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.
3 Apple Hill Drive
Natick, MA 01760-2098 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 1984 - 2000 by The MathWorks, Inc.

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.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by or for the federal government of the United States. By accepting delivery of the Program, the government hereby agrees that this software qualifies as "commercial" computer software within the meaning of FAR Part 12.212, DFARS Part 227.7202-1, DFARS Part 227.7202-3, DFARS Part 252.227-7013, and DFARS Part 252.227-7014. The terms and conditions of The MathWorks, Inc. Software License Agreement shall pertain to the government's use and disclosure of the Program and Documentation, and shall supersede any conflicting contractual terms or conditions. If this license fails to meet the government's minimum needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to MathWorks.

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

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

Printing History: May 1997 First printing New for Version 1
October 1998 Second printing Revised for Version 1.1
September 2000 Third printing Revised for Version 1.2 (Release 12)

Contents

Preface

Acknowledgments xxii

Related Products List xxiii

1 Getting Started

An Introduction to the Mapping Toolbox 1-2

Creating Base Maps 1-3

Displaying Data on a Base Map 1-10

Importing High-Resolution Atlas Data 1-13

Geographic Calculations 1-15

Getting More Help 1-18

2 Mapping Fundamentals

What Is a Map? 2-2

Types of Maps in the Mapping Toolbox 2-3
Vector Maps 2-3
Matrix Maps 2-4
Composite Maps 2-5

Geographic Measurement **2-7**
Latitude and Longitude **2-7**
Great Circles and Rhumb Lines **2-9**
Measuring Distance **2-10**
Measuring Azimuth and Elevation **2-11**
Reckoning Positions **2-13**
Small Circles **2-14**
Measuring Area **2-16**
The Geoid Model **2-17**
Measuring the Planets **2-20**

Angles, Times, and Distances – Units and Notation **2-22**
Angular Notation **2-22**
Time Notation **2-25**
Distance Notation **2-26**

Working with Vector Maps **2-27**
Data Format **2-27**
Segments versus Polygons **2-28**
Extracting Polygon or Line Segments **2-29**
Merging Line Segments **2-30**
Creating Vector Data **2-31**
Geographic Interpolation **2-33**
Polygon Area **2-36**
Vector Calculations – Intersections **2-37**
Polygon Boolean Operations **2-40**
Cutting Polygons at the Dateline **2-41**
Buffer Zones **2-42**
Trimming Vector Data to a Defined Region **2-43**
Simplifying Vector Data **2-45**

Working with Matrix Maps **2-48**
Regular Matrix Maps **2-48**
General Matrix Maps **2-61**

3 Displaying Maps

Introduction to Mapping Graphics 3-2

Map Displays Made Easy 3-3

Map Axes 3-5

Accessing and Manipulating Map Axes Properties 3-8

The Map Frame 3-13

The Map Grid 3-17

Map and Frame Limits 3-21

Switching Between Projections 3-22

Projected and Unprojected Objects 3-25

Displaying Vector Maps as Lines 3-28

Displaying Vector Maps as Patches 3-31

Displaying Matrix Maps 3-34
- The Graticule 3-35
- Data Representation 3-37
- Mapped Light Objects 3-41
- Draping Data on Elevation Maps 3-44

The Geographic Data Structure 3-49

Interacting with Displayed Maps 3-53
- Interactive Tracks and Circles 3-53
- Interactive Text 3-55
- Working with Objects by Name 3-55

Specialized Map Functions **3-58**
Inset Maps **3-58**
Graphic Scale **3-59**
Choropleth Maps **3-60**
Thematic Map Functions **3-62**
Cartesian MATLAB Display Functions **3-65**
Colormap and Colorbars **3-68**
Printing Maps **3-75**

4 Atlas Data

Types of Data **4-2**

World Vector Data **4-3**
Coastlines **4-3**
Low-Resolution World Atlas Data **4-5**
High-Resolution World Atlas Data **4-15**

World Matrix Data **4-21**
Political **4-21**
Terrain **4-24**
Geoid **4-25**

United States Vector Data **4-27**
Low Resolution Data **4-27**
Medium Resolution State Outlines **4-32**

United States Matrix Data **4-36**
Political **4-36**
Terrain **4-37**

Astronomical Data **4-40**

5 Map Projections

What Is a Projection? 5-2

Quantitative Properties 5-3

Geometric Surfaces 5-4

Projection Aspect 5-7
The Origin Vector 5-7
Coordinate Transformations 5-15

Projection Computations 5-19

Working with the Globe Projection 5-23

Working with the UTM projection 5-26

Summary Guide 5-31

6 Mapping Applications

Introduction to Geographic Statistics and Navigation 6-2

Geographic Statistics 6-3
Geographic Means 6-3
Geographic Standard Deviation 6-4
Equal-Areas in Geographic Statistics 6-7
Geographically Filtering Datasets 6-10

Navigation 6-11
Conventions for Navigational Functions 6-11
Fixing Position 6-12
Planning 6-24
Track Laydown – Displaying Navigational Tracks 6-26

Dead Reckoning 6-28
Drift Correction 6-33
Time Zones 6-35
Time Notation 6-37

7 GUI Tools

An Overview 7-2

The Complete GUI Environment: maptool 7-3
Starting maptool 7-3
The Menus 7-4
The Mouse Tool Buttons 7-5
Example One: Creating a World Map 7-5
Example Two: Creating a Regional Map 7-15

Constructing Personalized Map Data with GUIs 7-25
Trimming an Existing Data Set 7-25
Encoding a Regular Surface Map 7-27
Creating a Personalized Colormap 7-34

8 External Data Interface

Working with External Data 8-2

Global Vector Data 8-3
The Digital Chart of the World/VMAP0 8-3
Global Self-Consistent Hierarchical High-Resolution Shoreline 8-20
USGS On-Line Coastline Extractor 8-23

U.S. Vector Data **8-25**
TIGER/Line **8-25**
TIGER Thinned Boundary **8-29**

Global Matrix Data **8-34**
ETOPO5 and TerrainBase **8-34**
Global Bathymetry Predicted from Satellite Altimetry **8-38**
GTOPO30 and GLOBE **8-42**
Digital Terrain Elevation Data Model **8-48**
Reading Elevation Data Interactively **8-51**
Advanced Very High Resolution Radiometer (AVHRR) Global Change Data **8-55**
EGM96 Geoid Model **8-67**
U. S. Gridded Elevation Data **8-68**

Astronomical Data **8-74**

Importing Other Data **8-80**

Reference

9 Mapping Reference

Function Tables **9-2**
Map Definition and Display **9-2**
Map Appearance and Interaction **9-6**
Geographic Computation **9-9**
Data Operations **9-12**
Matrix Map Data Operations **9-14**
Reference Data **9-16**
Geoid Manipulation **9-18**
Data Conversion **9-20**
Mapping Toolbox Demos **9-23**

Alphabetical List of Functions **9-24**
almanac **9-25**
angl2str **9-29**
angledim **9-31**
antipode **9-33**
areaint **9-34**
areamat **9-36**
areaquad **9-39**
aut2geod **9-41**
axes2ecc **9-42**
axesm **9-43**
axesscale **9-56**
azimuth **9-58**
bufferm **9-61**
camposm **9-63**
camtargm **9-65**
camupm **9-67**
cart2grn **9-69**
cen2geod **9-70**
changem **9-71**
clabelm **9-72**
clegendm **9-74**
clma **9-76**
clmo **9-77**
cnf2geod **9-78**
coast **9-79**
combntns **9-80**
comet3m **9-81**
cometm **9-82**
contor3m **9-83**
contorm **9-85**
contourcmap **9-87**
contourfm **9-90**
country2mtx **9-95**
crossfix **9-98**
daspectm **9-101**
defaultm **9-103**
deg2dms, deg2dm **9-107**
deg2km, deg2nm, deg2sm **9-108**
deg2rad **9-110**

demcmap **9-111**
departure **9-113**
displaym **9-115**
dist2str **9-118**
distance **9-120**
distortcalc **9-123**
distdim **9-125**
dms2deg, dms2rad **9-127**
dms2mat **9-128**
dms2dm **9-129**
dreckon **9-130**
driftcorr **9-132**
driftvel **9-134**
eastof **9-135**
ecc2flat **9-136**
ecc2n **9-137**
elevation **9-138**
ellipse1 . . . . . **9-140**
encodem **9-144**
epsm **9-146**
eqa2grn **9-147**
extractm **9-148**
fill3m **9-150**
fillm **9-152**
filterm **9-153**
findm **9-154**
flat2ecc **9-155**
flatearthpoly **9-156**
framem **9-158**
gc2sc **9-159**
gcm **9-161**
gcpmap **9-163**
gcwaypts **9-164**
gcxgc **9-166**
gcxsc **9-168**
geod2aut **9-170**
geod2cen **9-171**
geod2cnf **9-172**
geod2iso **9-173**
geod2par **9-174**

geod2rec **9-175**
geographic data structure **9-176**
geoid vector **9-179**
getm **9-180**
getseeds **9-181**
gradientm **9-182**
grepfields **9-187**
gridm **9-190**
grn2eqa **9-191**
gtextm **9-192**
handlem **9-193**
hidem **9-196**
hista **9-197**
histr **9-199**
hms2hm **9-201**
hms2hr, hms2sec **9-202**
hms2mat **9-203**
hr2hms, hr2hm **9-204**
hr2sec **9-205**
imagem **9-206**
imbedm **9-208**
inputm **9-209**
interpm **9-210**
intrplat **9-211**
intrplon **9-213**
ismap **9-215**
ismapped **9-216**
iso2geod **9-217**
km2deg, km2nm, km2rad, km2sm **9-218**
legs **9-219**
lightm **9-221**
limitm **9-223**
linem **9-224**
los2 . . . . . **9-226**
ltln2val **9-230**
majaxis **9-231**
makemapped **9-232**
map legend vector **9-234**
mapprofile **9-235**
maps **9-241**

maptriml **9-243**
maptrimp **9-244**
maptrims **9-246**
maskm **9-248**
mat2dms **9-249**
mat2hms **9-250**
mdistort **9-251**
meanm **9-255**
meshgrat **9-257**
meshlsrm **9-259**
meshm **9-261**
mfwdtran **9-263**
minaxis **9-265**
minvtran **9-266**
mlabel **9-269**
n2ecc **9-270**
namem **9-271**
nanclip **9-272**
nanm **9-273**
navfix **9-274**
navigational track format **9-278**
neworig **9-280**
newpole **9-283**
nm2deg, nm2km, nm2rad, nm2sm **9-284**
npi2pi **9-285**
onem **9-286**
org2pol **9-287**
paperscale **9-288**
par2geod **9-291**
patchesm **9-292**
patchm **9-294**
pcolorm **9-296**
plabel **9-298**
plot3m **9-299**
plotm **9-301**
polcmap **9-303**
polybool **9-305**
polyjoin **9-307**
polymerge **9-308**
polysplit **9-310**

polyxpoly **9-311**
previewmap **9-313**
project **9-315**
putpole **9-317**
quiver3m **9-319**
quiverm **9-321**
rad2deg **9-323**
rad2dms, rad2dm **9-324**
rad2km, rad2nm, rad2sm **9-325**
rc2yx **9-327**
rcurve **9-329**
readfields **9-331**
readmtx **9-335**
rec2geod **9-339**
reckon **9-340**
reducem **9-342**
resizem **9-344**
restack **9-346**
rhxrh **9-348**
rootlayr **9-350**
rotatem **9-351**
rotatetext **9-353**
roundn **9-355**
rsphere **9-356**
scaleruler **9-358**
scatterm **9-365**
scircle1 . . . . . **9-367**
scircle2 . . . . . **9-370**
scircleg **9-372**
scxsc **9-374**
sec2hms, sec2hm **9-376**
sec2hr **9-377**
setltln **9-378**
setm **9-379**
setpostn **9-382**
shaderel **9-383**
showaxes **9-385**
showm **9-386**
sizem **9-387**
sm2deg, sm2km, sm2nm, sm2rad **9-388**

smoothlong **9-389**
spcread **9-391**
spzerom **9-392**
stdist **9-393**
stdm **9-395**
stem3m **9-397**
str2angle **9-399**
surfacem **9-401**
surflm **9-403**
surflsrm **9-405**
surfm **9-408**
tagm **9-410**
textm **9-411**
tightmap **9-413**
time2str **9-414**
timedim **9-416**
timezone **9-417**
tissot **9-419**
track **9-422**
track1 . . . . . **9-425**
track2 . . . . . **9-428**
trackg **9-430**
trimcart **9-432**
unitstr **9-433**
usahi **9-434**
usalo **9-435**
usamap **9-437**
utmzone **9-441**
utmgeoid **9-443**
vec2mtx **9-444**
vfwdtran **9-446**
viewshed **9-449**
vinvtran **9-454**
westof **9-457**
worldhi **9-458**
worldlo **9-462**
worldmap **9-464**
yx2rc **9-469**
zdatam **9-471**
zero22pi **9-472**

zerom 9-473

10 Projections Reference

Alphabetical List of Map Projections 10-2
Aitoff Projection 10-3
Albers Equal-Area Conic Projection 10-5
Apianus II Projection 10-7
Balthasart Cylindrical Projection 10-9
Behrmann Cylindrical Projection 10-11
Bolshoi Sovietskii Atlas Mira Projection 10-13
Bonne Projection 10-15
Braun Perspective Cylindrical Projection 10-17
Breusing Harmonic Mean Projection 10-19
Briesemeister Projection 10-21
Cassini Cylindrical Projection 10-23
Central Cylindrical Projection 10-25
Collignon Projection 10-27
Craster Parabolic Projection 10-29
Eckert I Projection 10-31
Eckert ll Projection 10-33
Eckert lll Projection 10-35
Eckert IV Projection 10-37
Eckert V Projection 10-39
Eckert VI Projection 10-41
Equal-Area Cylindrical Projection 10-43
Equidistant Azimuthal Projection 10-45
Equidistant Conic Projection 10-47
Equidistant Cylindrical Projection 10-49
Fournier Projection 10-51
Gall Isographic Projection 10-53
Gall Orthographic Projection 10-55
Gall Stereographic Projection 10-57
Globe 10-59
Gnomonic Projection 10-61
Goode Homolosine Projection 10-63

Hammer Projection **10-65**
Hatano Asymmetrical Equal-Area Projection **10-67**
Kavraisky V Projection **10-69**
Kavraisky VI Projection **10-71**
Lambert Azimuthal Equal-Area Projection **10-73**
Lambert Conformal Conic Projection **10-75**
Lambert Equal-Area Cylindrical Projection **10-77**
Loximuthal Projection **10-79**
McBryde-Thomas Flat-Polar Parabolic Projection **10-81**
McBryde-Thomas Flat-Polar Quartic Projection **10-83**
McBryde-Thomas Flat-Polar Sinusoidal Projection **10-85**
Mercator Projection **10-87**
Miller Cylindrical Projection **10-89**
Mollweide Projection **10-91**
Murdoch I Conic Projection **10-93**
Murdoch III Minimum Error Conic Projection **10-95**
Orthographic Projection **10-97**
Plate Carrée Projection **10-99**
Polyconic Projection **10-101**
Putnins P5 Projection **10-103**
Quartic Authalic Projection **10-105**
Robinson Projection **10-107**
Sinusoidal Projection **10-109**
Stereographic Projection **10-111**
Tissot Modified Sinusoidal Projection **10-113**
Trystan Edwards Cylindrical Projection **10-115**
Universal Polar Stereographic Projection **10-117**
Universal Transverse Mercator Projection **10-119**
Van der Grinten I Projection **10-121**
Vertical Perspective Azimuthal Projection **10-123**
Wagner IV Projection **10-125**
Werner Projection **10-127**
Wetch Cylindrical Projection **10-129**
Wiechel Projection **10-131**
Winkel I Projection **10-133**

11 GUI Reference

Graphical User Interface Tables **11-2**
Map Definition Tools **11-2**
Mapping Tools **11-2**
Object Projection Tools **11-3**
Display Manipulation Tools **11-3**
Thematic Map Tools **11-4**
Object Property Tools **11-4**
Track Tools **11-5**
Map Data Construction Tools **11-5**

Alphabetical List of GUI Functions and Tools **11-6**
axesm, axesmui **11-7**
clmo **11-19**
clrmenu **11-20**
colorm **11-21**
cometm, comet3m **11-23**
contorm, contor3m **11-25**
demcmap **11-27**
fillm, fill3m, patchm, patchesm **11-29**
handlem **11-31**
hidem **11-33**
lightm **11-34**
limitm **11-36**
linem, plotm, plot3m **11-38**
maptool **11-40**
maptrim **11-46**
meshm **11-49**
mlayers **11-51**
mobjects **11-55**
originui **11-58**
panzoom **11-59**
parallelui **11-61**
pcolorm, surfacem, surfm **11-62**
property editors **11-64**
qrydata **11-68**
quiver3m **11-74**
quiverm **11-77**

scatterm 11-79
scirclui 11-81
seedm 11-85
showm 11-87
stem3m 11-88
surfdist 11-90
surflm 11-94
tagm 11-96
textm 11-98
trackui 11-100
uimaptbx 11-104
zdatam 11-105

12 External Data Reference

External Data Function Tables 12-2
External Data Interface 12-2

Alphabetical listing of External Interface Functions 12-4
avhrrgoode 12-5
avhrrlambert 12-9
dcwdata 12-12
dcwgaz 12-16
dcwread 12-18
dcwrhead 12-21
demdataui 12-23
dted 12-27
dteds 12-30
egm96geoid 12-32
etopo5 12-34
fipsname 12-36
globedem 12-37
globedems 12-40
gshhs 12-41
gtopo30 12-43
loadmoonalb 12-46

readfk5 . . . . . **12-47**
satbath **12-50**
tbase **12-52**
tgrline **12-54**
tigermif **12-56**
tigerp **12-60**
usgs24kdem **12-63**
usgsdem **12-67**
usgsdems **12-69**
vmap0data **12-70**
vmap0read **12-74**
vmap0rhead **12-77**
vmap0ui **12-79**
Bibliography **12-83**

A Geographic Terms

Glossary Notes **A-2**

Glossary **A-3**

B Bibliography

Dedication

In memory of John Snyder, whose prior work on Map Projections inspired and made possible the creation of the Mapping Toolbox.

Preface

Acknowledgments . . . . . **xxii**

Related Products List . . . . . **xxiii**

Acknowledgments

Systems Planning and Analysis, Inc. (SPA) is proud to have developed the Mapping Toolbox. The Mapping Toolbox is the product of the contributions of numerous SPA employees. Ed Byrns and Eric Brown, who are no longer at SPA, conceived the Toolbox, designed the architecture, and supported the development to completion of Version 1.0. Walter Stumpf managed the completion and continued development of the product and also developed the external interface and the atlas data.

Tomi Debole, Lenox Job and Andy Kim were essential to the development, testing, and documentation of the Mapping Toolbox. Tom Seaman wrote many of the testing functions. Juliet Crumrine edited the documentation. Wendell Nix helped with the rhumb line calculations and the geometry of track intersections.

The product was developed with long-term support from The MathWorks and many of its key employees, most notably Loren Shure, Jim Tung, Matthew Simoneau, Scott Gray, Jane Carmody, Peg Theriault, Kathy Ford, Jennifer French, and Clay Thompson.

Related Products List

The MathWorks provides several products that are especially relevant to the kinds of tasks you can perform with the Mapping Toolbox.In particular, the Mapping Toolbox *requires* these products:

- MATLAB®

For more information about any of these products, see either:

- The online documentation for that product if it is installed or if you are reading the documentation from the CD
- The MathWorks Web site, at `http://www.mathworks.com`; see the "products" section

Note The products listed below complement the functionality of the Mapping toolbox products.

Product	Description
Database Toolbox	Tool for connecting to, and interacting with, most ODBC/JDBC databases from within MATLAB
Image Processing Toolbox	Complete suite of digital image processing and analysis tools for MATLAB
Statistics Toolbox	Tool for analyzing historical data, modeling systems, developing statistical algorithms, and learning and teaching statistics

1

Getting Started

An Introduction to the Mapping Toolbox 1-2

Creating Base Maps 1-3

Displaying Data on a Base Map 1-10

Importing High-Resolution Atlas Data 1-13

Geographic Calculations 1-15

Getting More Help 1-18

An Introduction to the Mapping Toolbox

Welcome to the Mapping Toolbox for MATLAB®. The Mapping Toolbox allows you to read, analyze, and display geographic information within MATLAB. Since the earth and most heavenly bodies are generally spherical in shape, geographic data is usually defined on a spherical or elliptical coordinate system. Distance, azimuth, area, and even straight lines on the curved surface of the earth are defined differently than in MATLAB's cartesian coordinates.

Displaying geographic data from round bodies on flat surfaces also requires special techniques from the art of cartography. The Mapping Toolbox can create maps using simple commands that make reasonable cartographic choices or allow you to control every detail of the map display. Atlas data provided with the toolbox allows you to create detailed base maps on which you can plot your own results. You can also import high-resolution map data available over the Internet from government or academic sources.

This chapter will acquaint you to with capabilities of the toolbox. As you go through the following examples, you can consult the online help, which includes reference material, or later sections of this document for more details. To make full use of the Mapping Toolbox, you will need to be familiar with the geographic and cartographic concepts described in the later chapters of this *Mapping Toolbox User's Guide*.

Creating Base Maps

Geographic data is usually displayed on a base map containing features like coastlines or topography. The first step in the preparation of a base map is the selection of an appropriate projection. Projections are ways of representing features from a round body on a flat piece of paper or computer screen. The art of displaying a three-dimensional body on a two-dimensional surface is called cartography. Cartographers have devised many different projections, each a different compromise between fidelity of scale, shape, and direction. The art in cartography is partly the selection of a projection that is appropriate to the task at hand. The next step in the preparation of a base map is the selection and display of appropriately detailed atlas data. It is important that the detail and amount of data be matched to the coverage of the base map.

Although the Mapping Toolbox provides full control over a large number of projections and several kinds of atlas data, it can also create base maps that are good for most applications. The command to create a base map for the earth is `worldmap`. The `worldmap` command chooses a projection and atlas data detail appropriate to the area you specify. You can define the area by the name of a continent or a country, or by latitude and longitude limits. If you are unsure of the spelling of a name, use `worldmap` with no input arguments to pick the name from a list.

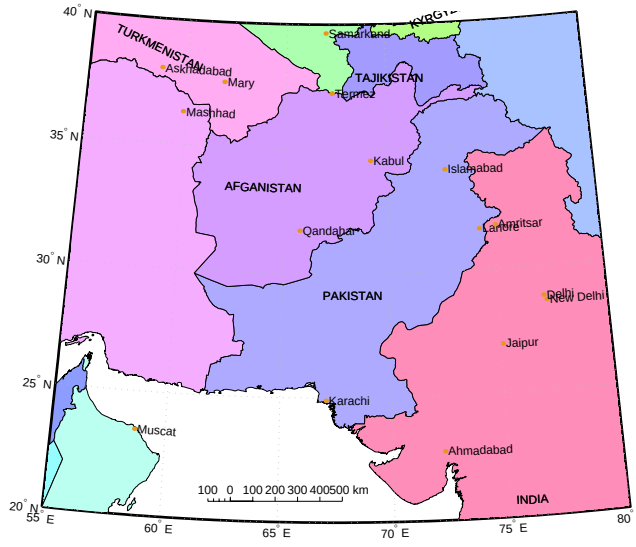
```
worldmap 'south america'
hidem(gca)
```

The `hidem(gca)` command hides the MATLAB axes box for printing. It is usually best to hide the axes just before you print.

In this case, `worldmap` selected low-resolution atlas data because of the large area covered by the continent. The atlas data includes coastlines, international boundaries, lakes, and cities. For maps of large areas, such as South America, `worldmap` chooses the lower resolution data and omits details like country names and cities. You can override `worldmap`'s choices using optional arguments.

As you specify smaller regions, `worldmap` switches to higher resolution atlas data. Here is a base map of Pakistan using patches (filled polygons) instead of lines. This display also adds a common cartographic element, a graphic scale. You can use your mouse to drag the scale ruler to a better position. See the help for `scaleruler` and `setm` to change other properties.

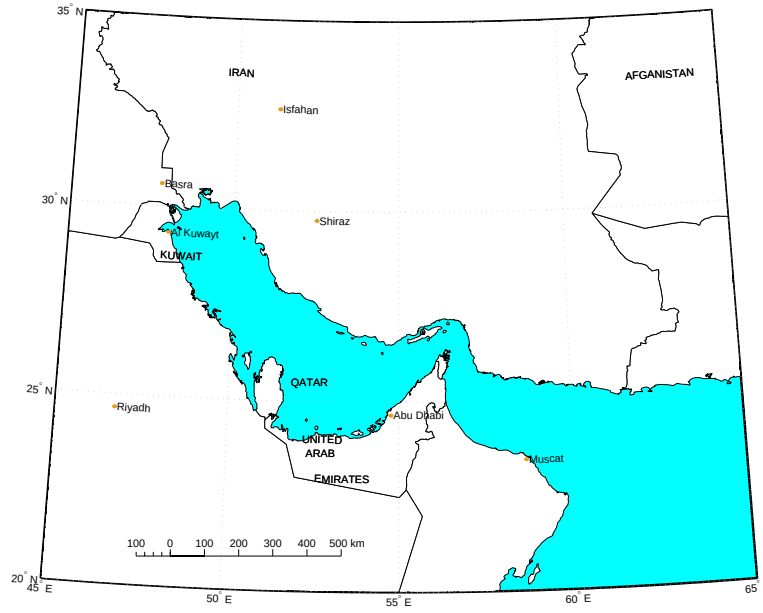
```
figure
worldmap('pakistan','patch')
scaleruler
hidem(gca)
```



To change the alignment or adjust the positions of the city names, use the click and drag tools described in Chapter 7, “GUI Tools”.

You can also specify the region of interest by latitude and longitude limits. The following example constructs a base map of the Persian Gulf. This example also shows how to use MATLAB Handle Graphics® and Mapping Toolbox commands to change graphic properties like color of the countries and the underlying map frame. For more information on map and Handle Graphics properties, see Chapter 2, “Mapping Fundamentals” in this document or in the online Mapping Toolbox reference documentation; you can also find guidance in the MATLAB graphics documentation.

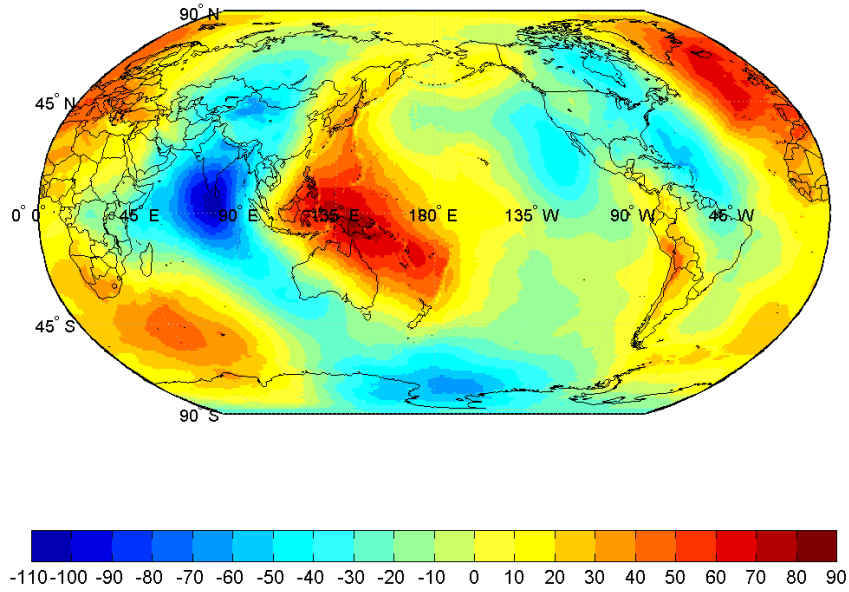
```
figure
worldmap([20 35],[45 65],'patch')
scaleruler
set(handlem('allpatch'),'Facecolor','w')
setm(gca,'FFaceColor','c')
hidem(gca)
```



You can also construct a base map using matrix data. The Mapping Toolbox defines two kinds of matrix data. Regular matrix maps are on an equal-angle grid, with rows and columns aligned to the east-west and north-south. General matrix maps can have arbitrary spacing and alignment of matrix elements.

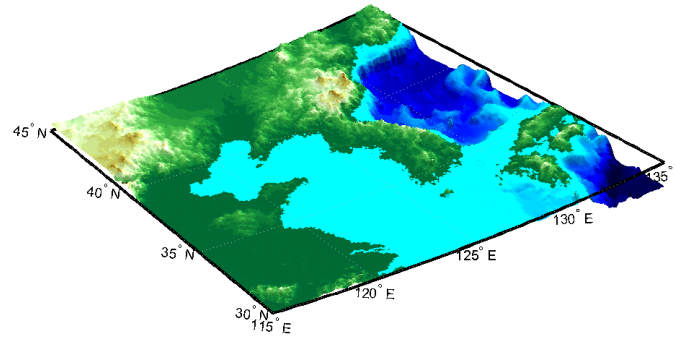
Here is an example of a regular matrix map for the whole world. The `geoid` workspace file contains a matrix of geoid heights on a one-degree grid. The geoid can be thought of as the surface of the ocean without the influence of waves, tides, or land. When you provide `worldmap` a regular matrix map, the geographic limits are taken from the extent of the matrix. Here the geoid height (in meters) is shown as a colored surface. The `contourcmap` command creates a colormap with the color steps aligned to contour levels, making values easier to read from the display. Chapter 3, “Displaying Maps” will show you how to add matrix displays to an existing base map.

```
load geoid
figure
worldmap(geoid,geoidlegend)
contourcmap(10,'jet','colorbar','on','location','horizontal')
```



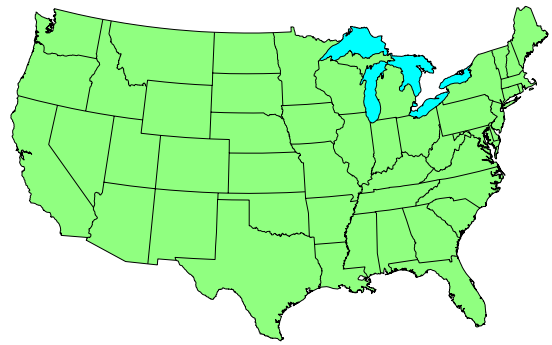
Other colors are typically used to display elevation matrices. Data of this type is often referred to as a Digital Elevation Map, or DEM. Here is an example that shows elevation and bathymetry for the Korean peninsula as a three-dimensional lighted surface. There is a very large amount of vertical exaggeration to bring out the topographic features. Use the `daspectm` command to control the vertical scaling.

```
load korea
figure
worldmap(map,maplegend,'ldem3d')
view(3)
```



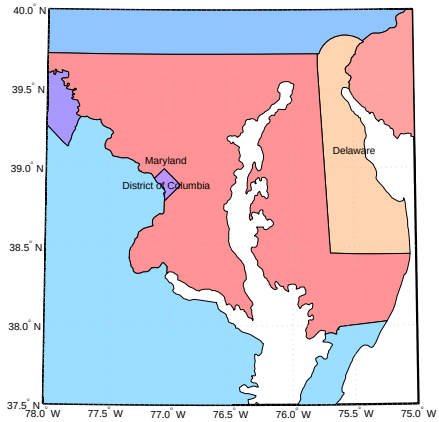
For base maps of the United States of America, use the similar `usamap` command; to display just the conterminous states, use `usamap conus`.

```
figure
usamap conus
```



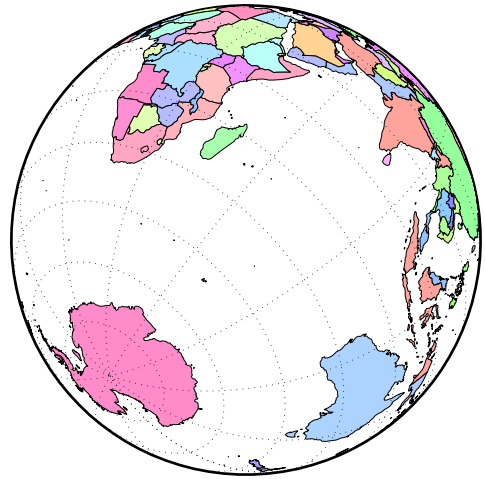
To display smaller regions with higher resolution atlas data, use the state name or latitude and longitude limits.

```
figure
usamap([37.5 40],[-78 -75])
hidem(gca)
```



When you need more control over the projection and base map data, use the lower level `axesm` command instead of `worldmap` or `usamap`. The `axesm` command is used to prepare a map axes and set the initial projection parameters. You can then add the atlas data of your choice. Here is a display using the orthographic projection with low-resolution filled country outlines. The orthographic projection looks like the globe viewed from an infinite distance. Use the `maps` command to see a full list of the available projections.

```
figure
axesm('MapProjection','ortho','Origin',[-35 70 45])
framem;gridm
displaym(worldlo('POpatch'))
polcmap
hidem(gca)
```



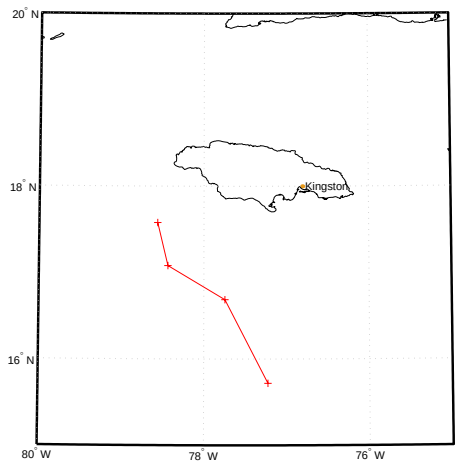
Displaying Data on a Base Map

The Mapping Toolbox can display geographic data contained in MATLAB data types such as vectors, matrices, and structures. The command names are usually similar to the MATLAB graphics commands, but with an “m” at the end. The “m” indicates that the data is geographic, and should be displayed using the already defined map projection. For example, in MATLAB you would display a vector of points as a line using `plot(x,y)`, whereas for geographic data the command would be `plotm(lat,long)`.

Since most map displays contain several kinds of information, new plot commands add to, rather than replace graphic elements on a map axes. Also, the scaling of x and y coordinates is set equal to make the map projections look right. Use the `daspectm` command to control the scaling of the z axis.

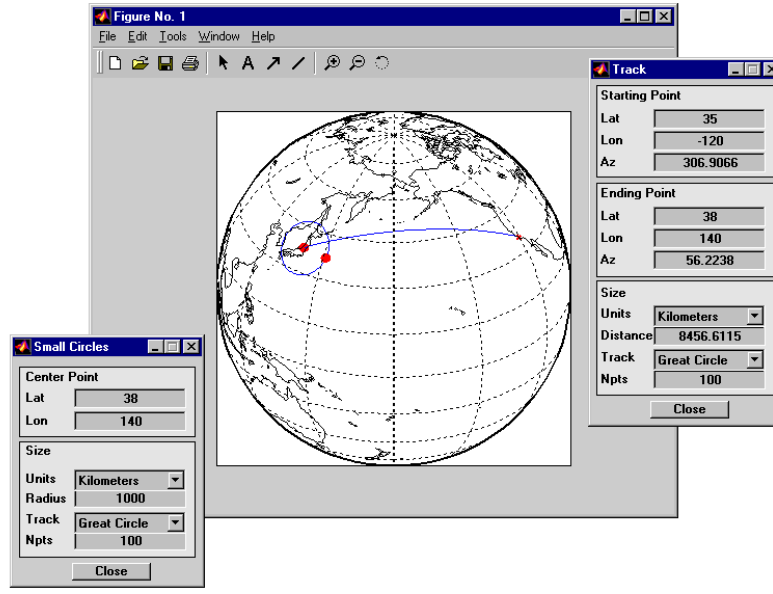
The following example shows how you can find the geographic locations of points by clicking on a map, and then display the result with a mapping version of the plot command. Of course, your points will be different. To display the data as lines without symbols, or symbols without lines, change the line style specification.

```
worldmap jamaica
[lat,lon] = inputm(4);
plotm(lat,lon,'-+r')
hidem(gca)
```



Important geographic line data such as great circle tracks and small circles can also be calculated or created interactively. Great circle tracks are the shortest distance between points, and small circles are the locus of points a constant distance from a center point. You can use `trackg` and `scircleg` to create them by clicking on the map. If you extend-click on the tracks or circles, you can move the control points to modify the lines, or type specific numbers into a control panel. The control panels also allow the tracks and circles to be used for measurements on the map. Just adjust the lines or circles to the right place on the map, and read the dimensions off the control panel. Command-line functions accomplish similar tasks without map interaction.

Here are a great circle track from the United States to Japan and a 1000 km radius small circle centered on Tokyo. On a globe, the line would be straight, and the circle would be circular. The map projection introduces some characteristic distortion.

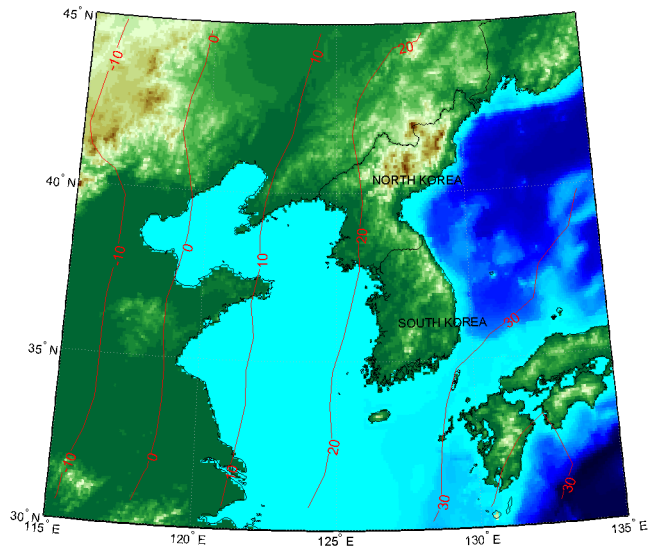



```
axesm('ortho','origin',[30 180])
framem;gridm
plotm(coast,'k')
trackg
scircleg
```

A common way to display matrix data is as contour lines. The Mapping Toolbox provides a number of ways to represent contours. Contour lines are created by `contourm`, while filled contours drawn as patches are created by `contourfm`. The same effect as `contourfm` with equally spaced contours can be achieved by displaying the matrix as a surface and controlling the colormap with `contourcmap`. The `worldmap` geoid display is an example of this approach.

Here is the same geoid data displayed on top of a digital elevation map of the Korean peninsula. The `contourm` and `clabelm` commands display and label contours of geoid heights in steps of 10 meters.

```
load korea
worldmap(map,maplegend,'dem')
load geoid
[c,h] = contourm(geoid,geoidlegend,-100:10:100,'r');
ht = clabelm(c,h); set(ht,'color','r')
hidem(gca)
```

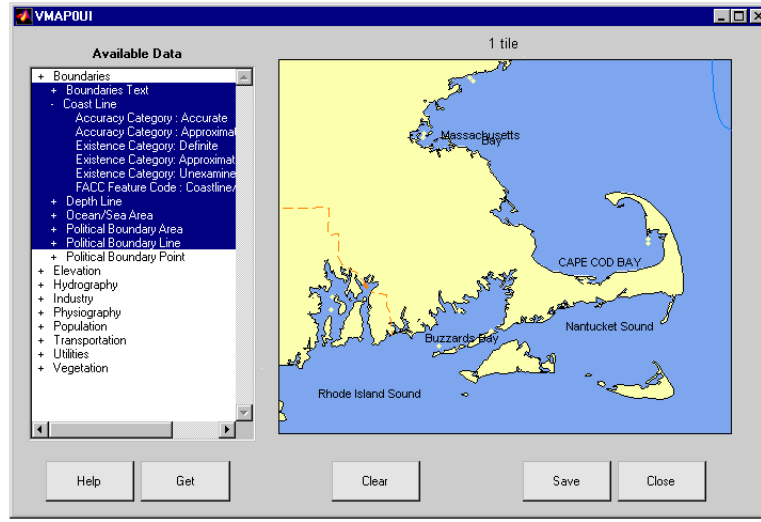


Importing High-Resolution Atlas Data

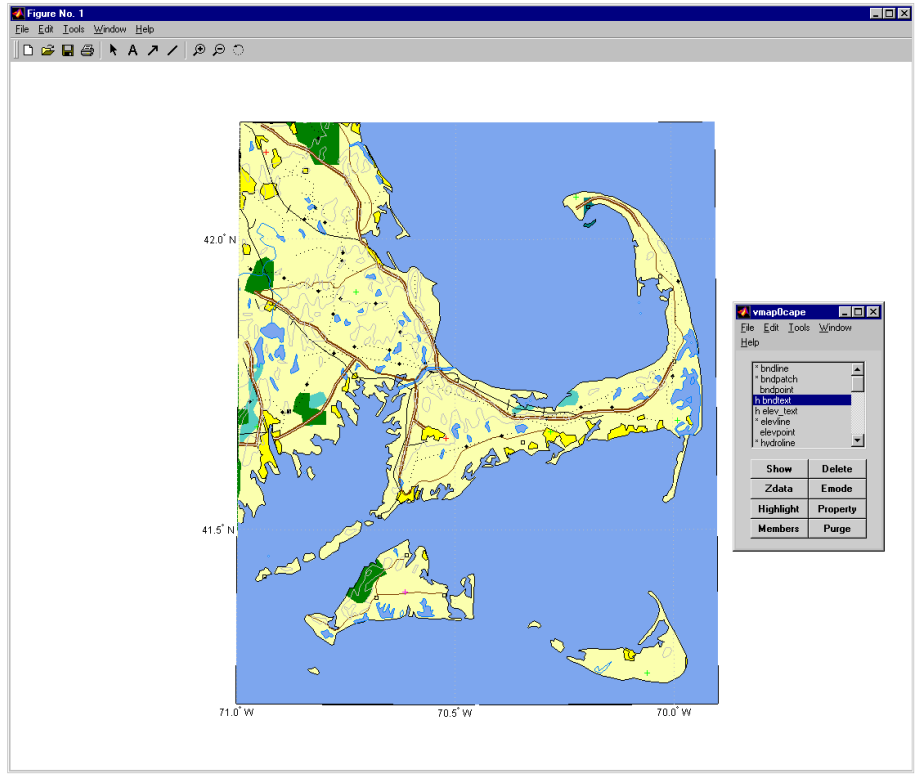
In addition to the atlas data provided with the Mapping Toolbox, you can import a number of high-resolution datasets that are available over the Internet or on CD-ROMs. Important examples of matrix data accessible though external interface functions include vegetation and digital elevation models. These datasets cover the world at horizontal resolutions ranging from 10 kilometers to 100 meters. Supported vector data includes a detailed vector shoreline, the Digital Chart of the World, and its successor VMAP0. Data can be imported using command-line functions and graphical user interfaces.

The highest resolution global vector data available to the public is the Vector Map Level 0, or VMAP0. It comes on four inexpensive CD-ROMs that can be purchased from the U.S. Geological Survey. The `vmap0ui` graphical user interface makes it easy to select and extract data from the CDs. The following example shows how to start up the `vmap0ui` panel. You can then zoom in on the map to the area of interest and select layer from the list. Clicking the **Get** button then extracts the selected layers for that region. Here most of the boundary data has been extracted.

```
vmap0ui
```



After extracting all of the desired data, you can save the result to a MAT-file, or return it to the workspace for display in another map axes. The data is stored in Geographic Data Structures, which contain enough information to allow the data to be displayed automatically. In the next figure, all of the geographic data for Cape Cod was extracted and saved in the MAT-file `vmap0cape`. The `worldmap` command creates a blank base map, and the `mlayers` panel is used to interactively display the data layer by layer. The equivalent actions at the command line would be to load the `vmap0cape` data and use the `displaym` command.




```
figure
worldmap([41.2 42.2],[-71 -69.9],'none')
hidem(gca)
set(gcf,'color','w')
mlayers vmap0cape
```

Geographic Calculations

Beyond its geographic display capabilities, the Mapping Toolbox also includes an extensive set of commands to analyze geographic data. There are unit conversions and a database of the dimensions of the earth and other heavenly bodies. Other commands include geographical geometry such as distance, azimuth, elevation, reckoning, and intersections. Geographical data such as track lines, circles, ellipses, and buffer zones can also be constructed.

```
nm2km(1)

ans =

        1.852

sm2km(1)

ans =

       1.6093

dst = distance(45,115,30,135)

dst =

       21.721

deg2km(dst)

ans =

       2415.3

azimuth(45,115,30,135)

ans =

        126.84
```

Regular matrix maps can also be used in a variety of computations. Matrix map capabilities include inserting or extracting values by geographic location, converting vector to matrix data, coding enclosed regions, and computing the slope, aspect, gradient, and visibility of terrain. For example, here are the water depth and geoid height at a point in the Yellow Sea.

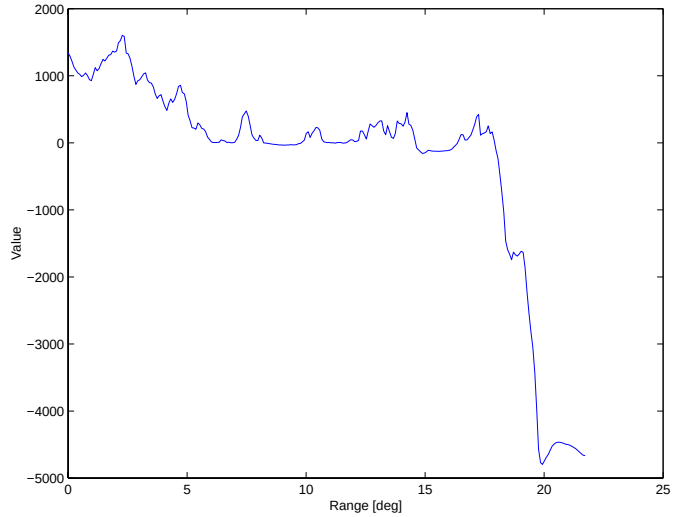
```
load korea
ltln2val(map,maplegend,35,125)

ans =

   -64

load geoid
ltln2val(geoid,geoidlegend,35,125)

ans =

                    19.7490
```

In which country is this location?

```
load worldmtxmed
code = ltln2val(map,maplegend,36,127)

code =

   104

names{code}

ans =

Korea, Republic of
```

Here is the elevation profile for the Korea elevation data you displayed earlier. The figure shows the elevations and depths along a path from the upper left to the lower right corner of the map.

```
load korea
mapprofile(map,maplegend,[45;30],[115;135]);
```


2000
1000
0
−1000
−2000
−3000
−4000
−5000
Value
0
5
10
15
20
25
Range [deg]

Getting More Help

You can learn more about the Mapping Toolbox from the remainder of this User's Guide. The Mapping Toolbox reference material is available in electronic form as PDF and HTML documents through the `helpdesk` command. You may want to print out the reference chapters to browse through them. This is best done from the PDF version. For an overview of the commands, type `help map` for computational functions, `help mapdisp` for display functions, and `help mapproj` for projections. For online help on a specific function, type `help functionname`. For the corresponding reference page, including examples, type `doc functionname`.

2

Mapping Fundamentals

What Is a Map? 2-2

Types of Maps in the Mapping Toolbox 2-3

Geographic Measurement 2-7

Angles, Times, and Distances – Units and Notation . . 2-22

Working with Vector Maps 2-27

Working with Matrix Maps 2-48

What Is a Map?

The simplest definition of a map is *a representation of geographic data*. To the ancient Egyptians, this representation first took the form of lists of city names in the order they would be encountered when following a given road. Two-dimensional renditions, such as paper road maps, are more familiar to us today. Even classroom globes are maps under this definition.

This toolbox addresses electronic representations of geographic data. It allows the creation, use, and presentation of geographic data in a variety of forms and to a variety of ends.

In the Mapping Toolbox, a *map* is any variable or set of variables representing or assigning values to a geographic location or region, from a single point to an entire planet.

Types of Maps in the Mapping Toolbox

Vector Maps

A series of latitude-longitude coordinate pairs representing, for example, points along the coast of Greenland, or along Interstate 80, or even the two sets together, form a map. In this case, the geographic data is in *vector* format and is referred to as a *vector map*. This format consists of specific points, along with some indication as to how they should or should not be connected to each other. In the Mapping Toolbox, vector data consists of sequentially ordered pairs of latitude and longitude coordinates. The pairs are considered to be connected in sequence; breaks in connectivity must be delineated by the creation of separate variables or by inserting `NaN`s into the sets at the appropriate break points. For vector map data, the connectivity of the data is often only a concern during display.

To illustrate a vector map, enter the following:

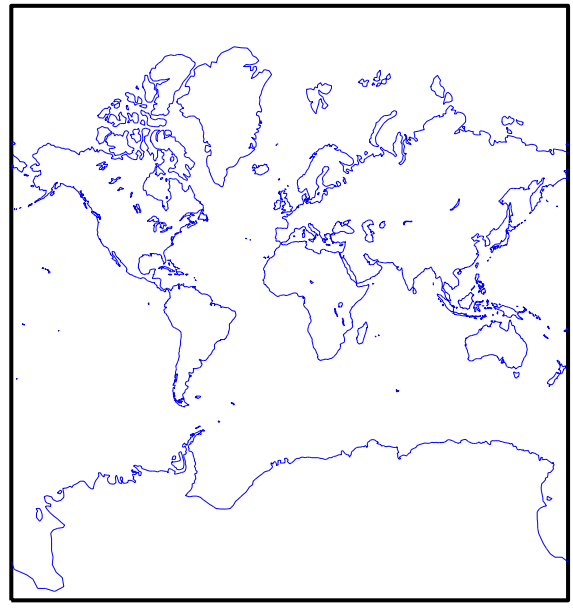
```
load coast
whos
```

```
   Name       Size       Bytes     Class

   lat         9589x1    76712     double array
   long        9589x1    76712     double array
```

```
axesm mercator
framem
plotm(lat,long)
```

The variables `lat` and `long` are vectors in the `coast` MAT-file, which together form a vector map of the coastlines of the world.

We have chosen to view the map in a Mercator projection. A map projection displays the spherical map data in a two-dimensional plane. There are many possible ways to project a map.



Matrix Maps

A matrix in which each element represents a value corresponding to a specific geographic area also forms a map. In this case, the geographic data is in *matrix* format, which is often called *raster* format. However, the Mapping Toolbox makes great use of the powerful matrix manipulation capabilities of MATLAB to fully exploit this map type, so the term *matrix map* is more appropriate.

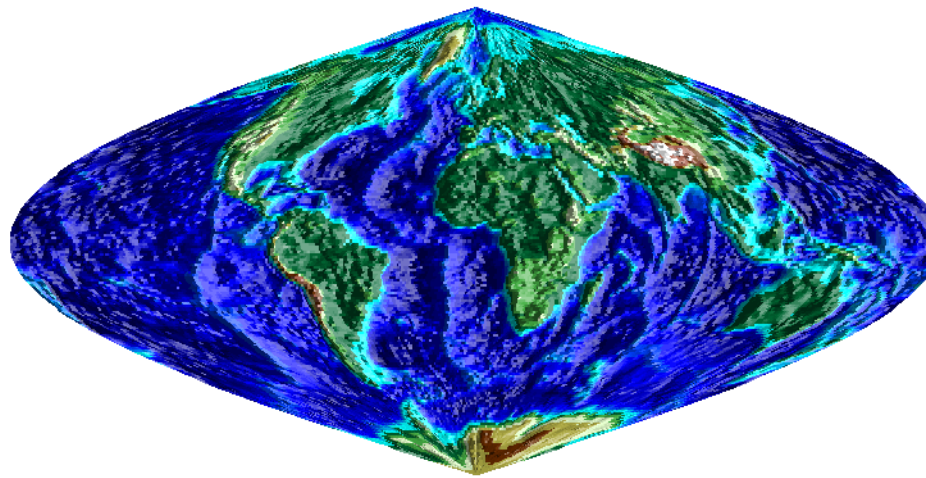
When the data consists of surface elevations, the map can also be referred to as a *digital elevation map* (DEM) or *topographical map*.

As an example of this map type, consider a 180-by-360 matrix. Each row represents one degree of latitude, and each column represents one degree of longitude. Each entry of this matrix is the average elevation, in meters, for the one-degree-by-one-degree region of the Earth to which its row and column correspond. This is, in fact, the `topo` matrix that is provided in the MATLAB product in the workspace of the same name.

To display the matrix map, type the following:

```
clma % clears map
load topo
axesm sinusoid
meshlsrm(topo,topolegend)
```

The `topo` matrix is displayed as a lighted, shaded relief map in a Sinusoidal projection:



Note that the meaning of the map and the projection in which it is displayed are completely independent. The `topo` variable is a map whether it is displayed or not, and it is the same map no matter how it is displayed.

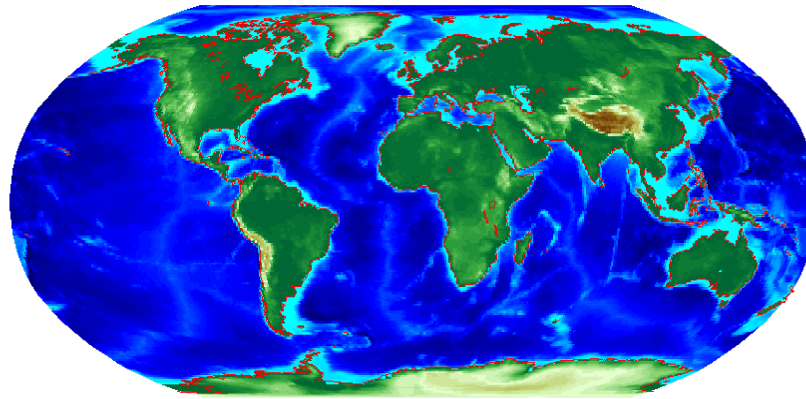
Composite Maps

Many occasions arise in which vector map variables and matrix map variables are used or displayed together. For example, continental coastlines in vector form might be displayed with matrix temperature data to make the latter more useful. When several map variables are used together, regardless of type, they can be correctly referred to as a single map.

Since we have our `coast` and `topo` workspaces available from the previous examples, we can combine the two in a single map and see how well the two types of data correspond:

```
figure; axesm robinson
meshm(topo,topolegend); demcmap(topo)
plotm(lat,long,'r')
```

Here is the resulting map:



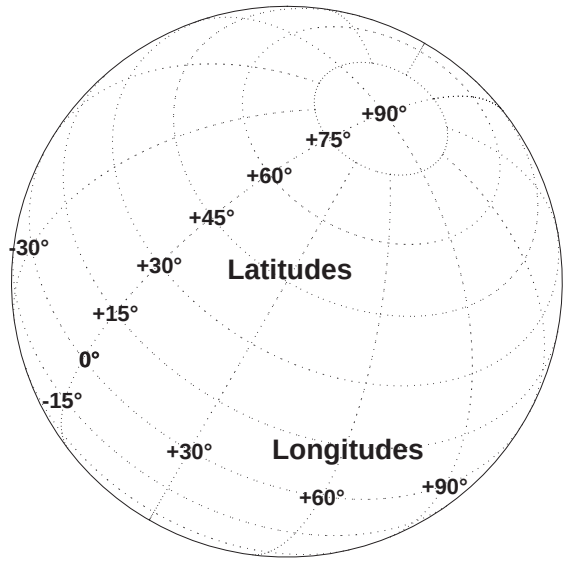
The remainder of this chapter will focus on the fundamental principles of geographic measurement and data manipulation that are a prerequisite for creating map displays. Chapter 3, “Displaying Maps”, takes up the topic of map display and interaction.

Geographic Measurement

Latitude and Longitude

The position of a point on the surface of the Earth, or any other planet, for that matter, can be specified with two angles, *latitude* and *longitude*. These angles can be specified in degrees or radians; however, degrees are far more common in geographic usage.

Latitude is the angle at the center of the planet between the plane of the Equator and a line through the center passing through the surface at the point in question. Latitude is positive in the Northern Hemisphere, reaching a limit of +90 at the North Pole, and negative in the Southern Hemisphere, reaching a limit of -90 at the South Pole. Lines of constant latitude are called*parallels*.



Longitude is the angle at the center of the planet between two planes passing through the center and perpendicular to the plane of the Equator. The first plane also includes the surface point in question; the second plane contains the Prime Meridian, which currently means it includes the point defined by the Royal Observatory in Greenwich, England. Lines of constant longitude are called *meridians*.

Longitudes ranging from -180 to +180 are unique. Longitudes may be given outside this range, but such values can always be renamed to a value within the range. The Mapping Toolbox provides a number of functions to convert data between different latitude and longitude ranges. For example, the command `npi2pi` wraps data to this domain:

```
longitudes = [-560 125 190];
newlongitudes = npi2pi(longitudes)
newlongitudes =
   160.0000  125.0000 -170.0000
```

Sometimes it is more useful to consider longitude as strictly positive, proceeding from the Prime Meridian (0) eastward around and back to the Prime Meridian (360). Any longitude data can be converted to this domain using the `zero22pi` command:

```
positivelongs = zero22pi(newlongitudes)
positivelongs =
   160.0000  125.0000  190.0000
```

If you need this data in radians, you can use an angle conversion function:

```
radianlongs = deg2rad(positivelongs)
radianlongs =
   2.7925    2.1817    3.3161
```

Several angle conversion functions are available in this toolbox, supporting degrees, radians, and degrees-minutes-seconds notation.

What is the antipodal point (on the opposite side of the Earth) of Natick, Massachusetts (about 42.3 N, 71.35 W)?

```
[antilat,antilong] = antipode(42.3,-71.35)
antilat =
   -42.3000

antilong =
   108.6500
```

The result (42.3 S,108.65 E) lies in the Indian Ocean southwest of Australia.

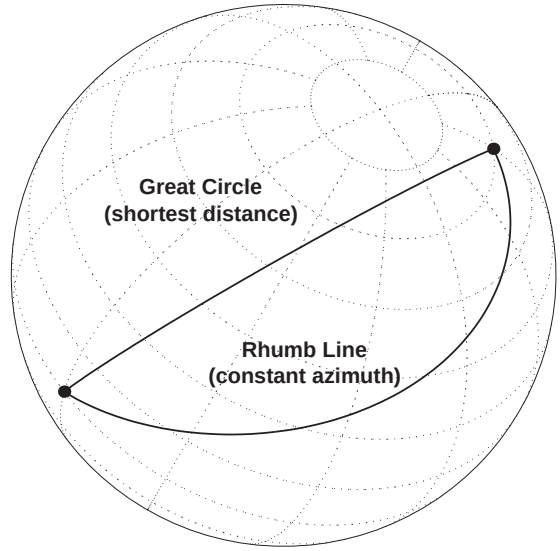
Great Circles and Rhumb Lines

In plane geometry, lines have two important characteristics. A line represents the shortest path between two points, and the slope of such a line is constant. When describing lines on the surface of a spheroid, however, only one of these characteristics may be guaranteed at a time.

In geography, a *great circle* is the shortest path between two points along the surface of a sphere. The precise definition of a great circle is the intersection of the surface with a plane passing through the center of the planet. In general, great circles do not have a constant azimuth, the spherical analog of slope; they cross successive meridians at different angles. When the Earth is taken as a sphere, the Equator and all meridians are great circles. Great circles always bisect the sphere.

A *rhumb line* is a complex curve, the special property of which is that it crosses each meridian at the same angle. This curve is also referred to as a *loxodrome*. In general, a rhumb line is not the shortest path between two points on the rhumb line. All parallels, including the Equator, are rhumb lines, since they cross all meridians at 90 .Additionally, all meridians are rhumb lines. Rhumb lines always terminate at the poles, unless the azimuth is true east or west, in which case the rhumb line closes on itself to form a parallel of latitude.

A description of how to calculate points along great circles and rhumb lines appears later in this document.



Measuring Distance

When you calculate the distance between two points with the Mapping Toolbox, the result will depend upon whether you want a great circle or a rhumb line distance. The `distance` command will return the appropriate distance between two points as an angular arc length, employing the same angular units as the input latitudes and longitudes. The default path type is the shorter great circle, and the default angular units are degrees. The previous figure shows two points at (15 S, 0) and (60 N, 150 E). The great circle distance between them, in degrees of arc, is as follows:

```
distgc = distance(-15,0,60,150)
distgc =
   129.9712
```

The rhumb line distance is greater:

```
distrh = distance('rh',-15,0,60,150)
distrh =
```

```
    145.0288
```

To determine how much longer the rhumb line path is in, say, kilometers, you can use a distance conversion function on the difference:

```
kmdifference = deg2km(distrh-distgc)
kmdifference =
    1.6744e+03
```

Several distance conversion functions are available in the toolbox, supporting degrees, radians, kilometers, meters, statute miles, nautical miles, and feet. Converting distances between angular arc length units and surface length units requires the radius of a planet or spheroid. By default, the radius of the Earth is used.

Measuring Azimuth and Elevation

Azimuth is the angle a line makes with a meridian, taken clockwise from north. When the azimuth is calculated from one point to another using the Mapping Toolbox, the result will depend upon whether a great circle or a rhumb line azimuth is desired. For great circles, the result will be the azimuth at the starting point of the connecting great circle path. In general, the azimuth along a great circle is not constant. For rhumb lines, the resulting azimuth is constant along the entire path.

Azimuths, or bearings, are returned in the same angular units as the input latitudes and longitudes. The default path type is the shorter great circle, and the default angular units are degrees. In our example, the great circle azimuth from the first point to the second is:

```
azgc = azimuth(-15,0,60,150)
azgc =
    19.0391
```

For the rhumb line, the constant azimuth is:

```
azrh = azimuth('rh',-15,0,60,150)
azrh =
    58.8595
```

One feature of rhumb lines is that the inverse azimuth, from the second point to the first, is the complement of the forward azimuth and can be calculated by simply adding 180 to the forward value:

```
inverserh = azimuth('rh',60,150,-15,0)
inverserh =
   238.8595

difference = inverserh-azrh
difference =
   180
```

This is not true, in general, of great circles:

```
inversegc = azimuth('gc',60,150,-15,0)
inversegc =
   320.9353

difference = inversegc-azgc
difference =
   301.8962
```

The azimuths associated with cardinal and intercardinal compass directions are the following:

North	0 or 360
Northeast	45
East	90
Southeast	135
South	180
Southwest	225
West	270
Northwest	315

Elevation is the angle above the local horizontal of one point relative to the other. To compute the elevation angle of a second point as viewed from the first, provide the position and altitude of the points. The default units are degrees for latitudes and longitudes and meters for altitudes, but you can specify other units for each. What is the elevation and slant range of a point 10 kilometers east and 10 kilometers above a surface point?

```
[elevang,slantrange] = elevation(0,0,0, 0,km2deg(10),10000)
```

```
elevang =

       44.901


slantrange =

        14156
```

The answer is slightly different from that expected from plane geometry because of the curvature of the Earth.

Reckoning Positions

A common problem in geographic applications is the determination of a destination given a starting point, an initial azimuth, and a distance. In the Mapping Toolbox, this process is called *reckoning*. A new position may be reckoned in a great circle or a rhumb line sense (great circle or rhumb line track).

As an example, an airplane takes off from La Guardia Airport in New York (40.75 N, 73.9 W) and follows a northwestern rhumb line flight path at 200 knots (nautical miles per hour). Where would it be after 1 hour?

```
[rhlat,rhlong] = reckon('rh',40.75,-73.9,nm2deg(200),315)
rhlat =
   43.1054
rhlong =
   -77.0665
```

Notice that the distance, 200 nautical miles, must be converted to degrees of arc length with the `nm2deg` conversion function to match the latitude and longitude inputs. If the airplane had a flight computer that allowed it to follow an exact great circle path, what would the aircraft's new location be?

```
[gclat,gclong] = reckon('gc',40.75,-73.9,nm2deg(200),315)
gclat =
   43.0615
gclong =
   -77.1238
```

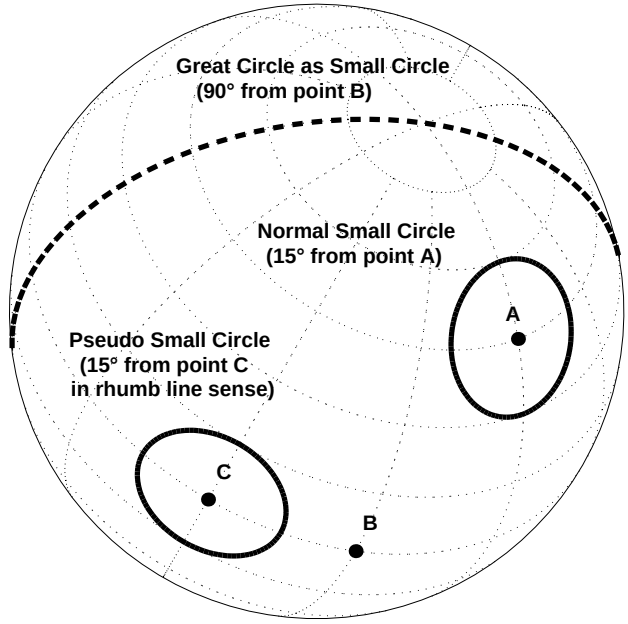
Notice also that for short distances at these latitudes, the result hardly differs between great circle and rhumb line. The two destination points are less than 4 nautical miles apart. Incidentally, after 1 hour, the airplane would be just north of New York's Finger Lakes.

Small Circles

In addition to rhumb lines and great circles, one other geographic construction is significant in geography and the Mapping Toolbox: the *small circle*. The precise definition of a small circle is the intersection of a plane with the surface of the planet. In the Mapping Toolbox, this definition includes planes passing through the center of the planet, so the set of all small circles includes all great circles as limiting cases. This usage is not universal.

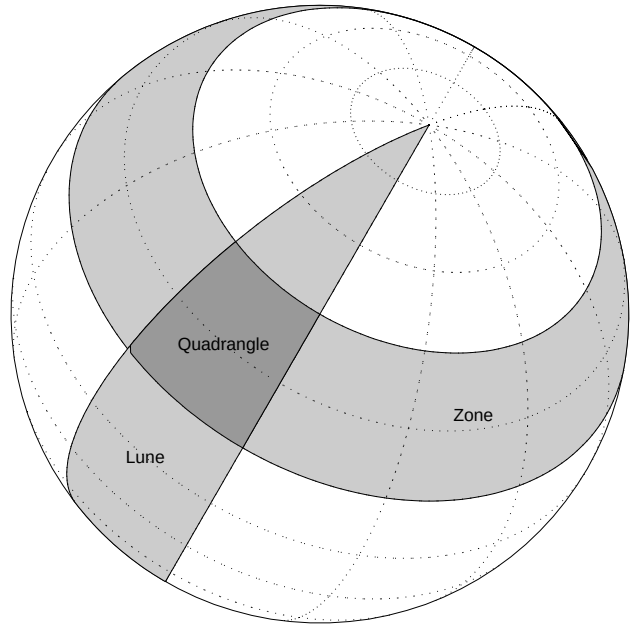
Small circles are most easily defined by distance. *All points 45 nm (nautical miles) distant from (45 N,60 E)* would be the description of one small circle. If degrees of arc length are used as a distance measurement, then a great circle is the set of all points 90 distant from a particular *center* point.

For true small circles, the distance must be defined in a great circle sense, the shortest distance between two points on the surface of a sphere. However, the Mapping Toolbox also allows the calculation of a *pseudo small circle*, for which distances are measured in a rhumb line sense (along lines of constant azimuth). These should not be confused with true small circles.


Great Circle as Small Circle
(90° from point B)
Normal Small Circle
(15° from point A)
A
Pseudo Small Circle
(15° from point C
in rhumb line sense)
C
B

Measuring Area

In solid geometry, the area of a spherical quadrangle can be exactly calculated. A spherical quadrangle is the intersection of a *lune* and a *zone*. In geographic terms, a *quadrangle* is defined as a region bounded by parallels north and south, and meridians east and west.



In the pictured example, a quadrangle is formed by the intersection of a zone, which is the region bounded by 15 N and 45 N latitudes, and a lune, which is the region bounded by 0 and 30 E longitude. Under the spherical planet assumption, the fraction of the entire spherical surface area inscribed in the quadrangle can be calculated:

```
area = areaquad(15,0,45,30)
area =
   0.0187
```

That is, less than 2% of the planet's surface area is in this quadrangle. To get an absolute figure in, for example, square miles, you must provide the appropriate spherical radius. The radius of the Earth is about 3958.9 miles.

```
area = areaquad(15,0,45,30,3958.9)
area =
       3.6788e+06
```

The surface area within this quadrangle is over 3.6 million square miles for a spherical Earth.

The Geoid Model

Although the Earth is round, it is not exactly a sphere. However, for many users and most applications, the difference is so small that a spherical assumption is sufficient. This section addresses how the Mapping Toolbox handles more accurate models of the shape, or figure, of the Earth and other planets.

What Is a Geoid?

Literally, *geoid* means *Earth-shaped*. The geoid is the precise figure of the Earth. Specifically, it is an equipotential surface with respect to gravity. It is approximately an oblate ellipsoid, but not exactly so because of local variations in gravity.

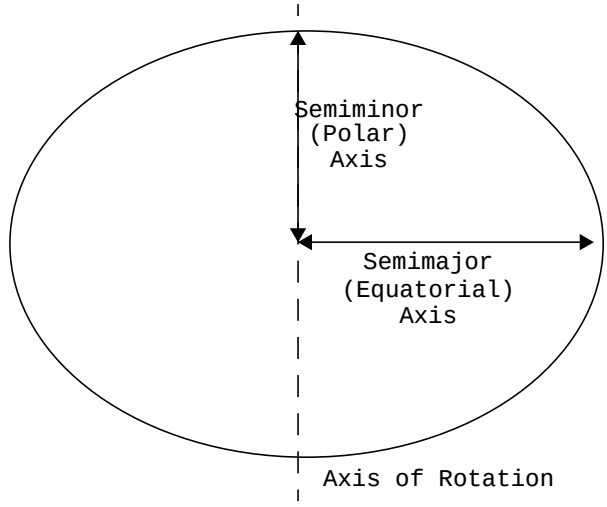
For the purposes of map display, the geoid of the Earth is assumed to be an exact ellipsoid. This is a simplification of the precise definition of geoid; however, it is a good approximation, which allows a great variety of calculations to be performed that would otherwise be prohibitively complicated. Ellipsoidal models of the Earth's figure are considered sufficient for cartography. The use of the term geoid for this ellipsoid is perhaps peculiar to the toolbox. Further, in the Mapping Toolbox, the term geoid is extended to ellipsoidal models for the figures of the Sun, Moon, and planets.

For purposes other than map display, a more detailed model of the figure of the Earth is used. For more information on detailed geoid models, see Chapter 4, “Atlas Data” and Chapter 8, “External Data Interface.”

The Geoid Vector

Since the toolbox treats the geoid as an ellipsoid, the geoid can be described with a two-element vector, which is often called the *geoid vector* in this guide. The geoid vector has the form [*semimajor-axis eccentricity*]. The semimajor axis can be in any unit of distance; the choice of these units often drives the units used for distance outputs in the toolbox functions.

An *ellipsoid* is defined as a rotation of an ellipse around its minor axis. Ellipses can be defined with a semimajor and a semiminor axis, *flattening*, the parameter n, or the preferred method of a semimajor axis and eccentricity. The toolbox has functions to convert elliptical definitions from any of these forms to the form of the geoid vector. For example, the command `axes2ecc` returns an eccentricity when given the semimajor and semiminor axes.



The geoid vector, along with several other kinds of planetary data for each of the planets and the Earth's moon, can be queried using the `almanac` function provided by the Mapping Toolbox (see "Measuring the Planets" on page 2-20).

For the Earth, the semimajor axis is about 21 kilometers longer than the semiminor axis. Use the `almanac` function to verify this:

```
geoid = almanac('Earth','geoid','kilometers')
geoid =
      6378.14      0.08

semiminor = minaxis(geoid)
semiminor =
      6356.75

geoid(1) – semiminor
ans = 21.38
```

When compared to the semimajor axis, which is almost 6400 kilometers, this difference seems insignificant and can be neglected for most purposes. For this reason, most functions in the Mapping Toolbox default to a spherical model of the Earth. The ellipse in the previous diagram is very exaggerated. The difference between the semimajor and semiminor axes of a correct ellipse at this scale would be far smaller than the width of the line used to draw it.

What Is the "Correct" Geoid Vector?

A variety of ellipsoid models have been proposed through the years. They differ because of the surveying information upon which they are based, or because certain regions of interest of the true geoid are best modeled by a particular ellipsoid that may be less accurate for other parts of the world. The Mapping Toolbox default geoid vector (after the sphere) is based on the 1980 Geodetic Reference System ellipsoid. This geoid vector is returned by the statement `almanac('Earth','geoid')`. It is also the reference ellipsoid for the 1984 World Geodetic System equipotential model.

Several other geoid vectors are supported, for models ranging from Everest's 1830 ellipsoid (used for India) to the International Astronomical Union ellipsoid of 1965 (used for Australia). These can be referenced by the following statements:

```
geoid1 = almanac('Earth','geoid',[],'everest');
geoid2 = almanac('Earth','geoid',[],'iau65');
```

The reference page for the `almanac` function has more information on which geoids are available.

You should probably use a spherical Earth unless you have a good reason to use an ellipsoidal model. If you use an ellipsoidal model, you should probably use the default unless you need a specific one. If you cannot find the geoid vector you need, you can create it in the following form:

```
geoidvec = [semimajor-axis eccentricity]
```

Note that the default units for the geoid semimajor axis in the `almanac` function are kilometers, which can be used by simply passing in an empty matrix in place of the input units string, as in the previous example. Eccentricity is dimensionless.

Measuring the Planets

The Mapping Toolbox contains a function that provides almanac data on the major bodies of our solar system. Geophysical data, such as geoid vectors, radii, surface areas, and volumes, can be accessed for the Sun, the Earth's moon, and all of the planets, in any of the supported units of distance measurement.

Many planets have ellipsoidal geoid vectors available. Some planets return spherical geoid vectors only:

```
almanac('Earth','geoid','nauticalmiles')
ans =
   3443.92          0.08

almanac('mars','geoid','kilometers')
ans =
   3396.90          0.11

almanac('moon','geoid','statutemiles')
ans =
   1079.97             0
```

When a radius is desired, a scalar is returned representing the radius of the best spherical model of the planet. Notice that for a spherical model, the radius in radians is 1:

```
almanac('mercury','radius','kilometers')
ans =
   2439

almanac('neptune','radius','radians')
ans =
   1
```

Surface areas and volumes are calculated based on a spherical model by default. In most cases, you can use the geoid model instead, and for the Earth, you can specify any of the supported geoid models. You can also request the actual tabulated values of the Earth:

```
almanac('mars','surfarea','kilometers','geoid')
ans =
   1.4441e+08
```

```
almanac('Earth','volume','kilometers','international')
ans =
   1.0833e+12

almanac('Earth','volume','kilometers','actual')
ans =
   1.0832e+12
```

For a complete description of available data, see the `almanac` function in the online Mapping Toolbox reference documentation.

Angles, Times, and Distances – Units and Notation

Angular Notation

Angles can be represented as variables in the Mapping Toolbox in three ways: degrees, radians, and degrees-minutes-seconds. The toolbox provides functions for converting between these formats.

Degrees

This is the default angular unit notation for the toolbox. Although for most scientific applications, radians are more commonly used, in geographic usage, degrees are much more convenient.

Degree notation is simply decimal notation in terms of degrees. Forty-three and one-half degrees would be 43.5.

Radians

Radian notation is simply decimal notation in terms of radians. Two radians would be 2.0.

Degrees-Minutes-Seconds

Degrees-minutes-seconds, or *dms*, notation is a little more complicated. In text, a dms angle would be *ddd mm' ss"*. For example, 142 15'27"is 142 degrees, 15 minutes, and 27 seconds. There are 60 seconds in a minute and 60 minutes in a degree. In the Mapping Toolbox, when “dms” angles are represented by a single number, the format is *dddmm.ss*. For example, 142 15'27" is 14215.27.

The real value of this notation is for the input of data already in this format. The toolbox includes the `mat2dms` function for easily entering dms data.

A special case of the dms format is the *dm* format, in which seconds are not included.

If you have a three-column matrix in which the columns are degrees, minutes, and seconds, respectively, `mat2dms` will convert it to dms format:

```
dmsmatrix = [45 13 46; 156 45 01; -7 34 12.1]
dmsmatrix =
 45.0000   13.0000   46.0000
156.0000   45.0000    1.0000
 -7.0000   34.0000   12.1000
```

```
dmsformat = mat2dms(dmsmatrix)
dmsformat =
 1.0e+04 *
 0.4513
 1.5645
-0.0734
```

Note: Care must be exercised when working with the dms format; for example, two angles in this format cannot simply be added. It is advisable to convert "dms" data to decimal degrees before working extensively with it.

Converting Between Angle Unit Formats

The toolbox includes a variety of angle unit conversion functions. For example, to convert the dms format values to degrees or to radians, you can use `dms2deg` and `dms2rad`, respectively:

```
degformat = dms2deg(dmsformat)
degformat =
 45.2294
156.7503
 -7.5700
radformat = dms2rad(dmsformat)
radformat =
 0.7894
 2.7358
-0.1321
```

Similar functions include `deg2rad`, `rad2deg`, `deg2dms`, etc. There is also a more general function, `angledim`, which will convert from one format to another. For example, how many degrees are in radians?

```
degs = angledim(1/4*pi,'radians','degrees')
degs =
   45
```

Converting Formatted Angle Strings to Numbers

Many sources of geographic data consist of text with the angles in degrees-minutes-seconds format such as *ddd mm' ss"*. These formatted angle

strings may include the characters for degrees, minutes and seconds, as well as letters for north, south, east, and west or other special characters. These kinds of angle strings cannot be converted to numbers by using the MATLAB `num2str` command. However, you can convert many of these string formats to numeric decimal degrees using the `str2angle` command. The `str2angle` command accepts string matrices or cell arrays of strings containing values formatted in a number of commonly used angle formats.

```
strs = {'123˚30''00"S','123-30-00S','123d30m00sS','1233000S'}

strs =

    '123˚30'00"S'    '123-30-00S'    '123d30m00sS'    '1233000S'

str2angle(strs)

ans =

       -123.5
       -123.5
       -123.5
       -123.5

strs = strvcat(strs{:})

strs =

123˚30'00"S
123-30-00S
123d30m00sS
1233000S

» str2angle(strs)

ans =

       -123.5
       -123.5
       -123.5
       -123.5
```

Time Notation

Times can be represented as variables in the Mapping Toolbox in three ways: hours, seconds, and hours-minutes-seconds. The toolbox provides functions for converting between these formats.

Hours

This is the default time unit notation for the toolbox.

Hour notation is simply decimal notation in terms of hours. Two hours and fifteen minutes would be 2.25.

Seconds

Seconds notation is simply decimal notation in terms of seconds. One hour would be 3600.

Hours-Minutes-Seconds

Hours-minutes-seconds, or *hms* notation, is analogous to dms notation for angles. In text, an hms time would be *hh*:*mm*:*ss*. For example, 12:36:15 is 12 hours, 36 minutes, and 15 seconds. In the Mapping Toolbox, when hms times are represented by a single number, the format is *hhmm.ss*. For example, 12:36:15 is 1236.15.

The real value of this notation is in entering data already in this format. The toolbox includes the `mat2hms` function to easily input hms data, which functions very similarly to the `mat2dms` function described earlier.

Note Care must be exercised when working with the hms format; for example, two times in this format cannot simply be added. It is advisable to convert hms data to decimal hours before working extensively with it.

Converting Between Time Unit Formats

Time units can be converted using functions similar to those described for angle unit conversions. These include `hr2sec` and `hms2hr`, as well as a general conversion function `timedim`, which works just like `angledim`.

Distance Notation

Distances in the Mapping Toolbox can be measured in a number of different units. Functions are provided to convert between nautical miles (nm), statute miles (sm), feet (ft), kilometers (km), meters (m), degrees of arc length (deg), and radians of arc length (rad). The names of these functions are of the form `sm2km`, `km2rad`, etc. A general distance conversion function, `distdim`, is available as well.

There is no single default unit of distance measurement in the toolbox. Navigation functions use nautical miles as a default, almanac functions default to kilometers, and the `distance` function defaults to degrees of arc length. It is essential that you understand the default units of any function you use.

Note When distances are given in terms of angular units (degrees or radians), be careful to remember that these are in terms of arc length. While a degree of latitude always subtends a degree of arc length, this is not necessarily the case for degrees of longitude. Also, using arc length as distance assumes a knowledge of the underlying radius.

