface Map Input** dialog box for projecting a general matrix map onto the map axes.

The **Comet** option activates the **Comet Map Input** dialog box for a projecting two- or three-dimensional comet plot onto the map axes.

The **Contours** option activates the **Contour Map Input** dialog box for projecting a two- or three-dimensional contour plot onto the map axes.

The **Quiver 2D** option activates the **Quiver Map Input** dialog box for projecting a two-dimensional quiver plot onto the map axes.

The **Quiver 3D** option activates the **Quiver3 Map Input** dialog box for projecting a three-dimensional quiver plot onto the map axes.

The **Stem** option activates the **Stem Map Input** dialog box for projecting a stem plot onto the map axes.

The **Scatter** option activates the **Scatter Map Input** dialog box for projecting a scatter plot onto the map axes.

The **Text** option activates the **Text Map Input** dialog box for projecting text objects onto the map axes.

The **Light** option activates the **Light Map Input** dialog box for projecting light objects onto the map axes.

Display Menu

The **Projection** option activates the **Projection Control** dialog box for editing map projection properties and map display settings.

The **Graticule** option is used to view and edit the graticule size for surface maps.

The **Legend** option is used to display a contour map legend.

The **Frame** option is used to toggle the map frame on and off.

The **Grid** option is used to toggle the map grid on and off.

The **Meridian Labels** option is used to toggle the meridian grid labels on and off.

The **Parallel Labels** option is used to toggle the parallel grid labels on and off.

The **Tracks** option activates the **Define Tracks** input box for calculating and displaying Great Circle and Rhumb Line tracks on the map axes.

The **Small Circles** option activates the **Define Small Circles** input box for calculating and displaying small circles on the map axes.

The **Surface Distances** option activates the **Surface Distance** dialog box for distance, azimuth, and reckoning calculations.

Tools Menu

The **Hide** option is used to hide the mouse tool buttons.

The **Off** option is used to turn off the current mouse tool.

The **Zoom Tool** option is used to toggle Panzoom (`panzoom`) mode on and off. It is used for zooming in on a two-dimensional map display.

The **Set Limits** option is used to define the zoom out limits to the current settings on the axes.

The **Full View** option is used to zoom out to the current axes limit settings.

The **Rotate** option is used to toggle Rotate 3-D (`rotate3d`) mode on and off. Rotate 3-D mode is used to interactively rotate the view of a three-dimensional plot.

The **Origin** option is used to toggle Origin (`originui`) mode on and off. Origin mode is used to interactively modify the map origin.

The **2D View** option is used to set the default two-dimensional view (azimuth=0, elevation=90).

The **Objects** option activates the **Object Sets** dialog box, which allows for property manipulation of objects displayed on the map axes.

The **Edit** option activates the Guide Property Editor to manipulate properties of a plotted object. Choose from the **Current Object** or **Last Object** options, or choose the **Object** option to activate the **Select Object** dialog box.

The **Show** option is used to set the `Visible` property of mapped objects to `'on'`. The **All** option shows all currently mapped objects. The **Object** option activates the **Select Object** dialog box.

The **Hide** option is used to set the `Visible` property of mapped objects to `'off'`. Choose from the **All** or **Map** options, or choose the **Object** option to activate the **Select Object** dialog box.

The **Delete** option is used to clear the selected objects. The **All** option clears the current map, frame, and grid lines. The map definition is left in the axes definition. The **Map** option clears the current map, deleting objects plotted on the map but leaving the frame and grid lines displayed. The **Object** option activates the **Select Object** dialog box.

The **Axes** option is used to manipulate the MATLAB Cartesian axes. The **Show** option shows this axes, the **Hide** option hides this axes, and the **Color** option allows for custom color selection for this axes.

Colormaps Menu

The **Colormaps** menu allows for manipulation of the colormap for the current figure. See the `clrmenu` reference page for details on the **Colormaps** menu options.

Zoom Button

The Zoom button toggles Zoom mode on and off. Zoom mode is used for zooming in on a two-dimensional map display.

Rotate Button

The Rotate button toggles Rotate 3-D mode on and off. Rotate 3-D mode is used to interactively rotate the view of a three-dimensional plot.

Origin Button

The **Origin** button toggles Origin mode on and off. Origin mode is used to interactively modify the map origin.

See Also

`maptool tools`

maptrim

Purpose

Interactively trim and convert map data from vector to matrix format

Activation

Command Line

`maptrim(lat,lon)`
`maptrim(lat,lon,`*`linespec`*`)`
`maptrim(map,maplegend)`
`maptrim(map,maplegend,`
`  `*`PropertyName`*`,PropertyValue,...)`

Description

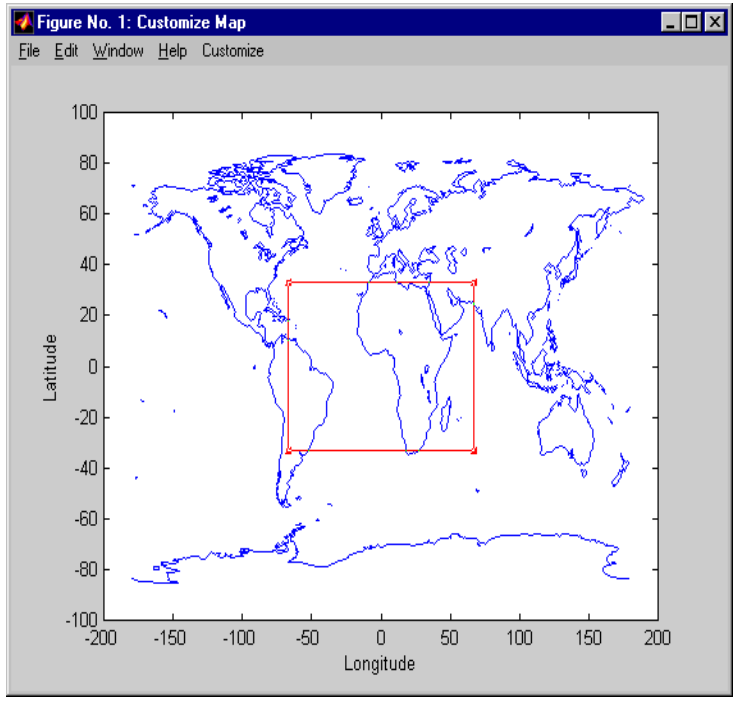
`maptrim(lat,lon)` displays the supplied map data in a new figure window and allows a region of the map to be selected and saved in the workspace. `lat` and `lon` must be vector map data. The output can be line, patch, or regular surface (matrix) data. If patch map output is selected, the inputs `lat` and `lon` must originally be patch map data.

`maptrim(lat,lon,`*`linespec`*`)` displays the supplied map data using the *`linespec`* string.

`maptrim(map,maplegend)` displays matrix map data in a new figure window and allows a subset of this map to be selected and saved. The output is regular surface data.

`maptrim(map,maplegend,`*`PropertyName`*`,PropertyValue)` displays the matrix map data using the surface properties provided. The object `Tag`, `EdgeColor`, and `UserData` properties cannot be set.

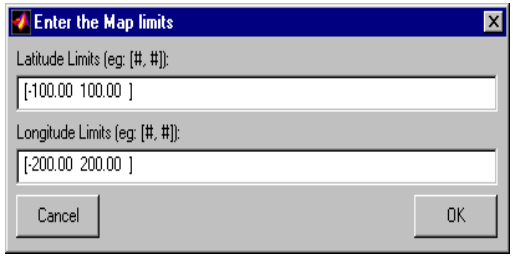
Controls



The `maptrim` tool displays the supplied map data in a new figure window and activates a **Customize** menu for that figure. The **Customize** menu has three menu options: **Zoom On/Off**, **Limits**, and **Save As**.

The **Zoom On/Off** menu option toggles the `panzoom` box on and off. The box can be moved by clicking on the new location or by dragging the box to the new location. The box size can be increased or decreased by dragging a corner of the box. Pressing the `Return` key or double-clicking in the center of the box zooms in.

The **Limits** menu option activates the **Enter Map Limits** dialog box, which is used to enter the latitude and longitude limits of the desired map subset. These entries are two-element vectors, enclosed in brackets. Pressing the **OK** button zooms in to the new limits. Pressing the **Cancel** button disregards the new limits and returns to the map display.



The **Save As** menu option is used to specify the variable names in which to save the map data subset. To save line and patch data, enter the new latitude and longitude variable names, along with the map resolution. For surface data, enter the new map and map legend variable names, along with the scale of the map. Latitude and longitude limits are optional.

See Also

`maptriml` `maptrimp` `maptrims` `panzoom`

Purpose

Display a regular matrix map warped to a projected graticule

Activation

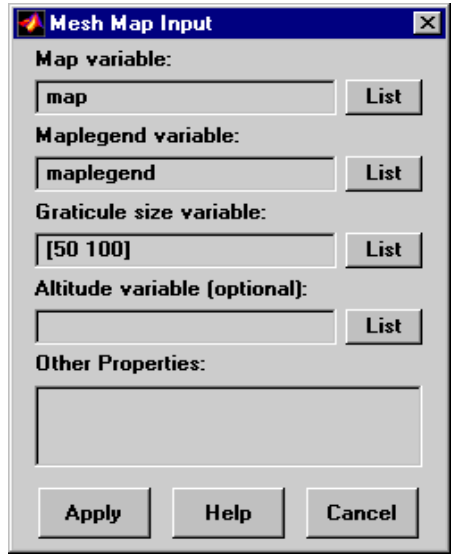
Command Line	Maptool
`meshm`	**Map⇒Regular Surfaces**

Description

`meshm` activates a **Mesh Map Input** dialog box that accepts input data to project a regular surface onto the current map axes.

If no map axes are current, a **No Map Axes** dialog box appears. Choose **Yes** to activate the **Projection Control** dialog box for defining map axes properties. Upon creation of the map axes, the **Mesh Map Input** dialog box appears.

Controls



The **Map variable** edit box is used to specify the workspace variable containing the matrix map.

The **Maplegend variable** edit box is used to specify the workspace variable containing the matrix map legend. Alternatively, a three-element map legend vector enclosed in brackets can be entered in place of a workspace variable.

The **Graticule size variable** edit box is used to specify the workspace variable containing the graticule resolution. A two-element vector of the form `[latitude-points longitude-points]` can be entered in place of a workspace variable. The default graticule resolution is `[50 100]`.

The **Altitude variable** edit box is used to specify the workspace variable containing the altitude data. A scalar value can be entered to specify the *z*-axis plane in which the graticule mesh is plotted.

Pressing the **List** button produces a list of all current workspace variables, from which the latitude, longitude, graticule size, and altitude variables can be selected.

The **Other Properties** edit box is used to specify additional properties of the surface object to be projected, such as `'EdgeColor',[1 1 0]`. String entries must be enclosed in quotes. The `CData` property contains the matrix map and therefore cannot be set by users.

Pressing the **Apply** button accepts the input data and projects the surface object onto the current map axes.

Pressing the **Cancel** button disregards any input data and closes the **Mesh Map Input** dialog box.

See Also

`meshm`

Purpose

Interactively display and control objects in a geographic data structure workspace

Activation

Command Line	Maptool

`mlayers(`*`workspace`*`)`
`mlayers(`*`workspace`*`,h)`
`mlayers(cellarray)`
`mlayers(cellarray,h)`
	Session⇒Layers

Description

The `mlayers` tool activates a dialog box for the specified geographic data structure *workspace*, which enables display and manipulation of the map objects that it comprises.

`mlayers(`*`workspace`*`)` associates the geographic data structures, which in this context are also called map layers, in the *workspace* MAT-file with the current map axes. The geographic data structure variables are accessible only through the `mlayers` tool, and not through the base workspace. *workspace* must be a string.

`mlayers(`*`workspace`*`,h)` assigns the layers in *workspace* to the map axes indicated by the handle `h`.

`mlayers(cellarray)` associates the layers specified by `cellarray` with the current map axes. `cellarray` must be of size n by 2. Each row of `cellarray` represents a map layer. The first column of `cellarray` contains the layer structure, and the second column contains the name of the layer structure. Such a cell array can be generated from data in the current workspace with the function `rootlayr`. In this case, the calling sequence would be `rootlayr; mlayers(ans)`.

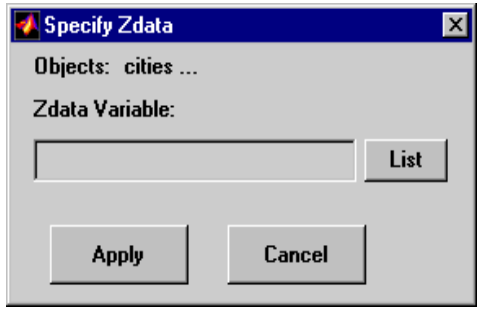
`mlayers(cellarray,h)` assigns the layers specified by `cellarray` to the map axes specified by the handle `h`.

Controls

The scrollable list box displays all of the map layers currently associated with the map axes. An asterisk next to the layer name indicates that the layer is currently visible. An h next to the layer name indicates a layer that is plotted, but currently hidden.

The **Plot** button plots the selected map layer. Once the selected layer is plotted, the button toggles between **Hide** and **Show**, to turn the `Visible` property of the plotted objects to `'off'` and `'on'`, respectively.

The **Zdata** button activates the **Specify Zdata** dialog box, which is used to enter the workspace variable containing the `ZData` for the selected map layer. Pressing the **List** button produces a list of all current workspace variables, from which the `ZData` variable can be selected. This entry can also be a scalar.

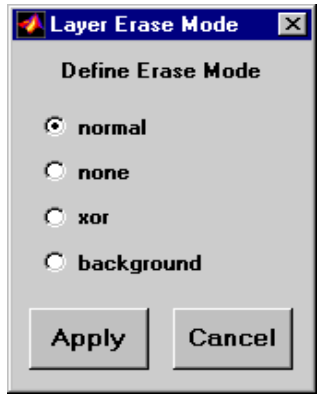


The **Highlight** button is used to toggle the selected map layer between highlighted and normal display.

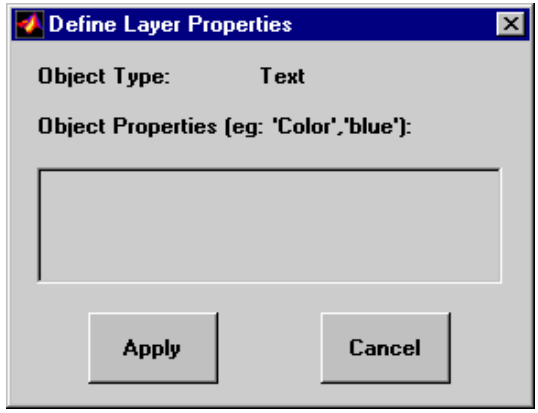
The **Members** button brings up a list of members of the selected map layer. Members of a layer are defined by their Tag property.

The **Delete** button deletes the selected map layer from the map.

The **Emode** button activates the **Layer Erase Mode** dialog box, which is used to specify the erase mode of the selected map layer.



The **Property** button activates the **Define Layer Properties** dialog box, which is used to specify or change properties of all objects in the selected map layer. String entries must be enclosed in single quotes.



The **Purge** button deletes the selected map layer from the `mlayers` tool. Selecting **Yes** from the **Confirm Purge** dialog box deletes the map layer from both the `mlayers` tool and the map display. Selecting **Data Only** from the **Confirm Purge** dialog box deletes the map layer from the `mlayers` tool, while retaining the plotted object on the map display.

See Also

`geographic data structure` `mobjects` `rootlayr`

Purpose

Manipulate object sets plotted on a map axes

Activation

Command Line	Maptool
`mobjects` `mobjects(h)`	**Tools⇒Objects**

Description

An object set is defined as all objects with identical tags. If no tags are supplied, object sets are defined by object type.

`mobjects` allows manipulation of the object sets on the current map axes.

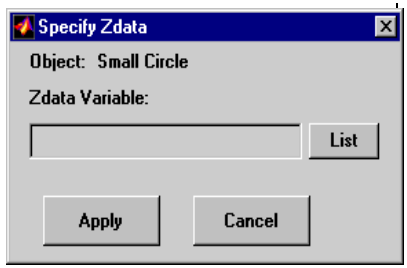
`mobjects(h)` allows manipulation of the objects set on the map axes specified by the handle `h`.

Controls

The scrollable list box displays all of the object sets associated with the map axes. An asterisk next to an object set name indicates that the object set is currently visible. An `h` next to an object set name indicates an object set that is plotted, but currently hidden. The order shown in the list indicates the stacking order of objects within the same plane.

The **Hide/Show** button toggles the `Visible` property of the selected object set to `'off'` and `'on'`, respectively, depending on the current `Visible` status.

The **Zdata** button activates the **Specify Zdata** dialog box, which is used to enter the workspace variable containing the `ZData`. The `ZData` property is used to specify the plane in which the selected object set is drawn. Pressing the **List** button produces a list of all current workspace variables, from which the `ZData` variable can be selected. Alternatively, a scalar value can be entered instead of a variable.

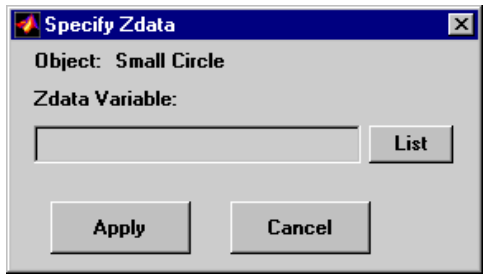


The **Highlight** button highlights all objects belonging to the selected object set.

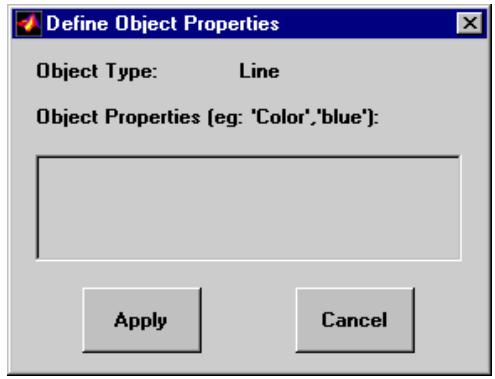
The **Tag** button brings up an **Edit Tag** dialog box, which allows the tag of all members of the selected object set to be modified.

The **Delete** button clears all objects belonging to the selected object set from the map. The cleared object set remains associated with the map axes.

The **Emode** button activates the **Object Erase Mode** dialog box, which is used to specify the erase mode of the selected object set.



The **Property** button activates the **Define Object Properties** dialog box, which is used to specify additional properties of all objects in the selected object set. String entries must be enclosed in single quotes.



The **Update** button updates the list box display with current objects sets.

The **Stacking Order** buttons are used to modify the drawing order of the selected object set in relation to other plotted object sets in the same plane. Objects drawn first appear at the bottom of the stack, and objects drawn last appear at the top of the stack. The **Top** button places the selected object set above all other object sets in its plane. The **Up** and **Dwn** buttons move the selected object set up and down one place in the stacking order, respectively. The **Btm** button places the selected object set below all other object sets in its plane. Note that the `ZData` property overrides stacking order, i.e., if an object is at the top of the stacking order for its plane, it can still be covered by an object drawn in a higher plane.

See Also

`mlayers`

originui

Purpose

Modify a map origin

Activation

Command Line	Maptool

`originui`
`originui on`
`originui off`
	Tools⇒Origin (menu)
	Origin (button)

Description

`originui` toggles the origin tool on and off.

`originui on` activates the origin tool.

`originui off` deactivates the origin tool.

The `originui` tool allows for interactive modification of the origin of a displayed map projection. A dot marker appears on the map at the location of the current origin. This dot can be moved using the mouse or keyboard, and the map can be reprojected with the location of the dot as the new origin.

Controls

Mouse Interaction

A single-click and drag moves the dot marker. A double-click on the dot marker reprojects the map with the dot location as the new origin. An extend-click moves the dot marker in the x or y direction only, depending on the direction of greatest mouse movement. Alternate-click exits the origin tool.

Keyboard Interaction

The following keyboard interaction is enabled if the figure containing the map axes is made the active window.

The `n`, `s`, `e`, and `w` keys move the dot marker north (up), south (down), east (right), and west (left) by one degree per keystroke. `N`, `S`, `E`, `W` keys move the dot marker ten degrees per keystroke in the same directions. Pressing the `Return` key reprojects the map with the dot location as the new origin. Pressing the **Enter** key reprojects the map with the new origin and exits the origin tool. Pressing the `Esc` or `Delete` keys exits the origin tool.

See Also

`axesm` `setm`

Purpose

Pan and zoom on a 2-D map display

Activation

Command Line	Maptool

`panzoom`
`panzoom on`
`panzoom off`
`panzoom setlimits`
`panzoom out`
`panzoom fullview`
	Tools⇒Zoom Tool (menu)
	Zoom (button)

Description

`panzoom` toggles the pan and zoom tool on and off.

`panzoom on` activates the pan and zoom tool.

`panzoom off` deactivates the pan and zoom tool.

`panzoom setlimits` sets the zoom out limits to the current settings on the map axes.

`panzoom out` zooms out to the current map axes limit settings.

`panzoom fullview` resets the axes to their full view range and resets the pan and zoom tool with these settings.

The pan and zoom tool provides an interactive means of defining zoom limits on a two-dimensional map display. A box that can be resized and moved appears on the map display and is used to define the zoom area. The box cannot be moved beyond the current axes limits.

Controls

Mouse Interaction

With the cursor inside the zoom box, a single-click and drag moves the box. The zoom box can be resized by dragging the corners of the box. A double-click in the center of the box zooms in to the current boundaries of the box. A single-click outside the zoom box moves the box to that location. An extend-click inside or outside of the zoom box zooms out by a factor of two. Alternate-click exits the pan and zoom tool.

Keyboard Interaction

The following keyboard interaction is enabled if the figure containing the map axes is made the active window.

Pressing the `Return` key sets the axes to the current zoom box and remains in pan and zoom mode. The `Enter` key sets the axes to the current zoom box and exits pan and zoom mode. Pressing the `Esc` or `Delete` keys exits pan and zoom mode.

See Also

`zoom`

Purpose

Modifying map parallels

Activation

Command Line	Maptool

`parallelui`
`parallelui on`
`parallelui off`
	Tools⇒Parallels (menu)

Description

`parallelui` toggles the parallel tool on and off.

`parallelui on` activates the parallel tool

`parallelui off` deactivates the parallel tool

The `parallelui` GUI provides a tool to modify the standard parallels of a displayed map projection. One or two red lines are displayed where the standard parallels are currently located. The parallel lines can be dragged to new locations, and the map reprojected with the locations of the parallel lines as the new standard parallels.

Controls

Mouse Interaction

A single-click-and-drag moves the parallel lines. A double-click on one of the standard parallels reprojects the map using the new parallel locations.

See Also

`axesm` `setm`

pcolorm, surfacem, surfm

Purpose

Project a general matrix map onto the current map axes

Activation

Command Line	Maptool

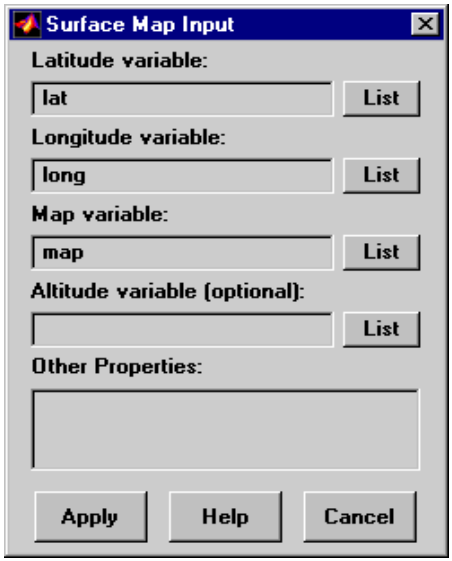
`pcolorm`
`surfacem`
`surfm`
	Map⇒General Surfaces

Description

`pcolorm`, `surfacem`, and `surfm` activate a **Surface Map Input** dialog box for projecting general surfaces onto the current map axes.

If no map axes are current, a **No Map Axes** dialog box appears. Choose **Yes** to activate the **Projection Control** dialog box for defining map axes properties. Upon creation of the map axes, the **Surface Map Input** dialog box appears.

Controls



The **Latitude variable** edit box is used to specify the workspace variable containing the latitude data of the surface to be projected.