On the Earth, a degree of arc length is about 60 nautical miles:

```
nauticalmiles = deg2nm(1)
nauticalmiles =
   60.0405
```

The Earth is the default assumption for these conversion functions. Other radii may be used, however:

```
nauticalmiles = deg2nm(1,almanac('moon','radius'))
nauticalmiles =
   30.3338
```

Working with Vector Maps

Data Format

Vector maps are usually contained in a pair of variables, which represent the latitude-longitude pairs for points of interest. For example, the following two variables can be a vector map:

```
lat = [45.6 -23.47 78]';
long = [13 -97.45 165]';
```

Since this is a made-up example, these points could mean anything. Perhaps they are the locations over which geosynchronous satellites are stationed. If so, you might think of them as three distinct points. Perhaps they represent the starting point, the mid-course marker, and the finish point of an airplane race. Think of them as three points connected by two line segments. Or perhaps they represent the vertices of a triangle bounding a region with a propensity for unexplained phenomena, in which case think of them as describing a polygon.

The Mapping Toolbox provides functionality for each of these interpretations. For many purposes, the distinction is irrelevant; for others, the choice of a function implies the appropriate interpretation. For example, the command `plotm` will display the data as a line, while `fillm` will display it as a filled polygon.

When vector variables are used to represent several segments or polygons, successive objects should be delineated by the insertion of `NaN`s in both variables. For example, if a second segment is to be added to the above map, the two objects can reside in the same pair of variables:

```
lat = [45.6 -23.47 78 NaN 43.9 -67.14 90 -89]';
long = [13 -97.45 165 NaN 0 -114.2 -18 0]';
```

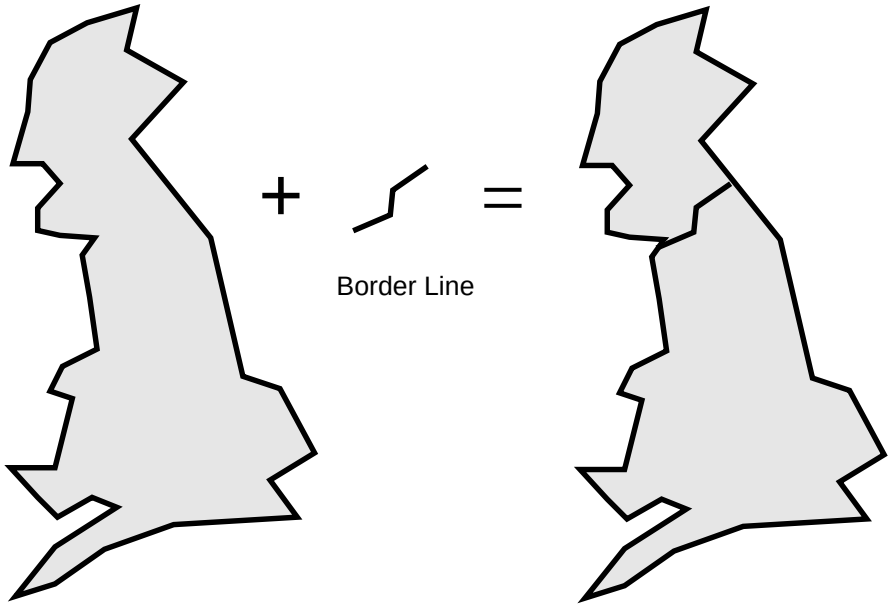
Notice that the `NaN`s must appear in the same locations in both variables. Here we have a segment of three points separated from a segment of four points. The `NaN`s perform two functions; they provide a simple means of identifying break points in the data, and they act as *pen-up* commands when plotting vector maps. The `NaN`s are also used in the Mapping Toolbox to separate non-connected patch faces.

Note This definition differs from MATLAB, which does not display `NaN` clipped data as patches.

Segments versus Polygons

Geographic objects represented by vector data may or may not be formatted as polygons. Imagine two variables, `latcoast`, and `loncoast`, which correspond to sequential points around the coast of the island of Great Britain. If this data returns to its starting point, then a polygon-formatted map of Great Britain exists. This data might be plotted as a patch or as a line, and it might be logically employed in calculations as either.

Now consider two more variables, `latborder` and `lonborder`, which correspond to points along the Anglo-Scottish border, proceeding from the west coast at Solway Firth to the east coast at Berwick-upon-Tweed. This data can really only be plotted as a line. The two pairs of variables together can form a map. They can be displayed as a patch of Great Britain with a line showing the border. However, you still would not have a polygon-formatted map of Scotland alone.



Extracting Polygon or Line Segments

It may be difficult to identify line or patch segments once they have been combined into large NaN-clipped vectors. You can separate these polygon or line vectors in their component segments using `polysplit`. The results is cell arrays in which each element contains a segment. To reverse the results of `polysplit`, use `polyjoin`.

```
[latc,lonc] = polysplit(lat,long);
[latc{2} lonc{2}]

ans =

         43.9            0
       -67.14       -114.2
           90          -18
          -89            0
```

```
[lat2,lon2] = polyjoin(latc,lonc);
[lat long]

ans =

        45.6          13
      -23.47      -97.45
          78         165
         NaN         NaN
        43.9           0
      -67.14      -114.2
          90         -18
         -89           0
```

Merging Line Segments

A common operation on sets of line segments is the merging of segments that have matching ends. The `polymerge` command takes latitude and longitude vectors and compares the endpoints of the segments to see which have endpoints that match exactly or to some tolerance. The matching segments are then combined until no more matching segments are found.

```
lat = [3 2 NaN 1 2 NaN 5 6 NaN 3 4]' ;
lon = [13 12 NaN 11 12 NaN 15 16 NaN 13 14]' ;
[latm,lonm] = polymerge(lat,lon);
[latm lonm]

ans =

     5    15
     6    16
   NaN   NaN
     1    11
     2    12
     2    12
     3    13
     3    13
     4    14
```

Creating Vector Data

In addition to the vector map data available in the Mapping Toolbox and from external sources for such things as coastlines, political boundaries, and other more *permanent* objects, you may want to create your own vector data for such things as great circles, rhumb lines, and small circles.

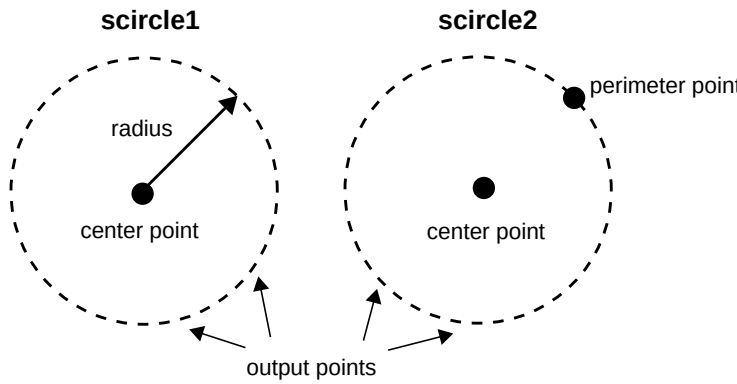
Calculating Small Circles

You can calculate vector data for points along a small circle in two ways. If you have a center point and a known radius, use `scircle1`; if you have a center point and a single point along the circumference of the small circle, use `scircle2`. For example, to get data points describing the small circle at 10 distance from (67 N, 135 W), use the following:

```
[latc,lonc] = scircle1(67,-135,10);
```

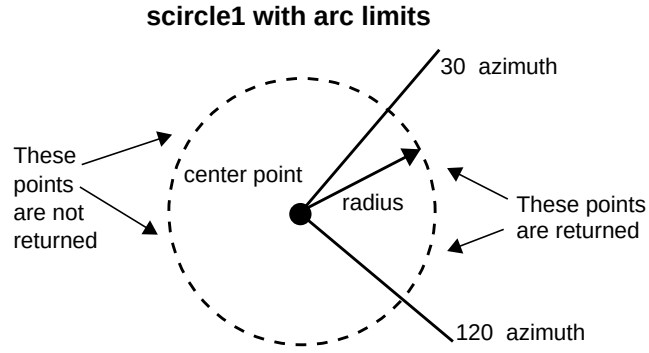
To get the small circle centered at the same point that passes through the point (55 N,135 W), use `scircle2`:

```
[latc,lonc] = scircle2(67,-135,55,-135);
```



The `scircle1` function also allows points along a specific arc of the small circle to be calculated. For example, if you want to know the points 10 in distance and between 30 and 120 in azimuth from (67 N,135 W), simply provide arc limits:

```
[latc,lonc] = scircle1(67,-154,10,[30 120]);
```



When an entire small circle is calculated, the data is in polygon format. For all calculated small circles, 100 points are returned unless otherwise specified. You can calculate several small circles at once by providing vector inputs. For more information, see the `scircle1` and `scircle2` functions in the online Mapping Toolbox reference documentation.

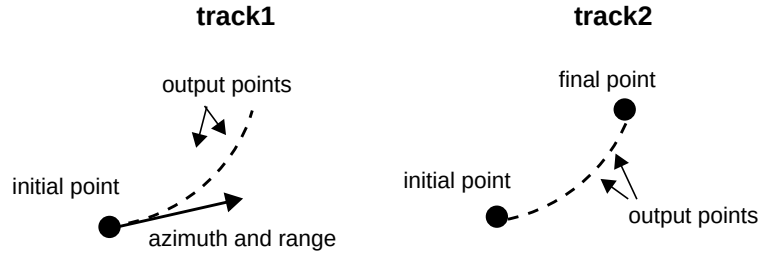
Calculating Tracks – Great Circles and Rhumb Lines

You can generate vector data corresponding to points along great circle or rhumb line tracks using `track1` and `track2`. If you have a point on the track and an azimuth at that point, use `track1`. If you have two points on the track, use `track2`. For example, to get the great circle path starting at (31 S, 90 E) with an azimuth of 45 with a length of 12 , use `track1`:

```
[latgc,longc] = track1('gc',-31,90,45,12);
```

For the great circle from (31 S, 90 E) to (23 S, 110 E), use `track2`:

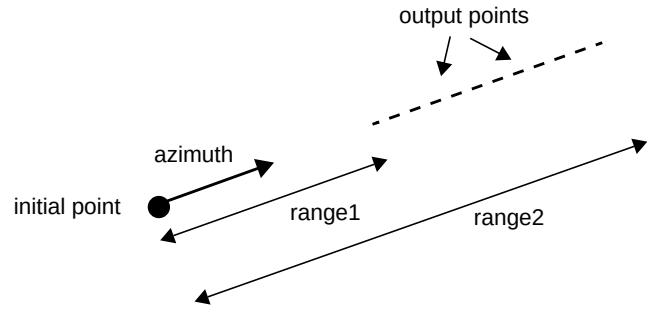
```
[latgc,longc] = track2('gc',-31,90,-23,110);
```



The `track1` function also allows range endpoints to be specified. For example, if you want points along a rhumb line starting 5 away from the initial point and ending 13 away, at an azimuth of 55 , simply specify the range limits:

```
[latrh,lonrh] = track1('rh',-31,90,55,[5 13]);
```

track1 with range limits



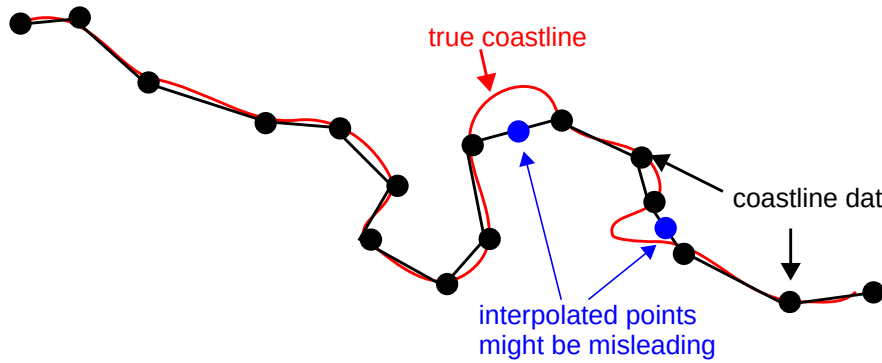
When no range is provided for `track1`, the returned points represent a *complete track*. For great circles, a complete track is 360 , encircling the planet and returning to the initial point. For rhumb lines, the complete track terminates at the poles, unless the azimuth is 90 or 270 ,in which case the complete track is a parallel that returns to the initial point.

For calculated tracks, 100 points are returned unless otherwise specified. You can calculate several tracks at one time by providing vector inputs. For more information, see the `track1` and `track2` functions in the online Mapping Toolbox reference documentation. More vector path calculations are described later in the section,“Navigation” on page 6-11.

Geographic Interpolation

When using vector data, you must be careful when you make assumptions concerning geographic reality between data points. For instance, when plotting vector data, you might connect each point with a straight line segment. This does not usually indicate any true knowledge about the region between known points. Data consisting of points along a coastline might be sparse; in the absence of other knowledge, filling in data can be misleading.

Interpreting Sparse Vector Data



Despite the dangers of misinterpretation, many circumstances exist in which geographic data interpolation is useful or necessary. Sparser data can be linearly *filled-in* with the `interpm` function.

Consider a set of latitude and longitude points that you want to be no further than one degree in separation in either direction.

```
lats = [1 2 4 5]; longs = [1 3 4 5]; maxdiff = 1;

[newlats,newlongs] = interpm(lats,longs,maxdiff)
newlats =
   1.0000
   1.5000
   2.0000
   3.0000
   4.0000
   5.0000
newlongs =
   1.0000
   2.0000
   3.0000
   3.5000
   4.0000
   5.0000
```

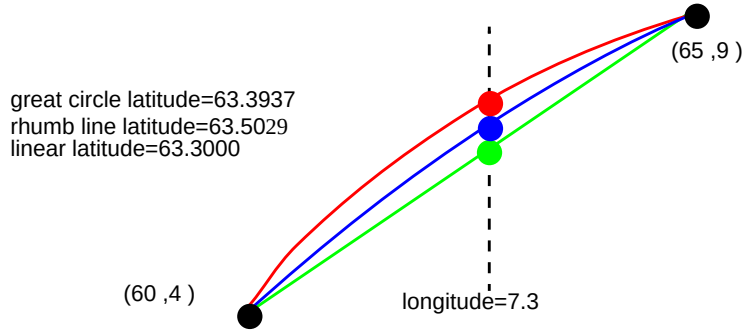
In the original lats, there is a gap of 2 degrees between the 2 and the 4. A linearly interpolated point, (3,3.5) was therefore inserted in newlats and newlongs. Similarly, in the original longs, there is a gap of 2 degrees between the 1 and the 3. The point (1.5,2) was therefore interpolated and placed into newlats and newlongs. Now, no adjacent points in either newlats or newlongs is greater than maxdiff in separation.

The interpm function returns the original data with new linearly interpolated points inserted. Sometimes, however, only the interpolated values are desired. The commands intrplat and intrplon provide a capability similar to MATLAB's interp1 command, allowing for different methods of interpolation.

Use intrplat to interpolate a latitude for a given longitude. Given a monotonic set of longitudes and their matching latitude points, you can interpolate a new latitude for a given longitude in a linear, spline, cubic, rhumb line, or great circle sense.

Here, find the latitude corresponding to a longitude of 7.3 in the following data in a linear, great circle, and rhumb line sense:

```
longs = [1 3 4 9 13]; lats = [57 68 60 65 56]; newlong = 7.3;

newlat = intrplat(longs,lats,newlong,'linear')
newlat =
   63.3000

newlat = intrplat(longs,lats,newlong,'gc')
newlat =
   63.5029

newlat = intrplat(longs,lats,newlong,'rh')
newlat =
   63.3937
```



The `intrplon` function provides the same capability for interpolating new longitudes for given latitudes.

Polygon Area

For vector data in polygon format, geographic areas can be calculated using the command `areaint`. The function performs a numerical integration using Green's Theorem for the area on a surface enclosed by a polygon. Since this is a discrete integration on discrete data, the results are not exact. Nevertheless, the method provides the best means of calculating the area of arbitrarily shaped regions. Better measures result from better data.

The Mapping Toolbox function `areaint` (for area by integration), like the other area functions `areaquad` and `areamat`, returns areas as a fraction of the entire planet's surface, unless a radius is provided. Here we calculate the area of the continental United States using the `usalo` workspace. Three areas are returned, because the data contains three polygons: Long Island, Martha's Vineyard, and the rest of the continental U.S.:

```
load usalo
earthradius = almanac('Earth','radius');
area = areaint(uslat,uslon,earthradius)
area =
  1.0e+06 *
        7.9256
        0.0035
        0.0004
```

Since the default Earth radius is in kilometers, the area is in square kilometers. From the same workspace, the areas of the Great Lakes can be calculated, this time in square miles:

```
earthradius = almanac('Earth','radius','miles');
area = areaint(gtlakelat,gtlakelon,earthradius)
area =
  1.0e+04 *
        8.0124
        1.0382
        0.7635
```

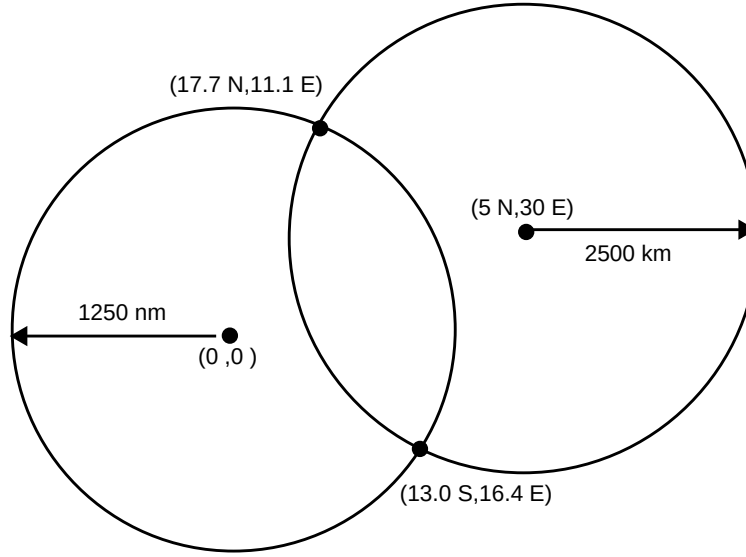
Again, three areas are returned, the largest for the polygon representing Superior, Michigan, and Huron together, the other two for Erie and Ontario.

Vector Calculations – Intersections

The Mapping Toolbox provides a set of functions to perform intersection calculations on vector data computed by the toolbox, which include great and small circles, as well as rhumb line tracks. The functions also determine intersections of arbitrary vector data.

Compute the intersection of a small circle centered at (0 ,0) with a radius of 1250 nautical miles and a small circle centered at (5 N,30 E) with a radius of 2500 kilometers:

```
[lat,long] = scxsc(0,0,nm2deg(1250),5,30,km2deg(2500))
lat =
   17.7487 -12.9839
long =
   11.0624 16.4170
```

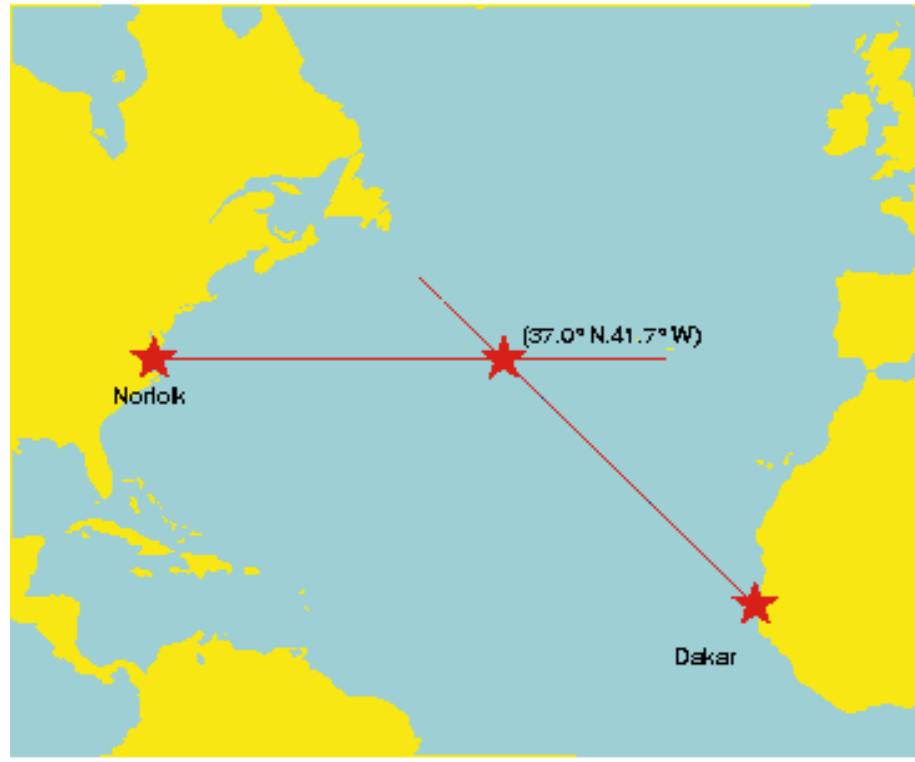


Notice that, in general, small circles intersect twice or never. For the case of exact tangency, scxsc returns two identical intersection points. Other similar commands include rhxrh for intersecting rhumb lines, gcxgc for intersecting great circles, and gcxsc for intersecting a great circle with a small circle.

Imagine a ship setting sail from Norfolk, Virginia (37 N,76 W), maintaining a steady due-east course (90),and another ship setting sail from Dakar, Senegal (15 N,17 W), with a steady northwest course (315). Where would the tracks of the two vessels cross?

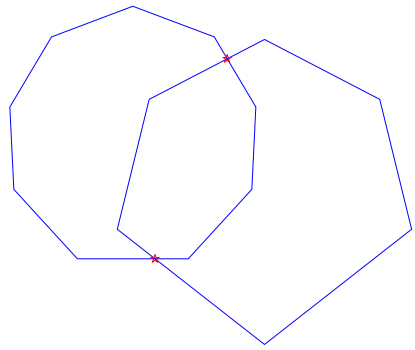
```
[lat,long] = rhxrh(37,-76,90,15,-17,315)
lat =
   37
long =
   -41.7028
```

The intersection of the tracks is at (37 N,41.7 W), which is roughly 600 nautical miles west of the Azores in the Atlantic Ocean.



You can also compute the intersection points of arbitrary vectors of latitude and longitude. The `polyxpoly` command finds the segments that intersect and interpolates to find the intersection points. The interpolation is done linearly, as if the points were in a cartesian x-y coordinate system. The `polyxpoly` command can also identify the line segment numbers associated with the intersections.

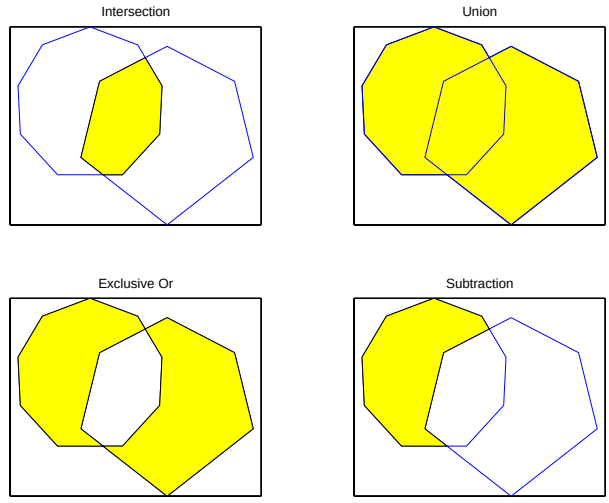
```
[xint,yint] = polyxpoly(x1,y1,x2,y2);
```



If the spacing between points is large, there may be some difference between the intersection points computed by `polyxpoly` and the intersections shown on a map display. This is a result of the difference between straight lines in the unprojected and projected coordinates. Similarly, there may be differences between the `polyxpoly` result and intersections assuming great circle or rhumb lines between points.

Polygon Boolean Operations

Polygon boolean operations are used to answer a variety of questions about vector data. Standard boolean operations include intersection, union, subtraction, and an exclusive or operation. The `polybool` command carries out these operations on pairs of vectors. These vectors can be thought of as x-y coordinates or as latitude-longitude pairs. As for the `intersection` command, the interpolations are carried out on the coordinates as if they were cartesian. Here is an example that shows all of the available operations.

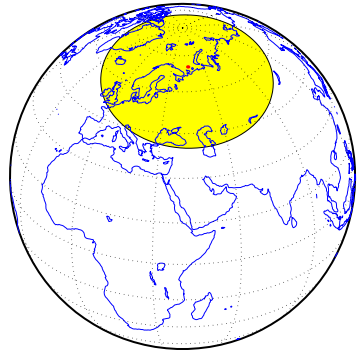


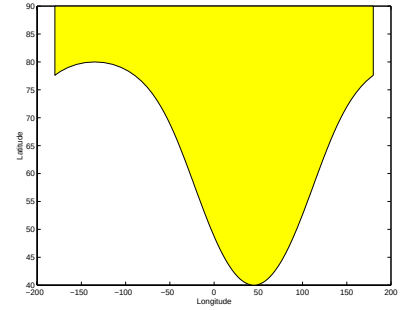
The result is returned as NaN-clipped vectors by default. In cases where it is important to distinguish outer contours of polygons from interior holes, `polybool` can also accept inputs and return outputs as cell arrays. In the cell array format, a cell array entry starts with the list of points making up the outer contour. Subsequent NaN-clipped faces within the cell entry are interpreted as interior holes.

Cutting Polygons at the Dateline

Polygon boolean operations treat the inputs as cartesian coordinates. The `polyxpoly` command can be confused by geographic data that has jumps in the longitude coordinate at dateline crossings. You can prepare such geographic data for use with `polybool` by cutting the polygons at the dateline with the `flatearthpoly` command. The result of `flatearthpoly` is a polygon with points inserted to follow the dateline up to the pole, traverse the longitudes at the pole, and return down to the dateline crossing along the other edge of the dateline.

```
[lat,lon] = scircle1(70,45,30);
[latf,lonf] = flatearthpoly(lat,lon);
```





For the purpose of picking a pole for the dateline cut, the interior of the polygon is assumed to be in the hemisphere that contains most of the edge points. Thus a polygon produced by `flatearthpoly` will not cover more than a hemisphere.

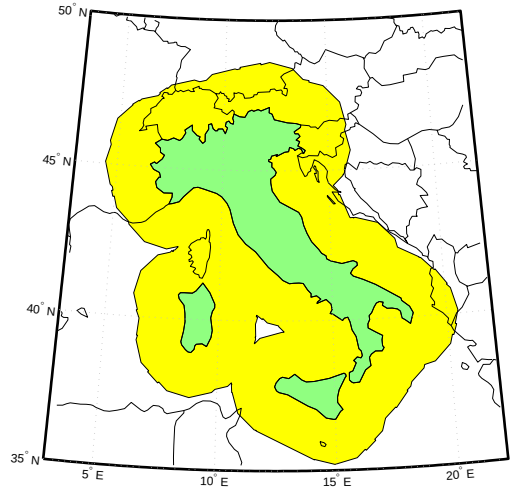
Note You do not need to use the `flatearthpoly` command to prepare data for a map display. The Mapping Toolbox display commands automatically cut and trim geographic data if required by the map projection.

Buffer Zones

A *buffer zone* is the area within a certain distance of a set of points. For a closed polygon, the buffer zone is defined as the locus of points that are a certain distance in or out of the polygon. A buffer zone for an open polygon is the locus of points a certain distance out from the polygon. The `bufferm` command computes the buffer points by placing small circles at the vertices of the polygon and rectangles on the line segments, and applying the appropriate polygon boolean operation.

If the outline of Italy were saved in NaN-clipped latitude and longitude vectors, you could compute the set of points that are a distance of 1.5 degrees out from the coast.

```
[latb,lonb] = bufferm(lat,lon,1.5,'out');
```



Trimming Vector Data to a Defined Region

Sometimes vector data may extend beyond the geographic region of interest. For example, if you have world coastline data, but you require a map of Australia only, you may want to create new variables that contain only the data you need. You might also want to trim your data to save memory or to speed up calculations and display.

We can distinguish between trimming line data and patch data. Line data can be trimmed by simply removing points outside the region of interest. Patch data requires a more complicated method to ensure that the patch objects are correctly displayed. For the vector data, two functions are available to achieve this. If the variables are to be treated as line data, the `maptriml` command will return variables containing only those points in the defined region. If polygon format must be retained, the `maptrimp` command will ensure this, although compared to the line formatted data, the patch-trimmed data will be larger and take more time to compute.

Note For viewing purposes, trimming the data before display is not required. The vector data is automatically trimmed to the region specified by the frame limits (`FLatLimit` and `FLonLimit` map axes properties) for azimuthal projections and frame or map limits (`MapLatLimit` and `MapLonLimit` map axes properties) for non-azimuthal projections. The trimming is done internally in the display routine, so the original data remains intact. For further information on trimming geographic vector data, see "Map Axes" on page 3-5 of this document, along with the reference pages for the trimming functions in the online Mapping Toolbox reference documentation,

Load the `coast` MAT-file, and trim the vector data to a region centered on Australia:

```
load coast
whos
   Name       Size       Bytes     Class

   lat        9589x1     76712     double array
   long       9589x1     76712     double array

latlim = [-50 0]; longlim = [105 160];
[linelat,linelong] = maptriml(lat,long,latlim,longlim);
[polylat,polylong] = maptrimp(lat,long,latlim,longlim);

whos
   Name       Size       Bytes     Class

   lat        9589x1     76712     double array
   latlim      1x2       16        double array
   linelat    870x1      6960      double array
   linelong   870x1      6960      double array
   long       9589x1     76712     double array
   longlim    1x2        16        double array
   polylat    1020x1     8160      double array
   polylong   1020x1     8160      double array
```

Simplifying Vector Data

Sometimes you may need to reduce the number of points in your vector data, while still maintaining an accurate representation of the geographical data. A quick and easy method would be to throw out every other element or set of elements in each vector. However, the result may be a very crude representation of the original data. Another option exists. The Mapping Toolbox provides a function implementing a powerful line simplification algorithm that selectively deletes points.

Note 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.

For example, reduce the number of points in the state of Massachusetts with the `reducem` function:

```
load usahi
whos

   Name       Size       Bytes     Class

   stateline  1x51       837656    struct array
```

```
   statepatch 1x51      837758    struct array
   statetext  1x51      51190     struct array
```

The `stateline` structure contains vector data for all 50 U.S. states and the District of Columbia. We are only interested in the state of Massachusetts, the 19th entry in the structure:

```
lat = stateline(19).lat;
long = stateline(19).long;
clear state*
```

Now simplify your vector data:

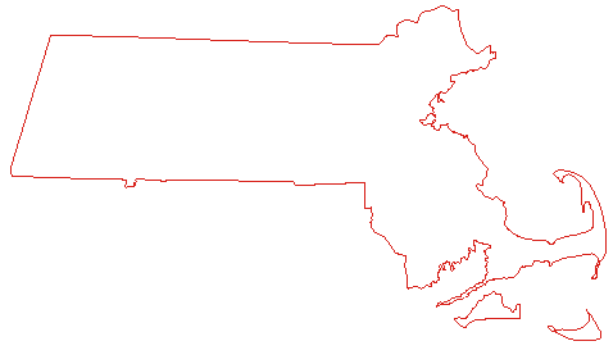
```
[newlat,newlong,cerr,tol] = reducem(lat,long);
whos
   Name       Size       Bytes     Class

   cerr        1x1       8         double array
   lat        958x1      7664      double array
   long       958x1      7664      double array
   newlat      253x1     2024      double array
   newlong    253x1      2024      double array
   tol         1x1       8         double array
```

The Massachusetts vector data has been reduced to almost a quarter of its original size:

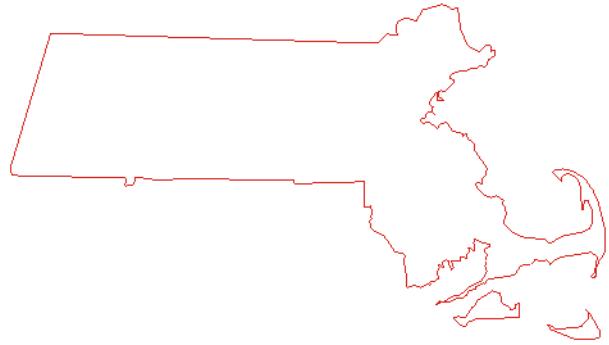
```
size(newlat)/size(lat)
ans =
    0.2641
```

Plots of the datasets, however, show two virtually identical maps:

original (958 points)



reduced (253 points)

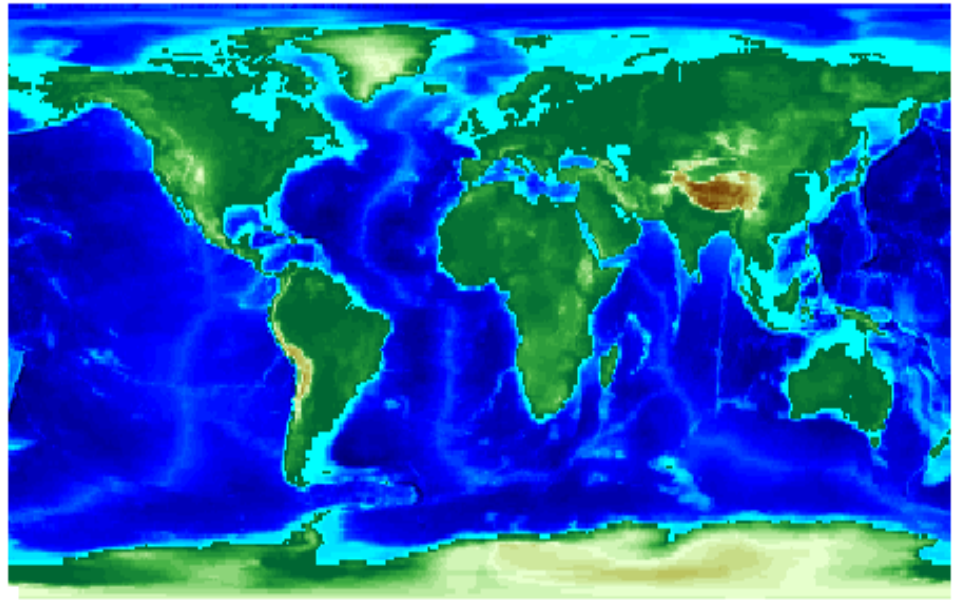


Working with Matrix Maps

One of the most powerful features of the Mapping Toolbox is its capacity for manipulating matrix maps. Two types of matrix maps are defined by the Mapping Toolbox: regular and general. Regular matrix maps are rectangular in appearance and are a subset of general matrix maps, which can be of practically any shape or orientation.

Regular Matrix Maps

A *regular matrix map* is one in which columns of data run south to north, rows of data run west to east, and each matrix element represents the same angular step in each direction, for all rows and columns. For example, if each row represents one degree of latitude, and each column one degree of longitude, and the matrix is aligned north and south, then it is a regular matrix map. The `topo` map provided with MATLAB is such a regular matrix map:



The Map Legend

Many regular matrix maps do not cover the entire planet. It is therefore necessary to identify the scale and placement of the map with a special data structure, the *map legend*. 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 map. The three entries are given in terms of angular units of degrees only. The specific format of the structure is as follows:

```
maplegend = [cells/angleunit north-latitude west-longitude]
```

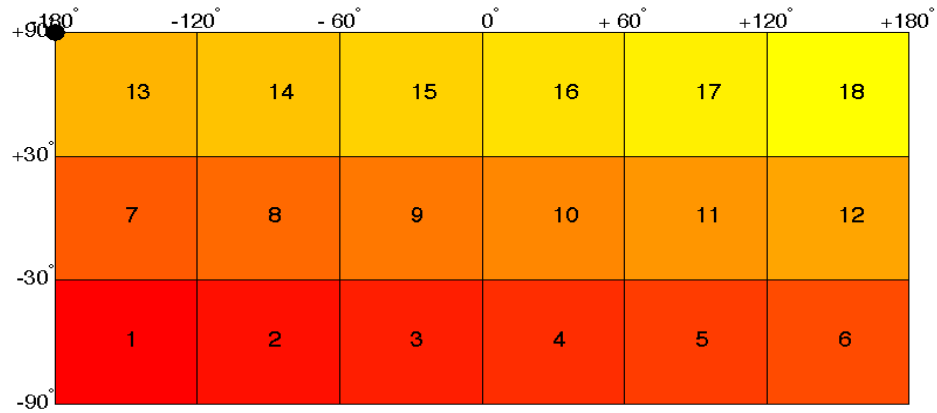
For the `topo` matrix map, each matrix cell represents one degree, the northern edge is the North Pole, and the western edge is the Prime Meridian. Therefore, the map legend for `topo` is:

```
topolegend = [1 90 0]
```

This map legend structure is stored in the `topo` workspace. Imagine an extremely coarse map of the world in which each cell represents 60 . Such a map matrix would be 3-by-6, and its map legend would be:

```
maplegend = [1/60 90 -180] = [0.0167 90 -180]
```

In this case, the map's western edge is the International Date Line, at 180 W:



Note that the first row of the matrix is displayed as the bottom of the map, while the last row is displayed as the top. All regular matrix maps in the

Mapping Toolbox, as well as regular surfaces in MATLAB, are displayed in this fashion.

As an example of a matrix map that does not encompass the entire world, load the Cape Cod image using the `loadcape` script.

```
loadcape
maplegend
maplegend =
   120    44   -72
```

The `maplegend` indicates that there are 120 cells per angular degree. This has a horizontal resolution 120 times finer than that of the `topo` matrix map, which was one cell per degree. The `cape` region covered here extends from 41 N to 44 N, and from 72 W to 69 W.

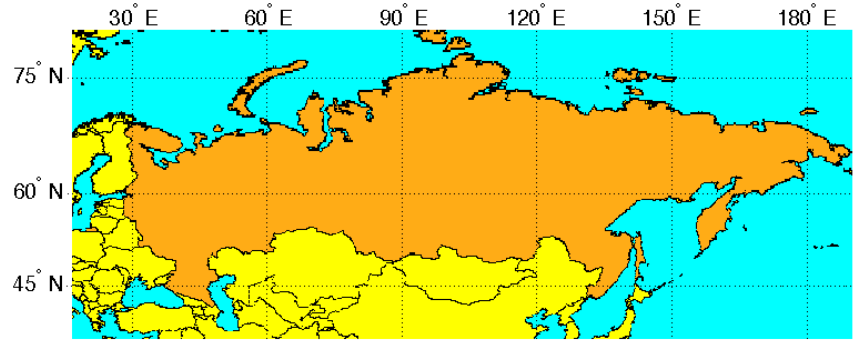
```
[latlimits,longlimits] = limitm(map,maplegend)
latlimits =
   41    44
longlimits =
   -72   -69
```

The Geographic Meaning

A matrix map and its associated map legend can provide a host of geographic information regarding the map and its entries. First, as was just demonstrated, the north, south, east, and west limits of the mapped area can be determined as follows:

```
load russia
[latlim,longlim] = limitm(map,maplegend)
latlim =
   35    80
longlim =
   15   190
```

The map in the `russia` workspace extends over the International Date Line (180 longitude). You could use the previously described command`npi2pi` to rename the eastern limit to be -170, or 170 W.



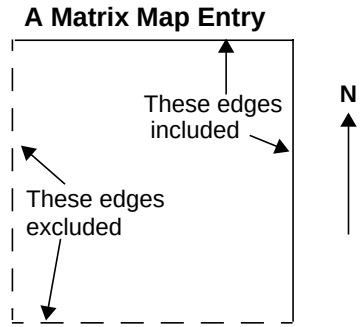
The command `setltln` allows you to determine the geographic coordinates of a particular matrix element. The returned coordinates actually show the center of the geographic area represented by the matrix entry:

```
row = 23; col = 79;
[lat,long] = setltln(map,maplegend,row,col)
lat =
   39.5000
long =
   30.7000
```

You can also determine the row and column of the matrix map entry containing a given point:

```
[r,c] = setpostn(map,maplegend,lat,long)
r =
   23
c =
   79
```

Each matrix entry can be thought of as an angular *square* which includes its northern and eastern edges, but not its western and southern edges:



The exceptions to this are that the southernmost row (row 1) also contains its southern edge, and the westernmost column (column 1) contains its western edge, except when the map encompasses the entire 360 of longitude. In that case, the westernmost edge of the first column is not included, because it is identical to the easternmost edge of the last column. These exceptions ensure that all points on the globe can be represented exactly once in a regular matrix map.

Although each map matrix entry represents an area, not a point, it is often convenient to assign singular coordinates for reference. An entry is defined by the point in the center of the area represented by the entry. For example, reference the center cell coordinate for the 3rd row, 17th column, of the Russia map:

```
row = 3; col = 17;
[lat,long] = setltln(map,maplegend,row,col)
lat =
   35.5000
long =
   18.3000
```

Since the cells in the Russia matrix map represent 0.2 squares (5 cells per degree), the cell in question extends from 35.6 S to 35.4 S and from 18.2 E to 18.4 E.

Accessing Matrix Map Elements

The actual values contained within the map matrix entries are important as well. The Mapping Toolbox provides several functions for accessing and altering the values of matrix map entries.

If the actual row and column of a desired entry are known, then a simple matrix index can return the appropriate value. Use the row and column from the last example (3rd row, 17th column) to determine the value of that cell:

```
value = map(row,col)
value =
   2
```

More often, the geographic coordinates are known:

```
value = ltln2val(map,maplegend,lat,long)
value =
   2
```

The latitude-longitude coordinates associated with particular values in a matrix map can be found with `findm`, which is similar to the standard MATLAB command `find`. Here we find the coordinates in the `topo` map with values greater than 5500 meters:

```
load topo
[lats,longs] = findm(topo>5500,topolegend);
[lats longs]
ans =
   34.5000   79.5000
   34.5000   80.5000
   30.5000   84.5000
   28.5000   86.5000
```

You can change all instances of a given value in a matrix map to a new value in one step. Here is an example using the map of Russia:

```
oldcode = ltln2val(map,maplegend,37,79)
oldcode =
   4
newmap = changem(map,5,oldcode);
newcode = ltln2val(newmap,maplegend,37,79)
newcode =
   5
```

All entries in newmap corresponding to 4s in map now have the value 5. You can also define a mask to determine map entries to change. A mask is a matrix the same size as the map matrix, with 1s everywhere the entry is to take on the new value. A mask is often generated by a logical operation on a map variable, a topic which is described in greater detail below. The map in the russia workspace contains 3s for Russia. If you want to set every non-Russia matrix entry to zero, you can use maskm:

```
nonrussia = (map~=3);
newmap = maskm(map,nonrussia,0);
newcode = ltln2val(newmap,maplegend,37,79)
newcode =
   0
```

Finally, if you know the latitude and longitude limits of a region, you can calculate the required matrix size and an appropriate map legend for a desired map scale. Before making a large, memory-taxing matrix map, you should first determine its size. For a map of the continental United States at a scale of 10 cells per degree, do the following:

```
scale = 10;
[r,c,maplegend] = sizem([25 55],[-130 -60],scale)
r =
   300
c =
   700
maplegend =
   10    55  -130
```

This matrix map would be 300-by-700. What if the scale were reduced to 5 rows/columns per degree?

```
scale2 = 5;
[r,c,maplegend] = sizem([25 55],[-130 -60],scale2)
r =
   150
c =
   350
maplegend =
   5    55  -130
```

A 150-by-300 matrix might be more manageable.

Valued Maps and Indexed Maps

A *valued map* is a matrix map in which the entries represent some value or measurement. A simple example of a valued map is the `topo` matrix in the `topo` workspace. Each entry of `topo` is an average elevation in meters for that piece of the Earth. The entries of a valued map answer the question, *how much is here?*

An *indexed map* is a matrix map in which the entries are an index value indicating where more information might be found. A simple example of an indexed map is the `map` variable in the `worldmtx` workspace. Each entry of `map` is an index to the `nations` structure indicating which nation contains that piece of the Earth. The entries of an indexed map answer the question, *what is this?*

To illustrate the difference between valued maps and indexed maps, consider two geographic points, (45 N,105 E) and (47 N,108 E). The valued map `topo` contains a different kind of information for these points from what is found in the indexed world `map`:

```
load topo
ltln2val(topo,topolegend,45,105)
   1270
ltln2val(topo,topolegend,47,108)
ans =
   1513
```

As might be expected, the two regions represented by the appropriate entries of `topo` have different average elevation values. However, they have the same index value in `map`, because they are in the same country:

```
load worldmtx
ltln2val(map,maplegend,45,105)
ans =
   119
ltln2val(map,maplegend,47,108)
ans =
   119

nations(119).name
ans =
Mongolia
```

Matrix Maps as Logical Variables

A powerful use of matrix maps is the application of logical conditions to create *logical maps*. Logical maps are matrix maps consisting entirely of 1s and 0s. They can be created by performing logical tests on matrix map variables. The resulting binary matrix is the same size as the original map(s) and can use the same map legend, as the following illustrates:

```
logicalmap = (realmap>0)
```

Multiple conditions can operate on the same map:

```
logicalmap = (realmap>-100)&(realmap<100)
```

Or, if several maps are the same size and share the same map legend, a logical map can be created by testing conditions on several *layers* of maps:

```
logicalmap = (population>10000)&(elevation<400)&...
             (country==nigeria)
```

The Mapping Toolbox provides functions that enable the creation of logical maps. For example, the 5-cell-per-degree maps covering the continental United States of all 1s and all 0s can be easily constructed:

```
latlims = [25 55]; longlims = [-130 -60]; scale = 5;
onesmap = onem(latlims,longlims,scale);
zeroesmap = zerom(latlims,longlims,scale);
```

Maps of all NaNs and sparse maps of all 0s can be made in a similar fashion with the commands `nanm` and `spzerom`.

A useful way of analyzing the results of logical map manipulations is to determine the area satisfying the conditions (i.e., with a logical value of 1). The command `areamat` can provide the fractional surface area on the globe associated with 1s in a logical map. Each matrix element is a quadrangle, and the sum of the areas meeting the logical condition provides the total area. For example, according to the `topo` map, what fraction of the Earth lies above sea level?

```
load topo
a = areamat((topo>0),topolegend)
a =
   0.2890
```

The answer is about 30%. If a planetary radius is provided in, for example, kilometers, the total physical area above 0 elevation can be returned in square kilometers:

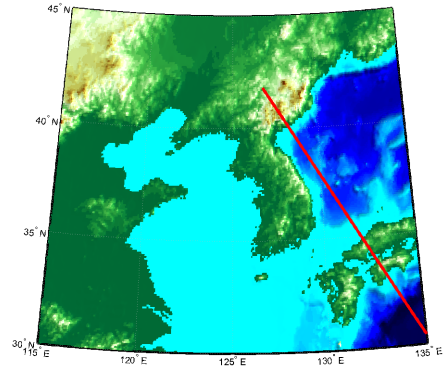
```
a = areamat((topo>0),topolegend,almanac('Earth','radius'))
a =
   1.4739e+08
```

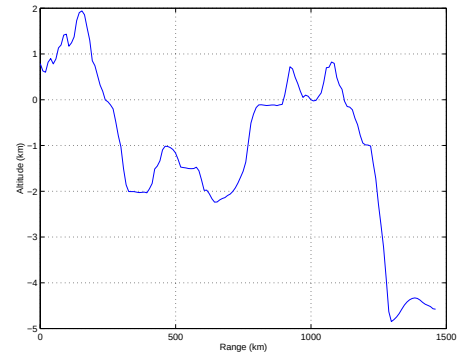
To interpret this graphically, remember that the logical matrix map (topo>0) is binary. The following is a display of the map with black where the condition is true (i.e., the logical map entry is "1") and white where it is false ("0"). It is displayed in an Equal-Area Cylindrical projection, which vertically shrinks rows near the poles to show their true relative area. Similarly, the `areamat` command *shrinks* the contribution of these rows to the total area.

Matrix Map Values Along a Path

A common calculation for geographic matrix data is determining the values along a path. Examples include calculating terrain height along a straight section or along the path of a road. Use the `mapprofile` command to calulate regular matrix map values along a path. You can define the path and data by clicking on an existing map display, or by providing the command with a regular matrix map and vectors of waypoints. Similarly, the result can be displayed automatically in a new plot or returned as output arguments for furthur analysis or display.

Here is an example that shows the elevation profile along a straight line in the Korean elevation data.



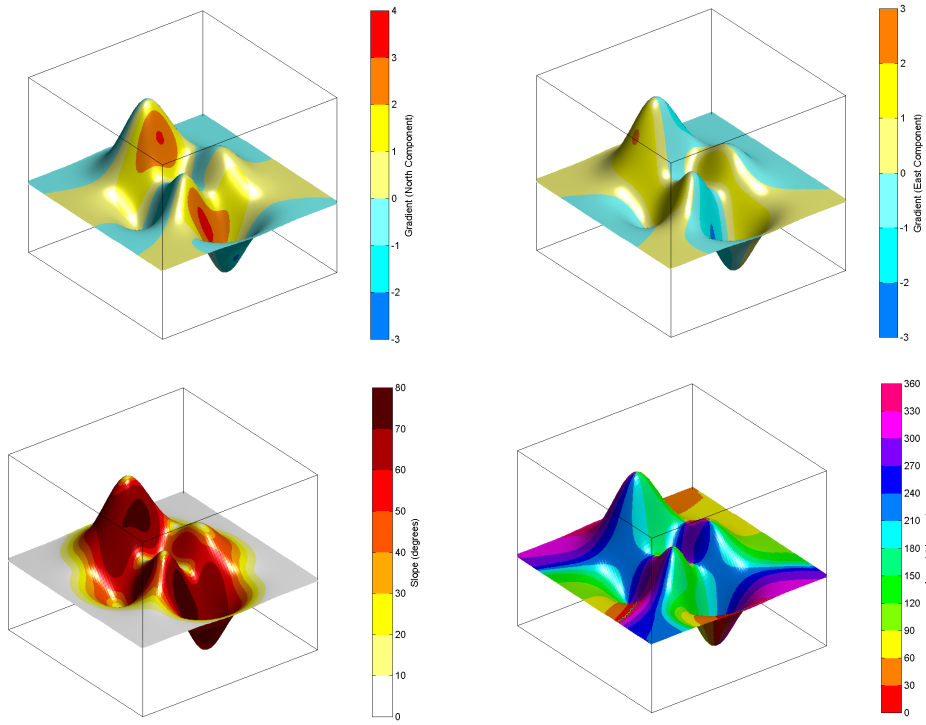


Matrix Map Gradient, Slope, and Aspect

A map profile is often computed to find the slope along a path. A related calculation is the calculation of the slope at all points on a matrix. The `gradientm` command uses a finite-difference approach to compute the gradient of a regular or matrix map. The function returns the components of the gradient in the north and east directions, as well as the slope and aspect. The *gradient* components are the change in the map variable per meter of distance in the north and east directions. If the map contains elevations in meters, the *aspect* and *slope* are the angles of the surface normal clockwise from north and up from the horizontal. The angles are in units of degrees by default.

Here are the components of the gradient, slope, and aspect for a regular matrix map based on the MATLAB peaks matrix. The examples show the `gradientm` outputs as colors on a three-dimensional surface of elevation. The surfaces are shown with no vertical exaggeration.

```
map = 500*peaks(100);
maplegend = [ 1000 0 0];
[aspect,slope,gradN,gradE] = gradientm(map,maplegend);
```



Visibility in Terrain

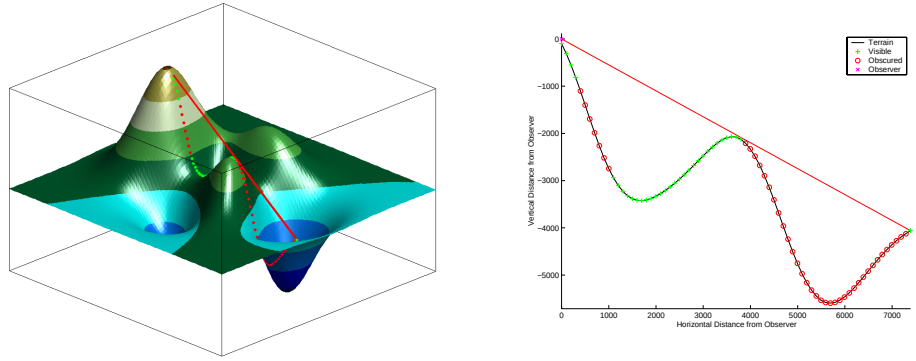
Regular matrix maps that contain elevation data can be used to answer questions about the mutual visibility of points. Is the line of sight from one point to another obscured by terrain? What area can be seen from a point, or the inverse question, what area can see that point?

The first question, on the line of sight between two points, can be answered with the `los2` command. In the simplest calling form, `los2` determines the visibility between two points on the surface of a digital elevation map. You can also specify the altitude of the observer and target points, as well as the datum with respect to which the altitudes are measured. For specialized applications, you can even control the actual and effective radius of the Earth, allowing use

of the assumption frequently used in predicting radio wave propagation that the Earth has a radius 1/3 larger than the actual value.

The following example shows the line of sight results between two points on the peaks regular matrix map. The calculation shows that the two points are mutually visible, but that the line of sight just barely clears an intervening peak. The `los2` command also returns visibility results along the line of sight that can be used to understand why the answer is true or false.

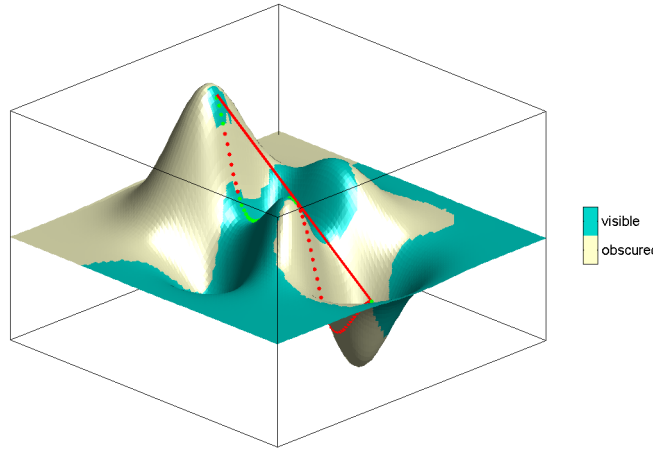
```
map = 500*peaks(100);
maplegend = [ 1000 0 0];
lat1 = -0.027; lon1 = 0.05; lat2 = -0.093; lon2 = 0.042;

los2(map,maplegend,lat1,lon1,lat2,lon2,100)

ans =

     1
```



You can also compute the *viewshed*, a name derived from watershed, which is all of the areas that are visible from a particular point. The `viewshed` command can be thought of as performing the `los2` line of sight calculation from one point on a digital elevation map to every other entry in the matrix. The `viewshed` command supports the same options as `los2`.

The following shows which parts of the peaks elevation map in the previous example are visible from the first point.

```
[vismap,vismapleg] = viewshed(map,maplegend,lat1,lon1,100);
```



General Matrix Maps

In addition to regular matrix maps, the Mapping Toolbox provides another matrix map format: *general matrix maps*. These maps can be displayed, and their values and coordinates can be queried, but much of the functionality available for regular matrix maps cannot be exploited for general matrix maps.

The examples thus far have shown maps that covered simple, regular quadrangles, that is, geographically rectangular. General matrix maps, in addition to these rectangular orientations, can have other shapes as well.

The Map Format

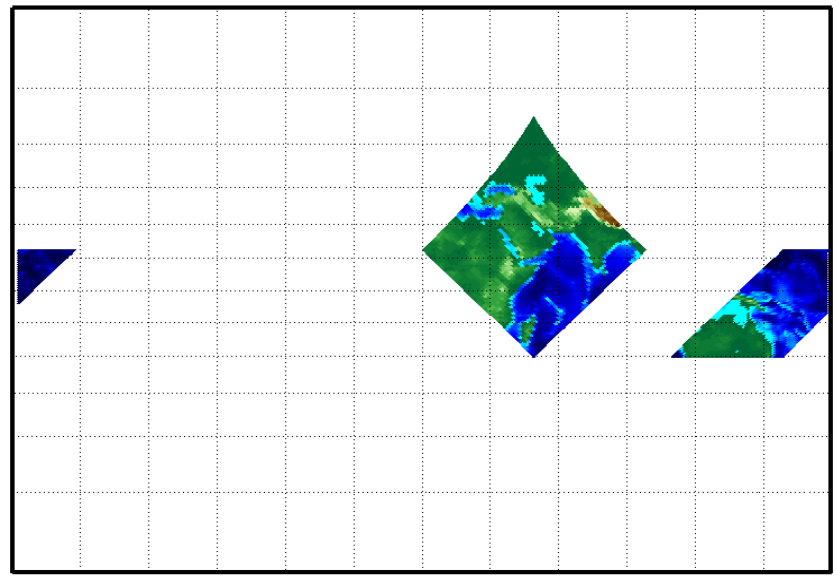
To define a general matrix map, you need three variables: the matrix of indices or values associated with the mapped region, a matrix giving cell-by-cell latitude coordinates, and a matrix giving cell-by-cell longitude coordinates.

An example of an irregularly shaped general matrix map is available in the Mapping Toolbox for examination:

```
load mapmtx
whos
   Name       Size       Bytes     Class

   lg1        50x50      20000     double array
   lg2        50x50      20000     double array
   lt1        50x50      20000     double array
   lt2        50x50      20000     double array
   map1       50x50      20000     double array
   map2       50x50      20000     double array
```

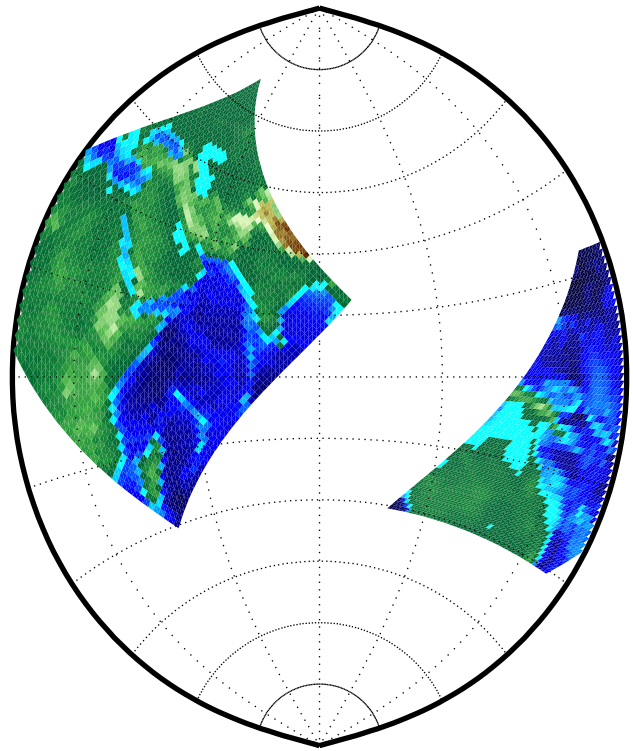
Two general matrix maps are in this workspace, each requiring three variables. The values contained in `map1` correspond to the latitude and longitude coordinates, respectively, in `lt1` and `lg1`. Notice that all three matrices are the same size. Similarly, `map2`, `lt2`, and `lg2` together form a second general matrix map. These datasets were extracted from the `topo` matrix map shown in previous examples. Neither of these maps is regular in that their columns do not run north-and-south.

To get an idea of their geography, display them together:



Notice that neither map is a regular rectangle. One looks like a diamond geographically, the other like a trapezoid. The trapezoid is displayed in two pieces because it crosses the edge of the map. These shapes can be thought of as the geographic organization of the data, just as rectangles are for regular matrix maps. But, just as for regular matrix maps, this organizational logic does not mean that displays of these maps are necessarily a specific shape.

Here you can view these maps together in a Polyconic projection:



Since the Polyconic is limited to a 150 range in longitude, those portions of the maps outside this region are trimmed automatically.

Geographic Interpretations

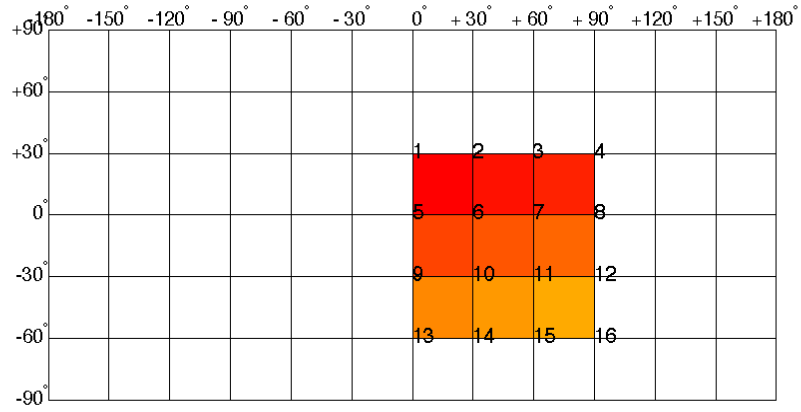
The Mapping Toolbox supports several different interpretations of general matrix maps. First, a map matrix having the same size as the latitude and longitude coordinate matrices represents the values of the map data at the corresponding geographic points. Next, a map matrix having one fewer row and column than the geographic coordinate matrices represents the values of the map data within the area formed by the four adjacent latitudes and longitudes. Finally, if the latitude and longitude matrices are still smaller than the map matrix, you can interpret them as describing a coarse *graticule*, or mesh of latitude and longitude cells into which the blocks of map data are warped.

This section discusses the first two interpretations of general matrix maps. For more information on the use of graticules, see “The Graticule” on page 3-35.

As an example of the first interpretation, consider a 4-by-4 map matrix whose cell size is 30-by-30 degrees, along with its corresponding 4-by-4 latitude and longitude matrices:

```
map = [ 1  2  3  4;...
   5  6  7  8;...
   9 10 11 12;...
   3 14 15 16];

lat = [ 30  30  30  30;...
   0   0   0   0;...
   -30 -30 -30 -30;...
   -60 -60 -60 -60];

long = [0 30 60 90;...
   0 30 60 90;...
   0 30 60 90;...
   0 30 60 90];
```

This general matrix map is displayed with the values of `map` shown at the associated latitudes and longitudes:

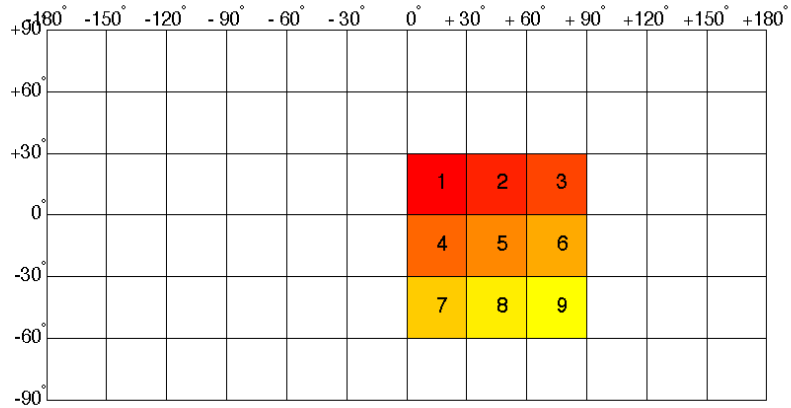


Notice that only 9 of the 16 total cells are displayed. The value displayed for each cell is the value at the upper left corner of that cell, whose coordinates are given by the corresponding `lat` and `long` elements. By MATLAB convention, the last row and column of the map matrix is not displayed, although they exist in the `CData` property of the surface object.

For the second interpretation, consider a 3-by-3 map matrix with the same `lat` and `long` variables:

```
map = [1 2 3;...
       4 5 6;...
       7 8 9];
```

Here is a surface plot of the map matrix, with the values of map shown at the center of the associated cells:



All of the map data is displayed for this general matrix map. The value of each cell is the value at the center of the cell, and the latitudes and longitudes in the coordinate matrices are the boundaries for the cells.

You may have noticed that the first row of the matrix is displayed as the top of the map, whereas for a regular matrix map, the opposite was true; the first row corresponded to the bottom of the map. This difference is due to the lat and long matrices and how they are ordered. The lat and long coordinate matrices determine the arrangement of the general matrix map.

Of course, a regular matrix map can also be considered a general matrix map. All that are required are the graticule or coordinate matrices, which can be produced from the regular matrix map itself using the meshgrat function:

```
load topo
[lat,lon] = meshgrat(topo,topolegend);
whos
   Name       Size       Bytes     Class

   lat         180x360   518400    double array
   lon         180x360   518400    double array
   topo        180x360   518400    double array
   topolegend 1x3        24        double array
```

```
topomap1   64x3      1536      double array
topomap2   128x3     3072      double array
```

A regular matrix map can also be constructed from a general matrix map. The coordinates and values can be “imbedded” in a new regular matrix map with a somewhat coarser resolution. This ensures that every cell in the new map receives a value from the old one. Convert the diamond-shaped general matrix map back to a regular matrix map.

```
load mapmtx

latlim = [min(lt1(:)) max(lt1(:))];
lonlim = [min(lg1(:)) max(lg1(:))];

[map,maplegend] = nanm(latlim,lonlim,1/2);
% original was about 1 cell per degree

map = imbedm(lt1,lg1,map1,map,maplegend);
```

3

Displaying Maps

Introduction to Mapping Graphics 3-2

Map Displays Made Easy 3-3

Map Axes 3-5

Accessing and Manipulating Map Axes Properties . . . 3-8

The Map Frame 3-13

The Map Grid 3-17

Map and Frame Limits 3-21

Switching Between Projections 3-22

Projected and Unprojected Objects 3-25

Displaying Vector Maps as Lines 3-28

Displaying Vector Maps as Patches 3-31

Displaying Matrix Maps 3-34

The Geographic Data Structure 3-49

Interacting with Displayed Maps 3-53

Specialized Map Functions 3-58

Introduction to Mapping Graphics

One of the fundamental concepts underlying the Mapping Toolbox is the idea that maps are variables that exist independently of their display. However, for visual geographic data analysis, application development, and presentation of results, quality display capabilities are a must.

With the Mapping Toolbox, you can display geographic information almost as easily as you can plot regular data in MATLAB. Most mapping commands are similar to MATLAB commands, except they deal with latitude and longitude spherical coordinates instead of x and y Cartesian coordinates. Mapping commands typically have the same names as their MATLAB counterparts, with the addition of an `'m'`, for maps, at the end. For example, the Mapping Toolbox analog to MATLAB's `plot` command is `plotm`.

The Mapping Toolbox manages most of the steps in displaying a map. It will project your data, cut and trim it to desired limits, and display the resulting map for you. The toolbox also provides the means to add other elements to your displayed map, such as a frame, gridlines, coordinate labels, and text labels. If you change your projection properties, or even the projection itself, the Mapping Toolbox will redraw the map with the new settings, undoing any cuts or trims if necessary. See Chapter 5, "Map Projections" for information on how to project data without displaying it.

The toolbox also makes it easy to modify and manipulate maps. The map display and mapped objects can be modified either from the command line or through graphical user interfaces and property editing tools that can be invoked by clicking on the display. Most mapping display commands have graphical interfaces. See Chapter 7, "GUI Tools" for more on these capabilities.

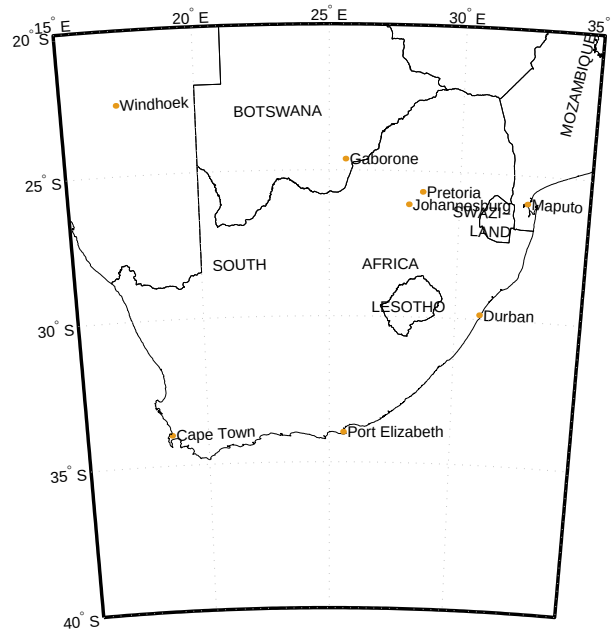
Note An important restriction accompanies this ease of use. The Mapping Toolbox manages the map display with the `UserData` property field in the Axes control. The `UserData` property of mapped objects is also used by the toolbox. This may cause conflicts with other functions that use the `UserData` property field.

Map Displays Made Easy

The Mapping Toolbox allows you to control many aspects of a map display. The majority of this chapter is devoted to the elements of a map display and the means to control them. However, the ability to control numerous properties is accompanied by the need to set them appropriately. You can trade control for simplicity by using the `worldmap` and `usamap` functions. These functions create maps of regions of the world or the United States using appropriate map projections and settings. You can then modify or add to the map displays using the methods described later in this chapter. You can also use these functions to prepare a map display without any atlas data, adding your own instead.

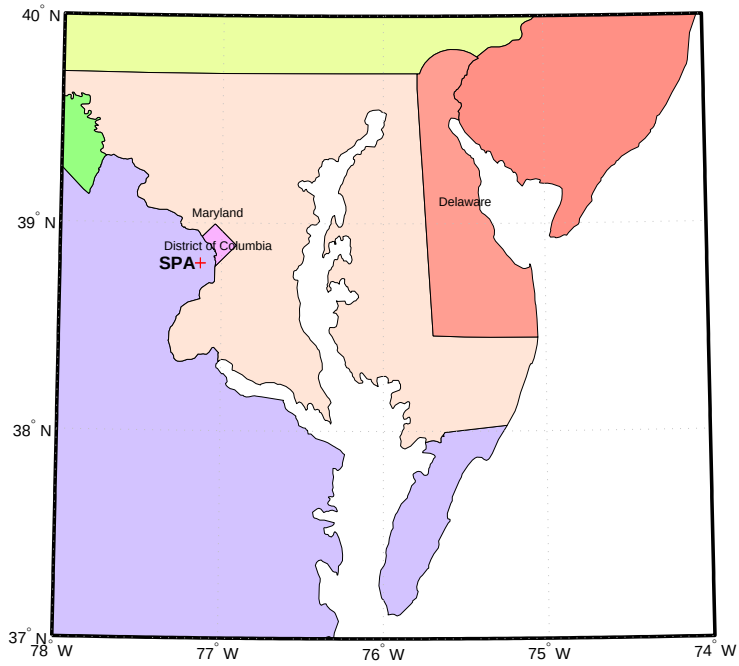
Here are some examples. Create a base map of South Africa.

```
figure
worldmap 'south africa'
hidem(gca)
```



Create a map of the Chesapeake Bay using geographic limits, and add some annotation.

```
latlim = [37 40]; lonlim = [-78 -74];
figure

usamap(latlim,lonlim)

plotm(38.8063,-77.1312,'r+')
textm(38.8063,-77.1312,' SPA ',...
'fontweight','bold','HorizontalAlignment','r')
```



Map Axes

To display maps of any kind in the Mapping Toolbox, the user must first define a map axes. A *map axes* is a normal axes that has been prepared for mapping using the `axesm` command that defines the map projection properties. Before you execute any mapping command, you need to define a map axes in the current MATLAB axes and choose a map projection using `axesm`.

Note To successfully employ any map display function, such as `plotm`, `fillm`, or `surfacem`, the current axes must be a map axes as created by `axesm`. An axes is considered a valid map axes if its `UserData` property contains a structure defining a map projection.

A *projection* is the systematic means of representing three-dimensional geographic data in two dimensions, a topic that is covered more extensively in Chapter 5, “Map Projections”. Here, it is sufficient to understand that many different projections are available, each with different features. Some projections maintain correct relative area between objects, for example, or correct angular relationships. Some are well-suited to displaying features at certain latitudes or with certain orientations. The command `maps` displays a list of available projections.

When creating a map axes object, you must define its projection by using an appropriate ID string. All other properties can be set by default. For example, the following are all valid commands for creating a Miller projection map axes:

```
axesm miller
axesm('miller')
axesm('MapProjection','miller')
```

Other properties can be set for the map axes during creation as well:

```
axesm('MapProjection','mercator',...
      'Frame','on',...
      'AngleUnits','degrees',...
      'FontName','courier',...
      'parallellabel','on',...
      'Origin',[0 180 0])
```

If you specify some map axes parameters with `axesm`, the Mapping Toolbox determines appropriate values for other related map axes properties. As an example, create a map axes with an Equal-Area Conic projection for part of the world:

```
axesm('MapProjection','eqaconic',...
      MapLatLimit',[30 60],...
      MapLonLimit',[-15 45],...
      MapParallels',[])
```

The `axesm` function will automatically set the origin and frame limits for a map centered on the limited geographic region, with the results of subsequent mapping display commands shown only within the specified limits. Empty matrices are used to force recalculation of a property or a return to the default value. For example, setting the map parallels to an empty matrix will automatically determine map parallels appropriate for the current map limits, minimizing distortion in the projection.

Since the `MapProjection` property is the only property that must be explicitly set, it can be specified without the property name, as long as the projection name is the first argument in `axesm`. For instance:

```
axesm('miller','PropertyName1',PropertyValue1,...
      'PropertyName2',PropertyValue2,...)
```

The `axesm` command without any arguments activates a GUI that allows for easy manipulation of the map axes properties.

For proper map appearance and ease of use, the Mapping Toolbox changes some of the default MATLAB axes properties. The altered properties are:

- The `NextPlot` property is set to `'add'`.
- The `UserData` property contains the map projection structure that defines the specific map axes properties.
- The `Box` property is set to `'on'`.
- The `ButtonDownFcn` property is set to the function `uimaptbx`, which processes callbacks for Mapping Toolbox objects.
- The command `showaxes off` is executed to remove axes ticks from the MATLAB Cartesian axes (i.e., the `XTick`, `YTick`, and `ZTick` properties are set to empty matrices).

More information on MATLAB axes properties can be found in the “Axes” section of *Using MATLAB Graphics*. The map axes properties are defined under the `axesm` reference page in the online Mapping Toolbox reference documentation, while information on the use of the `axesm` GUI can be found in the “GUI Reference” chapter in the online Mapping Toolbox reference documentation.

Accessing and Manipulating Map Axes Properties

Just as the properties of the underlying standard axes can be accessed and manipulated using the MATLAB commands `set` and `get`, map axes properties can also be accessed and manipulated using the commands `setm` and `getm`.

Note Use the `axesm` command to *create* a map axes. Use the `setm` command to *modify* the map axes.

As an example, create a map axes:

```
axesm('MapProjection','miller','Frame','on')
```

Suppose you would like to make the frame lines bordering the map thicker. First you need to determine the current line width of the frame. Access the property value by typing:

```
getm(gca,'FLineWidth')
ans =
   2
```

Try resetting the line width to four points:

```
setm(gca,'FLineWidth',4)
```

You can set any number of properties simultaneously with `setm`. Reduce the line width, change the projection to an Equidistant Cylindrical, and verify the changes:

```
setm(gca,'FLineWidth',3,'MapProjection','eqdcylin')

getm(gca,'FLineWidth')
ans =
   3

getm(gca,'MapProjection')
ans =
eqdcylin
```

To inspect the entire set of map axes properties at their current settings, use the following command:

```
getm(gca)
ans =
     mapprojection: 'eqdcylin'
              zone: []
        angleunits: 'degrees'
            aspect: 'normal'
      falseeasting: []
     falsenorthing: []
       fixedorient: []
             geoid: [1 0]
       maplatlimit: [-90 90]
       maplonlimit: [-180 180]
      mapparallels: 30
        nparallels: 1
            origin: [0 0 0]
       scalefactor: []
           trimlat: [-90 90]
           trimlon: [-180 180]
             frame: 'on'
             ffill: 100
        fedgecolor: [0 0 0]
        ffacecolor: 'none'
         flatlimit: [-90 90]
        flinewidth: 3
         flonlimit: [-180 180]
              grid: 'off'
         galtitude: Inf
            gcolor: [0 0 0]
        glinestyle: ':'
        glinewidth: 0.5000
    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'
     meridianlabel: 'off'
    mlabellocation: 30
    mlabelparallel: 90
       mlabelround: 0
     parallellabel: 'off'
    plabellocation: 15
    plabelmeridian: -180
       plabelround: 0
```

Similarly, the possible property values and their defaults can be displayed with the `setm` command alone:

```
setm(gca)

AngleUnits[ {degrees} | radians | dms | dm ]
Aspect[ {normal} | transverse ]
FalseEasting
FalseNorthing
FixedOrientFixedOrient is a read-only property
Geoid
MapLatLimit
MapLonLimit
MapParallels
MapProjection
NParallelsNParallels is a read-only property
Origin
ScaleFactor
TrimLatTrimLat is a read-only property
TrimLonTrimLon is a read-only property
Zone
Frame[ on | {off} ]
FEdgeColor
FFaceColor
FFill
```

```
FLatLimit
FLineWidth
FLonLimit
Grid[ on | {off} ]
GAltitude
GColor
GLineStyle[ - | -- | -. | {:} ]
GLineWidth
MLineException
MLineFill
MLineLimit
MLineLocation
MLineVisible[ {on} | off ]
PLineException
PLineFill
PLineLimit
PLineLocation
PLineVisible[ {on} | off ]
FontAngle[ {normal} | italic | oblique ]
FontColor
FontName
FontSize
FontUnits[ inches | centimeters | normalized | {points} | pixels ]
FontWeight[ {normal} | bold ]
LabelFormat[ {compass} | signed | none ]
LabelUnits[ {degrees} | radians | dms | dm ]
MeridianLabel[ on | {off} ]
MLabelLocation
MLabelParallel
MLabelRound
ParallelLabel[ on | {off} ]
PLabelLocation
PLabelMeridian
PLabelRound
```

Properties can also be displayed individually:

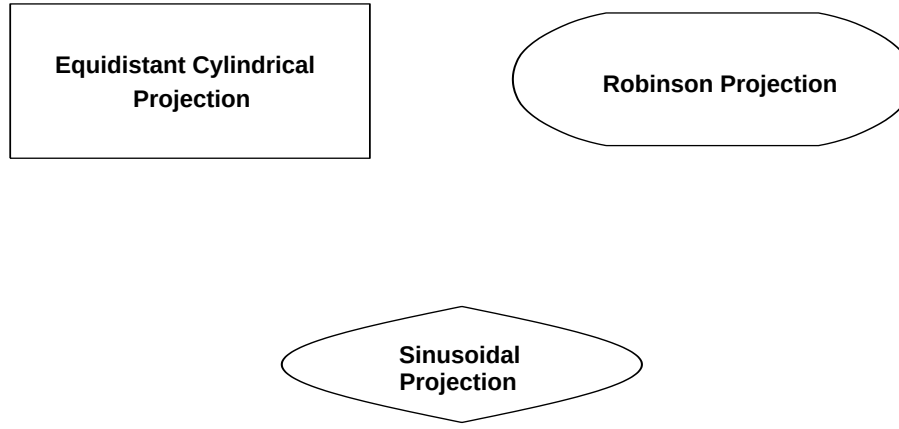
```
setm(gca,'AngleUnits')
AngleUnits[ {degrees} | radians | dms | dm ]

setm(gca,'MapProjection')
```

```
An axes's "MapProjection" property does not have a fixed set
   of property values.

setm(gca,'Frame')
Frame[ on | {off} ]

setm(gca,'FixedOrient')
FixedOrientFixedOrient is a read-only property
```

The Map Frame

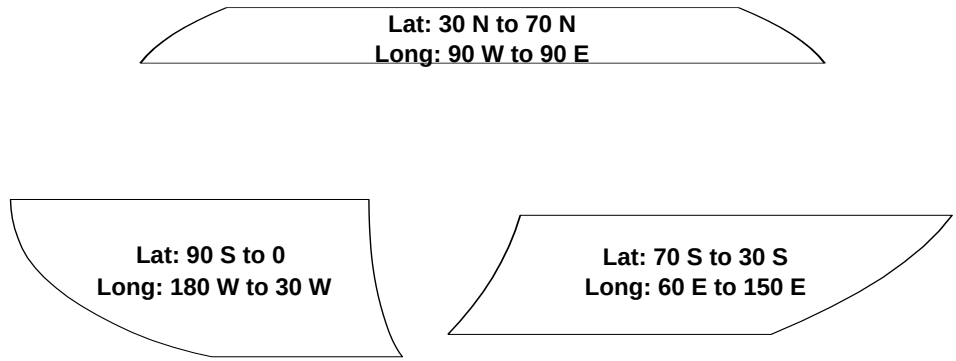
In the Mapping Toolbox, the *map frame* is a *box* enclosing the geographic display. The frame is displayed if the map axes property `Frame` is set to `'on'`. This can be accomplished upon map axes creation with `axesm`, with `setm`, or with the direct command `framem on`. The frame is geographically defined as a latitude-longitude quadrangle that is projected appropriately. For example, on a map of the world, the frame might extend from pole to pole and a full 360 in longitude. In appearance, the frame would take on the characteristic shape of the projection. The examples below are full-world frames shown in three very different projections:

Full-World Map Frames



As a map object, each of the previously displayed frames is identical; however, the selection of a display projection has varied their appearance. Since each of the examples shows the entire world, `FLatLimit` is `[-90 90]`, and `FLonLimit` is `[-180 180]` for each case. The frame quadrangle can encompass smaller regions, as well:

Frame Quadrangles Shown in the Robinson Projection (Symmetric about Prime Meridian)



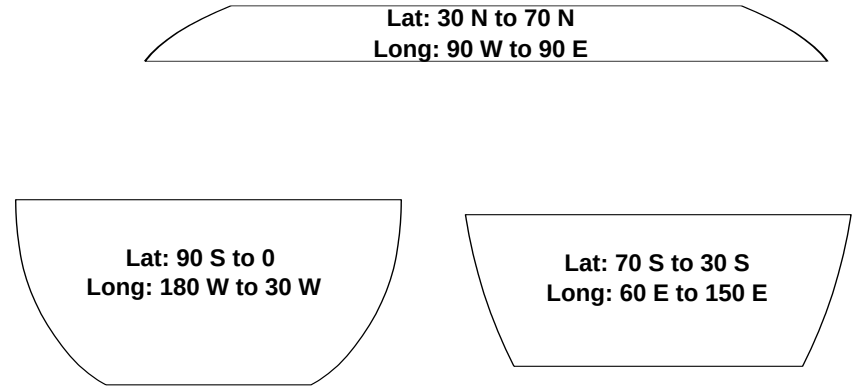
For the frames shown above, the projection is centered on the Prime Meridian, or 0 longitude. Such a frame would be the result of creating a map axes with the defaults for the Robinson projection and then resetting the frame limits to cover just part of the world.

For example, to view the asymmetric frame in the lower right of the previous figure, type:

```
axesm robinson
setm(gca,'FLatLimit',[-70 -30],...
     'FLonLimit',[60 150],...
     'Frame','on')
```

When you want your frame to be symmetric about the region of interest, have `axesm` determine the proper settings for you. If you specify the map limits without specifying the map origin and frame limits, `axesm` will automatically set the appropriate values for a proper symmetric frame.

Frame Quadrangles Shown in the Robinson Projection (Symmetric about Map Limits)



For example, to view the symmetric frame in the lower right of the previous figure, set the map limits with `axesm`:

```
axesm('MapProjection','robinson',...
     'MapLatLimit',[-70 -30],...
     'MapLonLimit',[60 150],...
     'Frame','on')
```

In addition to the latitude and longitude limits of the frame, other properties can be manipulated. The frame is actually a patch with a default face color set to `'none'` and a default edge color of black. These map axes properties can be altered by manipulating the `FFaceColor` and `FEdgeColor` properties. For example, the command:

```
setm(gca,'FFaceColor','cyan')
```

can make the background region of your display resemble water. Since the frame patch is always the lowest layer of a map display, other patches, perhaps representing land, will appear above the “water.” If an object is subsequently plotted “below” the frame patch, the frame altitude can be recalculated to lie below this object with the command `framem reset`. The frame is replaced and not reprojected.

The line width of the edge, which is 2 points by default, can be set using the `FLineWidth` property.

The primary advantage of displaying the map frame is that it can provide positional context for other displayed map objects. For example, when vector data of the coasts is displayed, the frame provides the “edge” of the world.

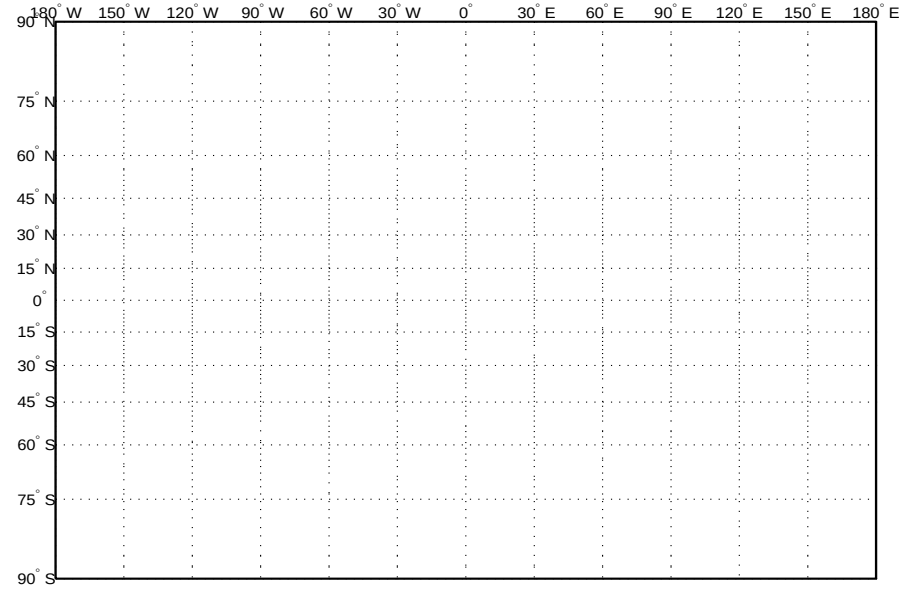
In addition to establishing frame properties upon map axes creation, or through the `setm` command, you can use the `framem` command. The command `framem` alone is a toggle for the `Frame` property, which controls the visibility of the frame. You can also call `framem` with property names and values to alter the appearance of the frame:

```
framem('FlineWidth',4,'FEdgeColor','red')
```

The Map Grid

In The Mapping Toolbox, the *map grid* is the set of displayed meridians and parallels. Display the grid by setting the map axes property `Grid` to `'on'`. This can be accomplished upon map axes creation with `axesm`, with `setm`, or with the direct command `gridm on`.

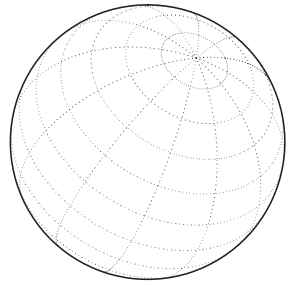
To control display of meridians and parallels, set a scalar meridian spacing or a vector of desired meridians in the `MLineLocation` property. The corresponding property `PLineLocation` serves the same purpose for parallels. The default values are every 30 for meridians and every 15 for parallels.



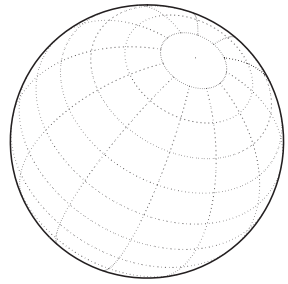
By default, the grid is placed as the top layer of any display. You can alter this by changing the `GAltitude` property, so that other map objects can be placed "above" the grid. The new grid is drawn at its new altitude.

To reposition the grid back to the top of the display, use the command `gridm reset`. You can also control the appearance of grid lines with the `GLineStyle` and `GLineWidth` properties, which are ':' and `0.5`, respectively, by default.

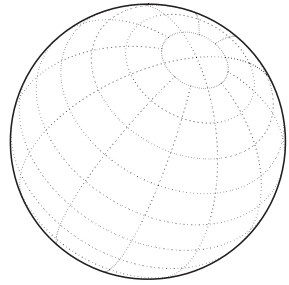
The Miller projection is an example in which all the meridians can extend to the poles without appearing to be cluttered. Look, however, at the following Orthographic projections. When all the meridians extend to the poles, the look is “congested.” The `MLineLimit` property allows you to specify a pair of latitudes at which to terminate the meridians. For example, setting `MLineLimit` to `[-75 75]` completely clears the polar region of displayed meridians. Sometimes you might like to allow some of the meridians to ignore the limit and extend to the pole. If, for example, `MLineException` is set to `[-90 0 90 180]`, then the meridians corresponding to those four longitudes will continue past the limit on to the pole:



Default grid allows all displayed meridians to extend to the poles.



The property `MLineLimit` can truncate meridians at a given latitude, here at 75 N and S.



The property MLineException allows certain meridians to extend to the poles despite the `MLineLimit`. Here, four meridians 90 W, 0 , 90 E, and 180 are excepted.

There are also corresponding map axes properties for controlling the extent of displayed parallels: `PLineLimit` and `PLineException`.

Users can label displayed parallels and meridians. `MeridianLabel` and `ParallelLabel` are on-off properties for the display of labels on the meridians and parallels, respectively. They are both `'off'` by default. Initially, the label locations coincide with the default displayed grid lines, but you can alter this by using the `PLabelLocation` and `MLabelLocation` properties. These grid lines are labeled across the north edge of the map for meridians and along the west edge of the map for parallels. However, the property `MLabelParallel` allows

you to specify `'north'`, `'south'`, `'equator'`, or a specific latitude at which the meridian labels will be displayed, and the `PLabelMeridian` allows the choice of `'west'`, `'east'`, `'prime'`, or a specific longitude for the parallel labels. For complete descriptions of all map axes properties, refer to the `axesm` reference page in the online Mapping Toolbox reference documentation.

Map and Frame Limits

In the Mapping Toolbox, the map and frame limits are two related map axes properties that limit the map display to a defined region. The map latitude and longitude limits restrict the displayed parallels and meridians, while the frame limits control the extent of the frame around the displayed data. Any object that extends outside the frame limits is automatically trimmed.

The frame limits are also specified differently from the map limits. The map limits are in Greenwich coordinates referenced to an origin at the intersection of the Prime Meridian and the Equator, while the frame limits are in coordinates referenced to the center of the frame, which is the latitude and longitude of the map axes origin.

For all non-azimuthal projections, frame limits are specified as quadrangles ([*latmin latmax*] and [*longmin longmax*]) in the frame coordinate system. In the case of azimuthal projections, the frames are circular and are described by a polar coordinate system. One of the frame latitude limits must be a negative infinity (-Inf) to indicate an azimuthal frame (think of this as the center of the circle), while the other limit determines the radius of the circular frame (*rlatmax*). The longitude limits of azimuthal frames are inconsequential, since a full circle is always displayed.

If you are uncertain about the correct format for a particular projection frame limit, you can reset them to the default values using empty matrices.

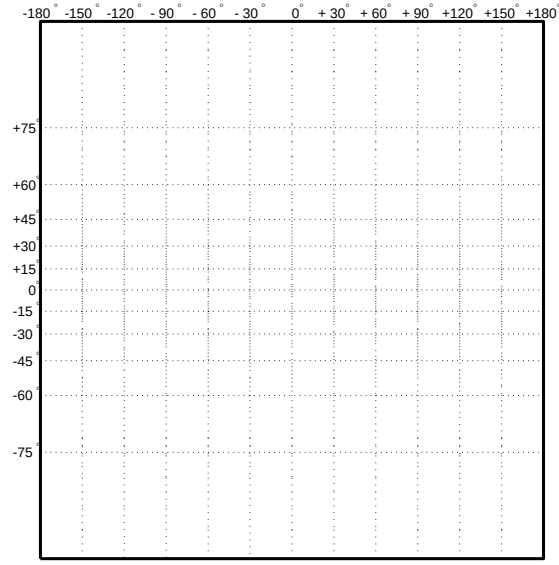
Note For non-azimuthal projections in the normal aspect, the map is limited to the minimum of the map and frame limits; hence, the two limits will coincide after evaluation. When changing one set of limits, you may want to clear the other set to get consistent limits.

Switching Between Projections

When switching between projections using `setm` or the graphical interfaces, you may need to change some of the map axes properties for proper appearance. Settings that are suitable for one projection may not be appropriate for another. Some projections have default properties that define *that* particular projection and may not be altered; for example, the Balthasart Cylindrical projection is defined to have standard parallels (`MapParallels`) at 50 . Other projections have default properties that are initially set for proper world display; for example, the Mercator projection limits the latitude range to 86 to avoid “blowing up” at the poles.

Although similar projections may share the same set of properties (Miller Cylindrical and Plate Carree Cylindrical), others may be drastically different (Polyconic and Stereographic azimuthal). The classification of the map projections is often a good indicator of whether changes need to be made. For instance, switching from a cylindrical to an azimuthal projection requires a few modifications:

```
axesm mercator
framem on; gridm on; mlabel on; plabel on
setm(gca,'LabelFormat','signed')
```



To display the default map and frame latitude limits for the Mercator projection, type the following:

```
[getm(gca,'MapLatLimit'); getm(gca,'FLatLimit')]
ans =
      -86   86
      -86   86
```

Both the frame and map latitude limits are set to 86 for the Mercator projection to maintain a safe distance from the singularity at the poles.

Now switch the projection to an Orthographic azimuthal:

```
setm(gca,'MapProjection','ortho')
```

What happened to the map frame and labels? If you recall, the frame latitude limits have not been changed and still correspond to the default values for a Mercator projection, as do all the other properties. Only those properties that *must* have particular values are updated for the current projection.

```
getm(gca,'FLatLimit')
ans =
      -86   86
```

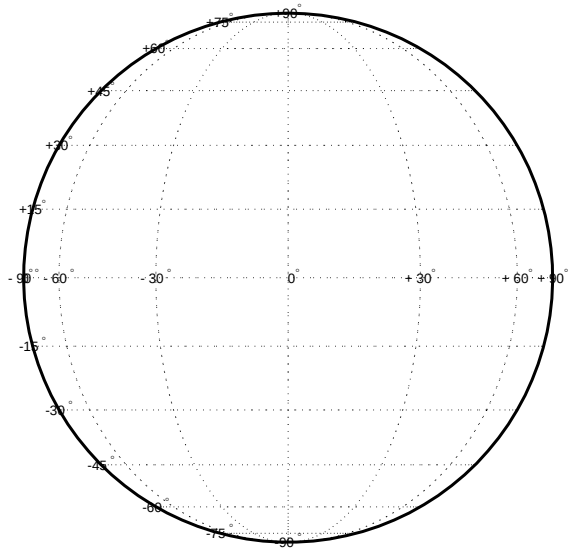
You must manually reset the frame and map limits to appropriate values for an Orthographic projection so that the circular frame is displayed. If you don't know the default or appropriate values, provide an empty matrix for any of the property values:

```
setm(gca,'FLatLimit',[],'MapLatLimit',[])
[getm(gca,'MapLatLimit'); getm(gca,'FLatLimit')]
ans =
      -90        90
      -Inf       89
```

You will also need to manually specify the locations of the meridian and parallel labels:

```
setm(gca,'MLabelParallel',0,'PLabelMeridian',-90)
```

Now the map is displayed correctly, with the frame:



Default property values can be accessed by the use of empty matrices, as shown in the last example, and the properties can be reset according to the “new” values. The entire set of properties can be reset to their default values by pressing the **Reset** button on the `axesmui` GUI.

For complete descriptions of all map axes properties, consult the `axesm` reference page in the “Mapping Reference” chapter of the online Mapping Toolbox reference documentation. For more information on the use of `axesmui`, also refer to its reference page in the “GUI Reference” chapter of the Mapping Toolbox online reference documentation.

Projected and Unprojected Objects

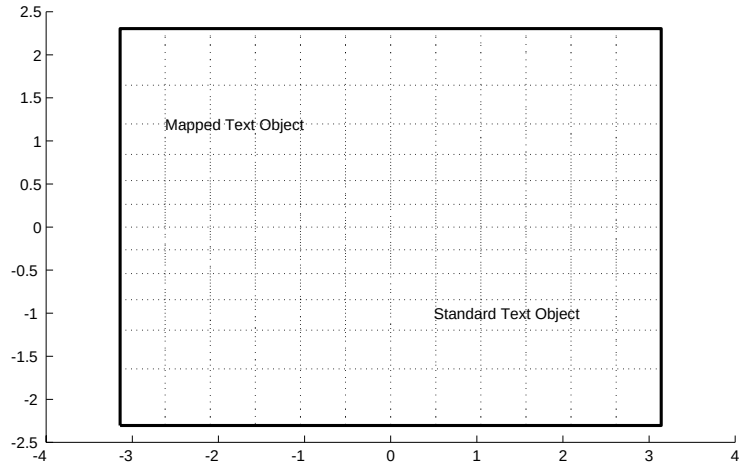
All objects displayed using Mapping Toolbox commands are projected on the map axes based on their designated latitude-longitude positions. The latitudes and longitudes are mathematically transformed to x and y positions using the formulas for the current map projection. Objects displayed using standard MATLAB commands (like `plot`, `fill`, `surface`, and `text`) are placed in the designated *x-y* location. If the projection changes for a map axes (for example, by using the `setm` command to alter the `MapProjection` property), those objects displayed with standard commands are unaffected, but those displayed using mapping commands are reprojected into new positions. Here is an example:

```
axesm miller; framem on; gridm on;
showaxes; grid off;
```

This set of commands creates a map axes, a map frame enclosing the region of interest, and geographic grid lines. The x-y axes, which are normally hidden, are displayed, and the MATLAB x-y grid is turned off. It should be noted that the MATLAB x-y `grid` is separate from the Mapping Toolbox geographic version `gridm`.

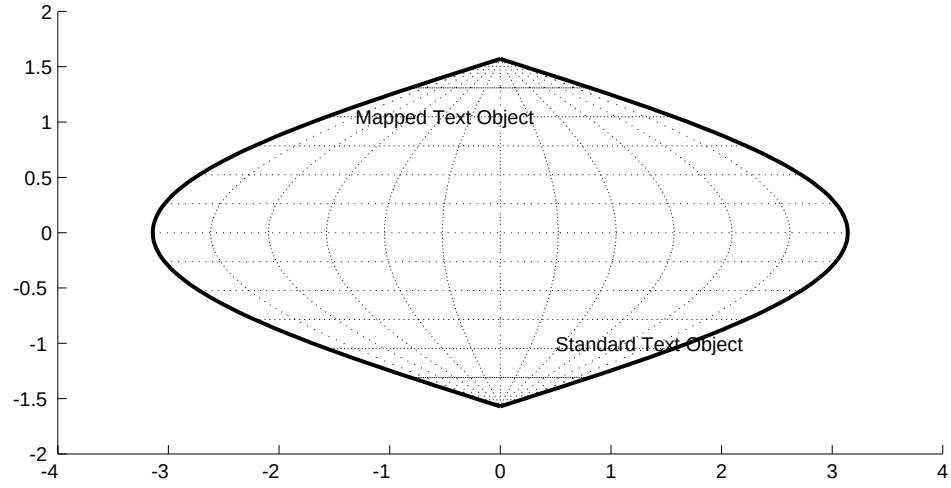
```
text(.5,-1,'Standard Text Object')
textm(60,-150,'Mapped Text Object')
```

Here, a standard text object is placed at x=0.5 and y=-1, while a mapped text object has been placed at (60 N,150 W) in the Miller projection.



If we change the projection to a Sinusoidal, the standard text object remains at the same Cartesian position, so its latitude-longitude position is altered. The mapped text object remains at the same geographic location, so its x-y position is altered. Also, the frame and gridlines are replaced according to the new map projection:

```
setm(gca,'MapProjection','sinusoid')
showaxes; grid off;
```



Similarly, vector and matrix data can be displayed using either mapping or standard commands.

Displaying Vector Maps as Lines

The Mapping Toolbox lets you display vector map data as line objects much like the line display commands in MATLAB. The Mapping Toolbox line graphics functions have MATLAB analogs, the names of which can usually be determined by appending an `'m'` to the MATLAB function name. For instance, the Mapping Toolbox version of `plot` is `plotm`.

The following table lists the available Mapping Toolbox line display functions:

Function	Used to Create
`contorm`	Contour plot of map data
`contor3m`	Contour plot of map data in 3-D space
`linem`	Line objects projected on map axes
`plotm`	Lines projected on map axes
`plot3m`	Lines projected on map axes in 3-D space

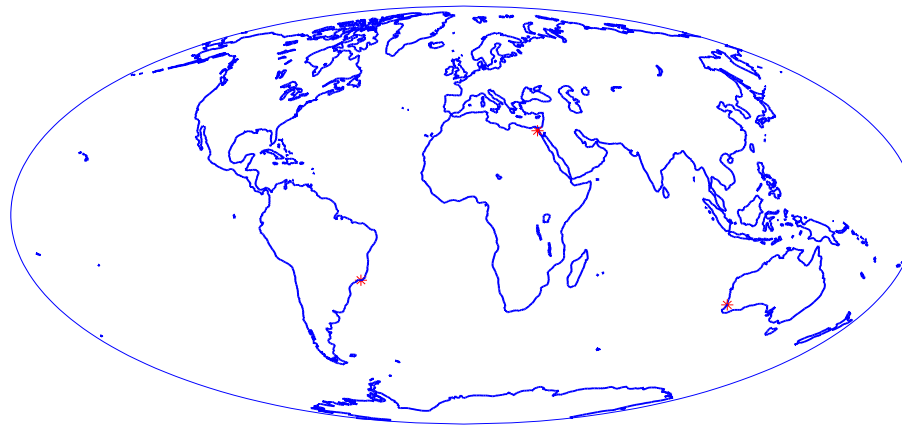
Plot the `coast` vector data using `plotm`. Just as with `plot`, you can specify line property names and values in the command:

```
load coast
axesm mollweid
framem('FEdgeColor','blue','FLineWidth',0.5)
plotm(lat,long,'LineWidth',1,'Color','blue')
```

Sometimes vector data represent specific points. Suppose you have variables representing the locations of Cairo (30 N,32 E), Rio de Janeiro (23 S,43 W), and Perth (32 S,116 E), and you want to plot them as markers only, without connecting line segments.

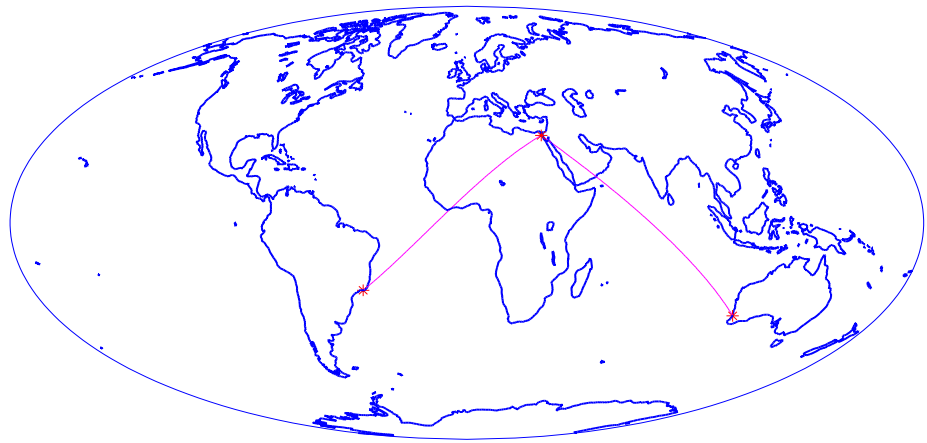
Here is the map:

```
citylats = [30 -23 -32]; citylongs = [32 -43 116];
plotm(citylats,citylongs,'r*')
```



In addition to these sorts of “permanent” geographic data, you can also display calculated vector data. Calculate and plot a great circle track from Cairo to Rio de Janeiro, and a rhumb line track from Cairo to Perth:

```
[gclat,gclong] = track2('gc',citylats(1),citylongs(1),...
                            citylats(2),citylongs(2));
[rhlat,rhlong] = track2('rh',citylats(1),citylongs(1),...
                            citylats(3),citylongs(3));
plotm(gclat,gclong,'m-'); plotm(rhlat,rhlong,'m-')
```



Displaying Vector Maps as Patches

Vector map data that is properly formatted can be displayed as a patch, or filled-in polygon. Like many Mapping Toolbox functions, the names of the patch map display functions can be determined by appending an `'m'` to its MATLAB counterpart (e.g., add an `'m'` to `patch` to get `patchm`).

Note The Mapping Toolbox patch display commands differ from their MATLAB equivalents by allowing you to display patch vector data that use `NAN`s to separate faces.

The following table lists the available Mapping Toolbox patch display functions:

Function	Used to Create
`fillm`	Filled 2-D map polygons
`fill3m`	Filled 3-D map polygons in 3-D space
`patchm`	Patch objects projected on map axes
`patchesm`	Patches projected as individual objects on map axes

The `usalo` MAT-file is useful for illustrating patches:

```
load usalo
who
Your variables are:

conus          gtlakelon      statelat       uslon
greatlakes     state          statelon
gtlakelat      stateborder    uslat
```

The variables `uslat` and `uslon` together describe three polygons (separated by `NaN`s) representing the continental United States. The smaller polygons are necessary to display Long Island and Martha's Vineyard. The variables `gtlakelat` and `gtlakelon` describe three polygons (separated by `NaN`s) for the

Great Lakes. The variables `statelat` and `statelon` contain line-segment data (separated by `NaN`s) for the borders between states, which is not formatted for patch display.

Conic projections make good displays of the United States, so create a map axes using an Albers Equal-Area Conic projection. By specifying map limits that contain the region of interest, you automatically center the projection on an appropriate longitude, and the frame encloses just the mapping area, not the entire globe.

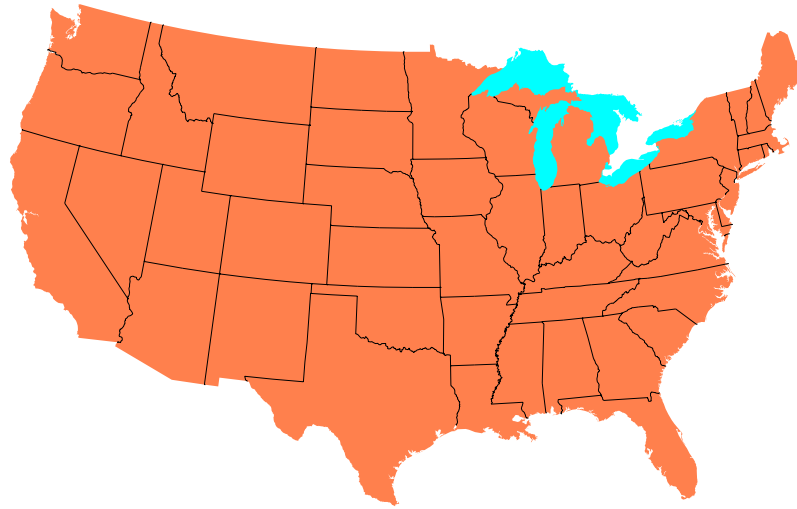
A good rule-of-thumb for conic projections is to set the standard parallels at one-sixth of the distance from both latitude limits. You can do this automatically by providing an empty matrix as the standard parallel:

```
axesm('MapProjection','eqaconic','MapParallels',[],...
      'MapLatLimit',[23 52],'MapLonLimit',[-130 -62])
```

When patch data is displayed, layering can become important as objects above can hide objects below. You can control the visibility of objects by the order of their display and by their “altitude” position. For this reason, the primary patch display function in the Mapping Toolbox, `fillm`, allows you to control the z-axis level of displayed patches. Here the land data is displayed at the default level, z=0. The Great Lakes data is assigned the altitude of z=1. To plot the line segment data between these layers, use the `plot3m` command and specify an altitude of 0.5:

```
fillm(uslat,uslon,'FaceColor',[1 .5 .3],'EdgeColor','none')
fillm(gtlakelat,gtlakelon,1,...
      'FaceColor','cyan','EdgeColor','none')
plot3m(statelat,statelon,0.5,'k')
```

Here is the result:



The `fillm` function makes use of the low-level function `patchm`. The Mapping Toolbox provides another patch drawing function called `patchesm`. The optimal use of either depends on the application and user preferences. The `patchm` function creates one displayed object and returns one handle for a patch, which may contain multiple, not necessarily connecting, faces. The Mapping Toolbox uses `NaN`s to separate non-connected patch faces, unlike MATLAB, which does not handle `NaN` clipped data for patches. For the `patchesm` function, each face is a separate object with its own handle. In general, `patchm` requires more memory but is faster than `patchesm`.

Displaying Matrix Maps

The Mapping Toolbox provides functions for the display and enhancement of both regular and general matrix maps in a variety of formats. Recall that regular matrix maps require a map legend vector that describes the scale and location of the map, while general matrix maps require matrices of latitude and longitude coordinates.

The matrix map display functions are geographic analogies to the MATLAB surface drawing commands. Like the line plotting functions discussed in the previous chapter, the Mapping Toolbox function names are usually identical to their MATLAB counterparts, except for the addition of an 'm.' For instance, the Mapping Toolbox functions, `surfacem` and `surfm`, are analogous to MATLAB's `surface` and `surf` commands. You will most likely find a Mapping Toolbox version of a MATLAB surface drawing function by appending an 'm' to the MATLAB function name.

Note In the Mapping Toolbox, the set of functions beginning with '`mesh`' are used for regular matrix maps, while those with '`surf`' are reserved for general matrix maps. It should be noted here that this usage differs entirely from MATLAB's definition; that is, `mesh` plots are used for colored, wire-frame views of the surface, while `surf` displays colored, faceted surfaces.

Surface map objects can be displayed in a variety of different ways. You can assign colors from the figure colormap to surfaces according to the value of their data. You can also display images where the matrix data consists of indices into a colormap or display the matrix as a three-dimensional surface, with the z coordinates given by the map matrix. You can use monochrome surfaces that reflect a “pseudo-light” source, thereby producing a three-dimensional, shaded relief model of the surface. Finally, you can use a combination of color and light shading to create a lighted shaded relief map.

The following table lists the available Mapping Toolbox surface map display functions:

Function	Used to Create
`meshm`	Regular matrix map warped to projected graticule mesh
`surfm`	General matrix map projected on map axes
`imagem`	Regular matrix map displayed as image
`pcolorm`	Projected matrix map in z = 0 plane
`surfacem`	Matrix map warped to projected graticule mesh
`surflm`	3-D shaded surface with lighting projected on map axes
`meshlsrm`	3-D lighted shaded relief of regular matrix map
`surflsrm`	3-D lighted shaded relief of general matrix map

The Graticule

The Mapping Toolbox projects surface objects in a manner very similar to the traditional methods of map making. The cartographer traditionally lays out a grid of meridians and parallels, called the *graticule*. Each graticule cell is therefore a geographic quadrangle. The cartographer calculates the appropriate *x-y* locations for every vertex in the graticule grid and draws the projected graticule by "connecting the dots." Finally, the cartographer draws the map data in "freehand," attempting to account for the shape of the graticule cells. Similarly, the Mapping Toolbox calculates the x-y locations of the four vertices of each graticule cell and warps the matrix data to fit the resulting quadilateral.

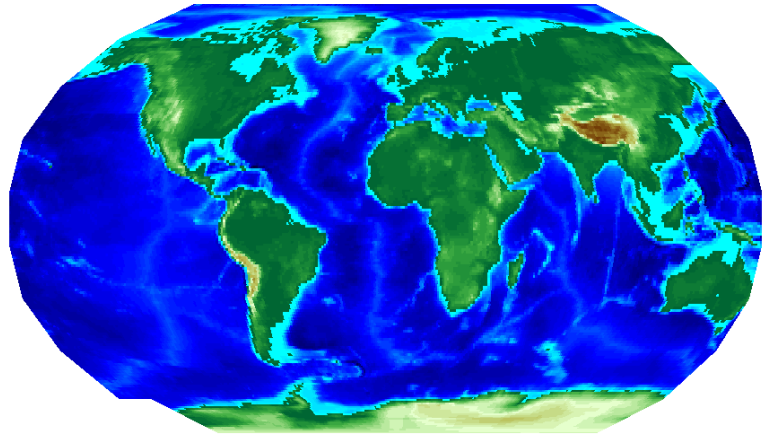
In the toolbox, as in traditional cartography, the finer the graticule mesh (i.e., the more meridians and parallels used), the greater the precision of the projected map display, at the cost of greater effort and time. Graticules for regular matrix maps are defined in the toolbox as a two-element vector, of the form [*number-of-parallels*, *number-of-meridians*]. For general matrix maps, the graticule is related to the size of the latitude and longitude

coordinate matrices, where *number-of-parallels*=*mrows*-1 and *number-of-meridians*=*ncols*-1.

While the graticule cell for a regular matrix map is restricted to equal-angle quadrangles (i.e., length of cell in latitude must equal length of cell in longitude), general matrix maps have no such constraint. Their graticule cells can be of any size.

The `topo` regular matrix map can be displayed quickly using a coarse graticule. The cost is in precision:

```
load topo
figure; axesm robinson
graticule = [10 20];
h = meshm(topo,topolegend,graticule);
demcmap(topo)
```

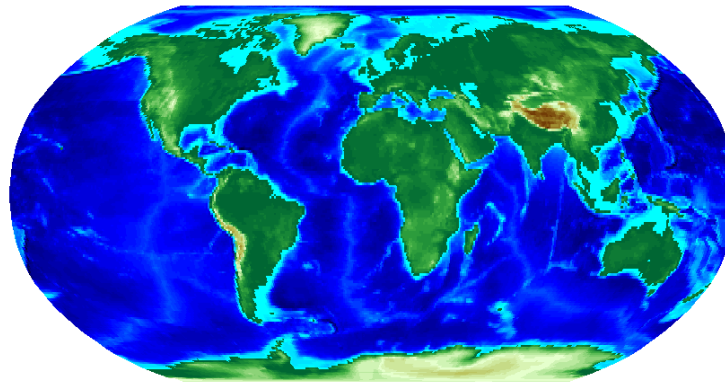


Notice that for this coarse graticule, the edges of the map do not appear as smooth curves. What may not be as obvious is that the easternmost column of graticule cells and the southwesternmost cell are sometimes invisible on displayed matrix maps. This is necessary for the proper projection of the surface object and is not a concern except with the coarsest graticules. Previous displays used the default [50 100] graticule, for which this effect is negligible.

Regardless of the graticule resolution, the matrix map data is unchanged. In this case, the matrix map is the 180-by-360 `topo` matrix, and regardless of projection fidelity, the resolution of its value data is unchanged.

Map objects displayed as surfaces have all the properties of any MATLAB surface, which can be set at object creation or by using the MATLAB `set` command. The mapping `setm` command allows the `MeshGrat` graticule property to be manipulated for regular matrix surfaces. Since you saved the handle of the last displayed map, reset its graticule to a very fine grid. Remember, the trade-off is between resolution and time, so this could take a while:

```
setm(h,'MeshGrat',[200 400])
```



You'll probably notice that the result does not appear to be any better than the original display with the default [50 100] graticule, but it took much longer to produce. Actually, the 200-by-400 graticule grid is finer than the 180-by-360 data resolution. There is really no reason to ever use a graticule finer than your data. In practice, you will probably find that coarse graticules are good for development tasks and fine graticules serve well for final graphics production.

Data Representation

Image and Surface Coloring

The simplest way to display a matrix map is to assign colors to matrix elements according to the value of their data. The map can be displayed either as a 2-D image or a 3-D surface using the map values as the z data.

Consider the matrix map located in the `korea` workspace. It contains a matrix of land elevations and bathymetry data for the region around the Korean peninsula, along with a map legend variable, which indicates a regular matrix

map format. Calculate a graticule for the regular matrix map using the `meshgrat` function so that a general matrix map is also available. Also, convert the units for the map matrix from meters to degrees, so they are consistent with the latitude and longitude coordinate matrices.

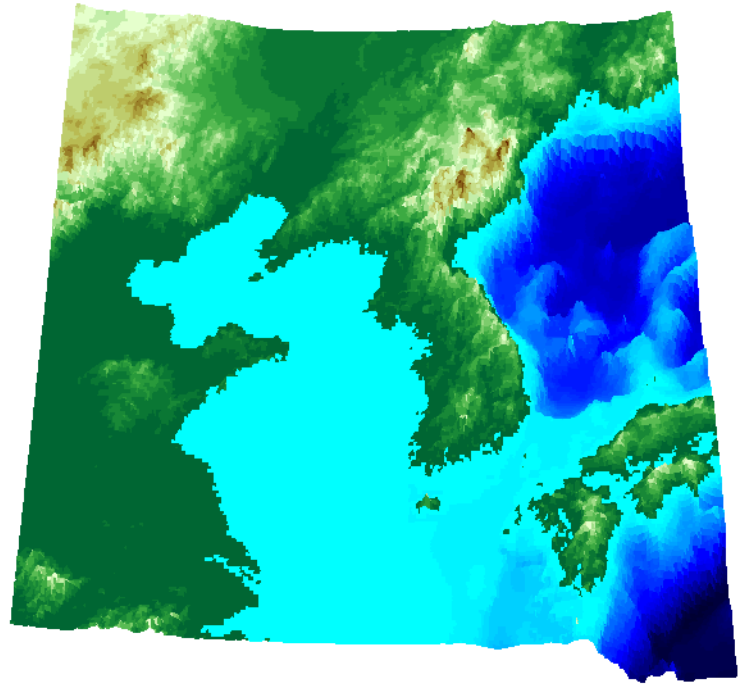
```
load korea
[lat,lon] = meshgrat(map,maplegend);
map = km2deg(map/1000);
whos
   Name       Size       Bytes     Class

   lat        180x240    345600    double array
   lon        180x240    345600    double array
   map        180x240    345600    double array
   maplegend  1x3        24        double array
```

Notice that the `lat` and `lon` coordinate matrices, or graticule, are the same size as the `map` matrix, a requirement for constructing 3-D surfaces.

You have already seen the `meshm` command used to display regular matrix maps, so display the `korea` matrix map using the `surfm` command:

```
axesm('MapProjection','eqaconic','MapParallels',[],...
     'MapLatLimit',[30 45],'MapLonLimit',[115 135])
surfm(lat,lon,map,map); demcmap(map)
set(gca,'DataAspectRatio',[1 1 .5])
view(0,75)
```

Here is the result:

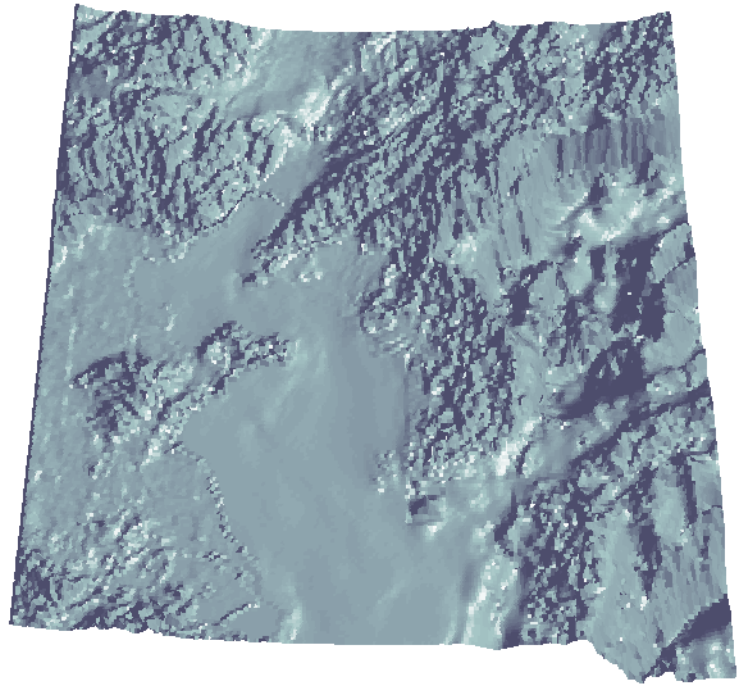


Surface Light Shading

A monochrome three-dimensional, shaded-relief map can be displayed with the function `surflm`, which is analogous to the MATLAB `surfl` command. This functionality is available only for general matrix maps. Remember that regular matrix maps are a subset of general matrix maps and can be easily converted.

To display a map in this format, enter the following:

```
clma
surflm(lat,lon,map); colormap(bone)
```



This is the same region as the previous map, viewed in three dimensions. Notice the dramatic dip in the lower-right portion of the map caused by the deep ocean trenches of the North Pacific Ocean. The view is from the south, at an elevation angle of 75 . The light source in this case is the default 45 counterclockwise from the view direction.

This kind of representation shows much more of the fine structure of the land and sea floor, but because of the lack of color, it is difficult to distinguish land from sea.

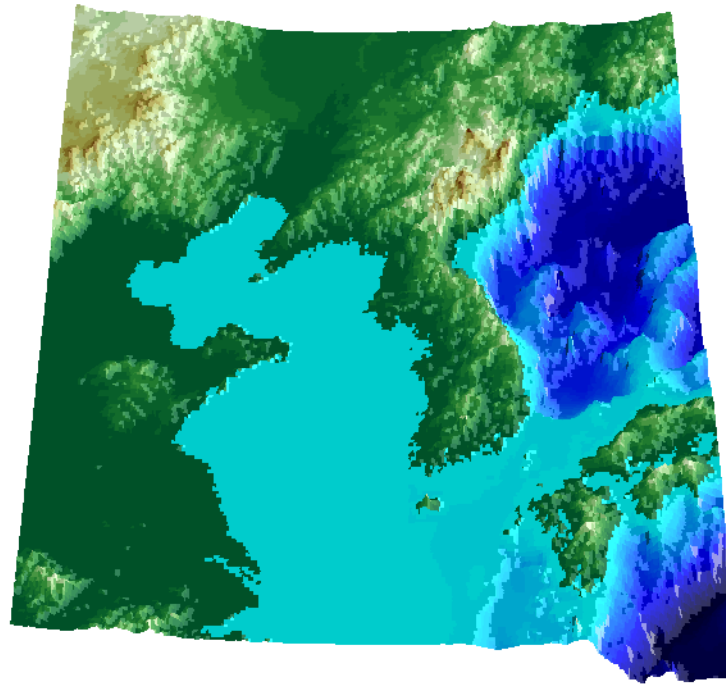
Surface Lighted Shaded Relief

The commands `meshlsrm` and `surflsrm` display maps in a lighted shaded relief format and can be thought of as extensions to `surflm` that combine surface coloring and surface light shading. Again, `meshlsrm` is used for regular matrix maps and `surflsrm` for general matrix maps. Although there are no analogous

MATLAB functions, you can obtain similar results using MATLAB light objects. However, unlike MATLAB lighted surfaces that do not modify the surface's `CData`, these mapping commands construct a new colormap and associated `CData` matrix by using grayscales to lighten or darken a matrix component based on its calculated surface normal to a light source.

To display a lighted shaded relief map, type the following:

```
clma
surflsrm(lat,lon,map)
```

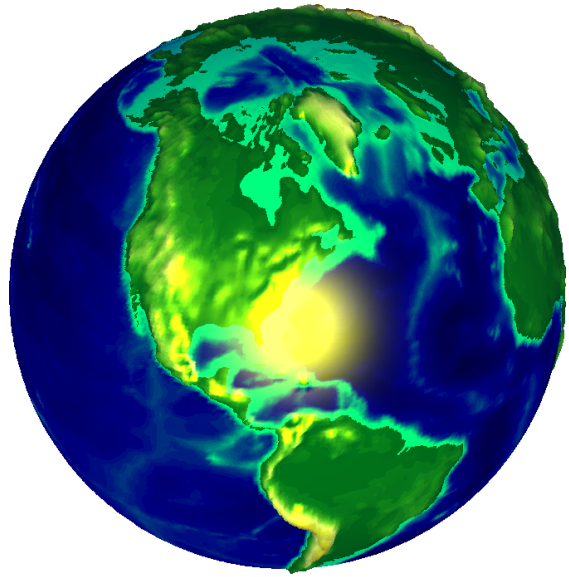


Mapped Light Objects

The Mapping Toolbox allows for the mapping of light objects using the `lightm` command, which is similar to the `light` command in MATLAB. Here, place a

lighted source at an infinite distance, representing the Sun at the summer solstice (23.5 N) at local apparent noon in Natick, Massachusetts (71 W):

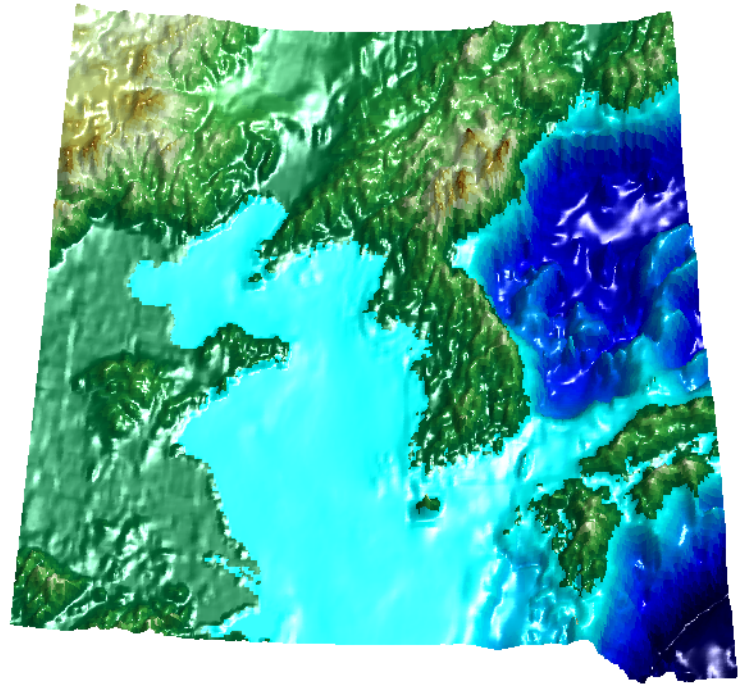
```
load topo
ptopo = topo; ptopo(topo<0) = 0;
ptopo = 100*ptopo/(almanac('earth','radius')*1000);

axesm globe; view([15 38])
meshm(topo,topolegend,size(topo),ptopo); demcmap(topo)
shading interp
lightm(23.5,-71,'Color','y')
material([.5 .7 1.5]); lighting phong
```



The topo surface had been modified to show the 3-D features of land only, which is vertically exaggerated by a factor of 100. The ocean depths are shown by color only.

As a comparison to the lighted shaded relief example shown earlier, add a light source to the surface colored matrix map of the Korean peninsula region shown earlier:

```
figure; load korea
axesm('MapProjection','eqaconic','MapParallels',[],...
     'MapLatLimit',[30 45],'MapLonLimit',[115 135])
meshm(map,maplegend,size(map),map); demcmap(map)
light('Position',[1 0 1])
material([.5 .5 1]); lighting phong
daspectm('meters',125)
view(0,75)
```



The light effects employed here are different and separate from those used by `surfm`, `meshlsrm`, and `surflsrm`. Here, a light object is added directly to the map surface. Like all displayed surfaces, the light object is a separate MATLAB graphic object with its own object handle. For the functions `meshlsrm` and `surflsrm`, light effects are simulated within the functions by modifying the colormap with bands of light and dark. The map matrix is then converted to indices for the new “shaded” colormap based on calculated surface normals.

Using light objects allows for a wide range of different light effects, such as ambient light, light color, and surface reflectance properties.

For more information, consult the reference pages for `meshlsrm`, `surflsrm`, and `lightm` in the online Mapping Toolbox reference documentation, along with the section on lighting in the MATLAB graphics documentation.

Draping Data on Elevation Maps

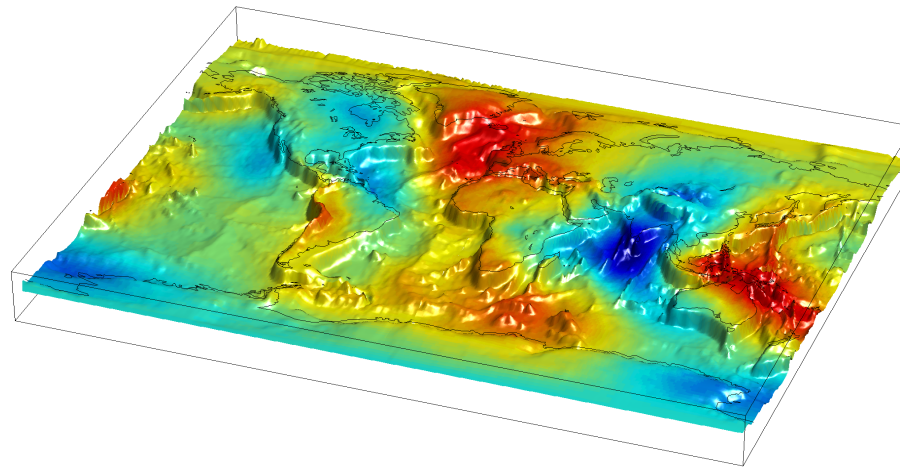
Lighting effects are also helpful when elevation maps are combined with other kinds of data. The shading resulting from lighting effects provides the necessary visual cues when you drape satellite data on a grid of elevations. This kind of display is commonly used to combine topography from digital elevation models with land use images from remote sensing satellites such as LANDSAT and SPOT. Such displays can also be created with the Mapping Toolbox.

If the elevation and color matrices correspond pixel-for-pixel to the same geographic locations, such a display can be created using the optional altitude arguments in the surface display commands. The following example shows the Earth's geoid draped on the corresponding topography. The geoid can be described as the surface of the ocean in the absence of waves, tides, or land obstructions. The geoid is influenced by the gravitational attraction of denser or lighter materials in the Earth's crust and by the shape of the crust. The geoid is also important in surveying because it defines the direction of "up," from which is measured the geodetic latitude. The geoid heights vary from a minimum of about 105 meters below sea-level, shown in blue, to a maximum of about 85 meters above sea-level, shown in red. You can see that the geoid mirrors the topography of the major mountain chains such as the Andes, the Himalayas, and the Mid-Atlantic Ridge. You can also see that large areas of high or low geoid heights are not simply a result of topography.

```
load topo
load geoid
axesm gstereo

meshm(geoid,geoidlegend,size(geoid),topo)

plotm(coast,'k')
zdatam(handlem('allline'),1000) % keep lines above surface
```

```
daspectm('m',200); tightmap
view(20,35)
camlight; lighting phong

set(gca,'projection','perspective') % allow axes to converge
```

Here is the displayed map:

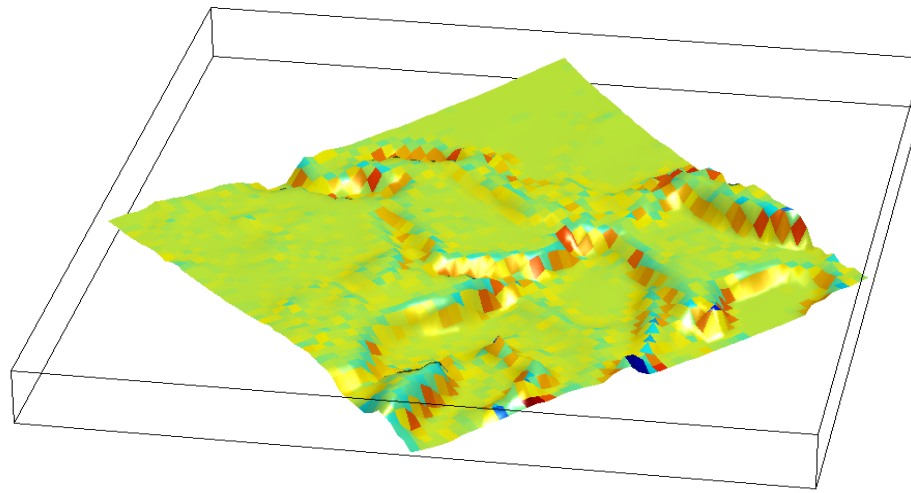


If your elevation and color data is based on different graticules, you must construct a new version of one matrix that is consistent with the other. Suppose you have elevation data that is in a regular matrix map and color data in a general matrix map. One approach is to interpolate elevation values for the general matrix map graticule. For example, if your elevation map is `topo` and `topolegend`, and the color data is the curvature of the diamond-shaped general matrix map in the `mapmtx` MAT-file, `lt1`, `lg1`, and `del2(map1)`, you can create a diamond-shaped map of elevations using `ltln2val`. When you use `ltln2val` for this purpose, it is important that the regular matrix map of elevations completely covers the area of the general matrix map. The four matrices can then be combined in a surface display. Notice how the greatest curvature values, shown in red, are found at the continental margins of Africa and where the Indian plains meet the Himalayas.

```
load topo
```

```
load mapmtx

map1z = ltln2val(topo,topolegend,lt1,lg1);

figure
axesm miller
surfm(lt1,lg1,del2(map1),map1z)
view(10,30); daspectm('m',100)
material shiny;camlight;lighting phong
axis tight; set(gca,'Projection','Perspective')
```

Here is the result:



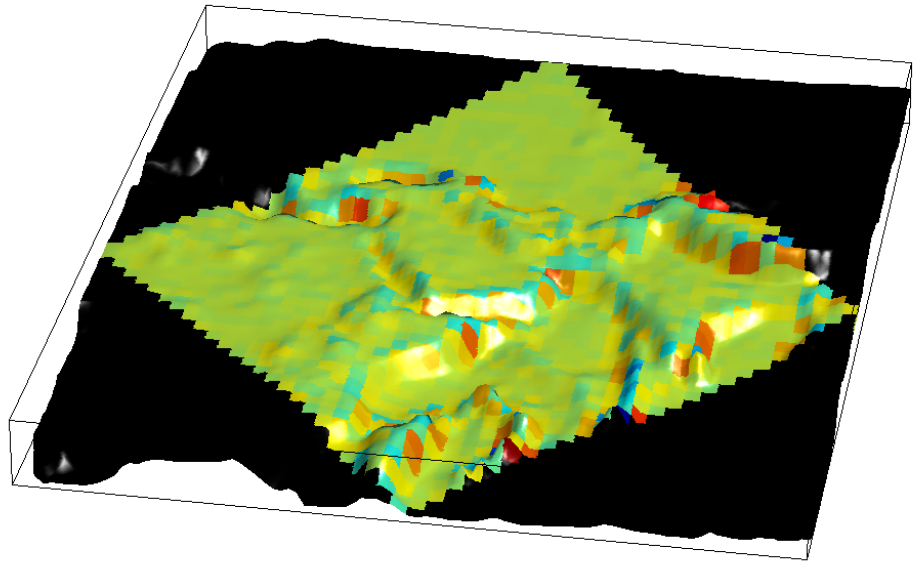
Alternatively, you can construct a regular matrix map version of your general matrix map color data. First, create a new regular matrix map that covers the region of the general matrix map, then imbed the color data values into the new matrix. The new matrix may need to be somewhat lower in resolution than the original, to ensure that every cell in the new map receives a value. The advantage of this approach is that a host of computational functions can be used with the resulting regular matrix map. This example also illustrates the fact that the color and elevation matrices do not have to be the same size. If the resolution of the two is different, you can create the surface as a

three-dimensional elevation map and later apply the colors. You do this by setting the surface `Cdata` property to contain the color matrix, and setting the surface face color to '`TextureMap`'.

Try the following:

```
load topo; load mapmtx

latlim = [min(lt1(:)) max(lt1(:))];
lonlim = [min(lg1(:)) max(lg1(:))];

[smalltopo,smalltopoleg] = maptrims(topo,topolegend,...
       latlim,lonlim);
[smalltopocurv,smalltopocurvleg] = nanm(latlim,lonlim,.5);
% 1/.5 cells per degree scale
smalltopocurv = imbedm(lt1,lg1,del2(map1),...
       smalltopocurv,smalltopocurvleg);

figure; axesm miller
h = meshm(smalltopo,smalltopoleg,size(smalltopo),smalltopo);
view(10,30); daspectm('m',100)
material shiny; camlight; lighting phong
axis tight; set(gca,'Projection','Perspective')
set(h,'Cdata',smalltopocurv,'FaceColor','TextureMap')

ltln2val(smalltopocurv,smalltopocurvleg,39,76.3)
ans =
   1.9238e+03
```

Here is the result:



The Geographic Data Structure

In the previous examples, all of the map data was in the form of individual variables and had to be displayed using different mapping commands according to the type of data at hand (i.e., line, patch, matrix, text, etc.). The Mapping Toolbox also provides an easy means of displaying, extracting, and manipulating collections of all types of map objects that have been organized in a specially defined and formatted *geographic data structure*. Note that this structure is different from the *map projection structure*, which defines a map projection along with its mapping properties. The geographic data structure contains the information required for the proper display of the graphic object. The following table lists the six object types along with their required fields of information:

fields	light	line	patch	regular	surface	text
`type`	•	•	•	•	•	•
`tag`	•	•	•	•	•	•
`lat`	•	•	•		•	•
`long`	•	•	•		•	•
`map`				•	•	
`maplegend`				•		
`meshgrat`				•		
`string`						•
`altitude`	•	•	•	•	•	•
`otherproperty`	•	•	•	•	•	•

Some fields may contain empty entries, but the fields themselves must exist for the object to be displayed correctly. For instance, the `altitude` field can be an empty matrix and `otherproperty` can be an empty cell array. The `tag` field must be a string not equal to the name of the object type (i.e., line, surface, text, etc.). For complete descriptions of these fields and the object types, consult the

`geographic data structure` reference page in the online Mapping Toolbox reference documentation.

The `usalo` workspace contains several variables in the geographic data structure format:

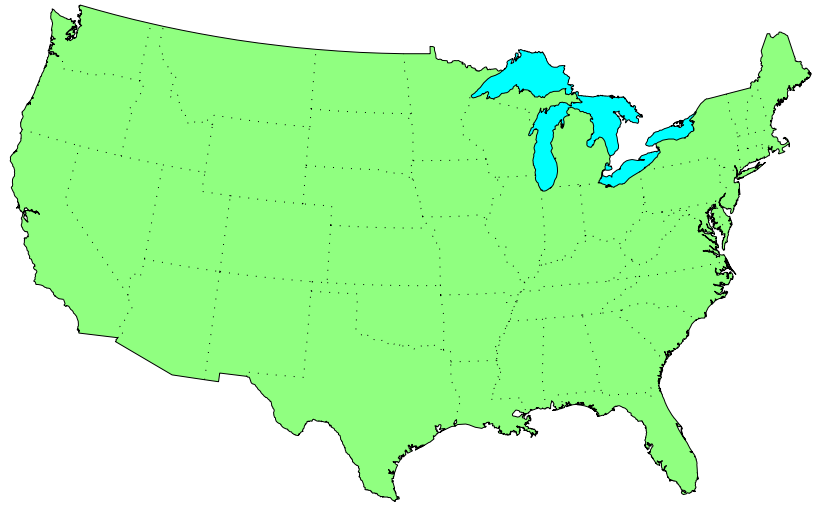
```
load usalo
conus
conus =
              lat: [4392x1  double]
             long: [4392x1  double]
             type: 'patch'
              tag: 'ContinentalUnitedStates'
    otherproperty: []
         altitude: []

stateborder
stateborder =
              lat: [2345x1 double]
             long: [2345x1 double]
             type: 'line'
              tag: 'StateBorder'
    otherproperty: {   1x1 cell  }
         altitude: []

greatlakes
greatlakes =
1x3 struct array with fields:
    type
    tag
    lat
    long
    altitude
    otherproperty
```

The structure `conus` contains a patch object of the continental U.S.; `greatlakes` consists of three patches representing the Great Lakes, and the structure `stateborder` contains line vector data for the state boundaries.

These can be displayed with the `displaym` function:

```
axesm('MapProjection','lambert','MapParallels',[],...
     'MapLatLimit',[23 52],'MapLonLimit',[-130 -62])
displaym(conus)
displaym(greatlakes)
displaym(stateborder)
```

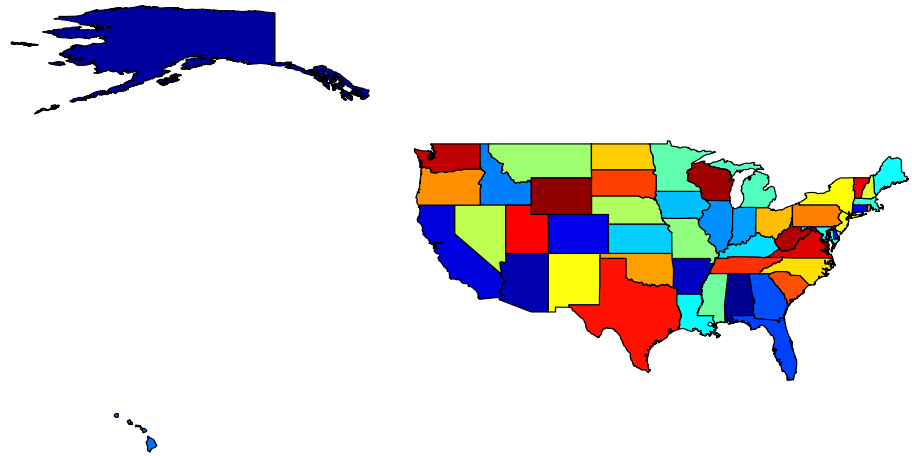
The map shown here is virtually identical to that of a previous example in which the vector variables `uslat`, `uslon`, `gtlakelat`, `gtlakelon`, `statelat`, and `statelon` were used:



Now display all 50 states and the District of Columbia using the patches located in the geographic data structure variable `state`:

```
figure
axesm trystan
displaym(state)
```

Here is the map in a Trystan Edwards Cylindrical projection:



To view individual states that reside in the `state` structure, you can either use the `displaym` function:

```
displaym(state,'mass');
```

Or you can extract and display the data from the formatted structure yourself:

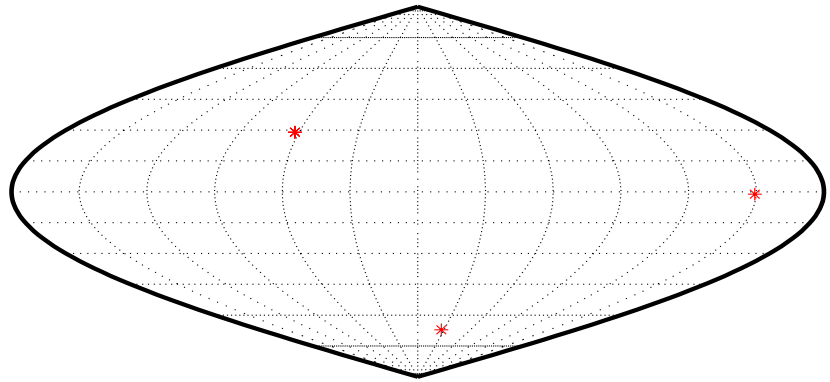
```
[lat,long] = extractm(state,'mass');
plotm(lat,long)
```

Note Most of the Mapping Toolbox atlas data and data from public sources processed by Mapping Toolbox external interface functions enters the MATLAB workspace as geographic data structures. You can also format your own data as geographic data structures.

Interacting with Displayed Maps

The Mapping Toolbox provides the means to interact with displayed maps. The `inputm` command allows you to get the latitude-longitude position of a mouse click; the `gcpmap` function returns the current mouse position, again in latitude and longitude. As an example, display a map axes with its grid. The lines of code below allow you to click on the map three times and store the geographic coordinates for the three points selected in `lat` and `long`. The final `plotm` command will plot the points you clicked on. The display shown here corresponds to the points chosen for the example:

```
axesm sinusoid
framem on; gridm on
points=inputm(3)
points =
       28.9373  -62.1631
       -1.1575  149.3467
       -67.1344   26.8096

plotm(points,'r*')
```



Your points will probably be different.

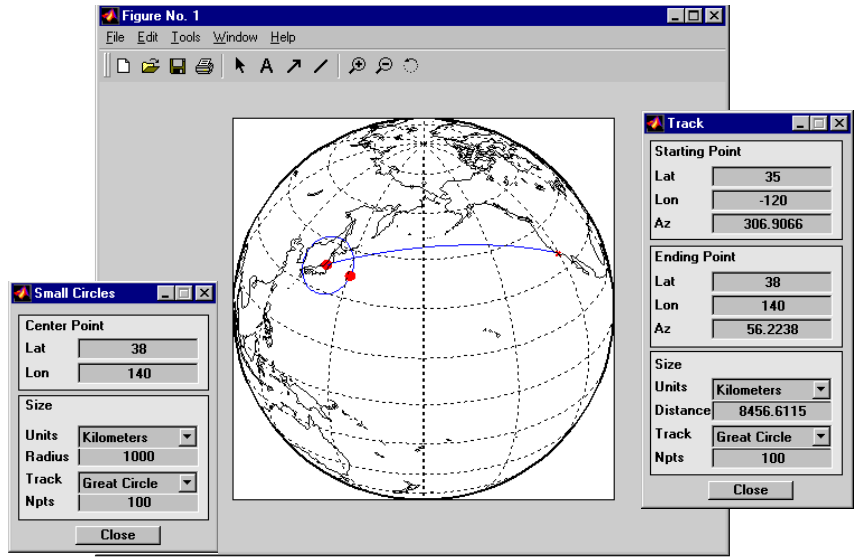
Interactive Tracks and Circles

Geographic line data such a great circle tracks and small circles can also be calculated or created interactively. Great circle tracks are the shortest distance

between points, and small circles are the locus of points a constant distance from a center point. You can use `trackg` and `scircleg` to create them by clicking on the map. If you extend-click on the tracks or circles, you can move the control points to modify the lines, or type specific numbers into a control panel. The control panels also allow the tracks and circles to be used for measurements on the map. Just adjust the lines or circles to the right place on the map, and read the dimensions off the control panel.

Here are a great circle track from the United States to Japan and a 1000 km radius small circle centered on Tokyo. The track and circles were created by clicking on the map at approximately correct positions, then extend-clicking on the track and circle to bring up the control panels. The exact numbers were entered into the control panels for the final result.

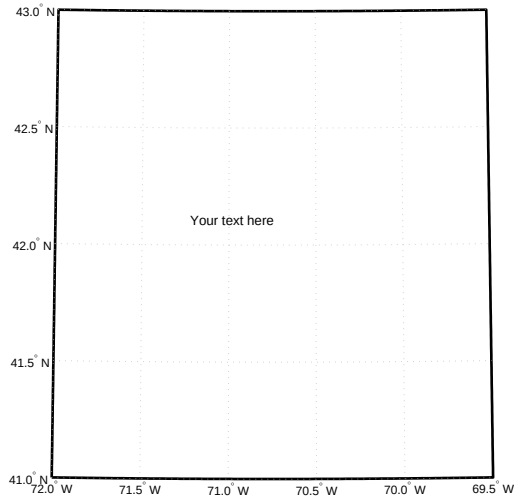
```
axesm('ortho','origin',[30 180])
framem;gridm
plotm(coast,'k')
trackg
scircleg
```



Interactive Text

You can also place text by clicking on a map display. The `gtextm` command takes the latitude and longitude for a text object from where you click on the map. After the text has been placed, you can move it interactively using the Click and Drag Property Editor described in Chapter 6, *GUI Tools*.

```
usamap([41 43],[-72 -69.5],'none')
gtextm 'Your text here'
hidem(gca)
```



Working with Objects by Name

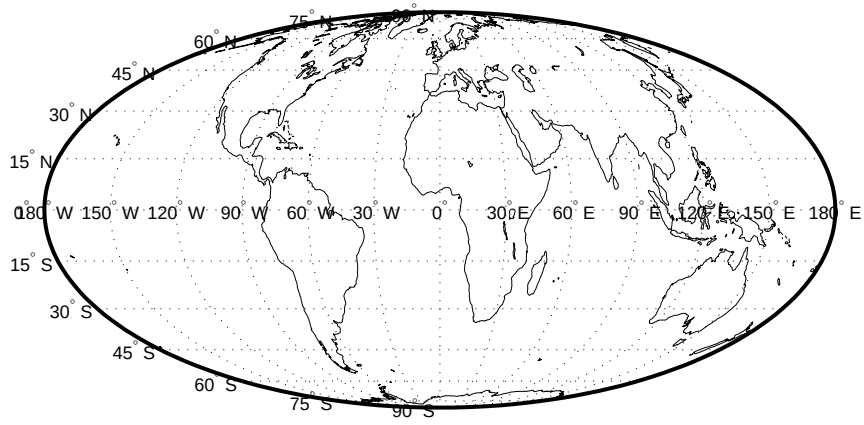
The Mapping Toolbox also allows you to manipulate displayed objects by name. Many mapping commands assign descriptive names to the `Tag` property of the objects they create. The name functions allow you to control the display of groups of similarly named objects, determine the names and change them if so desired, and use the name in the Handle Graphics `set` and `get` commands. There is also a Mapping Toolbox graphical user interface, `mobjects`, to help you manage the display and control of objects.

Some mapping display functions like `framem`, `gridm`, and `contourm` assign object tags by default. You can also set the name upon display by assigning a

string to the Tag property in mapping display functions that allow property-value pairs. If the Tag does not contain a string, the name falls back to an object's Type property, such as 'line' or 'text'.

Display a vector map of the world:

```
figure; axesm fournier
framem on; gridm on;
plabel on; mlabel('MLabelParallel',0)

load coast
plotm(lat,long,'k','Tag','Coastline')
```



You can view the names of the objects in the current axes using namem:

```
namem
ans =

Coastline
PLabel
MLabel
Meridian
Parallel
Frame
```

Change the line width of the coastline:

```
set(handlem('Coastline'),'LineWidth',2)
```

Use a graphical user interface to pick objects and change their colors. Be sure to use the right property name. Patches, for example, have a `'FaceColor'` and `'EdgeColor'`, while most other objects simply have `'Color'`.

```
set(handlem,'Color',uisetcolor)
```

Change the color of any objects with tags containing the substring `'line'`.

```
set(handlem('line',gca,'findstr'),'Color','r')
```

Hide, then show all of the objects of type 'text', regardless of their tags:

```
hidem('alltext')
showm('alltext')
```

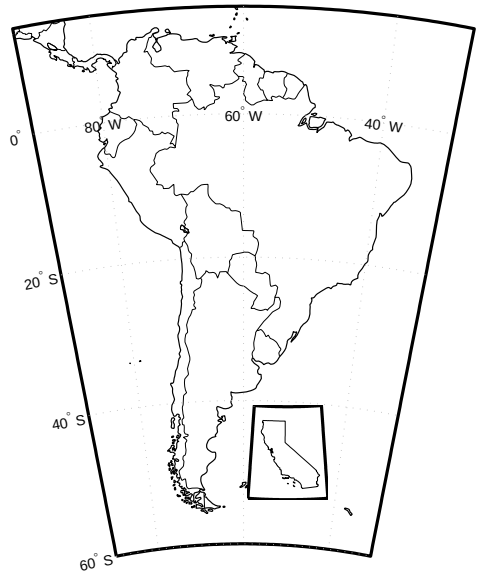
For more information on the use of these functions and tools, consult the `handlem`, `hidem`, `showm`, `clmo`, `namem`, `tagm`, and `mobjects` reference pages in the online Mapping Toolbox reference documentation.

Specialized Map Functions

Inset Maps

Inset maps are often used to display widely separated areas, generally at the same scale. You can create inset maps by adding multiple axes to a figure and defining appropriate map projections for each. If you want to ensure that the scale of each of the maps is the same, use `axesscale` to resize them. Here is an example of an inset map of California at the same scale as the map of South America, graphically relating the size of the continent to what may be a more familiar region.

```
h1 = worldmap('south america');
h2 = axes('pos',[.5 .2 .1 .1]);
usamap('californiaonly','lineonly')
setm(h2,'FFaceColor','w')
mlabel; plabel; gridm  % toggle off
axes(h2)
axesscale(h1)
hidem([h1 h2])
```



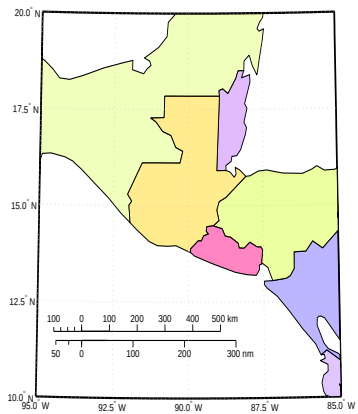
Graphic Scale

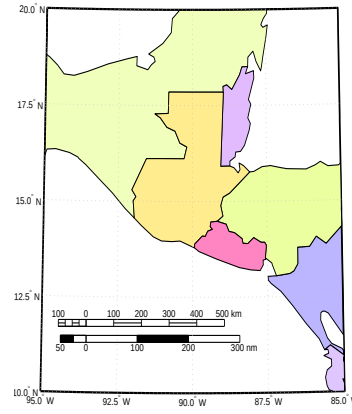
Another element used to provide an indication of size is the graphic scale. This is a ruler-like object that shows distances on the ground at the correct size for the projection. You can use the `scaleruler` function to add a graphic scale to the current map. The `scaleruler` settings can be checked or modified using `getm` and `setm`. You can also move the graphic scale to a new position by dragging its baseline.

As an example, 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. Show the same map with different ruler styles.

```
worldmap('lo','guatamala','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))

setm(handlem('scaleruler1'),'RulerStyle','lines')
setm(handlem('scaleruler2'),'RulerStyle','patches')
```





Choropleth Maps

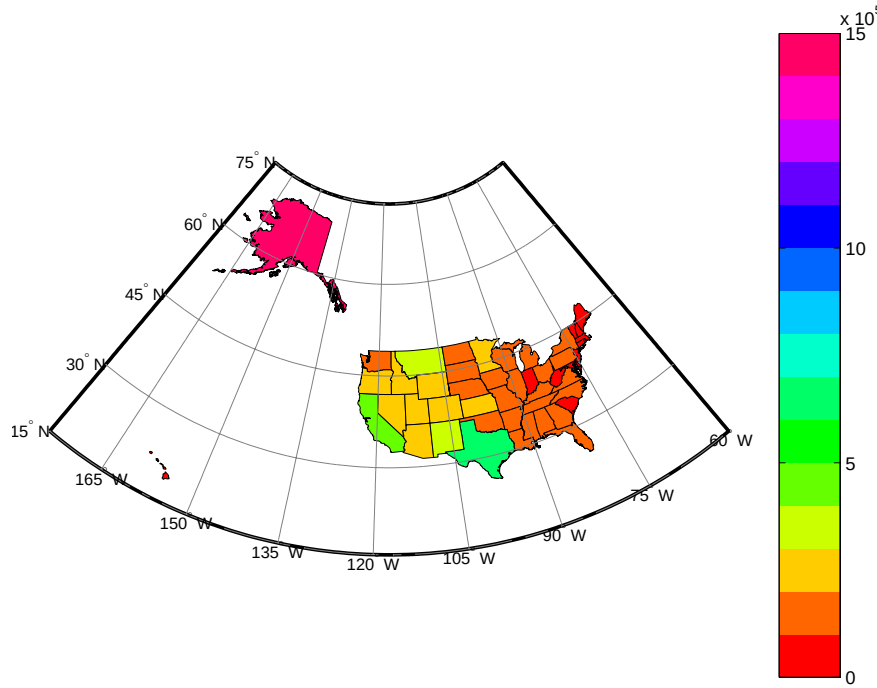
A common type of map display has patch faces colored proportional to some specified data. The data generally represents the value of something per unit area, such as population density, electoral votes, or incidence of disease. Maps of this type are generally called *choropleth* maps. It is easy to construct choropleth maps with MATLAB and the Mapping Toolbox. Simply assign the data values to the `CData` property of the displayed patches.

In the following example, patches representing the 50 states of the U.S. (and the District of Columbia) are displayed and colored according to the surface areas calculated by the `areaint` function. An equal-area projection is appropriate for this kind of display.

```
load usalo
```

```
axesm('MapProjection','eqaconic','MapParallels',[],...
      'MapLatLimit',[15 75],'MapLonLimit',[-175 -60],...
      'MLineLocation',15,'MLabelParallel','south',...
      'MeridianLabel','on','ParallelLabel','on',...
      'GLineStyle','-','GColor',0.5*[1 1 1],...
      'Grid','on','Frame','on')
displaym(state)

geoid = almanac('earth','geoid');
tags = {state.tag};
for i=1:length(state)
      lat = state(i).lat; long = state(i).long;
      surfarea = sum(areaint(lat,long,geoid));
      set(handlem(tags{i}),'CData',surfarea);
end

caxis([0 1.5E6])
colormap(hsv(15))
colorbar
```

The coloring scheme of the map makes it easy to distinguish which states are comparable in area. The color scale to the right of the map shows the surface areas in square kilometers:

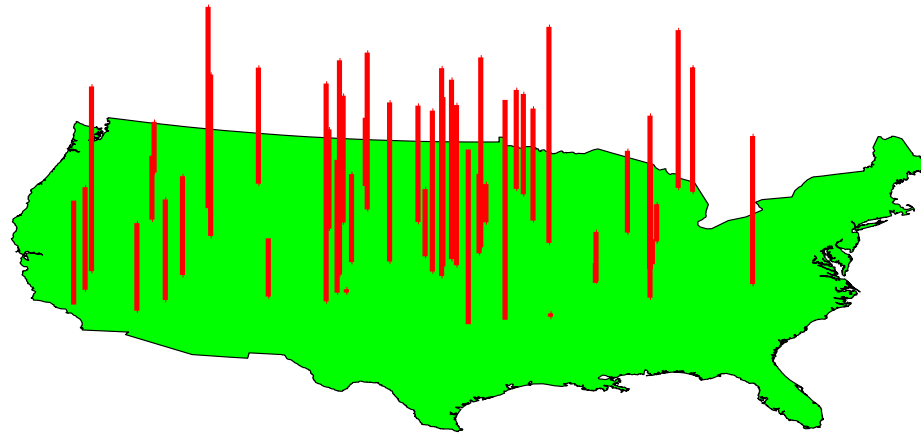


The largest state is Alaska (about 1.5 million square kilometers), while Rhode Island stands as the smallest state (about 2500 square km). The District of Columbia, of course, is the smallest entity of the dataset, at roughly 200 square km.

Thematic Map Functions

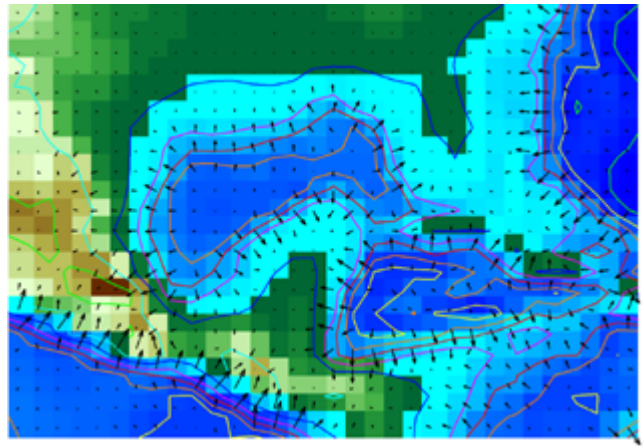
The Mapping Toolbox provides a wide variety of other display and symbology functions. The `cometm` and `quiverm` functions operate like their MATLAB counterparts, `comet` and `quiver`. The `stem3m` command allows you to display geographic bar graphs. Like the MATLAB `scatter` command, the `scatterm` function allows you to display a thematic map with proportionally sized symbols. The `tissot` function calculates and displays Tissot Indicatrices. For more information on these capabilities, consult the descriptions of these functions in the “Mapping Reference” section of the online Mapping Toolbox documentation.

Here is an example of a stem plot over a map of the continental United States. The bars could represent anything from selected city populations to the number of units of a product purchased at each location:

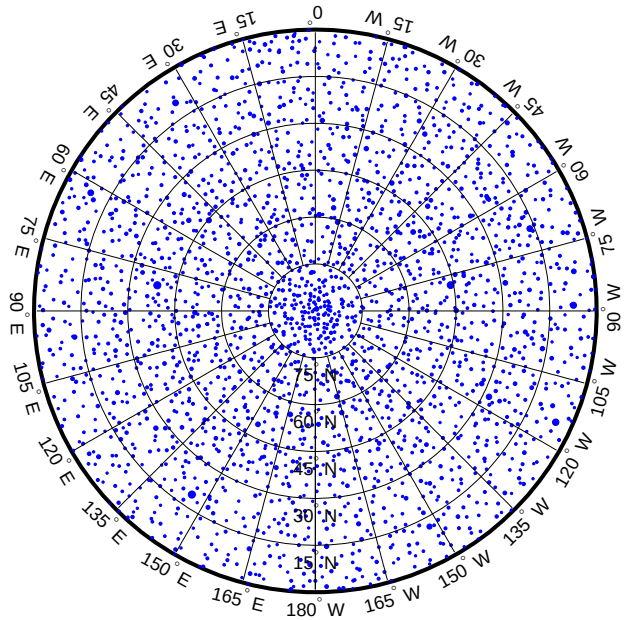


Contour and quiver plots can be extremely useful in analyzing matrix data. In the following example, contour elevation lines have been drawn over a topographical map. The region displayed is the Gulf of Mexico, obtained from the `topo` matrix. Quiver plots have been added to visualize the gradient of the topographical matrix.