The **Longitude variable** edit box is used to specify the workspace variable containing the longitude data of the surface to be projected.

The **Map variable** edit box is used to specify the workspace variable containing the matrix map.

The **Altitude variable** edit box is used to specify the workspace variable containing the altitude data of the surface to be projected. A scalar value can be entered to indicate the plane in which to display the object.

Pressing the **List** button produces a list of all current workspace variables, from which the latitude, longitude, map, and altitude variables can be selected.

The **Other Properties** edit box is used to specify additional properties of the surface object to be projected, such as `'EdgeColor',[1 1 0]`. String entries must be enclosed in single quotes.

Pressing the **Apply** button accepts the input data and projects the surface object onto the current map axes.

Pressing the **Cancel** button disregards any input data and closes the **Surface Map Input** dialog box.

See Also

`pcolorm` `surfacem` `surfm`

property editors

Purpose

Edit properties of mapped objects using display-activated property editors

Activation

map display:	alternate-click on mapped object (for **Click-and-Drag Property Editor**) double-click on mapped object (for MATLAB **Guide Property Editor**)
maptool:	**Tools, Edit** menu item (for MATLAB **Guide Property Editor**)

Description

Alternate-clicking on a mapped object activates a property editor, which allows modification of some basic properties of the object through simple mouse clicks and drags. The objects supported by this editor are map axes, lines, text, patches, and surfaces, and the properties supported for each object type are shown below.

Double-clicking on a mapped object activates the MATLAB Guide Property Editor for that object. The Guide Property Editor provides a complete list of the properties and values of the selected object, allowing for modifications of the object.

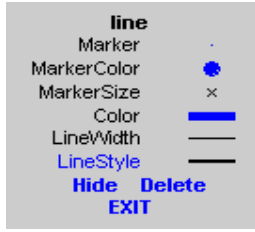
Controls

Click-and-Drag Property Editor

The **Click-and-Drag Editor** lists object properties and values. The object tag appears at the top of the editor. Property names and values that appear in blue are toggles. For example, clicking **Frame** in the axes editor toggles the value of the `Frame` property between `'on'` and `'off'`.

Click-and-Drag Editor for a map axes

Property values that appear on the right side of the editor box are modified by clicking and dragging. For example, to change the `MarkerColor` property of a line object, click and hold the dot next to **MarkerColor**, and drag the cursor until the dot appears in the desired color.

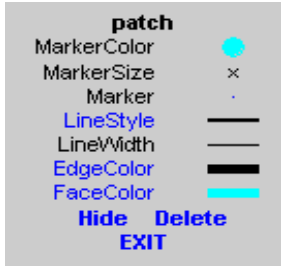


Click-and-Drag Editor for a line object

The **Drag** control in the text editor is used to reposition the text string. In drag mode, use the mouse to move the text to a new location, and click to reposition the text. The **Edit** control in the text editor activates a **Text Edit** window, which is used to modify text.

Click-and-Drag Editor for a text object

The **Marker** property name in the patch editor is used to toggle the marker on and off. The property value to the right of **Marker** can be modified by clicking and dragging until the desired marker symbol appears.



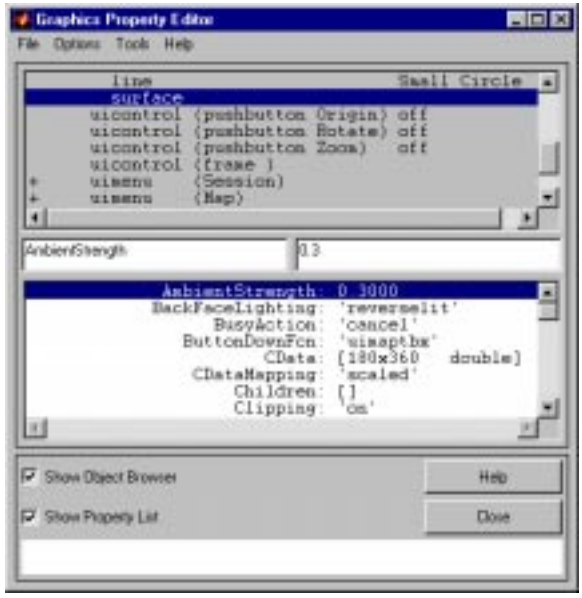
Click-and-Drag Editor for a patch object

The **Graticule** control on the surface editor activates a **Graticule Mesh** dialog box, which is used to alter the size of the graticule.

To move the property editor around the figure window, hold down the **Shift** key while dragging the editor box. Alternate-clicking on the background of the property editor closes the **Click-and-Drag** editing session.

Guide Property Editor

The MATLAB **Guide Property Editor** allows for modification of property values for all properties applicable to the selected object. The **Object Browser** is used to expand and collapse the hierarchy of objects, showing an object's parents and children. A plus sign (+) before an object indicates that the object can be expanded to show its children. A minus sign (-) before an object indicates an object can be collapsed to hide its children. To activate the Object Browser, check the **Show Object Browser** check box. The **Property List** shows all the property names of the selected object and their current values. To activate the **Property List**, check the **Show Property List** check box. To change a property value, use the edit boxes above the Property List. Pressing the **Close** button closes the **Guide Property Editor** and applies the property modifications to the object.



MATLAB Guide Property Editor for a surface object

See Also

`propedit` `guide` `uimaptbx`

qrydata

Purpose

Interactively perform data queries

Activation

Command Line
`qrydata(cellarray)`
`qrydata(`*`titlestr`*`,cellarray)`
`qrydata(h,cellarray)`
`qrydata(h,`*`titlestr`*`,cellarray)`
`qrydata(...,cellarray1,cellarray2,...)`

Description

A data query is used to obtain the data corresponding to a particular (x,y) or (lat,lon) point on a standard or map axes.

`qrydata(cellarray)` activates a data query dialog box for interactive queries of the data set specified by `cellarray` (described below). `qrydata` can be used on a standard axes or a map axes. `(x,y)` or `(lat,lon)` coordinates are entered in the dialog box, and the data corresponding to these coordinates is then displayed.

`qrydata(titlestr,cellarray)` uses the string *`titlestr`* as the title of the query dialog box.

`qrydata(h,cellarray)` and `qrydata(h,`*`titlestr`*`,cellarray)` associate the data queries with the axes specified by the handle `h`, which in turn allows the input coordinates to be specified by clicking on the axes.

The input `cellarray` is used to define the data set and the query. The first cell must contain the string used to label the data display line. The second cell must contain the type of query operation, either a pre-defined operation or a valid user-defined function name. This input must be a string. The pre-defined query operations are `'matrix'`, `'vector'`, `'mapmatrix'`, and `'mapvector'`.

The `'matrix'` query uses the MATLAB `interp2` function to find the value of the matrix `Z` at the input `(x,y)` point. The format of the `cellarray` input for this query is: `{'label','matrix',X,Y,Z,`*`method`*`}`. `X` and `Y` are matrices specifying the points at which the data `Z` is given. The rows and columns of `X` and `Y` must be monotonic. *`method`* is an optional argument that specifies the interpolation method. Possible *`method`* strings are `'nearest'`, `'linear'`, or `'cubic'`. The default is `'nearest'`.

The 'vector' query uses the MATLAB interp2 function to find the value of the matrix Z at the input (x,y) point, then uses that value as an index to a data vector. The value of the data vector at that index is returned by the query. The format of cellarray for this type of query is: {'label','vector',X,Y,Z, vector}. X and Y are matrices specifying the points at which the data Z is given. The rows and columns of X and Y must be monotonic. vector is the data vector.

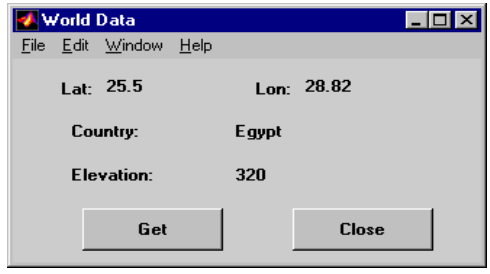
The 'mapmatrix' query interpolates to find the value of the map at the input (lat,lon) point. The format of cellarray for this query is: {'label','mapmatrix',map,maplegend,*method*}. map and maplegend are the matrix map and the corresponding matrix map legend. *method* is an optional argument that specifies the interpolation method. Possible *method* strings are 'nearest', 'linear', or 'cubic'. The default is 'nearest'.

The 'mapvector' query interpolates to find the value of the map at the input (lat,lon) point, then uses that value as an index to a data vector. The value of the vector at that index is returned by the query. The format of cellarray for this type of query is {'label','mapvector',map,maplegend, vector}. map and maplegend are the matrix map and the corresponding matrix map legend. vector is the data vector.

User-defined query operations allow for functional operations using the input (x,y) or (lat,lon) coordinates. The format of cellarray for this type of query is {'label',*function*,other arguments...} where the other arguments are the remaining elements of cellarray as in the four pre-defined operations above. *function* is a user-created function and must refer to an M-file of the form z = fcn(x,y,other_arguments...).

qrydata(...,cellarray1,cellarray2,...) is used to input multiple cell arrays. This allows more than one data query to be performed on a given point.

Controls



Sample data query dialog box

If an axes handle `h` is not provided, or if the axes specified by `h` is not a map axes, the currently selected point is labeled as **Xloc** and **Yloc** at the top of the query dialog box. If `h` is a map axes, the current point is labeled as **Lat** and **Lon**. Displayed below the current point are the results from the queries, each labeled as specified by the `'label'` input arguments.

The **Get** button appears if an axes handle `h` is provided. Pressing this button activates a mouse cursor, which is used to select the desired point by clicking on the axes. Once a point is selected, the queries are performed and the results are displayed.

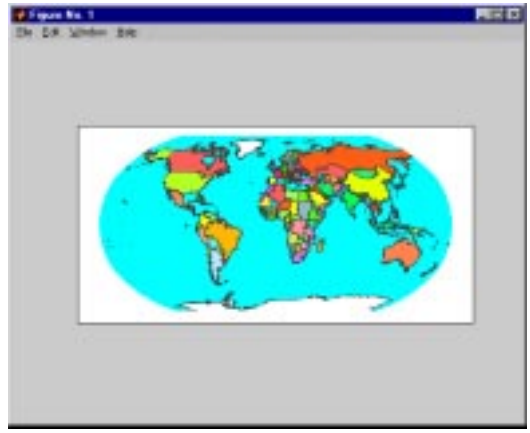
The **Process** button appears if the handle `h` is not provided. In this case, the `(x,y)` coordinates of the desired point are entered into the edit boxes. Pressing the **Process** button performs the data queries and displays the results.

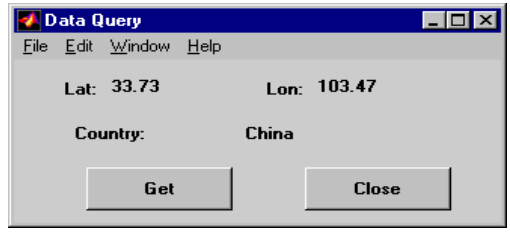
Pressing the **Close** button closes the query dialog box.

Examples

The following example can be found in the `wrlddemo` query demo provided with the Mapping Toolbox. The call to `qrydata` creates a query dialog box that enables the user to click on the map display and retrieve the name of the country corresponding to that point.

```
load worldmtx
axesm robinson
colormap(clrmap)
meshm(map,maplegend)
qrydata(gca,'Data Query',{'Country:','mapvector',...
        map,maplegend,strvcat(nations(:).name)})
```



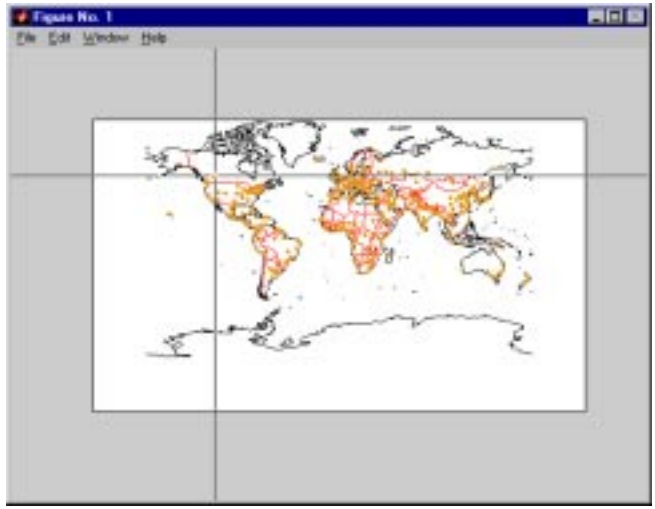


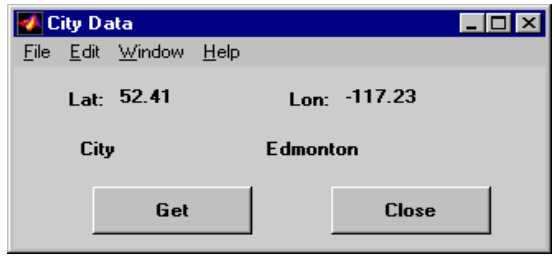
The next example makes use of a user-defined query to display city names for map points specified by a mouse click.

```
axesm miller
load worldlo
lat = [PPtext.lat]';
lon = [PPtext.long]';
mat = strvcat(PPtext.string);
displaym(POline)
displaym(PPpoint)
qrydata(gca,'City Data',{'City','qrytest',lat,lon,mat})
```

The following code would be contained in the M-file qrytest, created by the user.

```
% function QRYTEST returns city name for mouse click
% QRYTEST will find the closest city (min radius) from
% the mouse click, within an angle of 5 degrees.
%
latdiff=lt-lat;londiff=lg-lon;
rad = sqrt(latdiff.^2+londiff.^2);
[minrad,index]=min(rad);

if minrad > 5; index = []; end;

switch length(index)

  case 0, cityname='No city located near click';
  case 1, cityname=mat(index,:);
  end
```





Clicking the mouse over a city marker displays the name of the selected city. Clicking the mouse in an area away from any city markers displays the string `'No city located near click'`.

See Also

`interp2` `qrydemo` `wrlddemo`

quiver3m

Purpose

Project a 3-D quiver plot onto the current map axes

Activation

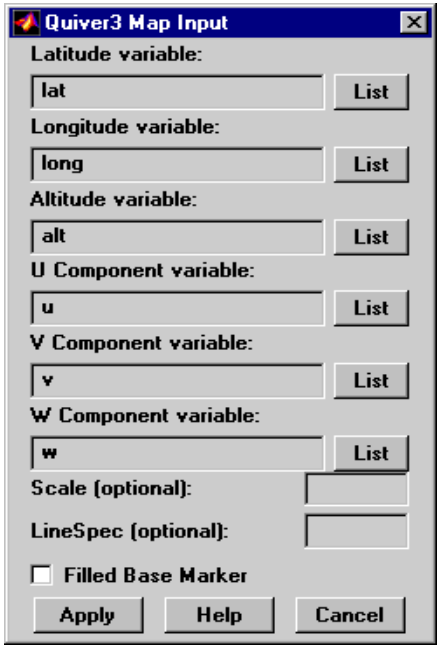
Command Line	Maptool
`quiver3m`	**Map⇒Quiver 3D**

Description

`quiver3m` activates a **Quiver3 Map Input** dialog box to project a three-dimensional quiver plot onto the current map axes. The vectors `(u,v,w)` are displayed at the points `(latitude,longitude,altitude)` on the map.

If no map axes are current, a **No Map Axes** dialog box appears. Choose **Yes** to activate the **Projection Control** dialog box for defining map axes properties. Upon creation of the map axes, the **Quiver3 Map Input** dialog box appears.

Controls



The **Latitude variable** edit box is used to specify the workspace variable containing the latitude data for the quiver plot.

The **Longitude variable** edit box is used to specify the workspace variable containing the longitude data for the quiver plot.

The **Altitude variable** edit box is used to specify the workspace variable containing the altitude data for the quiver plot.

The **U Component variable** edit box is used to specify the workspace variable containing the u vector component data.

The **V Component variable** edit box is used to specify the workspace variable containing the v vector component data.

The **W Component variable** edit box is used to specify the workspace variable containing the w vector component data.

Pressing the **List** button produces a list of all current workspace variables, from which the latitude, longitude, altitude, u, v, and w variables can be selected.

The **Scale** edit box is used to enter the workspace variable containing the scale factor applied to the projected vectors. The vector lengths are automatically determined to make them as long as possible without overlapping. The vector lengths are then multiplied by `scale`. A `scale` of 0.5 results in vectors half as long as they would be with the default `scale` of 1. A `scale` of 0 suppresses automatic scaling, and the vector lengths are determined from the inputs. In this case, the vectors are plotted from `(latitude,longitude,altitude)` to `(latitude+u,longitude+v,altitude+w)`. A scalar value for `scale` can be entered instead of a variable name.

The **Linespec** edit box is used to enter a line specification, such as `'-r*'`, for the quiver plot. If a symbol is given in the linespec string, it is plotted at the beginning of the vectors. If no symbol is given in the linespec string, arrows are plotted at the end of the vectors.

The **Filled Base Marker** check box is used to specify a filled-in symbol at the beginning of each vector.

Pressing the **Apply** button accepts the input data and projects the quiver plot onto the current map axes.

Pressing the **Cancel** button disregards any input data and closes the **Quiver3 Map Input** dialog box.

See Also

`quiver3m`

quiverm

Purpose

Project a 2-D quiver plot onto the current map axes

Activation

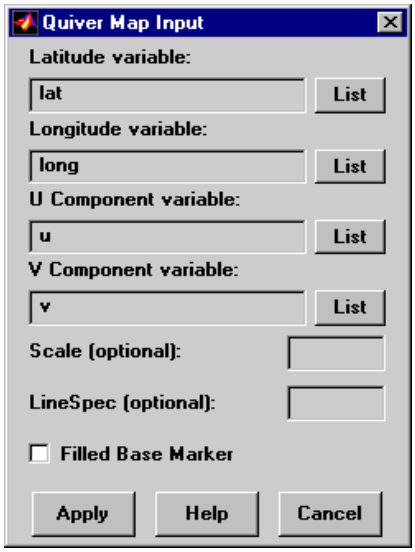
Command Line	Maptool
`quiverm`	**Map⇒Quiver 2D**

Description

`quiverm` activates a **Quiver Map Input** dialog box to project a two-dimensional quiver plot onto the current map axes. Vectors with components `(u,v)` are displayed at the points `(latitude,longitude)` on the map.

If no map axes are current, a **No Map Axes** dialog box appears. Choose **Yes** to activate the **Projection Control** dialog box for defining map axes properties. Upon creation of the map axes, the **Quiver Map Input** dialog box appears.

Controls



The **Latitude variable** edit box is used to specify the workspace variable containing the latitude data for the quiver plot.

The **Longitude variable** edit box is used to specify the workspace variable containing the longitude data for the quiver plot.

The **U Component variable** edit box is used to specify the workspace variable containing the u vector component data.

The **V Component variable** edit box is used to specify the workspace variable containing the v vector component data.

Pressing the **List** button produces a list of all current workspace variables, from which the latitude, longitude, u, and v variables can be selected.

The **Scale** edit box is used to enter the workspace variable containing the scale factor applied to the projected vectors. The vector lengths are automatically determined to make them as long as possible without overlapping. The vector lengths are then multiplied by `scale`. For example, a `scale` value of 0.5 results in vectors half as long as they would be with the default `scale` of 1. A `scale` of 0 suppresses automatic scaling, and the vector lengths are determined from the inputs. In this case, the vectors are plotted from `(latitude,longitude)` to `(latitude+u,longitude+v)`. A scalar value for `scale` can be entered instead of a variable name.

The **Linespec** edit box is used to enter a line specification, such as `'-r*'`, for the quiver plot. If a symbol is given in the linespec string, it is plotted at the beginning of the vectors. If no symbol is given in the linespec string, arrows are plotted at the end of the vectors.

The **Filled Base Marker** check box is used to specify a filled-in symbol at the beginning of each vector.

Pressing the **Apply** button accepts the input data and projects the quiver plot onto the current map axes.

Pressing the **Cancel** button disregards any input data and closes the **Quiver Map Input** dialog box.

See Also

`quiverm`

scatterm

Purpose

Project a symbol map on the current map axes

Activation

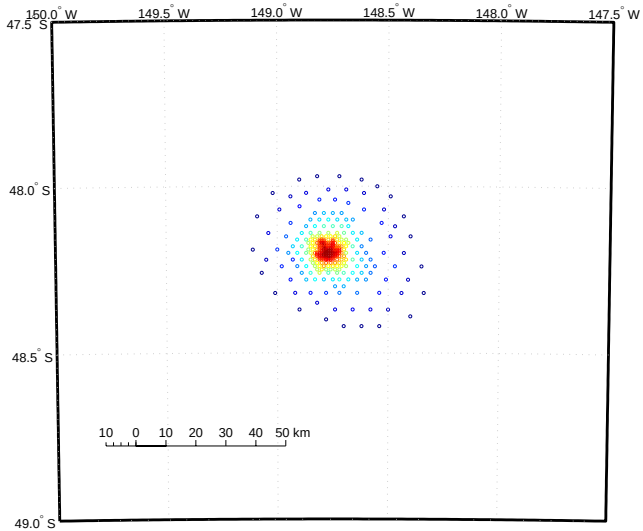
Command Line	Maptool
`scatterm`	**Map⇒Scatter**

Description

`scatterm` activates a **Scatter Map Input** dialog box to project a symbol plot onto the current map axes. A symbol map displays symbols proportionally sized to the data.

If no map axes are current, a **No Map Axes** dialog box appears. Choose **Yes** to activate the **Projection Control** dialog box for defining map axes properties. Upon creation of the map axes, the **Scatter Map Input** dialog box appears.

Controls



The **Latitude variable** edit box is used to specify the workspace variable containing the latitude coordinates for the scatter plot.

The **Longitude variable** edit box is used to specify the workspace variable containing the longitude coordinates for the scatter plot.

The **Marker size variable** edit box is used to specify the workspace variable containing the marker weights. The markers areas proportionally sized based on these weights. The marker size can also be a scalar, which is applied to all markers.

The **Marker Color Variable** edit box is used to specify the workspace variable containing the marker color data. The marker color data is linearly mapped to the colors in the colormap. The marker color data can also be a vector of RGB values or a color string.

Pressing the **List** button produces a list of all current workspace variables, from which the latitude, longitude marker size, and color variables can be selected.

The **Marker Style** popup menu is used to select the marker type..

The **Filled** check box is used to select unfilled (the default) or filled markers.

Pressing the **Apply** button accepts the input data and projects the scatter plot onto the current map axes.

Pressing the **Cancel** button disregards any input data and closes the **Scatter Map Input** dialog box.

See Also

`scatterm`

scirclui

Purpose

Display small circles on a map axes

Activation

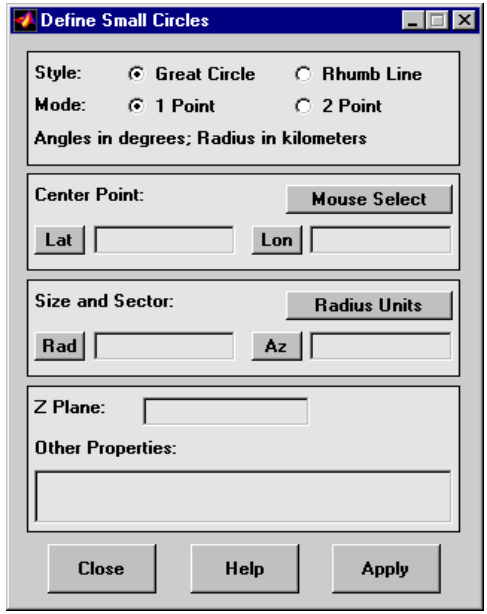
Command Line	Maptool
`scirclui` `scirclui(h)`	**Display⇒Small Circles**

Description

`scirclui` activates the **Define Small Circles** dialog box for adding small circles to the current map axes.

`scirclui(h)` activates the **Define Small Circles** dialog box for adding small circles to the map axes specified by the axes handle `h`.

Controls



Define Small Circles dialog box for one-point mode

The **Style** selection buttons are used to specify whether the circle radius is a constant great circle distance or a constant rhumb line distance.

The **Mode** selection buttons are used to specify whether one point or two points are to be used in defining the small circle. If one-point mode is selected, a center point, radius, and azimuth are the required inputs. If two-point mode is selected, a center point, and perimeter point on the circle are the required inputs.