Here is the displayed map:



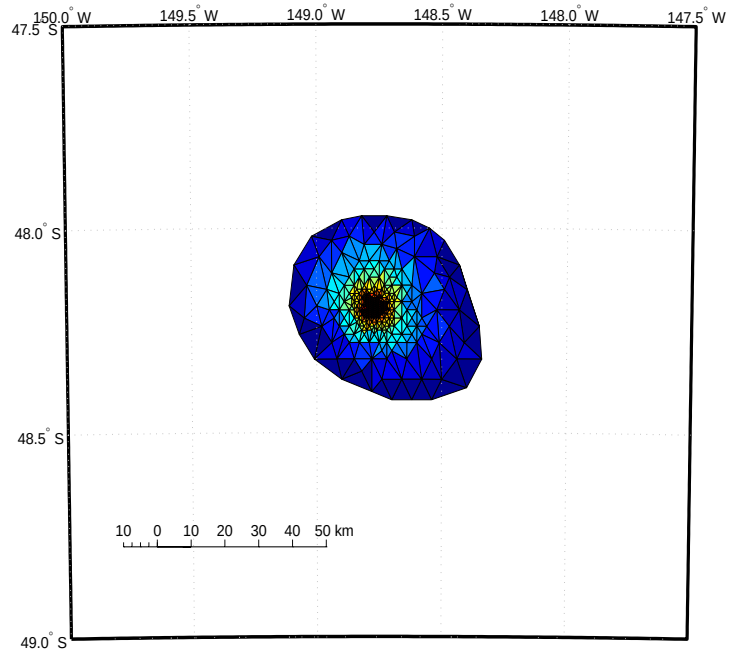
Here is an example of using the `scatterm` function to create a star chart of the northern sky. The stars are represented by filled circles whose size is proportional to visual magnitude. Star gazers may recognize the Big Dipper constellation, or Ursa Major, located at 180 W, 60 N:



Cartesian MATLAB Display Functions

If you cannot find a Mapping Toolbox display function that does what you need, you may be able to use a non-mapping MATLAB function. You can use the MATLAB function to add the graphic objects to the display, using latitude and longitude as x and y, and then project the data afterwards. For example, the Mapping Toolbox does not have a function that displays a triangulated surface from random data, but MATLAB does. Create such a map of the `seamount` data provided with MATLAB.

```
load seamount
worldmap([-49 -47.5],[-150 -147.5],'none')

tri = delaunay(y,x);
h = trisurf(tri,y,x,z);
project(h,'yx')

scaleruler
```

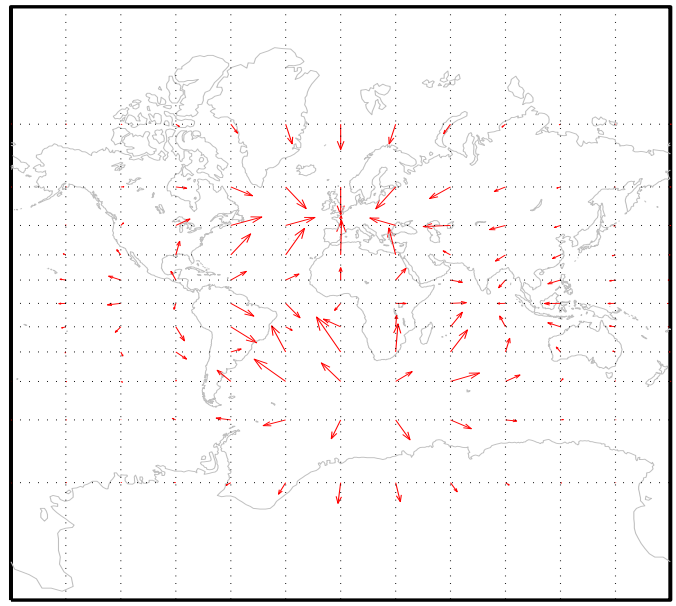


If the displayed objects are already in the right place and do not need to be projected, you can trim them to the map frame and convert them to mapped objects using `trimcart` and `makemapped`. They can then be manipulated as if they had been created with map display functions. This is useful when you want to get back the geographic coordinates.

You can also combine Mapping Toolbox and MATLAB commands to mix spherical and Cartesian coordinates. An example would be a quiver plot in which the locations of the vectors are geographic, but the lengths are not. In that case, you can use Mapping Toolbox projection calculations and MATLAB graphics commands. Cylindrical projections are the simplest. North is up and east is right, so just project the locations.

```
figure; axesm mercator; framem; gridm
plotm(coast,'color',[.75 .75 .75])

[u,v] = gradient(peaks(13)/10);
[lat,lon] = meshgrat(-90:15:90,-180:30:180);
```

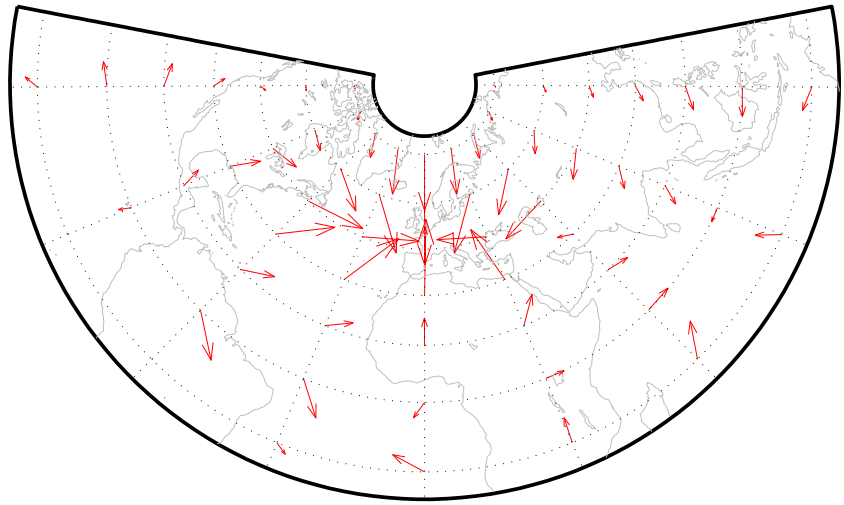
```
[x,y] = mfwdtran(lat,lon);

h = quiver(x,y,u,v,.2,'r');
trimcart(h)
```



An extra step may be required for non-cylindrical projections. In these projections, the directions of north and east vary with location. To make the directions agree with the map grid, vectors should be rotated to bring them into alignment. This can be done with the vector transformation function `vfwdtran`. Consider the same data displayed on a conic projection.

```
figure
axesm('lambert','MapLatLimit',[-20 80])
framem; gridm
plotm(coast,'color',[.75 .75 .75])

[x,y] = mfwdtran(lat,lon);
thproj = deg2rad(vfwdtran(lat,lon,90*ones(size(lat))));
[th,r] = cart2pol(u,v);
[uproj,vproj] = pol2cart(th+thproj,r);
```

```
h = quiver(x,y,uproj,vproj,0,'r') ;
trimcart(h)
```



Conformal projections are often the best choice for displays like this. They preserve angles, ensuring that the difference between north and east will always be 90 degrees in projected coordinates.

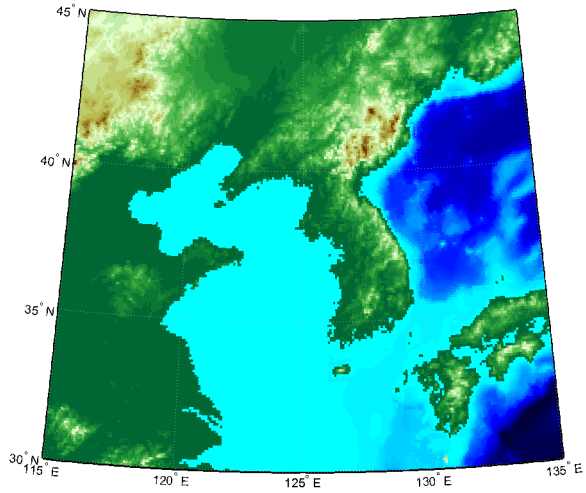
Colormap and Colorbars

Colormap for Terrain Data

In this tutorial, you may have noticed the use of the function `demcmap` in several digital elevation map (DEM), or topographic, display examples. This function creates colormaps appropriate for the display of DEMs, although it is certainly not limited to just DEMs.

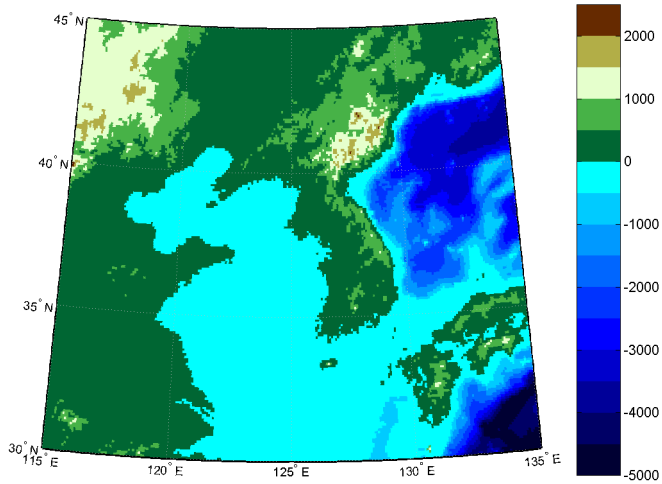
The colormaps, by default, have atlas-like colors varying with elevation or depth that properly preserve the land-sea interface.

```
load korea
worldmap(map,maplegend,'meshonly')
demcmap(map)
hidem(gca)
```



You can also require that all altitudes within a particular range be assigned the same color. This effectively results in a contour map with color breaks at multiples of some level. Here is the same map with colors constant within 500 meter bands.

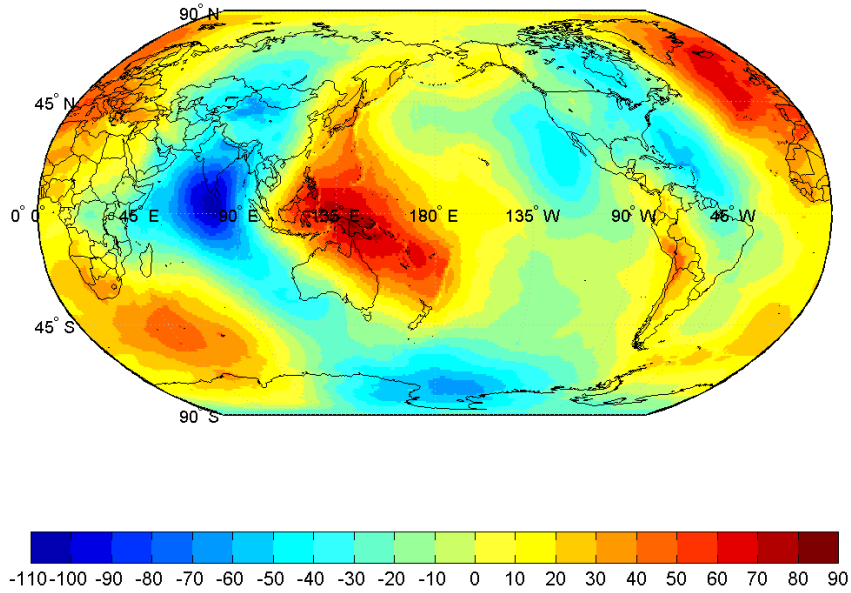
```
demcmap('inc',map,500)
colorbar
```



Contour Colormaps

You can create colormaps that make surfaces look like contour maps for other types of data besides terrain. The `contourcmap` command creates a colormap that has color changes at multiples of a contour level increment. You need to specify the color increment and the name of a colormap function. You can write your own colormap function, but it should work like those provided with MATLAB. Optionally, you can also have `contourcmap` add and label a colorbar that is similar to MATLAB's `colorbar`.

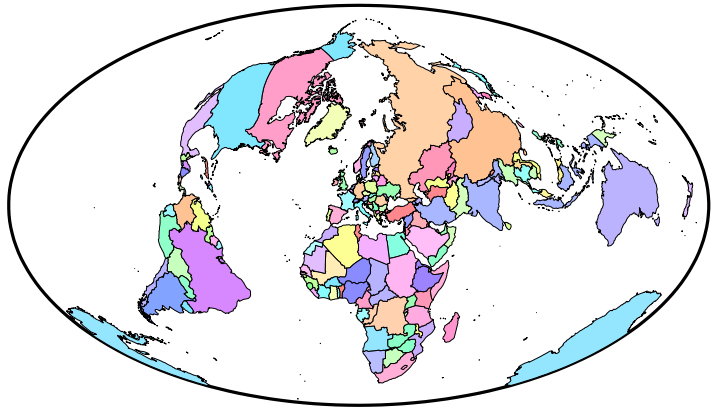
```
load geoid
worldmap(geoid,geoidlegend)
contourcmap(10,'jet','colorbar','on','location','horizontal')
```



Colormap for Political Data

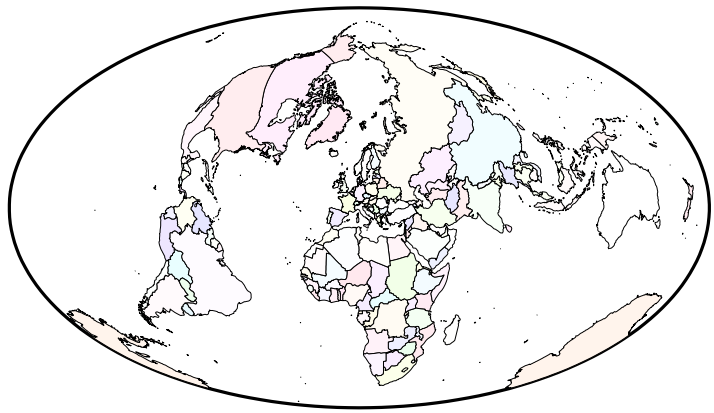
Political units are typically shown colored with muted, contrasting colors that make it easy to distinguish one country from another. A colormap of this kind can be created using the `polcmap` command. The `polcmap` command creates a colormap with randomly selected colors of all hues. Since the colors are random, if you don't like the result the first time, execute `polcmap` again to generate a different colormap. To ensure that the colormap is always the same, reset the seed on MATLAB's random number generator `rand`.

```
axesm bries
displaym(worldlo('POpatch'))
framem
polcmap
hidem(gca)
tightmap loose
```



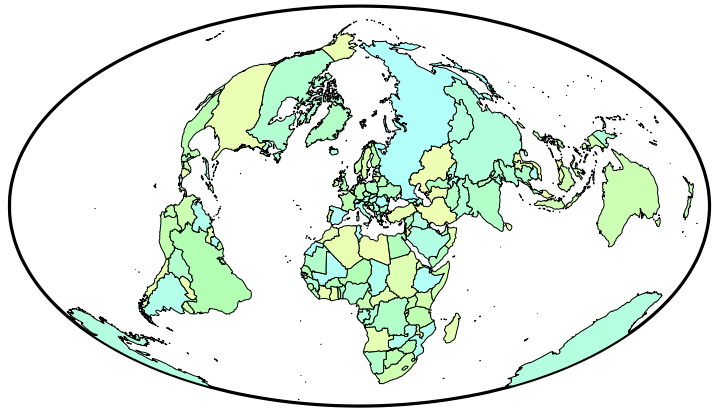
You can also control the saturation of the colors. Here is the same map with 256 colors and a maximum saturation of 0.1.

```
polcmap(256,.1)
```



For maximum control over the colors, specify the limits on hues, saturations, and values.

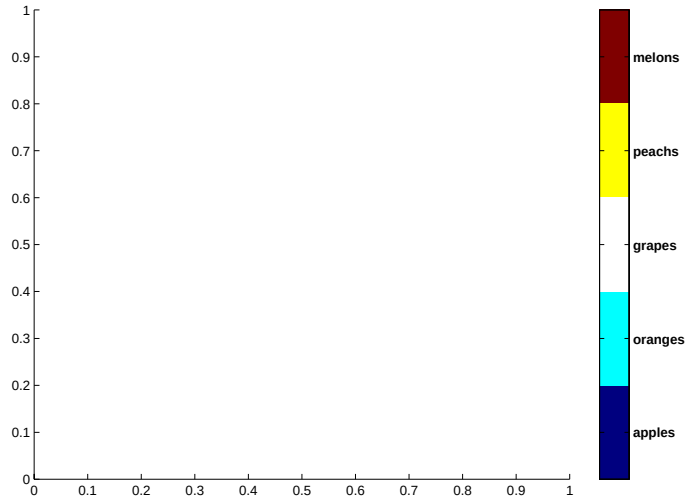
```
polcmap(256,[.2 .5],[.3 .3],[1 1])
```



Labeled Colorbars

Indexed maps often have names associated with the numeric index values. The `lcolorbar` command allows you to create a colorbar with text labels aligned with the colors or color breaks.

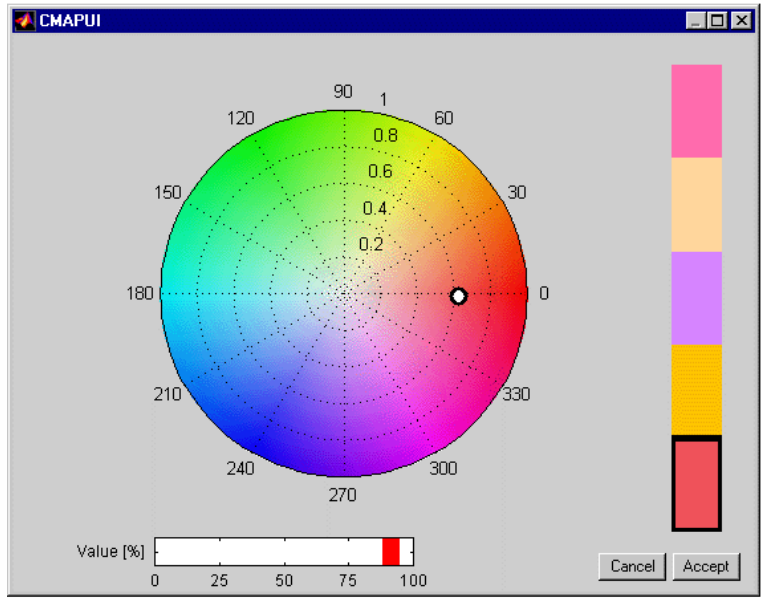
```
figure; colormap(jet(5))
labels = {'apples','oranges','grapes','peachs','melons'};
lcolorbar(labels,'fontweight','bold');
```



Create Colormaps Interactively

Indexed maps often require colormaps with special colors for each index value. To avoid building such colormaps by hand, use the Graphical User Interface for colormaps, `cmapui`. The `cmapui` panel allows you to select color entries in a colormap by clicking on the colorbar. To change the color's hue and saturation, drag the color **Marker** on the color wheel. To control the value of the color in HSV space, drag the red **Slider**. Clicking the **Accept** button returns the modified colormap.

Here is the colormap panel with colors more appropriate to the labeled colorbar in the previous example.



Printing Maps

Maps are often printed at a size that makes objects on paper a particular fraction of their real size. The ratio of the mapped and real object sizes is called scale, and it is usually notated with a colon as “1:1,000,000” or “1:24,000”. Another way of specifying scale is to call out the printed and real lengths, for example “1 inch = 1 mile.”

You can specify the printed scale using the `paperscale` function. It modifies the size of the printed area on the page to match the scale. If the resulting dimensions are larger than your paper, you can reduce the amount of empty space around the map using `tightmap`, `zoom`, or `panzoom`, and by changing the axes position to fill the figure. This also reduces the amount of memory needed to print with the `zbuffer` (raster image) renderer. Be sure to set the paper scale last. For example:

```
set(gca,'Units','Normalized','Position',[0 0 1 1])
tightmap
paperscale(1,'in', 5,'miles')
```

You can also check the size and extent of text and the relative position of axes using `previewmap`. This function resizes the figure to the printed size. For more information on printing see the “Basic Printing and Exporting” section of the MATLAB graphics documentation.

4

Atlas Data

Types of Data 4-2

World Vector Data 4-3
Coastlines 4-3
Low-Resolution World Atlas Data 4-5
High-Resolution World Atlas Data 4-15

World Matrix Data 4-21
Political 4-21
Terrain 4-24
Geoid 4-25

United States Vector Data 4-27
Low Resolution Data 4-27
Medium Resolution State Outlines 4-32

United States Matrix Data 4-36
Political 4-36
Terrain 4-37

Astronomical Data 4-40

Types of Data

The Mapping Toolbox includes a number of datasets for global and regional displays. Map data is available in both vector and matrix format, covering the world and the United States. The Mapping Toolbox also provides astronomical data for simple stellar cartography.

More detailed data is available over the Internet or on CD-ROM and can be imported to MATLAB using the data interface functions described in Chapter 8, “External Data Interface”.

World Vector Data

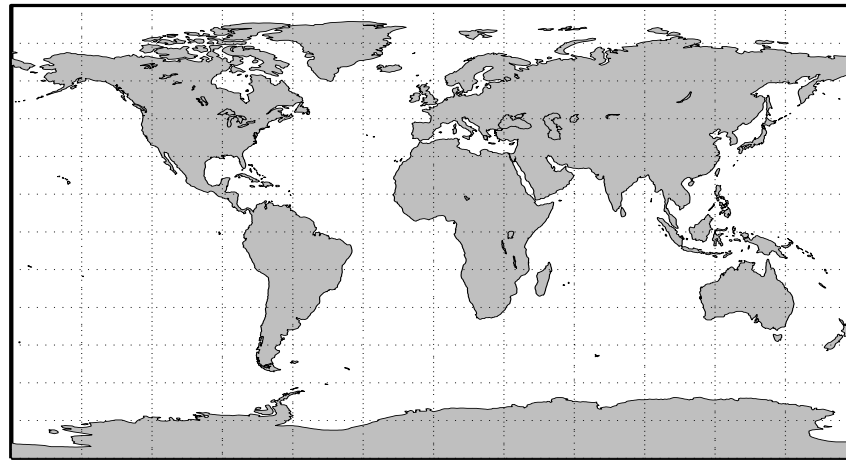
Coastlines

The coast MAT-file contains a set of vector shorelines intended for global displays in which political boundaries are not required. The data is stored in vectors of latitude and longitude, consisting of nearly 10,000 points. While this a considerable quantity of data, it is of very low resolution by cartographic standards.

```
load coast
whos
   Name       Size       Bytes     Class

   lat        9589x1     76712     double array
   long       9589x1     76712     double array
```

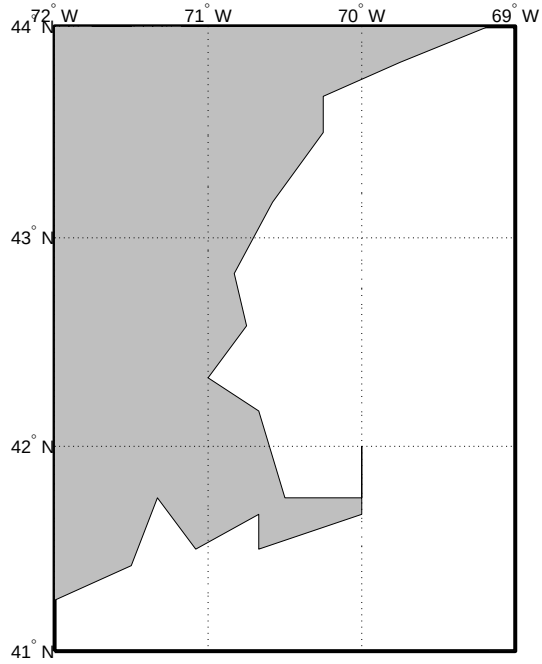
The dataset is in patch format, meaning it has polygon segments that return to their starting points and can therefore be usefully displayed using patch commands. The polygon segments are separated by NaNs into about 240 faces. The data can also be displayed as lines or points.

```
axesm giso; framem; gridm
patchesm(lat,long,'FaceColor',0.75*[1 1 1])
```



The resolution for this dataset makes it ideal for global-scale displays but inappropriate for significantly smaller regions, as can be seen when the region around Cape Cod in the northeastern United States is displayed. This is, incidentally, the area covered by the cape MAT-file provided with MATLAB.

```
figure
axesm('MapProjection','mercator',...
    'MapLatLimit',[41 44],'MapLonLimit',[-72 -69])
framem
gridm('MLineLocation',1,'PLineLocation',1);
mlabel('MLabelLocation',1)
plabel('PLabelLocation',1)
patchesm(lat,long,'FaceColor',.75*[1 1 1])
```



Low-Resolution World Atlas Data

The `worldlo` global atlas data contains a set of very small-scale (approximately 1:30,000,000) data for use in global displays. The data is provided in the form of Mapping Toolbox geographic data structures containing national boundaries, rivers, lakes, and cities. Here is a list of the structures in the `worldlo` workspace:

```
load worldlo
whos
   Name        Size       Bytes     Class

   DNline      1x2        112546    struct array
   DNpatch     1x39       59884     struct array
   POline      1x2        456600    struct array
   POpatch      1x206     682454    struct array
   POtext      1x194      188132    struct array
   PPpoint     1x319      400426    struct array
   PPtext      1x318      296988    struct array
   description 4x75       600       char array
   gazette     1x513      353504    struct array
```

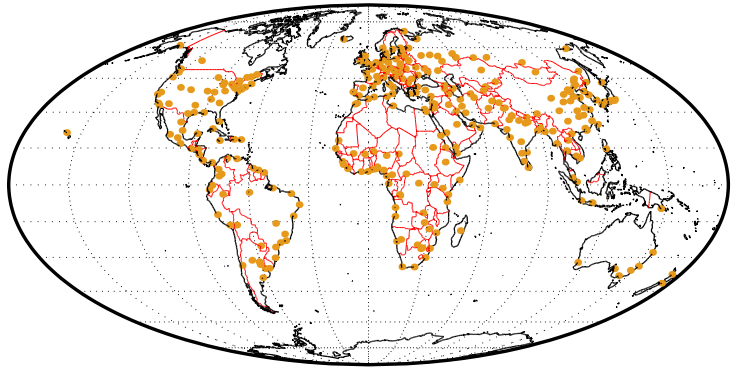
The data has been classified into Drainage (DN), Political/Ocean boundaries (PO), and Populated Places (PP). The data is further separated into structures containing patches, lines, points, or text.

In addition to containing more kinds of data than `coast`, this dataset is also more detailed. The drainage data consists of about 7,000 points, while the political lines and patches each have about 30,000 points. There are more than 200 political units in the political data and more than 300 major cities in populated places. Because of the quantity of data, complicated displays may take longer to project and render. Displays of patches require the greatest memory and computing time.

The contents of the structures can be displayed from the command line using `displaym`. For example, type the following:

```
axesm mollweid; framem; gridm
displaym(POline)
displaym(PPpoint)
```

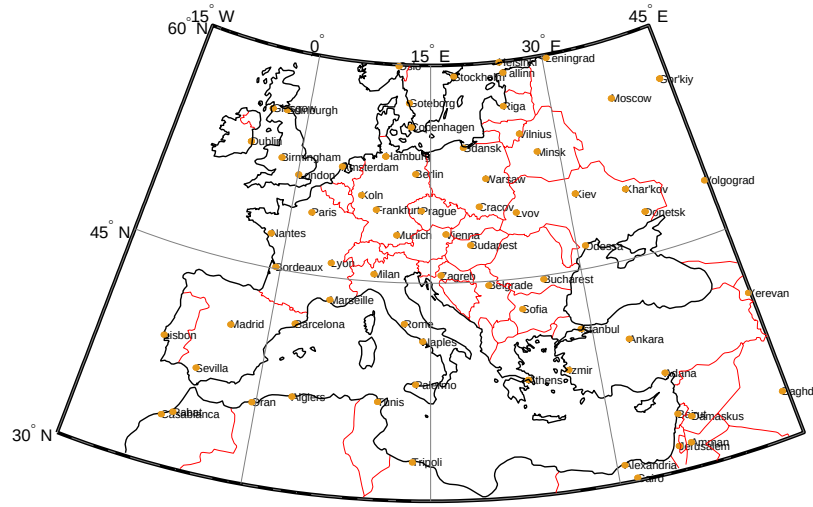
Here is the result:



The map displays the coastlines, international borders, and major cities of the world. We could also label the cities with data from the `PPtext` structure, but at this scale, the result would be unreadable. When we restrict ourselves to a smaller geographic region, more information can be shown:

```
figure
axesm('Mapprojection','eqaconic','MapParallels',[],...
   'MapLatLimit',[30 60],'MapLonLimit',[-15 45],...
   'MLabelLocation',15,'MLineLocation',15,...
   'PLabelLocation',15,'PLineLocation',15,...
   'GColor',.5*[1 1 1],'GLineStyle','-')
framem; gridm; mlabel; plabel
displaym(POline)
displaym(PPpoint)
h = displaym(PPtext); trimcart(h)
```

The zoomed-in region of Europe allows the names of the major cities to be read more easily:

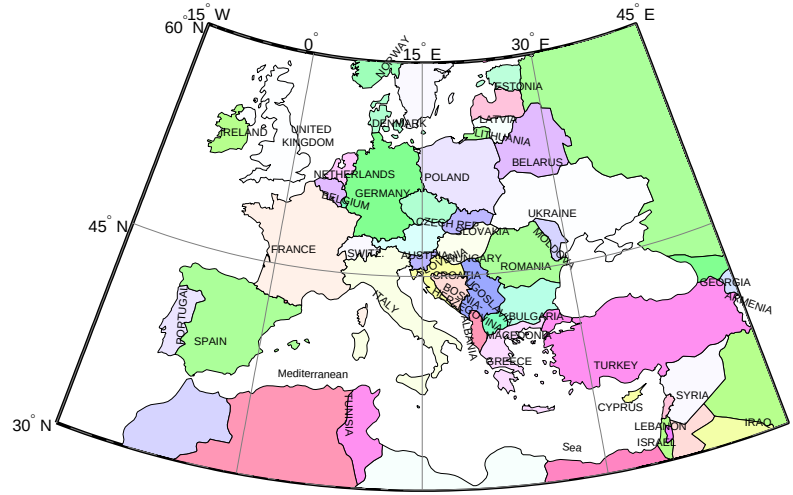


Note Unlike vector and matrix data, text objects are not automatically trimmed when they fall outside the map frame limits. You can remove them using the `trimcart` function or some other method (e.g., click-on-text and `deletem(gco)`, or use click-and-drag tools).

There are other ways to represent political data. We can display the countries as patches, label them with strings from the `POtext` structure, and pick random colors for the patch faces. Clear the previous line map, and redraw the patch map:

```
clma
displaym(POpatch); polcmap
h = displaym(POtext); trimcart(h); zdatam(h,1)
```

Here is the result:

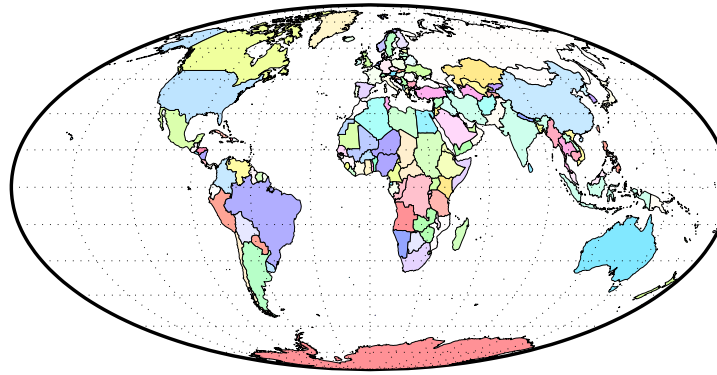


Note how the separated parts of countries are filled with the same colors. Sardinia and Sicily, for example, are shaded the same color as the boot of Italy. This is accomplished automatically because each of the components is tagged as “Italy.”

Every element in a `worldlo` structure has a tag field. Drainage data has been tagged to distinguish between rivers and lakes. The political/ocean lines are tagged as coastlines or international boundaries, and the political patches have been tagged with the names of the country. Both of the functions `displaym` and `mlayers` use the same colors for similarly tagged objects if no `otherproperty` is specified in the geographic data structure.

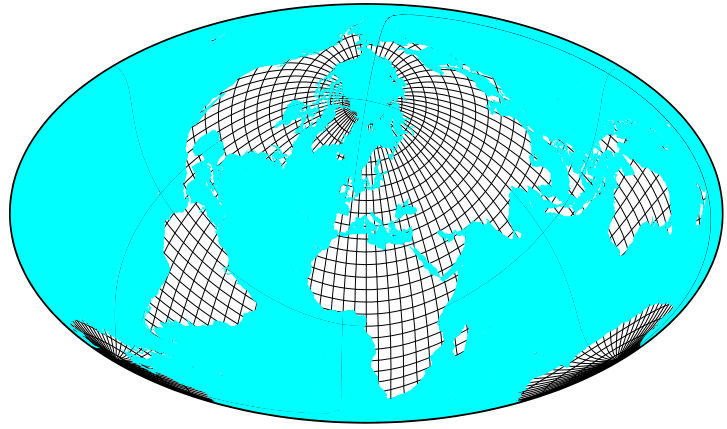
In the following example, it is evident that the many islands of Indonesia and Canada are similarly colored, as are all parts of the United States:

```
figure
axesm mollweid; framem; gridm
displaym(POpatch)
```



There is also a corresponding set of patches for the oceans. These can be used to *blank out* objects in ocean areas. The data is stored separately in the `oceanlo.mat` file, but it can be accessed using the `worldlo` atlas function. The patches are in large tiles with few points along the edges, which may result in visible seams for non-cylindrical projections. `interpm` can be used to fill in points at a sufficiently fine spacing.

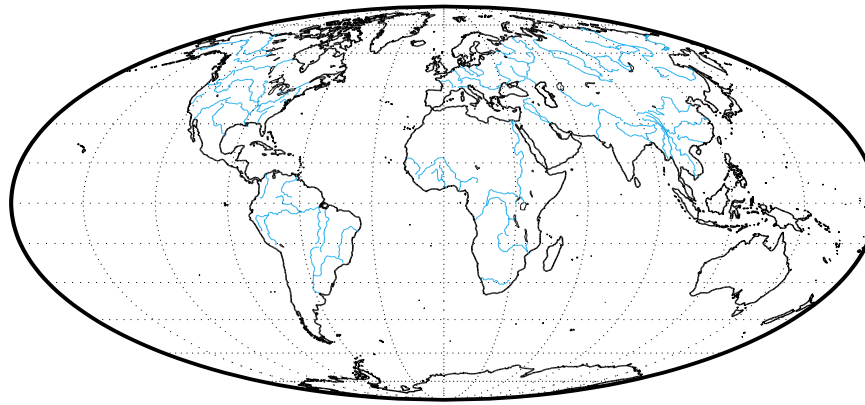
```
[lat,lon] = extractm(worldlo('oceanmask'));
[lat,lon] = interpm(lat,lon,5);

axesm('bries','GAlt',-1,'MLineLocation',5,'PLineLocation',5,...
      'GLineStyle','-')
framem; gridm
patchesm(lat,lon,'c','EdgeColor','none')
```



You can use the tags to reduce the amount of data in the display by selectively deleting data. This can be done by displaying all of the data in a structure and then using the tags to get the handles of the displayed objects you wish to remove. The first European example made use of this technique, as does the next one. It shows rivers, lakes, and coastlines. International boundaries have been removed using their tags:

```
clma
axesm mollweid; framem; gridm
displaym(POline)
delete(handlem('International Boundary'))
displaym(DNline)
```

Here is the map of world coastlines and drainage:



There are a number of other ways to work with the tags associated with this data. We can view the tags in the workspace by working directly with the geographic data structure.

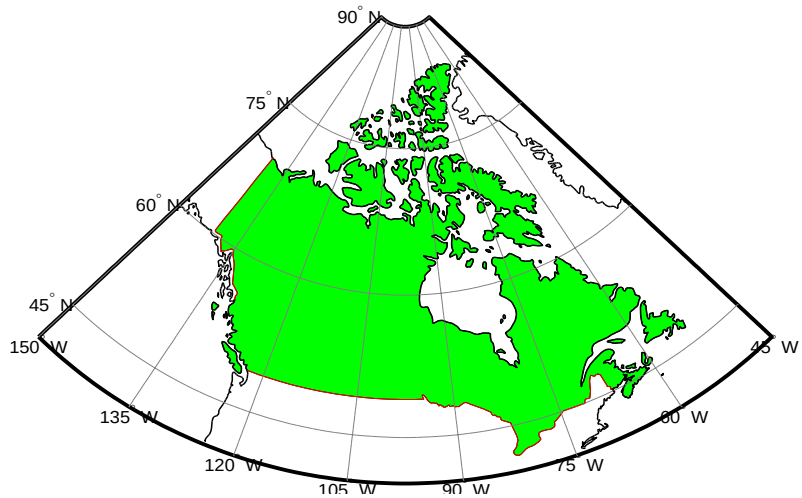
```
unique(strvcat(DNline.tag),'rows')
ans =
Inland shorelines
Streams,rivers,channelized rivers
```

Or we can use `mlayers` and `mobjects` to plot the structures, view the tags, and manipulate similarly tagged objects as a group. See Chapter 7, “GUI Tools” for some examples.

The tags can also be used to extract data from the structures. The following commands extract the patch data for Canada and plot it with the political line data:

```
figure
axesm('MapProjection','eqaconic','MapParallels',[],...
    'MapLatLimit',[40 90],'MapLonLimit',[-150 -45],...
    'MLabelLocation',15,'MLineLocation',15,...
    'PLabelLocation',15,'PLineLocation',15,...
    'GColor',.5*[1 1 1],'GLinestyle','-',...
    'MLabelParallel','south')
framem; gridm; mlabel; plabel
displaym(POline)
```

```
[clat,clon] = extractm(POpatch,'canada');
patchesm(clat,clon,'g')
```



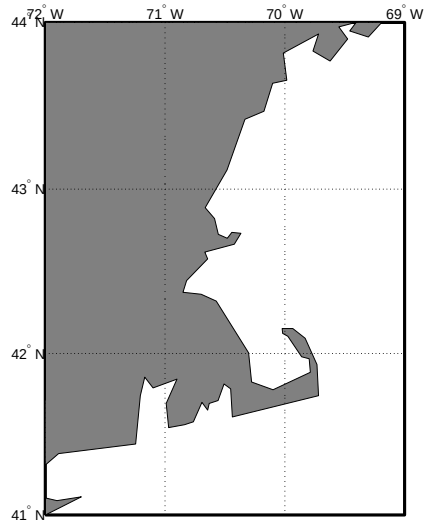
Note Extracting and displaying only the desired data can reduce the time and memory required to display a map.

The Mapping Toolbox automatically trims and saves data outside the current map latitude and longitude limits. If you use `displaym` to show a small part of the world, the rest of the world data is retained within the map axes structure. Every time the projection parameters are changed, all of the world data is restored, trimmed, and projected. This overhead can be significant for a dataset as large as `worldlo`. It is best to finalize the projection parameters before plotting a lot of data.

Apply the extraction technique to generate a display of Cape Cod. As was shown in the earlier examples, this data contains somewhat generalized shapes for the countries and rivers. The level of detail is great enough for global and regional maps, but the distinct data points become noticeable when displayed for smaller regions. This data should be considered accurate to no better than 10 or 20 kilometers. For more accurate data, see the `usahi` atlas

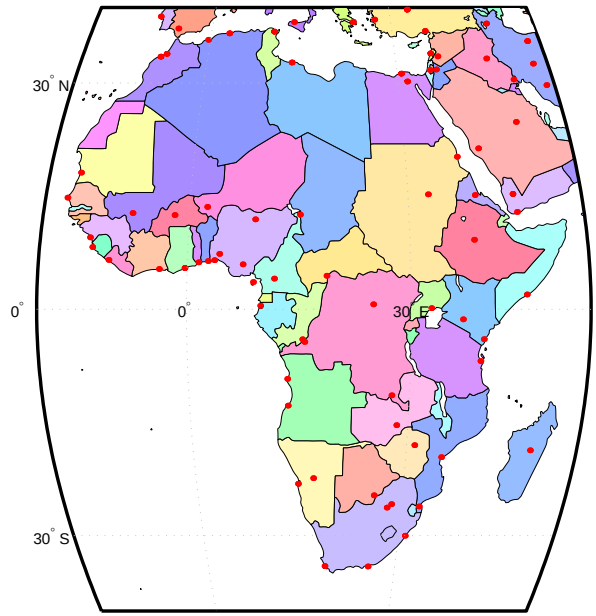
data or the high resolution data accessible through the external interface described in Chapter 8, “External Data Interface.”

```
figure
axesm('MapProjection','mercator','Frame','on',...
   'MapLatLimit',[41 44],'MapLonLimit',[-72 -69])
gridm('MLineLocation',1,'PLineLocation',1)
mlabel('MLabelLocation',1); plabel('PLabelLocation',1)
h = displaym(POpatch,'united states');
set(h,'FaceColor',.5*[1 1 1])
```



There are some more convenient ways of working with the `worldlo` atlas data. The `worldmap` function makes it easy to create map displays. Just specify the region or country you wish to map. This function will handle the cartographic details like selecting a projection, setting the map limits, and setting the origin and the grid and label spacings. You can also use the `worldlo` interface function to load one of the atlas data structures as an argument to a function, or load just that structure into the workspace. Make a map of Africa with filled patches, and add the locations of major cities to the display.

```
worldmap('africa','patch'); polcmap(256)
h = displaym(worldlo('PPpoint')); setm(h,'Color','r')
```



The `worldlo` atlas data can also be used to look up names and locations. All of the names mentioned in the `POtext` and `PPtext` structures are collected in a gazette structure, which can be queried using `extractm`. For example, the location of the city of Antananarivo on the island of Madagascar can be found by using the following commands:

```
[lat,long,indx] = extractm(gazette,'antan');
gazette(indx)
ans =
             type: 'line'
    otherproperty: []
              tag: 'Antananarivo'
           string: 'Populated Place Name'
         altitude: []
              lat: -18.9141
             long: 47.5258
```

A much more extensive gazette feature can be found in the Digital Chart of the World (DCW), containing more than 10,000 names, compared to about 500 in

the `worldlo` MAT-file. See Chapter 8, “External Data Interface” for more information on the DCW.

High-Resolution World Atlas Data

The `worldhi` global atlas data contains a set of very large-scale (approximately 1:1,000,000) data for use in regional displays. The data is provided in the form of Mapping Toolbox geographic data structures containing national boundaries, ocean names, and some cities.

You can request the outline of a particular country by providing the name. For a complete list of names, type `worldhi` with no arguments.

```
worldhi

WORLDLO atlas data:


  Geographic Data Structures:

    POpatch       - Countries as patches
    POtext        - Names of water bodies as text
    PPpoint       - Major cities as points
    PPtext        - Major cities as text labels

  Countries in the WORLDHI database:

    Afghanistan
    Agalega Island (MAU)
    Albania
    Alderney (UK)
    Algeria
    American Samoa (USA)
    Andorra
...
```

When you provide a country name, `worldhi` returns a geographic data structure containing the NaN-clipped vectors with the outline of the mainland and islands. The result also contains some additional fields over those required by the definition of the Geographic Data Structure. The `latlim` and `lonlim` fields contain the limits of a bounding box for the country in its entirety. The

area field contains the area (in units of square kilometers) and the latitude and longitude limits the bounding box for each of the mainland and islands.

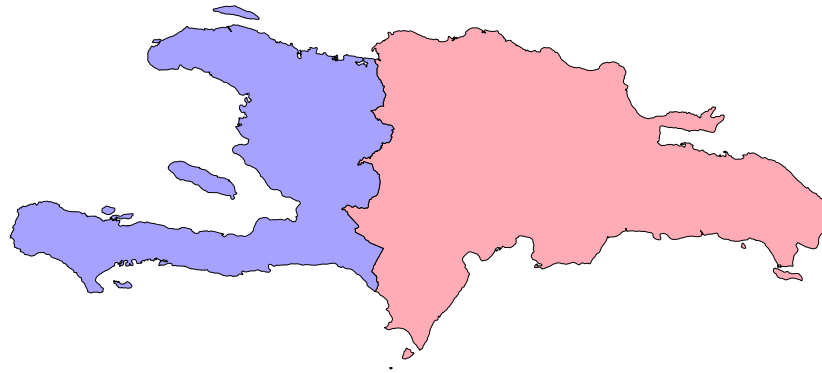
```
worldhi('Haiti')

ans =

              type: 'patch'
    otherproperty: {}
         altitude: []
              lat: [2011x1 double]
             long: [2011x1 double]
              tag: 'Haiti'
           latlim: [18.021 20.088]
          longlim: [-74.479 -71.613]
             area: [5x1 double]
          latlims: [5x2 double]
         longlims: [5x2 double]
            sname: 'Haiti'
```

You can request more than one country at a time by providing a string matrix or a cell array of strings.

```
worldhi({'Haiti','Dominican Republic'})

ans =

1x2 struct array with fields:
    type
    otherproperty
    altitude
    lat
    long
    tag
    latlim
    longlim
    area
    latlims
    longlims
    sname
```

The geographic data structures can be displayed on a map axes with `displaym`, or extracted to vectors of latitude and longitude with `extractm`.

```
axesm miller
displaym(worldhi({'Haiti','Dominican Republic'}))
polcmap
```



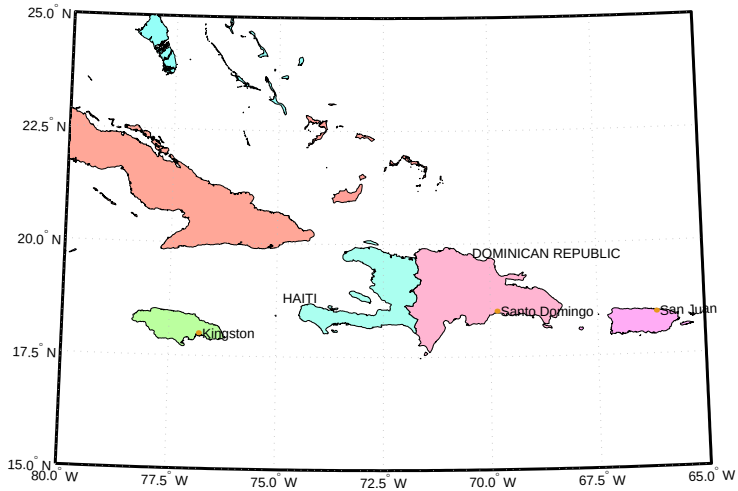
You can also have worldhi return all country outlines present within a geographic quadrangle. Here are the outlines for parts of countries that fall within the requested latitude and longitude limits. Note that worldhi will omit islands or the mainland of a country with bounding boxes that fall outside of the requested limits.

```
latlim = [15 25]; lonlim = [-80 -65];
s = worldhi(latlim,lonlim);
strvcat(s.tag)

ans =

Bahamas, The
Cayman Islands (UK)
Cuba
Dominican Republic
Haiti
Jamaica
Navassa Island (USA)
Puerto Rico (USA)
Turks and Caicas Islands (UK)
```

```
Virgin Islands, U.S. (USA)
```

While you can build up your own map displays using low level commands like `axesm` and `displaym`, the easy way to make a base map from this data is with the `worldmap` command. The `worldhi` command takes care cartographic details like selecting a projection, setting standard parallels, and grid and label increments. Here is a base map for the same area.

```
worldmap('allhi',latlim,lonlim,'patch')
hidem(gca)
```



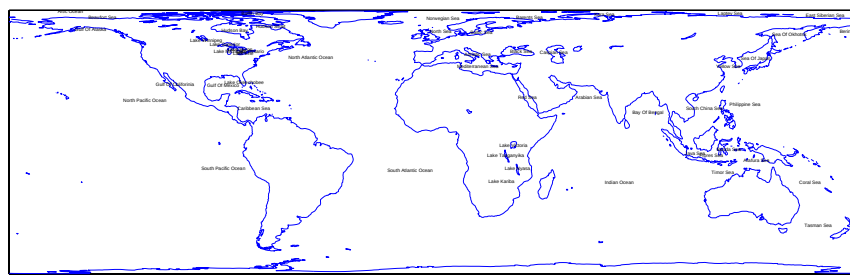
The optional arguments to `worldhi` can be used to omit islands smaller than some threshold, or to control how names are matched. You may have noticed the political affiliations in the names of some of the islands in the previous example. Here is an example that extracts all of entries in the `worldhi` database showing affiliation with the United Kingdom.

```
s = worldhi('(UK)','findstr');
strvcat(s.tag)

ans =

Alderney (UK)
Bermuda (UK)
```

```
British Indian Ocean Territory (UK)
Cayman Islands (UK)
Falkland Islands [Islas Malvinas] (UK)
Gibraltar (UK)
Gough Island (UK)
Guernsey (UK)
Jersey (UK)
Montserrat (UK)
Pitcairn Islands (UK)
Saint Helena (UK)
South Georgia and the South Sandwich Islands (UK)
Tristan da Cunha Group (UK)
Turks and Caicas Islands (UK)
United Kingdom (UK)
Virgin Islands, British (UK)
```

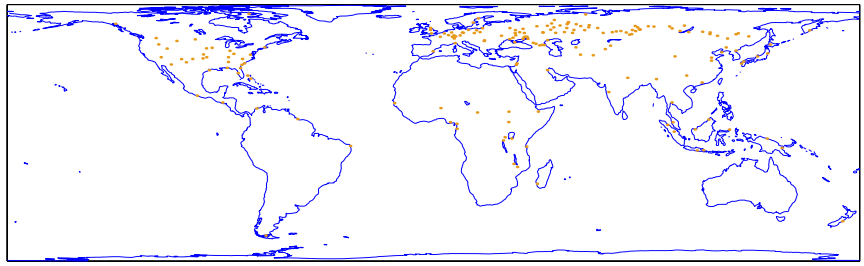
In addition to the country outline, the worldhi database also contains some information that complements the worldlo data. The POtext structure contains a list of ocean names that can be used to annotate map displays.

```
axesm lambcyln
plotm(coast)
tightmap
displaym(worldhi('POtext'))
set(handlem('alltext'),'fontsize',6)
```



The `worldhi` database also includes a set of city location markers and text labels. The `worldhi` populated place structures contain cities not in `worldlo`'s similar `PPpoint` and `PPtext` structures. Here the city markers in the `worldhi` data are shown in red, while the cities in `worldlo` are orange.

```
clmo all
h = displaym(worldhi('PPpoint'));
set(h,'color','r')
displaym(worldlo('PPpoint'))
plotm(coast)
```



World Matrix Data

Political

The Mapping Toolbox includes a set of matrix data coded with political information in the MAT-file `worldmtxmed`.

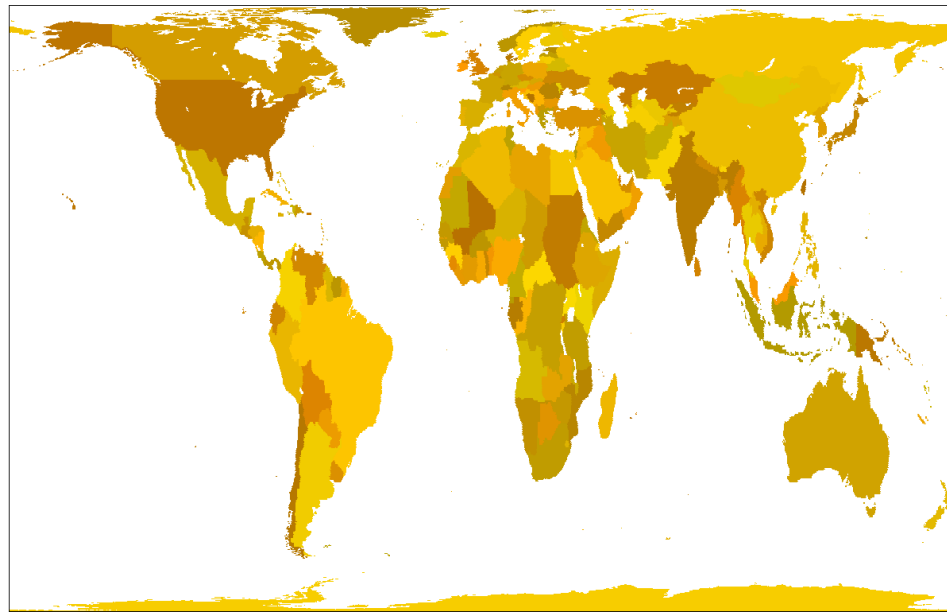
```
load worldmtx
whos
   Name       Size       Bytes     Class

   cmap       207x3      4968      double array
   map        720x1440   1036800   uint8 array
   maplegend  1x3        24        double array
   names      1x207      23140     cell array
```

The variable `map` is a regular matrix map containing indices of political entities. The `names` cell array relates the indices in the map to the names of countries, and the variable `cmap` is a colormap that provides a good political display of the world.

Display the political regular matrix map using the `meshm` function and the provided colormap:

```
axesm gortho
meshm(map,maplegend);
colormap(cmap)
tightmap
```

The map is shown in a Gall Orthographic projection, more recently made familiar as the Peters projection. This projection is equal-area, a property that is often desirable in representing competing political units.



The resolution of this data makes it suitable for global displays. Larger scale, regional maps are better made with vector data such as that in the `worldlo` or `worldhi` databases. More detailed political matrix maps can be created from vector data using `vec2mtx`, `country2mtx`, or the techniques described in Chapter 7, “GUI Tools”, of this document.

The primary use of this dataset is to determine which country contains a point known by its latitude and longitude. What country claims sovereignty over the point at 5 E and 13 N?

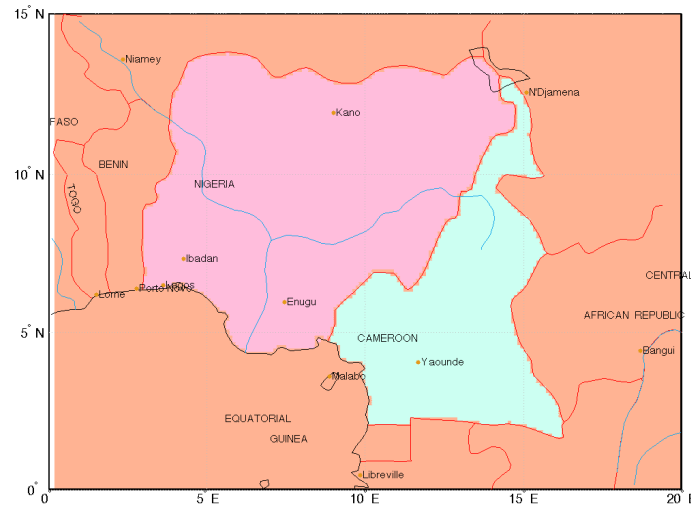
```
code = ltln2val(map,maplegend,5,13)
code =
    34

names{code}
ans =
Cameroon
```

Work through this example to see how the result was obtained. The matrix `map` contains integer country codes. By calling the function `ltln2val`, we extracted

the country code of the geographic point. The country code is an index into the names cell array, which returns the name of the country.

You can create your own matrix maps at higher resolutions. If the worldlo atlas data is good enough, use country2mtx. If you have higher resolution data, use vec2mtx. Here is a matrix map of Cameroon and Nigeria at a resolution of 10 cells per degree, or about 10 kilometers.

```
latlim = [0 15]; lonlim = [0 20];

[cmap,cmaplegend] = country2mtx('cameroon',10,latlim,lonlim);
[nmap,nmaplegend] = country2mtx('nigeria',10,latlim,lonlim);

newmap = zeros(size(cmap)); newmaplegend = cmaplegend;

newmap(cmap==0) = 34;  % find cells in the interior
newmap(nmap==0) = 130; % use codes from nations structure

worldmap(latlim,lonlim)
meshm(newmap,newmaplegend,size(newmap))
polcmap
```



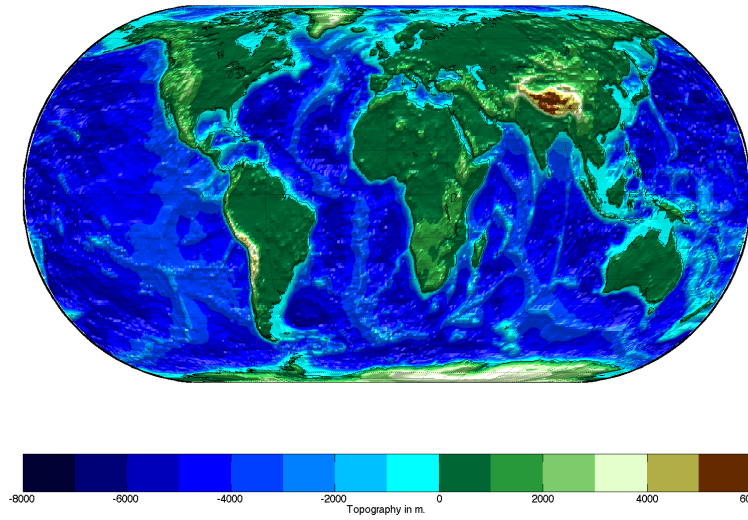
Terrain

A low resolution set of global elevation data is provided with MATLAB in the topo workspace. It is a regular matrix of average elevations and depths (in meters) for a constant grid spacing of one degree by one degree.

```
load topo
whos
   Name        Size       Bytes     Class

   topo        180x360    518400    double array
   topolegend  1x3        24        double array
   topomap1    64x3       1536      double array
   topomap2    128x3      3072      double array
```

Here it is displayed in an equal-area Eckert IV projection as a shaded-relief surface, which allows both large and small features to be seen:

```
axesm eckert4; framem; gridm
plotm(coast,'k'); zdatam(handlem('line'),max(topo(:)))

meshm(topo,topolegend,size(topo),topo,'FaceColor','interp')
demcmap('inc',topo,1000)

camlight(0,80);; material([0.7 0.8 .4]); daspectm('m',300)
hcb = colorbar('horiz')
set(get(hcb,'XLabel'),'String','Topography in m.')
```



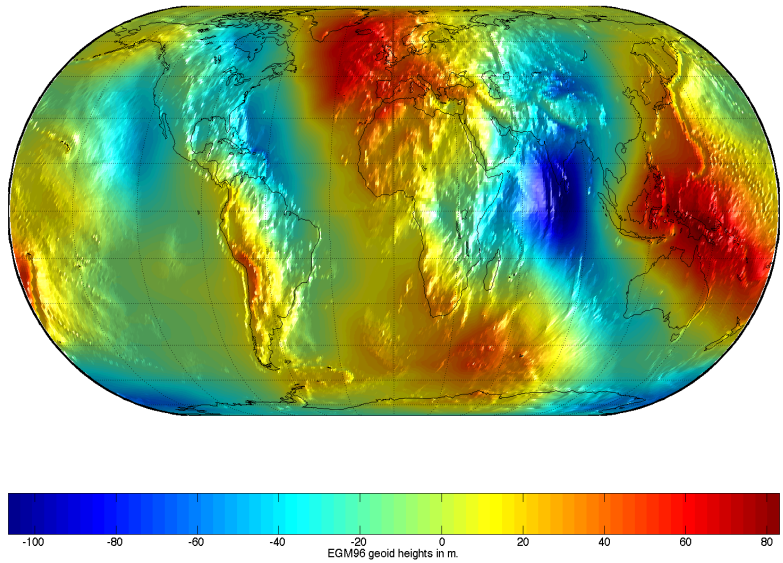
Geoid

While the `topo` matrix contains the heights and depths of the Earth's crust, the `geoid` matrix gives the shape of its gravitational field. The geoid can be regarded as the ocean surface formed in the absence of waves, currents, tides, and land obstructions. In technical terms, it is a gravitational equipotential surface. Elsewhere, this document refers to the ellipsoidal shape of the Earth as the geoid. Although this is sufficient for cartography, the ellipsoidal approximation is not accurate enough for applications like surveying and geodesy, and more detailed models are used. The EGM96 geoid model is a widely used spherical harmonic model of the geoid complete to degree and order 360. A 1-degree grid of geoid heights is provided in the `geoid.mat` atlas file. The full resolution version of the matrix of geoid heights with a grid spacing of 15 minutes can be read using the `egm96geoid` external interface function described in Chapter 8, "External Data Interface".

```
load geoid
figure; axesm eckert4; framem; gridm

plotm(coast,'k')
zdatam(handlem('line'),max(geoid(:)))
meshm(geoid,geoidlegend,size(geoid),geoid,'Facecolor','interp')
```

```
light; material(0.6*[ 1 1 1])

set(gca,'dataaspectratio',[ 1 1 200])
hcb = colorbar('horiz')
set(get(hcb,'Xlabel'),'String','EGM96 geoid heights in m.')
```

Here is the result:




```
z = ltln2val(geoid,geoidlegend,-11.1,130.22,'bicubic')
z =
   53.0055
```

Interpolating the full resolution grid gives a result that is about half a meter higher.

United States Vector Data

Low Resolution Data

Vector political and coastline data for the United States are provided in a number of useful formats in the `usalo` workspace.

```
load usalo
whos
```

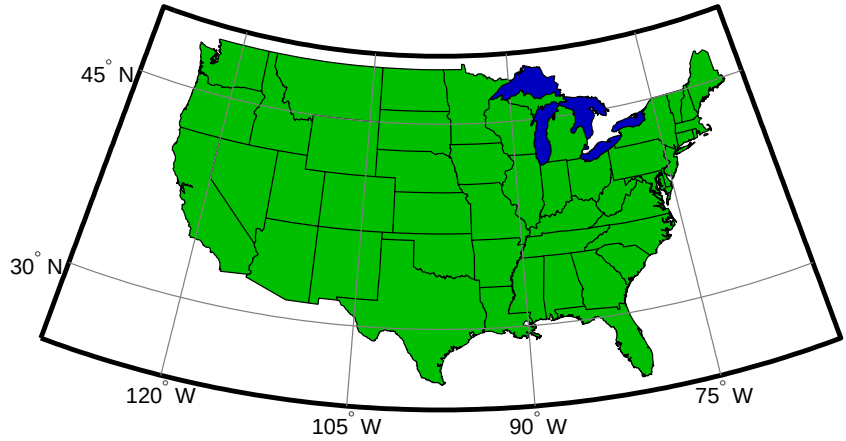
```
   Name        Size      Bytes     Class

   conus       1x1       71072     struct array
   greatlakes  1x3       22220     struct array
   gtlakelat   1229x1    9832      double array
   gtlakelon   1229x1    9832      double array
   state       1x51      256986    struct array
   stateborder 1x1       38390     struct array
   statelat    2345x1    18760     double array
   statelon    2345x1    18760     double array
   uslat       4339x1    34712     double array
   uslon       4339x1    34712     double array
```

There are several types of data here. The double arrays are vectors of latitude and longitude points. The pair `uslat` and `uslon` form the outline of the coast and political borders of the continental United States. The polygon closes on itself, so it may be filled as a patch. Next, the `statelat` and `statelon` form the borders between the states for display as lines. Finally, `gtlakelat` and `gtlakelon` are patchable outlines of the Great Lakes. The commands suitable for displaying vector data of this type are `plotm`, `linem`, `fillm`, `patchm`, and `patchesm`.

Display the data vectors as patches and lines:

```
axesm('MapProjection','eqaconic','MapParallels',[29.5 49.5],...
   'MapLatLimit',[24 50],'MapLonLimit',[-130 -65],...
   'MLabelLocation',15,'MLineLocation',15,...
   'PLabelLocation',15,'PLineLocation',15,...
   'GColor',.5*[1 1 1],'GLineStyle','-',...
   'MLabelparallel','south')
framem; gridm; mlabel; plabel
```

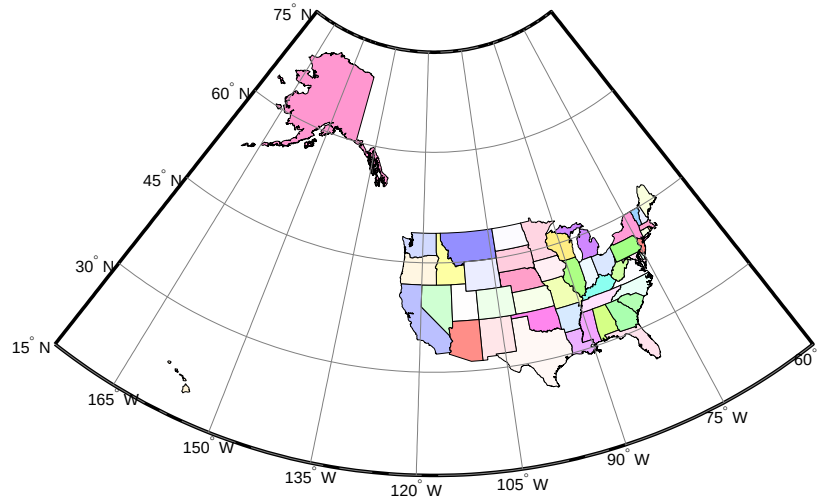
```
patchm(uslat,uslon,[0 .75 0])
patchm(gtlakelat,gtlakelon,1,[0 0 .75])
plotm(statelat,statelon,'k')
```



The structure arrays are Mapping Toolbox geographic data structures and can be displayed using `mlayers` or `displaym`. The data in `stateborder` is identical to that in `statelat` and `statelon`. Similarly, `conus` contains the same outline of the continental United States as `uslat` and `uslon` but is defined to be displayed as a patch. The 50 state and District of Columbia patches are located in the `state` structure.

Display the `state` structure using `displaym`. Notice the only one command is needed to plot 51 different patches.

```
figure
axesm('MapProjection','eqaconic','MapParallels',[],...
   'MapLatLimit',[15 75],'MapLonLimit',[-175 -60],...
   'MLabelLocation',15,'MLineLocation',15,...
   'PLabelLocation',15,'PLineLocation',15,...
   'GColor',.5*[1 1 1],'GLineStyle','-',...
   'MLabelParallel','south')
framem; gridm; mlabel; plabel
displaym(state); polcmap
```

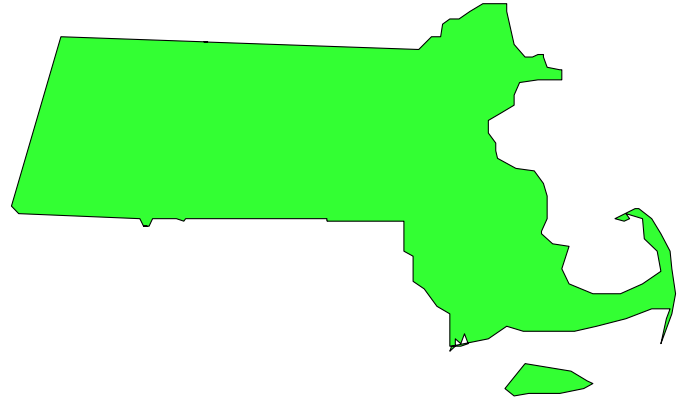


The states are all tagged, allowing them to be manipulated or to be extracted by name. You can see the available state names by entering the following:

```
strvcat(state.tag)
```

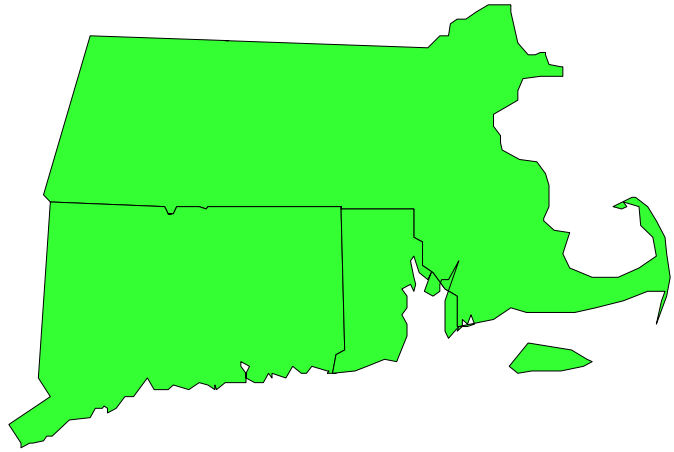
The state of Massachusetts can be displayed by itself:

```
displaym(state,'mass');
```



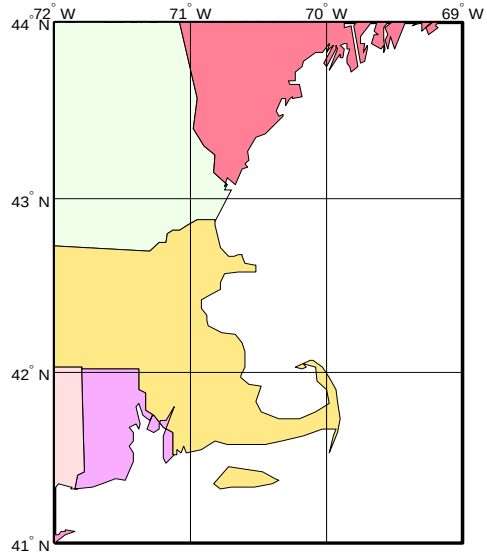
Or with its neighboring southern states, Connecticut and Rhode Island:

```
displaym(state,strvcat('mass','conn','rhode'));
```



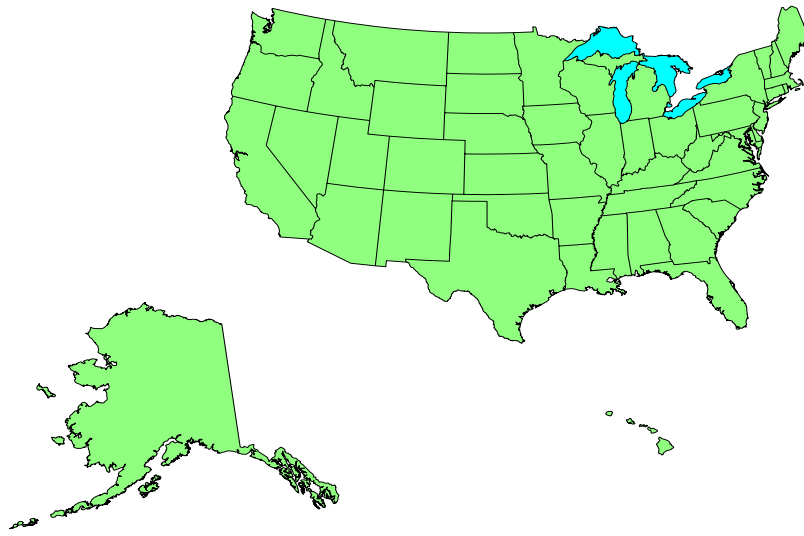
Look at the region around Cape Cod again, to get an impression of the detail of the data:

```
figure
axesm('MapProjection','mercator','Frame','on',...
   'MapLatLimit',[41 44],'MapLonLimit',[-72 -69])
gridm('MLineLocation',1,'PLineLocation',1,...
   'GLineStyle','-')
mlabel('MLabelLocation',1)
plabel('PLabelLocation',1)
displaym(state); polcmap
```



There is an obvious compromise between the size of a dataset and its accuracy and detail. This data can be displayed relatively quickly for displays of all the United States, but it is not suitable for mapping smaller regions. Note the generalized character of the coastline and the absence of small islands. More detailed and accurate data is available in the `usahi` workspace and through the Mapping Toolbox External Data Interface functions.

An easy way to display the `usalo` data is with the `usamap` function. The `usamap` function takes care of the cartographic details. The inset maps of the entire United States are made with the `usalo` data. Here are the fifty states displayed at the same scale.



Medium Resolution State Outlines

Another more detailed set of state outlines is available in the Mapping Toolbox. The data in the `usahi` MAT-file is similar in format to the `state` structure found in `usalo` but contains more detailed coastlines and islands. The resolution of `stateline` in the `usahi` workspace is about three times greater than the corresponding `state` in the `usalo` workspace.

```
load usahi
whos
```

```
  Name       Size      Bytes        Class

  statelin   1x51      837656       struct array
  statepatch 1x51      837758       struct array
  statetext  1x51      51190        struct array
```

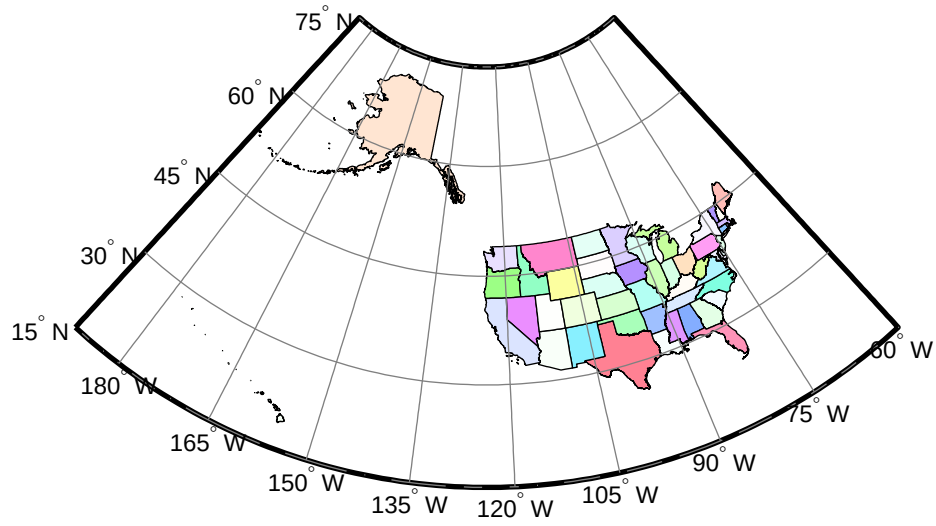
The listed variables are all formatted geographic data structures containing lines, patches, or text. The data in the `stateline` structure is the same as that

in `statepatch` but has been defined to display as lines rather than patches. The structure `statetext` contains the corresponding names of the states. Note that because of the higher resolution of this data, it may require more than the default memory size to display and will take longer to project and render. This difference is particularly marked for patches.

Display the map in an Equal-Area Conic projection:

```
axesm('MapProjection','eqaconic','mapparallels',[],...
   'MapLatLimit',[15 75],'MapLonLimit',[-188 -60],...
   'MLabelLocation',15,'MLineLocation',15,...
   'PLabelLocation',15,'PLineLocation',15,...
   'GColor',.5*[1 1 1],'GLineStyle','-',...
   'MLabelParallel','south')
framem; gridm; mlabel; plabel
displaym(statepatch); polcmap
```

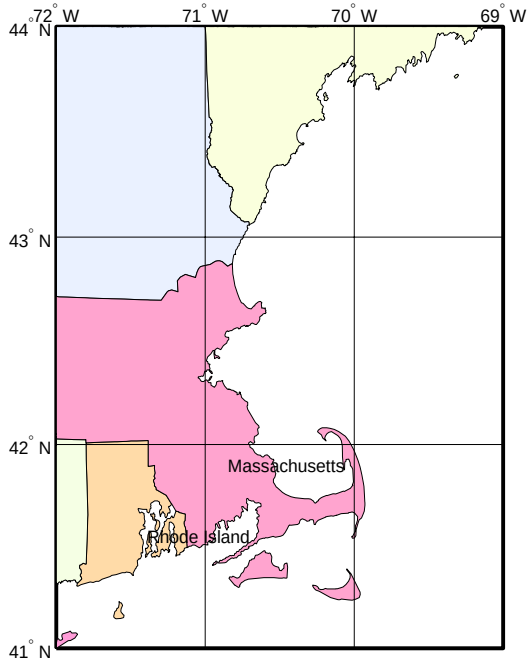
Here is the result:



At this scale, it is difficult to see the difference between this data and the much smaller `usalo` workspace. By focusing on the Cape Cod region, the higher resolution becomes more apparent:

```
figure
axesm('MapProjection','mercator','Frame','on',...
   'MapLatLimit',[41 44],'MapLonLimit',[-72 -69])
gridm('MLineLocation',1,'PLineLocation',1,'GLineStyle','-')
mlabel('MLabelLocation',1)
plabel('PLabelLocation',1)
displaym(statepatch)
h = displaym(statetext); trimcart(h)
polcmap
```

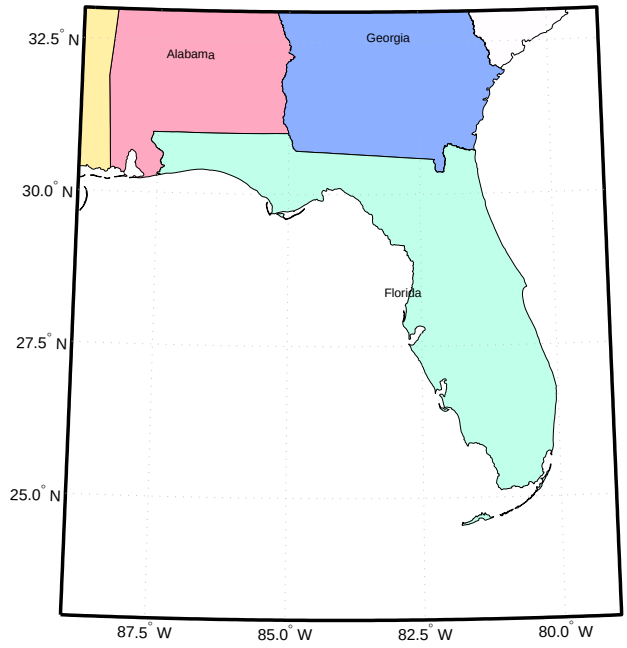
Here is the zoomed-in region of Cape Cod. The level of detail is noticeably greater than that for the data in the usalo MAT-file.



You can use usamap to quickly create base maps with this data. Just provide the state name or geographic limits. This function will handle the cartographic details like selecting a projection, setting the map limits, and setting the origin and the grid and label spacings. You can also use the usahi interface function

to load one of the atlas data structures as an argument to a function, or load just that structure into the workspace.

```
figure
usamap florida
```



United States Matrix Data

Political

The Mapping Toolbox also provides a set of political matrix data for the continental United States located in the `usamtx` MAT-file.

```
load usamtx
whos
```

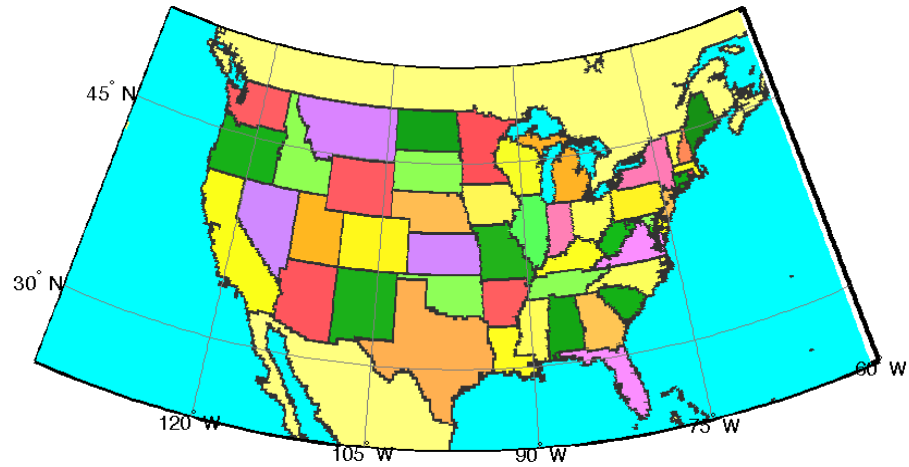
```
   Name       Size       Bytes     Class

   clrmap     54x3       1296      double array
   map        140x370    414400    double array
   maplegend  1x3        24        double array
   states     54x15      1620      char array
```

The data is in a regular matrix map format, similar to that discussed in the "World Matrix Data" section. The variable `map` is at a resolution of 5 cells per degree, or roughly 20 kilometers or better. Recall that the resolution of the `worldmtx` data is 1 cell per degree; hence, this dataset is five times finer. A colormap has been provided, along with a vector of state names, both corresponding to the political code, or *index*, used in the `map` matrix.

Display the matrix map in an Albers Equal-Area Conic projection:

```
axesm('MapProjection','eqaconic','MapParallels',[],...
   'MapLatLimit',[24 52],'MapLonLimit',[-134 -60],...
   'MLabelLocation',15,'MLineLocation',15,...
   'PLabelLocation',15,'PLineLocation',15,...
   'GColor',.5*[1 1 1],'GLinestyle','-',...
   'MLabelParallel','south')
framem; gridm; mlabel; plabel
meshm(map,maplegend); colormap(clrmap)
```

Here is the result:



As with the world matrix data, the low resolution of this data makes it suitable for only large area displays. Vector maps are more appropriate where more detail is required. Chapter 7, “GUI Tools” describes how to create higher resolution matrix maps from vector maps.

The primary use of this dataset is to determine which state contains a particular geographic point. In what state is Alamogordo (32.8990 N and 105.9570 W)?

```
code = ltln2val(map,maplegend,32.8990,-105.9570)
code =
   34

states(code,:)
ans =
New Mexico
```

You can create your own higher resolution political matrix maps using `vec2mtx`.