The **Center Point** controls are used in both one-point and two-point mode. The **Lat** and **Lon** edit boxes are used to enter the latitude and longitude of the center point of the small circle to be displayed. These values must be in degrees. To display more than one small circle, a vector of values can be entered, enclosed in brackets in each edit box. Pushing the **Lat** or **Lon** button brings up an expanded edit box for easier entry of long vectors. The **Mouse Select** button is used to select a center point by clicking on the displayed map. The coordinates of the selected point then appear in the **Lat** and **Lon** edit boxes and can be modified. The coordinates appear in degrees, regardless of the angle units defined for the current map projection.

The **Circle Point** controls are used only in two-point mode. The **Lat** and **Lon** edit boxes are used to enter the latitude and longitude of a point on the perimeter of the small circle to be displayed. These values must be in degrees. To display more than one small circle, a vector of values can be entered, enclosed in brackets in each edit box. Pushing the **Lat** or **Lon** button brings up an expanded edit box for easier entry of long vectors. The **Mouse Select** button is used to select a perimeter point by clicking on the displayed map. The coordinates of the selected point then appear in the **Lat** and **Lon** edit boxes and can be modified. The coordinates appear in degrees, regardless of the angle units defined for the current map projection.

The **Size and Sector** controls are used only in one-point mode. The **Radius Units** button brings up a **Define Radius Units** dialog box, which allows for modification of the small circle radius units and the normalizing geoid. The **Rad** edit box is used to enter the radius of the small circle in the proper units. The **Arc** edit box is used to specify the sector azimuth, measured in degrees, clockwise from due north. If the entry is omitted, a complete small circle is drawn. When entering radius and arc data for more than one small circle, vectors of values, enclosed in brackets, are entered in each edit box. Pushing the **Rad** or **Arc** button brings up an expanded edit box for that entry, which is useful for entering long vectors.

The **Z Plane** edit box is used to enter a scalar value that specifies the plane in which to display the small circles.

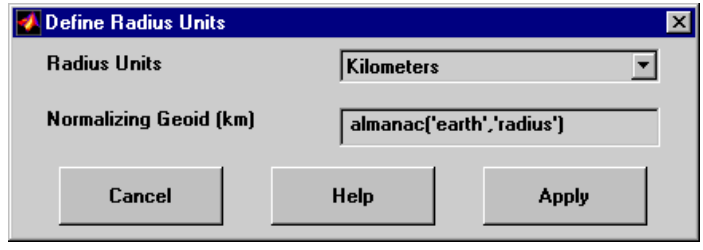
The **Other Properties** edit box is used to specify additional properties of the small circles to be projected, such as `'Color','b'`. String entries must be enclosed in quotes.

Pressing the **Apply** button accepts the input data and displays the small circles on the current map axes.

Pressing the **Cancel** button disregards any input data and closes the **Define Small Circles** dialog box.

Define Radius Units Dialog Box

This dialog box, available only in one-point mode, allows for modification of the small circle radius units and the normalizing geoid.



The **Radius Units** pull-down menu is used to select the units of the small circle radius. The unit selected is displayed near the top of the **Define Small Circles** dialog box, and all latitude and longitude entries must be entered in these

units. Users must also be sure to specify the normalizing geoid in the same units. If radians are selected, it is assumed the radius entry is a multiple of the radius used to display the current map, as defined by the map geoid property.

The **Normalizing Geoid** edit box is used modify the radius used to normalize the small circle radius to a radian value, which is necessary for proper calculations and map display. This entry must be in the same units as the small circle radius. If the small circle radius units are in radians, then the normalizing geoid must be the same as the geoid used for the current map axes.

Pressing the **Cancel** button disregards any modifications and closes the **Define Radius Units** dialog box.

Pressing the **Apply** button accepts any modifications and returns to the **Define Small Circles** dialog box.

See Also

`scircle1  scircle2  point`

seedm

Purpose

Encode a regular surface map

Activation

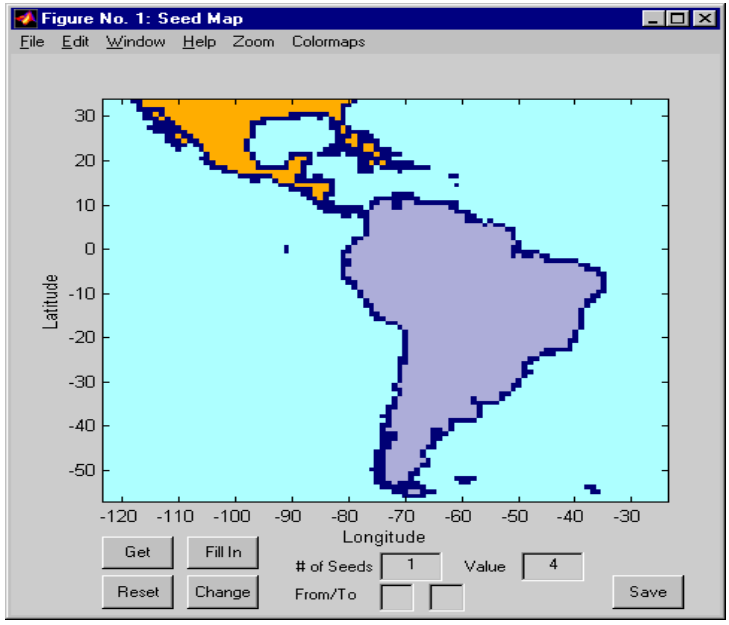
Command Line
`seedm(map,maplegend)`

Description

Encoding is the process of filling in specific values in regions of a matrix map up to specified boundaries, which are indicated by entries of 1 in the variable `map`. Encoding entire regions at one time allows indexed maps to be created quickly.

`seedm(map,maplegend)` displays the surface map in a new figure window and allows for seeds to be specified and the encoded map generated. The encoded map can then be saved to the workspace. `map` is the matrix map and must consist of positive integer index values. `maplegend` is the matrix map legend vector of the surface.

Controls



The **Zoom On/Off** menu toggles the zoom box on and off. The box can be moved by clicking on the new location or by dragging the box to the new location. The box size can be increased or decreased by dragging a corner of the box. Pressing the `Return` key or double-clicking in the center of the box zooms in to the box limits.

The **Colormaps** menu provides a variety of colormap options that can be applied to the map. See `clrmenu` in this guide for a description of the **Colormaps** menu options.

The **Get** button allows mouse selection of points on the map to which seeds are assigned. The number of points to be selected is entered in the **# of Seeds** edit box. The value of the seed is entered in the **Value** edit box. This seed value is assigned to each point selected with the mouse. The **Get** button is pressed to begin mouse selection. After all the points have been selected, the **Fill In** button is pressed to perform the encoding operation. The region containing the seed point is filled in with the seed value. The **Reset** button is used to disregard all points selected with the mouse before the **Fill In** button is pressed.

Alternatively, specific map values can be globally replaced by using the **From/To** edit boxes. The value to be replaced is entered in the first edit box, and the new value is entered in the second edit box. Pressing the **Change** button replaces all instances of the **From** value to the **To** value in the map.

Note: Values of 1 represent boundaries and should not be changed.

The **Save** button is used to save the encoded map to the workspace. A dialog box appears in which the map variable name is entered.

See Also

`colorm` `encodem` `getseeds` `maptrim`

showm

Purpose

Show mapped objects

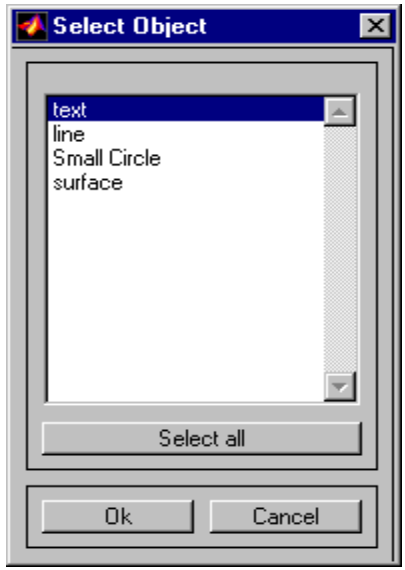
Activation

Command Line	Maptool
`showm`	**Tools⇒Show⇒Object**

Description

`showm` brings up a **Select Object** dialog box for selecting mapped objects to show (`Visible` property set to `'on'`).

Controls



The scroll box is used to select the desired objects from the list of mapped objects. Pushing the **Select all** button highlights all objects in the scroll box for selection. Pushing the **Ok** button changes the `Visible` property of the selected objects to `'on'`. Pushing the **Cancel** button aborts the operation without changing any properties of the selected objects.

See Also

`showm`

Purpose

Project a stem plot onto the current map axes

Activation

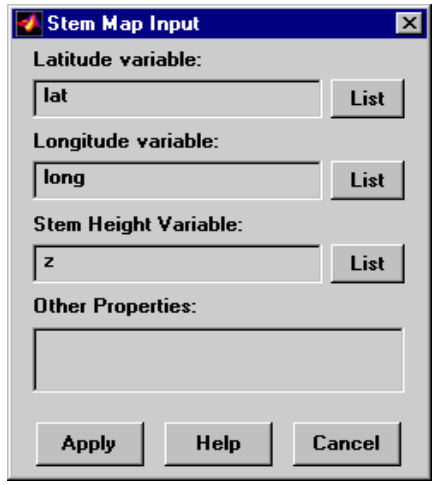
Command Line	Maptool
`stem3m`	**Map⇒Stem**

Description

`stem3m` activates a **Stem Map Input** dialog box for projecting a stem plot onto the current map axes. A stem plot displays data as lines extending perpendicular to the *xy*-plane on the map.

If no map axes are current, a **No Map Axes** dialog box appears. Choose **Yes** to activate the **Projection Control** dialog box for defining map axes properties. Upon creation of the map axes, the **Stem Map Input** dialog box appears.

Controls



he **Latitude variable** edit box is used to specify the workspace variable containing the latitude coordinates for the stem plot.

The **Longitude variable** edit box is used to specify the workspace variable containing the longitude coordinates for the stem plot.

The **Stem Height variable** edit box is used to specify the workspace variable containing the stem height data.

Pressing the **List** button produces a list of all current workspace variables, from which the latitude, longitude, and stem height variables can be selected.

The **Other Properties** edit box is used to specify additional properties of the stem lines to be projected, such as `'Color','r'`. String entries must be enclosed in quotes.

Pressing the **Apply** button accepts the input data and projects the stem plot onto the current map axes.

Pressing the **Cancel** button disregards any input data and closes the **Stem Map Input** dialog box.

See Also

`stem3m`

Purpose

Interactively calculate distance, azimuth, and reckoning

Activation

Command Line	Maptool

`surfdist`
`surfdist(h)`
`surfdist([])`
	Display⇒Surface⇒Distances

Description

`surfdist` activates the **Surface Distance** dialog box for the current axes only if the axes has a proper map definition. Otherwise, the **Surface Distance** dialog box is activated, but is not associated with any axes.

`surfdist(h)` activates the **Surface Distance** dialog box for the axes specified by the handle `h`. The axes must be a map axes.

`surfdist([])` activates the **Surface Distance** dialog box and does not associate it with any axes, regardless of whether the current axes has a valid map definition.

Controls

The **Style** selection buttons are used to specify whether a great circle or rhumb line is used to calculate the surface distance. When all other entries are provided, selecting a style updates the surface distance calculation.

The **Mode** selection buttons are used to specify whether one point or two points are to be used in defining the track distance. If one-point mode is selected, a starting point, azimuth, and range are the required inputs, and the ending point is computed. If two-point mode is selected, starting and ending points of the track are required, and the azimuth and distance along this track are then computed.

The **Show Track** check box is used to indicate whether the track is shown on the associated map display. The track is deleted when the **Surface Distance** dialog box is closed, or when the **Show Track** box is unchecked and the surface distance calculations are recomputed.

The **Starting Point** controls are used for both one-point and two-point mode. The **Lat** and **Lon** edit boxes are used to enter the latitude and longitude of the starting point of the track. These values must be in degrees. Only one starting point can be entered. The **Mouse Select** button is used to select a starting point by clicking on the displayed map. The coordinates of the selected point then appear in the **Lat** and **Lon** edit boxes and can be modified. The coordinates appear in degrees, regardless of the angle units defined for the current map projection.

The **Ending Point** controls are enabled only for two-point mode. The **Lat** and **Lon** edit boxes are used to enter the latitude and longitude of the ending point of the track. These values must be in degrees. Only one ending point can be entered. The **Mouse Select** button is used to select an ending point by clicking on the displayed map. The coordinates of the selected point then appear in the **Lat** and **Lon** edit boxes and can be modified. The coordinates appear in degrees, regardless of the angle units defined for the current map projection. During one-point mode, the **Ending Point** controls are disabled, but the ending point that results from the surface distance calculation is displayed.

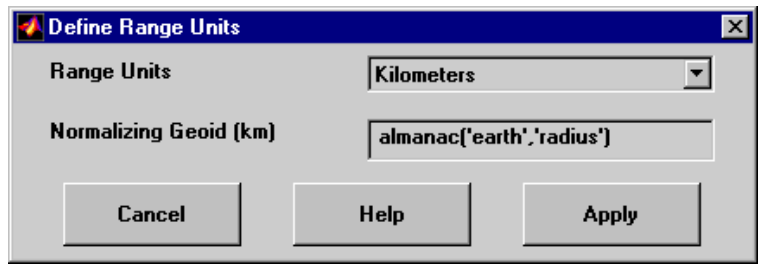
The **Direction** controls are enabled only for one-point mode. The **Range Units** button brings up a **Define Range Units** dialog box which allows for modification of the range units and the normalizing geoid. The **Az** edit box is used to enter the azimuth, which sets the initial direction of the track from the starting point. Azimuth is measured in degrees clockwise from due north. The **Rng** edit box is used to specify the reckoning range of the track, in the proper units. The azimuth and reckoning range, along with the starting point, are used to compute the ending point of the track in one-point mode. During two-point mode, the **Direction** controls are disabled, but the azimuth and range values resulting from the surface distance calculation are displayed.

Pressing the **Close** button disregards any input data, deletes any surface distance tracks that have been plotted, and closes the **Surface Distance** dialog box.

Pressing the **Compute** button accepts the input data and computes the specified distances.

Define Range Units Dialog Box

This dialog box, available only for one-point mode, allows for modification of the range units and the normalizing geoid.



The **Range Units** pull-down menu is used to select the units of the reckoning range. The unit selected is displayed near the top of the **Surface Distance** dialog box, and all latitude and longitude entries must be entered in these units. Users must also be sure to specify the normalizing geoid in the same units. If radians are selected, it is assumed the range entry is a multiple of the radius of the normalizing geoid. In this case, the normalizing geoid must be the same as the geoid used to display the current map.

The **Normalizing Geoid** edit box is used modify the radius used to normalize range entries to radian values, which is necessary for proper calculations and map display. This entry must be in the same units as the range units. If the range units are in radians, then the normalizing geoid must be the same as the geoid used for the current map axes.

Pressing the **Cancel** button disregards any modifications and closes the **Define Range Units** dialog box.

Pressing the **Apply** button accepts any modifications and returns to the **Surface Distance** dialog box.

Purpose

Display a lighted matrix map warped to a projected graticule

Activation

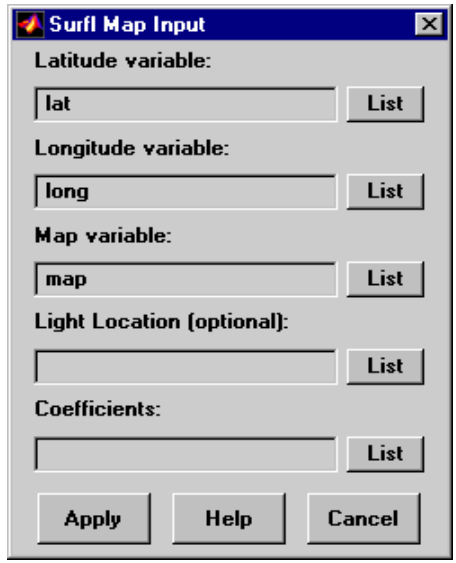
Command Line
`surflm`

Description

`surflm` activates a **Surflm Map Input** dialog box to project a lighted map surface onto the current map axes.

If no map axes are current, a **No Map Axes** dialog box appears. Choose **Yes** to activate the **Projection Control** dialog box for defining map axes properties. Upon creation of the map axes, the **Surflm Map Input** dialog box appears.

Controls



The **Latitude variable** edit box is used to specify the workspace variable containing the latitude data of the surface to be projected.

The **Longitude variable** edit box is used to specify the workspace variable containing the longitude data of the surface to be projected.

The **Map variable** edit box is used to specify the workspace variable containing the matrix map.

The **Light Location** edit box is used to specify the workspace variable containing the direction of the light source. This can be a three-element vector of the form `[x y z]` or a two-element vector of the form `[azimuth elevation]`. If omitted, the default is 45 degrees counterclockwise from the current view direction.

The **Coefficients** edit box is used to specify the workspace variable containing the relative contributions of ambient light, diffuse reflection, specular reflection, and the specular shine coefficient. This is a four-element vector of the form `[ka kd ks shine]`. If the entry is omitted, the default is `[.55 .6 .4 10]`.

Pressing the **List** button produces a list of all current workspace variables, from which the latitude, longitude, map, light location, and coefficient variables can be selected.

Pressing the **Apply** button accepts the input data and projects the lighted surface object onto the current map axes.

Pressing the **Cancel** button disregards any input data and closes the **Surflm Map Input** dialog box.

See Also

`surflm`

Purpose

Edit the tag of mapped objects

Activation

Command Line
`tagm` `tagm(h)`

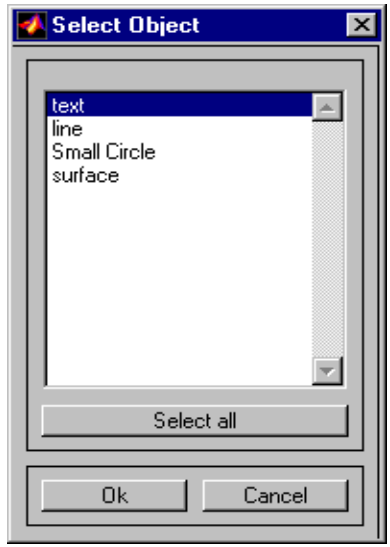
Description

`tagm` brings up a **Select Object** dialog box for selecting mapped objects and changing their `Tag` property. Upon selecting the objects, the **Edit Tag** dialog box is activated, in which the new tag is entered.

`tagm(h)` activates the **Edit Tag** dialog box for the objects specified by the handle `h`.

Controls

Select Object Dialog Box



The scroll box is used to select the desired objects from the list of mapped objects. Pushing the **Select all** button highlights all objects in the scroll box for selection. Pushing the **Ok** button activates the **Edit Tag** dialog box. Pushing the

Cancel button aborts the operation without changing any properties of the selected objects.

Edit Tag Dialog Box

The new tag string is entered in the edit box. Pressing the **Apply** button changes the `Tag` property of all selected objected to the new tag string. Pressing the **Cancel** button closes the **Edit Tag** dialog box without changing the `Tag` property of the selected objects.

See Also

`tagm`

Purpose

Project text on the current map axes

Activation

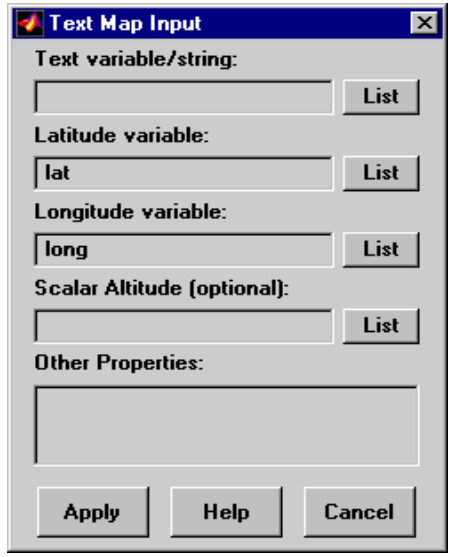
Command Line	Maptool
`textm`	**Map⇒Text**

Description

`textm` activates a **Text Map Input** dialog box, which accepts input data to project a text object onto the current map axes.

If no map axes are current, a **No Map Axes** dialog box appears. Choose **Yes** to activate the **Projection Control** dialog box for defining map axes properties. Upon creation of the map axes, the **Text Map Input** dialog box appears.

Control



The **Text variable/string** edit box is used to specify the workspace variable containing the text strings to be projected. A single text string can also be entered, provided it is enclosed in single quotes. Multiple lines of text can be entered using a cell array.

The **Latitude variable** edit box is used to specify the workspace variable containing the latitude data for the text string(s) to be projected. If a single text string is to be plotted, a scalar latitude value can be entered.

The **Longitude variable** edit box is used to specify the workspace variable containing the longitude data of the text object(s) to be projected. If a single text string is to be plotted, a scalar longitude value can be entered.

The **Scalar Altitude** edit box is used to specify the workspace variable containing the *z*-axis altitudes of the text object(s) to be projected. If a single text string is to be plotted, a scalar altitude value can be entered.

Pressing the **List** button produces a list of all current workspace variables, from which the latitude, longitude, and altitude variables can be selected.

The **Other Properties** edit box is used to specify additional properties of the text object(s) to be projected, such as `'FontSize',12`. String entries must be enclosed in quotes.

Pressing the **Apply** button accepts the input data and projects the text object(s) onto the current map axes.

Pressing the **Cancel** button disregards any input data and closes the **Text Map Input** dialog box.

See Also

`textm`

Purpose

Display great circles and rhumb lines on a map

Activation

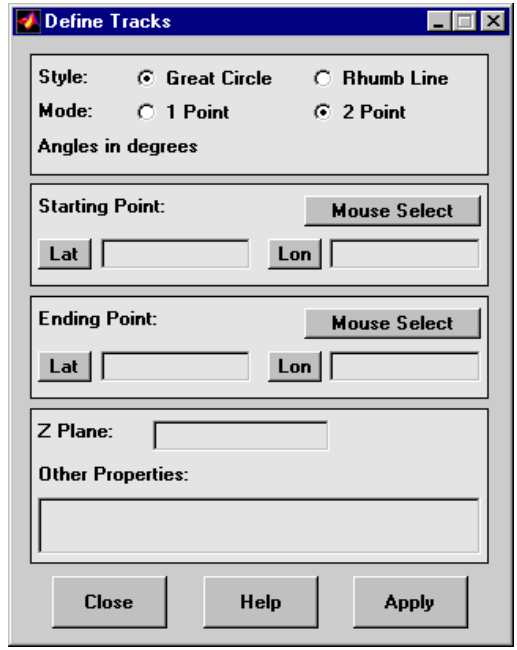
Command Line	Maptool
`trackui` `trackui(h)`	**Display⇒Tracks**

Description

`trackui` activates the **Define Tracks** dialog box for adding great circle or rhumb line tracks to the current map axes.

`trackui(h)` activates the **Define Tracks** dialog box for adding great circle or rhumb line tracks to the map axes specified by the axes handle `h`.

Controls



Define Tracks dialog box for two-point mode

The **Style** selection buttons are used to specify whether a great circle or rhumb line track is displayed.

The **Mode** selection buttons are used to specify whether one point or two points are to be used in defining the track. If one-point mode is selected, a starting point, azimuth, and range are the required inputs. If two-point mode is selected, starting and ending points are required.

The **Starting Point** controls are used for both one-point and two-point mode. The **Lat** and **Lon** edit boxes are used to enter the latitude and longitude of the starting point of the track to be displayed. These values must be in degrees. To display more than one track, a vector of values can be entered, enclosed in brackets in each edit box. Pushing the **Lat** or **Lon** button brings up an expanded edit box for easier entry of long vectors. The **Mouse Select** button is used to select a starting point by clicking on the displayed map. The coordinates of the selected point then appear in the **Lat** and **Lon** edit boxes and can be modified. The coordinates appear in degrees, regardless of the angle units defined for the current map projection.

The **Ending Point** controls are used only for two-point mode. The **Lat** and **Lon** edit boxes are used to enter the latitude and longitude of the ending point of the track to be displayed. These values must be in degrees. To display more than one track, a vector of values can be entered, enclosed in brackets, in each edit box. Pushing the **Lat** or **Lon** button brings up an expanded edit box for easier entry of long vectors. The **Mouse Select** button is used to select an ending point by clicking on the displayed map. The coordinates of the selected point then appear in the **Lat** and **Lon** edit boxes and can be modified. The coordinates appear in degrees, regardless of the angle units defined for the current map projection.

The **Direction** controls are used only for one-point mode. The **Range Units** button brings up a **Define Range Units** dialog box, which allows for modification of the range units and the normalizing geoid. The **Az** edit box is used to enter the azimuth, which sets the initial direction of the track from the starting point. Azimuth is measured in degrees clockwise from due north. The **Rng** edit box is used to specify the range of the track, in the proper units. If the range entry is omitted, a complete track is drawn. When inputting azimuth and range data for more than one track, vectors of values, enclosed in brackets, are entered in each edit box. Pushing the **Az** or **Rng** button brings up an expanded edit box for that entry, which is useful for entering long vectors.

The **Z Plane** edit box is used to enter a scalar value that specifies the plane in which to display the tracks.

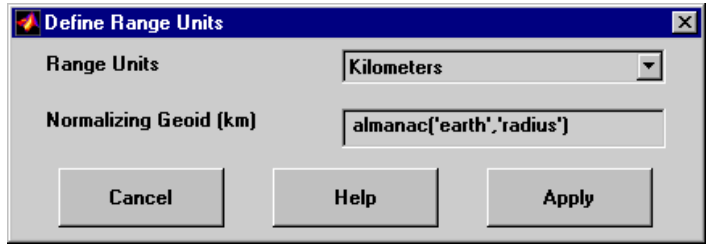
The **Other Properties** edit box is used to specify additional properties of the tracks to be projected, such as `'Color','b'`. String entries must be enclosed in quotes.

Pressing the **Apply** button accepts the input data and displays the tracks on the current map axes.

Pressing the **Cancel** button disregards any input data and closes the **Define Tracks** dialog box.

Define Range Units Dialog Box

This dialog box, available only for one-point mode, allows for modification of the range units and the normalizing geoid.



The **Range Units** pull-down menu is used to select the units of the track range. The unit selected is displayed near the top of the **Define Tracks** dialog box, and all latitude and longitude entries must be entered in these units. Users must also be sure to specify the normalizing geoid in the same units. If radians are selected, it is assumed the range entry is a multiple of the radius used to display the current map.

The **Normalizing Geoid** edit box is used to modify the radius used to normalize range entries to radian values, which is necessary for proper calculations and map display. This entry must be in the same units as the range units. If the range units are in radians, then the normalizing geoid must be the same as the geoid used for the current map axes.

Pressing the **Cancel** button disregards any modifications and closes the **Define Range Units** dialog box.

Pressing the **Apply** button accepts any modifications and returns to the **Define Tracks** dialog box.

See Also `track1  track2`

Purpose

Process mouse button down callbacks for mapped objects

Activation

set the `ButtonDownFcn` property to `'uimaptbx'`

Description

`uimaptbx` processes mouse events for mapped objects. `uimaptbx` can be assigned to an object by setting the `ButtonDownFcn` to `'uimaptbx'`. This is the default setting for all objects created with the Mapping Toolbox.

If `uimaptbx` is assigned to an object, the following mouse events are recognized: A single-click and hold on an object displays the object tag. If no tag is assigned, the object type is displayed. A double-click on an object activates the MATLAB Guide Property Editor. An extend-click on an object activates the **Projection Control** dialog box, which allows the map projection and display properties to be edited. An alternate-click on an object allows basic properties to be edited using simple mouse clicks and drags.

Definitions of extend-click and alternate-click on various platforms are as follows:

`For MS-Windows:`	Extend-click – Shift click left button or both buttons Alternate-click – Control click left button or right button
`For X-Windows:`	Extend-click – Shift click left button or middle button Alternate-click – Control click left button or right button

See Also

`axesm  axesmui  property editors`

Purpose

Adjust the z-plane of mapped objects

Activation

Command Line
`zdatam`
`zdatam(h)`
`zdatam(`*`str`*`)`

Description

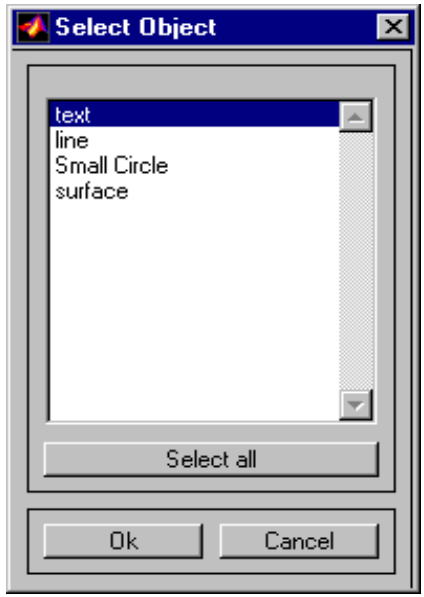
`zdatam` brings up a **Select Object** dialog box for selecting mapped objects and adjusting their `ZData` property. Upon selecting the objects, the **Specify Zdata** dialog box is activated, in which the new `ZData` variable is entered. Note that not all mapped objects have the `ZData` property (for example text objects).

`zdatam(h)` activates the **Specify Zdata** dialog box for the objects specified by the handle `h`.

`zdatam(`*`str`*`)` activates the **Specify Zdata** dialog box for the objects identified by *`str`*, where *`str`* is any string recognized by `handlem`.

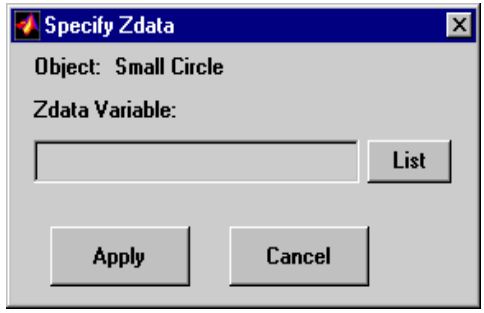
Controls

Select Object Dialog Box



The scroll box is used to select the desired objects from the list of mapped objects. Pushing the **Select all** button highlights all objects in the scroll box for selection. Pushing the **Ok** button activates another **Specify Zdata** dialog box. Pushing the **Cancel** button aborts the operation without changing any properties of the selected objects.

Specify ZData Dialog Box



The **Zdata Variable** edit box is used to specify the name of the `ZData` variable. Pressing the **List** button produces a list of all current workspace variables, from which the `ZData` variable can be selected. A scalar value or a valid MATLAB expression can also be entered. Pressing the **Apply** button changes the `ZData` property of all selected objected to the new values. Pressing the **Cancel** button closes the **Specify ZData** dialog box without changing the `ZData` property of the selected objects.

See Also

`zdatam`

4

External Data Reference

How to Use the External Data Reference Pages

The External Data Reference pages are organized alphabetically by the name of the interface function. The entries in this chapter contain the following:

Purpose	Provides concise descriptions.
Syntax	Summarizes the formats of the data interface function.
Background	Provides useful background information on the data format.
Description	Gives overall information about the function and describes how each syntax behaves.
Remarks	Provides tangential information about the function.
Examples	Shows concrete illustrations of how the function can be used.
Sources	Lists the various sites where the data can be obtained.
See Also	Refers you to the reference entries of related commands.
References	Provides pointers to additional resources.

External Data Function Tables

External Data Interface

Function	Description
avhrrgoode	Read AVHRR data stored in the Goode projection.
avhrrlambert	Read AVHRR data stored in the Lambert projection.
dcwdata	Read selected data from the Digital Chart of the World.
dcwdem	Read Digital Chart of the World elevation data.
dcwgaz	Search for entries in the Digital Chart of the World gazette.
dcwread	Read a Digital Chart of the World file.
dcwrhead	Read a Digital Chart of the World file header.
dted	Read U. S. Department of Defense Digital Terrain Elevation Data (DTED) data
egm96geoid	Read 15-minute gridded geoid heights from the EGM96 geoid model.
etopo5	Read data from the ETOPO5 dataset.
fipsname	Read TIGER thinned boundary file FIPS names.
gshhs	Read Global Self-consistent Hierarchical High-resolution Shoreline data.
gtopo30	Read GTOPO30 30-arc-second (1 km) global elevation data.
readfk5	Read data from the Fifth Fundamental Catalog of Stars.
satbath	Read Global 2-minute (4 km) topography from satellite bathymetry.
tbase	Read data from the TerrainBase dataset.
tgrline	Read data from TIGER/Line files.

Function	Description
tigermif	Read TIGER MapInfo Interchange Format thinned boundary files.
tigerp	Read TIGER ArcInfo Format thinned boundary files.
usgs24kdem	Read USGS 1:24,000 (30 m) digital elevation maps.
usgsdem	Read USGS 1:250,000 (100 m) digital elevation maps.
usgsdems	Read USGS digital elevation map filenames.
vmap0data	Extract selected data from the Vector Map Level 0 CD-ROMs.
vmapphead	Parse the Vector Map Level 0 header string.
vmap0read	Read a Vector Map Level 0 file.
vmap0rhead	Read the Vector Map Level 0 file headers.

Purpose

Read AVHRR data stored in the Goode Projection

Syntax

```
[latgrat,longrat,map] = avhrrgoode
[...] = avhrrgoode(region)
avhrrgoode(region,filename)
avhrrgoode(region,filename,scalefactor)
avhrrgoode(region,filename,scalefactor,latlim,lonlim)
avhrrgoode(region,filename,scalefactor,latlim,lonlim,gsize)
avhrrgoode(region,filename,scalefactor,latlim,lonlim,gsize,
   fnrows,fncols)
avhrrgoode(region,filename,scalefactor,latlim,lonlim,gsize,
   fnrows,fncols,resolution)
avhrrgoode(region,filename,scalefactor,latlim,lonlim,gsize,
   fnrows,fncols,resolution,precision)
```

Background

The United States plans to build a family of satellite-based sensors to measure climate change under the Earth Observing System (EOS) program. Early precursors to the EOS data are the datasets produced by NOAA and NASA under the Pathfinder program. These are data derived from the Advanced High Resolution Radiometer sensor flown on the NOAA Polar Orbiter satellites, NOAA-7, -9 and -11, and have spatial resolutions of about 1 km. The data from the AVHRR sensor is processed into separate land, sea and atmospheric indices. Land area data is processed to a non-dimensional vegetation index or land cover classification and stored in binary files in the Plate Carree, Goode and Lambert projections. Sea data is processed to surface temperatures, and stored in HDF formats. This function reads land data saved in the Goode projection with global and continental coverage at 1 km. It can also read 8 km data with global coverage.

Description

`[latgrat,longrat,map] = avhrrgoode` reads data from an AVHRR dataset with a nominal resolution of 1 km. These files have 17347 rows and 40031 columns of data, or somewhat more than the capacity of one CD-ROM. The file is selected interactively. Data is returned as a general matrix map with the graticule matrices in units of degrees.

`avhrrgoode(region)` Reads data from a file with data covering the specified *region*. Valid regions are 'g' or 'global', 'af' or 'africa', 'ap' or 'australia/

pacific', 'ea' or 'eurasia', 'na' or 'north america', and 'sa' or 'south america'. The file is selected interactively. If omitted, 'global' is assumed.

avhrrgoode(*region*,*filename*) uses the provided *filename*.

avhrrgoode(*region*,*filename*,scalefactor) uses the integer scalefactor to down-sample the data. A scale factor of 1 returns every point. A scale factor of 10 returns every 10th point. If omitted, 100 is assumed.

avhrrgoode(*region*,*filename*,scalefactor,latlim,lonlim) returns data for the specified region. The returned data will extend somewhat beyond the requested area. If omitted, the entire area covered by the data file is returned. The limits are two-element vectors in units of degrees, with latlim in the range [-90 90] and lonlim in the range [-180 180].

avhrrgoode(*region*,*filename*,scalefactor,latlim,lonlim,gsize) controls the size of the graticule matrices. gsize is a two-element vector containing the number of rows and columns desired. If omitted or empty, a graticule the size of the map is returned.

avhrrgoode(*region*,*filename*,scalefactor,latlim,lonlim,gsize,fnrows, fncols) overrides the standard file format for the selected region. This is useful for data stored on CD-ROM, which may have been truncated to fit. Some data was distributed with 16347 rows and 40031 columns of data on CD-ROMs. Non-dimensional vegetation index data at 8 km spatial resolution has 2168 rows and 5004 columns.

avhrrgoode(*region*,*filename*,scalefactor,latlim,lonlim,gsize,fnrows, fncols,resolution) reads a dataset with the spatial resolution specified in meters. If omitted, the full resolution of 1000 meters is assumed. Data is also available at 8000 meter resolution.

avhrrgoode(*region*,*filename*,scalefactor,latlim,lonlim,gsize,fnrows, fncols,resolution,*precision*) reads a dataset with the integer *precision* specified. If omitted, 'uint8' is assumed. 'uint16' is appropriate for some files. Check the data's README file for specification of the file format and contents.

Remarks

The AVHRR project and datasets are described in:

http://daac.gsfc.nasa.gov/DATASET_DOCS/avhrr_dataset.html

http://daac.gsfc.nasa.gov/CAMPAIGN_DOCS/FTP_SITE/readmes/pal.html

http://edcwww.cr.usgs.gov/landdaac/1KM/1kmhomepage.html

http://edcwww.cr.usgs.gov/landdaac/glcc/glcc_na.html

Some sources for the data include:

ftp://daac.gsfc.nasa.gov/data/avhrr/

ftp://edcftp.cr.usgs.gov/pub/data/glcc/

This function reads the binary files as-is. You should not use byte-swapping software on these files.

Examples

Read a 1 km Global Land Cover Classification (GLCC) file using the default parameters. Select the file 'gusgs1_2.img' interactively. This file is available from <ftp://edcftp.cr.usgs.gov/pub/data/glcc/globe/gusgs1_2.img.gz>.

```
[latgrat,longrat,map] = avhrrgoode;
```

Read the same file at full resolution for just the island of Cyprus.

```
[latgrat,longrat,map] = avhrrgoode('g','gusgs1_2.img',1,...
       [34.2 35.9],[32 35]);
```

Read the GLCC urban areas file covering North America in the Goode projection for just the area of eastern Massachusetts. This file is available from <ftp://edcftp.cr.usgs.gov/pub/data/glcc/na/goode/naurbang.img.gz>.

```
[latgrat,longrat,map] = avhrrgoode('north america',...
       'naurban.img',1,[41.13 42.75],[-71.7 -69.8]);
```

Read the global data on the “Global Land 1-km AVHRR Data Set – Vegetation Index 6/21-30, 1992” CD-ROM (distributed by the Land Processes Distributed Active Archive Center, EROS Data Center, Sioux Falls, South Dakota, 57198, USA). Sample every 100th point for the entire globe, returning one lat and long for value. Provide the non-standard number of rows and columns in the file.

```
[latgrat,longrat,map] = avhrrgoode('global','NDVI.IMG',...
       100,[-90 90],[-180 180],[],16347,40031);
```

Read the global 8 km resolution non-dimensional vegetation index available from <ftp://daac.gsfc.nasa.gov/data/avhrr/global_8km/.1994_1997/1994/jun/avhrrpf.ndvi.1ntfgl.940621.gz>. Sample every 10th point for the entire globe,

returning one lat and long for value. Provide the non-standard number of rows, columns and resolution in the file.

```
[latgrat,longrat,map] = avhrrgoode('global',...
       'avhrrpf.ndvi.1ntfgl.940621',10,[-90 90],...
       [-180 180],[],2168,5004,8000);
```

Read the global 8 km resolution data for AVHRR sensor channel 4 available from <ftp://daac.gsfc.nasa.gov/data/avhrr/global_8km/.1981_1985/1984/feb/avhrrpf.ch4.1ntfgl.840201.gz>. Read at the full 8 km resolution for the island of Cyprus, returning one lat and long for value. Provide the non-standard number of rows, columns, resolution and integer precision in the file.

```
[latgrat,longrat,map] = avhrrgoode('global',...
       'avhrrpf.ch4.1ntfgl.840201',10,[34.2 35.9],...
       [32 35],[],2168,5004,8000,'uint16');
```

Limitations

Most files store the data in scaled integers. Though this function returns the data as double, the scaling from integer to float is not performed. Check the data's README file for the appropriate scaling parameters.

Subsets of the land cover data are available in both the Goode and the uninterrupted Lambert azimuthal projections. Data can be read more quickly from the Lambert projection using `avhrrlambert`.

This function does not have the proper projection parameters to read the regional 8 km resolution datasets.

See Also

`avhrrlambert`	Read AVHRR data in the Lambert Azimuthal Equal-Area projection

Purpose

Read AVHRR data stored in the Lambert Azimuthal Projection

Syntax

```
[latgrat,longrat,map] = avhrrlambert(region)
avhrrlambert(region,filename)
avhrrlambert(region,filename,scalefactor)
avhrrlambert(region,filename,scalefactor,latlim,lonlim)
avhrrlambert(region,filename,scalefactor,latlim,lonlim,gsize)
avhrrlambert(region,filename,scalefactor,latlim,lonlim,gsize,
   precision)
```

Background

The United States plans to build a family of satellite-based sensors to measure climate change under the Earth Observing System (EOS) program. Early precursors to the EOS data are the datasets produced by NOAA and NASA under the Pathfinder program. These are data derived from the Advanced High Resolution Radiometer sensor flown on the NOAA Polar Orbiter satellites, NOAA-7, -9 and -11 with a spatial resolution of about 1 km. The data from the AVHRR sensor is processed into separate land, sea and atmospheric indices. Land area data is processed to a non-dimensional vegetation index or land cover classification and stored in binary files in the Plate Carree, Goode and Lambert projections. Sea data is processed to surface temperatures, and stored in HDF formats. This function reads land cover data for the continents saved in the Lambert azimuthal projection at 1 km.

Description

`[latgrat,longrat,map] = avhrrlambert(region)` reads data from an Advanced Very High Resolution Radiometer (AVHRR) dataset with a nominal resolution of 1 km that is stored in the Lambert projection. Data of this type includes the Global Land Cover Characteristics (GLCC). The region specifies the coverage of the file. Valid regions are 'g' or 'global', 'af' or 'africa', 'ap' or 'australia/pacific', 'e' or 'europe', 'a' or 'asia', 'na' or 'north america', 'sa' or 'south america'. Data is returned as a general matrix map with the graticule matrices in units of degrees.

`avhrrlambert(region,filename)` uses the provided *filename*.

`avhrrlambert(region,filename,scalefactor)` uses the integer `scalefactor` to downsample the data. A scale factor of 1 returns every point. A scale factor of 10 returns every 10th point. If omitted, 100 is assumed.

`avhrrlambert(`*`region`*`,`*`filename`*`,scalefactor,latlim,lonlim)` returns data for the specified region. The returned data will extend somewhat beyond the requested area. If omitted, the entire area covered by the data file is returned. The limits are two-element vectors in units of degrees, with `latlim` in the range `[-90 90]` and `lonlim` in the range `[-180 180]`.

`avhrrlambert(`*`region`*`,`*`filename`*`,scalefactor,latlim,lonlim,gsize)` controls the size of the graticule matrices. `gsize` is a two-element vector containing the number of rows and columns desired. If omitted or empty, a graticule the size of the map is returned.

`avhrrlambert(`*`region`*`,`*`filename`*`,scalefactor,latlim,lonlim,gsize,`*`precision`*`)` reads a dataset with the integer *`precision`* specified. If omitted, '`uint8`' is assumed. '`uint16`' is appropriate for some files. Check the data's README file for specification of the file format and contents.

Remarks

The AVHRR project and datasets are described in:

http://daac.gsfc.nasa.gov/DATASET_DOCS/avhrr_dataset.html

http://edcwww.cr.usgs.gov/landdaac/1KM/1kmhomepage.html

The Global Land Cover Characteristics dataset of described in:

http://edcwww.cr.usgs.gov/landdaac/glcc/glcc_na.html

The data can be obtained from:

ftp://edcftp.cr.usgs.gov/pub/data/glcc/

This function reads the binary files as-is. You should not use byte-swapping software on these files.

Example

Read the Global Land Cover Characteristics (GLCC) file covering North America in the Lambert projection with the USGS classification scheme. Use the default parameters to read the entire file, taking just every 100th point. This file is available from <ftp://edcftp.cr.usgs.gov/pub/data/glcc/na/lambert/nausgs1_2l.img.gz>.

```
[latgrat,longrat,map] = avhrrlambert('north america',...
       'nausgs1_2l.img');
```

Read the same file at full resolution for just the area of eastern Massachusetts.

```
[latgrat,longrat,map] = avhrrlambert('north america',...
       'nausgs1_2l.img',1,[41.2 41.5],[-70.9 -70.4]);
```

See Also

`avhrrgoode` Read AVHRR data in the Goode projection

dcwdata

Purpose

Read selected data from the Digital Chart of the World

Syntax

```
struct1 = dcwdata(library,latlim,lonlim,theme,topolevel)
struct1 = dcwdata(devicename,library,...)
[struct1,struct2,...] = dcwdata(...,{topolevel1,topolevel2,...})
```

Background

The Digital Chart of the World (DCW) is a detailed and comprehensive source of publicly available global vector data. It was digitized from the Operational Navigation Charts (scale 1:1,000,000) and Jet Navigation Charts (1:2,000,000), compiled by the U.S. Defense Mapping Agency (DMA) along with mapping agencies in Australia, Canada, and the United Kingdom. The digitized data was published on four CD-ROMS by the DMA and is distributed by the U.S. Geological Survey (USGS).

The DCW is out of print and has been succeeded by the Vector Map Level 0 (VMAP0).

The DCW organizes data into 17 different themes, such as political/oceans (PO), drainage (DN), roads (RD), or populated places (PP). The data is further tiled into 5 by 5 degree tiles and separated by topology level (patches, lines, points and text).

Description

`struct = dcwdata(library,latlim,lonlim,theme,topolevel)` reads data for the specified theme and topology level directly from the DCW CD-ROM. There are four CDs, one for each of the libraries: `'NOAMER'` (North America), `'SASAUS'` (Southern Asia and Australia), `'EURNASIA'` (Europe and Northern Asia), and `'SOAMAFR'` (South America and Africa). The desired theme is specified by a two-letter code string. A list of valid codes is displayed when an invalid code, such as `'?'`, is entered. The region of interest can be given as a point latitude and longitude or as a region with two-element vectors of latitude and longitude limits. The units of latitude and longitude are degrees. The data covering the requested region is returned, but will include data extending to the edges of the 5-by-5 degree tiles. The result is returned as a Mapping Toolbox geographic data structure.

`struct = dcwdata(devicename,library,...)` specifies the logical device name of the CD-ROM for computers that do not automatically name the mounted disk.

`[struct1, struct2,...] = dcwdata(...,{`*topolevel1*`,`*topolevel2*`,...})` reads several topology levels. The levels must be specified as a cell array with the entries `'patch'`, `'line'`, `'point'` or `'text'`. Entering {`'all'`} for the topology level argument is equivalent to {`'patch'`, `'line'`, `'point'`, `'text'`}. Upon output, the data structures are returned in the output arguments by topology level in the same order as they were requested.

Remarks

Latitudes and longitudes use WGS84 as a horizontal datum. Elevations are in feet above mean sea level. The dataset does not contain bathymetric data.

Some DCW themes do not contain all topology levels. In those cases, empty matrices are returned.

The data is tagged with strings describing the objects. Some data is provided with alternate tags in `tag2` and `tag3` fields. These alternate tags contain information that supplements the standard tag, such as the names of political entities or values of elevation. The `tag2` field generally has the actual values or codes associated with the data. If the information in the `tag2` field expands to more verbose descriptions, these are provided in the `tag3` field.

Point data for which there are descriptions of both the type and the individual names of objects is returned twice within the structure. The first set is a collection of points of the same type with appropriate tag. The second is as a set of individual points with the tag `'Individual Points'` and the name of the object in the `tag2` field.

Patches are broken at the tile boundaries. Setting the `EdgeColor` to `'none'` and plotting the lines will give the map a normal appearance.

The DCW was published in 1992 based on data compiled some years earlier. The political boundaries do not reflect recent changes such as the dissolution of the Soviet Union, Czechoslovakia, and Yugoslavia. In some cases, the boundaries of the successor nations are present as lower level political units. A new version, called VMAP0.