Terrain

As an example of a higher resolution digital elevation map, MATLAB provides the `cape` workspace, containing an image of elevation data for the northeastern

United States on a 30 arc-second grid (resolution of about one kilometer or better on the ground). The data can be defined as a regular matrix map using the loadcape function script, which rearranges the data and provides the necessary map legend:

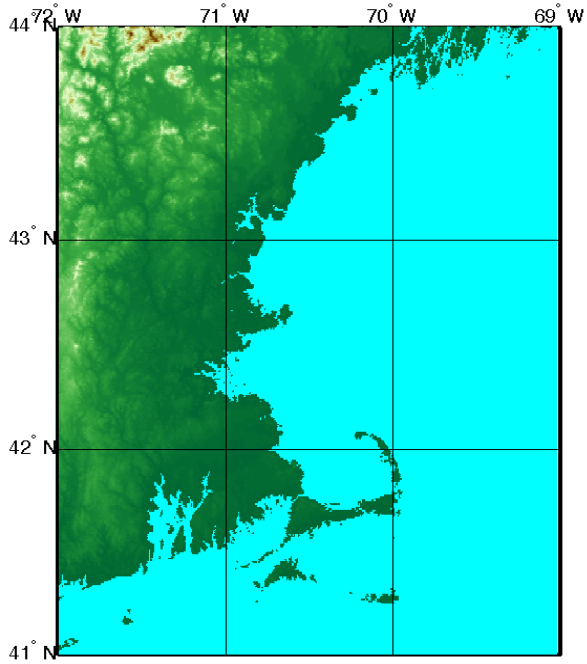
```
loadcape
whos
```

```
   Name       Size       Bytes     Class

   caption    2x55       220       char array
   cmap       192x3      4608      double array
   map        360x360    1036800   double array
   maplegend   1x3       24        double array
```

Here we display the elevation data using a conformal Mercator projection, so shapes of small regions suffer little distortion, while the distortion in relative areas is scarcely noticeable for such a small region.

```
axesm('MapProjection','mercator',...
    'MapLatLimit',[41 44],'MapLonLimit',[-72 -69])
framem
gridm('MLineLocation',1,'Plinelocation',1,'GLineStyle','-')
mlabel('MLabelLocation',1)
plabel('PLabelLocation',1)
meshm(map,maplegend); demcmap(map)
```

Here is the result:



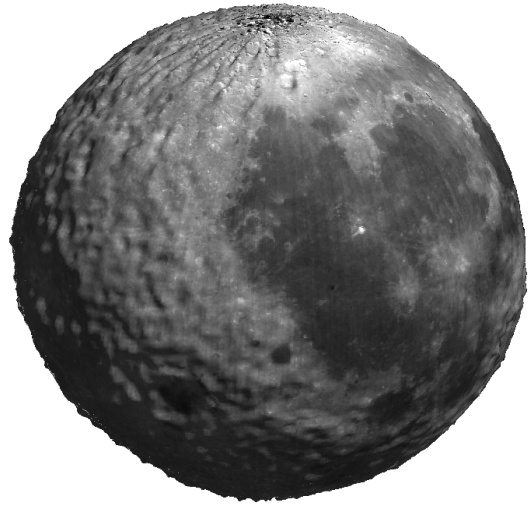
Global coverage of digital terrain and bathymetry at this resolution is provided through the Mapping Toolbox External Data Interface. A variety of freely available digital elevation maps is available over the Internet for import into MATLAB. These maps range in resolution from about 10 km to 30 meters. See Chapter 8, “External Data Interface” for more information on the data and import functions.

Astronomical Data

Although the Earth may be the most commonly mapped object, the same cartographic techniques are used to map the stars and planets. As an example of such astronomical data, the Mapping Toolbox includes map data for the Earth's moon and the stars.

The moon data is low resolution topography and albedo (reflectance) from the Clementine spacecraft. Topography, found in the `moontopo.mat` file, is a 1-degree regular matrix map in units of meters. Albedo is stored as a 3-cells-per-degree integer image in the `moonalb.mat` file. The `loadmoonalb` script reads the albedo image and converts it to double precision numbers for display. Here is the albedo image texture mapped onto the topography, vertically exaggerated by a factor of 10.

```
load moontopo; loadmoonalb
axesm('globe','geoid',almanac('moon','radius','m'))

h = meshm(moontopo,moontopolegend,size(moontopo),10*moontopo);
set(h,'CData',moonalb,'FaceColor','texturemap')
colormap(gray); view(20,20)
camlight; material dull; lighting phong
```



Star positions and magnitudes are found in the stars MAT-file.

```
load stars
whos
```

```
   Name         Size       Bytes     Class

   glat         4652x1     37216     double array
   glong        4652x1     37216     double array
   lat          4652x1     37216     double array
   long         4652x1     37216     double array
   relintensity 4652x1     37216     double array
   vmag         4652x1     37216     double array
```

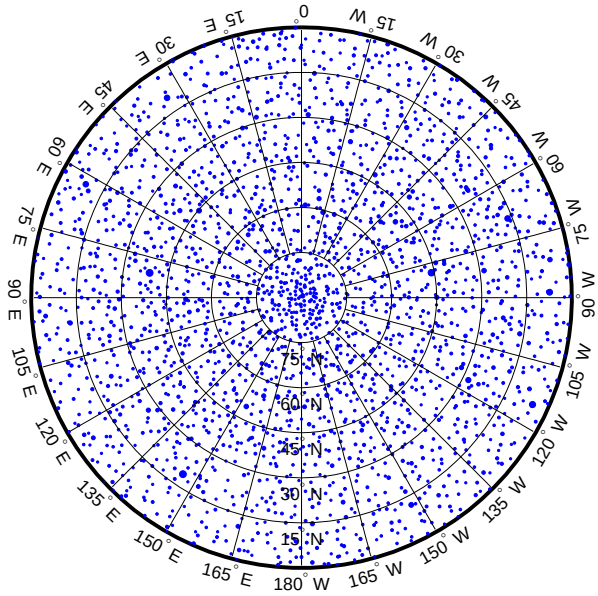
This is a set of more than 4500 star locations and visual magnitudes derived from the Fifth Fundamental Catalog of Stars parts I and II (FK5 and FK5e). Other star data such as mean errors, proper motions, spectral types, parallaxes, radial velocities, and cross identifications to other catalogues can be obtained from the catalog using the `readfk5` interface function described in Chapter 8, “External Data Interface”.

The positions of stars in equatorial coordinates are given in the vectors `lat` and `long`. These are latitudes and longitudes in degrees for the stars as seen from within the celestial sphere. The visual magnitude of each star is given in the vector `vmag`. Stars are typically plotted with diameters proportional to the relative intensity. The vector `relintensity` contains modified `vmag` data suitable for display with `scatterm` to exhibit this proportional diameter. All intensities are positive, and brighter stars have a larger `relintensity` number.

Display the stars of the northern sky in the Equidistant azimuthal projection. Scale the realive intensities to:

```
axesm eqdazim
framem; gridm; mlabel; plabel
setm(gca,'Origin',[90 180 0],'FLatLimit',[-inf 90],...
   'MLabelLocation',-180:15:165,'MLineLocation',15,...
   'MLineLimit',[-75 75],'MLabelParallel','equator',...
   'PLabelMeridian',180,'PLabelLocation',15:15:75,...
   'LabelRotation','on''GLineWidth',.01,...
   'GLineStyle','-')
set(handlem('alltext'),...
```

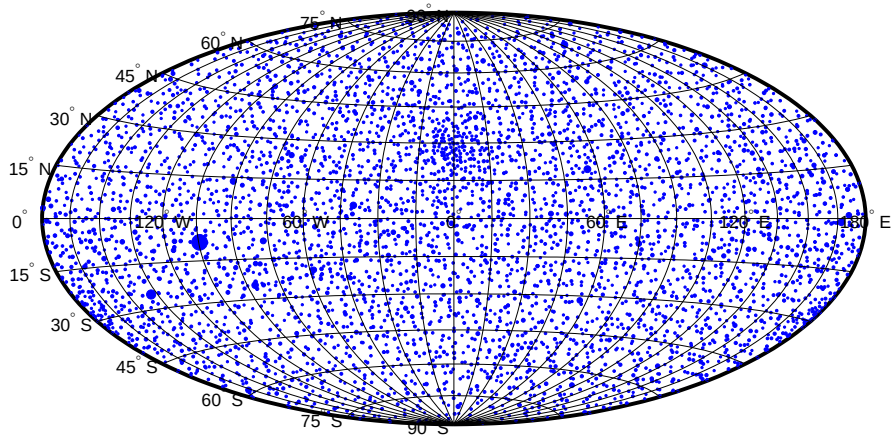
```
   'VerticalAlignment','top','HorizontalAlignment','center')
h = scatterm(lat,long,sqrt(relintensity)*30,'filled');
```



You may be able to identify the Big Dipper, or the constellation of Ursa Major, located at 180 E, 60 N.

Another system used to describe the positions of heavenly bodies is galactic coordinates. This coordinate system puts the center of our galaxy at the origin and places the north pole so that the Milky Way is aligned with the galactic equator. The positions of the stars in this coordinate system are given in the vectors `glat` and `glong`. Stars in galactic coordinates are typically plotted in a pseudo-cylindrical projection like the Hammer:

```
figure
axesm hammer
framem; gridm; mlabel; plabel
setm(gca,'GLineWidth',.01,'GLineStyle','-',...
   'MLineLocation',15,'MLabelParallel','equator',...
   'MLabelLocation',-120:60:180)
scatterm(glat,glong,sqrt(relintensity)*30,'filled')
```



See Chapter 8, “External Data Interface” for more information on how to convert between these two systems.

5

Map Projections

What Is a Projection? 5-2

Quantitative Properties 5-3

Geometric Surfaces 5-4

Projection Aspect 5-7
The Origin Vector 5-7
Coordinate Transformations 5-15

Projection Computations 5-19

Working with the Globe Projection 5-23

Working with the UTM projection 5-26

Summary Guide 5-31

What Is a Projection?

It has long been established that the shape of the Earth resembles a sphere and not a flat surface. If the world were indeed flat, cartography would be a trivial matter and map projections would be quite unnecessary.

To create a two-dimensional representation of a curved surface such as the Earth, you must determine how to transform that surface to a plane. The method of this transformation is called a *map projection*, from the geometric methods that were traditionally used to construct maps. While many map projections no longer rely on physical projections, it is useful to think of map projections in geometric terms.

Quantitative Properties

A sphere, unlike a cone or cylinder, cannot be simply reformed into a plane. In order to view the surface of a round body on a two-dimensional flat plane, you must first define a developable surface (i.e., one that can be *cut* and *flattened* onto a plane without stretching) and devise a means of systematically representing all or part of the spherical surface onto the plane. This inevitably leads to distortions of one kind or another. Three characteristic properties of map projections are subject to distortion: area, shape, and scale. A given projection cannot retain more than one of these properties over a large area of the Earth.

An *equal-area* projection is one in which equal spatial units at different points on the map represent the same true area measurement. For example, consider a coin of any size as our spatial unit. The area of the Earth covered by the coin on one part of the map display is exactly equal to that covered by the coin on any other part. Shape and scale must be distorted, although there are some equal-area maps designed to minimize these distortions for at least parts of the map and perhaps preserve shape or scale at specific points.

Other terms for equal-area projections include *equivalent*, *homolographic* or *homalographic* (from the Greek *homalos* or *homos*, same, and *graphos*, write), *authalic* (from the Greek *autos*, same, and *ailos*, area), and *equireal*.

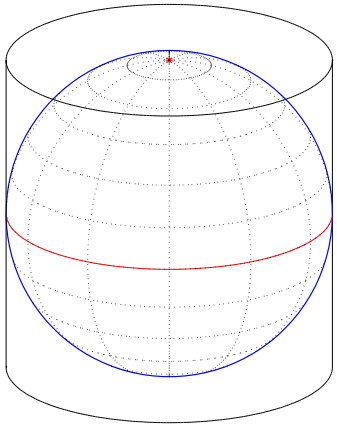
A *conformal* projection is one that preserves the relative local angles (shape) about every point on the map, except for one or more possible *singular* points. Since relative local angles are correct, meridians intersect parallels at right angles (90 degrees). No map can be both equal-area and conformal. Another term for conformal is *orthomorphic* (from the Greek *orthos*, straight, and *morphe*, shape).

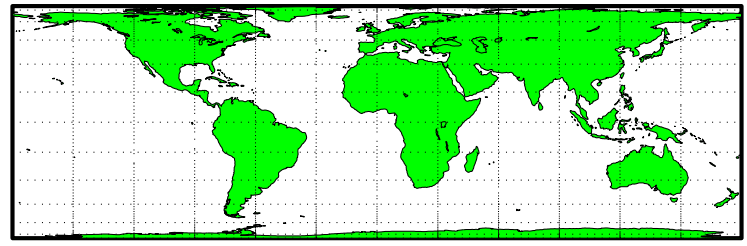
An *equidistant* projection shows true scale between one or two points and every other point on the map, or along every meridian. No map projection shows scale correctly in all directions throughout the entire map.

Geometric Surfaces

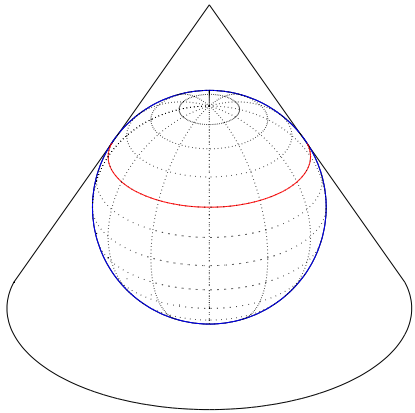
Three standard types of geometric surfaces are used to develop map projections: the cylinder, the cone, and the plane. A few projections, however, cannot be categorized as such, or are combinations of these. The three classifications are used for a wide variety of projections, including some that are not geometrically constructed.

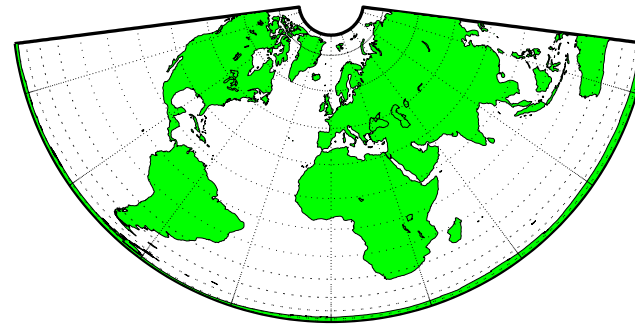
A *cylindrical* projection is produced by wrapping a cylinder around a globe representing the Earth. The map projection is the image of the globe projected onto the cylindrical surface, which is then unwrapped into a flat surface. Parallels appear as horizontal lines and meridians as vertical lines. The following shows a regular cylindrical or *normal aspect* orientation in which the cylinder is tangent to the Earth along the Equator and the projection is calculated horizontally from the axis.



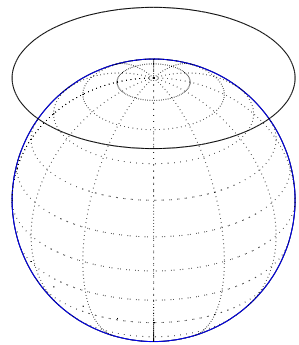


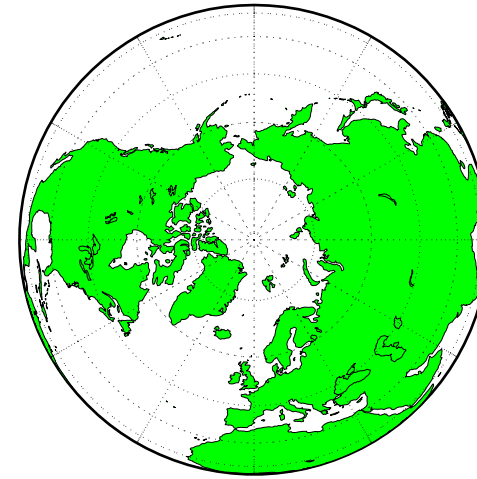
A *conic* projection is derived from the projection of the globe onto a cone placed over it. For the *normal aspect*, the apex of the cone lies on the polar axis of the Earth, and the surface of the cone touches the globe along a particular parallel of latitude. A *polyconic* projection considers each band of parallels as part of separate cones, each tangent to the globe at that particular parallel latitude.





An *azimuthal* projection is a projection of the globe onto a plane. For a polar azimuthal projection, the plane is tangent to the Earth at one of the poles, with meridians projected as straight lines radiating from the pole, and parallels shown as complete circles centered at the pole.





Each of these projection types can be further modified:

- Instead of being tangent to one standard parallel, the cylindrical or conic surface may cut the globe at two parallels. In like manner, the plane for an azimuthal projection may intersect the globe instead of being tangent.
- The vantage point of the projection onto the geometric surface may be changed. The perspective may be chosen from the center of the Earth or along the polar axes (*central* or *gnomonic*), from the opposite side of the Earth's surface (*stereographic*), or from an infinite point in space (*orthographic*).

Projection Aspect

Until now, the discussion of map displays has focused on the *normal* aspect, by far the most commonly used. This section discusses the use of *transverse*, *oblique* and, *skew-oblique* aspects.

Projection aspect is primarily of interest in the display of maps. However, this section will also discuss how the idea of projection aspect as a coordinate system transformation can be applied to map variables for analytical purposes.

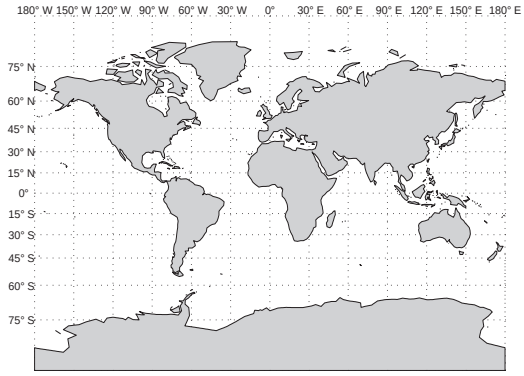
The Origin Vector

A map axes `Origin` property is a vector describing the geometry of the displayed projection. The form of this vector is of the following:

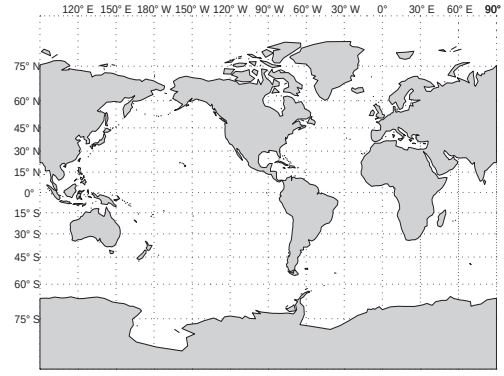
```
origvec = [latitude longitude orientation]
```

The latitude and longitude represent the geographic coordinates of the center point of the display from which the projection is calculated. The orientation refers to the angle from *straight up* at which the North Pole bears from this center point. The default origin vector is [0 0 0], that is, the projection is centered on the geographic point (0 ,0) and the North Pole is *straight up* from this point. Such a display is in a *normal* aspect. In fact, changes to only the longitude value of the origin vector do not change the aspect; a *normal* aspect is one centered on the Equator in latitude with an orientation of 0 .

Both of these Miller projections have normal aspects, despite having different origin vectors:



Origin at (0 ,0), with a 0 Orientation.
(origin vector = [0 0 0])



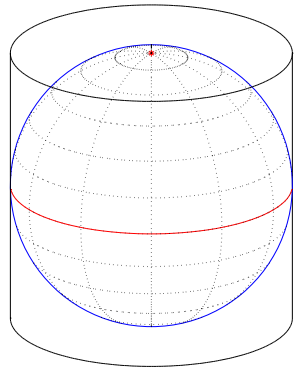
Origin at (0 ,90 W), with a 0 Orientation.
(origin vector = [0 -90 0])

This makes sense if you think about a simple, true cylindrical projection. This is the projection of the globe onto a cylinder wrapped around it. For normal aspects, this cylinder is tangent to the globe at the Equator, and changing the origin longitude simply corresponds to rotating the sphere about the longitudinal axis of the cylinder. If we continue with the wrapped-cylinder model, we can understand the other aspects as well.

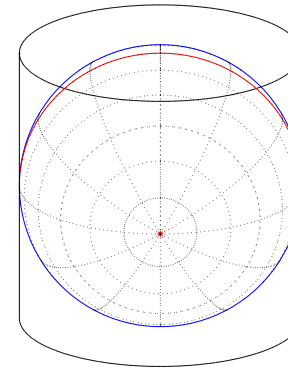
Following this description, a *transverse* projection can be thought of as a cylinder wrapped around the globe tangent at the poles and along a meridian. Finally, when such a cylinder is tangent along an arbitrary great circle, the result is an *oblique* projection.

Here are diagrams of the four cylindrical map orientations, or aspects:

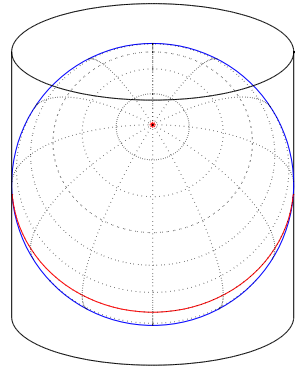
Normal



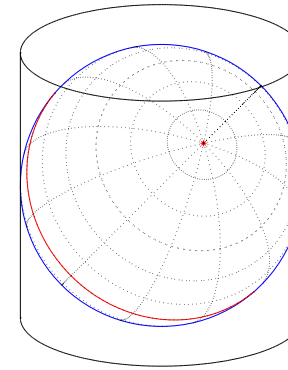
Transverse



Oblique

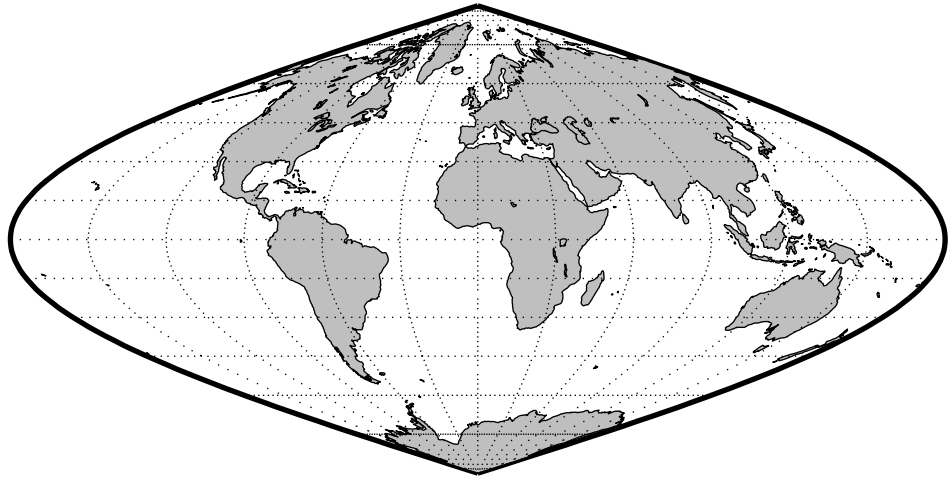


Skew-Oblique



Of course, few projections are actually true cylindrical projections, but the concept of the wrapped cylinder is nonetheless a convenient way to think about it. Perhaps the best way to gain an appreciation for projection aspect is to look at a few examples.

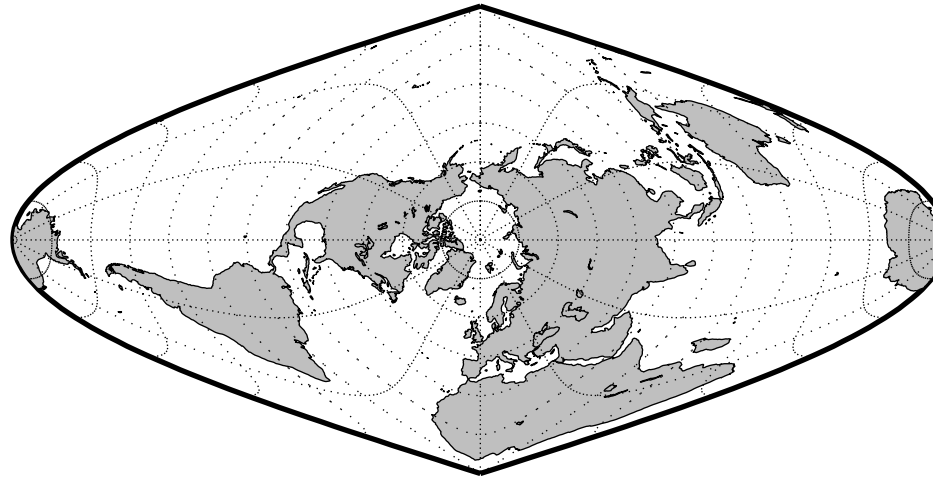
Here, a psuedocylindrical projection, the Sinusoidal, is used. First, look at the normal aspect of this projection:



Normal Aspect: Origin at (0 ,0), Orientation 0
(origin vector = [0 0 0])

In the normal aspect, the North Pole is at the *top* of the image. To create a transverse aspect, imagine *pulling* the North Pole down to the center of the image, which was originally occupied by the point (0 ,0).

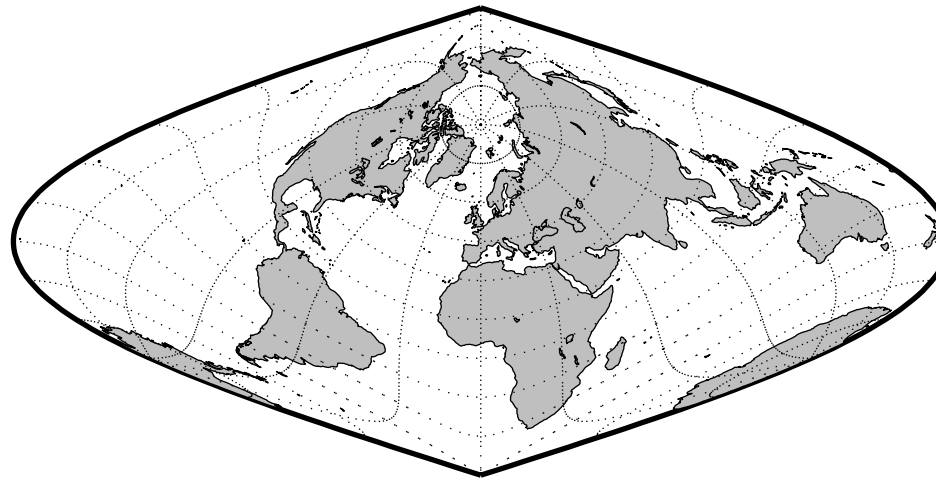
The shape of the frame is unaffected; this is still a Sinusoidal projection:



Transverse Aspect: Origin at (90 N,0), Orientation 0
(origin vector = [90 0 0])

The normal and transverse aspects can be thought of as limiting conditions. Anything else is an oblique aspect. Conceptually, if you push the North Pole halfway back to its original position, that is, to the position originally occupied by the point (45 N, 0) in the normal aspect, the result will be a simple oblique aspect.

Here is the oblique aspect Sinusoidal projection:

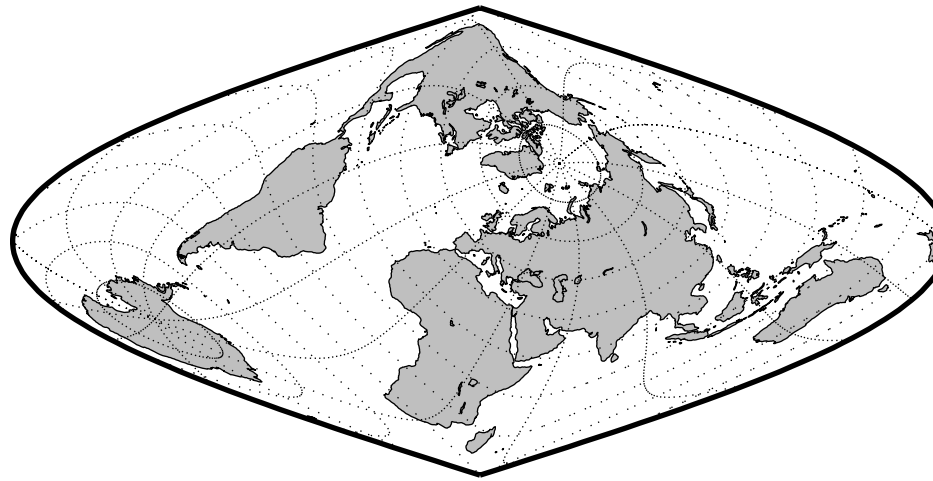


Oblique Aspect: Origin at (45 N,0), Orientation 0
(origin vector = [45 0 0])

Another way of understanding this is to imagine the new origin point (45 N,0) being pulled to the center of the image, to the point originally occupied by (0 ,0) in the normal aspect.

The examples so far have left the aspect orientation at 0 . If the orientation is altered, an oblique aspect becomes a *skew-oblique*. Imagine the previous example with an orientation of 45 . Conceptually, the skew-oblique aspect corresponds to pulling the new origin point, (45 N,0) down to the center of the projection and then rotating the projection until the North Pole lies at an angle of 45 clockwise from straight up with respect to the new origin.

Here is the result:



Skew-Oblique Aspect: Origin at (45 N,0), Orientation 45
(origin vector = [45 0 45])

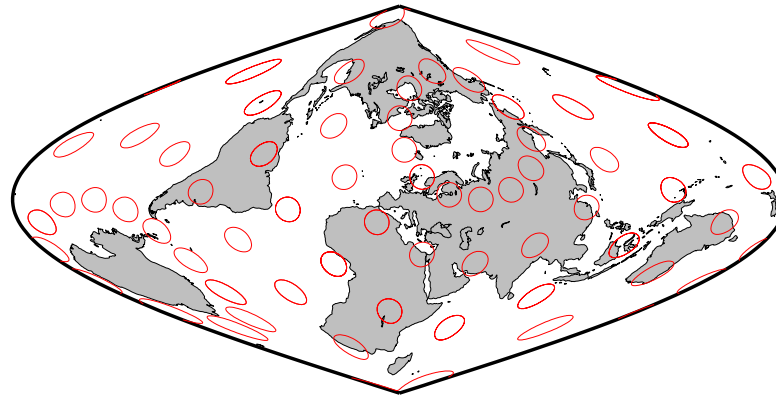
Any projection can be viewed in alternate aspects. Some of these are quite useful. For example, the transverse aspect of the Mercator projection is widely used in cartography, especially for mapping regions with predominantly north-south extent. One candidate for such handling might be Chile. Oblique Mercator projections might be used to map long regions that run neither north and south nor east and west, such as New Zealand.

Note The projection aspect discussed in this section is different from the map axes `Aspect` property. The map axes `Aspect` property controls the orientation of the figure axes. For instance, if a map is in a `'normal'` setting with a *landscape* orientation, a switch to a `'transverse'` aspect rotates the axes by 90 , resulting in a*portrait* orientation. To display a map in the transverse aspect, combine the `'transverse'` aspect property with a -90 skew angle The skew angle is the last element of the `Origin` vector. For example, a `[0 0 -90]` vector would produce a transverse map.

The features of a projection are maintained in any aspect, *relative to the base projection*. As we have seen, the *outline*, or frame doesn't change. Non-directional features also do not change. For example, the Sinusoidal projection is equal-area, and is so in any aspect. Directional features must be considered carefully, however. In the normal aspect of the Sinusoidal projection, scale is true along every parallel and the central meridian. This is not the case for the skew-oblique aspect; however, scale is true along the lines where these parallels and meridian would be in the corresponding base projection.

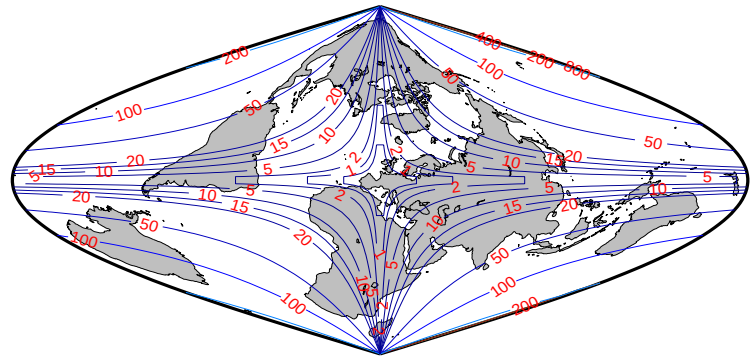
The base projection can be thought of as a standard coordinate system, and the normal aspect conforms to it. The other aspects can be thought of as coordinate transformations.

A standard method of visualizing the distortions introduced by the map projection is to display small circles at regular intervals across the globe. After projection, the small circles appear as ellipses of various sizes, elongations, and orientations. The sizes and shapes of the ellipses reflect the projection distortions. Conformal projections have circular ellipses, while equal area projections have ellipses of the same area. This method was invented by Tissot, and the ellipses are called Tissot indicatrices in his honor. You can add the indicatrices to a map display with the command `tissot` and remove them with `clmo Tissot`.



For a more quantitative look at the projection distortion, use `mdistort`. This function draws contour lines of constant scale, area, and angular distortion on the map. This is useful in selecting projections and projection parameters to

keep distortion within limits. Here are the lines of percent scale distortion for the skew-oblique Sinusoidal projection. The lines of zero distortion along the axes of the base projection are apparent in the contour plot.



Coordinate Transformations

Previously, we discussed the concept of altering the display aspect of a map projection. One way to think of this is to consider redefining the coordinate system, and then displaying a projection based on the new system. For example, think about redefining your coordinate system so that your hometown is the origin. If you calculated a map projection in a normal aspect with respect to this *transformed* coordinate system, the resulting display would look like an oblique aspect with respect to the *true* coordinate system.

This transformation of coordinate systems can be useful independent of map displays. If a transformed coordinate system is selected so that your home town is the new *north pole*, then the transformed coordinates of all other points will provide interesting information.

The distance of each point from your hometown is then simply 90 minus the transformed latitude. For example, the point antipodal to your hometown would be the transformed *south pole*. -90 . Its distance from your hometown is then 90 -(-90), or 180 , as expected. Points 90 distant from your hometown all have a transformed latitude of 0 . In fact, these points make up the transformed *equator*. The transformed longitudes correspond to the great circle azimuths of those points from your hometown.

When regular matrix maps are transformed in this manner, distance and azimuth calculations with the map variable reduce to row and column operations.

Vector Data – rotatem

As an example of transforming a coordinate system, suppose you live in Midland, Texas, at (32 N,102 W). You have a brother in Tulsa (36.2 N,96 W) and a sister in New Orleans (30 N,90 W).

```
midl_lat = 32;   midl_lon = -102;
tuls_lat = 36.2; tuls_lon = -96;
newo_lat = 30;   newo_lon = -90;
```

What can we say about their homes, relative to yours?

```
dist2tuls = distance(midl_lat,midl_lon,tuls_lat,tuls_lon)
dist2tuls =
   6.5032

dist2newo = distance(midl_lat,midl_lon,newo_lat,newo_lon)
dist2newo =
   10.4727
```

Tulsa is about 6.5 degrees distant, New Orleans about 10.5 degrees distant. The great circle azimuths are the following:

```
az2tuls = azimuth(midl_lat,midl_lon,tuls_lat,tuls_lon)
az2tuls =
   48.1386

az2neworl = azimuth(midl_lat,midl_lon,newo_lat,newo_lon)
az2neworl =
   97.8644
```

The absolute difference in these azimuths is 49.7258 , a fact we will use later.

Today, you feel on top of the world, so make Midland, Texas, the *north pole* of a transformed coordinate system. To do this, first determine the origin required to put Midland at the pole:

```
origin = newpole(midl_lat,midl_lon)
origin =
   58    78     0
```

The origin of the new coordinate system will be (58 N,78 E). Midland is now at a *new latitude* of 90 .

What are the *new coordinates* of Tulsa and New Orleans? To calculate these, use the `rotatem` command. Since it defaults to radians, be sure to include the units input `'degrees'`:

```
[tuls_lat1,tuls_lon1] = rotatem(tuls_lat,tuls_lon,...
                                origin,'forward','degrees')
tuls_lat1 =
   83.4968
tuls_lon1 =
   -48.1386

[newo_lat1,newo_lon1] = rotatem(newo_lat,newo_lon,...
                                origin,'forward','degrees')
newo_lat1 =
   79.5273
newo_lon1 =
   -97.8644
```

Now, test the new coordinate system for the advertised properties. The distance to Tulsa was 6.5032 distant. Its colatitude in the new coordinate system is 90 -83.4968 , or 6.5032 . Similarly, the new colatitude of New Orleans, 90 -79.5273 , is the distance 10.4727 .

Additionally, the absolute difference in the azimuths of the two cities from Midland was 49.7258 . The difference in their *new longitudes* is -48.1386 - (-97.8644), which is also 49.7258 , as advertised.

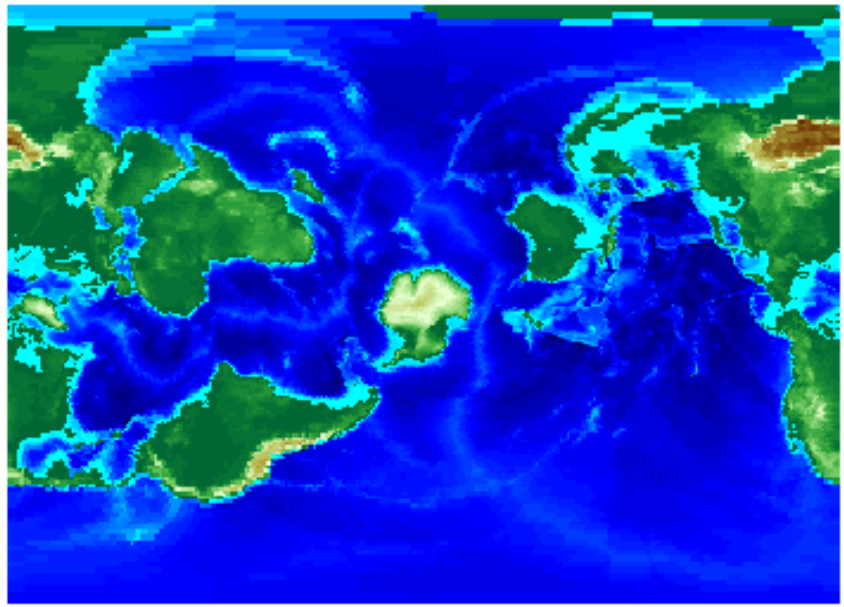
Matrix Data – neworig

A regular matrix map can be used to create a new general matrix map with its data rearranged to correspond to a new coordinate system. Suppose you want to transform the topo data to a new coordinate system in which Sri Lanka (7 N,80 E) is the *North Pole*.

```
load topo
origin = newpole(7,80);
[map,lat,lon] = neworig(topo,topolegend,origin);
```

Display the new map:

```
axesm miller
surfm(map,[30 30]); demcmap(topo)
```



An interesting feature of this new matrix map is that every cell in its first row is 0 -1 distant from the Sri Lankan point (7 N,80 E), and every cell in its second row is 1 -2distant, etc. Another feature is that every cell in a particular column has the same great circle azimuth from the point.

Projection Computations

Most of the examples in this document assume that the end product of a map projection is a graphical representation of the map. While this is true for most cases, there may be times when you need access to the non-geographic coordinates of your map data in the projected space.

An easy way to retrieve the projected coordinates of a map that has already been displayed is with the MATLAB `get` command. The projected coordinates are stored in the object's `XData` and `YData` properties. As an example, display a map frame for the Mollweide projection, and extract its x-y coordinates:

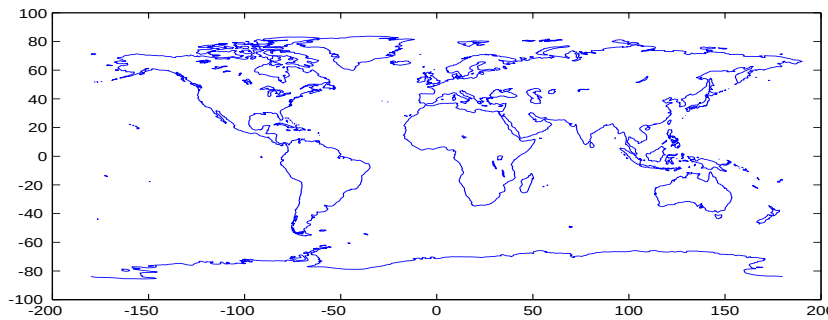
```
axesm mollweid
h = framem;
x = get(h,'XData');
y = get(h,'YData');
```

Of course, you do not need to display a map object to get its projected coordinates. You can perform the same projection computations that are done within the Mapping Toolbox display commands.

Begin by displaying a map of the `coast` vector data:

```
figure
load coast
plot(long,lat); axis equal
set(gca,'XLim',[-200 200],'YLim',[-100 100])
```

Here is the result. Note that some of the data extends beyond 180 longitude.



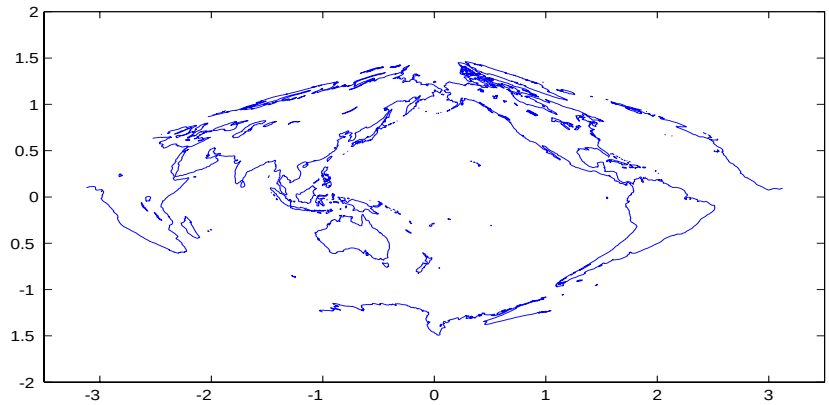
Before projecting the data, you must define the projection parameters, just as you would prepare a map axes with `axesm` before displaying a map. The projection parameters are stored in a map projection structure that normally resides in the `UserData` property of a MATLAB axes, but you can use it directly for the projection computation.

The following commands create an empty map projection structure for a Sinusoidal projection, change the map origin, and fill in the rest of the structure fields with default property values. Just as you can change the property settings of a map axes with `setm`, you can use the entries of the map projection structure to control the projection properties.

```
mstruct = defaultm('sinusoid');
mstruct.origin = [0 180 0];
mstruct = defaultm(sinusoid(mstruct));
```

Having defined the map projection parameters, you can project the latitude and longitude vectors into the proper coordinates of the Sinusoidal projection and display the result using non-mapping, MATLAB commands.

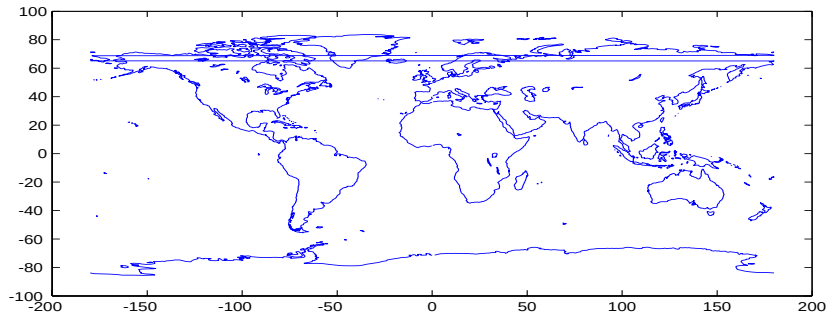
```
[x,y] = mfwdtran(mstruct,lat,long,[],'line');
plot(x,y); axis equal
set(gca,'XLim',[-3.5 3.5],'YLim',[-2 2])
```



For the transformation function, an empty matrix is used to take the place of altitude data, and the string 'line' identifies the object type and allows the vector data to be clipped and trimmed. Notice that the International Date Line is now at the center of the map (180), as specified earlier.

You can transform the projected x-y data back into the Greenwich geographic coordinates with the inverse transformation function. Plot the result:

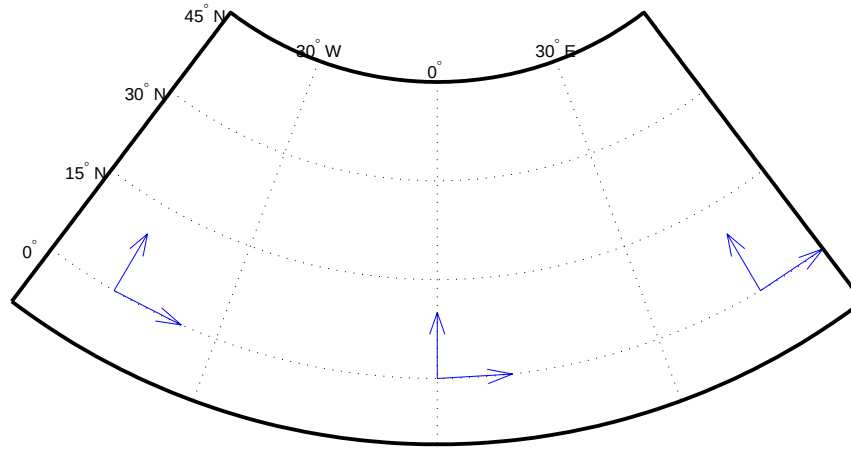
```
[lat2,long2] = minvtran(mstruct,x,y);
plot(long2,lat2); axis equal
set(gca,'XLim',[-200 200],'YLim',[-100 100])
```



What happened? Recall that the original data extended beyond 180 longitude. In the projection transformation process, longitude data outside `[-180 180]` degrees is projected back into this range since angles differing by 360 are geographically equivalent. The data from the inverse transformation process therefore jumps from 180 to -180 , as depicted by the straight horizontal lines in the figure above. The `smoothlong` function can be used to remove discontinuities in longitude.

In addition to projecting geographic positions into Cartesian coordinates, you can project angles between the sphere and the plane. For cylindrical projections, north maps to up on the y-axis, and east maps to right on the x-axis. This is not necessarily true of other projections. In conic projections, for example, north may be to the left or right of the y-axis, depending on the geographic coordinates.

What are the angles of north and east in projected coordinates for an equidistant conic? Geographic angles are measured clockwise from north, while projected angles are measured counterclockwise from the x-axis.

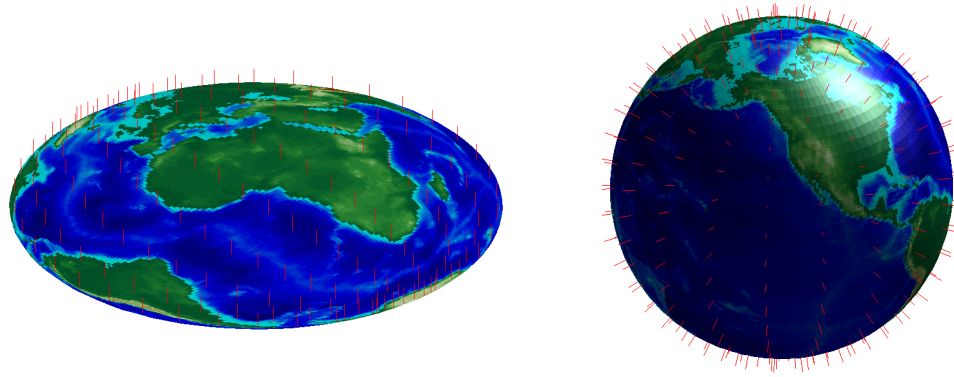



```
vfwdtran([0 0 0],[-45 0 45],[0 0 0])
ans =
       59.614           90       120.39

vfwdtran([0 0 0],[-45 0 45],[90 90 90])
ans =
      -30.385    0.0001931       30.386
```

Working with the Globe Projection

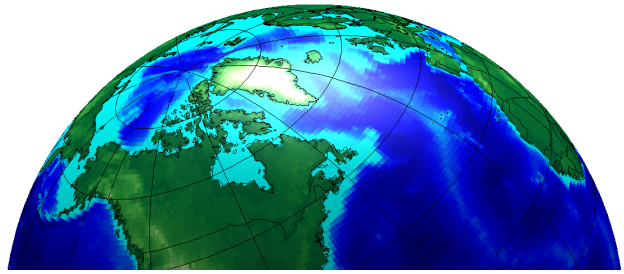
Projection is the process of transforming coordinates from the round spherical coordinate system to flat Cartesian coordinates. The basic assumption of most projections in the Mapping Toolbox is that latitudes and longitudes map to x- and y-coordinates. The single exception is the globe projection, which maps latitudes and longitudes to points on a sphere in Earth-centered Cartesian (x-,y-,z-) coordinates. This makes the globe useful in applications where the proper three-dimensional relationship between objects is important. The following examples illustrate the difference between the two-dimensional orthographic projection, which looks like a globe but is really flat, and the three-dimensional globe

```
load topo
axesm ortho; framem
meshm(topo,topolegend);demcmap(topo)
view(3); daspectm('m',1)

[latgrat,longrat] = meshgrat(topo,topolegend,[20 20]);
stem3m(latgrat,longrat,500000*ones(size(latgrat)),'r')

figure
axesm('globe','Geoid',almanac('earth','radius','m'))
meshm(topo,topolegend); demcmap(topo)
view(3)

[latgrat,longrat] = meshgrat(topo,topolegend,[20 20]);
stem3m(latgrat,longrat,500000*ones(size(latgrat)),'r')
light
```



Because the globe is three-dimensional, having an opaque surface underneath vector data helps to hide lines on the other side of the globe. The following example shows how you can display lines on an underlying surface and use camera position functions to control the view. To make the surface of the globe one color, change the `FaceColor` property of the `topo` surface.

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

load topo
hs = meshm(topo,topolegend,size(topo));
hl = displaym(worldlo('POline')); set(hl,'color','k')
demcmap(topo); camlight; material(0.6*[ 1 1 1])

[tlat,tlon] = extractm(worldlo('gazette'),'Moscow');
[plat,plon] = extractm(worldlo('gazette'),'Washington');

camtargm(tlat,tlon,0); camposm(plat,plon,3)
camupm(tlat,tlon); camva(20)

hidem(gca)
```

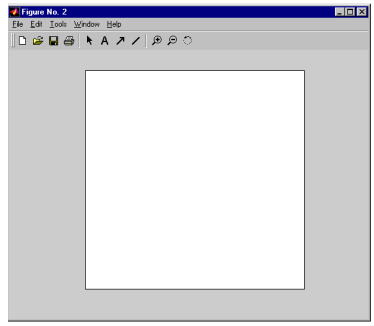


Working with the UTM projection

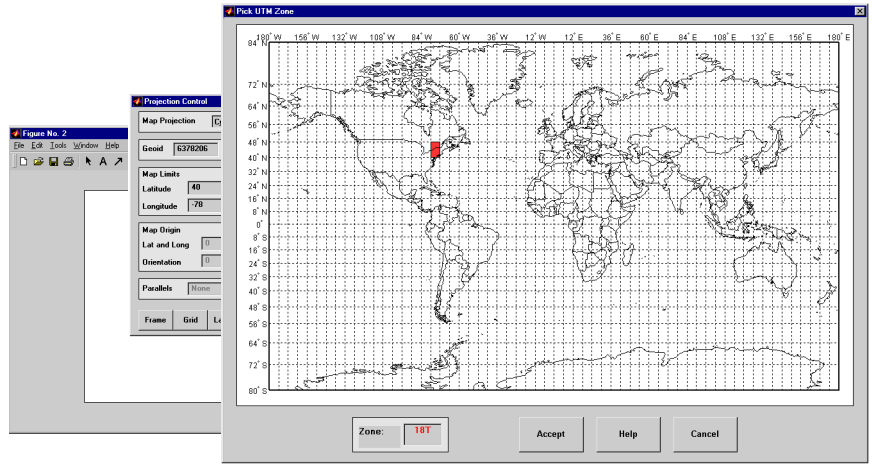
An important projection in scientific and surveying applications is the Universal Tranverse Projection, or UTM. The UTM projection divides the world into a regular grid of quadrangles, each 8 by 6 degrees in extent, called zones. Each zone uses formulas for a transverse version of the Mercator projection with projection and ellipsoid parameters designed to limit distortion. The UTM projection is defined between 80 degrees south and 84 degrees north. Outside these limits, the Universal Polar Stereographic (UPS) projection applies. The UPS has two zones, north and south, which also have special projection and ellipsoid parameters.

The easiest way to use the UTM projection is through graphical user interfaces. First create the map axes.

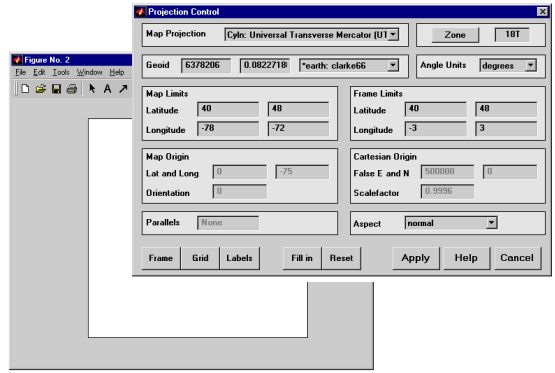
```
axesm utm
```



Then use the projection control panel to pick the zone for your area of interest. You can either extend-click on the map axes, or type `axesmui` to display the projection control panel. Click on the **Zone** button to open the `utmzoneui` panel. Click on the map near your area of interest to pick the zone.

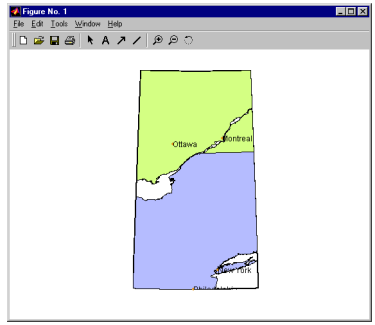


Clicking the **Accept** button closes the zone panel, and applies the chosen zone designation to the zone field. The geoid parameters are also automatically set to an ellipsoid definition that is recommended for that zone. You can override the default choice by selecting another ellipsoid from the list, or by typing the parameters in the **Geoid** field. Clicking **Apply** closes the projection control panel.



The projection is then ready for projection calculations or map display commands. Here is some basemap data from the `worldlo` and `worldhi` atlas data.

```
displaym(worldhi([40 48],[-78 -72]))
polcmap
framem
displaym(worldlo('PPpoint'))
displaym(worldlo('PPtext'))
trimcart alltext; tightmap
hidem(gca)l set(gcf,'color','w')
```



You can also calculate projected UTM grid coordinates from latitudes and longitudes.

```
[x,y] = mfwdtran(40.5,-73.5)

x =

     627106.47


y =

    4484124.43
```

Of course, you can set up a UTM projection to calculate coordinates without a map display. There are commands to help you pick the zone designation and an appropriate ellipsoid.

```
utmzone(40.5,-73.5)

ans =

18T

[ellipsoid,estr] = utmgeoid('18T')

ellipsoid =

    6378206.40          0.08


estr =

clarke66
```

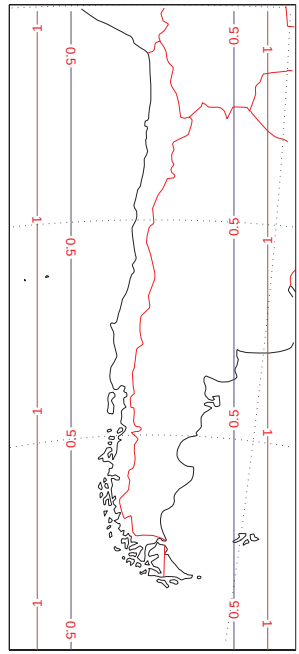
Set up the UTM projection and calculate the grid coordinates without a map display.

```
close all
mstruct = defaultm('utm');
mstruct.zone = '18T';
mstruct.geoid = almanac('earth','geoid','m','clarke66');
mstruct = defaultm(utm(mstruct));
[x,y] = mfwdtran(mstruct,40.5,-73.5)

x =

     627106.47


y =

    4484124.43
```

To display areas that extend across more than one UTM zone, it may be more appropriate to use the Mercator projection with parameters that make it transverse. Here is an example of a transverse Mercator projection appropriate to Chile. Note how the projection's line of zero distortion is aligned with the predominantly north-south axis of the country. The zero distortion line could

be put exactly on the midline of the country by a better choice of the origin vector's central meridian and orientation angle.

```
latlim = [-60 -15];centralMeridian = -70; width = 20;
axesm('mercator',...
    'Origin',[0 centralMeridian -90],...
    'Flatlimit',[-width/2 width/2],...
    'Flonlimit',sort(-latlim),...
    'Aspect','transverse')
displaym(worldlo('POline'));framem
gridm; setm(gca,'plinefill',1000)
tightmap
mdistort scale
```



Summary Guide

Literally an infinite number of possible map projections can be constructed. The Mapping Toolbox provides 60 different map projections. To view all the projections in the toolbox, use more than the command `maps`.

Cartographers choose a map projection by determining which property distortions they want to minimize or eliminate, if any. They also determine which of the three projection types (cylindrical, conic, or azimuthal) best suits their purpose, or they may decide upon special desired features, such as straight rhumb lines or great circles, true direction, or some other interesting trait. The following table lists the available projections along with their properties. Detailed information on all map projections provided by the Mapping Toolbox can be found in Chapter 10, “Projections Reference.”

Projection	Syntax	Type	Equal-Area	Conformal	Equidistant	Special Features
Balthasart	balthsrt	Cylindrical	•			
Behrmann	behrmann	Cylindrical	•			
Bolshoi Sovietskii Atlas Mira	bsam	Cylindrical				
Braun Perspective	braun	Cylindrical				
Cassini	cassini	Cylindrical			•	
Central	ccylin	Cylindrical				
Equal-Area Cylindrical	eqacylin	Cylindrical	•			
Equdistant Cylindrical	eqdcylin	Cylindrical			•	
Gall Isographic	giso	Cylindrical			•	
Gall Orthographic	gortho	Cylindrical	•			

Projection	Syntax	Type	Equal-Area	Conformal	Equidistant	Special Features
Gall Stereographic	gstereo	Cylindrical				
Lambert Equal-Area Cylindrical	lambcyln	Cylindrical	•			
Mercator	mercator	Cylindrical		•		1
Miller	miller	Cylindrical				
Plate Carree	pcarree	Cylindrical			•	
Trystan Edwards	trystan	Cylindrical	•			
Universal Transverse Mercator (UTM)	utm	Cylindrical		•		
Wetch	wetch	Cylindrical				
Apianus II	apianus	Pseudocylindrical				
Collignon	collig	Pseudocylindrical	•			
Craster Parabolic	craster	Pseudocylindrical	•			
Eckert I	eckert1	Pseudocylindrical				
Eckert II	eckert2	Pseudocylindrical	•			
Eckert III	eckert3	Pseudocylindrical				
Eckert IV	eckert4	Pseudocylindrical	•			
Eckert V	eckert5	Pseudocylindrical				
Eckert VI	eckert6	Pseudocylindrical	•			
Fournier	fournier	Pseudocylindrical	•			

Projection	Syntax	Type	Equal-Area	Conformal	Equidistant	Special Features
Goode Homolosine	goode	Pseudocylindrical	•			
Hatano Assymetrical Equal-Area	hatano	Pseudocylindrical	•			
Kavraisky V	kavrsky5	Pseudocylindrical	•			
Kavraisky VI	kavrsky6	Pseudocylindrical	•			
Loximuthal	loximuth	Pseudocylindrical				2
McBryde-Thomas Flat-Polar Parabolic	flatplrp	Pseudocylindrical	•			
McBryde-Thomas Flat-Polar Quartic	flatplrq	Pseudocylindrical	•			
McBryde-Thomas Flat-Polar Sinusoidal	flatplrs	Pseudocylindrical	•			
Mollweide	mollweid	Pseudocylindrical	•			
Putnins P5	putnins5	Pseudocylindrical				
Quartic Authalic	quartic	Pseudocylindrical	•			
Robinson	robinson	Pseudocylindrical				
Sinusoidal	sinusoid	Pseudocylindrical	•			
Tissot Modified Sinusoidal	modsine	Pseudocylindrical	•			
Wagner IV	wagner4	Pseudocylindrical	•			
Winkel I	winkel	Pseudocylindrical				
Albers Equal-Area Conic	eqaconic	Conic	•			
Equidistant Conic	eqdconic	Conic			•	

Projection	Syntax	Type	Equal-Area	Conformal	Equidistant	Special Features
Lambert Conformal Conic	`lambert`	Conic		•		
Murdoch I Conic	`murdoch1`	Conic			•	3
Murdoch III Minimum Error Conic	`murdoch3`	Conic			•	3
Bonne	`bonne`	Pseudoconic	•			
Werner	`werner`	Pseudoconic	•			
Polyconic	`polycon`	Polyconic				
Van Der Grinten I	`vgrint1`	Polyconic				
Breusing Harmonic Mean	`breusing`	Azimuthal				
Equidistant Azimuthal	`eqdazim`	Azimuthal			•	
Gnomonic	`gnomonic`	Azimuthal				4
Lambert Azimuthal Equal-Area	`eqaazim`	Azimuthal	•			
Orthographic	`ortho`	Azimuthal				
Stereographic	`stereo`	Azimuthal		•		5
Universal Polar Stereographic (UPS)	`ups`	Azimuthal		•		5
Vertical Perspective Azimuthal	`vperspec`	Azimuthal				
Wiechel	`wiechel`	Pseudoazimuthal	•			
Aitoff	`aitoff`	Modified Azimuthal				
Briesemeister	`bries`	Modified Azimuthal	•			

Projection	Syntax	Type	Equal-Area	Conformal	Equidistant	Special Features
Hammer	`hammer`	Modified Azimuthal	•			
Globe	`globe`	Spherical	•	•	•	6

1. Straight rhumb lines.
2. Rhumb lines from central point are straight, true to scale, and correct in azimuth.
3. Correct total area.
4. Straight line great circles.
5. Great and small circles appear as circles or lines.
6. Three-dimensional display.

6

Mapping Applications

Introduction to Geographic Statistics and Navigation . 6-2

Geographic Statistics 6-3
Geographic Means 6-3
Geographic Standard Deviation 6-4
Equal-Areas in Geographic Statistics 6-7
Geographically Filtering Datasets 6-10

Navigation 6-11
Conventions for Navigational Functions 6-11
Fixing Position 6-12
Planning 6-24
Track Laydown – Displaying Navigational Tracks 6-26
Dead Reckoning 6-28
Drift Correction 6-33
Time Zones 6-35
Time Notation 6-37

Introduction to Geographic Statistics and Navigation

The Mapping Toolbox provides a set of functions for applying mapping computations to geographic statistical analysis and navigation. The geographic statistics functions are necessary for proper analysis of geographical data as opposed to simple Cartesian (x-y) data. The navigation functions provide such useful navigational tasks as establishing position and planning routes or tracks.

Geographic Statistics

Most statistical functions implicitly assume that the data resides in a Cartesian coordinate system. Simply using geographic data in a Cartesian manner can give statistically inappropriate results. While the Cartesian assumption may be insignificant for small geographic regions, for larger areas it can lead to incorrect conclusions due to faulty distance measures, faulty area assumptions, or both. The Mapping Toolbox provides several functions for analyzing geographic data statistically.

Geographic Means

Consider the problem of calculating the mean position of a collection of geographic points. You might be inclined to simply take the arithmetical mean of the latitudes and longitudes using the standard MATLAB `mean` command, but this could be very misleading.

Take two points at the same latitude, 180 apart in longitude, for example (30 N,90 W) and (30 N,90 E). The *mean* latitude is (30+30)/2=30, which seems right. However, the mean longitude must be (90+(-90))/2=0. From the point of view at the Prime Meridian this seems fine, too. At the 0 meridian, each point has the same longitude difference from the mean. What about the point of view at the Date Line? At the 180 meridian, each point also has the same longitude difference. Why isn't 180 the mean longitude?

This problem is further complicated when some points are at different latitudes. Remember, a degree of longitude at the Arctic Circle covers a much smaller distance than a degree at the Equator.

In fact, in the first example, is 30 Nthe right mean latitude, either? The mean position of two points should be equidistant from those two points, and further, it should be the equidistant point that minimizes this distance. Is (30 N,0) a reasonable mean point?

```
dist1 = distance(30,90,30,0)
dist1 =
   75.5225
dist2 = distance(30,-90,30,0)
dist2 =
   75.5225
```

Here is another point, call it (lat,lon), that is also equidistant from both points, and the distance is much shorter:

```
dist1 = distance(30,90,lat,lon)
dist1 =
   60.0000
dist2 = distance(30,-90,lat,lon)
dist2 =
   60.0000
```

What is this mystery point? The lat is 90 N, and anylon will do. The North Pole is the true geographic mean of these two points. Note that the great circle containing both points also runs through the North Pole, and a great circle represents the shortest path between two points.

The Mapping Toolbox includes the function meanm to determine the geographic mean of any number of points. This is accomplished through the three-dimensional vector addition of all the points. For example, try the following:

```
lats = [30 30];
longs = [-90 90];
[latbar,longbar] = meanm(lats,longs)
latbar =
   90
longbar =
   0
```

This is the answer we now expect. This geographic mean can result in one oddity; if the vectors all cancel each other, the mean is the center of the planet. In this case, the returned mean point is (NaN,NaN) and a warning is displayed. This phenomenon is highly improbable in *real* data, but can be easily constructed. For example, it occurs when all the points are equally spaced along a great circle. Try taking the geographic mean of (0 ,0), (0 ,120), and (0 ,240), which trisect the Equator.

Geographic Standard Deviation

As you might now expect, the Cartesian definition of standard deviation provided in the standard MATLAB function std is also inappropriate for most geographic data. Depending upon your purpose, you may want to use the separate geographic deviations for latitude and longitude provided by the

function `stdm`, or the single standard distance provided in `stdist`. Both methods measure the deviation of points from the mean position calculated by `meanm`.

The Meaning of stdm

The `stdm` function handles the latitude and longitude deviations separately.

```
[latstd,lonstd] = stdm(lat,lon)
```

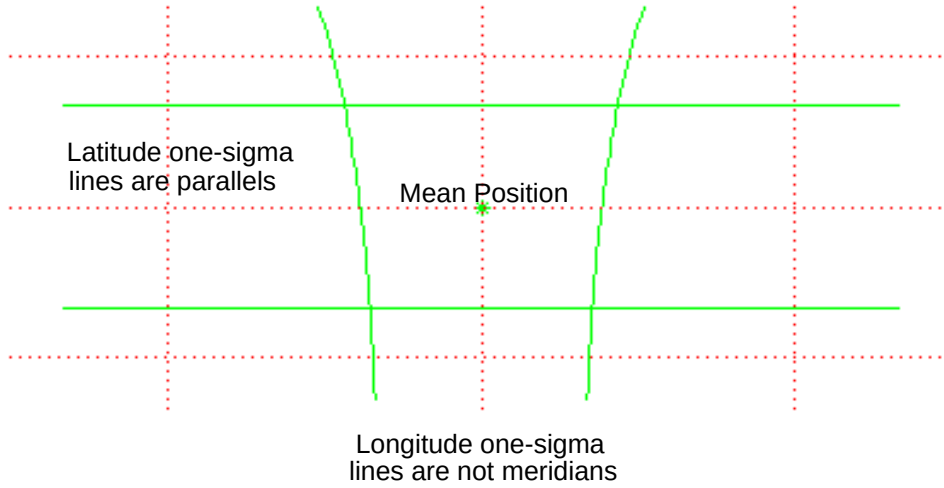
The function returns two deviations, one for latitudes and one for longitudes.

Latitude deviation is a straightforward standard deviation calculation from the mean latitude (mean parallel) returned by `meanm`. This is a reasonable measure for most cases, since a degree of latitude always has the same arc length.

Longitude deviation is another matter. Simple calculations based on sum-of-squares angular deviation from the mean longitude (mean meridian) are misleading. The arc length represented by a degree of longitude at extreme latitudes is significantly smaller than that at low latitudes.

The term *departure* is used to represent the arc length distance along a parallel of a point from a given meridian. For example, assuming a spherical planet, the departure of a degree of longitude at the Equator is a degree of arc length, but the departure of a degree of longitude at a latitude of 60 is one-half a degree of arc length. The `stdm` function calculates a sum-of-squares departure deviation from the mean meridian.

If you want to plot the one-sigma lines for `stdm`, the latitude sigma lines are parallels. However, the longitude sigma lines are not meridians; they are lines of constant departure from the mean parallel.



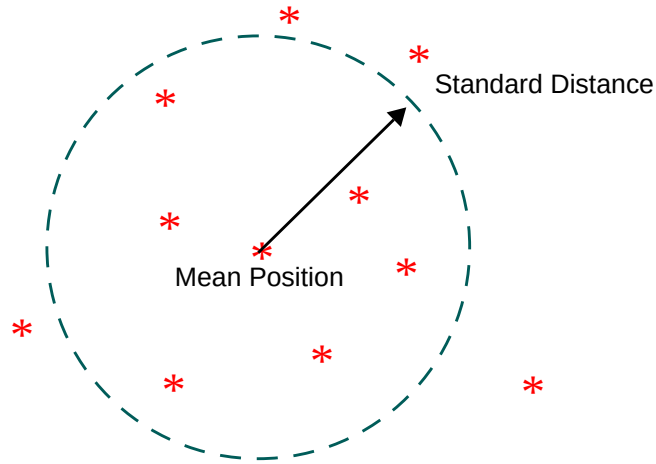
This handling of deviation has its problems. For example, its dependence upon the logic of the coordinate system can cause it to break down near the poles. For this reason, the standard distance provided by `stdist` is often a better measure of deviation. The `stdm` handling is useful for many applications, especially when the data is not global. For instance, these potential difficulties would not be a danger for data points confined to the country of Mexico.

The Meaning of stdist

The standard distance of geographic data is a measure of the dispersion of the data in terms of its distance from the geographic mean. Among its advantages are its applicability anywhere on the globe and its single value:

```
dist = stdist(lat,lon)
```

In short, the standard distance is the average, norm, or *cubic-norm* of the distances of the data points in a great circle sense from the mean position. It is probably a superior measure to the two deviations returned by `stdm` except when a particularly latitude- or longitude-dependent feature is under examination



Equal-Areas in Geographic Statistics

A common error in applying two-dimensional statistics to geographic data lies in ignoring equal-area treatment. It is often necessary to *bin-up* data to statistically analyze it. In a Cartesian plane, this is easily done by dividing the space into equal *x-y* squares. The geographic equivalent of this is to bin up the data in equal latitude-longitude *squares*. Since such squares at high latitudes cover smaller areas than their low-latitude counterparts, the observations in these regions are underemphasized. The result can be conclusions that are biased towards the Equator.

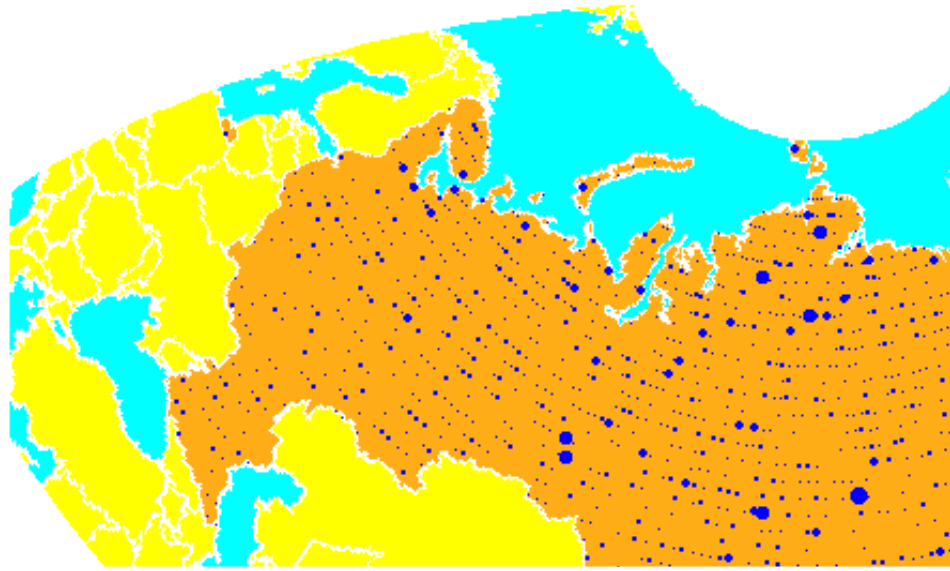
Geographic Histograms

The geographic histogram command `histr` allows you to display *binned-up* geographic observations. The `histr` command results in equirectangular binning. Each bin has the same angular measurement in both latitude and longitude, with a default measurement of 1 degree. The center latitudes and longitudes of the bins are returned, as well as the number of observations per bin:

```
[binlat,binlon,num] = histr(lats,lons)
```

As previously noted, these equirectangular bins result in counting bias towards the Equator. Here is a display of the one-degree-by-one-degree binning of

approximately 5,000 random data points in Russia. The relative size of the circles indicates the number of observations per bin:

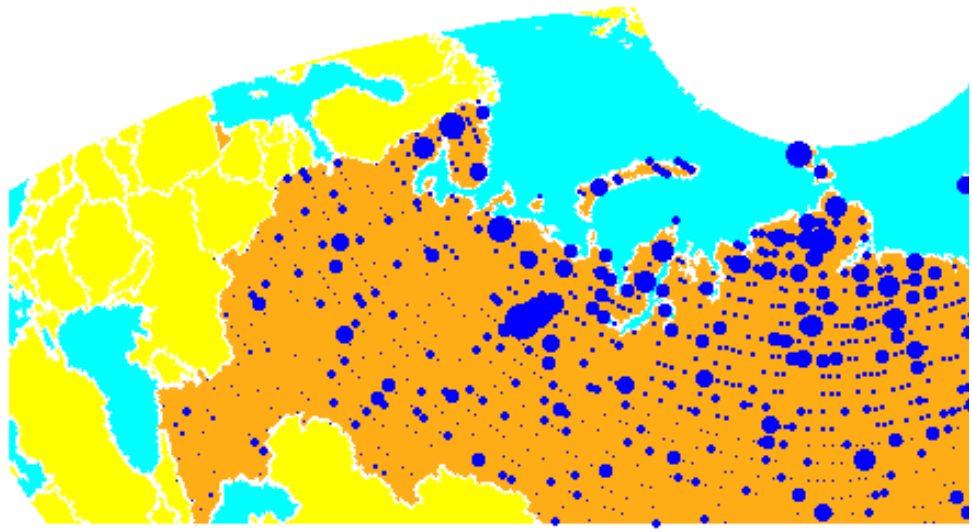


This is a portion of the whole map, displayed in an equal-area Bonne projection. The first step in creating data displays without area bias is to choose an equal-area projection. The proportionally sized symbols are a result of the specialized display function `scatterm`.

The area bias can be eliminated by adding a fourth output argument to `histr`, which will be used to weight each bin's observation by that bin's area:

```
[binlat,binlon,num,wnum] = histr(lats,lons)
```

The fourth output is the weighted observation count. Each bin's observation count is divided by its normalized area. Therefore, a high-latitude bin will have a larger weighted number than a low-latitude bin with the same number of actual observations. The same data and bins look much different when they are area-weighted:



Notice that there are larger symbols to the north in this display. The previous display suggested that the data was relatively uniformly distributed. When equal-area considerations are included, it is clear that the data is skewed to the north. In fact, the data used is northerly skewed, but a simple equirectangular handling failed to demonstrate this.

The `histr` command, therefore, does provide for the display of area-weighted data. However, the actual bins used are of varying areas. Remember, the one-degree-by-one-degree bin near a pole is much smaller than its counterpart near the Equator.

The `hista` command provides for actual equal-area bins.

Converting to an Equal-Area Coordinate System

Finally, on the topic of equal-area data handling, the actual data itself can be converted to an equal-area coordinate system for analysis with other statistical functions. It is easy to convert a collection of geographic latitude-longitude points to an equal-area *x-y* Cartesian coordinate system. The `grn2eqa` command applies the same transformation used in calculating the Equal-Area Cylindrical projection:

```
[x,y] = grn2eqa(lat,lon)
```

For each Greenwich `lat - long` pair, an equal-area `x - y` is returned. The variables x and y can then be operated on under the equal-area assumption, using a variety of two-dimensional statistical techniques. Tools for such analysis can be found in the Statistics Toolbox and elsewhere. The results can then be converted back to Greenwich coordinates using the `eqa2grn` command:

```
[lat,lon] = eqa2grn(x,y)
```

Remember, when converting back and forth between systems, latitude corresponds to y and longitude corresponds to x.

Geographically Filtering Datasets

Often, a set of data will contain unwanted data mixed in with the desired values. For example, your data might include points for the entire United States, but you only want to work with those points falling in Alabama, or perhaps the dataset is *untidy*– out of 4,000 points, you notice that 3 or 4 obviously fall outside of reality (for example, one of your city-points is in the middle of the ocean). It can be quite a chore to look at each dataset element individually. Perhaps selecting a portion of the data is part of your analysis.

The `filterm` command works with a matrix map to filter a vector dataset. The form is of the following:

```
[flats,flons] = filterm(lats,lons,map,maplegend,allowed]
```

In short, each location defined by `lats` and `lons` is compared to the value at that point in `map`. If the value is `allowed`, that point is included in `flats` and `flons`.

The map might be a politically indexed map, and the allowed values might be the code or codes corresponding to the states or countries desired (e.g., Alabama). The map might also be a valued map, or a logical condition thereon, and the allowed value might be 1 for *true*. Here's what an example might look like using the `topo` map:

```
[flats,flons] = filterm(lats,lons,topo>0,topolegend,1)
```

The result would be those points corresponding to land.

Navigation

One field that makes extensive use of geographic information is navigational science and practice. The Mapping Toolbox includes a variety of specialized functions for navigation.

The practice of navigation includes a variety of tasks for the operation of both watercraft and aircraft. One task is establishing position, using known, fixed landmarks (piloting), employing the stars, Sun, and Moon (celestial navigation), utilizing technological fixing systems (radio and satellite navigation, including GPS), or deducing net movement from a past known position (dead reckoning).

Another navigational task involves planning a voyage or flight, which includes determining a short route (great circle approximation), weather avoidance (optimal track routing), and setting out a plan of intended movement (track laydown).

The Mapping Toolbox contains functions for some of these navigational activities.

Conventions for Navigational Functions

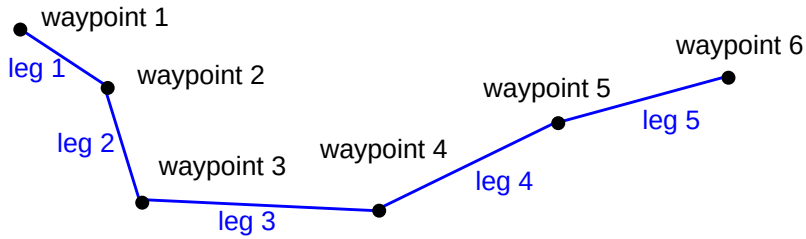
Units

The Mapping Toolbox is, in general, very flexible in allowing a variety of angular and distance measurement units. To make the strictly navigational functions easy to handle and to conform to common navigational practice, *for these specific functions*, angles are always in degrees, distances are always in nautical miles, and speeds are always in knots (nautical miles per hour). The functions for which these conventions hold are `dreckon`, `gcwaypts`, `legs`, and `navfix`. Related functions that do not carry this restriction include `rhxrh`, `scxsc`, `gcxgc`, `gcxsc`, `track`, `timezone`, and `crossfix`, because of their potential for application outside of navigation.

Navigational Track Format

Navigational track format requires column-vector variables for the latitudes and longitudes of track waypoints. A *waypoint* is a point through which a track passes, usually corresponding to a course (or speed) change. Navigational tracks are made up of the line segments connecting these waypoints, which are

called *legs*. In this format, therefore, *n* legs are described using *n+1* waypoints, because an endpoint for the final leg must be defined.



Here, five track legs require six waypoints. In navigational track format, the waypoints would be represented by two 6-by-1 vectors, one for the latitudes and one for the longitudes.

Fixing Position

The fundamental objective of navigation is to determine at a given moment how to proceed to your destination, avoiding hazards on the way. The first step in accomplishing this is to establish your current position. Early sailors kept within sight of land to facilitate this. Today, navigation within sight (or radar range) of land is called *piloting*. Positions are fixed by correlating the bearings and/or ranges of landmarks. In real-life piloting, all sighting bearings are treated as rhumb lines, while in fact they are actually great circles.

Over the distances involved with visual sightings (up to 20 or 30 nautical miles), this assumption causes no measurable error and it provides the significant advantage of allowing the navigator to plot all bearings as straight lines on a Mercator projection.

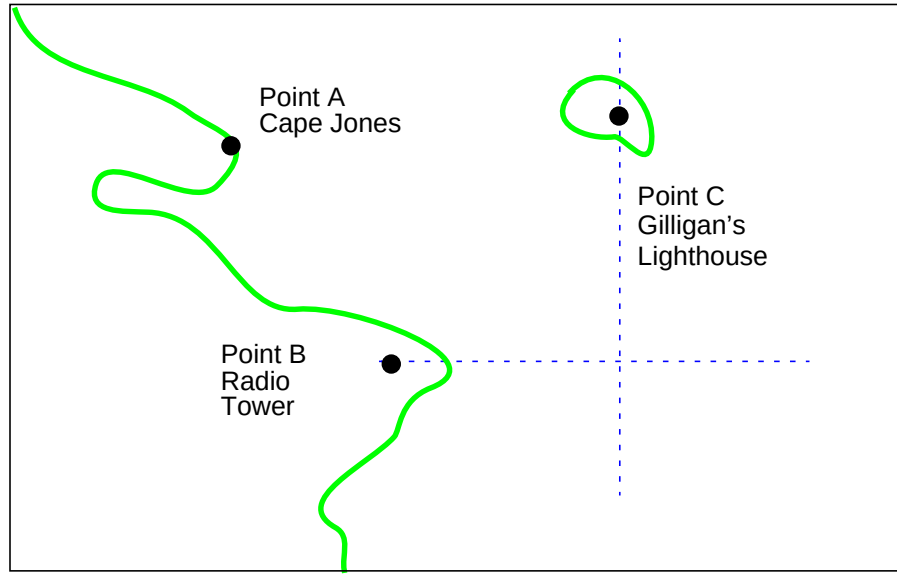
The Mercator was designed exactly for this purpose. Range circles, which might be determined with a radar, are assumed to plot as true circles on a Mercator chart. This allows the navigator to manually draw the range arc with a compass.

These assumptions also lead to computationally efficient methods for fixing positions with a computer. The Mapping Toolbox includes the `navfix` function, which mimics the manual plotting and fixing process using these assumptions.

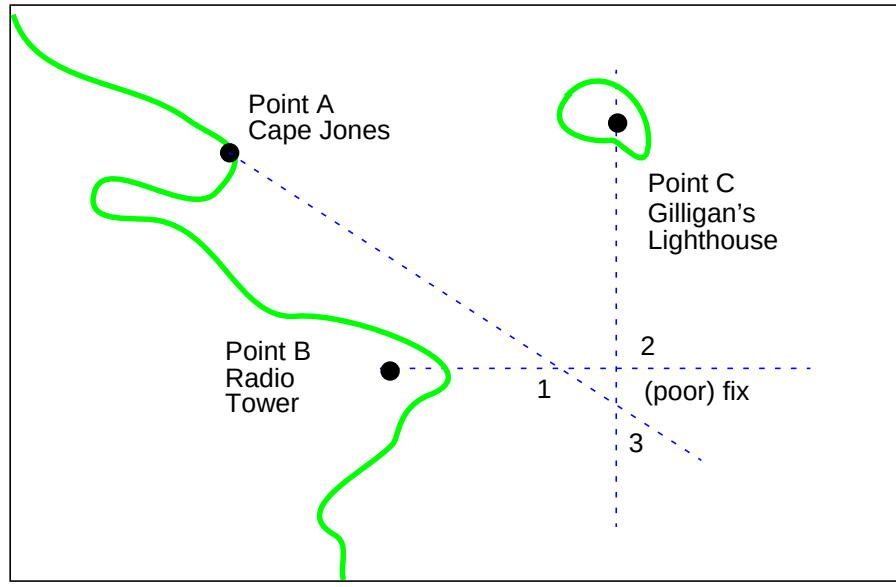
To obtain a good navigational fix, your relationship to at least three known points is considered necessary. A questionable or poor fix can be obtained with two known points.

Some Possible Situations

In this imaginary coastal region, you take a visual bearing on the radio tower of 270 . At the same time, Gilligan's Lighthouse bears 0 . If you plot a 90 -270 line through the radio tower, and a 0 -180 line through the lighthouse on your Mercator chart, the point at which the lines cross is a fix. Since you have used only two lines, however, its quality is questionable.



But wait; your port lookout says he took a bearing on Cape Jones of 300 .If that line exactly crosses the point of intersection of the first two lines, you will have a perfect fix.

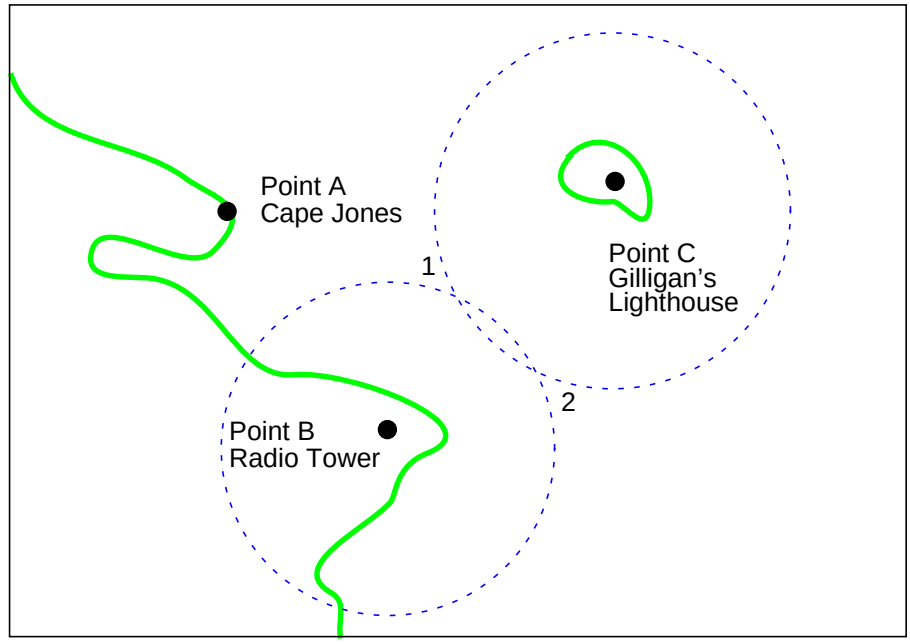


Whoops. What happened? Is your lookout in error? Possibly, but perhaps one or both of your bearings was slightly in error. This happens all the time. Which point, 1, 2, or 3, is correct? As far as you know, they are all equally valid.

In practice, the little triangle is plotted, and the fix position is taken as either the center of the triangle or the vertex closest to a danger (like shoal water). If the triangle is large, the quality is reported as *poor*, or even as *no fix*. If a fourth line of bearing is available, it can be plotted to try to resolve the ambiguity. When all three lines appear to cross at exactly the same point, the quality is reported as *excellent* or *perfect*.

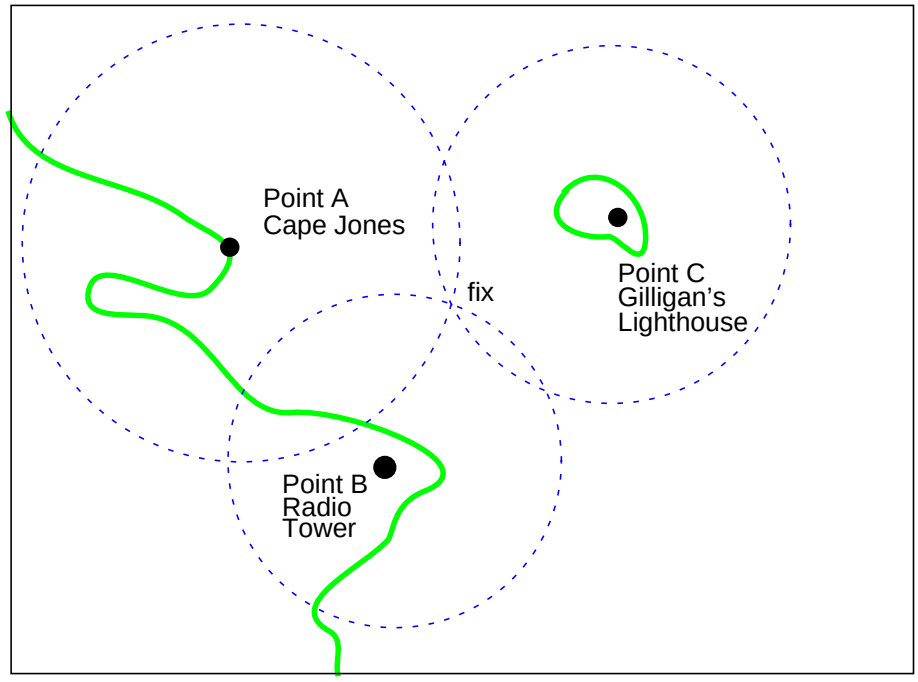
Notice that three lines resulted in three intersection points. Four lines would return six intersection points. This is a case of combinatorial counting. Each intersection corresponds to choosing two lines to intersect from among *n* lines, so the total number of intersection points will be *n*-choose-2.

The next time you traverse these straits, it is a very foggy morning. You can't see any landmarks, but luckily, your navigational radar is operating. Each of these landmarks has a good radar signature, so you're not worried. You get a range from the radio tower of 14 nautical miles and a range from the lighthouse of 15 nautical miles.



Now what? You took ranges from only two objects, and yet you have two possible positions. This ambiguity arises from the fact that circles can intersect twice.

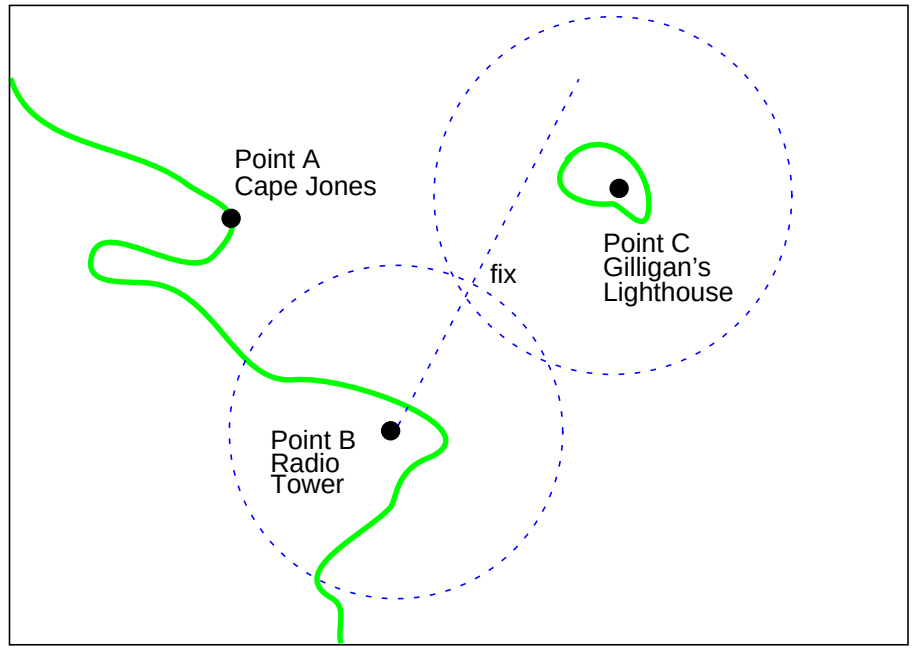
Luckily, your radar watch reports that he has Cape Jones at 18 nautical miles. This should resolve everything.



You were lucky this time. The third range resolved the ambiguity and gave you an excellent fix. Three intersections practically coincide. Sometimes the ambiguity is resolved, but the fix is still poor because the three closest intersections form a sort of circular triangle.

Sometimes the third range only adds to the confusion, either by bisecting the original two choices, or by failing to intersect one or both of the other arcs at all. In general, when *n* arcs are used, 2x(*n*-choose-2) possible intersections result. In this example, it is easy to tell which ones are *right*.

Bearing lines and arcs can be combined. If instead of reporting a third range, your radar watch had reported a bearing from the radar tower of 20 , the ambiguity could have also been resolved. Note, however, that in practice, lines of bearing for navigational fixing should only be taken visually, except in desperation. A radar's beamwidth can be a degree or more, leading to uncertainty.



As you begin to wonder whether this manual plotting process could be automated, your first officer shows up on the bridge with a laptop and the Mapping Toolbox.

Using navfix

The `navfix` command can be used to determine the points of intersection between any number of lines and arcs. Be warned, however, that due to the combinatorial nature of this process, the computation time grows rapidly with the number of objects. To illustrate this function, assign positions to the landmarks. Point A, Cape Jones, is at (`latA`,`lonA`). Point B, the radio tower, is at (latB,lonB). Point C, Gilligan's Lighthouse, is at (`latC`,`lonC`).

For the bearing-lines-only example, the syntax is:

```
[latfix,lonfix] = navfix([latA latB latC],[lonA lonB lonC],...
                         [300 270 0])
```

This defines the three points and their bearings as taken *from the ship*. The outputs would look something like this, with actual numbers of course:

```
latfix =
   latfix1       NaN          % A intersecting B
   latfix2       NaN          % A intersecting C
   latfix3       NaN          % B intersecting C
lonfix =
   lonfix1       NaN          % A intersecting B
   lonfix2       NaN          % A intersecting C
   lonfix3       NaN          % B intersecting C
```

Notice that these are two-column matrices. The second column consists of `NaN`s because it is used only for the two-intersection ambiguity associated with arcs.

For the range-arcs-only example, the syntax is:

```
[latfix,lonfix] = navfix([latA latB latC],[lonA lonB lonC],...
                         [16 14 15],[0 0 0])
```

This defines the three points and their ranges as taken from the ship. The final argument indicates that the three cases are all ranges.

The outputs have the following form:

```
latfix =
   latfix11  latfix12          % A intersecting B
   latfix21  latfix22          % A intersecting C
   latfix31  latfix32          % B intersecting C
lonfix =
   lonfix11  lonfix12          % A intersecting B
   lonfix21  lonfix22          % A intersecting C
   lonfix31  lonfix32          % B intersecting C
```

Here, the second column is used, because each pair of arcs has two potential intersections.

For the bearings and ranges example, the syntax requires the final input to indicate which objects are lines of bearing (indicated with a 1) and which are range arcs (indicated with a 0):

```
[latfix,lonfix] = navfix([latB latB latC],[lonB lonB lonC],...
                         [20 14 15],[1 0 0])
```

The resulting output is mixed:

```
latfix =
   latfix11       NaN          % Line B intersecting Arc B
   latfix21  latfix22          % Line B intersecting Arc C
   latfix31  latfix32          % Arc B intersecting Arc C
lonfix =
   lonfix11       NaN          % Line B intersecting Arc B
   lonfix21  lonfix22          % Line B intersecting Arc C
   lonfix31  lonfix32          % Arc B intersecting Arc C
```

Only one intersection is returned for the line from B with the arc about B, since the line originates inside the circle and intersects it once. The same line intersects the other circle twice, and hence it returns two points. The two circles taken together also return two points.

Usually, you have an idea as to where you are before you take the fix. For example, you might have a dead reckoning position for the time of the fix (see below). If you provide `navfix` with this estimated position, it will choose from each pair of ambiguous intersections the point closest to the estimate. Here's what it might look like:

```
[latfix,lonfix] = navfix([latB latB latC],[lonB lonB lonC],...
                         [20 14 15],[1 0 0],drlat,drlon)
latfix =
   latfix11                    % the only point
   latfix21                    % the closer point
   latfix31                    % the closer point
lonfix =
   lonfix11                    % the only point
   lonfix21                    % the closer point
   lonfix31                    % the closer point
```

A Numerical Example of Using navfix

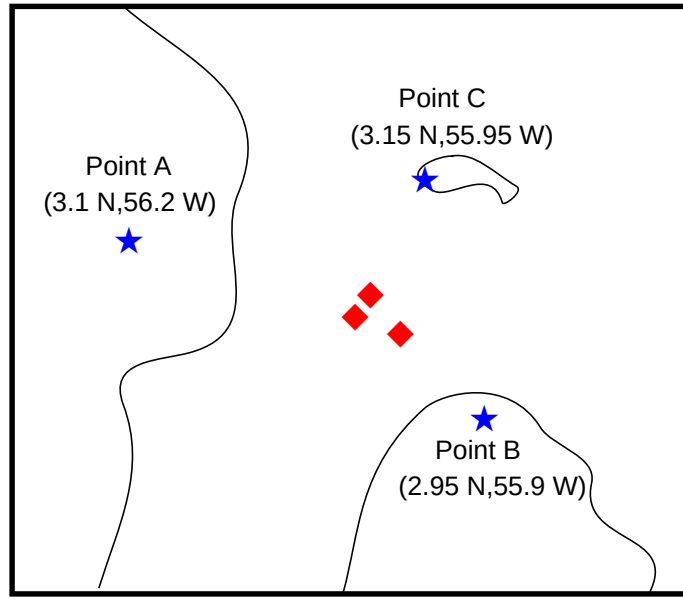
For a numerical example, define some specific points in the middle of the Atlantic Ocean. These are strictly arbitrary; perhaps they correspond to points in Atlantis:

```
lata = 3.1;  lona = -56.2;
latb = 2.95; lonb = -55.9;
latc = 3.15; lonc = -55.95;
```

Plot them on a Mercator projection:

```
axesm('MapProjection','mercator','Frame','on',...
   'MapLatLimit',[2.8 3.3],'MapLonLimit',[-55.8 -56.3])
plotm([lata latb latc],[lona lonb lonc],...
   'LineStyle','none','Marker','pentagram',...
   'MarkerEdgeColor','b','MarkerFaceColor','b',...
   'MarkerSize',12)
```

Here's what it looks like (the labeling and imaginary coastlines are added after the fact for illustration):



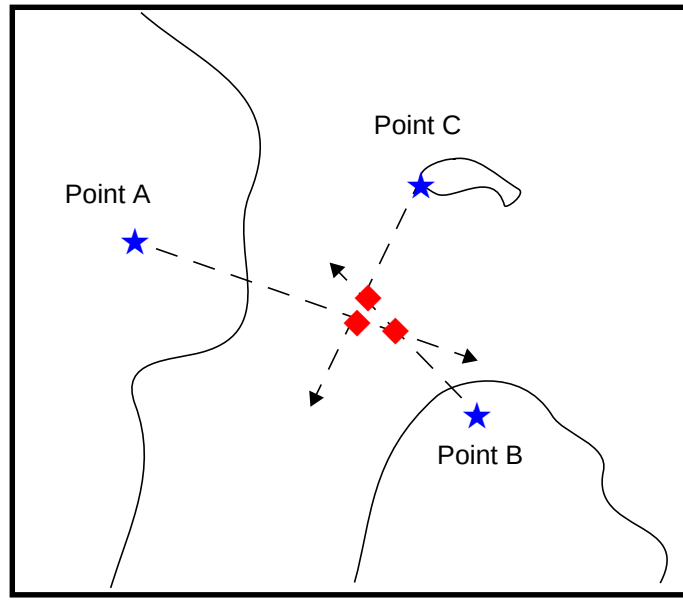
Take three visual bearings: Point A bears 289 , Point B bears 135 , and Point C bears 026.5 . Calculate the intersections:

```
[newlat,newlong] = navfix([lata latb latc],[lona lonb lonc],...
                          [289 135 26.5],[1 1 1])
newlat =
   3.0214       NaN
   3.0340       NaN
   3.0499       NaN
```

```
newlong =
   -55.9715       NaN
   -56.0079       NaN
   -56.0000       NaN
```

Add the bearing lines and intersection points to the map:

```
plotm(newlat,newlong,'LineStyle','none',...
   'Marker','diamond','MarkerEdgeColor','r',...
   'MarkerFaceColor','r','MarkerSize',9)
```



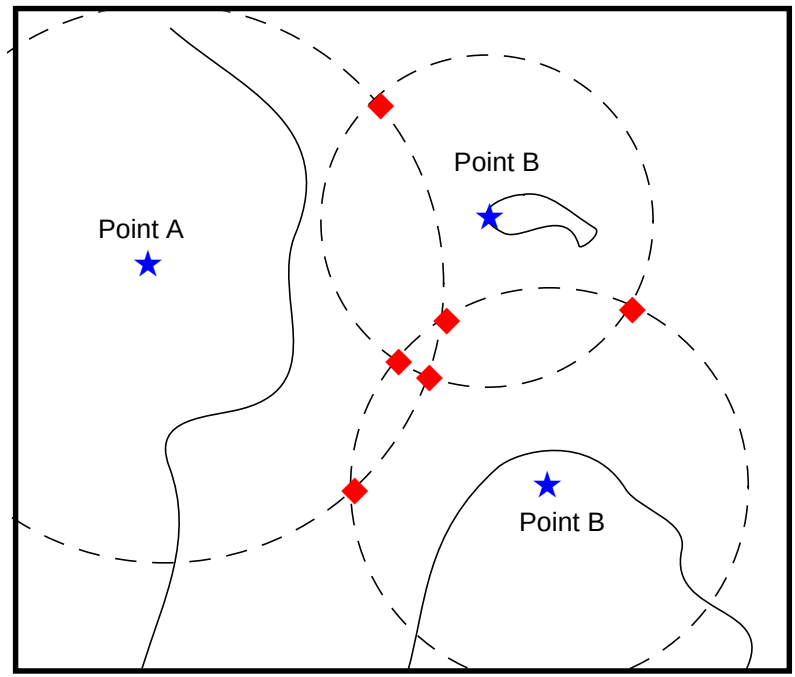
Notice that each pair of objects results in only one intersection, since all are lines of bearing.

What if instead, you had ranges from the three points, A, B, and C, of 13nm, 9nm, and 7.5nm, respectively.

```
[newlat,newlong] = navfix([lata latb latc],[lona lonb lonc],...
                          [13 9 7.5],[0 0 0])
newlat =
   3.0739    2.9434
   3.2413    3.0329
   3.0443    3.0880
```

```
newlong =
   -55.9846   -56.0501
   -56.0355   -55.9937
   -56.0168   -55.8413
```

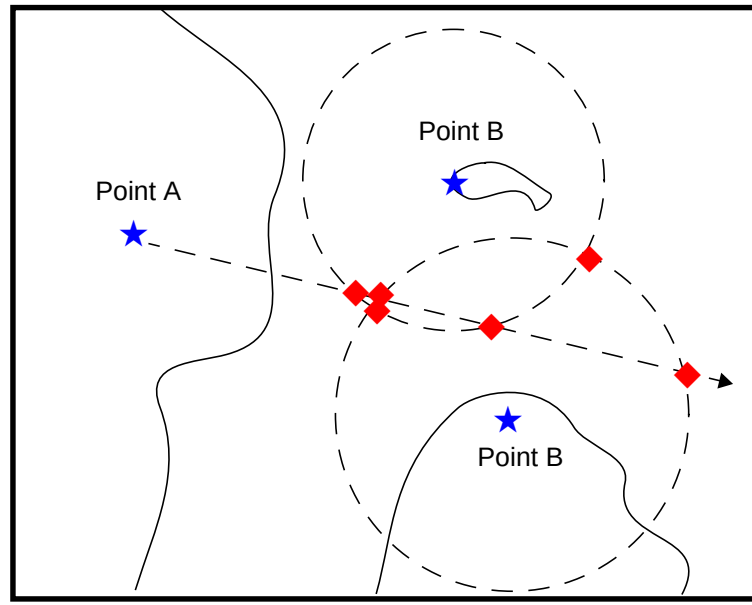
Here's what these points look like:



Three of these points look reasonable, three do not. What if, instead of a range from Point A, you had a bearing to it of 284 ?

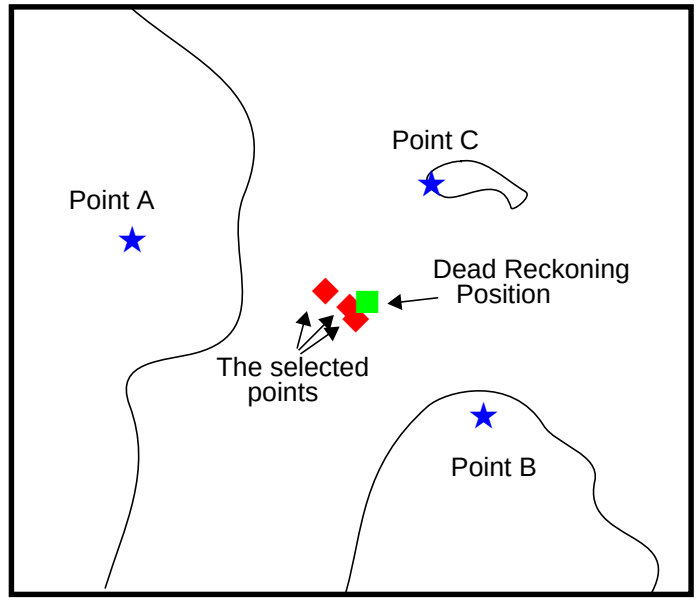
```
[newlat,newlong] = navfix([lata latb latc],[lona lonb lonc],...
                         [284 9 7.5],[1 0 0])
newlat =
   3.0526    2.9892
   3.0592    3.0295
   3.0443    3.0880
```

```
newlong =
   -56.0096  -55.7550
   -56.0360  -55.9168
   -56.0168  -55.8413
```



Again, visual inspection of the results indicates which three of the six possible points seem like *reasonable* positions. When using the dead reckoning position (3.05 N,56.0 W), the closer, more reasonable candidate from each pair of intersecting objects is chosen:

```
drlat = 3.05; drlon = -56;
[newlat,newlong] = navfix([lata latb latc],[lona lonb lonc],...
                          [284 9 7.5],[1 0 0],drlat,drlon)
newlat =
   3.0526
   3.0592
   3.0443
newlong =
   -56.0096
   -56.0360
   -56.0168
```



Planning

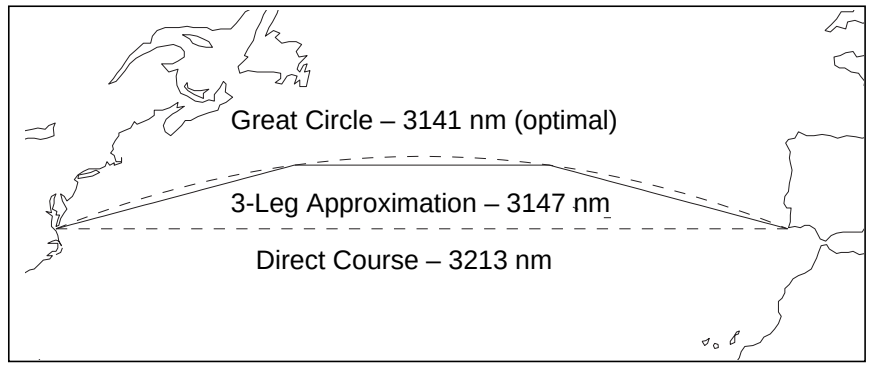
We know that the shortest path between two geographic points is a great circle. Sailors and aviators are interested in minimizing distance travelled, and hence delay. We also know that the rhumb line is a path of constant heading, the *natural* means of travelling. In general, to follow a great circle path, one would have to continuously alter course. This is impractical. However, a great circle path can be approximated by rhumb line segments so that the added distance is minor and the number of course changes minimal.

Surprisingly, very few rhumb line *track legs* are required to closely approximate the distance of the great circle path.

Consider the voyage from Norfolk, Virginia (37 N,76 W), to Cape St. Vincent, Portugal (37 N,9 W), one of the most heavily trafficked routes in the Atlantic. A due-east rhumb line track is 3,213 nautical miles in length, while the optimal great circle distance is 3,141 nautical miles.

Although the rhumb line path is only a little more than 2% longer, this is an additional 72 miles over the course of the trip. For a 12-knot tanker, this

results in a 6-hour delay, and in shipping, time is money. If just three rhumb line segments are used to approximate the great circle, the total distance of the trip is 3,147 nautical miles. Our tanker would suffer only a half-hour delay over optimal.



The Mapping Toolbox provides the function `gcwaypts` to quickly calculate waypoints in navigation track format in order to approximate a great circle with rhumb line segments. The syntax is simple:

```
[latpts,lonpts] = gcwaypts(lat1,lon1,lat2,lon2,numlegs)
```

All of the inputs for this command are scalars. The `numlegs` input is the number of equal length legs desired, which is 10 by default. The outputs are column vectors representing waypoints in navigational track format. The size of each of these vectors will be [(numlegs+1) 1]. Here are the points for the example illustrated above:

```
[latpts,lonpts] = gcwaypts(37,-76,37,-9,3)
latpts =
   37.0000
   41.5076
   41.5076
37.0000
lonpts =
   -76.0000
   -54.1777
   -30.8223
   -9.0000
```

These points represent waypoints along the great circle between which the approximating path follows rhumb lines. Four points are needed for three legs, because the final point at Cape St. Vincent must be included.

Track Laydown – Displaying Navigational Tracks

Navigational tracks are most useful when graphically displayed. Traditionally, the navigator identifies and plots waypoints on a Mercator projection and then connects them with a straightedge, which on this projection results in rhumb line tracks. In the previous example, waypoints were chosen to approximate a great circle route, but they may be selected for a variety of other reasons.

Let's say that after arriving at Cape St. Vincent, your tanker must traverse the Straits of Gibraltar and then travel on to Port Said, the northern terminus of the Suez Canal. On the scale of the Mediterranean Sea, following great circle paths is of little concern compared to ensuring that the many straits and passages are safely transited. The navigator selects appropriate waypoints and plots them.

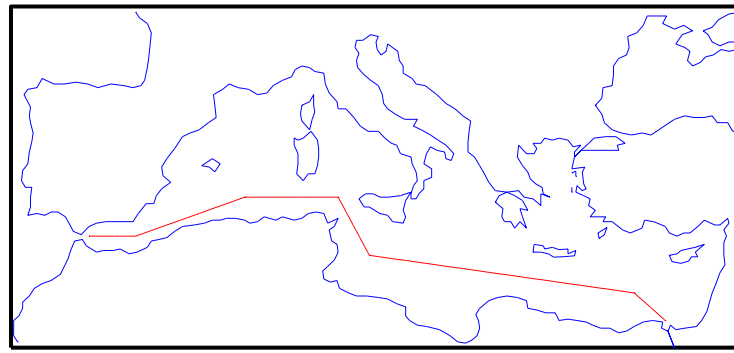
To do this with the Mapping Toolbox, you can display a map axes with a Mercator projection, select appropriate map latitude and longitude limits to isolate the area of interest, plot coastline data, and interactively mouse-select the waypoints with the `inputm` command. The `track` command will generate points to connect these waypoints, which can then be displayed with `plotm`.

For illustration, assume that the waypoints are known (or were gathered using `inputm`). To learn about using `inputm`, see "Interacting with Displayed Maps" on page 3-53, or the online Mapping Toolbox reference documentation.

```
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

load coast
axesm('MapProjection','mercator',...
'MapLatLimit',[30 47],'MapLonLimit',[-10 37])
```

```
framem
plotm(lat,long)

[lttrk,lntrk] = track(waypoints);
plotm(lttrk,lntrk,'r')
```

Although these track segments are straight lines on the Mercator projection, they will be curves on others:



The segments of a track like this are called *legs*. Each of these legs can be described in terms of course and distance. The function `legs` will take the waypoints in navigational track format and return the courses and distances required for each leg. Remember, the order of the points in this format determines the direction of travel. Courses are therefore calculated from each waypoint to its successor, not vice-versa.

```
[courses,distances] = legs(waypoints)
courses =
   90.0000
   70.3132
   90.0000
   151.8186
   98.0776
   131.5684
distances =
   145.6231
   356.2117
   283.6839
   204.2073
```

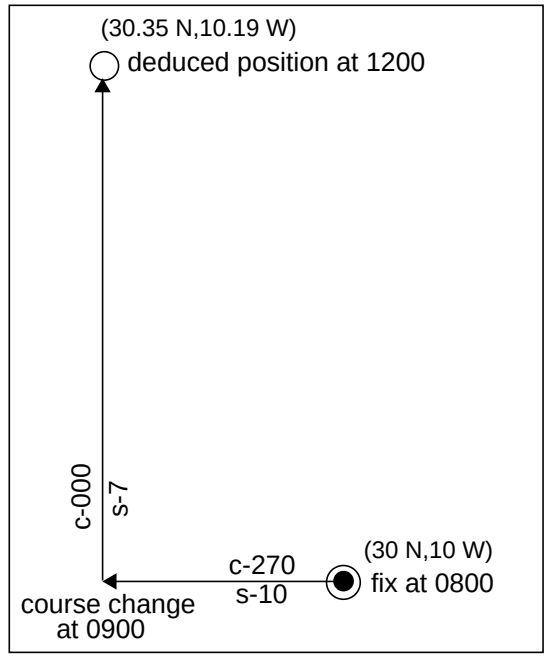
```
854.0092
135.6415
```

Since this is a navigation function, the courses are all in degrees and the distances are in nautical miles. From these distances, speeds required to arrive at Port Said at a given time can be calculated. Southbound traffic is allowed to enter the canal only once per day, so this information might be economically significant, since unnecessarily high speeds can lead to high fuel costs.

Dead Reckoning

When sailors first ventured out of sight of land, they faced a daunting dilemma. How could they find their way home if they didn't know where they were? The practice of *dead reckoning* is an attempt to deal with this problem. The term is derived from *deduced reckoning*.

Briefly, dead reckoning is vector addition plotted on a chart. For example, if you had a fix at (30 N,10 W) at 0800, and you proceeded due west for 1 hour at 10 knots, and then you turned north and sailed for 3 hours at 7 knots, you should be at (30.35 N,10.19 W) at 1200. All's well, ring eight bells on time, Mr. Bowditch, right?



However, Mr. Bowditch *shoots the sun* at local apparent noon and discovers that the ship's latitude is actually 30.29 N. What's worse, he lives before the invention of a reliable chronometer, and so he cannot calculate his longitude at all from this sighting. What happened?

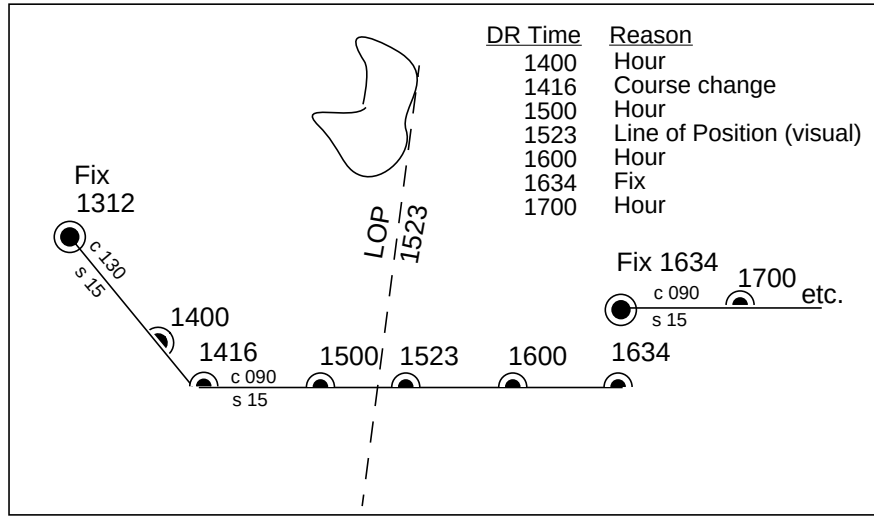
Leaving aside the difficulties in speed determination and the need to tack off course, even modern craft have to contend with winds and currents. However, despite these limitations, dead reckoning is still used for determining position between fixes and for forecasting future positions. This is because dead reckoning provides a certainty of assumptions that estimations of wind and current drift cannot.

When navigators establish a fix from some source, be it from piloting, celestial, or satellite observations, they plot a dead reckoning (DR) track, which is a plot of the intended positions of the ship forward in time. In practice, dead reckoning is usually plotted for 3 hours in advance, or for the time period covered by the next three expected fixes. In open ocean conditions, hourly fixes are sufficient; in coastal pilotage, three-minute fixes are common.

Specific DR positions, which are sometimes called *DR*s, are plotted according to the *Rules of DR*:

- DR at every course change
- DR at every speed change
- DR every hour on the hour
- DR every time a fix or running fix is obtained
- DR three hours ahead or for the next three expected fixes
- DR for every line of position (LOP), either visual or celestial

For example, the navigator plots these DRs:



Notice that the 1523 DR does not coincide with the LOP at 1523. Although note is taken of this variance, one line is insufficient to calculate a new fix.

The Mapping Toolbox includes the function `dreckon`, which calculates the DR positions for a given set of courses and speeds. The function provides DR positions for the first three rules of dead reckoning. The approach is to provide a set of waypoints in navigational track format corresponding to the plan of intended movement.

The time of the initial waypoint, or fix, is also needed, as well as the speeds to be employed along each leg. Alternatively, a set of speeds and the times for which each speed will apply can be provided. dreckon will return the positions and times required of these DRs:

- dreckon calculates the times for positions of each course change, which will occur at the waypoints
- dreckon calculates the positions for each whole hour
- if times are provided for speed changes, dreckon calculates positions for these times, if they do not occur at course changes

Imagine you have a fix at midnight at the point (10 N,0):

```
waypoints(1,:) = [10 0]; fixtime = 0;
```

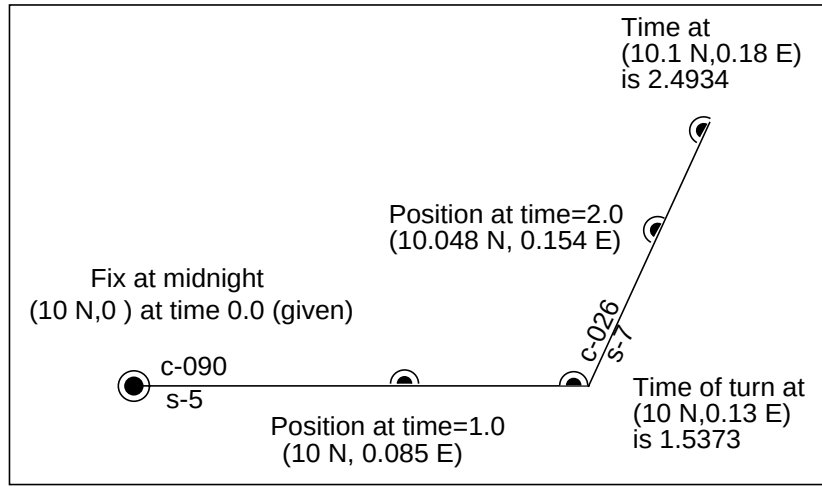
You intend to travel east and alter course at the point (10 N,0.13 E) and head for the point (10.1 N,0.18 E). On the first leg, you will travel at 5 knots, and on the second leg you will speed up to 7 knots.

```
waypoints(2,:) = [10 .13];
waypoints(3,:) = [10.1 .18];
speeds = [5;7];
```

To determine the DR points and times for this plan, use dreckon:

```
[drlat,drlon,drtime] = dreckon(waypoints,fixtime,speeds);
[drlat drlon drtime]
ans =
   10.0000    0.0846    1.0000     % Position at 1 am
   10.0000    0.1301    1.5373     % Time of course change
   10.0484    0.1543    2.0000     % Position at 2 am
   10.1001    0.1801    2.4934     % Time at final waypoint
```

Here is an illustration of this track and its DR points:

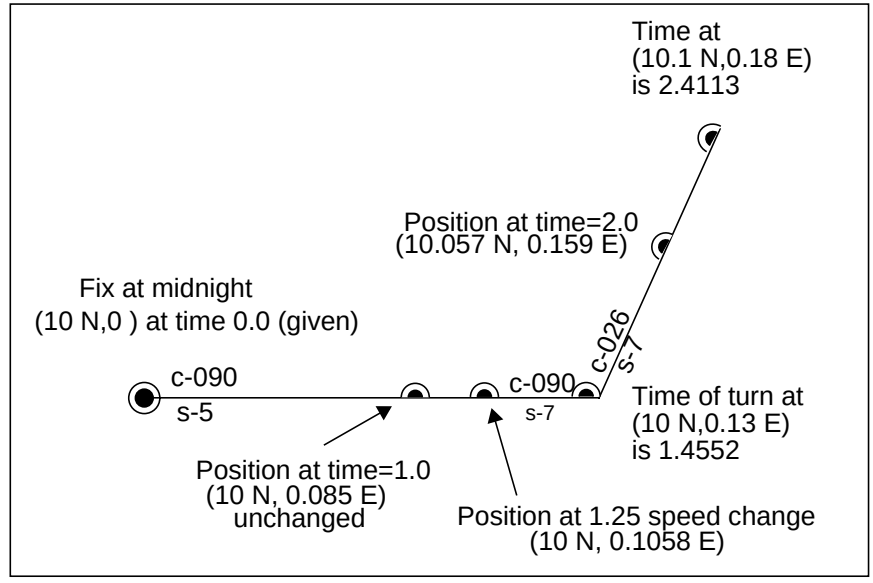


However, you would like to get to the final point a little earlier to make a rendezvous. You decide to recalculate your DRs based on speeding up to 7 knots a little earlier than planned. The first calculation tells you that you were going to increase speed at the turn, which would occur at a time 1.5373 hours after midnight, or 1:32 a.m. (at time 0132 in navigational time format). What time would you reach the rendezvous if you increased your speed to 7 knots at 1:15 a.m. (0115, or 1.25 hours after midnight)?

To indicate times for speed changes, another input is required, providing a time interval after the fix time at which each ordered speed is to end. The first speed, 5 knots, is to end 1.25 hours after midnight. Since you don't know when the rendezvous will be made under these circumstances, set the time for the second speed, 7 knots, to end at infinity. No DRs will be returned past the last waypoint.

```
spdtimes = [1.25; inf];
[drlat,drlon,drtime] = dreckon(waypoints,fixtime,...
                                speeds,spdtimes);
[drlat,drlon,drtime]
ans =
   10.0000    0.0846    1.0000   % Position at 1 am
   10.0000    0.1058    1.2500   % Position at speed change
   10.0000    0.1301    1.4552   % Time of course change
   10.0570    0.1586    2.0000   % Position at 2 am
   10.1001    0.1801    2.4113   % Time at final waypoint
```

This following illustration shows the difference:



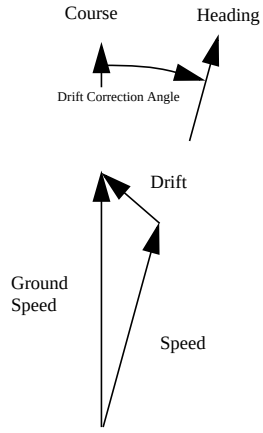
The times at planned positions after the speed change are a little earlier; the position at the known time (2 a.m.) is a little farther along. With this plan, you will arrive at the rendezvous about 4 1/2 minutes earlier, so you may want to consider a more authoritative speed change.

Drift Correction

Dead reckoning is a reasonably accurate method for predicting position if the vehicle is able to maintain the planned course. Aircraft and ships can be pushed off the planned course by winds and current. An important step in navigational planning is to calculate the required drift correction.

In the standard drift correction problem, the desired course and wind are known, but the heading needed to stay on course is unknown. This problem is well suited to vector analysis. The wind velocity is a vector of known magnitude and direction. The vehicle's speed relative to the moving air mass is a vector of known magnitude, but unknown direction. This heading must be chosen so that the sum of the vehicle and wind velocities gives a resultant in the specified course direction. The ground speed may be larger or smaller than the air speed

because of headwind or tailwind components. A navigator would like to know the required heading, the associated wind correction angle, and the resulting ground speed.



What heading puts an aircraft on a course of 250˚ when the wind is 38 knots from 285˚ ? The aircraft flies at an airspeed of 145 knots.

```
course = 250; airspeed = 145; windfrom = 285; windspeed = 38;
[heading,groundspeed,windcorrangle] =
driftcorr(course,airspeed,windfrom,windspeed)

heading =

        258.65


groundspeed =

        112.22


windcorrangle =

          8.65
```

The required heading is about 9° to the right of the course. There is a 33-knot headwind component.

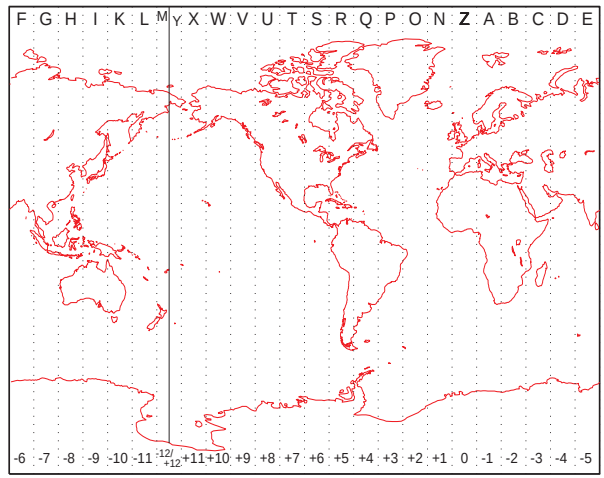
A related problem is the calculation of the wind speed and direction from observed heading and course. The wind velocity is just the vector difference of the ground velocity and the velocity relative to the air mass.

```
[windfrom,windspeed] =
driftvel(course,groundspeed,heading,airspeed)

windfrom =

        285.00


windspeed =

         38.00
```

Time Zones

Early in this manual, we gave an example of a time zone calculation. The `timezone` function returns a navigational time zone, that is, one based solely on longitude with no regard for statutory divisions. So, for example, Chicago, Illinois, lies in the statutory U.S. Central time zone, which has irregular boundaries devised for political or convenience reasons. However, from a navigational standpoint, Chicago's longitude places it in the *S* (Sierra) time zone. The zone's *description* is +6, which indicates that 6 hours must be added to local time to get Greenwich, or *Z* (Zulu) time. So, if it is noon, standard time in Chicago, it is 12+6, or 6 p.m. at Greenwich.

Each navigational time zone is 15 of longitude in width and has a distinct description and designating letter. The exceptions to this are the two zones on either side of the Date Line, *M* and *Y* (Mike and Yankee). These zones are only 7-1/2 wide, since on one side of the Date Line, the description is +12, and on the other, it is -12. Navigational time zones are very important for celestial navigation calculations. Although the Mapping Toolbox does not contain any functions designed specifically for celestial navigation, a simple example can be devised.



It is possible with a sextant to determine *local apparent noon*. This is the moment when the Sun is at its zenith from your point of view. At the exact center longitude of a time zone, the phenomenon occurs exactly at noon, local time. Since the Sun traverses a 15 time zone in 1 hour, it crosses one degree every 4 minutes. So, if you observe local apparent noon at 11:54, you must be 1.5 east of your center longitude.

You must know what time zone you are in before you can even attempt a fix. This concept has been understood since the spherical nature of the Earth was first accepted, but early sailors had no ability to keep accurate time on ship, and so were unable to determine their longitude. The engineering exploitation of this scientific truism had to wait until the invention of the chronometer.

The `timezone` function is quite simple. It returns the description, `zd`, an integer for use in calculations, a string `zltr` of the zone designator, and a string fully naming the zone. For example, the information for a longitude 123 E is the following:

```
[zd,zltr,zone] = timezone(123)
zd =
    -8
zltr =
H
zone =
    -8 H
```

Returning to our simple celestial navigation example, the center longitude of this zone is:

```
-(zd*15)
ans =
   120
```

This means that at our longitude, 123 E, we should experience local apparent noon at 11:48 a.m., 12 minutes early.

Time Notation

Navigational practice has its own peculiar notation for times. Time labels on navigation plots are always in a special format. Times are given in four digits, hours from 00 to 23 followed by minutes from 00 to 59. So, one minute before noon is 1159, or 1159Z or 1159Q, etc., based on time zone. Similarly, one minute after midnight is 0001. When more precision is required, the seconds are rounded to the nearest quarter minute and zero, one, two or three apostrophes are suffixed to the time, one for each 15-second block. So, 15 seconds before noon would be 1159'"; 14 seconds before noon would have the exact same notation.

The Mapping Toolbox includes the function `time2str` that returns a string in a variety of formats corresponding to a given time. These strings can then be plotted on map displays as desired. Two other clock formats are also allowed – the 12-hour and the 24-hour digital clock readouts. Consider some string notations for the time 13.21 hours after midnight. The default 24-hour clock is:

```
time2str(13.21)
ans =
13:12:36
```

The 12-hour clock reads:

```
time2str(13.21,'12')
ans =
01:12:36 PM
```

And the navigation format for this time is:

```
time2str(13.21,'nav')
ans =
1312''
```

Each of these can be rounded to the nearest minute with the third argument 'hm' (for hours-minutes – the default is 'hms'):

```
time2str(13.21,'nav','hm')
ans =
1313
```

7

GUI Tools

An Overview 7-2

The Complete GUI Environment: maptool 7-3
Starting maptool 7-3
The Menus 7-4
The Mouse Tool Buttons 7-4
Example One: Creating a World Map 7-5
Example Two: Creating a Regional Map 7-15

Constructing Personalized Map Data with GUIs 7-25
Trimming an Existing Data Set 7-25
Encoding a Regular Surface Map 7-27
Creating a Personalized Colormap 7-34

An Overview

The Mapping Toolbox includes a number of graphical user interface (GUI) tools that provide an alternative to entering toolbox commands from the MATLAB command window. The *Mapping Toolbox* GUIs cover most, but not all, aspects of the mapping process.

The *Mapping Toolbox* GUI system is a collection of independent GUIs, most of which can be activated from the command line and/or from a map display window at any time during a MATLAB session. The Mapping Toolbox also provides `maptool`, a comprehensive tool that allows access to most of the toolbox's independent GUIs through a system of figure window menus and buttons.

Most of the Mapping Toolbox GUIs include a **Help** button used to enter help mode for that particular GUI. Once in help mode, pressing any enabled control will display the help text for that control in a separate help box. Pressing the **Done** button will close the help box and end help mode.

This chapter is a short tutorial on several of the Mapping Toolbox GUI tools. For a more detailed description of each tool, consult the "GUI Reference" chapter of the online Mapping Toolbox reference documentation.

The Complete GUI Environment: maptool

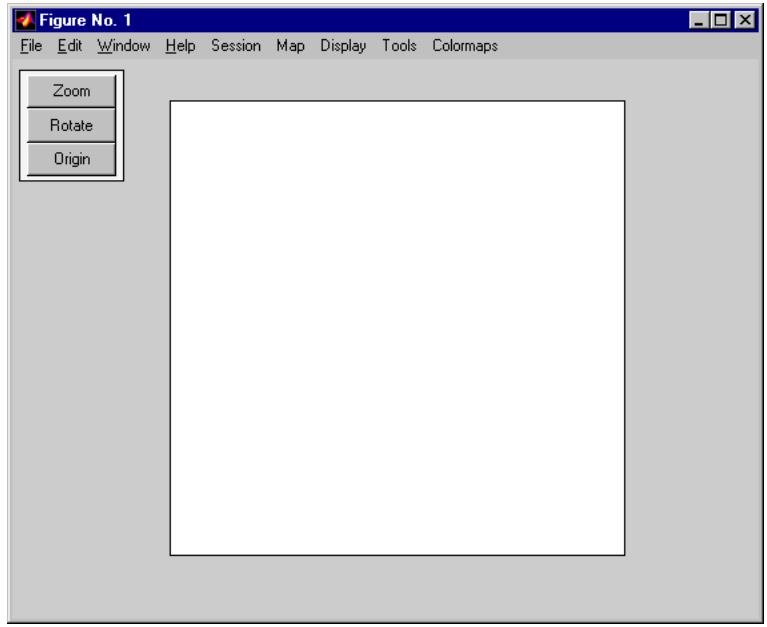
In this section, `maptool` is used to access many of the Mapping Toolbox GUIs. It should be noted, however, that most of the Mapping Toolbox GUIs can be activated independently at any time during a mapping session without the use of `maptool`. The "GUI Reference" chapter of the online Mapping Toolbox reference documentation provides details on the methods for activating each tool.

Starting maptool

You can start `maptool` in one of three ways:

```
maptool
maptool MapProjectionName
maptool('MapPropertyName','MapPropertyValue')
```

If a figure does not already exist, starting maptool creates a new figure window with additional pull-down menus and mouse buttons. If a figure window already exists, starting maptool adds the menus and mouse buttons to the existing figure.



If an initial map projection is not provided at the command line, the **Projection Control** dialog box will appear, with the Robinson projection as the default. This dialog is used to define the map axes settings. All mapping functions activated through maptool will be associated with this map axes. If maptool is applied to a figure that already has a map axes, the menus and mouse buttons are simply added to the existing figure window, and the **Projection Control** box does not appear.

In the upper left corner of the figure is a set of three mouse tool buttons that allow interactive display manipulation.

The Menus

There are five pull-down menus associated with `maptool`.

- The **Session** menu manipulates the workspace and sets the figure window renderer for the `maptool` session.
- The **Map** menu command activates GUIs that are used to project objects onto the map axes.

- The **Display** menu edits map axes properties, manipulates display settings, calculates and displays navigational tracks, small circles, surface distances and map distortion, and provides print preview.
- The **Tools** menu provides a variety of miscellaneous mapping tools, including means to hide and delete mapped objects, and adjust axes display properties.
- The **Colormaps** menu allows for manipulation of the colormap for the current figure.

The Mouse Tool Buttons

There are three mouse tool buttons associated with `maptool`.

- The **Zoom** button toggles *Zoom* mode on and off. Zoom mode zooms in on a two-dimensional map display.
- The **Rotate** button toggles *Rotate 3-D* mode on and off. Rotate 3-D interactively rotates the view of a three-dimensional plot.
- The **Origin** button toggles *Origin* mode on and off. Origin mode interactively modifies the map origin.

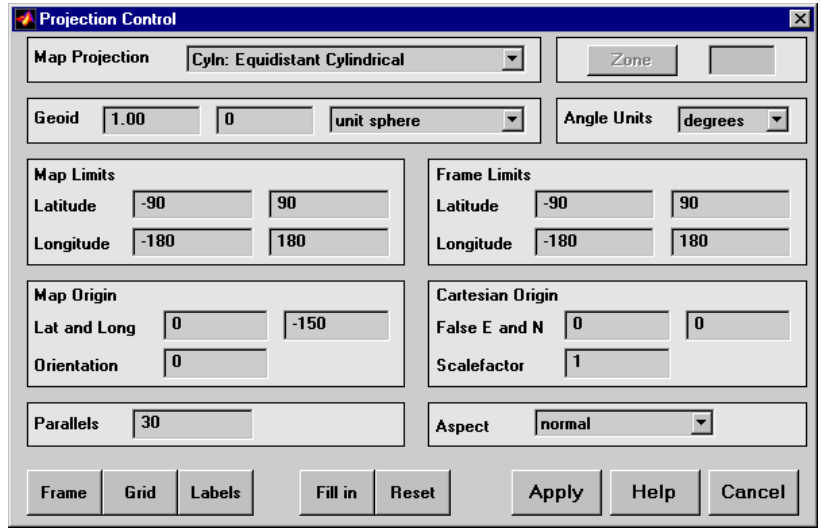
For more details on the `maptool` menus and buttons, see the "GUI Reference" chapter of the online Mapping Toolbox reference documentation.

Example One: Creating a World Map

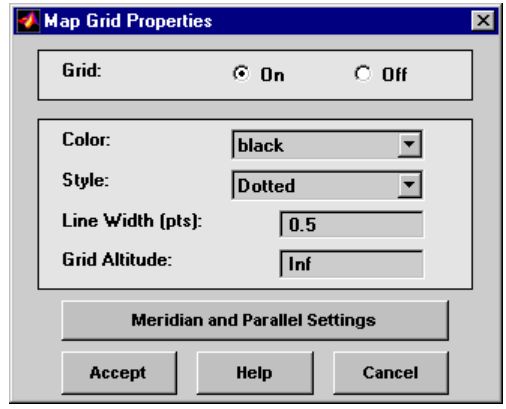
The Mapping Toolbox comes with world atlas data in the `worldlo` workspace. This data will be used in the following example to create a simple line map of the world. The first step is to create a map axes. From the MATLAB command prompt, type:

```
maptool
```

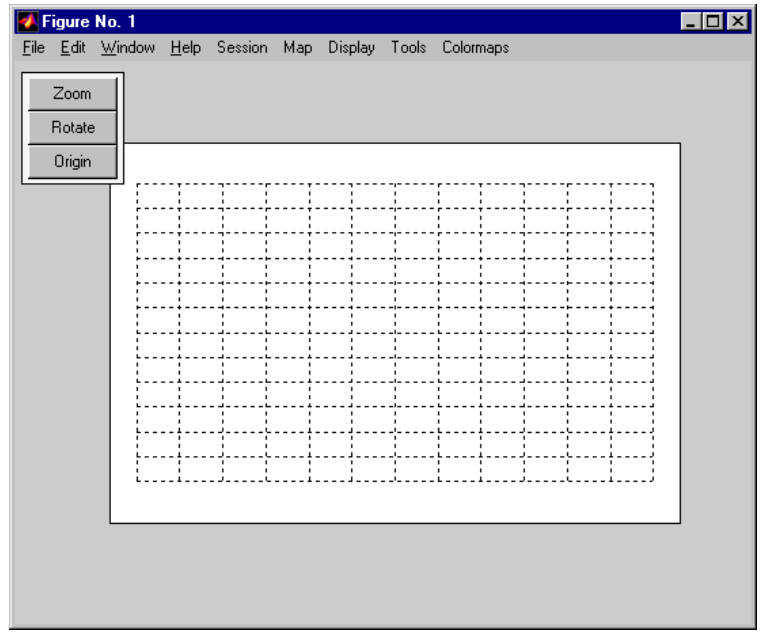
This creates a new figure window and brings up the **Projection Control** dialog box. Use the Map Projection pull-down menu to define an Equidistant Cylindrical projection and change the map origin longitude to center the map at 150 degrees West:



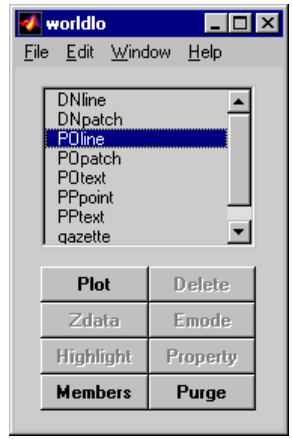
Press the **Grid** button to activate the **Map Grid Properties** dialog box and turn the grid on:



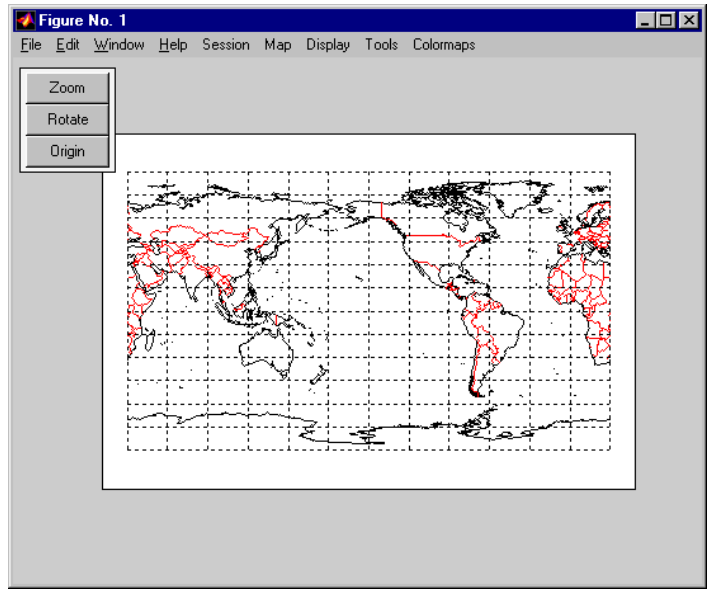
Press **Accept** to return to the **Projection Control** box, and press **Apply** to apply the settings to the map axes:



Select **Session⇒Layers⇒World LoRes** from the menu bar to activate the `mlayers` GUI with the `worldlo` workspace. This brings up the **worldlo** dialog box, which lists all the structures in the `worldlo` workspace:



Select `POline` in the list box, and press the **Plot** button to display political/ ocean boundaries. Your figure now displays coastlines and international boundaries:



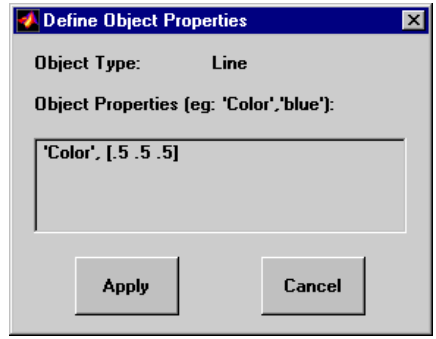
Close the **worldlo** dialog box. The `worldlo` variables associated with the dialog box have been removed from the workspace, but the plotted objects remain on the map.

To view and edit properties of currently mapped objects, use the `mobjects` GUI. Select **Tools⇒Objects** from the menu bar. The **Object Sets** dialog box appears, which lists all the objects on the current map axes. An asterisk(*) next to an object indicates that it is visible, and an *h* next to an object indicates that it is currently hidden.

Select `International Boundary` and press the **Delete** button. A **Confirm Deletion** dialog box appears. Deleting an object set removes it from the current map axes. Press **Yes** to continue the deletion process.

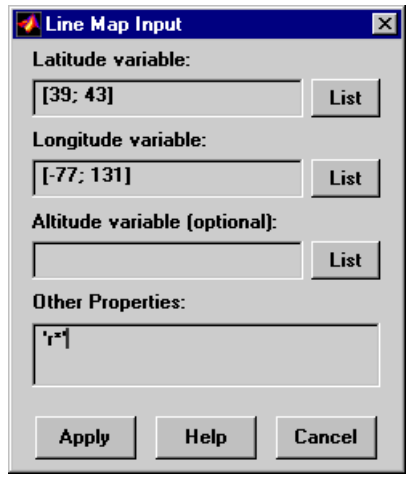
The figure now displays only coastlines. Change the coastlines from black to gray by selecting `Coastline` from the list box and pressing the **Property** button. The **Define Object Properties** box appears. Multiple pairs of property names and values can be entered in this dialog box, but you would simply like to change the color of your coastlines from black to gray.

Type `'Color',[.5 .5 .5]` in the **Object Properties** edit box, and press **Apply**:

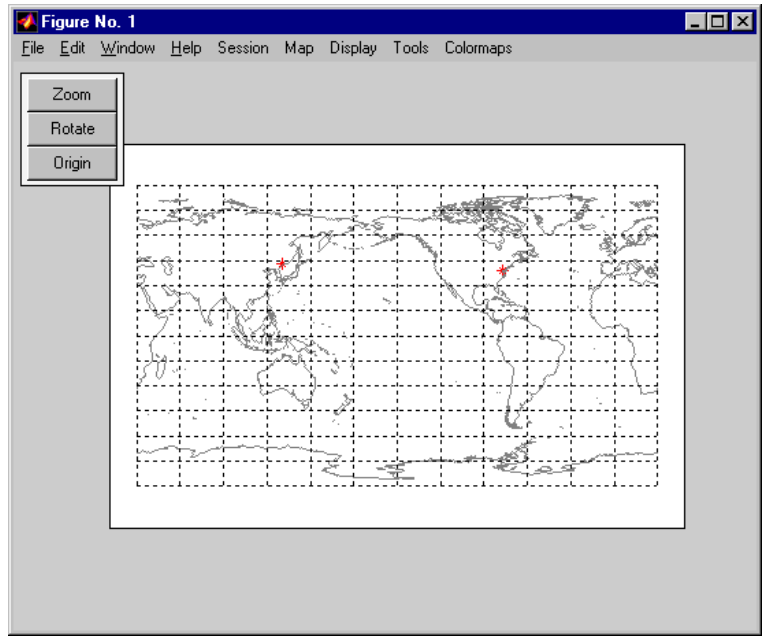


The coastlines change to gray. Close the **Objects Sets** dialog box.

Next we will plot two cities, Washington, D.C. and Vladivostok, Russia. Choose **Map⇒Lines** from the menu bar. The **Line Map Input** box appears. Enter the latitude vector `[39; 43]` and the longitude vector `[-77; 131]`. Note that you may also enter workspace variables as the latitude and longitude values. In the **Other Properties** edit box, type `'r*'` to plot a red marker at each city location. Press **Apply**.



The city markers now appear on the map:

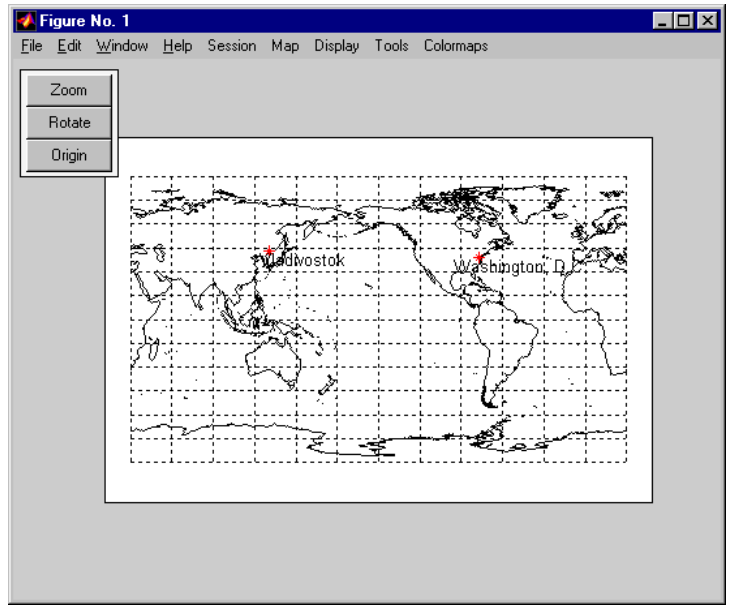


Select **Session⇒Command** from the menu bar. The **Workspace Commands** dialog box appears. This dialog box allows command line statements to be executed directly from `maptool`, rather than from the MATLAB command window. To plot city labels on the map, use the `gtextm` interactive tool, which will position strings at locations on the map specified by a mouse click. Enter the following in the **Workspace Commands** edit box:

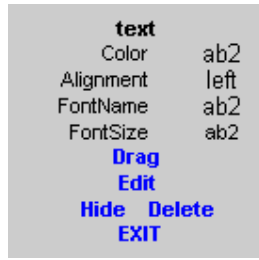
```
gtextm(strvcat('Washington, D.C.','Vladivostok'))
```

Press **Apply**. A crosshair appears on the map. The first input string is `'Washington, D.C.'`, so click in the vicinity of the Washington, D.C. city marker on the map. The string will be placed at that location. Next click near the Vladivostok city marker, and the second string will be placed there.

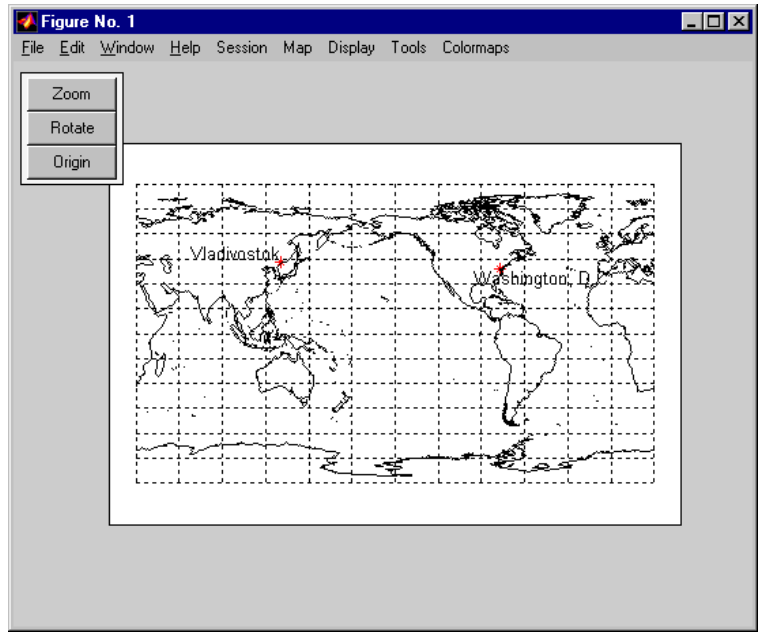
A **Status Report** box appears after each workspace command that is successfully completed. Here is the result of the `gtextm` operation:



The properties of many mapped objects can be edited with the **Click-and-Drag Property Editor** tool. Right-clicking on an object activates this functionality and allows for property manipulation through simple mouse clicks and drags. To reposition the Vladivostok label to make it easier to read, right-click on the text string. A **Click-and-Drag Property Editor** box for text objects appears:



Click **Drag**, and the label will now follow the mouse pointer as it is moved across the map display. (The **Click-and-Drag** tool can be moved out of the way by holding down the **Shift** key and dragging the box aside.) Position the label where you like, and click to place the text:



For more information on using the **Click-and-Drag Property Editor** tool, see the "GUI Reference" chapter of the online Mapping Toolbox reference documentation.

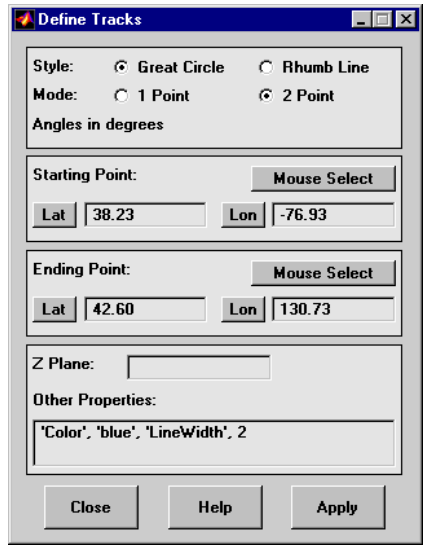
A great circle is the shortest distance between two places and can be used to determine approximate airplane flight routes. To plot a route between Washington, D.C. and Vladivostok, you can use the `trackui` tool. Select **Display⇒Tracks** from the menu bar. The **Define Tracks** dialog box appears.

Select **Great Circle, 2 Point** mode. To specify Washington, D.C. as the starting point for your first track, press the **Mouse Select** button.

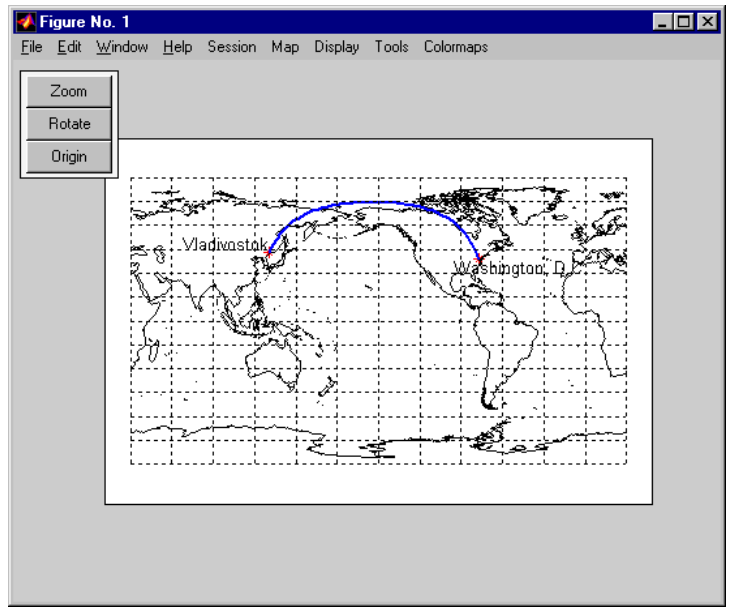
A crosshair appears on the map display. Position it over the city marker and click. The coordinates appear in the **Starting Point Lat and Lon** boxes. Using the **Ending Point Mouse Select** button, select Vladivostok as your ending point.

In the **Other Properties** edit box, type

```
'Color','blue', 'Linewidth',2
```



Press Apply, and the flight track is displayed:



If you would like to save your map, select **File⇒Save** from the menu bar to save the figure window in an M-file. The **Zoom** and **Rotate** buttons should be turned off before saving. When you execute the M-file, the figure and uicontrols will be recreated to their original appearance.

Example Two: Creating a Regional Map

Another way to easily plot `worldlo` atlas data is by using the `worldmap` tool. `worldmap` plots a specified region using appropriate default projection parameters and map axes settings. `worldmap` can be used to plot individual countries or entire regions. Here are some examples of valid `worldmap` commands:

```
worldmap france
worldmap asia
worldmap 'north pole'
```

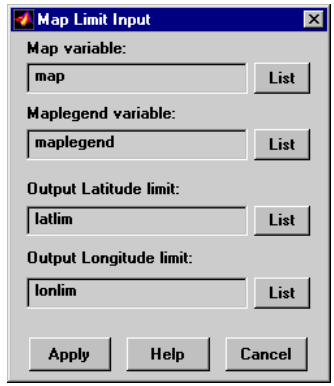
`worldmap` also accepts latitude and longitude limits to define the area to be plotted. The following example uses `worldmap` to create a map of North and South Korea. The maptool menu will then be added to the `worldmap` figure and GUIs will be used to add a digital elevation map and plot the locations of earthquake epicenters. The `korea` and `koreaEQdata` workspaces used in this example are shipped with the Mapping Toolbox. The first step is to load the data we will be using:

```
load korea
load koreaEQdata
who

Your variables are:
kdata       map         maplegend
```

To plot North and South Korea, first define the geographic limits of the area with the `limitm` tool. Enter the following at the MATLAB command prompt.