Examples

On the Macintosh:

```
s = dcwdata('NOAMER',41,-69,'?','patch');
??? Error using ==> dcwdata
Theme not present in library NOAMER
```

```
Valid two-letter theme identifiers are:
PO: Political/Oceans
PP: Populated Places
LC: Land Cover
VG: Vegetation
RD: Roads
RR: Railroads
UT: Utilities
AE: Aeronautical
DQ: Data Quality
DN: Drainage
DS: Supplemental Drainage
HY: Hypsography
HS: Supplemental Hypsography
CL: Cultural Landmarks
OF: Ocean Features
PH: Physiography
TS: Transportation Structure

POpatch = dcwdata('NOAMER',[41 44],[-72 -69],'PO','patch')
POpatch =
1x234 struct array with fields:
    type
    otherproperty
    tag
    altitude
    lat
    long
    tag2
    tag3
```

On a MS-DOS based operating system with the CD-ROM as the 'd:' drive:

```
[RDtext,RDline] = dcwdata('d:','SASAUS',[-48 -34],[164 180],...
     'RD',{'text','line'});
```

On a UNIX operating system with the CD-ROM mounted as '\cdrom':

```
[POpatch,POline,POpoint,POtext] = dcwdata('\cdrom',...
     'EURNASIA',-48 ,164,'PO',{'all'});
```

See Also

`vmap0data`	Read selected data from the Vector Map Level 0
`dcwgaz`	Search for entries in the Digital Chart of the World gazette
`dcwread`	Read a Digital Chart of the World file
`dcwrhead`	Read a Digital Chart of the World file header
`displaym`	Project data contained in a map structure
`extractm`	Extract vector data from a map structure
`mlayers`	GUI for manipulating map layers

References

The format and the history of the DCW are described in references [1], [2], and [3] located in the Bibliography at the end of this chapter.

dcwgaz

Purpose

Search for entries in the Digital Chart of the World gazette

Syntax

```
dcwgaz(library,object)
dcwgaz(devicename,library,object)
mtextstruc = dcwgaz(...)
[mtextstruc,mpointstruc] = dcwgaz(...)
```

Background

In addition to the geographic data, the Digital Chart of the World (DCW) also includes an extensive gazette feature. The gazette is a collection of the names of geographic items mentioned in the various themes of a DCW disk. One DCW disk may contain about 10,000 to 15,000 names. This function allows the gazette to be searched for names beginning with a particular string.

Description

`dcwgaz(library,object)` searches the DCW library for items beginning with the *object* string. There are four CDs, one for each of the libraries: `'NOAMER'` (North America), `'SASAUS'` (Southern Asia and Australia), `'EURNASIA'` (Europe and Northern Asia), and `'SOAMAFR'` (South America and Africa). Items which exactly match or begin with the *object* string are displayed on screen.

`dcwgaz(devicename,library,object)` specifies the logical device name of the CD-ROM for computers which do not automatically name the mounted disk.

`mtextstruc = dcwgaz(...)` displays the matched items on screen and returns a Mapping Toolbox geographic data structure with the matches as text entries.

`[mtextstruc,mpointstruc] = dcwgaz(...)` returns the matches in structures formatted both as text and as points.

Remarks

The search is not case sensitive. Items that match are those that begin with the *object* string. Spaces are significant.

Examples

On the Macintosh:

```
s = dcwgaz('EURNASIA','apatin')
APATIN
s =
             type: 'text'
    otherproperty: {1x2 cell}
              tag: 'Built up area'
           string: 'APATIN'
         altitude: []
```

```
                lat: 45.6660
               long: 18.9830
```

On a UNIX operating system with the CD-ROM mounted as '\cdrom':

```
[mtextstruc,mpointstruc] = dcwgaz('\cdrom','SOAMAFR',...
   'cape good')
Cape Goodenough
Cape Goodenough
Cape Goodenough

mtextstruc =
1x3 struct array with fields:
    type
    otherproperty
    tag
    string
    altitude
    lat
    long

mpointstruc =
1x3 struct array with fields:
    type
    otherproperty
    tag
    string
    altitude
    lat
    long
```

See Also

dcwdata	Read selected data from the Digital Chart of the World
dcwread	Read a Digital Chart of the World file
dcwrhead	Read a Digital Chart of the World file header
displaym	Project data contained in a map structure
mlayers	GUI for manipulating map layers

dcwread

Purpose

Read a Digital Chart of the World file

Syntax

```
dcwread
dcwread(filepath,filename)
dcwread(filepath,filename,recordIDs)
dcwread(filepath,filename,recordIDs,field,varlen)

struc = dcwread(...)
[struc,field] = dcwread(...)
[struc,field,varlen] = dcwread(...)
[struc,field,varlen,desciption] = dcwread(...)
[struc,field,varlen,desciption,narrativefield] = dcwread(...)
```

Background

The Digital Chart of the World (DCW) uses binary files in a variety of formats. This function determines the format of the file and returns the contents in a structure. The field names of this structure are the same as the field names in the DCW file.

Description

`dcwread` reads a DCW file. The user selects the file interactively.

`dcwread(filepath,filename)` reads the specified file. The combination [*filepath filename*] must form a valid complete filename.

`dcwread(filepath,filename,recordIDs)` reads selected records or fields from the file. If `recordIDs` is a scalar or a vector of integers, the function returns the selected records. If `recordIDs` is a cellarray of integers, all records of the associated fields are returned.

`dcwread(filepath,filename,recordIDs,field,varlen)` uses previously read field and variable length record information to skip parsing the file header (see below).

`struc = dcwread(...)` returns the file contents in a structure.

`[struc,field] = dcwread(...)` returns the file contents and a structure describing the format of the file.

`[struc,field,varlen] = dcwread(...)` also returns a vector describing which fields have variable length records.

`[struc,field,varlen,description] = dcwread(...)` also returns a string describing the contents of the file.

`[struc,field,varlen,description,narrativefield] = dcwread(...)` also returns the name of the narrative file for the current file.

Remarks

This function reads all DCW files except index files (files with names ending in `'X'`), thematic index files (files with names ending in `'TI'`), and spatial index files (files with names ending in `'SI'`).

File separators are platform dependent. The *filepath* input must use appropriate file separators, which can be determined using the MATLAB `filesep` function.

Examples

The following examples use the Macintosh directory system and file separators for the pathname:

```
s = dcwread('NOAMER:DCW:NOAMER:','GRT')
s =
                    ID: 1
             DATA_TYPE: 'GEO'
                 UNITS: '014'
             ELLIPSOID: 'WGS 84'
        ELLIPSOID_DETAIL: 'A=6378137,B=6356752 Meters'
          VERT_DATUM_REF: 'MEAN SEA LEVEL'
         VERT_DATUM_CODE: '015'
             SOUND_DATUM: 'MEAN SEA LEVEL'
        SOUND_DATUM_CODE: '015'
          GEO_DATUM_NAME: 'WGS 84'
          GEO_DATUM_CODE: 'WGE'
         PROJECTION_NAME: 'DECIMAL DEGREES'

s = dcwread('NOAMER:DCW:NOAMER:AE:','INT.VDT')
s =
5x1 struct array with fields:
    ID
    TABLE
    ATTRIBUTE
    VALUE
    DESCRIPTION

for i = 1:length(s); disp(s(i)); end
               ID: 1
            TABLE: 'AEPOINT.PFT'
        ATTRIBUTE: 'AEPTTYPE'
            VALUE: 1
        DESCRIPTION: 'Active civil'

               ID: 2
            TABLE: 'AEPOINT.PFT'
        ATTRIBUTE: 'AEPTTYPE'
            VALUE: 2
        DESCRIPTION: 'Active civil and military'
```

```
              ID: 3
           TABLE: 'AEPOINT.PFT'
       ATTRIBUTE: 'AEPTTYPE'
           VALUE: 3
     DESCRIPTION: 'Active military'

              ID: 4
           TABLE: 'AEPOINT.PFT'
       ATTRIBUTE: 'AEPTTYPE'
           VALUE: 4
     DESCRIPTION: 'Other'

              ID: 5
           TABLE: 'AEPOINT.PFT'
       ATTRIBUTE: 'AEPTTYPE'
           VALUE: 5
     DESCRIPTION: 'Added from ONC when not available from DAFIF'
```

```
s = dcwread('NOAMER:DCW:NOAMER:AE:','AEPOINT.PFT',1)
s =
           ID: 1
     AEPTTYPE: 4
     AEPTNAME: 'THULE AIR BASE'
      AEPTVAL: 251
     AEPTDATE: '19900502000000000000'
     AEPTICAO: '1261'
     AEPTDKEY: 'BR17652'
      TILE_ID: 94
       END_ID: 1

s = dcwread('NOAMER:DCW:NOAMER:AE:','AEPOINT.PFT',{1,2})
s =
4678x1 struct array with fields:
    ID
    AEPTTYPE
```

See Also

`dcwdata`	Read selected data from the Digital Chart of the World
`dcwgaz`	Search for entries in the Digital Chart of the World gazette
`dcwrhead`	Read a Digital Chart of the World file header

Purpose

Read the header of a Digital Chart of the World file

Syntax

```
dcwrhead
dcwrhead(filepath,filename)
dcwrhead(filepath,filename,fid)

dcwread(...)
str = dcwread(...)
```

Background

The Digital Chart of the World (DCW) uses header strings in most files to document the contents and format of that file. This function reads the header string, displays a formatted version in the command window, or returns it as a string.

Description

`dcwrhead` allows the user to select the header file interactively.

`dcwrhead(`*filepath*`,`*filename*`)` reads from the specified file. The combination [`filepath filename`] must form a valid complete filename.

`dcwrhead(`*filepath*`,`*filename*`,fid)` reads from the already open file associated with `fid`.

`dcwrhead(...)`, with no output arguments, displays the formatted header information on the screen.

`str = dcwrhead(...)` returns a string containing the DCW header.

Remarks

This function reads all DCW files except index files (files with names ending in `'X'`), thematic index files (files with names ending in `'TI'`) and spatial index files (files with names ending in `'SI'`).

File separators are platform dependent. The *filepath* input must use appropriate file separators, which may be determined using the MATLAB `filesep` function.

Examples

The following example uses the Macintosh file separators and path name:

```
dcwrhead('NOAMER:DCW:NOAMER:AE:','AEPOINT.PFT')
Aeronautical Points
AEPOINT.DOC
ID=I,    1,P,Row Identifier,-,-,
```

```
AEPTTYPE=I, 1,N,Airport Type,INT.VDT,-,
AEPTNAME=T,50,N,Airport Name,-,-,
AEPTVAL=I, 1,N,Airport Elevation Value,-,-,
AEPTDATE=D, 1,N,Aeronautical Information Date,-,-,
AEPTICAO=T, 4,N,International Civil Organization Number,-,-,
AEPTDKEY=T, 7,N,DAFIF Reference Number,-,-,
TILE_ID=S, 1,F,Tile Reference Identifier,-,AEPOINT.PTI,
END_ID=I 1,F,Entity Node Primitive Foreign Key,-,-,

s = dcwrhead('NOAMER:DCW:NOAMER:AE:','AEPOINT.PFT')
s =
;Aeronautical Points;AEPOINT.DOC;ID=I,1,F,Identifier,-,-,:AEPTTYPE=I,
1,N,Airport Type,INT.VDT,-,:AEPTNAME=T,50,N,Airport Name,-,-,:AEPTVAL=I,
1,N,Airport Elevation Value,-,-,:AEPTDATE=D, 1,N,Aeronautical
Information Date,-,-,:AEPTICAO=T, 4,N,International Civil
Organization Number,-,-,:AEPTDKEY=T,7,N,DAFIF Reference Number,-,-,:TILE_ID=S,
1,F,Tile Reference Identifier,-,AEPOINT.PTI,:END_ID=I 1,F,Entity
Node Primitive Foreign Key,-,-,:;
```

See Also

dcwdata	Read selected data from the Digital Chart of the World
dcwgaz	Search for entries in the Digital Chart of the World gazette
dcwread	Read a Digital Chart of the World file

Purpose

Read U. S. Department of Defense Digital Terrain Elevation Data (DTED) data

Syntax

```
[map,maplegend] = dted
[map,maplegend] = dted(filename)
[map,maplegend] = dted(filename,scalefactor)
[map,maplegend] = dted(filename,scalefactor,latlim,lonlim)
[map,maplegend,UHL,DSI,ACC] = dted(...)
```

Background

The U. S. Department of Defense, through the National Imagery and Mapping Agency, produces several kinds of digital cartographic data. One is digital elevation data, in a series called DTED, for Defense Digital Terrain Elevation Data. The data is available as 1-by-1 degree quadrangles at horizontal resolutions ranging from about 1 kilometer to 1 meter. The lowest resolution data is available to the public. Higher resolution data is restricted to the U.S. Department of Defense and its contractors.

Description

`[map,maplegend] = dted` returns all of the elevation data in a DTED file as a regular matrix map with elevations in meters. The file is selected interactively.

`[map,maplegend] = dted(`*`filename`*`)` returns all of the elevation data in the specified DTED file. The file must be found on the MATLAB path. If not found, the file may be selected interactively.

`[map,maplegend] = dted(`*`filename`*`,scalefactor)` returns data from the specified DTED file, downsampled by the scale factor. If omitted, a `scalefactor` of 1 is assumed.

`[map,maplegend] = dted(`*`filename`*`,scalefactor,latlim,lonlim)` reads the data for the part of the DTED file within the latitude and longitude limits. The limits must be two-element vectors in units of degrees.

`[map,maplegend,UHL,DSI,ACC] = dted(...)` also returns the DTED User Header Label (UHL), Data Set Identification (DSI) and Accuracy (ACC) metadata records as structures. Documentation describing the meaning of these records is available online at http://164.214.2.59/publications/specs/printed/DTED/dted1-2.doc.

Examples

```
[map,maplegend] = dted('n38.dt0');

[map,maplegend,UHL,DSI,ACC] = dted('n38.dt0',1,[38.5 38.8],...
       [-76.8 -76.6]);
```

Limitations

At higher latitudes the files have fewer longitude records. In those cases a warning is issued, and the coarser spacing is used in both directions.

Remarks

This function reads the DTED elevation files in the format used for files delivered on CD-ROM. The file names generally end in "`.dtN`", where N is 0,1,2,3,etc.

The 1 kilometer data is available online at http://164.214.2.59/geospatial/products/DTED/dted.html

The higher resolution data is available to the U.S. Department of Defense and its contractors from the National Imagery and Mapping Agency (NIMA).

DTED0 files have 120x120 points. DTED1 files have 1201x1201. The edges of adjacent tiles have redundant records. Maps will extend a half a cell outside the requested map limits.

A third-party description of the DTED data may be found at

http://www.fas.org/irp/program/core/dted.htm

Detailed official documentation is available at

http://164.214.2.59/publications/specs/printed/DTED/dted1-2.doc.

The DTED files are binary. No line ending conversion or byte-swapping is required.

See Also

`usgsdem`	USGS 1-Degree (3-arc-sec resolution) digital elevation data
`gtopo30`	30-Arc-Sec global digital elevation data
`tbase`	TerrainBase Global 5-Min digital terrain data
`etopo5`	ETOPO5 Global 5-Min digital terrain data

Purpose

Read 15-minute gridded geoid heights from the EGM96 geoid model of the earth

Syntax

```
[map,maplegend] = egm96geoid(scalefactor)
[map,maplegend] = egm96geoid(scalefactor,latlim,lonlim)
```

Background

Although the earth is round, it is not exactly a sphere. The shape of the Earth is usually defined by the geoid, which is defined as a gravitational equipotential surface, but may conceptualized as the shape the ocean surface would take in the absence of waves, weather and land. For cartographic purposes it is generally sufficient to treat the earth as a sphere or ellipsoid of revolution. For other applications, a more detailed model of the geoid such as EGM 96 may be required. EGM 96 is a spherical harmonic model of the geoid complete to degree and order 360. This function reads from a file of gridded geoid heights derived from the EGM 96 harmonic coefficients.

Description

`[map,maplegend] = egm96geoid(scalefactor)` reads the data for the entire world, downsampling the data by the scale factor. The result is returned as a regular matrix map and associated map legend. Heights are given in meters in the tide-free system.

`[map,maplegend] = egm96geoid(scalefactor,latlim,lonlim)` reads the data for the part of the world within the latitude and longitude limits. The limits must be two-element vectors in units of degrees. Longitude limits can be defined in the range `[–180 180]` or `[0 360]`. For example, `lonlim = [170 190]` returns data centered on the dateline, while `lonlim = [–10 10]` returns data centered on the prime meridian.

Examples

Read the EGM 96 geoid grid for the world, taking every 10th point.

```
[map,maplegend] = egm96geoid(10);
```

Read the a subset of the geoid grid at full resolution and interpolate to find the geoid height at a point between grid points.

```
[map,maplegend] = egm96geoid(1,[-10 -12],[129 132]);
z = ltln2val(map,maplegend,-11.1,130.22,'bicubic')
z =
       52.444
```

Remarks

This function reads the 15-minute EGM96 grid file "WW15MGH.GRD". PC users should download the DOS self-extracting executable in:

ftp://164.214.2.59/pub/gg/gravity3/ww15mgh.exe.

Other users should download and extract the unix compressed file at:

ftp://164.214.2.59/pub/gg/gravity3/WW15MGH.GRD.Z.

Do not modify the file once it has been extracted.

Information on the dataset may be found at:

http://cddisa.gsfc.nasa.gov/926/egm96/egm96.html.

Maps will extend a half a cell outside the requested map limits.

There are 721 rows and 1441 columns of values in the grid at full resolution. The lower resolution atlas data in GEOID.MAT is derived from the EGM 96 grid.

See Also

ltln2val	Returns map code value associated with positions

Purpose

Read data from the ETOPO5 global 5-minute Digital Terrain Model

Syntax

```
[map,maplegend] = etopo5(scalefactor)
[map,maplegend] = etopo5(scalefactor,latlim,lonlim)
```

Background

ETOPO5 is a global database of elevations and depths on a regular 5-minute grid. It is a compilation of data from a variety of different sources, including the U.S. Naval Oceanographic Office, U.S. Defense Mapping Agency, U.S. Navy Fleet Numerical Oceanographic Center, Bureau of Mineral Resources, Australia, and the Department of Industrial and Scientific Research, New Zealand. These databases were assembled by Margo Edwards at Washington University, St. Louis, Missouri.

Description

`[map,maplegend] = etopo5(scalefactor)` reads the data for the entire world, downsampling the data by the scale factor. The result is returned as a regular matrix map and an associated map legend.

`[map,maplegend] = etopo5(scalefactor,latlim,lonlim)` reads the data for the part of the world within the latitude and longitude limits. The limits must be two-element vectors in units of degrees.

Remarks

Data values are in whole meters, representing the elevation of the center of each cell. Some parts of the world are represented by data with a horizontal resolution as coarse as 1 degree by 1 degree. The vertical resolution varies from 1 meter for Australia and New Zealand to as much as 150 meters for parts of Africa, Asia, and South America. Oceanographic data in areas shallower than 200 meters contain little detail, because of how depth contours were converted to gridded depths.

ETOPO5 is being superseded by the development of a new digital terrain model called TerrainBase. See the `tbase` external interface function for more information.

The `etopo5` function reads the version of the database contained in two text files, `etopo5.southern.bat` and `etopo5.northern.bat`. These files are available over the Internet from The MathWorks:

```
ftp://ftp.mathworks.com
```

An overview of the data can be found at the ETOPO5 information page at the U.S. Geological Survey:

```
http://edcwww.cr.usgs.gov/glis/hyper/guide/etopo5
```

and from the U.S. National Geophysical Data Center Web page:

```
http://www.ngdc.noaa.gov/mgg/global/etopo5.HTML
```

Examples

Read every tenth point in the dataset:

```
scalefactor = 10;
[map,maplegend] = etopo5(scalefactor);
whos
  Name              Size         Bytes  Class
  map             216x432       746496  double array
  maplegend         1x3             24  double array
  scalefactor       1x1              8  double array
limitm(map,maplegend)
ans =
   -90    90     0   360
```

Read in data for Korea and Japan at the full resolution:

```
scalefactor = 1; latlim = [30 45]; lonlim = [115 145];
[map,maplegend] = etopo5(scalefactor,latlim,lonlim);
whos map
  Name      Size          Bytes  Class
  map     180x360        518400  double array
```

See Also

`gtopo30`	Read elevation data from GTOPO30
`tbase`	Read data from the TerrainBase model
`usgsdem`	Read USGS digital elevation maps

References

More information on ETOPO5 can be found in reference [4] located in the Bibliography at the end of this appendix.

Purpose

Read the FIPS (Federal Information Processing Standard) name file used with the TIGER thinned boundary files

Syntax

```
struc = fipsname
struc = fipsname(filename)
```

Background

The TIGER thinned boundary files provided by the U.S. Census use FIPS codes to identify geographic entities. This function reads the FIPS files as provided with the TIGER files. These files generally have names of the format *_name.dat*.

Description

`struc = fipsname` opens a file selection window to pick the file, reads the FIPS codes and returns them in a structure.

`struc = fipsname(filename)` reads the specified file.

Remarks

The FIPS name files, along with the TIGER thinned boundary files are available over the Internet at:

```
ftp://ftp.census.gov/pub/tiger/boundary/
```

Example

```
struc = fipsname('st_name.dat')
struc =
1x57 struct array with fields:
    name
    id

s(1)
ans =
    name: 'Alabama'
      id: 1
```

See Also

`tigermif`	Read TIGER MapInfo Interchange Format thinned boundary files
`tigerp`	Read TIGER ArcInfo Format thinned boundary files

gshhs

Purpose

Read the Global Self-consistent Hierarchical High-resolution Shoreline data

Syntax

```
struc = gshhs(filename)
struc = gshhs(filename,latlim,lonlim)
gshhs(filename,'createindex')
```

Background

The Global Self-consistent Hierarchical High-resolution Shoreline was created by Paul Wessel of the University of Hawaii and Walter H. F. Smith of the NOAA Geosciences Lab. At the full resolution the data requires 85 Megabytes uncompressed, but lower resolution versions are also provided. This database includes coastlines, major rivers, and lakes.

Description

`struc = gshhs(filename)` reads the specified GSHHS file and extracts data for the entire world. The result is returned as a geographic data structure. Each element of `struc` represents a unique polygon. The tag field for each element contains the topographic level represented by the polygon, and will be either `'land'`, `'lake'`, `'island'` for an island in a lake, or `'pond'` for a pond on an island in a lake. GSHHS files have file names of the form `'gshhs_X.b'`, where `X` is one of the letters '`c`', '`l`', '`i`', '`h`' and '`f`', corresponding to increasing resolution (and file size).

`struc = gshhs(filename,latlim,lonlim)` reads the data for the part of the world within the latitude and longitude limits. The limits must be two-element vectors in units of degrees. Longitude limits should be between `[-180 195]`.

`gshhs(filename,'createindex')` creates an index file for faster reading. The index file has the same name the GSHHS data file, but with the extension '`i`', instead of '`b`'. The file is written in the present working directory, which can be identified with the command `pwd`. This file is needed for acceptable performance with the larger datasets. No map data is returned while creating the index.

Example

Read all of the lowest resolution database.

```
s = gshhs('gshhs_c.b')
```

Read the intermediate resolution database for South America.

```
s = gshhs('gshhs_i.b',[-60 -15],[-90 -30])
```

Read the full resolution file for East and West Falkland Islands (Islas Malvinas).

```
s = gshhs('gshhs_f.b',[-55 -50],[-65 -55])
```

Create the index file for the high resolution database.

```
gshhs('gshhs_h.b','createindex')
```

Limitations

Polygons with more than 1 million points are treated as line objects rather than patches. This occurs only for the full resolution file.

Remarks

The GSHHS data in various resolutions is available over the Internet from

ftp://ftp.ngdc.noaa.gov/MGG/shorelines and

ftp://kiawe.soest.hawaii.edu/pub/wessel/gshhs/

Information on the datasets is available from

http://www.ngdc.noaa.gov/mgg/shorelines/gshhs.html

If you are extracting data within smaller geographic limits, it is much faster to create the index file first, and then extract the data. With very large amounts of data, you may want to plot the data as `NaN`-clipped lines, rather than a very larger number of patches. Use `extractm` to combine the data into one vector.

See Also

`extractm`	Extracts vector data from geographic data structures
`vmap0data`	Extracts selected data from the Vector Map Level 0 CD-ROMs
`dcwdata`	Extracts selected data from the Digital Chart of the World
`tgrline`	TIGER/Line data extraction
`tigermif`	Reads the TIGER MIF thinned boundary file
`tigerp`	Reads the TIGER p and pa thinned boundary file (ArcInfo format)

gtopo30

Purpose

Read elevation data from GTOPO30 into a regular matrix map

Syntax

```
[map,maplegend] = gtopo30(filename,scalefactor)
[map,maplegend] = gtopo30(filename,scalefactor,latlim,lonlim)
```

Background

A global model of elevations on a 30 arc-second grid (approximately 1 km) has been developed by the Earth Resources Observation System Data Center and is available over the Internet. The model uses a variety of international data sources, but is primarily based on raster data from the Digital Terrain Elevation Model (DTED) and vector data from the Digital Chart of the World (DCW). Gridded elevations are computed from DCW elevation contours, points and drainage lines using ANUDEM, a Digital Elevation Model generator developed by the Australian National University. The dataset consists of a total of 21,600 rows and 43,200 columns on 33 tiles, generally covering 40 by 50 degrees. Each tile requires about 15 megabytes compressed, 80 megabytes uncompressed and twice that to extract.

Description

`[map,maplegend] = gtopo30(`*filename*`,scalefactor)` reads the GTOPO30 files and returns a regular matrix map. The *filename* is given as a string, which does not include the extension. If the files are not found on the MATLAB path, they can be selected though a dialog box. `scalefactor` is an integer, which when equal to 1 gives the data at its full resolution. When `scalefactor` is an integer number larger than one, every n^{th} point is returned. This scale factor must divide evenly into the number of rows and columns in the data file. The map data is returned as a matrix of elevations and an associated regular matrix map legend.

`[map,maplegend] = gtopo30(`*filename*`,scalefactor,latlim,lonlim)` allows a subset of the map data to be read. The limits of the desired data are specified as vectors of latitudes and longitudes in degrees. The elements of `latlim` and `lonlim` must be in ascending order.

Remarks

The data is for elevations only. Elevations are given in meters above mean sea level using WGS 84 as a horizontal datum. Areas with no data, such as the oceans, are coded with `NaN`s.

The data is available over the Internet via anonymous ftp from:

ftp://edcftp.cr.usgs.gov/pub/data/gtopo30/global.

The data and some documentation is also available over the World-Wide-Web from:

http://edcwww.cr.usgs.gov/landdaac/gtopo30/gtopo30.html and

http://edcwww.cr.usgs.gov/landdaac/gtopo30/README.html.

Examples

Extract every 20th point from the tile covering the northeastern United States and eastern Canada. Provide an empty file name, and select the file interactively.

```
[map,maplegend] = gtopo30([],20);
size(map)
ans =
   300   240
```

Extract a subset of the data for Massachusetts at the full resolution.

```
latlim = [41 42.5]; lonlim = [-73 -69.9];
[map,maplegend] = gtopo30('W100N90',1,latlim,lonlim);
size(map)
ans =
   180   391
```

Replace the NaNs in the ocean with –1 to color them blue.

```
map(isnan(map)) = -1;
```

See Also

dted	Read Digital Terrain Elevation Data (DTED) data
satbath	Global 2-minute topography from satellite bathymetry
tbase	Read data from the TerrainBase dataset
usgsdem	Read USGS digital elevation maps

loadmoonalb

Purpose

Load albedo image of the Earth's moon

Syntax

```
loadmoonalb
```

Description

`loadmoonalb` loads an albedo map of the Earth's moon into the regular matrix map variables `moonalb` and `moonalblegend`. The string variable `CAPTION` contains a description and source of the data.

Example

```
loadmoonalb
whos
  Name                Size          Bytes  Class

  CAPTION            20x82           3280  char array
  moonalb          540x1080        4665600  double array
  moonalblegend      1x3                24  double array
```

See Also

`moontopo.mat` Moon topography

Purpose

Read the Fifth Fundamental Catalog of Stars

Syntax

```
struc = readfk5(filename)
struc = readfk5(filename,d)
```

Background

The Fifth Fundamental Catalog of Stars (FK5), Parts I and II, is a compilation of data on more than 4500 stars. The catalog contains positions, errors in positions, proper motions, and characteristics such as magnitudes, spectral types, parallaxes, and radial velocities. There are also cross-references to the identities of stars in other catalogs. It was compiled by researchers at the Astronomisches Rechen-Institut in Heidelberg.

Description

`struc = readfk5(filename)` reads the FK5 file and returns the contents in a structure. Each star is an element in the structure, with the different data items stored in appropriately named fields.

`struc = readfk5(filename,struc)` appends the data in the file to the existing structure, `struc`.

Remarks

Positions are given in terms of right ascension and declination. The section entitled "Astronomical Data" in Chapter 7 of the *Mapping Toolbox User's Guide* shows how to convert these to latitude and longitude for display by the Mapping Toolbox.

The Fifth Fundamental Catalog of Stars (FK5), Parts I and II, is available over the Internet by anonymous FTP at the following:

```
ftp://adc.gsfc.nasa.gov/pub/adc/archives/catalogs/1/1149A/
ftp://adc.gsfc.nasa.gov/pub/adc/archives/catalogs/1/1175/
```

The readme files have more complete descriptions of the data.

Examples

```
FK5 = readfk5('FK5.dat');
FK5e = readfk5('FK5_ext.dat');
whos
  Name       Size          Bytes  Class
  FK5        1x1535      5042752  struct array
  FK5e       1x3117     10226424  struct array

FK5e(1)
ans =
```

```
       FK5: 2003
       RAh: 0
       RAm: 5
       RAs: 1.1940
     pmRA: 0.6230
       DEd: 27
       DEm: 40
       DEs: 29.0100
     pmDE: -1.1100
  RAh1950: 0
  RAm1950: 2
  RAs1950: 26.5900
 pmRA1950: 0.6210
  DEd1950: 27
  DEm1950: 23
  DEs1950: 47.4400
 pmDE1950: -1.1100
 EpRA1900: 51.7200
    e_RAs: 2
   e_pmRA: 9
 EpDE1900: 46.8200
    e_DEs: 3.4000
   e_pmDE: 14
     Vmag: 6.4700
   n_Vmag: ''
   SpType: 'G5'
      plx: []
       RV: 12
    AGK3R: '38'
      SRS: ''
       HD: '225292'
       DM: 'BD+26 4744'
       GC: '48'
```

See Also

`dms2deg`	Convert angles from deg:min:sec to degrees
`hms2hr`	Convert time from hrs:min:sec to hours
`scatterm`	Construct a thematic map with proportional symbols

References

See references [5] and [6] in the Bibliography located at the end of this chapter.

Purpose

Read predicted global 2-minute (4 km) topography from satellite bathymetry

Syntax

```
[latgrat,longrat,map] = satbath
[latgrat,longrat,map] = satbath(scalefactor)
[latgrat,longrat,map] = satbath(scalefactor,latlim,lonlim)
[latgrat,longrat,map] = satbath(scalefactor,latlim,lonlim,gsize)
```

Background

This is a global bathymetric model derived from ship soundings and satellite altimetry by W.H.F. Smith and D. T. Sandwell. The model was developed by iteratively adjusting gravity anomaly data from Geosat and ERS-1 against historical track line soundings. This technique takes advantage of the fact that gravity mirrors the large variations in the ocean floor as small variations in the height of the oceans surface. The computational procedure uses the ship track line data to calibrate the scaling between the observed surface undulations and the inferred bathymetry. Land elevations are reduced resolution versions of GTOPO30 data.

Description

`[latgrat,longrat,map] = satbath` reads the global topography file for the entire world, returning every 50th point. The result is returned as a general matrix map.

`[latgrat,longrat,map] = satbath(scalefactor)` returns the data for the entire world, subsampled by the integer `scalefactor`. A `scalefactor` of 10 returns every 10th point. The matrix at full resolution has 6336 by 10800 points.

`[latgrat,longrat,map] = satbath(scalefactor,latlim,lonlim)` returns data for the specified region. The returned data will extend slightly beyond the requested area. If omitted, the entire area covered by the data file is returned. The limits are two-element vectors in units of degrees, with `latlim` in the range `[-90 90]` and `lonlim` in the range `[-180 180]`.

`[latgrat,longrat,map] = satbath(scalefactor,latlim,lonlim,gsize)` controls the size of the graticule matrices. `gsize` is a two-element vector containing the number of rows and columns desired. If omitted, a graticule the size of the map is returned.

Remarks

Land elevations are given in meters above mean sea level. The data is stored in a Mercator projection grid. As a result, spatial resolution varies with latitude. The grid spacing is 2 minutes (about 4 kilometers) at the equator.

This data is available over the Internet, but subject to copyright. The data file is binary, and should be transferred with no line-ending conversion or byte-swapping. This function carries out any byte-swapping that may be required. The data requires about 133 Megabytes uncompressed.

The data is available over the Internet via anonymous FTP from

<ftp://topex.ucsd.edu/pub/global_topo_2min/topo_6.2.img>.

Some documentation is also available over the World-Wide-Web from

<http://topex.ucsd.edu/marine_topo/mar_topo.html>

and

<http://www.ngdc.noaa.gov/mgg/announcements/announce_predict.html>.

Examples

Read the data for the Falklands Islands (Islas Malvinas) at full resolution.

```
[latgrat,longrat,mat] = satbath(1,[-55 -50],[-65 -55]);
whos
  Name          Size         Bytes  Class

  latgrat     247x301       594776  double array
  longrat     247x301       594776  double array
  mat         247x301       594776  double array
```

See Also

`tbase`	TerrainBase Global 5-Min digital terrain data extraction
`gtopo30`	30-Arc-Sec global digital elevation data extraction
`egm96geoid`	15-minute gridded geoid heights from the EGM96 geoid model of the Earth

Purpose

TerrainBase global 5-minute digital terrain data extraction

Syntax

```
[map,maplegend] = tbase(scalefactor)
[map,maplegend] = tbase(scalefactor,latlim,lonlim)
```

Background

TerrainBase is a global model of terrain and bathymetry on a regular 5-minute grid (approximately 10 km resolution). It is a compilation of the best available public domain data from almost 20 different sources, including the DCW-DEM and ETOPO5. The model is currently under development and will be updated as new data sources become available. The dataset was created by the National Geophysical Data Center and World Data Center-A for Solid Earth Geophysics in Boulder, Colorado.

Description

`[map,maplegend] = tbase(scalefactor)` reads the data for the entire world, reducing the resolution of the data by the specified scale factor. The result is returned as a regular matrix map and an associated map legend.

`[map,maplegend] = tbase(scalefactor,latlim,lonlim)` reads the data for the part of the world within the latitude and longitude limits. The limits must be two-element vectors in units of degrees.

Remarks

Elevations and depths are given in meters above or below mean sea level.

The `tbase.bin` file is available on CD-ROM from:

```
NOAA/NGDC
Mail Code E/GC3
325 Broadway
Boulder, CO  80303
USA
Tel: (303) 497-6338
Fax: (303) 497-6513
```

and via the Internet at:

```
ftp://ftp.ngdc.noaa.gov/Solid_Earth/CD_ROMS/TerrainBase_94/
   data/
```

No byte-swapping or line-ending conversion is required.

A summary of the model can be found at:

```
ftp://ftp.ngdc.noaa.gov/Solid_Earth/Topography/tbase_5min/
   tbase.txt
```

Examples

Read every tenth point in the dataset:

```
[map,maplegend] = tbase(10);
whos
  Name              Size         Bytes  Class

  map            216x432       746496  double array
  maplegend        1x3             24  double array

limitm(map,maplegend)
ans =
   -90    90     0   360
```

Read data for Korea and Japan at the full resolution:

```
scalefactor = 1; latlim = [30 45]; lonlim = [115 145];
[map,maplegend] = tbase(scalefactor,latlim,lonlim);
whos map
  Name      Size          Bytes  Class

  map     180x360        518400  double array
```

See Also

`gtopo30`	Read elevation data from GTOPO30
`etopo5`	Read data from the ETOPO5 dataset
`usgsdem`	Read USGS digital elevation maps

Purpose

Read data from U.S. Census Bureau TIGER/Line files

Syntax

```
[CL,PR,SR,RR,H,AL,PL] = tgrline(filename)
[CL,PR,SR,RR,H,AL,PL] = tgrline(filename,year)
[CL,PR,SR,RR,H,AL,PL] = tgrline(filename,year,countyname)
```

Background

TIGER/Line files contain vector map data used to support mapping for the U.S. Census Bureau. TIGER is an acronym for Topographically Integrated Geographic Encoding and Referencing. These files contain data for political boundaries, including states, counties, Indian reservations and census tracts, as well as roads, railroads, hydrography, and landmarks. In addition to the geographically referenced information, the files also contain data to determine the address of an object. The data covers the United States of America and its territories or administrative units: Puerto Rico, the Virgin Islands of the United States, American Samoa, Guam, the Commonwealth of the Northern Mariana Islands, the Republic of Palau, the other Pacific entities that were part of the Trust Territory of the Pacific Islands (the Republic of the Marshall Islands and the Federated States of Micronesia), and the Midway Islands. The most common application of this data is to commercial CD-ROM road atlases.

Description

`[CL,PR,SR,RR,H,AL,PL] = tgrline(filename)` reads the set of 1994 TIGER/Line files which share the same filename, but different extensions. The results are returned in a set of geographic data structures containing the county boundaries, primary roads, secondary roads, railroads, hydrography, area landmarks, and point landmarks, respectively. The data is tagged with the name of the features.

`[CL,PR,SR,RR,H,AL,PL] = tgrline(filename,year)` reads the TIGER/Line files in the format from the specified year. The files are updated periodically, and the format and file name extensions may change from year to year. Valid years are 1990, 1992, 1994, and 1995.

`[CL,PR,SR,RR,H,AL,PL] = tgrline(filename,year,countyname)` uses the string *countyname* to tag the county data.

Remarks

This function reads only a subset of the data in the TIGER/Line files. For example, the function does not return local roads, Zip codes or census tract numbers.

TIGER/line files are available from the U.S. Census Bureau on CD-ROMs. Ordering information and an overview of other Census Bureau data products is available at:

```
http://www.census.gov/ftp/pub/geo/www/tiger/
```

Example files are available over the Internet at:

```
ftp://ftp.census.gov/pub/tiger/line/
```

Examples

Read from the data for Washington, D.C.:

```
[CL,PR,SR,RR,H,AL,PL] = tgrline('TGR11001',1994,'Wash,DC');
```

See Also

`dcwdata`	Read selected data from the Digital Chart of the World
`tigermif`	Read TIGER MapInfo Interchange Format thinned boundary files
`tigerp`	Read TIGER ArcInfo Format thinned boundary files

Purpose

Read TIGER MIF (MapInfo Interchange Format) thinned boundary files

Syntax

```
tigermif(namesstruc)
tigermif(namesstruc,filename)
tigermif(namesstruc,filename,pstruc)
tigermif(namesstruc,filename,pstruc,tstruc)
tigermif(namesstruc,filename,pstruc,tstruc,getcodes)
pstruc = tigermif(...)
[pstruc,tstruc] = tigermif(...)
```

Background

TIGER Thinned Boundary files are lower resolution extracts from the U.S. Census Bureau's detailed TIGER/Line database. U.S. state and county boundaries are available in the MapInfo Interchange format (MIF).

Description

`tigermif(namesstruc)` reads a TIGER thinned boundary file in the MIF format. The user selects the file interactively, but must provide the structure containing the names (as returned by the `fipsname` function). The patch data is returned in a Mapping Toolbox geographic data structure.

`tigermif(namesstruc,`*`filename`*`)` reads the MIF file named in the string *filename*. The filename is provided with the '.MIF' extension. If the file is not found, a dialog box is activated to allow the user to select a file interactively.

`tigermif(namesstruc,`*`filename`*`,pstruc)` appends the patch data to the existing structure, `pstruc`.

`tigermif(namesstruc,`*`filename`*`,pstruc,tstruc)` appends the data in the file to the existing patch and text geographic data structures, `pstruc` and `tstruc`. The text structure contains labels for the patches. This form would be used with two output arguments. The arguments for the existing structures can be set to empty matrices if none are available.

`tigermif(namesstruc,`*`filename`*`,pstruc,tstruc,getcodes)` returns only the data matching the scalar or vector of numeric FIPS codes.

`pstruc = tigermif(...)` saves the returned patch data in `pstruc`.

`[pstruc,tstruc] = tigermif(...)` saves the returned patch data in `pstruc` and text labels in `tstruc`. Both are geographic data structures.

Remarks

The data files are available over the Internet from:

```
ftp://ftp.census.gov/pub/tiger/boundary/
```

Extremely limited documentation on the files is available on the World Wide Web from:

```
http://www.census.gov/ftp/pub/geo/www/tiger/mif.txt
```

and

```
http://www.census.gov/ftp/pub/geo/www/tiger/resource.html
```

Examples

Read the names file (contains the names of U.S. states and territories):

```
namestruc = fipsname('st_name.dat')
namestruc =
1x57 struct array with fields:
    name
    id
```

Read the file containing Hawaii's thinned state boundaries and text labels into a Mapping Toolbox geographic data structure:

```
[ps,ts] = tigermif(namestruc,'ST15.MIF')
ps =
              lat: [1585x1 double]
             long: [1585x1 double]
             type: 'patch'
    otherproperty: {}
              tag: 'Hawaii'
         altitude: []
ts =
              lat: 21.1343
             long: -157.9524
             type: 'text'
              tag: 'maptext'
    otherproperty: {1x2 cell}
           string: {1x1 cell}
         altitude: []
```

Read the file containing Alaska's thinned state boundaries, and append it to the Hawaii data:

```
[ps,ts] = tigermif(namestruc,'ST02.MIF',ps,ts)
ps =
1x2 struct array with fields:
    lat
    long
```

```
    type
    otherproperty
    tag
    altitude
ts =
1x2 struct array with fields:
    lat
    long
    type
    tag
    otherproperty
    string
    altitude
```

Get the state boundaries and text labels for part of New England. The FIPS codes for Connecticut, Massachusetts, and Rhode Island are 9, 25, and 44, respectively:

```
[ps,ts] = tigermif(namestruc,'ST_LOW48.MIF',[],[],[9 25 44])
ps =
1x3 struct array with fields:
    lat
    long
    type
    otherproperty
    tag
    altitude
ts =
1x3 struct array with fields:
    lat
    long
    type
    tag
    otherproperty
    string
    altitude
```

See Also

`dcwdata`	Read selected data from the *Digital Chart of the World*
`fipsname`	Read TIGER thinned boundary file FIPS names
`tgrline`	Read data from TIGER/Line files
`tigerp`	Read TIGER ArcInfo Format thinned boundary files

Purpose

Read TIGER p and pa (ArcInfo Format) thinned boundary files

Syntax

```
tigerp(namestruc)
tigerp(namestruc,filename)
tigerp(namestruc,filename,pstruc)
tigerp(namestruc,filename,pstruc,tstruc)
tigerp(namestruc,filename,pstruc,tstruc,getcodes)
pstruc = tigerp(...)
[pstruc,tstruc] = tigerp(...)
```

Background

TIGER Thinned Boundary files are lower resolution extracts from the U.S. Census Bureau's more detailed TIGER/Line database. State, county, minor civil division, census tract/block numbering area, american Indian reservation/ Alaska native village statistical area, Alaska native regional corporation, urbanized areas, metropolitan areas, and congressional district boundaries are available in the ArcInfo format.

Description

`tigerp(namesstruc)` reads a TIGER thinned boundary file in the ArcInfo format. The user selects the file interactively, but must provide the structure containing the names (as returned by the `fipsname` function). The patch data is returned in a Mapping Toolbox geographic data structure.

`tigerp(namesstruc,`*`filename`*`)` reads the ArcInfo file named in the string *filename*. The file name is provided without the `'_p'` or `'_pa'` extension.

`tigerp(namesstruc,`*`filename`*`,pstruc)` appends the patch data to the existing structure, `pstruc`.

`tigerp(namesstruc,`*`filename`*`,pstruc,tstruc)` appends the data in the file to the existing patch and text geographic data structures, `pstruc` and `tstruc`. The text structure contains labels for the patches. This form would be used with two output arguments. The arguments for the existing structures can be set to empty matrices if none are available.

`tigerp(namesstruc,`*`filename`*`,pstruc,tstruc,getcodes)` returns only the data matching the scalar or vector of numeric FIPS codes.

`pstruc = tigerp(...)` saves the returned patch data in `pstruc`.

`[pstruc,tstruc] = tigerp(...)` saves the returned patch data in `pstruc` and text labels in `tstruc`. Both are geographic data structures.

Remarks

Coordinate values are based on Clarke's spheroid of 1866 and the North American Datum, 1927 (NAD27).

The data files are available over the Internet from:

```
ftp://ftp.census.gov/pub/tiger/boundary
```

Documentation for the files is available from:

```
ftp://ftp.census.gov/pub/tiger/boundary/readme.txt
```

Examples

Read the names file with the names of all counties in the U.S. and territories. This file is in FIPS format:

```
namestruc = fipsname('co_name.dat')
namestruc =
1x3248 struct array with fields:
    name
    id
```

Read the file containing Alaska's thinned county boundaries into a Mapping Toolbox geographic data structure:

```
[ps,ts] = tigerp(namestruc,'co_02_p.dat')
ps =
1x26 struct array with fields:
    lat
    long
    type
    otherproperty
    altitude
    tag
ts =
1x26 struct array with fields:
    lat
    long
    type
    tag
    otherproperty
    altitude
    string
```

Read only the Aleutians East and West:

```
[ps,ts] = tigerp(namestruc,'co_02_p.dat',[],[],[2013 2016])
ps =
1x2 struct array with fields:
    lat
    long
    type
    otherproperty
    altitude
    tag
ts =
1x2 struct array with fields:
    lat
    long
    type
    tag
    otherproperty
    altitude
    string
```

See Also

`dcwdata`	Read selected data from the Digital Chart of the World
`fipsname`	Read TIGER thinned boundary file FIPS names
`tgrline`	Read data from TIGER/Line files
`tigermif`	Read TIGER MapInfo Interchange Format thinned boundary files

usgs24kdem

Purpose

Read USGS 7.5 minute 1:24,000 (30 m) Digital Elevation Model files

Syntax

```
[latgrat,longrat,mat] = usgs24kdem
[latgrat,longrat,mat] = usgs24kdem(filename)
[latgrat,longrat,mat] = usgs24kdem(filename,scalefactor)
[latgrat,longrat,mat] = usgs24kdem(filename,scalefactor,
   latlim,lonlim)
[latgrat,longrat,mat] = usgs24kdem(filename,scalefactor,
   latlim,lonlim,gsize)
[latgrat,longrat,mat,A,B] = usgs24kdem(...)
```

Background

The U. S. Geological Survey has created a series of digital elevation models based on their paper 1:24,000 scale maps. The grid spacing for these elevations models is 30 meters on a Universal Transverse Mercator grid. Each file covers a 7.5 minute quadrangle. The map and data series is available for much of the contiguous United States, Hawaii, and Puerto Rico. The data has been released in a number of formats. This function reads the data in the 'standard' file format.

Description

`[latgrat,longrat,mat] = usgs24kdem` reads a USGS 1:24,000 Digital Elevation Map (DEM) file in standard format. The file is selected interactively. The entire file is read and subsampled by a factor of 5, and returned as a general matrix map.

`[latgrat,longrat,mat] = usgs24kdem(`*filename*`)` specifies the name of the DEM file.

`[latgrat,longrat,mat] = usgs24kdem(`*filename*`,scalefactor)` subsamples the file by the `scalefactor`. If omitted, the default is 5, which returns every 5th point.

`[latgrat,longrat,mat] = usgs24kdem(`*filename*`,scalefactor,latlim,lonlim)` returns data for the requested geographic area. The area is specified as two-element vectors in units of degrees. The data will extend somewhat outside the requested area. If omitted, the entire area covered by the DEM file is returned.

`[latgrat,longrat,mat] = usgs24kdem(`*filename*`,scalefactor,latlim,lonlim, gsize)` also controls the graticule size. `gsize` is a two-element vector specifying the number of rows

and columns in the latitude and longitude matrices. If omitted, a graticule the same size as the map is returned.