```
limitm
```

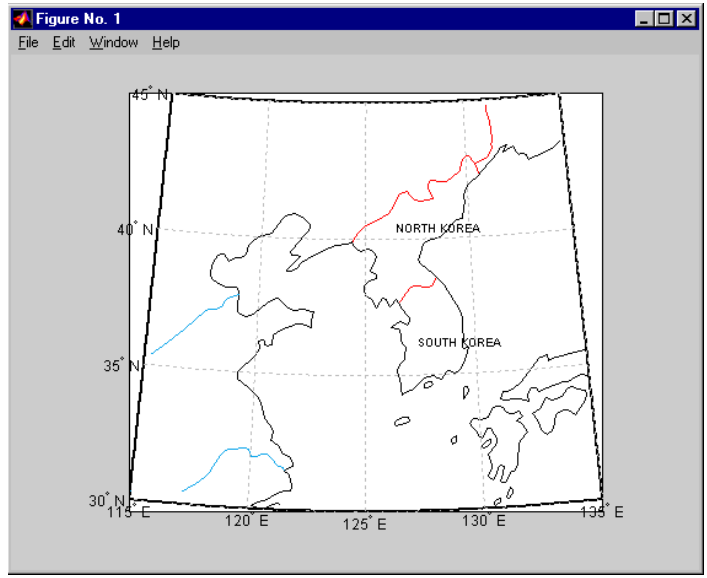
The **Map Limit Input** dialog box appears.



Accept the default variable names and press **Apply**. The `latlim` and `lonlim` variables now contain the latitude and longitude limits of the `korea` data. Now `worldmap` can be used to plot the `worldlo` data within these limits. At the MATLAB command prompt, enter:

```
worldmap('lo', latlim, lonlim)
```

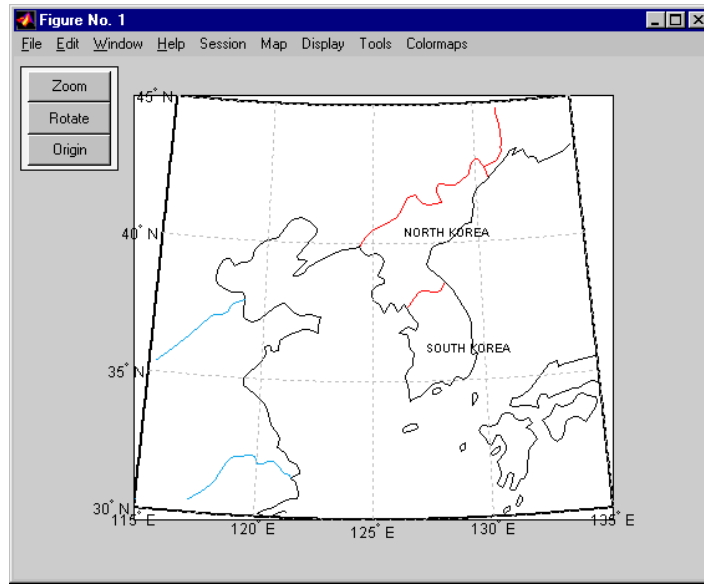
The following map is created:



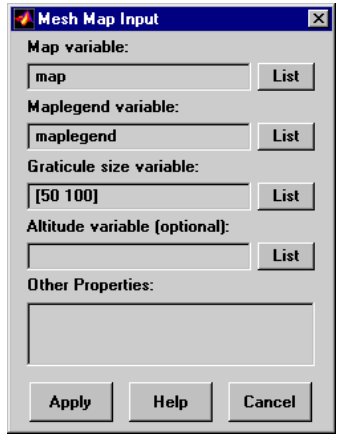
For easy access to the Mapping Toolbox GUIs, we can now add the `maptool` menu to the `worldmap` figure. At the command prompt, enter:

```
maptool
```

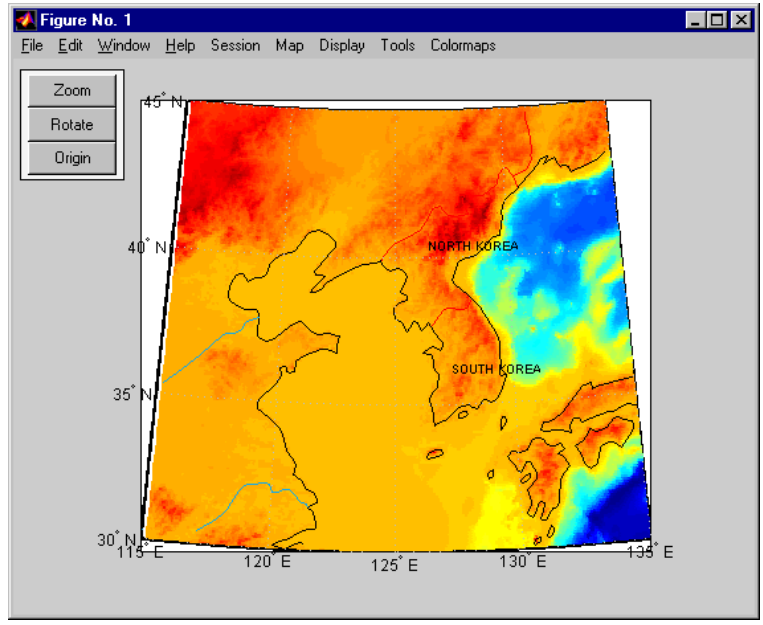
The `maptool` menu and mouse buttons are automatically added to the figure window:



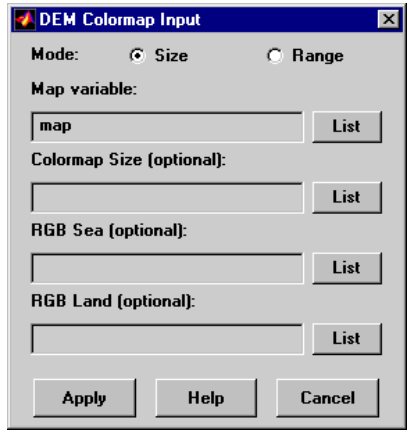
Next we will plot the digital elevation data contained in the korea variables `map` and `maplegend`. Select **Map⇒Regular Surfaces** from the menu bar. The **Mesh Map Input** dialog box appears:



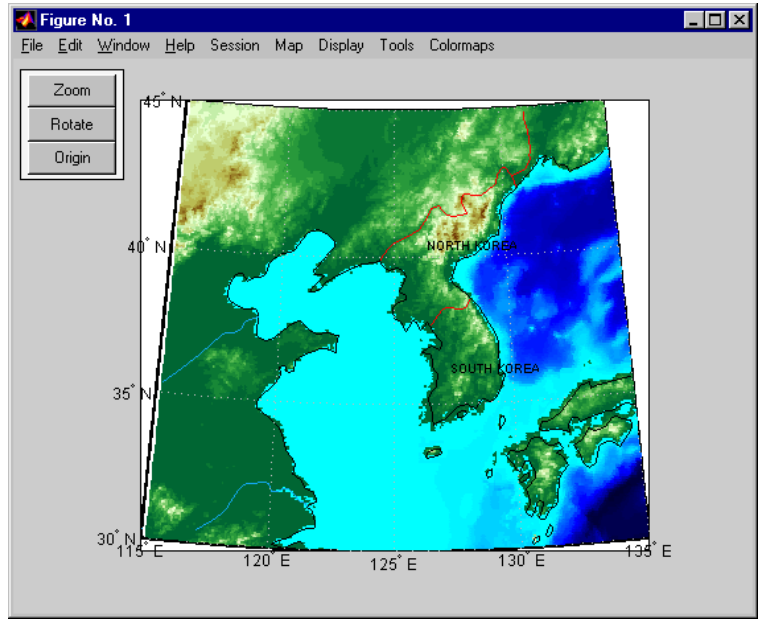
The default `map` and `maplegend` variable names match the `korea` variable names, so just press **Apply**. The surface object is added to the map:



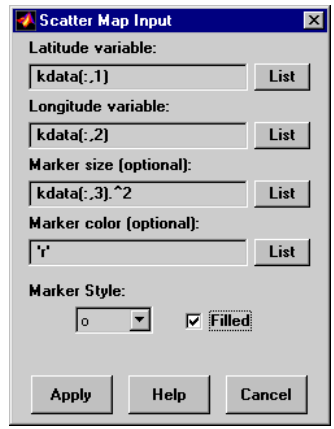
To change the colormap, select **Colormaps⇒Digital Elevation** from the menu bar. The **DEM Colormap Input** box appears. Enter the map variable `map` and press **Apply**.



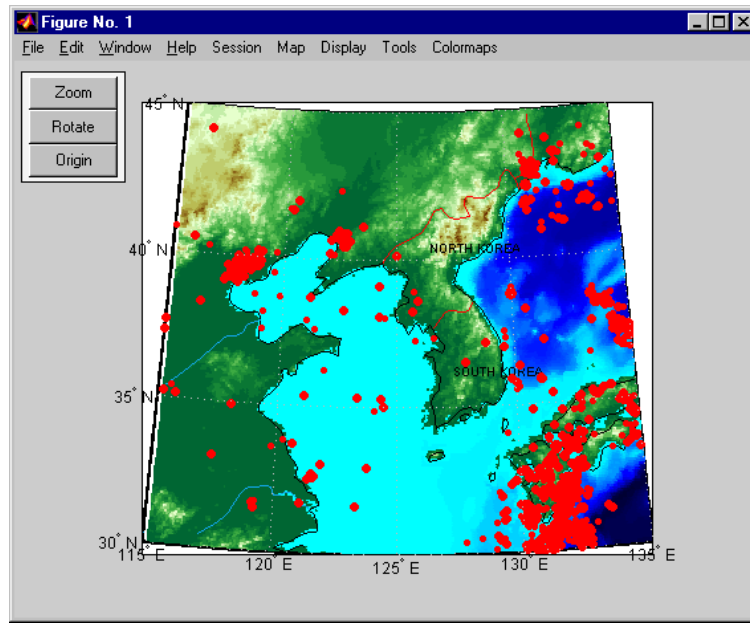
The new colormap is applied:



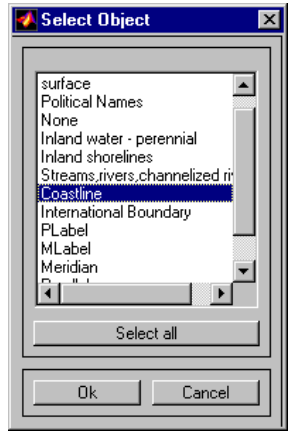
To plot the locations of earthquakes, we will use the `scatterm` GUI. Select **Map⇒Scatter** from the menu bar. The **Scatter Map Input** box appears. The variable containing the earthquake data is `kdata`. For the **Latitude variable**, enter `kdata(:,1)`. For the **Longitude variable**, enter `kdata(:,2)`. And for the **Marker size**, enter `kdata(:,3).^2`, to display the markers in proportion to the earthquake magnitude. Squaring the magnitudes increases the difference in marker sizes, making them easier to distinguish. Change the **Marker color** to 'r', and check the **Filled** box.



Press **Apply** and the earthquake markers are displayed:



To hide the coastlines, select **Tools⇒Hide⇒Object** from the menu bar. The **Select Object** dialog box appears. Select `Coastline` and press **Ok**.

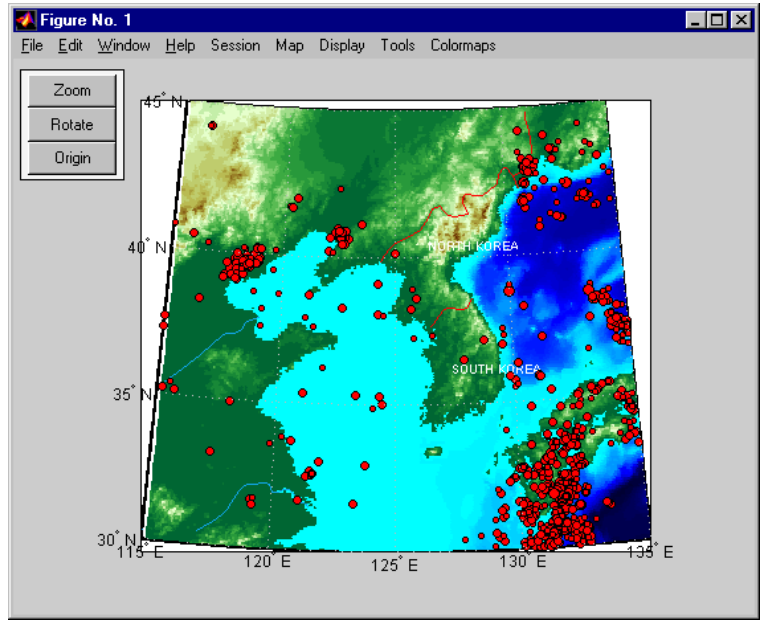


The coastlines are no longer displayed.

To make the country names a little bit easier to read, we will change the color from black to white. Select **Tools⇒Objects** from the menu bar. The **Object Sets** box appears. Select `Political Names` and press **Property**. In the **Define Object Properties** edit box, type `'Color','white'` and press **Apply**.

Next, we will change the edge color of the earthquake markers to better distinguish each earthquake site. The earthquake markers are patch objects, and are indicated in the **Object Sets** box as `patch`. Select `patch` and press **Property**. Enter `'MarkerEdgeColor','black'` and press **Apply**.

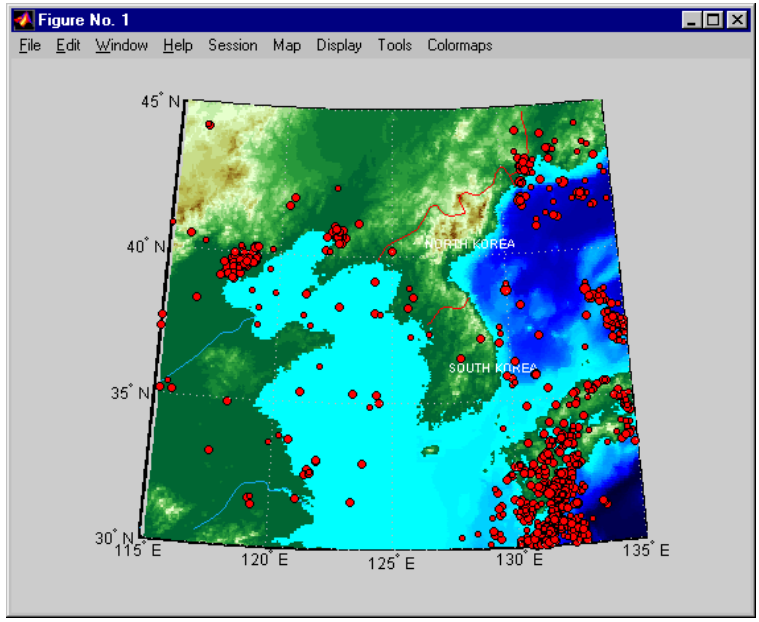
The map now appears as follows:



When we first plotted this map of korea, `worldmap` selected the Lambert Equidistant Conic projection as appropriate for the location and size of the region. Suppose you would now like to switch to an equal area projection. Select **Display⇒Projection** from the menu bar. From the **Map Projection** drop-down list, select the Equal Area Conic (Albers) projection. Press **Apply**, and all of the mapped objects are now projected onto the new projection.

Select **Tools⇒Hide** from the menu bar to hide the **Zoom**, **Rotate**, and **Origin** buttons on the figure window. Select **Tools⇒Axes⇒Invisible** from the menu bar to hide the axes box.

Here is the final map of Korean earthquake locations:



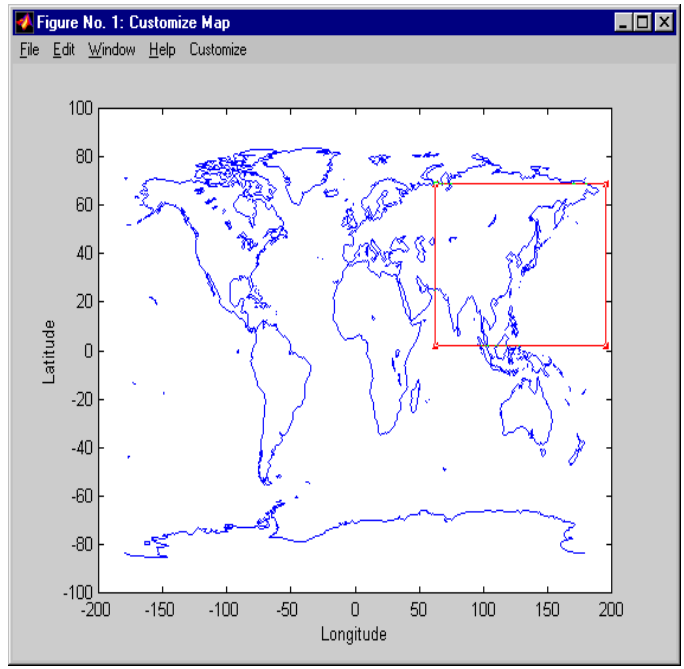
Constructing Personalized Map Data with GUIs

Trimming an Existing Data Set

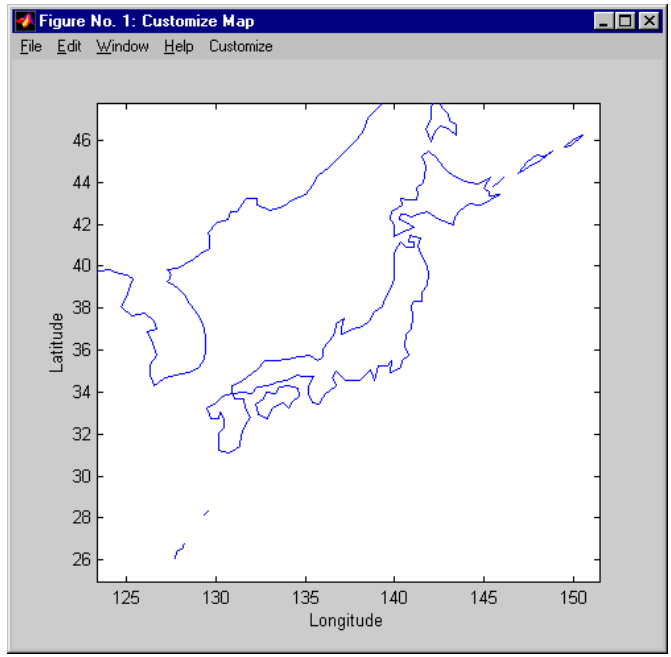
You can use the `maptrim` tool to trim a map dataset. For example, you want to work with just a portion of the coastline data found in the `coast` workspace, say the region around Japan. Use the `maptrim` tool to trim the coastline data to this region, then save the trimmed dataset as a new vector or matrix map.

```
load coast
maptrim(lat,long)
```

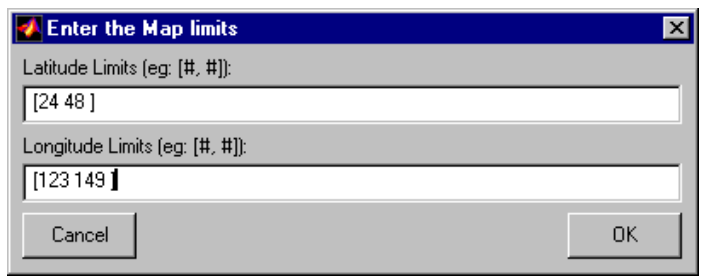
A new figure window displays the unprojected coastline data. Drag the zoom box to the region around Japan.



Double-click in the box to zoom in. Repeat this process until the area of interest fills the figure window.

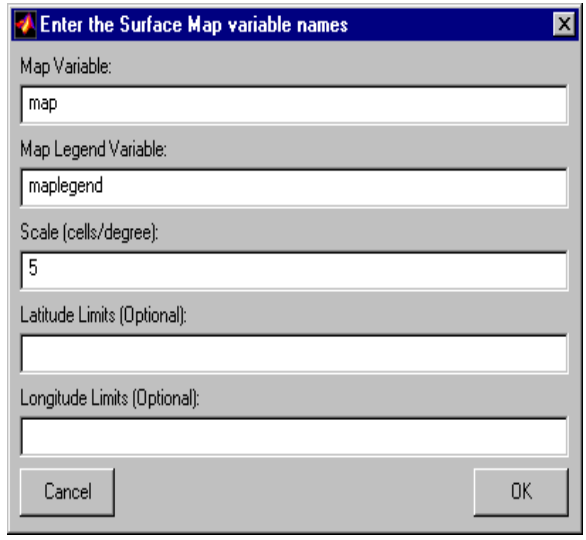


Select **Customize⇒Limits** to specify the exact limits of the region of interest. A dialog box appears with the latitude and longitude limits of the current display. Set the latitude limits to [24 48] and the longitude limits to [123 149]. Press **OK**.



To save this trimmed region as vector data, choose **Customize⇒Save As⇒Line** or **Customize⇒Save As⇒Patch** from the menu bar. A dialog box will appear prompting you to enter variable names for your new dataset.

The `maptrim` tool can also be used to convert data from vector format to matrix format. The command line equivalent function is `rec2mtx`. To convert the trimmed region around Japan, choose **Customize⇒Save As⇒Regular Surface** from the menu bar. Save the trimmed map data under the variable name `map`, and the map legend under variable name `maplegend`. Enter a scale of 5 cells per degree. Press **OK**.



You now have a regular matrix map of Japan in your workspace. The **Customize Map** figure window can now be closed.

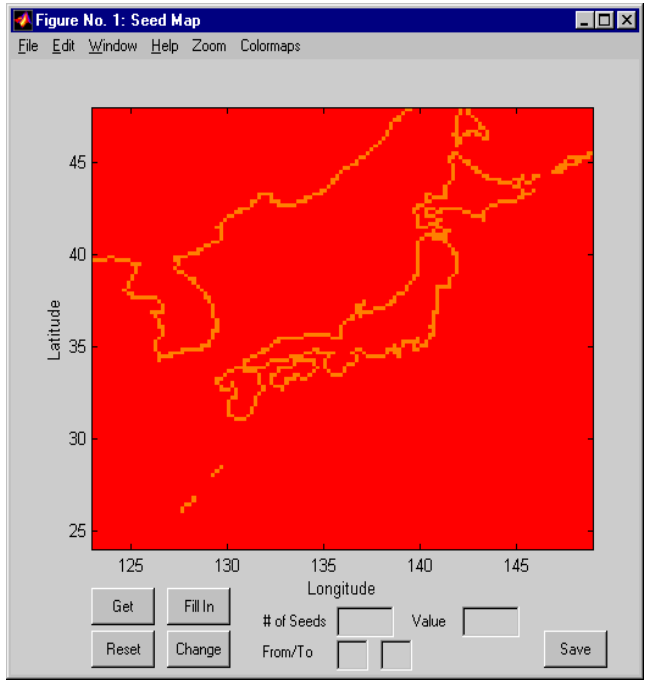
Encoding a Regular Surface Map

Encoding is the process of filling in specific values in regions of a matrix map up to specified boundaries. For example, you can fill in the interior of an island, bounded by its coastline, with the value six. Encoding entire regions at one time allows indexed maps to be created quickly. Some parts of this process are automated in the `vec2mtx` and `country2mtx` commands.

A regular matrix map can be encoded using the `seedm` tool. To encode the map of Japan just created, activate the `seedm` tool by typing the following:

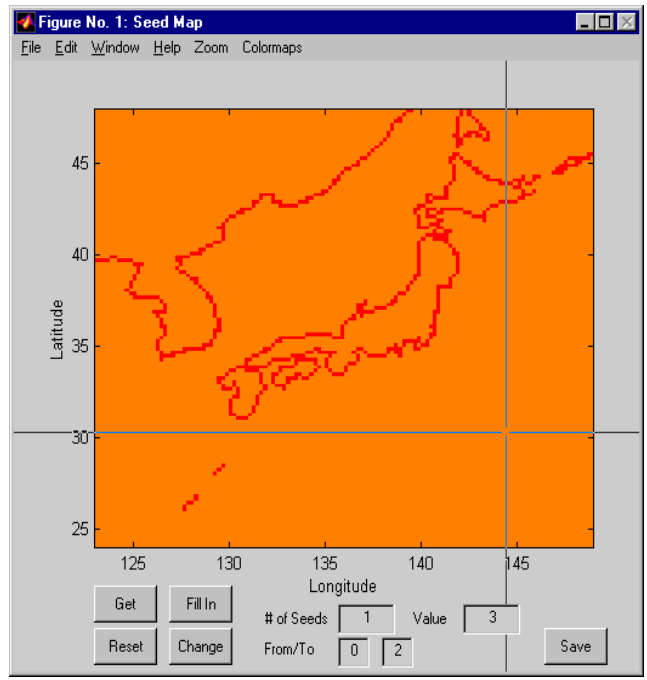
```
seedm(map,maplegend);
```

A new figure window displays the surface map of Japan.



Currently, your map consists of 1's where there is a coastline and 0's elsewhere. Suppose you would like different values for four types of data: coastlines = 1, Japan = 2, water = 3, and other countries = 4. Enter the values `0` and `2` in the **From/To** boxes and press **Change**. This will change all values of 0 to 2.

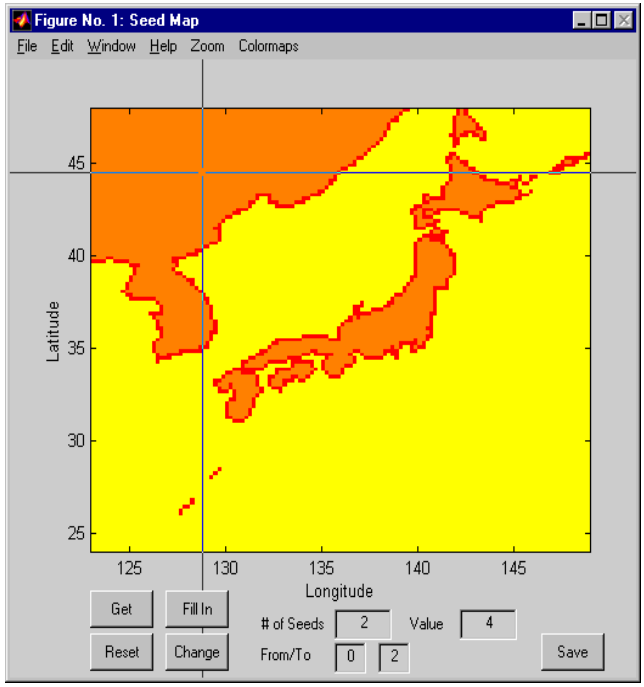
To encode the regions of water, enter a 1 in the **# of Seeds** box and a 3 in the **Value** box. Press **Get**, and a cross hair will appear on the map. Position the cross hair over a region of water and click.



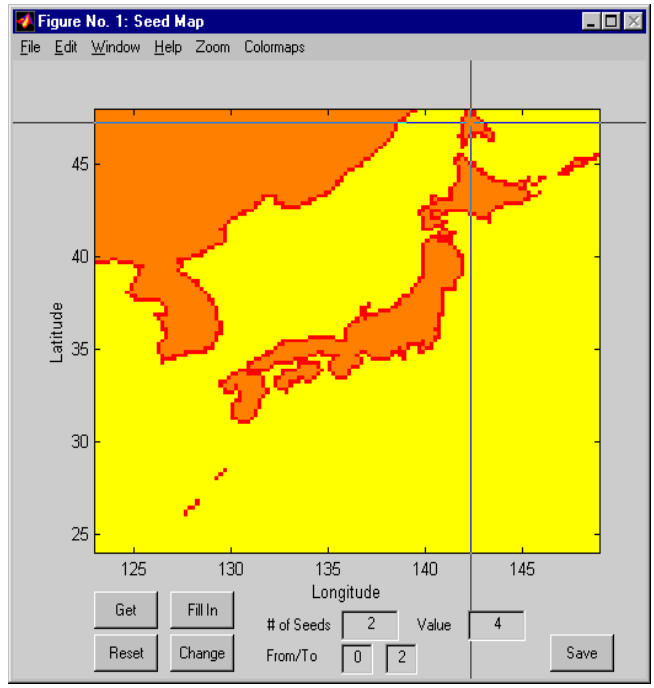
Press **Fill In**, and all values corresponding to water will be changed to 3.

Encode the other countries, there are three regions that will need to be filled with the value 4. Encode the large region of Russia, China, North Korea, and South Korea in the upper left of the map and the Russian Island of Sakhalin at the top of the map in a single step by using two seeds. Enter a 2 in the **# of Seeds box**, and a 4 in the **Value** box.

Press **Get** and the cross hair appears. Click in the first region at the upper left of the map.

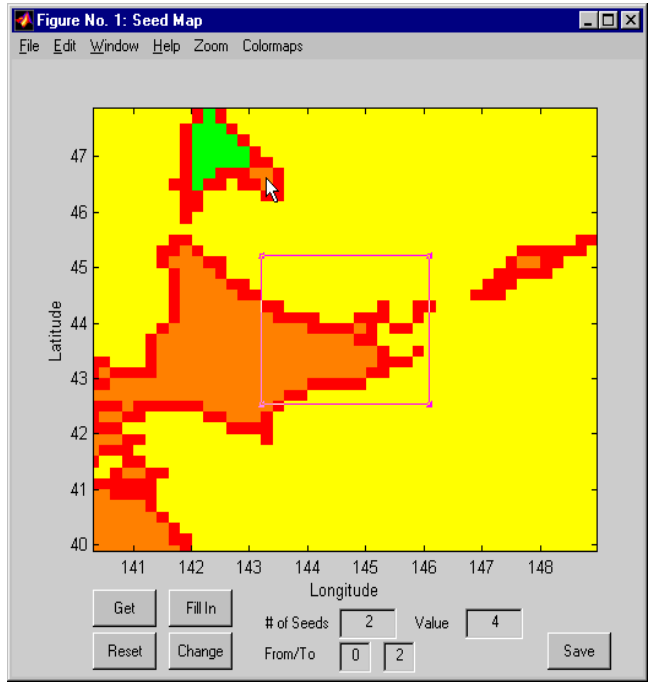


For the second seed, click inside Sakhalin Island.



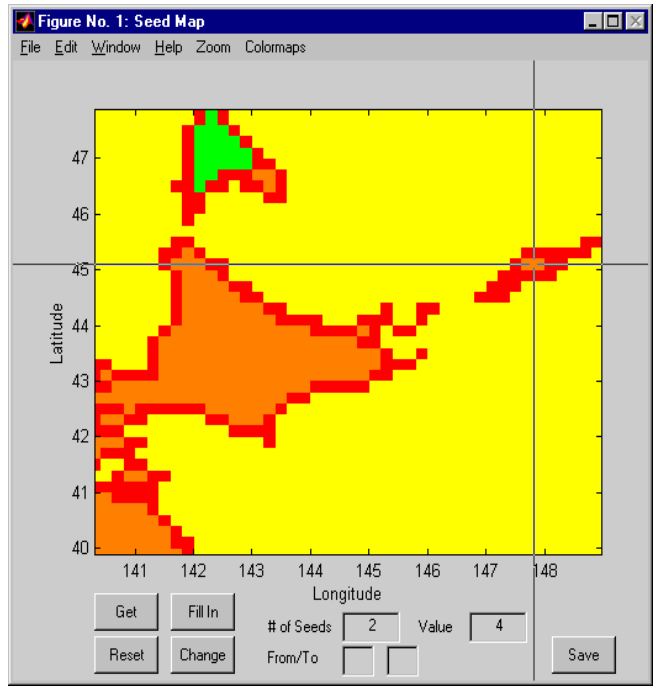
Press **Fill In**, and the values corresponding to those two regions will be changed to 4. The last region you need to encode is the small Kuril island that appears to the east of Hokkaido in the upper right corner of the map.

Drag the zoom box over the island and double-click to zoom in.

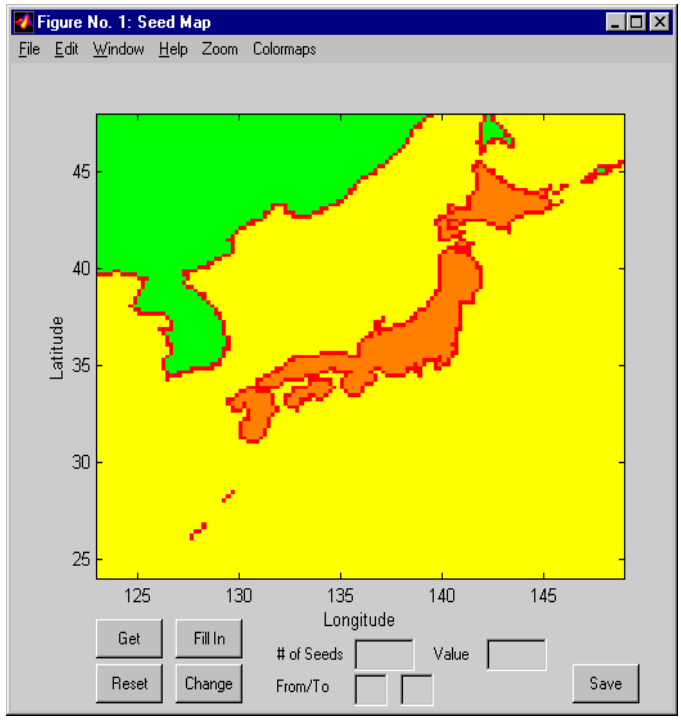


Because the island is so small, and much of its shape consists of coastline, this zoomed-in view will make it much easier to select the interior of the island, rather than its coast.

From this view, you can also see that you missed a spot on Sakhalin Island (see mouse pointer in the picture above), so you will need two seeds for this step. Enter a 2 in the **# of Seeds** box and a 4 in the **Value** box. Press **Get** and click inside the missed region of Sakhalin Island and inside the Kuril island.



Press **Fill In** and the values corresponding to those two regions will be changed to 4. The final map appears as follows:



Press the **Save** button and enter the variable name `map` to replace the old matrix map of Japan with this new encoded map. The **Seed Map** dialog box can now be closed.

Creating a Personalized Colormap

Now you can use the `colorm` tool to make a custom colormap for your map of Japan. To activate the `colorm` GUI, type:

```
colorm(map,maplegend)
```

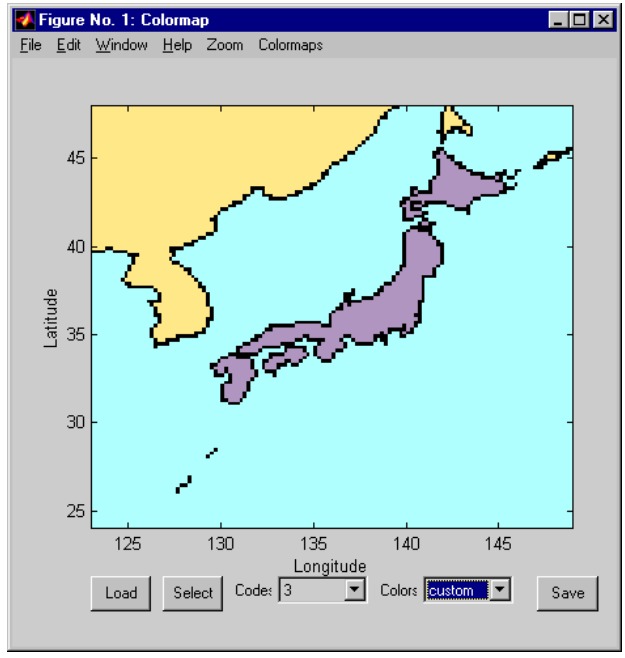
A new figure window displays the map. Recall that the values for this matrix map are as follows: coastlines = 1, Japan = 2, water = 3, and other countries = 4.

To change the coastlines to black, select `1` from the **Codes** menu, and choose `black` from the **Color** menu. Next, make Japan a light purple by selecting `2`

from the **Codes** menu and choosing `custom` from the **Color** menu. The **Custom Color** GUI appears.

Select a custom color, and choose **OK**. Japan will change to the new color. Next, change the color of all other countries to a light beige by selecting `4` from the **Codes** menu, and `custom` from the **Color** menu. Use the **Custom Color** GUI to choose a light beige color, then press **OK**.

Finally, to change the color of water to a light blue, select `3` from the **Codes** menu and `custom` from the **Color** menu. Choose a light blue from the **Custom Color** GUI and press **OK**. The map of Japan now appears:



Press the **Save** button, then press **OK** to accept the default name `clrmap` for the new colormap variable. The **Color Map** dialog box can now be closed.

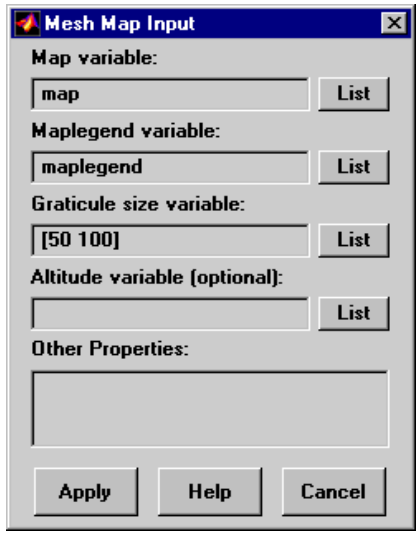
To display the map of Japan, type `axesm`.

In the **Projection Control** box, choose a Miller Cylindrical projection and define the map latitude and longitude limits to be [24 48] and [123 149], respectively.

Use the **Grid** button to turn the grid on, then press the **Parallel and Meridian Settings** button and set the longitude locations and latitude locations to 5 to display grid lines every 5 degrees.

Use the **Label** button to turn the meridian and parallel labels on. Change the label format to none. Press **Parallel and Meridian Settings** button and change the longitude and latitude label locations to 5 to display labels every five degrees. Press **Apply** to apply the settings to the map axes.

To plot the map of Japan using the **Mesh Map Input** dialog box, type `meshm`.

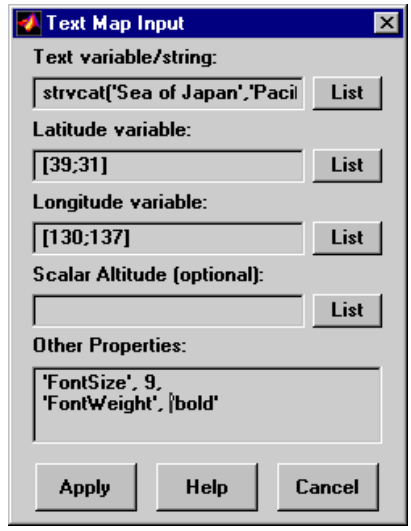


Your variable names are the same as the defaults, so press **Apply** to plot the map. After the map is plotted, type `colormap(clrmap)`. This will set the current colormap to the customized one you created. To annotate the major bodies of water in the display, type `textm`.

The **Text Map Input** dialog box appears. In the **Text variable/string** edit box, enter the following:

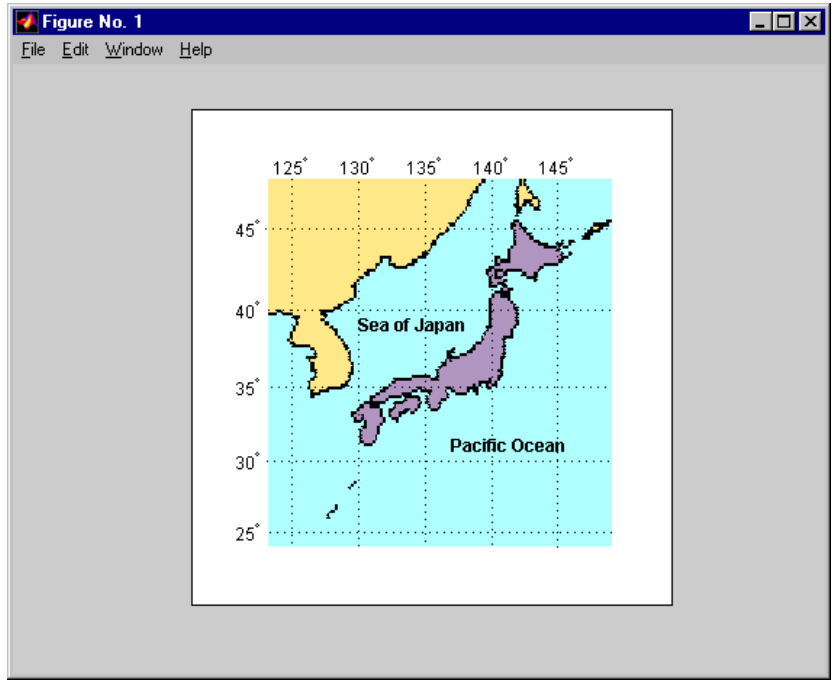
```
strvcat('Sea of Japan','Pacific Ocean')
```

In the **Latitude variable** edit box enter `[39;31]`, and in the **Longitude variable** edit box enter `[130;137]`. In the **Other Properties** edit box, enter `'FontSize',9` and `'FontWeight','bold'`.



Press **Apply** to plot the text.

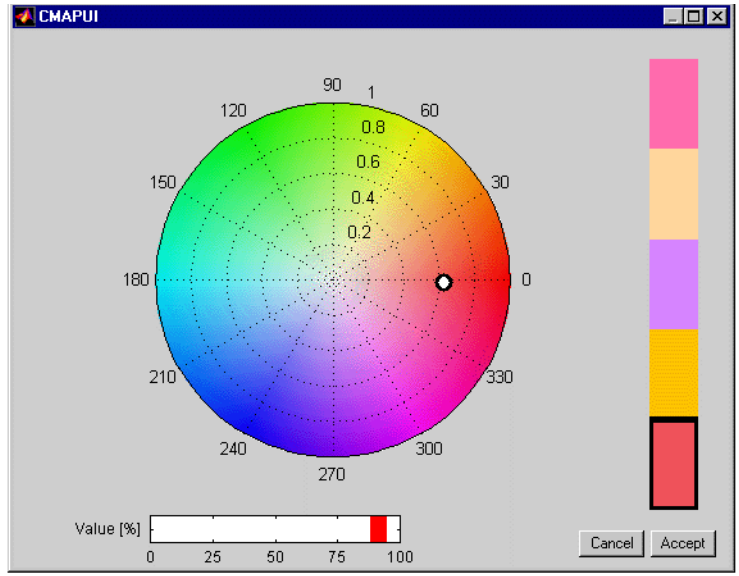
The final display of your customized map of Japan appears as follows:



The primary utility of such matrix maps is not display, but analysis. You can use this map to quickly identify whether a particular geographic location belongs to Japan, for example. See the political matrix maps section in Chapter 4, “Atlas Data” for more examples.

You can also create personalized colormaps that do not depend on a matrix map using `cmapui`. The `cmapui` panel allows you to select color entries in a colormap by clicking on the colorbar. To change the color's hue and saturation, drag the color **Marker** on the colorwheel. To control the value of the color in HSV space, drag the red **Slider**. Clicking the **Accept** button returns the modified colormap.

Here is an example of a colormap with five colors. The first color in the colormap is currently selected.


CMAPUI
90
1
120
60
0.8
0.6
150
30
0.4
0.2
180
0
210
330
240
300
270
Value [%]
0
25
50
75
100
Cancel
Accept

8

External Data Interface

Working with External Data 8-2

Global Vector Data 8-3
The Digital Chart of the World/VMAP0 8-3
Global Self-Consistent Hierarchical High-Resolution Shoreline 8-20
USGS On-Line Coastline Extractor 8-23

U.S. Vector Data 8-25
TIGER/Line 8-25
TIGER Thinned Boundary 8-29

Global Matrix Data 8-34
ETOPO5 and TerrainBase 8-34
Global Bathymetry Predicted from Satellite Altimetry 8-38
GTOPO30 and GLOBE 8-42
Digital Terrain Elevation Data Model 8-48
Reading Elevation Data Interactively 8-51
Advanced Very High Resolution Radiometer (AVHRR) Global Change Data 8-55
EGM96 Geoid Model 8-67
U. S. Gridded Elevation Data 8-68

Astronomical Data 8-74

Importing Other Data 8-80

Working with External Data

While the Mapping Toolbox atlas data described in Chapter 4 is suitable for world or regional maps, more detailed data is usually required for larger scale maps. The best sources of such data are often national mapping agencies, who make their data available to the public at little or no charge, often over the Internet or on CD-ROM. United States government publications are generally free of copyright, allowing free re-use of the data.

Although the Mapping Toolbox can easily display such geographic data once it is in the MATLAB environment, importing it is often difficult because of complicated data file formats. To make these detailed datasets readily accessible, The Mapping Toolbox provides a number of external data interface functions. These functions allow you to read the data into Mapping Toolbox data types such as regular and general matrix maps and geographic data structures.

A wide variety of data is accessible through these external interface functions. There are vector map data of political entities ranging from national and local government boundaries, infrastructure such as roads, railroads, airports and utilities, and geographic data such as coastlines, rivers, elevation contours, and land use. The Earth's surface and ocean elevations are available on regular grids at a number of different resolutions. Astronomical data in the form of star databases is also available.

Many other types of data can be imported to MATLAB beyond the specialized data file formats accessed by the external interface functions. The most important are mapped scientific data in the form of HDF files and JPEG files. See the MATLAB documentation for the `imfinfo` and `imread` functions for more information on how to read these file formats.

Global Vector Data

The Digital Chart of the World/VMAP0

The most detailed, publicly available set of global vector data with consistent coverage of important map features is the Digital Chart of the World (DCW) and the revised successor to the DCW, Vector MAP level 0. Both contain vector data scanned from printed Operational Navigation Charts (ONC) and Jet Navigation Charts (JNC). The ONC charts were compiled at a scale of 1:1,000,000 and the JNC at 1:2,000,000. The publisher of the DCW was the U. S. Defense Mapping Agency, now incorporated into the U. S. National Imagery and Mapping Agency (NIMA). NIMA publishes the VMAP0.

Both contain data for the entire world in separate layers of coverage, including political and ocean features, drainage, hypsography, land cover and vegetation, populated places, roads, railroads, airports, transportation structure, and cultural landmarks. The features are formatted as patches, lines, points, or text.

The DCW was published in 1992 on four CD-ROMs. A revised version, called VMAP0, also distributed on CD-ROMs, was released in 1997. The DCW is now out of print. For more information on purchasing VMAP0 CDs, see the `vmap0data` entry in the "External Data Reference" chapter of the online Mapping Toolbox reference documentation.

The Mapping Toolbox provides high-level functions that compile and extract information from the DCW and VMAP0 into Mapping Toolbox geographic data structures. There is also a graphical user interface to the VMAP0. The following sections introduce these functions and show how to work effectively with the data. Since the DCW is no longer available, the command line examples focus on VMAP0, but working with the DCW is very similar. The exception is the gazetteer feature, which was dropped in VMAP0. This data can be retrieved from The MathWorks FTP site.

Looking Up Names and Locations in the DCW Gazetteer

One of the best sources of place name information is the Digital Chart of the World Gazetteer. This is an extensive collection of place names and types within a library. The `NOAMER` library, containing the data for North America, has 37,696 names in its gazette. The entire DCW gazette has more than 100,000 names. It can be used to quickly find the locations of places and can help to pick the latitude and longitude limits for other DCW functions.

The Mapping Toolbox interface to the DCW gazette is the dcwgaz function. If the EURNASIA CD is available and mounted on the desktop of a Macintosh computer, you can search for the geographic location of the city of Apatin in Yugoslavia by typing:

```
dcwgaz('EURNASIA','apatin')
APATIN
ans =
             type: 'text'
    otherproperty: {1x2 cell}
              tag: 'Built up area'
           string: 'APATIN'
         altitude: []
              lat: 45.6660
             long: 18.9830
```

On other computers, you must provide a device name as the first argument. The string you provide will depend on how the CD-ROM is mounted. For a Windows PC, the command might be:

```
dcwgaz('F:','EURNASIA','apatin')
```

and on a UNIX computer with the CD-ROM mounted as '/cdrom', you should type:

```
dcwgaz('/cdrom/','EURNASIA','apatin')
```

See the documentation that came with your CD-ROM drive for more information on how disks are mounted.

If you do not have access to the DCW CDs, you can retrieve just the gazette data from The MathWorks FTP site. The gazette data for all four of the libraries has been collected into one directory called DCW_GAZETTE. If you use this version of the data, your device name will need to be the full path down to and including the DCW_GAZETTE directory.

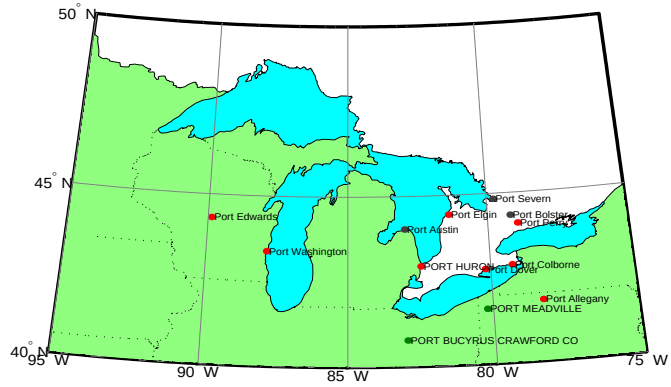
Matches are displayed on screen during the search. The output is returned in a Mapping Toolbox geographic data structure, with the tag identifying the database layer name within the DCW. If more than one entry matches the name, the results are contained in an arrayed structure.

For example, the following command prints to the screen the 95 names in the North American library that begin with the word *port* and returns the

structures `txtstruc` and `ptstruc` that can be displayed using `displaym` or `mlayers`.

```
[txtstruc,ptstruc] = dcwgaz(devicename,'NOAMER','port');
```

Here is a map that shows the “ports” around the Great Lakes:

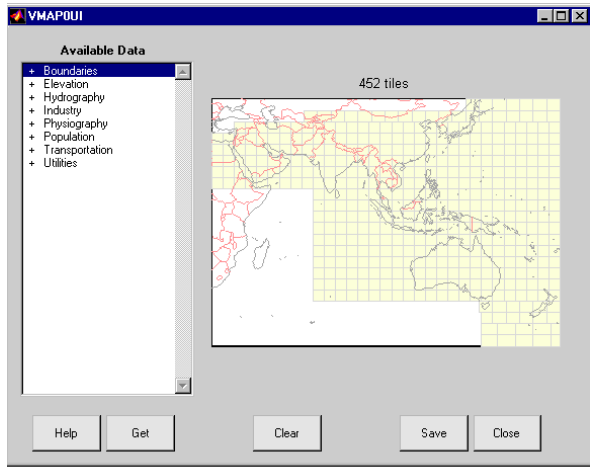


There are also databases of geographic names that you can search over the Internet. For global data, try the National Imagery and Mapping Agency GEOnet Name Server at <http://164.214.2.59/gns/html/index.html>. For the United States, try the U. S. Geological Survey Geographic Names Information System at <http://www-nmd.usgs.gov/www/gnis/>.

Importing VMAP0 Themes Interactively

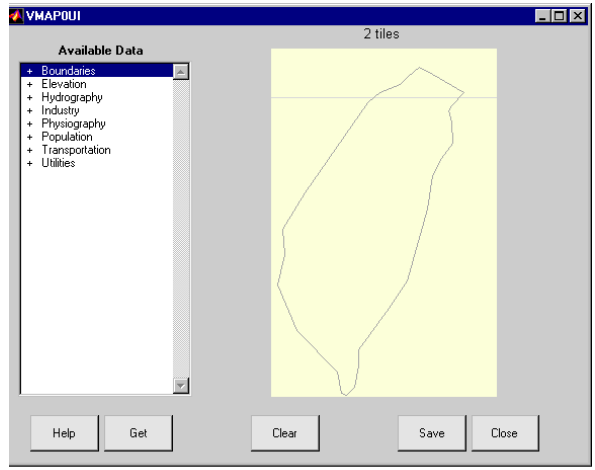
You can easily extract data from the VMAP0 CD-ROMs with the `vmap0ui` graphical user interface. Simply insert one of the CD-ROMs in your computer's CD-ROM drive, and type `vmap0ui`. If `vmap0ui` cannot find the CD, or finds the wrong one when several CDs are mounted, provide the device name of the drive.

```
vmap0ui
```

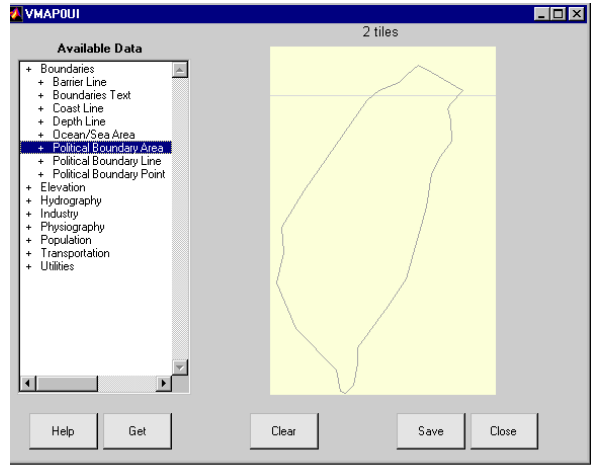


The `vmap0ui` panel shows the CD's coverage in yellow. The VMAP0 database breaks the world into small quadranges, called tiles. Each tile's boundary is shown in light gray. Just above the map is some text that shows how many tiles are in the area currently shown on the map. The list on the left shows the available data categories. These categories are called *themes*. Since this data is at a fairly high resolution, you will want to limit the kind and geographic extent of the data you extract.

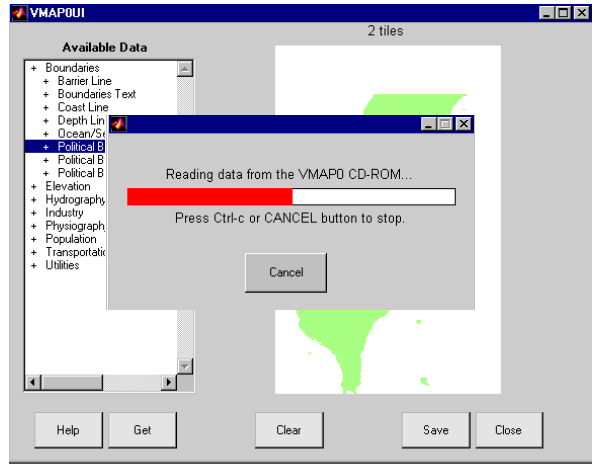
Let's go through the process of extracting some data for Taiwan. We will extract the land area, popoulation centers, and roads. First zoom into Taiwan on the map. The map is always in a zoom mode, so you can zoom in by clicking or clicking and dragging. Extend-click zooms out.



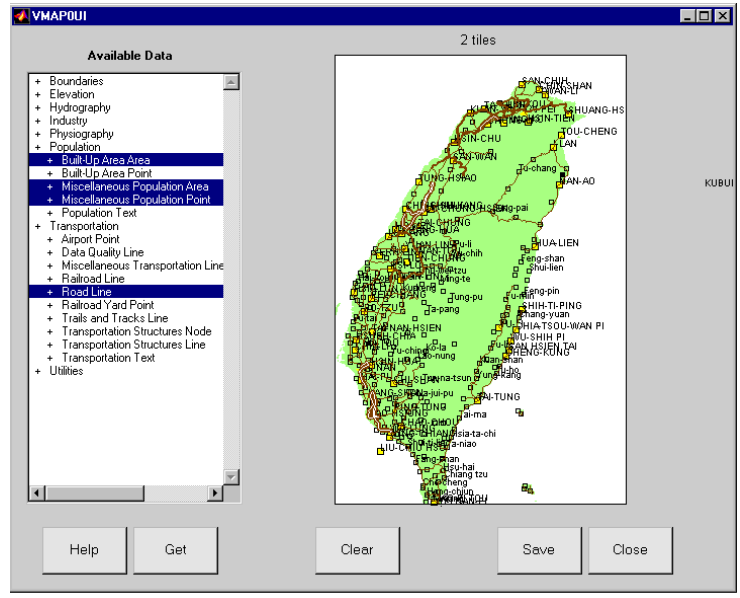
Next, select the theme from the list. Themes have subcategories called *features* that can be accessed by double-clicking on the theme. The + indicates that a theme can be opened to reveal the sub-categories. Double-click on the `Boundaries` theme, then click once to select the `Political Boundary Area` feature. We will extract just the land areas as patches. Selecting the `Boundaries` theme would extract all of the features.



Now you can extract the data by pressing the **Get** button. The progress bar shows where you are in the process, as does the map data which is added to the map display. You can interrupt the extraction by typing **Ctrl-C** or clicking the **Cancel** button. **Ctrl-C** stops the process more quickly. Clicking the **Clear** button removes any data that you have read.

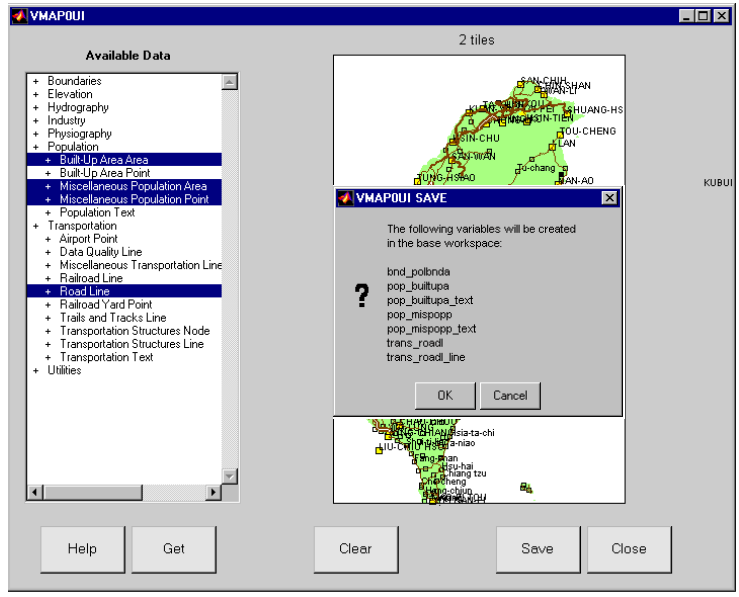


You can repeat this process for the population and transportation themes. Extract all of the population area features and the transportation theme's road line feature.



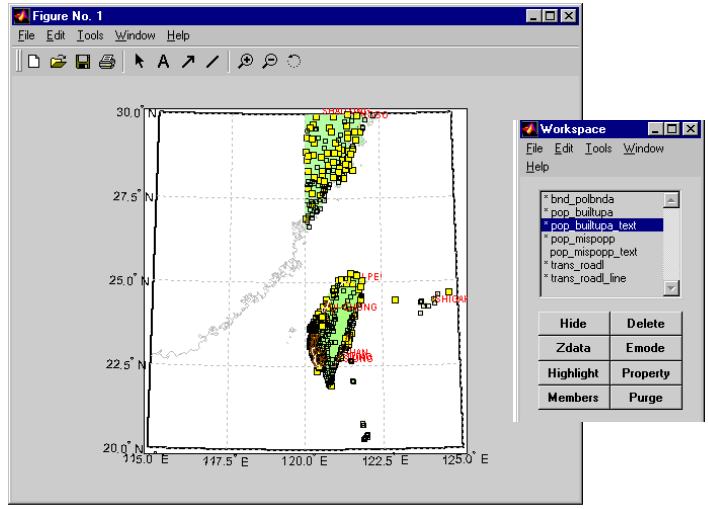
The `vmap0ui` applies a set of conventional map styles to extracted objects. Notice that the road lines have the appearance of roads on a map, line width indicating importance and double parallel lines for divided highways. Railroad lines are shown with cross ties, powerlines have dots along the path, and population points are large or small squares, as they are on the paper charts from which VMAP0 was derived. The `vmap0ui` also automatically creates text objects for features that lack them, like built-up areas.

This is really more data than is appropriate for a map of this size. Next we'll return the data to the MATLAB workspace and create a new map that is less cluttered. Click the **Save** button, and then click the **Workspace** button . You will be shown a list of geographic data structures to which the data is written. The names are abbreviated versions of the theme and coverage names.

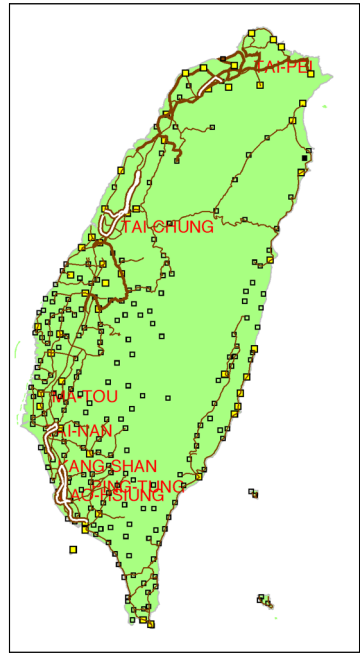


The geographic data structures can now be displayed on a map axes using the command line function `displaym` or the graphical user iterface `mlayers`. To interactively display data from the workspace, use `rootlayr` and then `mlayers`. If the data was saved in a MAT-file, you can use `mlayers` directly. Here most of the data has been displayed, and the built-up area text color changed to red. You may need to move the land areas down with the **Zdata** button to ensure that all of the lines and text are visible.

```
figure
worldmap('taiwan','lineonly')
set(handlem('allline'),'color',0.75*[ 1 1 1])
rootlayr; mlayers(ans)
```



Here is the result after zooming in on the island of Taiwan.



Importing VMAP0 Themes from the Command Line

The vmap0data function is used to access the map data at the command line. The volume of data on the VMAP0 is so large that you must limit your requests to a reasonable amount. First, select the geographical area of interest by giving the name of one of the libraries in the VMAP0. There is one CD for each library: 'NOAMER' (North America), 'EURNASIA' (Europe and Northern Asia), 'SOAMAFR' (South America and Africa), and 'SASAUS' (Southern Asia and Australia).

You may also need to specify the device name of the CD, just as you did for the dcwgaz function. The device name depends on your operating system and how it is configured. If more than one CD can be mounted at once, there will probably be a different device name for each CD. With the correct device name, you should see the following for the SASAUS CD-ROM and library:

```
cd(devicename)
ls
FGDC_DAT.TXT README1.TXT  VIEW         VMAPLV0

cd vmaplv0
ls
DHT      LAT      RFERENCE SASAUS
```

The geographic region of interest is further restricted by latitude and longitude limits, given in units of degrees.

You also need to specify which *theme*, or type of data, you want to extract. Themes are layers of data related to a kind of feature, like roads and drainage, and are given as strings. If you enter a code that is not recognized, a list of valid codes and their descriptions are displayed.

Finally, you need to indicate what type of graphic objects you want (patches, lines, points, or text). These are called *topology levels*.

You can see the list of available themes by entering an incorrect one, like '?'. The latitude and longitude limits correspond to the Cape Cod region:

```
Struc = vmap0data(devicename,'NOAMER',...
                  [41 44],[-72 -69],'?','line')
??? 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
```

If you are using a Windows PC, with the 'NOAMER' CD-ROM in the 'F:' drive, you can read the Political/Ocean line features for the same Cape Cod region by entering the following:

```
BNDline = vmap0data('F:','NOAMER',...
                    [41 44],[-72 -69],'bnd','line')
BNDline =
1x8 struct array with fields:
    type
    otherproperty
    altitude
    lat
    long
    tag
```

The result is a Mapping Toolbox geographic data structure. Look at the geographic limits by concatenating the data with `extractm`.

```
[lat,long] = extractm(BNDline);
[max(lat) max(long); min(lat) min(long)]
ans =
    45   -66
    40   -72
```

Notice that this region is larger than was requested. VMAP0 tiles the world into quadrangles of various sizes. When a tile overlaps the requested region, `vmap0data` extracts the data for the entire tile. Similarly, if you request a point location, the entire quadrangle is returned. All objects are cut at the edges of the tiles.

The kind of information in an element of the structure is described by the tag field. There are eight kinds of boundary lines in this region:

```
unique(strvcat(BNDline.tag),'rows')
ans =
Accurate; Definite; Coastline/Shoreline
Accurate; Definite; International; Administrative Boundary
Accurate; Definite; Primary/1st Order; Administrative Boundary
Approximate; Definite; International; Administrative Boundary
Depth Contour; 1000
Depth Contour; 200
Depth Contour; 2000
Depth Contour; 600
```

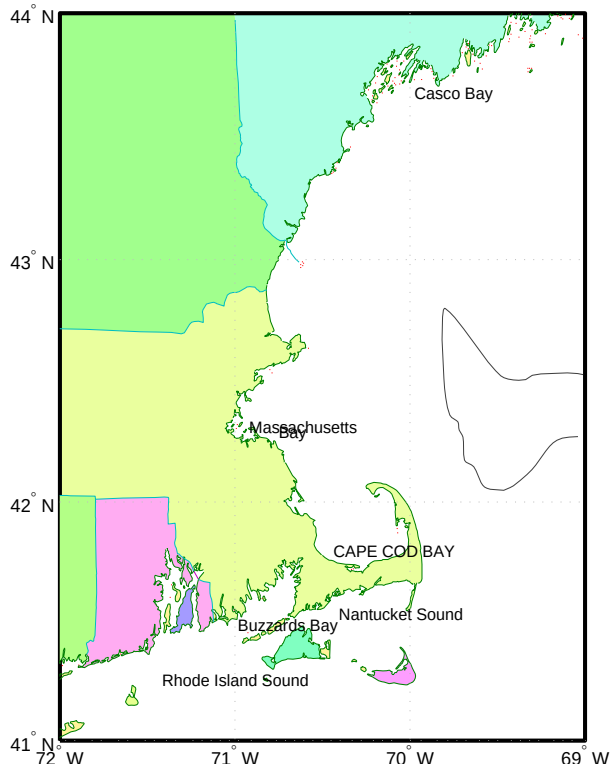
You can extract more than one topology level using a cell array of level names. Extracting patches and points is more time-consuming, so this will take longer than the previous example. The following example assumes that the 'NOAMER' CD-ROM is mounted on a UNIX computer as '/cdrom':

```
[BNDpatch,BNDpoint,BNDtext] = vmap0data('/cdrom','NOAMER',...
          [41 44],[-72 -69],'bnd',{'patch' 'point' 'text'});
BNDpatch(1)
ans =
             type: 'patch'
    otherproperty: {}
         altitude: []
              lat: [44x1  double]
             long: [44x1  double]
              tag: 'BAY OF FUNDY; Water (except Inland)'

BNDpatch(end).tag
ans =
North America; United States; VERMONT; Administrative Area
```

The following commands plot the political and ocean features on a Mercator projection, hiding the ocean patches. They also override the default colormap used for the land patches and remove the edge color to hide any tile joints:

```
worldmap([41 44], [-72 -69],'none')
setm(gca,'MapProjection','mercator'); tightmap

hBNDpatch = displaym(BNDpatch); hBNDline = displaym(BNDline);
```

```
hBNDpoint = displaym(BNDpoint); hBNDtext = displaym(BNDtext);
set(hBNDpatch,'EdgeColor','None')
hidem(handlem('NORTH ATLANTIC OCEAN; Water (except Inland)'));
colormap(polcmap)
zdatam('allpatch',-1)
```

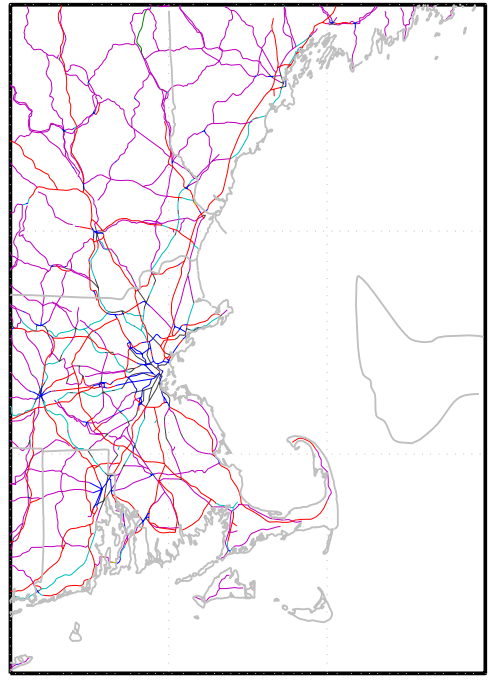


Notice how much more detailed this map is than even the best of the atlas data provided in the Mapping Toolbox. The highest resolution atlas data is in the `usahi` workspace, which is limited to the United States. VMAP0 covers the entire world at this level of detail. Unlike the earlier DCW, islands are not tagged with a state name.

The VMAP0 does not contain all topology levels in each theme. Roads, for example, have no patches. If data is requested for topology levels that are not

available, the function issues a warning and returns an empty matrix. Here is a display of the road and railroad line data for the Cape Cod map.

```
[TRpatch,TRline,TRpoint,TRtext] = vmap0data('cdrom','NOAMER',...
          [41 44],[-72 -69],trans',{'all'});
clma
hBNDline = displaym(BNDline);
set(hBNDline,'Color',[1 1 1]*0.75,'LineWidth',1)
hTRline = displaym(TRline);
```



You can use the `handlem` graphical user interface to select only the roads or railroads. This would let you change, for example, all of the railroad lines to a dashed style.

The network of roads around Boston gives away its location. VMAP0 also includes a populated places theme, containing the boundaries of built-up areas. Extract the 'pop' theme, reduce the displayed area to Boston and Cape Cod, and plot the populated places:

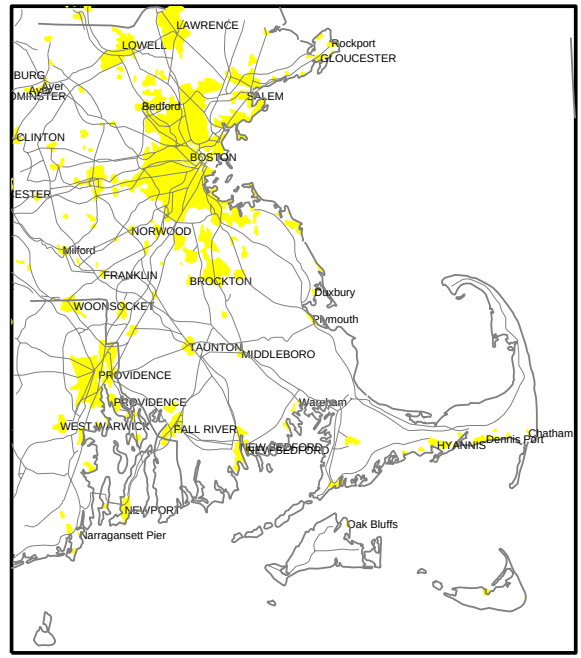
```
PPpatch = vmap0data('NOAMER',[41 44],[-72 -69],'pop','patch');

PPpatch = vmap0data('/cdrom','NOAMER',[41 44],[-72 -69],...
                    'pop','patch');

setm(gca,'MapLatLimit',[41.13 42.75],...
         'MapLonLimit',[-71.7 -69.8])
tightmap

hPPpatch = displaym(PPpatch);
set(hPPpatch,'FaceColor','y','Edgecolor','None')
zdatam(hPPpatch,-1)
```

The names of many of the cities are stored in the tag field of populated place patches and points. Unlike the DCW, VMAP0 does not provide text objects with city names, but they can be extracted from the tags.

Create a new geographic data structure of named cities using the geographic mean of the populated place outlines.

```
k = 0;
for i=1:length(PPpatch)
   str = PPpatch(i).tag;
   if isempty(strmatch('No entry present;',str))
      k=k+1;
      lat = PPpatch(i).lat;
      lon = PPpatch(i).long;
      [lat,lon] = meanm(lat(~isnan(lat)), lon(~isnan(lon)) );
      indx = findstr(';',str);
      PPtext(k).string = str(1:indx(1)-1);
      PPtext(k).type = 'text';
      PPtext(k).tag = 'Place Name';
      PPtext(k).otherproperty = [];
      PPtext(k).lat = lat;
      PPtext(k).long = lon;
      PPtext(k).altitude = [];
   end
end

hPPtext = displaym(PPtext); set(hPPtext,'FontSize',6)
set(handlem('allline'),'Color',[.5 .5 .5])
```

Here is the result:

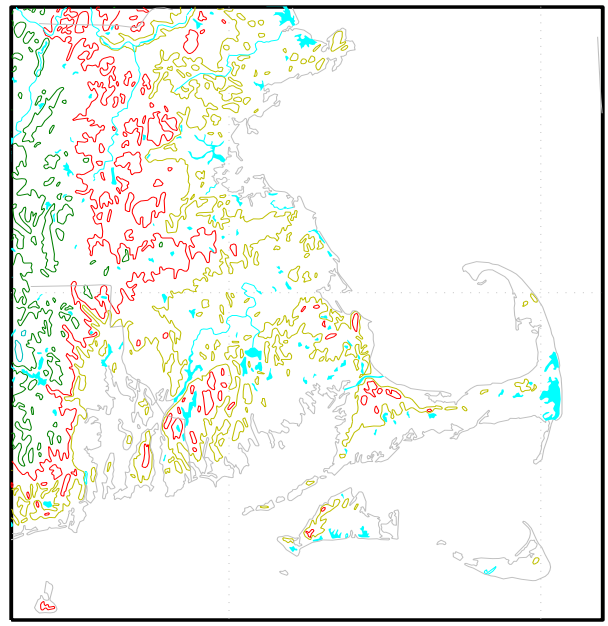


There is more data describing man-made features, including airports, utilities and cultural landmarks. The DCW also contains data for the natural world. The drainage and elevation contours are usually the most extensive data themes.

```
clma
[DNpatch,DNline,DNtext] = vmap0data(devicename,'NOAMER',...
     [41 44],[-72 -69],'hydro',{'patch','line','text'});
ELline = vmap0data(devicename,'NOAMER',...
     [41 44],[-72 -69],'elev','line');

hBNDline = displaym(BNDline);
hDNline = displaym(DNline);
hDNpatch = displaym(DNpatch);
hELline = displaym(ELline);
```

```
set(hBNDline,'Color',[1 1 1]*0.75)
set(hDNline,'Color','c');
set(hDNpatch,'FaceColor','c','EdgeColor','none')
```

Here is the displayed map:



Reading VMAP0 Files Directly

The high-level `vmap0data` function is the most convenient way to read data from the VMAP0 because it integrates information from many different files in different directories. Individual VMAP0 files are less useful because few of them contain complete information. Most people have little need to read the raw VMAP0 data files. Still, the Mapping Toolbox can read most VMAP0 files into MATLAB structures using the `vmap0read` function. The “External Data Reference” chapter of the online Mapping Toolbox reference documentation has more information on directly accessing individual VMAP0 files.

More information on the relationships between the files and the meaning of the data in the files can be found in the U. S. Government *Military Specifications and Standards*.

Global Self-Consistent Hierarchical High-Resolution Shoreline

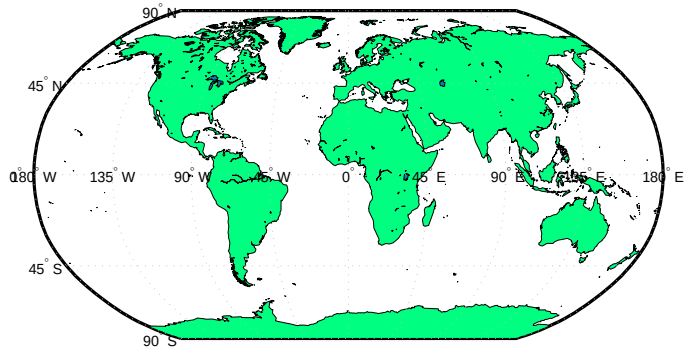
Another source of high-resolution coastline data is the Global Self-consistent Hierarchical High-resolution Shoreline (GSHHS). The GSHHS contains coastlines, major rivers, and lakes for the entire globe. The data requires 85 megabytes of storage at full resolution, but four successively lower resolution versions are also available. The GSHHS was developed by Paul Wessel of the University of Hawaii and Walter H. F. Smith of the NOAA Geosciences. The sources of data were the CIA World Data Bank II, designed for a scale of about 1:2,000,000, and the NOAA World Vector Shoreline, which is at a scale of 1:250,000.

The GSHHS data in the various resolutions is available over the Internet from `<ftp://ftp.ngdc.noaa.gov/MGG/shorelines>` and `<ftp://kiawe.soest.hawaii.edu/pub/wessel/gshhs/>`. These are binary files compressed using gzip. You should download the files in binary mode and decompress them. No byte swapping or line ending conversion is needed.

You can read any of the GSHHS data files with the `gshhs` function. GSHHS files have filenames 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).

Read the entire crude resolution version and display it.

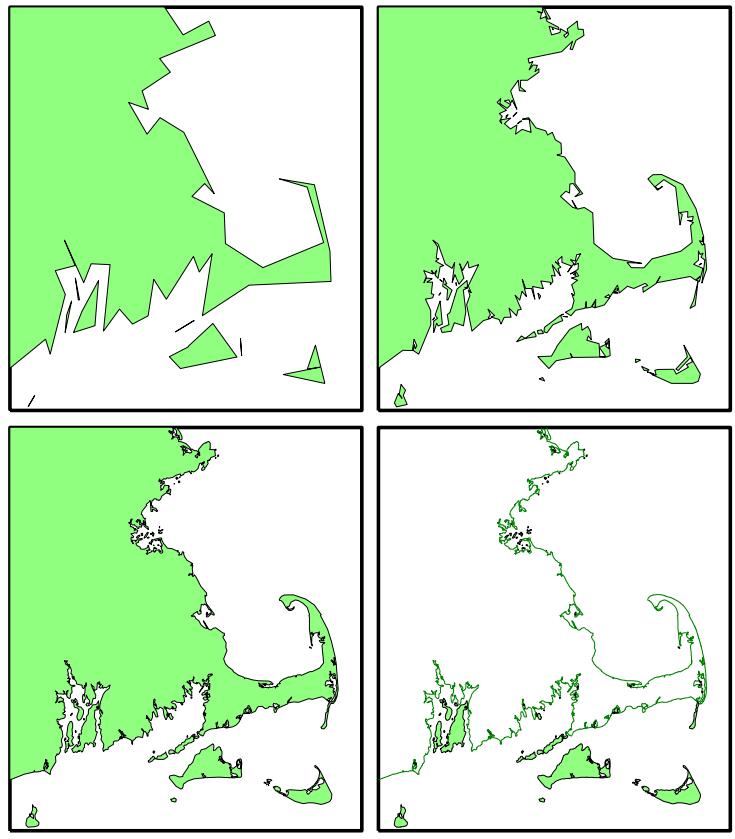
```
s = gshhs('gshhs_c.b');
worldmap('world','none')
displaym(s)
colormap(winter)
```



The higher resolution versions of the data are better suited for maps covering smaller regions. Here is a comparison of the other four data files. Before extracting the data, be sure to create an index file using the 'createindex' option. This option writes a file that allows gshhs to read much faster. You only need to do this once.

```
latlim = [41.13 42.75]; lonlim = [-71.7 -69.8];
figure

subplot('position',[.01 .51 .48 .48])
axesm('mercator','MapLatLimit',latlim,'MapLonLimit',lonlim,...
      'Frame','on')
gshhs('gshhs_l.b','createindex')
s = gshhs('gshhs_l.b',latlim,lonlim); displaym(s); tightmap

subplot('position',[.51 .51 .48 .48])
axesm('mercator','MapLatLimit',latlim,'MapLonLimit',lonlim,...
      'Frame','on')
gshhs('gshhs_i.b','createindex')
s = gshhs('gshhs_i.b',latlim,lonlim); displaym(s); tightmap

subplot('position',[.01 .01 .48 .48])
axesm('mercator','MapLatLimit',latlim,'MapLonLimit',lonlim,...
      'Frame','on')
gshhs('gshhs_h.b','createindex')
s = gshhs('gshhs_h.b',latlim,lonlim); displaym(s); tightmap

subplot('position',[.51 .01 .48 .48])
axesm('mercator','MapLatLimit',latlim,'MapLonLimit',lonlim,....
      'Frame','on')
gshhs('gshhs_f.b','createindex')
s = gshhs('gshhs_f.b',latlim,lonlim); displaym(s); tightmap
```

Here are the results:



The data is returned as geographic data structure of filled patches with the tags 'land', 'lake', 'island' for an island in a lake, or 'pond' for a pond on an island in a lake. The full resolution version displays the continent as a line rather than as a filled patch, because of the memory required to trim a polygon with more than a million points. The high resolution version is virtually indistinguishable in shape from the full version and requires only 20 megabytes to store.

USGS On-Line Coastline Extractor

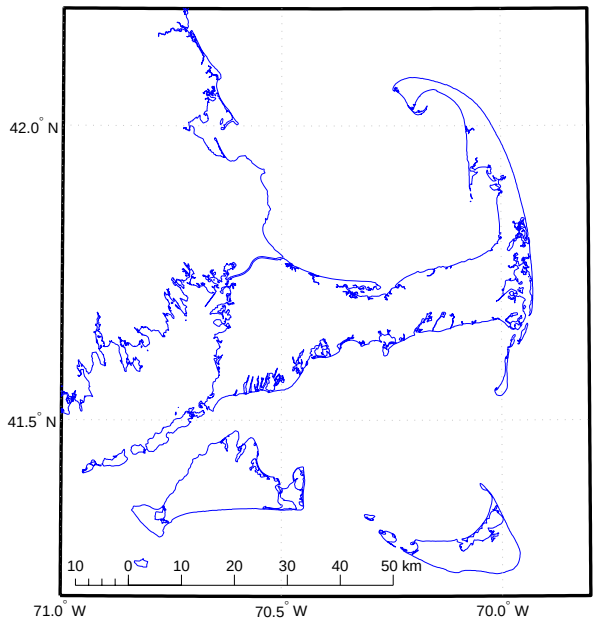
There is also an on-line source for a variety of coastline data. The U. S. Geological Survey has a Web page that allows you to extract coastlines, political boundaries, rivers, and lakes for a selected quadrangle, and download the data in a MATLAB compatible text file. You can extract data from the NOAA/NOS Medium Resolution Coastline (U. S. coverage only, designed for a scale of 1:70,000), the World Vector Shoreline (designed for 1:250,000), the CIA World Data Bank II (designed for 1:2,000,000), and the World Coast Line (designed for 1:5,000,000). The location of the Web page is

```
<http://crusty.er.usgs.gov/coast/getcoast.html>
```

Here is the NOAA/NOS Medium Resolution Coastline for Cape Cod. The data was saved in the file, 15162.dat.

```
load 15162.dat
whos
  Name         Size          Bytes  Class

  X15162   21087x2          337392  double array

usamap([41.2 42.2],[-71 -69.8],'none')
plotm(X15162(:,2),X15162(:,1)); scaleruler
```

Here is the displayed map:



U.S. Vector Data

One of the best sources of public vector data for the United States is the U.S. Census Bureau. This organization has developed a digital database and system for automated mapping in support of the United States decennial census. The Topographically Integrated Geographic Encoding and Referencing (TIGER) system is used within the Bureau to carry out and analyze the census. TIGER contains hydrographic features, roads, railroads, and other transportation features scanned from 1:100,000 U.S. Geological Survey (USGS) topographic maps. The database also contains census tracts and political boundaries for states, counties, and indian reservations; it does not include topography. The Census Bureau periodically publishes extracts from the TIGER database for public use in the form of TIGER/Line and TIGER thinned boundary files. The Mapping Toolbox provides interfaces to both formats.

TIGER/Line

TIGER/Line files are extracts from the U.S. Census Bureau's TIGER map database. Each set of files covers one municipal district and contains transportation features such as roads and railroads, cultural features, hydrographic features, and political boundaries. The data is distributed on CD-ROMs by the Census Bureau. TIGER/Line data is not copyrighted, but the Bureau does charge for the production of the CDs. Ordering and other information can be found from the U.S. Census Bureau's Web site at `<http://www.census.gov/>`, along with sample TIGER/Line files. CD-ROMs may also be accessible over the Internet through various organizations.

You can import TIGER/Line data into MATLAB using the `tgrline` function. Files can be several thousand lines long, and reading them may require considerable time and memory. Once the data has been read and processed, you may want to save the results into a MATLAB workspace MAT-file for future use.

The following example reads the Washington, D.C., data from the 1994 TIGER/Line edition. These files are available over the Internet from The MathWorks at `<ftp://ftp.mathworks.com/>` and are delivered as text files in a compressed PC Zip archive. When extracting, make sure the text files are formatted for your computer platform. Generally, the names of the TIGER/Line datasets begin with the string 'tgr' and contain the 5-digit Federal Information Processing Standards (FIPS) code for the state and county.

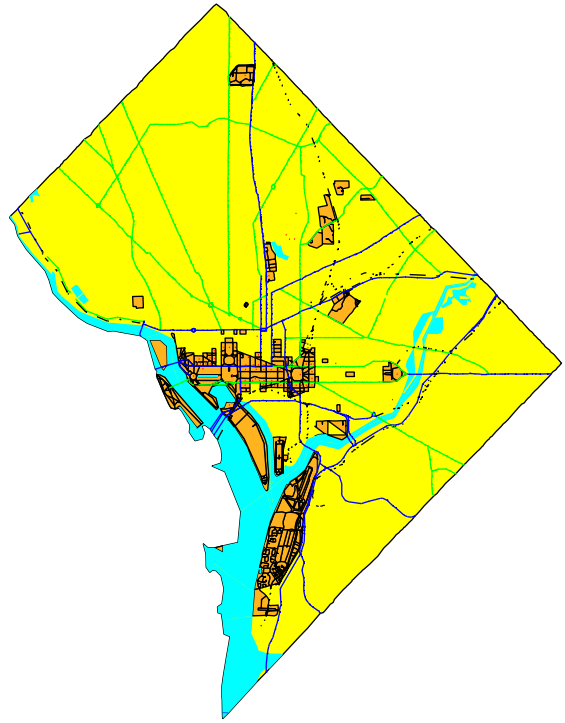
The dataset for the District of Columbia is 'TGR11001':

```
[CL,PR,SR,RR,H,AL,PL] = tgrline('TGR11001');
whos
   Name       Size       Bytes     Class

   AL         45x1       112952    struct array
   CL         1x1         7444     struct array
   H           16x1      25030     struct array
   PL         64x1       59730     struct array
   PR         1x32        53534    struct array
   RR          1x8       16632     struct array
   SR         1x40       76870     struct array
```

The `tgrline` function returns Mapping Toolbox geographic data structures for the county line (CL), primary roads (PR), secondary roads (SR), railroads (RR), hydrography (H), area landmarks (AL), and point landmarks (PL). This is a subset of the total available data in the TIGER/Line files, which contain more detailed local roads and other geographic reference data, such as Zip codes and census tract numbers, that are not handled by the function.

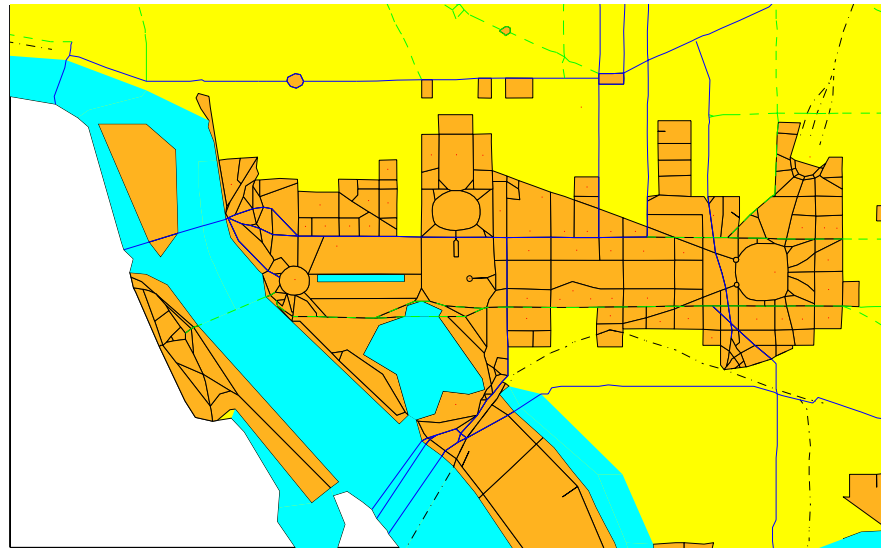
Since the data is now formatted into geographic data structures, you can display the objects using `displaym`. A Mercator projection is appropriate for small-scale, regional maps:

```
axesm mercator
displaym(CL)
hh = displaym(H); set(hh,'EdgeColor','None')
displaym(AL); displaym(PL)
displaym(PR); displaym(SR); displaym(RR)
```



You can also produce the same map with graphical user interfaces. Using the maptool application, invoke the `mlayers` tool by selecting **Session⇒Layers⇒Workspace** from the menu bar, or invoke mlayers from the command line by typing `rootlayr; mlayers(ans)`.

The amount of detail in the data becomes apparent when you zoom into the federal district of Washington, D.C. While the map has much detail, the street patterns contain displaced points, and the number of points in the paths of the streets is not enough to give smooth representations of slowly curving streets.



If you hold down the mouse button on an object, you will see its name appear in the lower left part of the figure window. All objects have been tagged with a name. Here are the names of places in the hydrography layer:

```
unique(strvcat(H.tag),'rows')

ans =
Anacostia River
Channel
Dalecarlia Reservoir
Georgetown Reservoir
Kenilworth Aquatic Gardens
Kingman Lake
Lagoon
Mc Millan Reservoir
Potomac River
Reflecting Pool
Tidal Basin
Washington Channel
```

The other layers have tags with street names, interstate highway numbers, railroad names, and landmarks names. The names can be useful in finding the

location of a site or in the selective display of data. What is the latitude and longitude of the Washington Monument? Mark the spot:

```
[lat,long,indx] = extractm(PL,'washington mon');
PL(indx)
ans =
               lat: 38.8891
              long: -77.0354
               tag: 'Washington Monument'
              type: 'line'
          altitude: 0.4000
     otherproperty: {1x3 cell}

h = displaym(PL(indx)); set(h,'Marker','x','Color','r')
```

You can also use the `mobjects` tool to browse the names of objects that have already been plotted.

TIGER Thinned Boundary

The U.S. Census Bureau has published lower resolution, thinned extracts of the TIGER database in two data file formats: ARC/INFO and MapInfo Interchange Format (MIF). ARC/INFO and MapInfo are geographic information systems (GIS) developed by the Environmental Systems Research Institute, Inc. (ESRI) and MapInfo Corporation, respectively. Data files for both formats are provided by the U.S. Census Bureau via the Internet <http://www.census.gov/>, along with further descriptions and documentation.

TIGER ArcInfo Format

The following boundary types are available in the ArcInfo format: state, county, minor civil division, census tract/block numbering area, American Indian reservation/Alaska native village statistical area, Alaska native regional corporation, urbanized area, metropolitan area, and congressional district.

The ArcInfo data files have names ending in 'p' and 'pa'; the first represents polygon coordinate data, and the latter polygon attributes. You will also need the '_name.dat' file, or *names* file, containing the FIPS codes and corresponding county names.

Plot the counties in the state of Alaska as an example. You will need the following three files: 'co_name.dat,' 'co_02_p.dat,' and 'co_02_pa.dat.' For these files, 'co' indicates county boundaries, and '02' is the code for Alaska. The code '99' is used for files within the continental United States, and '15' corresponds to Hawaii. The files are compressed in a PC Zip format. When extracting, make sure the text files are formatted for your computer platform.

Read all U.S. county names and FIPS code from the names file. There are more than 3000 entries, so reading the file may take a while:

```
namestruc = fipsname('co_name.dat')
namestruc =
1x3248 struct array with fields:
   name
   id
```

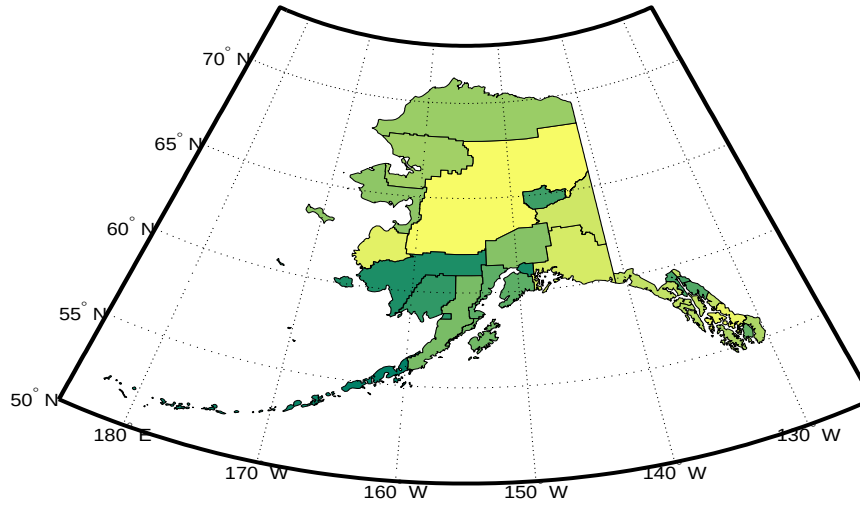
You can query the FIPS code of a particular city or county in Alaska, Anchorage, for example, by the following commands:

```
indx = strmatch('Anchorage',{namestruc.name})
indx =
   70
code = namestruc(indx).id
code =
   2020
```

Use the `tigerp` function to read and process Alaska's thinned county boundaries data into Mapping Toolbox geographic data structures.

Plot the data in a Lambert Conformal Conic projection:

```
[COpatch,COtext] = tigerp(namestruc,'co_02_p.dat');
axesm('MapProjection','lambert',...
   'MapLatLimit',[50 73],'MapLonLimit',[175 235] ,...
   'MLabelLocation',10,'MLineLocation',10,...
   'PLabelLocation',5,'PLineLocation',5,...
   'MLabelParallel','south','MapParallels',[]);
framem; gridm; mlabel; plabel
displaym(COpatch)
colormap(summer)
```



There is one entry in the `COpatch` structure for each county, with the county name in the tag field. The `COtext` structure contains the text labels for the counties.

You can also append new data to existing structures or extract only the counties you want. The following example reads the county boundaries for Nome and then adds the boundaries for the Aleutians East and Aleutians West to the output structure. You can look in the `namestruc` structure to find the codes you need.