`[latgrat,longrat,mat,A,B] = usgs24kdem(...)` also returns the contents of the `A` and `B` records of the DEM file. The `A` record is a header to the file containing descriptions of the data. The `B` record is the raw profile data from which the matrix map is constructed.

Examples

Get the 1:24,000 DEM for south San Francisco, available from the Bay Area Regional Database at <ftp://bard.wr.usgs.gov/bard/dem/dems24k/sanfrancisco/sanfranciscos.dem>. Read the entire file, taking every second point.

```
[latgrat,longrat,mat] = usgs24kdem('sanfranciscos.dem',2);
whos
  Name          Size         Bytes  Class

  latgrat     232x186       345216  double array
  longrat     232x186       345216  double array
  mat         232x186       345216  double array
```

Read the DEM at full resolution, limiting the area to the San Bruno mountains. These limits can be defined using inputm on a display of the above data. Also return the header record.

```
[latlim,lonlim] = inputm(2);
[latgrat,longrat,mat,A] = usgs24kdem('sanfranciscos.dem',1,...
                                     latlim,lonlim);
whos
  Name          Size         Bytes  Class

  A             1x1           4740  struct array
  latgrat     162x202       261792  double array
  latlim        1x2             16  double array
  longrat     162x202       261792  double array
  lonlim        1x2             16  double array
  mat         162x202       261792  double array


A
A =

                    Quadranglename: 'SAN FRANCISCO SOUTH, CA  BIG
                                    BASIN DEM'
                   TextualInfo: 'WMC                          CTOG'
                              Filler: ''
                         ProcessCode: ''
                             Filler2: ''
                  SectionalIndicator: ''
                        MCoriginCode: ''
                        DEMlevelCode: 2
                  ElevationPatternCode: 'regular'
      PlanimetricReferenceSystemCode: 'UTM'
                                Zone: 10
                ProjectionParameters: [15x1  double]
                     HorizontalUnits: 'meters'
                      ElevationUnits: 'feet'
                   NsidesToBoundingBox: 4
                           BoundingBox: [ 8x1  double]
                      MinMaxElevations: [ 2x1  double]
                         RotationAngle: 0
```

```
 AccuracyCode: 'accuracy information in record C'
    XYZresolutions: [ 3x1  double]
        NrowsCcols: [ 2x1  double]
     MaxPcontourInt: ''
SourceMaxCintUnits: ''
   SmallestPrimary: ''
SourceMinCintUnits: ''
     DataSourceDate: ''
    DataInspRevDate: ''
        InspRevFlag: ''
DataValidationFlag: ''
   SuspectVoidFlag: ''
      VerticalDatum: ''
    HorizontalDatum: ''
        DataEdition: ''
        PercentVoid: ''
```

Remarks

This function reads USGS DEM files stored in the UTM projection. Use `usgsdem` for data stored in geographic grids.

The number of points in a file will vary with the geographic location. Unlike the USGS DEM products that use an equal angle grid, the UTM projection grid DEMs cannot simply be concatenated to cover larger areas. There may be data gaps between DEMs.

The data files can be obtained by contacting the U.S. Geological Survey. Other agencies have made some of the data available online. Source for data for the San Francisco Bay area are

<http://bard.wr.usgs.gov/>

and

<ftp://bard.wr.usgs.gov/bard/dem/dems24k/>.

Extensive documentation on the data format and standards are available from

<http://mapping.usgs.gov/www/ti/DEM/standards_dem.html>

and

<ftp://mapping.usgs.gov/pub/ti/DEM/demguide/>.

The DEM files are ASCII files, and can be transferred as text. Line-ending conversion is not necessarily required.

See Also

`usgsdem`	USGS 1-Degree (3-arc-sec resolution) digital elevation data
`dted`	U. S. Department of Defense Digital Terrain Elevation Data (DTED) data
`gtopo30`	30-Arc-Sec global digital elevation data
`tbase`	TerrainBase Global 5-Min digital terrain data
`etopo5`	ETOPO5 Global 5-Min digital terrain data

Purpose

Read USGS 1-Degree (3-arc-sec resolution) DEM data

Syntax

```
[map,maplegend] = usgsdem(filename,scalefactor)
[map,maplegend] = usgsdem(filename,scalefactor,latlim,lonlim)
```

Background

The U.S. Geological Survey has made available a set of digital elevation maps of 1-degree quadrangles covering the contiguous United States, Hawaii, and limited portions of Alaska. The data is on a regular grid with a spacing of 30 arc-seconds (or about 100-meter resolution). 1-degree DEMs are also referred to as *3-arc-second* or *1:250,000 scale* DEM data.

The data is derived from the U.S. Defense Mapping Agency's DTED-1 digital elevation model, which itself was derived from cartographic and photographic sources. The cartographic sources were maps from the 7.5-minute through 1-degree series (1:24,000 scale through 1:250,000 scale).

Description

`[map,maplegend] = usgsdem(filename,scalefactor)` reads the specified file and returns the data in a regular matrix map. The data can be read at full resolution (`scalefactor` = 1), or can be downsampled by the `scalefactor`. A `scalefactor` of 3 returns every third point, giving 1/3 of the full resolution.

`[map,maplegend] = usgsdem(filename,scalefactor,latlim,lonlim)` reads data within the latitude and longitude limits. These limits are two-element vectors with the minimum and maximum values specified in units of degrees.

Remarks

The grid for the digital elevation maps is based on the 1984 World Geodetic System (WGS84). Older DEMs were based on WGS72. Elevations are in meters relative to National Geodetic Vertical Datum of 1929 (NGVD 29) in the continental U.S. and local mean sea level in Hawaii.

The absolute horizontal accuracy of the DEMs is 130 meters, while the absolute vertical accuracy is ±30 meters. The relative horizontal and vertical accuracy is not specified, but is probably much better than the absolute accuracy.

These DEMs have a grid spacing of 3 arc-seconds in both the latitude and longitude directions. The exception is DEM data in Alaska, where latitudes between 50 and 70 degrees North have grid spacings of 6 arc-seconds, and latitudes greater than 70 degrees North have grid spacings of 9 arc-seconds.

Statistical data in the files is not returned.

The DEMs are available over the Internet from:

```
ftp://edcftp.cr.usgs.gov/pub/data/DEM/250/
```

The files are available sorted by state or can be selected from an index map at:

```
http://edcwww.cr.usgs.gov/doc/edchome/ndcdb/2MIL/2milmap.html
```

Some documentation for the datasets is available at:

```
http://edcwww.cr.usgs.gov/glis/hyper/guide/1_dgr_dem
```

Examples

Read every fifth point in the file containing part of Rhode Island and Cape Cod:

```
[map,maplegend] = usgsdem('providence-e',5);
```

Read the elevation data for Martha's Vineyard at full resolution:

```
[map,maplegend] = usgsdem('providence-e',1,...
   [41.2952 41.4826],[-70.8429 -70.4392]);
whos map
  Name      Size          Bytes  Class

  map     226x485       876880  double array
```

See Also

`usgs24kdem`	Read elevation data from 1:24,000 USGS DEM files
`gtopo30`	Read elevation data from GTOPO30
`etopo5`	Read data from the ETOPO5 dataset
`tbase`	Read data from the TerrainBase dataset
`usgsdems`	Read USGS digital elevation map file names

References

See reference [7] in the Bibliography located at the end of this chapter.

Purpose

USGS 1-Degree (3-arc-sec resolution) DEM file names

Syntax

```
[fname,qname] = usgsdems(latlim,lonlim)
```

Background

The U.S. Geological Survey has made available a set of digital elevation maps of 1-degree quadrangles covering the contiguous United States, Hawaii, and limited portions of Alaska. These are referred to as *1-degree*, *3-arc second* or *1:250,000 scale* DEMs. Because the file names of these 1 degree datasets are taken from the names of cities or features in the quadrangle, determining which files are needed to cover a particular region generally requires consulting an index map or other reference. This function takes the place of such a reference by returning the file names for a given geographic region.

Description

`[fname,qname] = usgsdems(latlim,lonlim)` returns cell arrays of the DEM file names and quadrangle names covering the geographic region. The region is specified by scalar latitude and longitude points or two-element vectors of latitude and longitude limits in units of degrees.

Remarks

This function only returns file names for the contiguous United States.

Examples

Which files are needed to map part of New England?

```
usgsdems([41 44], [-72 -69])
ans =
    'providence-w'
    'providence-e'
    'chatham-w'
    'boston-w'
    'boston-e'
    'portland-w'
    'portland-e'
    'bath-w'
```

See Also

`usgsdem` Read USGS digital elevation maps

References

See reference [7] in the Bibliography located at the end of this chapter.

vmap0data

Purpose

Read selected data from the Vector Map Level 0

Syntax

```
struct1 = vmap0data(library,latlim,lonlim,theme,topolevel)
struct1 = vmap0data(devicename,library,...)
[struct1,struct2,...] = vmap0data(...,{topolevel1,topolevel2,...})
```

Background

The Vector Map (VMAP) Level 0 database represents the third edition of the Digital Chart of the World. The second edition was a limited release item published in 1995. The product is dual named to show its lineage to the original DCW, published in 1992, while positioning the revised product within a broader emerging-family of VMAP products. VMAP Level 0 is a comprehensive 1:1,000,000 scale vector basemap of the world. It consists of cartographic, attribute, and textual data stored on compact disc read only memory (CD-ROM). The primary source for the database is the National Imagery and Mapping Agency's (NIMA) Operational Navigation Chart (ONC) series. This is the largest scale unclassified map series in existence that provides consistent, continuous global coverage of essential basemap features. The database contains more than 1,900 megabytes of vector data and is organized into 10 thematic layers. The data includes major road and rail networks, major hydrologic drainage systems, major utility networks (cross-country pipelines and communication lines), all major airports, elevation contours (1000 foot(ft), with 500ft and 250ft supplemental contours), coastlines, international boundaries and populated places. The database can be accessed directly from the four optical CD-ROMs that store the database or can be transferred to a magnetic media. (From the NIMA Metadata referenced below).

Description

`struct = vmap0data(library,latlim,lonlim,theme,topolevel)` reads the data for the specified theme and topology level directly from the VMAP0 CD-ROM. There are four CDs, one for each of the libraries: `'NOAMER'` (North America), `'SASAUS'` (Southern Asia and Australia), `'EURNASIA'` (Europe and Northern Asia), and `'SOAMAFR'` (South America and Africa). The desired *theme* is specified by a two-letter code string. A list of valid codes is displayed when an invalid code, such as `'?'`, is entered. *topolevel* defines the type of data returned. It is a string containing `'patch'`, `'line'`, `'point'`, or `'text'`. The region of interest can be given as a point latitude and longitude or as a region with two-element vectors of latitude and longitude limits. The units of latitude and longitude are degrees. The data covering the requested region is returned, but

will include data extending to the edges of the tiles. The result is returned as a Mapping Toolbox geographic data structure.

`struct = vmap0data(`*`devicename`*`,`*`library`*`,...)` specifies the logical device name of the CD-ROM for computers that do not automatically name the mounted disk.

`[struct1, struct2,...] = vmap0data(...,{`*`topolevel1`*`,`*`topolevel2`*`,...})` reads several topology levels. The levels must be specified as a cell array with the entries 'patch', 'line', 'point' or 'text'. Entering {'all'} for the topology level argument is equivalent to {'patch', 'line', 'point', 'text'}. Upon output, the data structures are returned in the output arguments by topology level in the same order as they were requested.

Remarks

Latitudes and longitudes use WGS84 as a horizontal datum. Elevations and depths are in meters above mean sea level.

Some VMAP0 themes do not contain all topology levels. In those cases, empty matrices are returned.

Patches are broken at the tile boundaries. Setting the `EdgeColor` to 'none' and plotting the lines will give the map a normal appearance.

The major differences between VMAP0 and the DCW are the elimination of the gazette layer, addition of bathymetric data and updated political boundaries.

The VMAP0 is available from the following sources:

```
USGS Open-File Section
Box 25286
Denver, CO  80225
USA
Tel: (303) 236-7476
```

The price as of early 1998 is US$82.50 per four disk set.

Information on the VMAP0 data can be found at <http://164.214.2.54/mel/metadata/vmap0.meta.html>.

Examples

The *devicename* is platform dependent. On a MS-DOS based operating system it would be something like 'd:', depending on the logical device code assigned to the CD-ROM drive. On a UNIX operating system, the CD-ROM might be

mounted as '\cdrom', '\CDROM', '\cdrom1' or something similar. Check your computer's documentation for the right *devicename*.

```
s = vmap0data(devicename,'NOAMER',41,-69,'?','patch');

??? Error using ==> vmap0data
Theme not present in library NOAMER

Valid theme identifiers are:
libref : Library Reference
tileref: Tile Reference
bnd    : Boundaries
dq     : Data Quality
elev   : Elevation
hydro  : Hydrography
ind    : Industry
phys   : Physiography
pop    : Population
trans  : Transportation
util   : Utilities
veg    : Vegetation

BNDpatch = vmap0data(devicename,'NOAMER',...
                     [41 44],[-72 -69],'bnd','patch')
BNDpatch =
1x169 struct array with fields:
    type
    otherproperty
    altitude
    lat
    long
    tag
```

Other examples:

```
[TRtext,TRline] = vmap0data(devicename,'SASAUS',...
     [-48 -34],[164 180],'trans',{'text','line'});

[BNDpatch,BNDline,BNDpoint,BNDtext] = vmap0data(devicename,...
     'EURNASIA',-48 ,164,'bnd',{'all'});
```

See Also

`vmap0read`	Read a VMAP0 file
`vmap0rhead`	Read a VMAP0 file header
`displaym`	Project data contained in a map structure
`extractm`	Extract vector data from a map structure
`mlayers`	GUI for manipulating map layers

vmap0read

Purpose

Read a Digital Chart of the World file

Syntax

```
vmap0read
vmap0read(filepath,filename)
vmap0read(filepath,filename,recordIDs)
vmap0read(filepath,filename,recordIDs,field,varlen)

struc = vmap0read(...)
[struc,field] = vmap0read(...)
[struc,field,varlen] = vmap0read(...)
[struc,field,varlen,description] = vmap0read(...)
[struc,field,varlen,description,narrativefield] = vmap0read(...)
```

Background

The Vector Map level 0 (VMAP0) uses binary files in a variety of formats. This function determines the format of the file and returns the contents in a structure. The field names of this structure are the same as the field names in the VMAP0 file.

Description

`vmap0read` reads a VMAP0 file. The user selects the file interactively.

`vmap0read(`*filepath*`,`*filename*`)` reads the specified file. The combination [filepath filename] must form a valid complete filename.

`vmap0read(`*filepath*`,`*filename*`,recordIDs)` reads selected records or fields from the file. If `recordIDs` is a scalar or a vector of integers, the function returns the selected records. If `recordIDs` is a cell array of integers, all records of the associated fields are returned.

`vmap0read(`*filepath*`,`*filename*`,recordIDs,field,varlen)` uses previously read field and variable length record information to skip parsing the file header (see below).

`struc = vmap0read(...)` returns the file contents in a structure.

`[struc,field] = vmap0read(...)` returns the file contents and a structure describing the format of the file.

`[struc,field,varlen] = vmap0read(...)` also returns a vector describing which fields have variable length records.

`[struc,field,varlen,description] = vmap0read(...)` also returns a string describing the contents of the file.

`[struc,field,varlen,description,narrativefield] = vmap0read(...)` also returns the name of the narrative file for the current file.

Remarks

This function reads all VMAP0 files except index files (files with names ending in 'X'), thematic index files (files with names ending in 'TI'), and spatial index files (files with names ending in 'SI').

File separators are platform dependent. The *filepath* input must use appropriate file separators, which can be determined using the MATLAB `filesep` function.

Examples

The following examples use the Macintosh directory system and file separators for the path name:

```
s = vmap0read('VMAP:VMAPLV0:NOAMER:','GRT')
s =
                    id: 1
             data_type: 'GEO'
                 units: 'M'
       ellipsoid_name: 'WGS 84'
     ellipsoid_detail: 'A=6378137 B=6356752 Meters'
      vert_datum_name: 'MEAN SEA LEVEL'
      vert_datum_code: '015'
     sound_datum_name: 'N/A'
     sound_datum_code: 'N/A'
       geo_datum_name: 'WGS 84'
       geo_datum_code: 'WGE'
      projection_name: 'Dec. Deg. (unproj.)'

s = vmap0read('VMAP:VMAPLV0:NOAMER:TRANS:','INT.VDT')
s =
34x1 struct array with fields:
    id
    table
    attribute
    value
    description

s(1)
ans =
              id: 1
           table: 'aerofacp.pft'
       attribute: 'use'
           value: 8
     description: 'Military'
```

```
s = vmap0read('VMAP:VMAPLV0:NOAMER:TRANS:','AEROFACP.PFT',1)
s =
          id: 1
      f_code: 'GB005'
         iko: 'BGTL'
         nam: 'THULE AIR BASE'
         na3: 'GL52085'
         use: 8
         zv3: 77
     tile_id: 10
      end_id: 1

s = vmap0read('VMAP:VMAPLV0:NOAMER:TRANS:','AEROFACP.PFT',{1,2})
s =
1x4424 struct array with fields:
    id
    f_code
```

See Also

`vmap0data`	Read selected data from the VMAP0
`vmap0rhead`	Read a VMAP0 file header

vmap0rhead

Purpose

Read the header of a VMAP0 file

Syntax

```
vmap0rhead
vmap0rhead(filepath,filename)
vmap0rhead(filepath,filename,fid)

str = vmap0rhead(...)
```

Background

The Vector Map Level 0 (VMAP0) uses header strings in most files to document the contents and format of that file. This function reads the header string and displays a formatted version in the command window, or returns it as a string.

Description

`vmap0rhead` allows the user to select the header file interactively.

`vmap0rhead(`*filepath*`,`*filename*`)` reads from the specified file. The combination [*filepath filename*] must form a valid complete *filename*.

`vmap0rhead(`*filepath*`,`*filename*`,fid)` reads from the already open file associated with `fid`.

`vmap0rhead(...)`, with no output arguments, displays the formatted header information on the screen.

`str = vmap0rhead(...)` returns a string containing the VMAP0 header.

Remarks

This function reads all VMAP0 files except index files (files with names ending in '`X`'), thematic index files (files with names ending in '`TI`') and spatial index files (files with names ending in '`SI`').

File separators are platform dependent. The *filepath* input must use appropriate file separators, which may be determined using the MATLAB `filesep` function.

Examples

The following example uses the Macintosh file separators and pathname:

```
s = vmap0rhead('VMAP:VMAPLV0:NOAMER:','GRT')
s =
L;Geographic Reference Table;-;id=I,1,P,Row
Identifier,-,-,-,:data_type=T,3,N,Data
Type,-,-,-,:units=T,3,N,Units of Measure Code for
Library,-,-,-,:ellipsoid_name=T,15,N,Ellipsoid,-,-,-,:ellipsoid_
detail=T,50,N,Ellipsoid
Details,-,-,-,:vert_datum_name=T,15,N,Datum Vertical
Reference,-,-,-,:vert_datum_code=T,3,N,Vertical Datum
Code,-,-,-,:sound_datum_name=T,15,N,Sounding
Datum,-,-,-,:sound_datum_code=T,3,N,Sounding Datum
Code,-,-,-,:geo_datum_name=T,15,N,Datum Geodetic
Name,-,-,-,:geo_datum_code=T,3,N,Datum Geodetic
Code,-,-,-,:projection_name=T,20,N,Projection Name,-,-,-,:;

vmap0rhead('VMAP:VMAPLV0:NOAMER:TRANS:','AEROFACP.PFT')
L
Airport Point Feature Table
aerofacp.doc
id=I,1,P,Row Identifier,-,-,-,
f_code=T,5,N,FACC Feature Code,char.vdt,-,-,
iko=T,4,N,ICAO Designator,char.vdt,-,-,
nam=T,*,N,Name,char.vdt,-,-,
na3=T,*,N,Name,char.vdt,-,-,
use=S,1,N,Usage,int.vdt,-,-,
zv3=S,1,N,Airfield/Aerodrome Elevation (meters),int.vdt,-,-,
tile_id=S,1,N,Tile Reference ID,-,tile1_id.pti,-,
end_id=I,1,N,Entity Node Primitive ID,-,end1_id.pti,-,
```

See Also

`vmap0data`	Read selected data from the VMAP0
`vmap0read`	Read a VMAP0 file

Bibliography

[1] Military Specification: Digital Chart of the World Database (MIL-D-89009).

[2] Military Standard – Vector Product Format (MIL-STD-600006).

[3] The Development of the Digital Chart of the World, available from the U.S. Department of Commerce, National Technical Information Service, 5285 Port Royal Rd., Springfield, VA 22161.

[4] Data Announcement 88-MGG-02, Digital Relief of the Surface of the Earth. NOAA, National Geophysical Data Center, Boulder, Colorado, 1988.

[5] Fricke, W., Schwan, H., and Corbin, T. (in collaboration with Bastian, U., Bien, R., Cole, C., Jackson, R., Jaehrling, R., Jahreiss, H., Lederle, T., and Roeser, S.) 1991, Fifth Fundamental Catalogue FK5, Part II. The FK5 Extension, New Fundamental Stars, Veroeff. Astron. Rechen-Institut Heidelburg No. 33.

[6] Fricke, W., Schwan, H., and Lederle, T. (in collaboration with Bastian, U., Bien, R., Burkhardt, G., du Mont, B., Hering, R., Jaehrling, R., Jahreiss, H., Roeser, S., Schwerdtfeger, H. M., and Walter, H. G.) 1988, Fifth Fundamental Catalogue (FK5), Part I. The Basic Fundamental Stars, Veroeff. Astron. Rechen-Institut Heidelburg No. 32, catalog.

[7] Digital Elevation Models, National Mapping Program Technical Instructions, Data Users Guide 5, U.S. Geological Survey, Second Printing (Revised), Reston, Virginia, 1990.

Index

A

accuracy of map computations 1-131
Adams, O. S. 2-30
Adams, Oscar Sherman 2-106
Airy Minimum Error Azimuthal projection 2-20
Airy, George 2-20
aitoff 2-4
Aitoff projection 2-4, 2-66
Aitoff, David 2-4
Albers Equal-Area Conic projection 2-6
Albers, Heinrich Christian 2-6
almanac 1-22
American Geographical Society 2-90
American Polyconic projection 2-102
angledim 1-28
angles
  converting units 1-28, 1-98, 1-101, 1-118, 1-120, 1-288, 1-289
  normalizing range 1-258, 1-422
antipode 1-30
Apian, Peter 2-8
apianus 2-8
Apianus II projection 2-8
areaint 1-31
areamat 1-33
areaquad 1-36
ASCII file
  converting delimiters to NaNs 1-246
  reading 1-356
aut2geod 1-38
auxiliary sphere
  aut2geod 1-38
  calculating radius 1-321
  cen2geod 1-64
  cnf2geod 1-72
  geod2aut 1-156
  geod2cen 1-157
  geod2cnf 1-158
  geod2iso 1-159
  geod2par 1-160
  geod2rec 1-161
  iso2geod 1-200
  par2geod 1-264
  rec2geod 1-304
avhrrgoode 4-5
avhrrlambert 4-9
axes, Cartesian *See* Cartesian axes
axes, map *See* map axes
axes2ecc 1-39
axesm 1-40, 3-7
axesmui 3-7
axesscale 1-53
azimuth
  between track waypoints 1-202
  calculating 1-55, 3-89
  finding cross fix position 1-88
azimuth 1-55

B

Babinet projection 2-92
Balthasart Cylindrical projection 2-10, 2-44
balthsrt 2-10
Bartholomew, John 2-66
bearing *See* azimuth
behrmann 2-12
Behrmann Cylindrical projection 2-6, 2-12, 2-44
Behrmann, Walter 2-12
Bienewitz 2-8
Bolshoi Sovietskii Atlas Mira 2-14
Bolshoi Sovietskii Atlas Mira projection 2-14, 2-18
bonne 2-16

Bonne projection 2-16
Bonne, Rigobert 2-16
Bordone Oval projection 2-80
Braun 2-18
braun 2-18
Braun Perspective Cylindrical projection 2-14, 2-18, 2-58
breusing 2-20
Breusing Harmonic Mean projection 2-20
Breusing, F. A. Arthur 2-20
bries 2-22
Briesemeister projection 2-22, 2-66
Briesemeister, William 2-22
bsam 2-14
BSAM projection 2-14

C

camposm 1-57
camtargm 1-59
camupm 1-61
cart2grn 1-63
Cartesian axes, displaying 1-350
cassini 2-24
Cassini Cylindrical projection 2-24
Cassini de Thury, César François 2-24
Cassini projection 2-100
ccylin 2-26
cen2geod 1-64
Central Cylindrical projection 2-26, 2-130
Central projection 2-62
Ch'ien Lo-Chih 2-88
changem 1-65
clabelm 1-66
clegendm 1-68
Click-and-Drag property editor 3-64
clma 1-70
clmo 1-71, 3-19
clrmenu 3-20
cnf2geod 1-72
coast 1-73
collig 2-28
Collignon projection 2-28
Collignon, Édouard 2-28
colorm 3-21
colormap menu in figure window 3-20
colormaps
  digital elevation map 1-102, 3-27
  regular matrix map 3-21
  shaded relief map 1-348
combinations 1-74
combntns 1-74
comet3m 1-75, 3-23
cometm 1-76, 3-23
Conic projection 2-48
Conical Orthomorphic projection 2-76
contor3m 1-77, 3-25
contorm 1-79, 3-25
contour map
  adding legend 1-68
  creating 1-77, 1-79, 3-25
  labeling 1-66
contourfm 1-81
conversion
  angle units 1-28, 1-98, 1-101, 1-118, 1-120, 1-288, 1-289
  ASCII file delimiters 1-246
  coordinate types 1-63, 1-132, 1-172
  distance to string 1-109
  distance units 1-99, 1-116, 1-201, 1-257, 1-290, 1-353
  dms to matrix elements 1-119
  ellipsoid parameters 1-39, 1-124, 1-125, 1-141, 1-244

geographic coordinates to matrix coordinates 1-347
great circles to small circles 1-143
hms to matrix elements 1-185
latitude types 1-38, 1-64, 1-72, 1-156, 1-157, 1-158, 1-159, 1-160, 1-161, 1-200, 1-264, 1-304
matrix coordinates to geographic coordinates 1-343
matrix elements to dms 1-222
matrix elements to hms 1-223
time to string 1-376
time units 1-183, 1-184, 1-186, 1-187, 1-341, 1-342, 1-378
coordinate system transforming 1-316
coordinates
converting from geographic to matrix 1-347
converting from matrix to geographic 1-343
converting types 1-63, 1-132, 1-172
selecting with mouse 1-192
Cossin, Jean 2-110
country2mtx 1-86
craster 2-30
Craster Parabolic projection 2-30
Craster, John Evelyn Edmund 2-30
cross fix positions 1-88
crossfix 1-88
current point from map axes 1-148

D

daspectm 1-91
dcwdata 4-12
dcwdem 4-5
DCW-DEM data 4-5
dcwgaz 4-16
dcwread 4-18
dcwrhead 4-21
de l'Isle, Nicolas 2-48
dead reckoning 1-121
Deetz, Charles H. 2-30
defaultm 1-93
deg2dm 1-98
deg2dms 1-98
deg2km 1-99
deg2nm 1-99
deg2rad 1-101
deg2sm 1-99
demcmap 1-102, 3-27
depart 1-104
departure 1-104
Die Rechteckige Plattkarte 2-50
Digital Chart of the World (DCW)
reading digital elevation data 4-5
reading files 4-18
reading gazette 4-16
reading headers 4-21
reading selected data 4-12
digital elevation map, colormap 1-102
displaying
light objects 1-204, 3-34
lighted surfaces 1-366, 3-93
lines 1-207, 1-272, 1-274, 3-38
patches 1-136, 1-138, 1-265, 1-267, 3-29
surfaces 1-234, 1-269, 1-364, 1-370, 3-49, 3-62
text 1-174, 1-373, 3-97
displaym 1-106
dist2str 1-109
distance
converting to string 1-109
converting units 1-99, 1-116, 1-201, 1-257, 1-290, 1-353
See also surface distance
distance 1-111
distdim 1-116

distortcalc 1-114
dms2deg 1-118
dms2dm 1-120
dms2mat 1-119
dms2rad 1-118
dreckon 1-121
dted 4-23

E

Earth *See* almanac 1-22
eastof 1-123
ecc2flat 1-124
ecc2n 1-125
eccentricity 1-39
Eckert I projection 2-32
Eckert II projection 2-34
Eckert III projection 2-36
Eckert IV projection 2-38
Eckert V projection 2-40, 2-134
Eckert VI projection 2-42
Eckert, Max 2-32, 2-34, 2-36, 2-38, 2-40, 2-42
eckert1 2-32
eckert2 2-34
eckert3 2-36
eckert4 2-38
eckert5 2-40
eckert6 2-42
Edwards, Trystan 2-116
egm96geoid 4-25
Egyptians 2-46, 2-98, 2-112
elevation map *See* digital elevation map
ellipse1 1-126
ellipsoid
  approximating planetary geoid *See* almanac
  radius of curvature 1-294
ellipsoid parameters
  conversion 1-124, 1-125, 1-141, 1-244
  converting 1-39
Elliptical projection 2-92
encodem 1-129
epsm 1-131
eqa2grn 1-132
eqaazim 2-74
eqaconic 2-6
eqacylin 2-44
eqdazim 2-46
eqdconic 2-48
eqdcylin 2-50
Equal-Area Cylindrical projection 2-10, 2-12, 2-44, 2-56, 2-78, 2-116
Equidistant Azimuthal projection 2-4, 2-46, 2-47, 2-48
Equidistant Conic projection 2-48
Equidistant Cylindrical projection 2-48, 2-50, 2-51, 2-54, 2-100, 2-134
Equirectangular projection 2-50, 2-51
Erastosthenes 2-100
etopo5 4-27
ETOPO5 model 4-27
Etzlaub, Erhard 2-88
Everett 2-96
external data
  DCW data 4-12, 4-18
  DCW gazette 4-16
  DCW headers 4-21
  DCW-DEM data 4-5
  ETOPO5 model 4-27
  Fifth Fundamental Catalog of Stars 4-35
  TIGER ArcInfo files 4-47
  TIGER FIPS name files 4-29
  TIGER MIF files 4-43
  TIGER/Line data 4-41
  USGS DEM data 4-50, 4-55

USGS DEM filenames 4-57
extractm 1-133

F

Fifth Fundamental Catalog of Stars 4-35
fill3m 1-136, 3-29
fillm 1-138, 3-29
filterm 1-139
findm 1-140
fipsname 4-29
fixing *See* navigational fixing
flat2ecc 1-141
flatplrp 2-82
flatplrq 2-84
flatplrs 2-86
Flat-Polar Quartic projection 2-84
fournier 2-52
Fournier II projection 2-52
Fournier projection 2-52, 2-53
Fournier, Georges 2-52
framem 1-142

G

Gall Isographic projection 2-50, 2-54
Gall Orthographic projection 2-44, 2-56
Gall projection 2-58
Gall Stereographic projection 2-18, 2-58
Gall, James 2-56, 2-58
Gauss-Krüger 2-120
gc2sc 1-143
gcm 1-145
gcpmap 1-148
gcwaypts 1-149
gcxgc 1-152
gcxsc 1-154
general matrix map
projecting 1-269, 1-364, 1-366, 1-370, 3-62, 3-93
projecting shaded relief 1-367
geod2aut 1-156
geod2cen 1-157
geod2cnf 1-158
geod2iso 1-159
geod2par 1-160
geod2rec 1-161
geographic data structure
creating input mlayers 1-315
definition 1-162
displaying 1-106
extracting data 1-133
interacting with objects 3-51
geographic points
mean 1-228
standard deviation 1-360
standard distance 1-358
geoid vector
components 1-165
planets *See* almanac
getm 1-166
getseeds 1-167
giso 2-54
Globe 2-60, 2-61
globe 2-60
Gnomic projection 2-62
gnomonic 2-62
Gnomonic projection 2-62
goode 2-64
Goode Homolosine projection 2-64, 2-92
Goode, J. Paul 2-64
gortho 2-56
graticule mesh 1-230
great circle track

calculating from one point 1-387
calculating from two points 1-390
displaying 3-99
great circles
converting to small circles 1-143
intersection 1-152
intersection with small circles 1-154
Great Soviet World Atlas 2-14
Greeks 2-98, 2-112
grepfields 1-168
gridm 1-171
grn2eqa 1-172
gshhs 4-30
gstereo 2-58
gtextm 1-174
gtopo30 4-32
Guide property editor 3-64

H

hammer 2-66
Hammer projection 2-4, 2-22, 2-66
Hammer-Aitoff projection 2-66
handlem 1-175, 3-31
Hassler, Ferdinand Rudolph 2-102
hatano 2-68
Hatano Asymmetrical Equal-Area projection 2-68
Hatano, Masataka 2-68
hidem 1-178, 3-33
hista 1-179
histogram
equal area 1-179
equirectangular 1-181
histr 1-181
hms2hm 1-183
hms2hr 1-184
hms2mat 1-185
hms2sec 1-184
Homolographic projection 2-92
Homolosine projection 2-64
Hondius, Jodocus 2-110
hr2hm 1-186
hr2hms 1-186
hr2sec 1-187

I

imagem 1-188
imbedm 1-190
inputm 1-192
interpm 1-193
interpolation
latitude and longitude 1-193
latitude given longitude 1-194
longitude given latitude 1-196
intersection
great circles 1-152
great circles and small circles 1-154
object sets 1-88
rhumb lines 1-313
small circles 1-339
intrplat 1-194
intrplon 1-196
ismap 1-198
ismapped 1-199
iso2geod 1-200

J

Jupiter *See* almanac 1-22

K

Kavraisky V projection 2-70
Kavraisky VI projection 2-72
Kavraisky, V. V. 2-70, 2-72
kavrsky5 2-70
kavrsky6 2-72
km2deg 1-201
km2nm 1-201
km2rad 1-201
km2sm 1-201

L

La Carte Parallélogrammatique 2-50
lambcyln 2-78
lambert 2-76
Lambert Azimuthal Equal Area projection 2-66
Lambert Azimuthal Equal-Area projection 2-6, 2-74
Lambert Conformal Conic projection 2-76
Lambert Equal-Area Azimuthal projection 2-20
Lambert Equal-Area Conic projection 2-6
Lambert Equal-Area Cylindrical projection 2-6, 2-44, 2-78
Lambert, Johann Heinrich 2-44, 2-74, 2-76, 2-78
latitude and longitude
  converting latitude types 1-38, 1-64, 1-72, 1-156, 1-157, 1-158, 1-159, 1-160, 1-161, 1-200, 1-264, 1-304
  finding corresponding time zone 1-379
  finding for map entries 1-140
  interpolation 1-193, 1-194, 1-196
  limits of regular matrix map 1-206, 3-36
legs 1-202
light object 3-34
light objects 1-204
lightm 1-204, 3-34
limitm 1-206, 3-36
line objects 1-207, 1-272, 1-274, 3-38
linem 1-207, 3-38
loadmoonalb 4-34
Lorgna projection 2-74
loximuth 2-80
Loximuthal projection 2-80
ltln2val 1-209

M

majaxis 1-210
makemapped 1-211
map
  deleting 1-70
  precision 1-131
map axes
  defining map projection 1-40, 3-7
  modifying properties 1-344
  retrieving map structure 1-145
  retrieving properties 1-166
  setting properties 1-40, 3-7
  testing 1-198
map data
  extracting from data structures 1-133
  filtering 1-139
  querying 3-68
  See also matrix data
  See also vector data
map frame
  displaying 1-142
  modifying properties 1-344
  setting properties 1-40, 1-142, 3-7
map grid
  displaying 1-171
  modifying properties 1-344
  setting properties 1-40, 1-171, 3-7

map grid labels
displaying 1-243, 1-271
modifying properties 1-344
setting properties 1-40, 3-7
map layers 3-51
map legend vector 1-213
map origin *See* origin
map projection
changing 1-344
defining 1-40, 3-7
identification strings 1-214
names 1-214
`maps` 1-214
`maptool` 3-40
`maptrim` 3-46
`maptriml` 1-216
`maptrimp` 1-217
`maptrims` 1-219
Marinus of Tyre 2-50, 2-100
Mars *See* `almanac` 1-22
`maskm` 1-221
`mat2dms` 1-222
`mat2hms` 1-223
matrix data
displaying 1-234, 1-269, 1-364, 1-370, 3-49, 3-62
displaying as lighted 1-366, 3-93
displaying as shaded relief 1-232, 1-367
resizing 1-309
trimming 1-219, 3-46
matrix map
constructing graticule mesh 1-230
encoding regions 1-129
`NaNs` 1-247
ones 1-259
replacing elements 1-65, 1-221
resizing 1-309
sparse zeros 1-357
zeros 1-423
McBryde, F. Webster 2-82, 2-84, 2-86
McBryde-Thomas Flat-Polar Parabolic projection 2-82
McBryde-Thomas Flat-Polar Quartic projection 2-84
McBryde-Thomas Flat-Polar Sinusoidal projection 2-86
`mdistort` 1-224
mean of geographic points 1-228
`meanm` 1-228
`mercator` 2-88
Mercator Equal-Area projection 2-110
Mercator projection 2-26, 2-76, 2-88, 2-90
Mercator, Gerardus 2-48, 2-88
Mercury *See* `almanac` 1-22
meridian labels 1-243
mesh *See* graticule mesh
`meshgrat` 1-230
`meshlsrm` 1-232
`meshm` 1-234, 3-49
`mfwdtran` 1-236
`miller` 2-90
Miller Cylindrical projection 2-90
Miller, Osborn Maitland 2-90
`minaxis` 1-239
`minvtran` 1-240
`mlabel` 1-243
`mlayers` 3-51
`mobjects` 3-55
`modsine` 2-114
`mollweid` 2-92
Mollweide projection 2-64, 2-92
Mollweide, Carl B. 2-92
Moon *See* `almanac` 1-22
mouse interactions

defining small circles 1-337
placing text 1-174
processing button down callbacks 3-103
selecting coordinates 1-192
Murdoch I Conic projection 2-94
Murdoch III Minimum Error Conic projection 2-96
Murdoch, Patrick 2-94, 2-96
murdoch1 2-94
murdoch3 2-96

N

n2ecc 1-244
name 1-245
nanclip 1-246
nanm 1-247
NaNs, matrix map 1-247
National Geographic Society 2-108
navfix 1-248
navigational fixing 1-248
navigational track
calculating segments between waypoints 1-384
navigational track format 1-251
Neptune *See* almanac 1-22
neworig 1-253
newpole 1-256
nm2deg 1-257
nm2km 1-257
nm2rad 1-257
nm2sm 1-257
Nordic projection 2-66
npi2pi 1-258

O

objects
assigning tag 1-372, 3-95
deleting 1-71, 3-19
displaying 1-351, 3-86
editing properties of 3-64
hiding 1-178, 3-33
interacting 3-55
modifying zdata 1-421, 3-104
retrieving handle 1-175, 3-31
retrieving name 1-245
testing if mapped 1-199
onem 1-259
ones matrix map 1-259
Ordinary Polyconic projection 2-102
org2pol 1-260
origin
computing 1-256, 1-282
interactive modification 3-58
transformation 1-253
originui 3-58
ortho 2-98
Orthographic projection 2-60, 2-98, 2-124
Orthophanic projection 2-108

P

panzoom 3-59
paperscale 1-261
par2geod 1-264
parallel labels 1-271
parallelui 3-61
patch objects 1-136, 1-138, 1-265, 1-267
patches
projecting 3-29
patchesm 1-265, 3-29
patchm 1-267, 3-29

pcarree 2-100
pcolorm 1-269, 3-62
Peters projection 2-56
plabel 1-271
planetary data 1-22
Plate Carree projection 2-100
Plate Carrée projection 2-24, 2-40, 2-48, 2-50
plot3m 1-272, 3-38
plotm 1-274, 3-38
Pluto *See* almanac 1-22
polcmap 1-276
pole transformation 1-260
polycon 2-102
Polyconic projection 2-102
polygon surface area 1-31
position
  dead reckoning 1-121
  reckoning 1-305
Postel projection 2-46, 2-47
Postel, Guillaume 2-46
previewmap 1-278
project 1-280
projection
  data 1-236, 1-240
  objects 1-280
Projection of Marinus 2-50, 2-51
property editors 3-64
  Click-and-Drag 3-64
  Guide 3-64
Ptolemy, Claudius 2-16, 2-48
Putnins 2-30
Putnins P4 projection 2-30
Putnins P5 projection 2-104
Putnins, Reinholds V. 2-104
putnins5 2-104
putpole 1-282

Q

qrydata 3-68
quadrangle surface area 1-36
quartic 2-106
Quartic Authalic projection 2-106
querying map data 3-68
quiver3m 1-284, 3-74
quiverm 1-286, 3-76

R

rad2deg 1-288
rad2dm 1-289
rad2dms 1-289
rad2km 1-290
rad2nm 1-290
rad2sm 1-290
radius of auxiliary sphere 1-321
radius of curvature 1-294
radius of planets *See* almanac
Rand McNally 2-108
range
  angles 1-258, 1-422
  finding cross fix position 1-88
raster data *See* matrix data
rc2yx 1-292
rcurve 1-294
readfields 1-296
readFK5 4-35
reading ASCII files 1-356
readmtx 1-300
rec2geod 1-304
reckon 1-305
reckoning 1-305, 3-89
Rectangular projection 2-50, 2-51
reducem 1-307
regular matrix map

calculating required matrix size 1-352
creating colormap 3-21
displaying as image 1-188
encoding 1-190
encoding regions 3-84
latitude and longitude limits 1-206, 3-36
projecting 1-234, 3-49
projecting shaded relief 1-232
retrieving values 1-209
seeds for encoding 1-167
surface area 1-33
transforming to new coordinate system origin 1-253
trimming 1-219
resizem 1-309
restack 1-311
rhumb line track
calculating from one point 1-387
calculating from two points 1-390
displaying 3-99
rhumb lines intersection 1-313
rhxrh 1-313
robinson 2-108
Robinson projection 2-108
Robinson, Arthur H. 2-108
rootlayr 1-315
rotatem 1-316
rotatetext 1-318
rounding 1-320
roundn 1-320
rsphere 1-321
Ruysch, Johannes 2-48

S

Sanson-Flamsteed projection 2-110
satbath 4-37
Saturn *See* almanac 1-22
scaleruler 1-323
scatterm 1-330, 3-78
scircle1 1-332
scircle2 1-335
scircleg 1-337
scirclui 3-80
scxsc 1-339
sec2hm 1-341
sec2hms 1-341
sec2hr 1-342
seedm 3-84
semimajor axis 1-210
semiminor axis 1-239
setltln 1-343
setm 1-344
setpostn 1-347
shaded relief map
constructing cdata 1-348
constructing colormap 1-348
general matrix map 1-367
regular matrix map 1-232
shaderel 1-348
showaxes 1-350
showm 1-351, 3-86
Siemon, Karl 2-80, 2-106
Simple Conic projection 2-48
Simple Cylindrical projection 2-100
sinusoid 2-110
Sinusoidal projection 2-16, 2-40, 2-64, 2-92, 2-110, 2-134
sizem 1-352
sm2deg 1-353
sm2km 1-353
sm2nm 1-353
sm2rad 1-353
small circles

calculating 1-332, 1-335
defining with mouse 1-337
displaying 3-80
intersection 1-339
intersection with great circles 1-154
smoothlong 1-354
spcread 1-356
spzerom 1-357
Stab 2-128
Stabius, Johannes 2-128
Stab-Werner projection 2-128
standard deviation of geographic points 1-360
standard distance of geographic points 1-358
stdist 1-358
stdm 1-360
stem3m 1-362, 3-87
stereo 2-112
Stereographic projection 2-20, 2-76, 2-112
Sun *See* almanac 1-22
surface area
planets *See* almanac
polygon 1-31
quadrangle 1-36
regular matrix map 1-33
surface distance
along a parallel *See* departure
between track waypoints 1-202
between two points 1-111
calculating 3-89
surface objects
constructing graticule mesh 1-230
projecting 1-234, 1-269, 1-364, 1-370, 3-49, 3-62
projecting lighted 1-366, 3-93
surfacem 1-364, 3-62
surfdist 3-89
surflm 1-366, 3-93
surflsrm 1-367
surfm 1-370, 3-62
Sylvanus, Bernardus 2-16

T

tagm 1-372, 3-95
text
mouse placement 1-174
projecting 1-373, 3-97
textm 1-373, 3-97
tgrline 4-41
Thales 2-62
Thomas, Paul D. 2-82, 2-84, 2-86
TIGER data
ArcInfo files 4-47
MIF files 4-43
reading FIPS name files 4-29
TIGER/Line data 4-41
tigermif 4-43
tigerp 4-47
tightmap 1-375
time
converting to matrix elements 1-185
converting to string 1-376
converting units 1-183, 1-184, 1-186, 1-187, 1-341, 1-342, 1-378
time zone
determining from longitude 1-379
time2str 1-376
timedim 1-378
timezone 1-379
tissot 1-381
tissot indicatrices
projecting 1-381
Tissot Modified Sinusoidal projection 2-114
Tissot, N. A. 2-114

Tobler, Waldo R. 2-80
track
  See great circle track
  See navigational track
  See rhumb line track
`track` 1-384
track waypoints
  azimuth 1-202
  distance 1-202
`track1` 1-387
`track2` 1-390
`trackg` 1-392
`trackui` 3-99
transformation of coordinate system 1-316
Transverse Mercator projection 2-120
`trimcart` 1-393
`trystan` 2-116
Trystan Edwards Cylindrical projection 2-44, 2-116
Tunhuang star chart 2-88

U

U.S. Army 2-120
`uimaptbx` 3-103
units
  converting *See* conversion
  testing for valid abbreviations 1-394
  testing for valid strings 1-394
`unitstr` 1-394
Universal Polar Stereographic projection 2-118
Universal Transverse Mercator projection 2-118, 2-120
ups 2-118
UPS projection 2-118
Uranus *See* `almanac` 1-22
`usahi` 1-395
`usalo` 1-396
`usamap` 1-398
USGS DEM data
  reading files 4-50, 4-55
  returning filenames 4-57
`usgs24kdem` 4-50
`usgsdem` 4-50, 4-55
`usgsdems` 4-57
`utm` 2-120
UTM projection 2-118, 2-120
`utmgeoid` 1-404
`utmzone` 1-402

V

Van der Grinten I projection 2-122
Van der Grinten projection 2-122
Van der Grinten, Alphons J. 2-122
`vec2mtx` 1-405
vector data
  converting to matrix format 3-46
  displaying as lines 1-207, 1-272, 1-274, 3-38
  displaying as patches 1-136, 1-138, 1-265, 1-267, 3-29
  reducing 1-307
  trimming 1-216, 1-217
Venus *See* `almanac` 1-22
Vertical Perspective Azimuthal projection 2-124
`vfwdtran` 1-407
`vgrint1` 2-122
`vinvtran` 1-410
`vmap0data` 4-58
`vmap0read` 4-62
`vmap0rhead` 4-66
volume of planets *See* `almanac`
von Hammer, H. H. Ernst 2-66
`vperspec` 2-124

W

Wagner I projection 2-72
Wagner IV projection 2-126
Wagner, Karlheinz 2-72, 2-126
`wagner4` 2-126
waypoints
  calculating 1-149
  See also track waypoints
`werner` 2-128
Werner projection 2-16, 2-128
Werner, Johannes 2-128
`westof` 1-413
`wetch` 2-130
Wetch Cylindrical projection 2-130
Wetch projection 2-26
Wetch, J. 2-130
`wiechel` 2-132
Wiechel projection 2-132
Wiechel, H. 2-132
`winkel` 2-134
Winkel I projection 2-134
Winkel, Oswald 2-134
`worldlo` 1-414
`worldmap` 1-416
Wright projection 2-88
Wright, Edward 2-88

Y

Young, A. E. 2-20
`yx2rc` 1-419

Z

`zdatam` 1-421, 3-104
Zenithal Equal-Area projection 2-74
Zenithal Equidistant projection 2-46
Zenithal Equivalent projection 2-74
Zenithal projection 2-46, 2-47
`zero22pi` 1-422
`zerom` 1-423
zeros
  creating matrix map 1-423
  creating sparse matrix map 1-357
zooming in and out of map displays 3-59