```
COpatch = tigerp(namestruc,'co_02_p.dat',[],[],2180);
COpatch = tigerp(namestruc,'co_02_p.dat',COpatch,[],...
   [2013 2016])
COpatch =
1x3 struct array with fields:
   lat
   long
   type
   otherproperty
   altitude
   tag
```

TIGER MapInfo Interchange Format

The U.S. Census Bureau provides thinned boundary files in the MapInfo Interchange Format. Both U.S. state and county boundaries are available in this format.

The MapInfo data files have the 'mif' and 'mid' extensions. You will also need the '_name.dat' file, or names file, containing the Federal Information Processing Standard (FIPS) code numbers and corresponding county names.

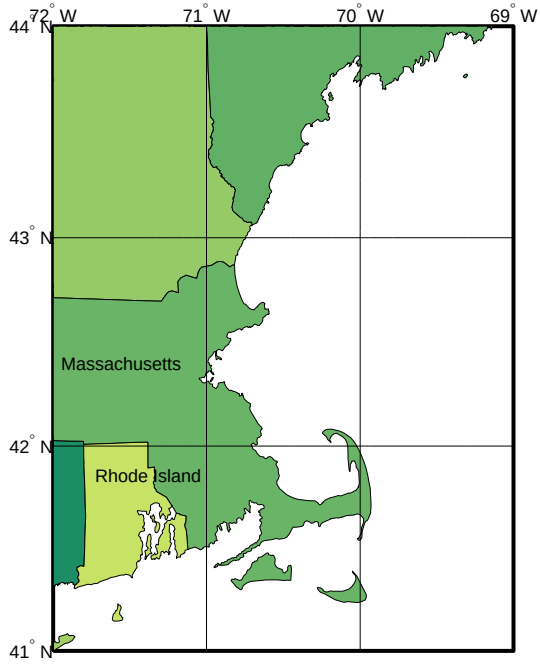
The following example reads the state boundaries of Alaska and Hawaii. Five files are needed: 'ST02.MID,' 'ST02.MIF,' 'ST15.MID,' 'ST15.MIF,' and 'st_name.dat.' You must first read the names file for the names and FIPS codes for the U.S. states and territories:

```
namestruc = fipsname('st_name.dat')
namestruc =
1x57 struct array with fields:
    name
    id
```

Read the file containing Hawaii's county boundaries into a geographic data structure, and append it with Alaska's state boundaries:

```
STpatch = tigermif(namestruc,'ST15.MIF');
STpatch = tigermif(namestruc,'ST02.MIF',STpatch)
STpatch =
1x2 struct array with fields:
    lat
    long
    type
    otherproperty
    tag
    altitude
```

Finally, to compare the resolution and quality of this data to other sources, look at the region around Cape Cod. This map may look familiar to you; it is the same data in the `usahi` workspace, plotted in a Mercator projection. The `usahi` data was derived from the Census Bureau's TIGER Thinned Boundary files.


72° W
71° W
70° W
69° W
44° N
43° N
42° N
41° N
Massachusetts
Rhode Island

Global Matrix Data

Some types of geographic data are provided as matrices of values on a geographic grid. The Mapping Toolbox provides interfaces to gridded elevation and bathymetry at resolutions ranging from 10 kilometers to 30 meters, geoid heights, and vegetation index and classification data. There are also functions to help you write your own interface functions.

ETOPO5 and TerrainBase

ETOPO5 and TerrainBase are digital elevation models with worldwide coverage at a resolution of 5 minutes (10 kilometers or better). They are both compilations 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 in Australia, and the Department of Industrial and Scientific Research in New Zealand. The ETOPO5 dataset was assembled by Margo Edwards at Washington University, in St. Louis, Missouri.

An overview of this dataset can be found at several Internet locations, including the U.S. Geological Survey Web site located at <http://edcwww.cr.usgs.gov/> and the U.S. National Geophysical Data Center home page at <http://www.ngdc.noaa.gov/>.

The ETOPO5 data is read using the `etopo5` external interface function for the version of the database contained in the two text files, 'etopo5.southern.bat' and 'etopo5.northern.bat.' These files occupy about 60 megabytes of total disk space when uncompressed and are provided via FTP by The MathWorks, Inc. at <ftp://ftp.mathworks.com/>. You can also read the much more compact binary file version 'new_etopo5.bil' available from <http://www/geo.cornell.edu/geology/seber/datasets/new_etopo5.bil>.

ETOPO5 has been superseded by TerrainBase, which is similar in coverage to ETOPO5, but corrects many of its flaws, including discontinuities at joins between datasets, systematic shifts in location, and lack of data in some regions. TerrainBase is a compilation of the best available, public domain data from almost 20 different sources.

The TerrainBase dataset was created by the National Geophysical Data Center and World Data Center-A for Solid Earth Geophysics in Boulder, Colorado. The model is currently under development and will be updated as new data sources become available. Further information on the TerrainBase DEM can be found

at the U.S. National Geophysical Data Center home page at <http://www.ngdc.noaa.gov/>.

TerrainBase data is read using the `tbase` function. The data files are available on CD-ROM and via Internet. NOAA/NGDC provides ordering information and downloadable files at their Web site. The files require about 18 megabytes of disk space when uncompressed.

You should use the TerrainBase dataset unless you have a good reason to want ETOPO5. TerrainBase corrects some errors, takes up less space, and can be read faster than ETOPO5. The following examples will read TerrainBase data, but you can substitute ETOPO5 data by simply using the `etopo5` function instead of `tbase`.

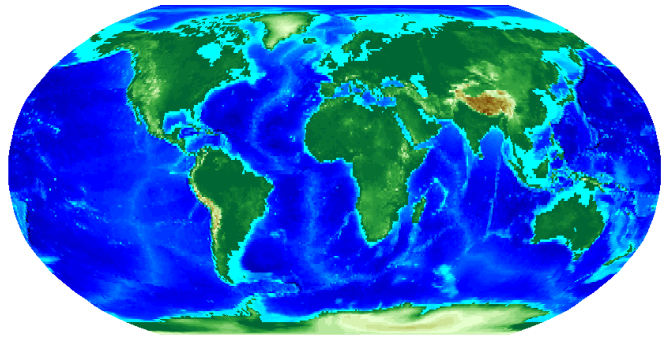
A 10 kilometer resolution DEM has so much data that it is usually impractical to read the entire file into memory. The full matrix has 2160 by 4320 elements. Both the `etopo5` and `tbase` functions allow you to reduce the resolution by discarding data points.

Read every twelfth point in the global data base.

```
[map,maplegend] = tbase(12);
whos
   Name       Size       Bytes     Class

   map        180x360     518400   double array
   maplegend  1x3         24       double array
```

Plot the map on a Robinson projection:

```
axesm robinson
meshm(map,maplegend); demcmap(map);
```



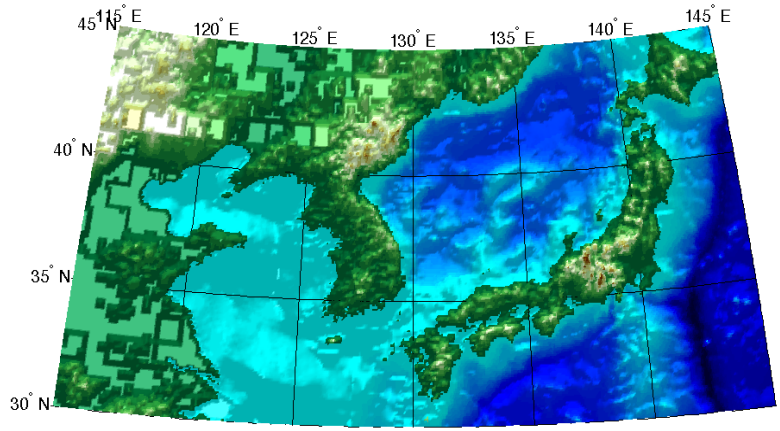
This is a regular matrix map at the same resolution as the `topo` map. It differs from `topo` in that the values represent nearest neighbor samples, rather than averages over the matrix cell, resulting in a rougher appearance.

Elevations and depths are given in meters above or below mean sea level. 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 that 200 meters contains little detail because of the way depth contours were converted to gridded depths.

The areas with 1 degree horizontal data can be clearly seen when you display data for a small region at the full resolution of the database.

Display the Korean peninsula and Japan:

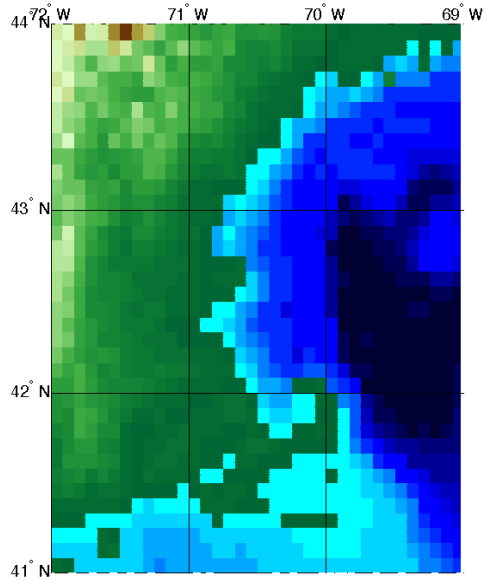
```
[map,maplegend] = tbase(1,[30 45],[115 145]);
axesm('MapProjection','polycon',...
   'MapLatLimit',[30 45],'MapLonLimit',[115 145],...
   'MLabelLocation',5,'MLineLocation',5,...
   'PLabelLocation',5,'PLineLocation',5,...
   'GLineStyle','-','GAltitude',inf) ;
gridm; mlabel; plabel
meshm(map,maplegend,size(map),map); demcmap(map);
lightm(37.5,130); material([.7 .7 1.5]); lighting gouraud
```



Note the discontinuities in some areas, particularly the regions in China.

You can get a sense for the resolution of this dataset by comparing a TerrainBase map of the Cape Cod region in the Northeastern United States with maps of the same region using other datasets. TerrainBase and ETOPO5 have a horizontal resolution of 12 cells per degree, so the Cape Cod map consists of 36 by 36 cells.

```
[map,maplegend] = tbase(1,[41 44],[-72 -69]);
whos
   Name       Size       Bytes     Class

   map        36x36      10368     double array
   maplegend  1x3        24        double array

axesm('Mapprojection','mercator',...
    'MapLatLimit',[41 44],'MapLonLimit',[-72 -69],...
    'MLabelLocation',1,'MLineLocation',1,...
    'PLabelLocation',1,'PLineLocation',1,...
    'GLineStyle','-','GAltitude',inf) ;
gridm; mlabel; plabel
meshm(map,maplegend); demcmap(map)
```



You can also read this data with the `demdataui` graphical user interface described later in this chapter.

Global Bathymetry Predicted from Satellite Altimetry

High resolution bathymetric data has been lacking for much of the world because few ships made sounding measurements away from heavily used sea lanes. Recently declassified satellite altimetry revealed that the topography of the ocean floor is mirrored in undulations of the ocean surface. Sandwell and Smith have developed a bathymetric model that combines ship soundings and satellite altimetry to infer the topography of the ocean floor. Land data is taken from the GTOPO30 elevation data, described below. The global topography is stored in a Mercator projection at a nominal resolution of 2 arc-seconds (or about 4 kilometers), covering the globe to 72 degrees north and south.

The data is available over the Internet via anonymous FTP from

```
<ftp://topex.ucsd.edu/pub/global_topo_2min/topo_6.2.img>
```

This function reads the binary files as they are, i.e., you should not use byte swapping software on these files. Some documentation is also available over the World Wide Web at

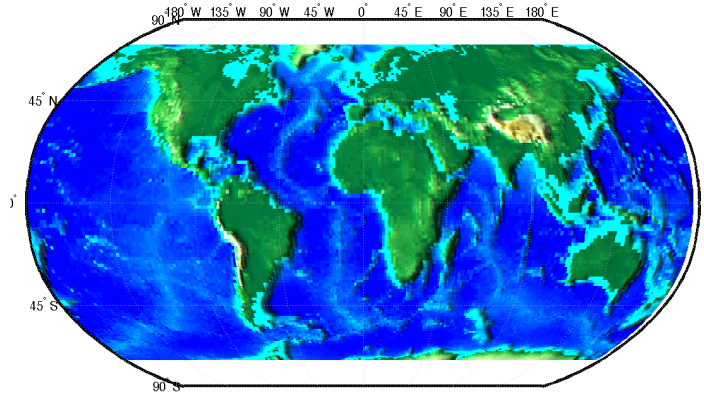
<http://topex.ucsd.edu/marine_topo/mar_topo.html>

and

<http://www.ngdc.noaa.gov/mgg/announcements/announce_predict.html>

The satbath function reads the data into a general matrix map that can be displayed in any projection. Here is the entire file reduced by a factor of 50. You can see the latitude limits in the data.

```
[latgrat,longrat,map] = satbath;

worldmap('world','none'); setm(gca,'MLabelParallel','n')
surfm(latgrat,longrat,map,map)
demcmap(map); daspectm('m',30)
camlight(-80,0); lighting phong; material([.3 5 0])
```

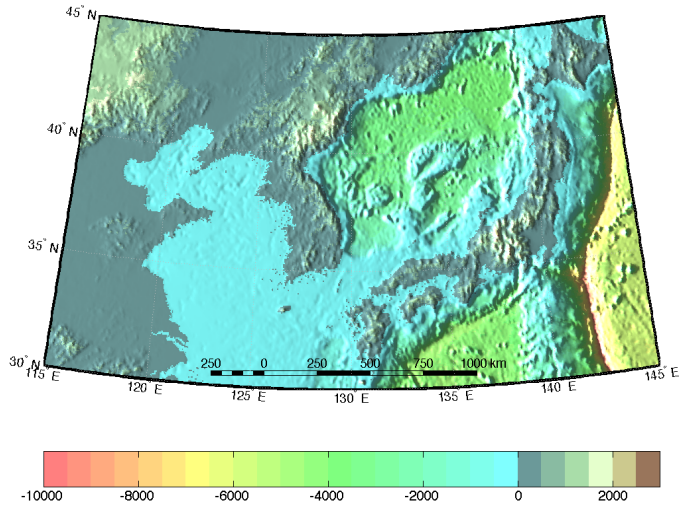


The richness of the data is evident in a map at close to the full resolution. You can bring out the small features of the data with a lightened hsv colormap for ocean depths, and a lighter version of the standard land colormap. This colormap is similar to one used in posters of this data by Sandwell and Smith.

```
latlim = [30 45]; lonlim = [115 145];
[latgrat,longrat,map] = satbath(2,latlim,lonlim);

worldmap(latlim,lonlim,'none')
```

```
surfm(latgrat,longrat,map,map)

landcmap = hsv2rgb([0.5 0.3 0.6; 0.25 0.2 1; 0.07 0.4 0.6]);
seacmap = [  1   0.5   0.5
             1   1     0.5
           0.5   1     0.5
           0.5   1     1  ];

demcmap('inc',map,500,seacmap,landcmap); daspectm('m',5)
material([0.4 3 0]); camlight(-80,0); lighting phong
scaleruler('MajorTick',0:250:1000,'MinorTick',0:50:250,...
           'RulerStyle','patches')
colorbar('horiz')
```

Here is the result:

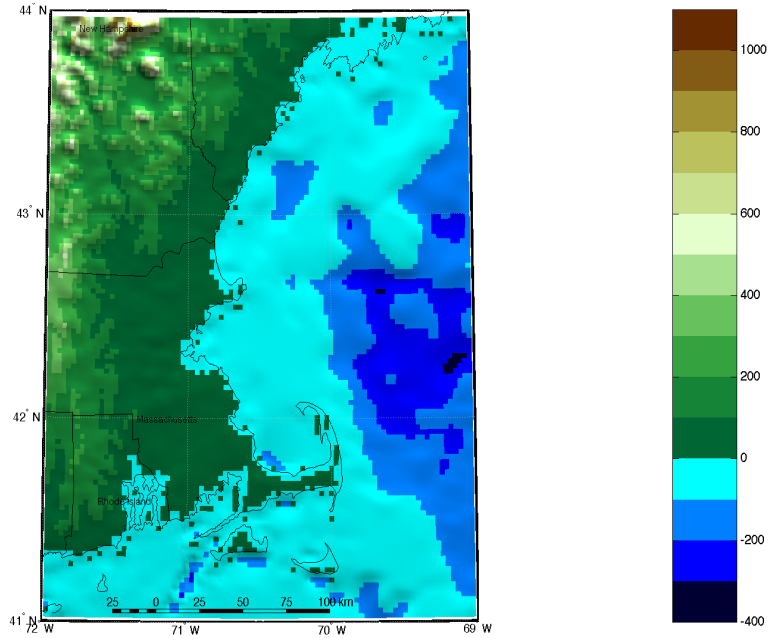


The data at the limit of resolution can be seen in this map of Cape Cod. The position of the land-sea interface is off in some places. This may be because the dataset uses odd or even numbers to indicate the source of the data.

```
latlim = [41 44]; lonlim = [-72 -69];
[latgrat,longrat,map] = satbath(1,latlim,lonlim);

usamap(latlim,lonlim,'line')
```

```
zdatam('alltext',1100); zdatam('allline',1100)
surfm(latgrat,longrat,map,map)
demcmap('inc',map,100); daspectm('m',5)
material([0.4 3 0]); camlight(0,80); lighting phong
scaleruler('MajorTick',0:25:100,'MinorTick',0:5:25,...
           'RulerStyle','patches')
colorbar
```

Here is the displayed map:



You can also read this data with the `demdataui` graphical user interface described later in this chapter.

GTOPO30 and GLOBE

GTOPO30 and GLOBE are other sources of gridded land elevation data at a relatively high resolution. The datasets provide global coverage of land elevations at a horizontal resolution of 30 arc-second (about 1 kilometer or better). These datasets do not include bathymetry.

GTOPO30 was derived primarily from DTED0 and the Digital Chart of the World (DCW). GTOPO30 was developed by the Earth Resources Observation Systems (EROS) Data Center. Where no existing grid was available, elevations were computed using the Australian National University's ANUDEM computer program. This program combines elevation and drainage lines and points to interpolate elevations at the grid locations.

GLOBE, the Global Land One-km Base Elevation data, was compiled by the National Geophysical Data Center from more than 10 different sources of gridded elevation data. GLOBE may be considered a higher resolution sucessor to TerrainBase. The Mapping Toolbox reads data from GLOBE version 1.0.

GTOPO30 is accessible over the Internet and is available on CD-ROM. Ordering information, as well as the actual data files, can be found on-line. 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 are 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>. The dataset consists of a total of 21,600 rows and 43,200 columns broken into 33 tiles, each generally covering 40-by-50 degrees. Each tile requires about 15 megabytes of storage when compressed, 80 megabytes uncompressed, and 160 megabytes to extract.

GLOBE is also available over the Internet and on CD-ROM. The data and some documentation are available over the World Wide Web from <http://www.ngdc.noaa.gov/seg/topo/globe.shtml>. The data is also available by anonymous ftp from <ftp://ftp.ngdc.noaa.gov/GLOBE_DEM/data/elev/>. The Web site also sells copies of the data on CD-ROM. The dataset is broken into 16 tiles, which take as much as 60 MB of storage. Uncompressed tiles take between 100 and 130 MB.

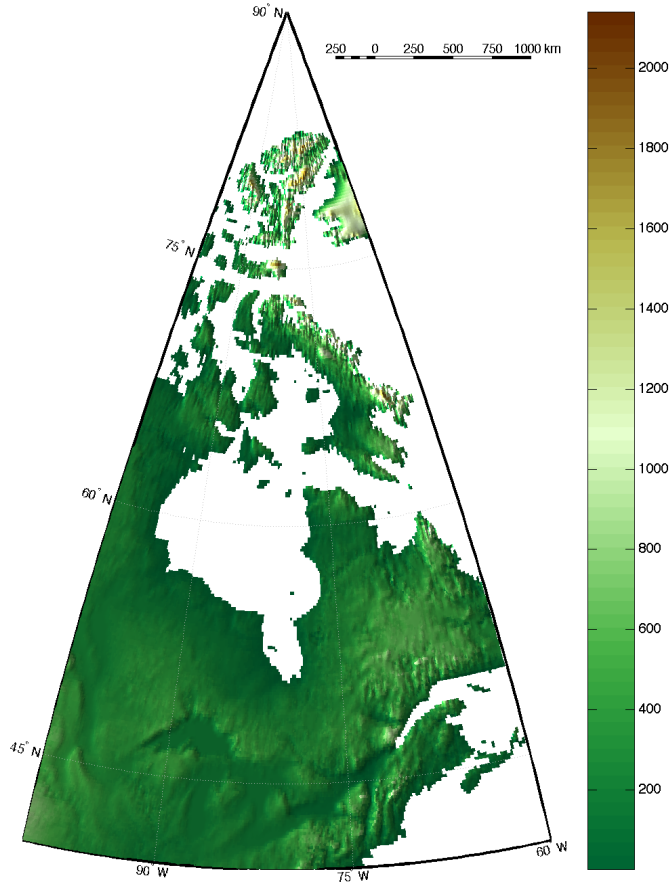
The Mapping Toolbox interface to GTOPO30 files is the `gtopo30` function. It reads the elevations into regular matrix maps, optionally reducing and trimming data in the process. The following examples show the `'W100N90'` tile, which covers the northeastern United States and eastern Canada. The files are

available from the <http://edcwww.cr.usgs.gov/landdaac/gtopo30/gtopo30.html> Web page.

Read the entire elevation file, taking every twentieth point. Display the data as a lighted surface in a map axes prepared by `worldmap`. `worldmap` selects a polyconic projection for these map limits. Add a graphic scale and colorbar.

```
[map,maplegend] = gtopo30('W100N90',20);
[latlim,lonlim] = limitm(map,maplegend);

 worldmap(latlim,lonlim,'none')
meshm(map,maplegend,size(map),map);demcmap(map)
demcmap(map); daspectm('m',15)
camlight(-80,0); lighting phong; material([.5 5 0])
scaleruler('MajorTick',0:250:1000,'MinorTick',0:50:250,...
           'RulerStyle','patches')
colorbar
```

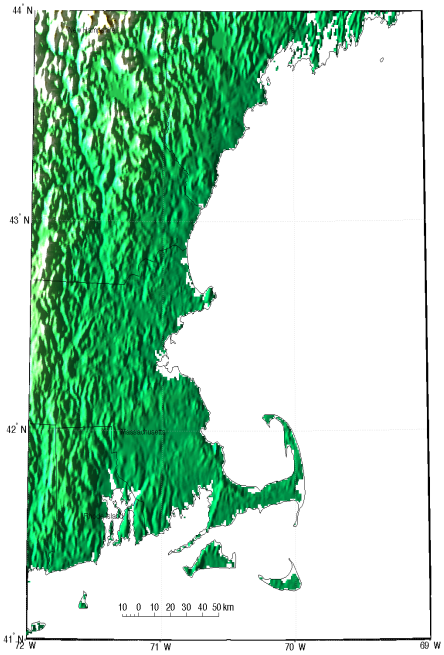
Here is the result:



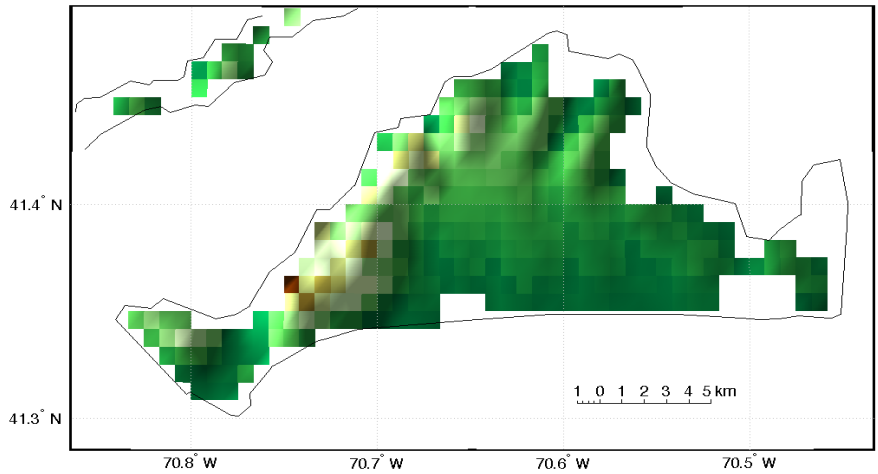
You can read a subset of the data by providing latitude and longitude limits. Here is the part of the dataset covering Cape Cod at the full resolution. The 30 arc-second GTOPO30 covers this region with a 360-by-360 matrix, which is 10 times finer than the 5 arc-minute (300 arc-second) ETOPO5 and TerrainBase data. This is also the same resolution as the data in the MATLAB cape workspace.

```
latlim = [41 44]; lonlim = [-72 -69];
[map,maplegend] = gtopo30('W100N90',1,latlim,lonlim);
```

```
figure
usamap(latlim,lonlim,'line')
zdatam('alltext',max(map(:)))
zdatam('allline',max(map(:)))
meshm(map,maplegend,size(map),map);demcmap(map)

demcmap(map); daspectm('m',15)
camlight(-80,0); lighting phong; material([.5 5 0])
scaleruler; colorbar
```



To see the ultimate resolution of the GTOPO30 data, extract only the area around Martha's Vineyard. Now you can see the individual pixels, and how the ocean areas are transparent. The ocean is coded as *no data*, and filled with NaNs.



You can determine which files contain data for a region using the `gtopo30s` command. Based on the latitude and longitude limits, `gtopo30s` determines which files you will need.

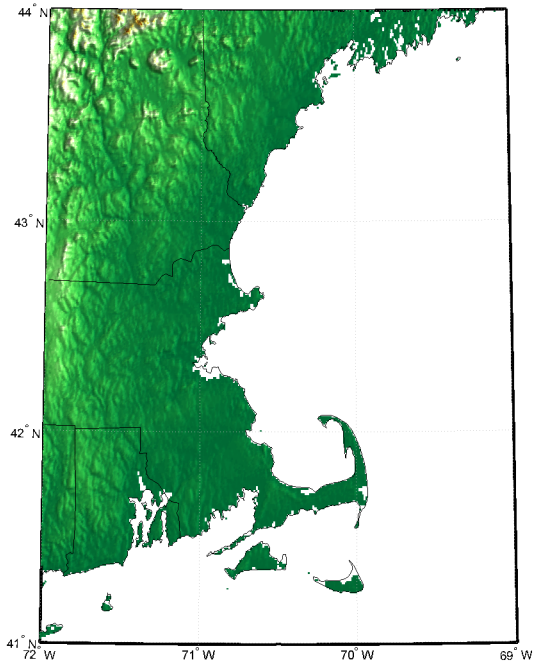
```
gtopo30s([41 44],[-72 -69])

ans =

    'w100n90'
```

If your region requires more than one file, give `gtopo30` the name of the directory containing the data. It will then read from each of the files and concatenate the data into a single matrix.

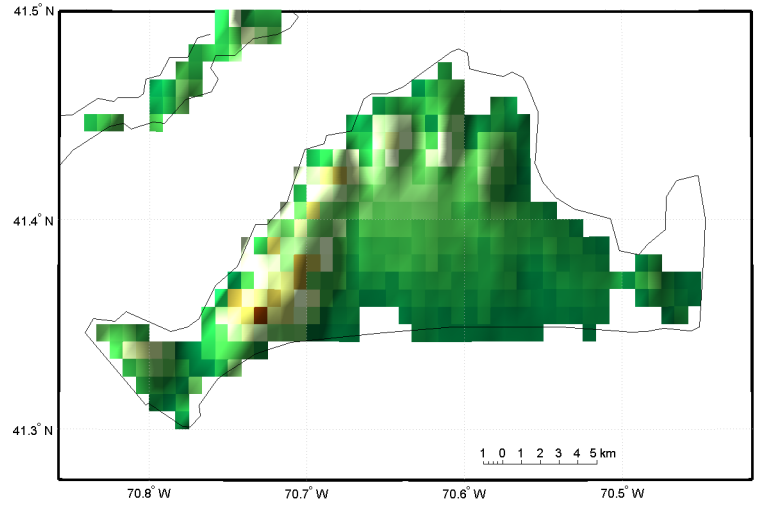
The GLOBE database is very similar to GTOPO30. It is also a tiled global land elevation database at 1 kilometer resolution. To read the GLOBE data, use the `globedem` command. This command should be distinguished from the three-dimensional `globe` projection. The `globedem` command works just like `gtopo30`, including the ability to concatenate data across tile boundaries. There is also a corresponding `globedems` command that lists the files required to cover an area.

It may be difficult to distinguish the GLOBE data from GTOPO30 over large areas. Here is the corresponding GLOBE data for Cape Cod.

```
 globedems([41 44],[-72 -69])

ans =

    'f10g'

[map,maplegend] = globedem('f10g',1,[41 44],[-72 -69]);
worldmap(map,maplegend,'ldem3d')
h = displaym(usahi('stateline'));
set(h,'color','k'); zdatam('allline',max(map(:)))
```



Some differences can be seen at higher resolutions. Here is the GLOBE data for Martha's Vineyard. The shape of the coastline differs because of different clipping at the coastlines. Also, the highest elevation is slightly shifted, with GLOBE's version in a more likely location.



You can read both the GTOPO30 and GLOBE data with the `demdataui` graphical user interface described later in this chapter.

Digital Terrain Elevation Data Model

The Digital Terrain Elevation Data (DTED) Model is a series of gridded elevation models with global coverage at resolutions of 1 kilometer or finer. It is a product of the U. S. National Imagery and Mapping Agency (NIMA), formerly the Defense Mapping Agency (DMA). The data is provided as 1-by-1 degree tiles or elevations on geographic grids with product-dependent grid spacing.

The lowest resolution data is the DTED Level 0, with a grid spacing of 30 arc-seconds, or about 1 kilometer. This version of the data has been made available over the Internet at <`http://164.214.2.59/geospatial/products/DTED/dted.html`>. The intended coverage is global, but there are currently large gaps in coverage for South America and Africa. DTED Level 1 has a resolution of 3 arc-seconds, or about 100 meters, and was the primary source for the USGS 1:250,000 (1 degree) DEMs. DTED Level 2 and above are at even higher resolutions. DTED Level 1 and above are available to the U. S. Department of Defense and its contractors from NIMA.

The DTED files are binary. No line ending conversion or byte swapping is required. The files have filenames with the extension `dtN`, where `N` is the level of the DTED product.

Which files are needed to cover the Cape Cod area?

```
dteds([41.1 43.9],[-71.9 -69.1])

ans =

    '\DTED\W072\N41.dt0'
    '\DTED\W071\N41.dt0'
    '\DTED\W070\N41.dt0'
    '\DTED\W072\N42.dt0'
    '\DTED\W071\N42.dt0'
    '\DTED\W070\N42.dt0'
    '\DTED\W072\N43.dt0'
    '\DTED\W071\N43.dt0'
    '\DTED\W070\N43.dt0'
```

When your region extends across more that one tile, you can have the `dted` command concatenate the tiles into a single matrix. Just tell the `dted` command the location of the `dted` directory.

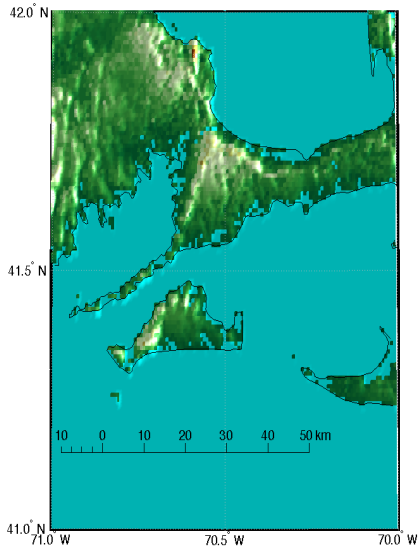
How many files are needed just around Martha's Vineyard?

```
dteds([41.1 41.9],[-70.9 -70.1])

ans =

    '\DTED\W071\N41.dt0'
```

Retrieve the DTED0 data for Cape Cod from the Web page, and read the 1° tile at full resolution. The data is provided in a tar file named `dn41w071.tar`. The DTED0 elevation data file is in the `w071` directory under the name `n41.dt0`. Read and display it as a lighted surface to show both large and small scale variations in the data. Because no bathymetric depths are provided, move elevations of zero down a bit to color them blue.

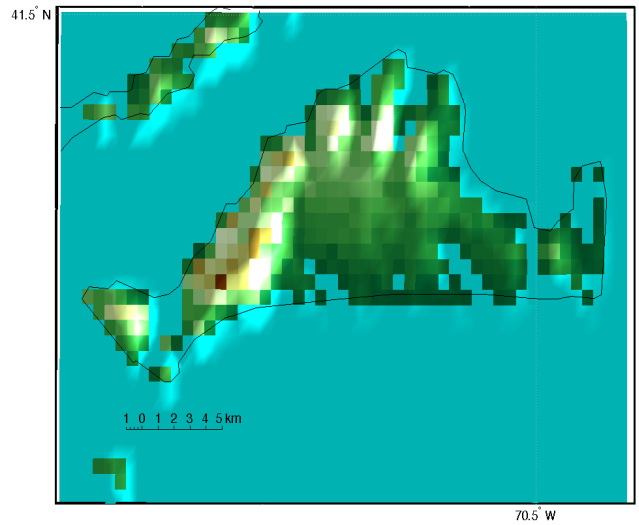
```
[map,maplegend] = dted('n41.dt0'); map(map==0)=-1;
[latlim,lonlim] = limitm(map,maplegend);

usamap(latlim,lonlim,'line')
```

```
meshm(map,maplegend,size(map),map); demcmap(map)
daspectm('m',40)
zdatam('allline',max(map(:)))
light; material([0.7 1 0.6]); lighting phong
scaleruler
```



Read and display the data covering just Martha's Vineyard at the full resolution.

```
clmo surface
latlim = [41.2344 41.5053]; lonlim = [-70.8613 -70.4249];
setm(gca,'MapLatLimit',latlim,'MapLonLimit',lonlim); tightmap

[map,maplegend] = dted('n41.dt0',1,latlim,lonlim);
map(map==0)=-1;

meshm(map,maplegend,size(map),map); demcmap(map)
light; material([0.7 1 0.6]); lighting phong; daspectm('m',30)
scaleruler off; scaleruler
```



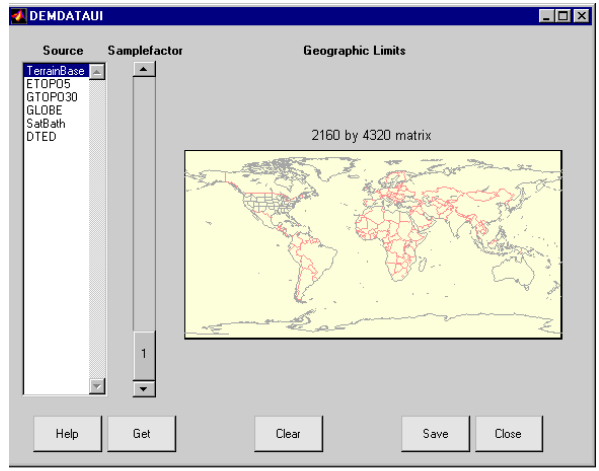
You can read the DTED data with the `demdataui` graphical user interface described below.

Reading Elevation Data Interactively

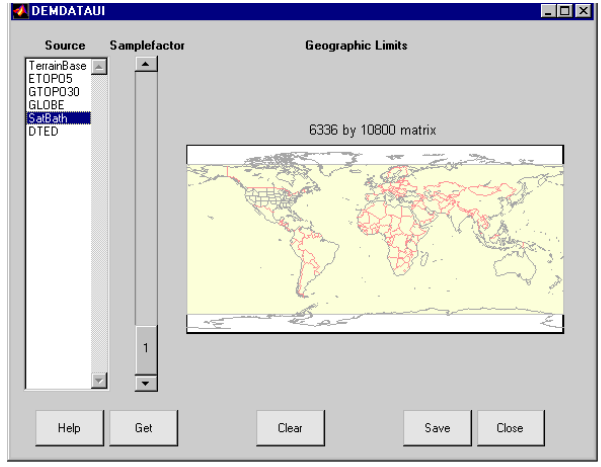
Most of the digital elevation map datasets can also be read using a graphical user interface called `demdataui`. The `demdataui` panel allows you to read data from ETOPO5, TerrainBase, the satellite bathymetry model, GTOPO30, GLOBE, and DTED.

When you start `demdataui`, it scans your MATLAB path for the presence of any of these types of data files. On PCs, it also checks the root directory of CD-ROMs and other drives. You can then choose to read from any of the datasets. If `demdataui` has trouble recognizing data that you think it should find, check your path and use the **Help** button to read about how files are identified.

```
demdataui
```



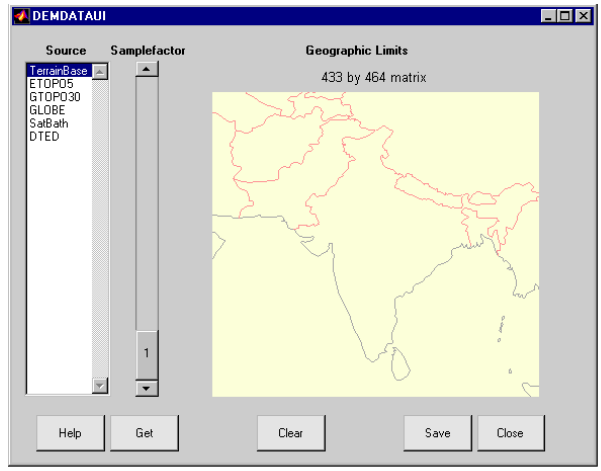
The **Source** list shows the datasets that were found. The coverage of each dataset is shown in yellow on the map. Clicking on a different source in the list will update the yellow coverage area. Here is the coverage area for the satellite bathymetry dataset



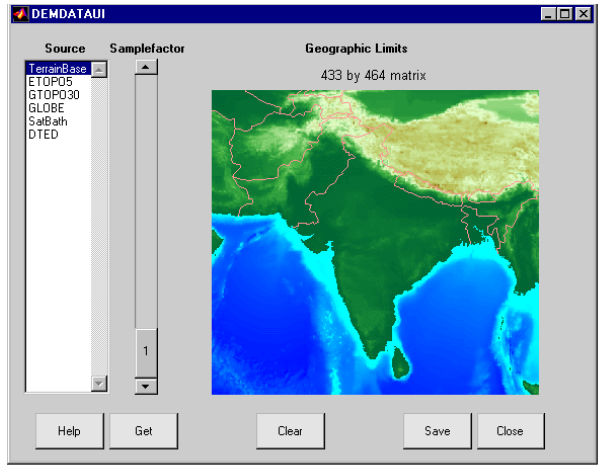
The map is used to determine how much data to extract. You can look above the map to get an estimate of how large the matrix will be. To reduce the

amount of data, click or click and drag on the map to zoom in, or use the **Samplefactor** slider. A sample factor of 1 reads every point, whereas 2 reads every other point.

Here is the panel after selecting the TerrainBase data and zooming in on the Indian subcontinent

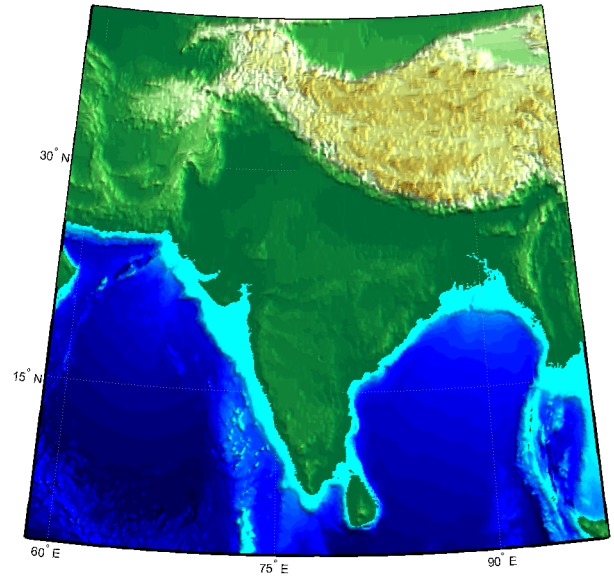


Clicking the **Get** button reads from the selected dataset at the slider's sample factor and with geographic limits taken from the map. The result of this operation should be a matrix with about 400 by 500 cells, a reasonable amount of data. The result is shown on the map.



If you are not satisfied with the result, press the **Clear** button to remove all previously read data, and change your choices. When you are ready, save the matrix maps to a MAT-file or the MATLAB workspace by pressing the **Save** button. The matrices are returned in a geographic data structure, with one entry for each separate “get”. You can then use `displaym` or `mlayers` to add the matrix maps to a map axes. To access the contents of the geographic data structure, use the field names. Here the map and maplegend are popped out of the structure and used to create a lighted three-dimensional elevation map display using `worldmap`.

```
map = demdata.map;
maplegend = demdata.maplegend;
figure
worldmap(map,maplegend,'ldem3d')
hidem(gca)
```



Advanced Very High Resolution Radiometer (AVHRR) Global Change Data

The National Oceanic and Atmospheric Administration (NOAA) and others have compiled a global high resolution set of land vegetation measurements to monitor global climate change. The data is derived from the Advanced Very High Resolution Radiometer (AVHRR) sensor carried on NOAA meteorological satellites (NOAA-7, -9, 11). The sensor measures reflected radiation in five different wave-bands with a 1 km spatial resolution, which is then processed into a Non-dimensional Vegetation Index (NDVI). The NDVI has been used as the basis for a global landcover classification dataset. Other applications of the AVHRR data include oceanographic and atmospheric science. Many of those datasets are stored in HDF formats, which can be read using the MATLAB `imread` or `hdfread` functions. This section focuses on the land datasets, which are stored in several different map projections and require special interface functions.

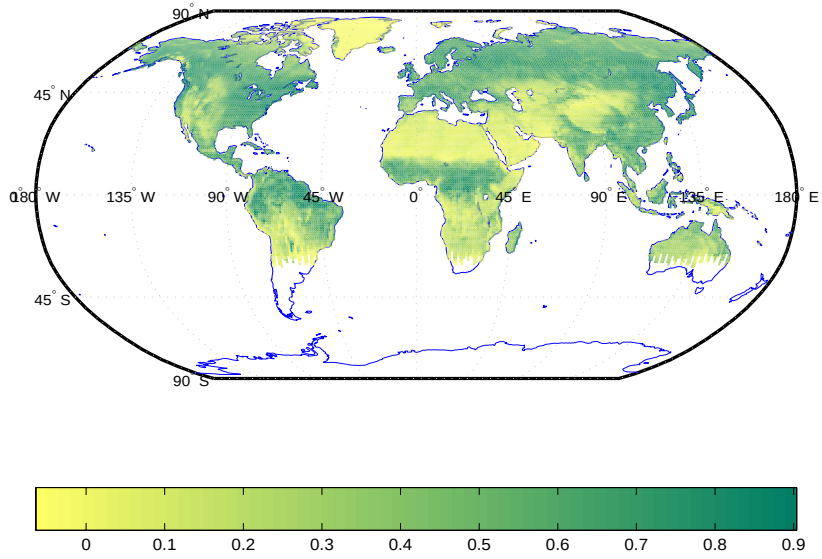
Low resolution monthly NDVI composites are available from

`<ftp://daac.gsfc.nasa.gov/data/inter_disc/biosphere/avhrr_ndvi/>`

These are binary files stored as a 1 degree geographic grid. You can read these files with `readmtx`, a Mapping Toolbox function that reads matrices stored in files based on a description of the file format. Here is an example of the non-dimensional vegetation index data for September 1994. This data was retrieved from <ftp://daac.gsfc.nasa.gov/data/inter_disc/biosphere/avhrr_ndvi/1994/avhrr_pf.ndvi.1nmegl.9409.bin>.

Try the following:

```
map = readmtx('avhrr_pf.ndvi.1nmegl.9409.bin',180,360,'real*4');
map = flipud(map);
map(map<-9)=NaN;
maplegend = [1 90 -180];

worldmap('world','none'); plotm(coast);
meshm(map,maplegend,size(map))
colormap(flipud(summer))
colorbar('horiz')
```



What are the values of the non-dimensional vegetation index at particular locations?

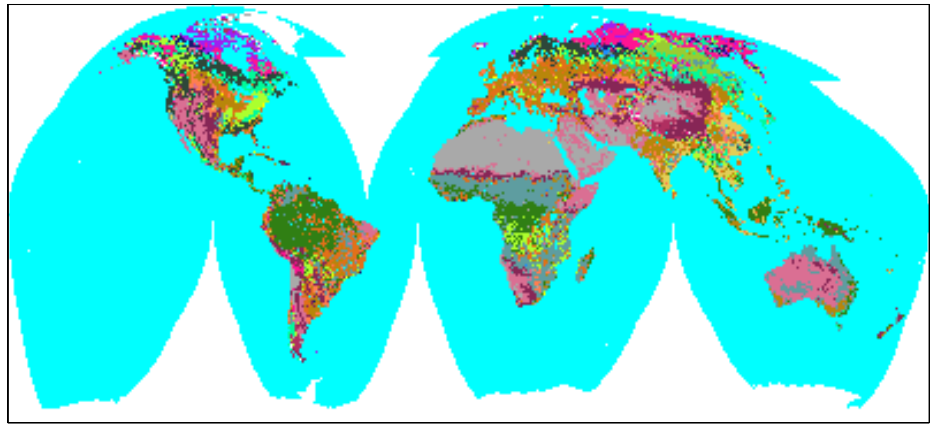
```
[lat,lon] = extractm(worldlo('gazette'),...
     {'Vancouver','Oslo','Cairo','Lagos','Tripoli'});
lat = lat(~isnan(lat)); lon = lon(~isnan(lon));
val = ltln2val(map,maplegend,lat,lon);
```

Other AVHRR products are stored as matrices in projected coordinates. The standard projection for the 1 kilometer resolution data is the interrupted Goode, which can be read using the `avhrrgoode` interface function. Some of the data is also available in the Lambert Equal-Area Azimuthal projection. Extracting data from the Lambert files with `avhrrlambert` is much faster. If you have a choice, you should probably read data in the Lambert format.

The global 1 kilometers data in the Goode projection is stored as a binary matrix with 17347 rows and 40031 columns. The uncompressed file size is about 695 megabytes, slightly more than the capacity of one CD-ROM disk. Here is an image of the data as it is stored. This example uses the Global Land Cover Classification data in the USGS classification scheme, available from <ftp://edcftp.cr.usgs.gov/pub/data/glcc/globe/gusgs1_2.img.gz>. The colors correspond to different types of vegetation and land use.

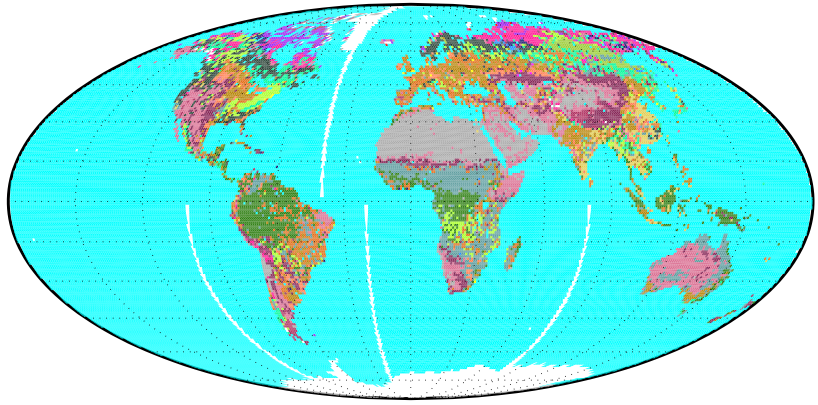
```
nrows = 17347;   % 16347 for some data on cd-rom
ncols = 40031;   % 40031
mtx = readmtx('gusgs1_2.img',nrows,ncols,'int8',...
              1:100:nrows,1:100:ncols);

h = image(mtx);
axis image

load usgslulegend
colormap(cmap); set(h,'CdataMapping','Direct')
showaxes off
```

Here is the displayed image:



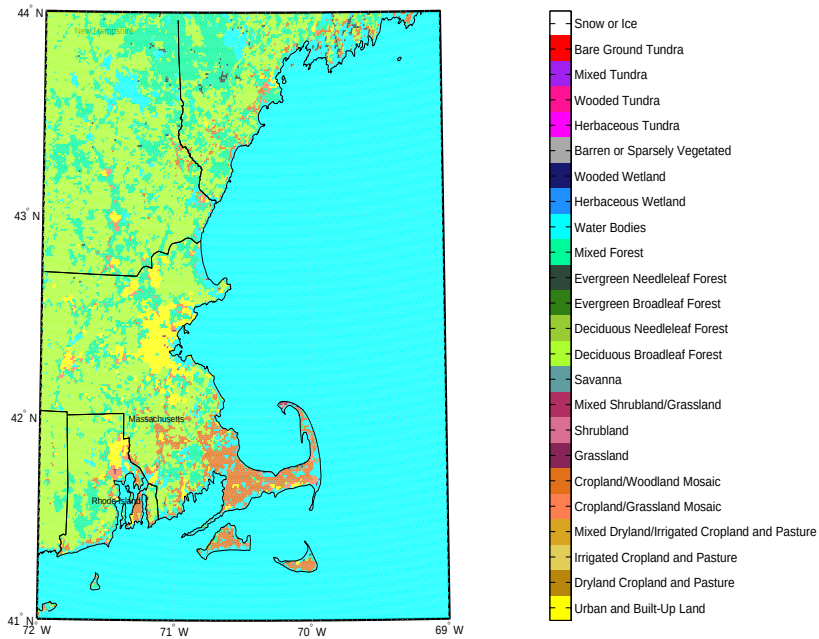
To display this data on a map projection, extract the data with the avhrrgoode function. The next example reads the data at the same resolution as above, returning a general matrix map consisting of the latitude, longitude, and data matrices.

```
[latgrat,longrat,map] = avhrrgoode('global','gusgs1_2.img',100);

figure; axesm mollweid; framem; gridm;
surfm(latgrat,longrat,map); colormap(cmap)
set(handlem('surface'),'CdataMapping','Direct')
```



You can read the data for just a part of the world and control the resolution of the extracted data and the size of the graticule matrices. Data is sometimes provided in regional subsets of the very large global Goode files. You can read these by supplying the name of the region. For example, read the USGS land cover data for Cape Cod at the full resolution from the North America file, available from <ftp://edcftp.cr.usgs.gov/pub/data/glcc/na/goode/nausgs1_2g.img.gz>.

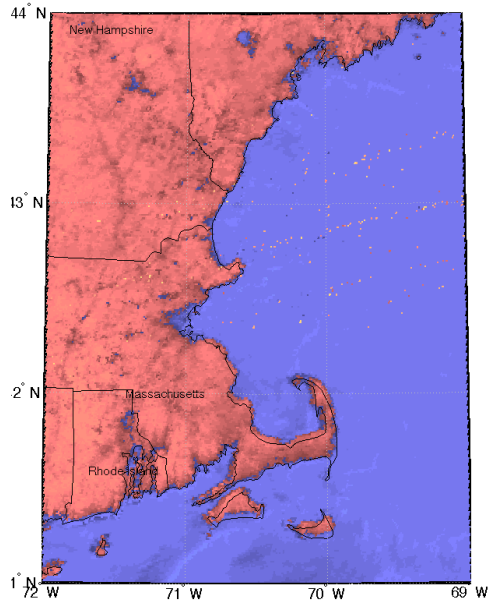
```
latlim = [41 44]; lonlim = [-72 -69];
[latgrat,longrat,map] = avhrrgoode('north america',...
     'nausgs1_2g.img',1,latlim,lonlim);

usamap(latlim,lonlim,'line')
surfm(latgrat,longrat,map); colormap(cmap)
set(handlem('surface'),'CDataMapping','Direct')

set(gca,'Position',[.1 .1 .5 .8])
caxis([.5 24.5]); hcb = colorbar;
set(hcb,'YTick',1:24,'YTickLabel',USGSLandUse,...
    'Position',[.6 .1 .02 .8])
```



The full matrix of AVHRR data is too large to fit on one CD-ROM disk. When the data is provided on CD-ROM, the size of the data is reduced by dropping the last rows. To read the truncated files, specify the number of rows and columns in the non-standard file. This information can usually be found in the 'Readme' file. Here is the non-dimensional vegetation index data for Cape Cod from 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). The empty matrix argument specifies that the graticule size will be the same size as the map. The geographic registration for this older CD seems to be less accurate.

```
[latgrat,longrat,map] = avhrrgoode('global','NDVI.IMG',...
     1,latlim,lonlim,[],16347,40031);

figure
usamap(latlim,lonlim,'line')
surfm(latgrat,longrat,map)

% Lighten the colormap
cmap = flipud(jet(64));
hsvmap = rgb2hsv(cmap); hsvmap(:,2) = .5; cmap = hsv2rgb(hsvmap);
colormap(cmap)
```

Here is the result:



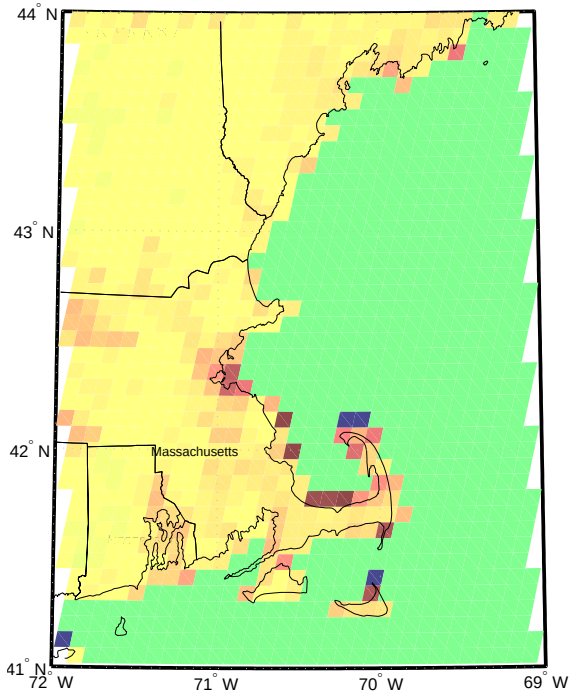
AVHRR data is also available in the Goode projection at a resolution of 8 kilometers. You can read the global 8 km files with avhrrgoode, but not the 8 km regional files.

Read the global 8 km resolution non-dimensional vegetation index for Cape Cod. This data file is available from

<ftp://daac.gsfc.nasa.gov/data/avhrr/global_8km/.1994_1997/1994/jun/avhrrpf.ndvi.1ntfgl.940621.gz>

Specify the non-standard number of rows, columns, and resolution in the file. The jagged edges in the surface are a result of trimming a slanted graticule to the map frame.

```
[latgrat,longrat,map] = avhrrgoode('global',...
     'avhrrpf.ndvi.1ntfgl.940621',1,...
     latlim,lonlim,[],2168,5004,8000);

clmo surface
surfm(latgrat,longrat,map)
```

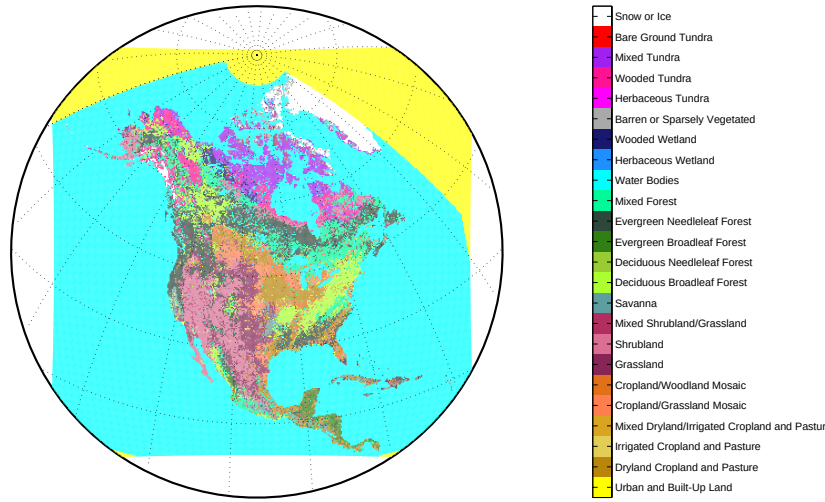


Regional versions of the 1 km Global Land Cover Classification data are also available saved in Lambert azimuthal equal area projections. These can be read using the `avhrrlambert` function. Reading data from the Lambert projection will be faster than from the interrupted Goode, because the graticule does not need to be trimmed to the boundaries of the individual projection zones.

Read the Global Land Cover Characteristics (GLCC) file covering North America in the Lambert projection with the USGS classification scheme. 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',25);

axesm('eqdazim','Origin',[50 -100],'FLatLimit',[-Inf 50],...
      'mlinelocation',15); framem; gridm
surfm(latgrat,longrat,map)
```

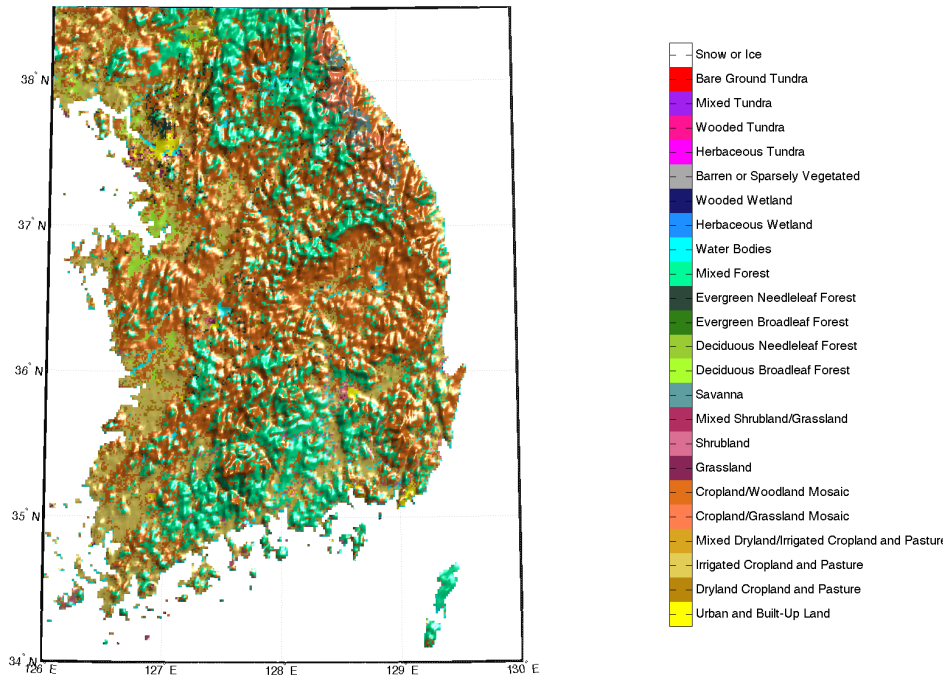
```
load usgslulegend; colormap(cmap)
set(handlem('surface'),'CDataMapping','Direct'); tightmap
caxis([.5 24.5]); hcb = colorbar;
set(hcb,'YTick',1:24,'YTickLabel',USGSLandUse,...
    'Position',[.7 .1 .02 .8])
set(gca,'Position',[-.05 .1 .8 .8])
```



It is sometimes useful to drape land cover data onto elevation maps. The elevation data is displayed as a three-dimensional surface, onto which the new land cover map is texture mapped. The lighting effects allow you to see the land use change from intensive agriculture in low-lying flatlands to mixed cropland and woodland, and finally to pure woodland in more mountainous areas. Here are two examples combining the Global Land Cover Classification data with the GTOPO30 elevations. The first converts the general matrix map of land cover codes to a regular matrix map and combines it with a regular matrix map of elevations.

```
latlim = [34 38.5]; lonlim = [126 130];

% <ftp://edcftp.cr.usgs.gov/pub/data/glcc/ea/lamberta/
%        eausgs1_2la.img.gz>
[latgrat,longrat,mat] = avhrrlambert('asia',...
     'eausgs1_2la.img',1,latlim,lonlim);
```

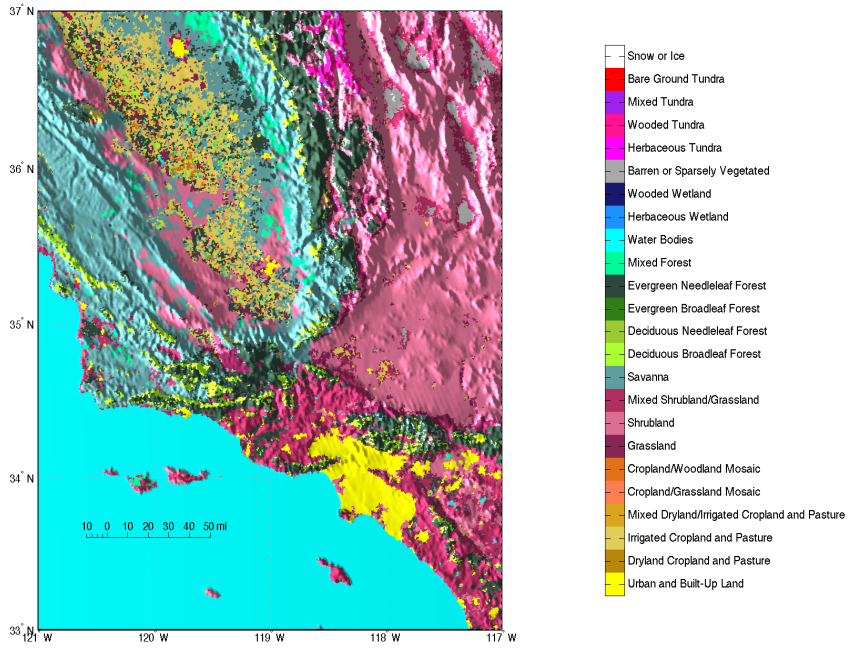
```
[lcmap,lcmaplegend] = nanm(latlim,lonlim,.7/km2deg(1));
lcmap = imbedm(latgrat,longrat,mat,lcmap,lcmaplegend);

% <ftp://edcftp.cr.usgs.gov/pub/data/gtopo30/global/
%        e100n40.tar.gz>
[map,maplegend] = gtopo30('e100n40',2,latlim,lonlim);

worldmap(latlim,lonlim,'none')

h = meshm(map,maplegend,size(map),map)
set(h,'FaceColor','texturemap','CData',lcmap)
colormap(cmap)  % predefined colormap
set(h,'CDataMapping','direct')  % integer land cover codes

set(gca,'Position',[.1 .1 .5 .8])
daspectm('m',10); camlight; lighting phong;

caxis([.5 24.5]); hcb = colorbar;
set(hcb,'YTick',1:24,'YTickLabel',USGSLandUse,...
     'Position',[.6 .1 .02 .8])
```



The other approach is to interpolate a regular matrix map of elevations into a the general matrix. Here is an example using the land cover data and elevations for Southern California. You can see how forests are predominantly at the higher elevations around Los Angeles and the San Joaquin Valley. Notice also the string of urban areas along the treeline of the Sierra Nevada Mountains. These do not appear in other map sources, and may be spurious.

```
latlim = [33 37]; lonlim = [-121 -117]; % socal

[latgrat,longrat,lcmap] = avhrrlambert('north america',...
     'nausgs1_2l.img',1,latlim,lonlim);

latgratlim = [min(latgrat(:)) max(latgrat(:))];
longratlim = [min(longrat(:)) max(longrat(:))];

[elmap,elmaplegend] = gtopo30('W140N40',1,...
     latgratlim,longratlim);
elmap(isnan(elmap)) = -1;
```

```
z = ltln2val(elmap,elmaplegend,latgrat,longrat);

figure; worldmap(latlim,lonlim,'none');
setm(gca,'MapProjection','mercator')
hs = surfm(latgrat,longrat,lcmap,z);
set(hs,'CDataMapping','direct')
load usgslulegend; colormap(cmap)

daspectm('m',5); material([0.6 2 0])
camlight(-80,0); lighting phong
caxis([.5 24.5]); hcb = colorbar;
set(hcb,'YTick',1:24,'YTickLabel',USGSLandUse,...
     'Position',[.7 .1 .02 .8])
set(gca,'Position',[-0.04.05 .8 .9])
scaleruler('Units','mi'); tightmap
```



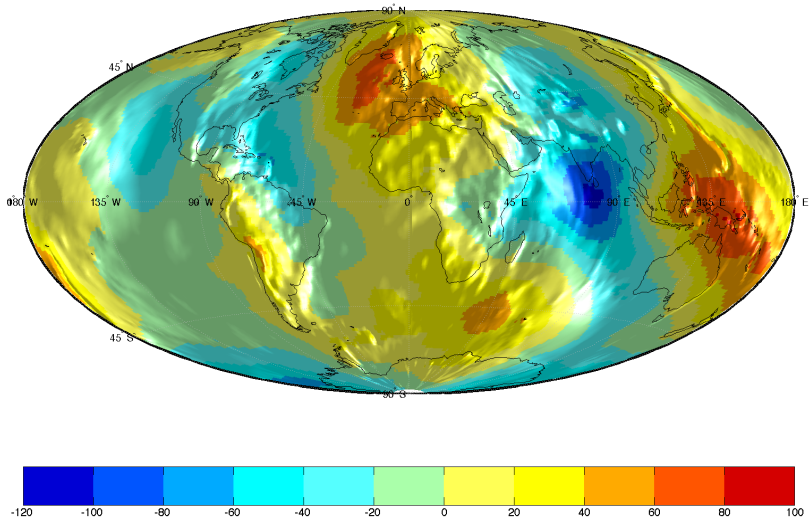
EGM96 Geoid Model

Much of the available matrix map data is elevations of the Earth's crust. For some applications it is important to know precisely the elevation and slope of sea level, or in the case of areas covered with land, where sea level would be. This surface of constant gravitational attraction is called the *geoid*, and improving man's understanding of it has been one of the primary concerns of the field of geodesy. Perhaps the best available global geoid model is EGM96, developed by NASA Goddard and NIMA. EGM 96 is a spherical harmonic model of the geoid complete to degree and order 360. The lower resolution atlas data in `geoid.mat` is derived from the EGM 96 grid.

The `egm96geoid` function reads the '`WW15MGH.GRD`' file of geoid heights derived from the EGM 96 harmonic coefficients on a 15 minute grid. There are 721 rows and 1441 columns of values in the grid at full resolution. PC users should download the DOS self-extracting executable file 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>`.

Read the EGM 96 geoid grid for the world, taking every fifth point. Display it as a lighted surface with colormap contours in a Hammer projection.

```
[map,maplegend] = egm96geoid(5);

worldmap('world','none');
setm(gca,'MapProjection','hammer'); tightmap
meshm(map,maplegend,size(map),map); plotm(coast,'k')

light; lighting phong; material(0.6*[ 1 1 1])
set(gca,'DataAspectRatio',[1 1 200]); gridm reset
zdatam(handlem('allline'),max(map(:)))
zdatam(handlem('alltext'),max(map(:)))

caxis([-120 100]); colormap(jet(11)); colorbar('horiz')
```



The high resolution grid is most useful for computing the geoid height at particular locations. If you plan to use a higher order interpolation method, be sure to get a matrix large enough to have several points about your locations.

Read 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 =
   52.7151
```

U. S. Gridded Elevation Data

U. S. Geological Survey 1:250,000 (100 meter) DEMs

The U. S. Geological Survey has released a series of Digital Elevation Maps in 1-degree quadrangles covering the contiguous United States, Hawaii, and limited portions of Alaska. The 1-degree DEMs are also referred to as *3-arc second* or *1:250,000 scale* DEM data. The horizontal resolution of this data is about 100 meters.

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 are maps from the 7.5 minute through 1-degree series (1:24,000 scale through 1:250,000 scale). There is no bathymetric data in this DEM.

Most of the 1-degree digital elevation models have grid spacings of 3 arc-seconds in both the latitude and longitude directions. Alaska has grid spacings of 6 arc-seconds for latitudes between 50 and 70 degrees North and 9 arc-seconds for latitudes greater than 70 degrees North.

The grid for the digital elevation maps is based on the World Geodetic System 1984 (WGS 84). Older DEMs were based on WGS 72. Elevations are in meters relative to the National Geodetic Vertical Datum of 1929 (NGVD 29) in the continental U.S. and local mean sea level in Hawaii.

The specified absolute horizontal accuracy of the DEM is 130 m, while the specified absolute vertical accuracy is 30m. The relative horizontal and vertical accuracies are not specified but should be much better than the absolute accuracies.

The DEM data files can be retrieved over the Internet from the EROS Data Center's Web site located at <http://edcwww.cr.usgs.gov/> or their FTP server at <ftp://edcftp.cr.usgs.gov/>. The files are available sorted by state and file name or can be selected from an index map. Further information on the dataset is also available at the sites listed.

The data files are named according to towns or features within the quadrangle region. If you know the geographic limits of the area you want to map, you can look up the names on the USGS Web site or use the Mapping Toolbox `usgsdems` function, which returns a list of map quadrangle filenames covered by your region. Here are the names of the files covered by the Cape Cod maps used throughout this document:

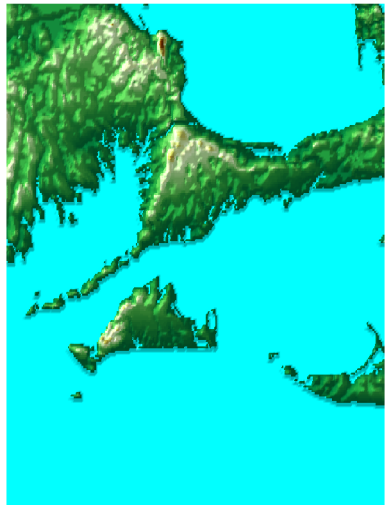
```
usgsdems([41 44], [-72 -69])
ans =
    'providence-w'
    'providence-e'
    'chatham-w'
    'boston-w'
    'boston-e'
    'portland-w'
    'portland-e'
```

```
'bath-w'
```

The Cape Cod region extends over several of the 1-degree quadrangles. Displaying the whole region at the full, 3 arc-second resolution would require enormous amounts of memory and time.

Display the region covered by the 'providence-e' quadrangle using the usgsdem interface function and a downsampling factor of 5. The resulting map is a 241-by-241 element matrix. At full resolution, the map would be 1201-by-1201.

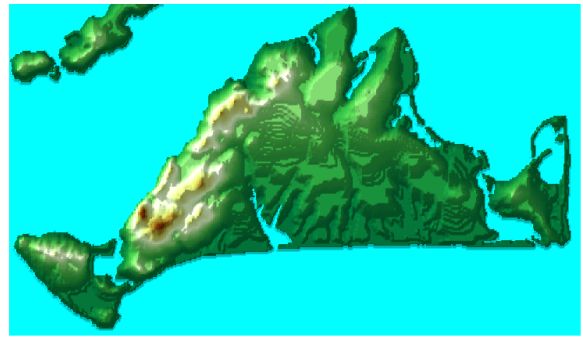
```
[map,maplegend] = usgsdem('providence-e',5);
map(map==0) = -1;

axesm mercator
meshm(map,maplegend,size(map),map); demcmap(map)
lightm(41.5,-70.5); material([.7 .7 1.5]); lighting gouraud
```



Zoom in further on Martha's Vineyards by providing appropriate latitude and longitude limits for the usgsdem function at full resolution. You can see some terracing effects in the data, an artifact of its derivation from topographic contour maps.

```
[map,maplegend] = usgsdem('providence-e',1,[41.2946 41.4829],...
                          [-70.8429 -70.4379]);
axesm mercator
```

```
meshm(map,maplegend,size(map),map); demcmap(map)
lightm(41.4,-70.65); material([.7 .7 1.5]); lighting gouraud
```



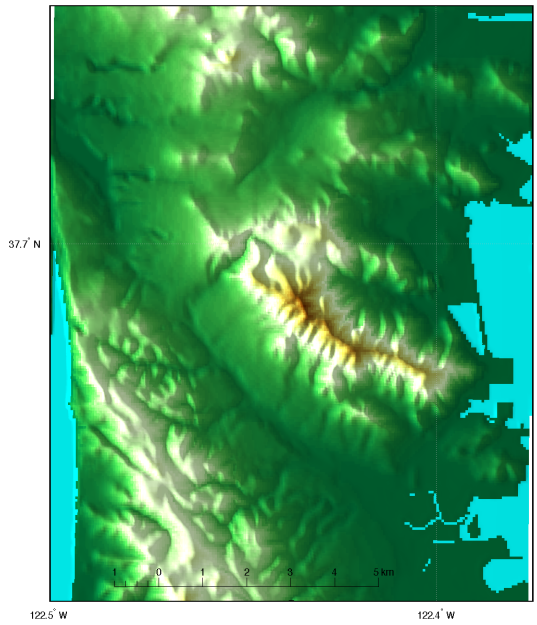
U.S. Geological Survey 1:24,000 (30m) DEMs

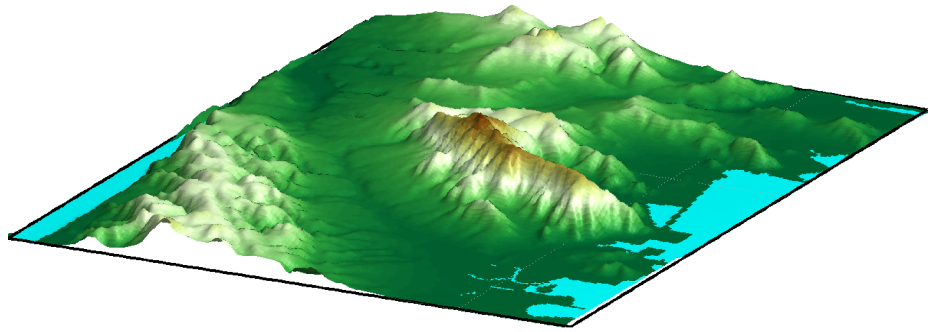
The highest resolution elevation data generally available is the USGS 1:24,000 scale DEMs. These maps have been generated from the contour lines on the paper 7.5 minute quadrangle map series. Data is available for much of the contiguous United States, Hawaii, and Puerto Rico. Because of the density of this data, little of it is available on-line. Some state agencies have collected and published files for smaller areas, such as the California Bay Area Regional Database at <ftp://bard.wr.usgs.gov/bard/dem/>. DEM files for other areas can be ordered from the USGS. These files have been released in several file formats. The USGS in the process of converting much of this data into the Spatial Data Transfer Standard. The `usgs24kdem` function reads data in the older standard format.

The data is stored as a series of profiles spaced 30 meters apart in a Universal Transverse Mercator grid. Unlike the 1:250,000 DEMs, adjacent quadrangles cannot be simply butted together to form a larger, gap-free map. There is no bathymetric data. The DEM files are ASCII files and can be transferred as text. Line ending conversion is not necessarily required.

Read 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. The result is a 232-by-186 general matrix map. Because there are no negative elevations, move points at sea level down to color them blue.

```
[latgrat,longrat,map] = usgs24kdem('sanfranciscos.dem',2);
map(map==0) = -1;

latlim = [min(latgrat(:)) max(latgrat(:))];
lonlim = [min(longrat(:)) max(longrat(:))];

usamap(latlim,lonlim,'none')
surfm(latgrat,longrat,map,map-max(map(:)))
demcmap(map); daspectm('m',1); scaleruler
camlight(-80,0); lighting phong; material([.7 1 0])
```





Astronomical Data

The Mapping Toolbox provides an interface to a set of astronomical data called FK5, the Fifth Fundamental Catalog of Stars. This catalog consists of positions, proper motions, and other characteristics for over 4500 stars. The data is provided by the Astronomical Data Center (ADC), located at the NASA Goddard Space Flight Center, and can be retrieved over the Internet at the ADC's home page at <http://adc.gsfc.nasa.gov/> or by anonymous FTP at <ftp://adc.gsfc.nasa.gov/>. Documentation and other information can be found at these sites as well.

The data comes in two files, 'FK5.dat' and 'fk5_ext.dat'. You can read both files using the readfk5 interface function. There are more than four thousand stars in the catalog, so reading and processing the data takes a while.

```
fk5  = readfk5('FK5.dat');
fk5e = readfk5('fk5_ext.dat');
whos
   Name       Size       Bytes     Class

   fk5        1x1535     5042752   struct array
   fk5e       1x3117     10226424  struct array
```

The star catalog is processed into MATLAB structures by the external interface function. The fields of the structures contain the different types of information for each star, while the structure elements represent the individual stars.

To view the fields of the structure, type the following:

```
fk5(1)
ans =
          FK5: 1
          RAh: 0
          RAm: 8
          RAs: 23.2650
        pmRA: 1.0390
          DEd: 29
          DEm: 5
          DEs: 25.5800
        pmDE: -16.3300
     RAh1950: 0
```

```
 RAm1950: 5
 RAs1950: 47.8770
pmRA1950: 1.0360
 DEd1950: 28
 DEm1950: 48
 DEs1950: 51.9600
pmDE1950: -16.3300
EpRA1900: 43.3100
   e_RAs: 0.7000
  e_pmRA: 2
EpDE1900: 33
   e_DEs: 1.3000
  e_pmDE: 3.1000
    Vmag: 2.0600
  n_Vmag: ''
  SpType: 'A0p'
     plx: 0.0240
      RV: -11.7000
   AGK3R: ''
     SRS: ''
      HD: '358'
      DM: 'BD+28    4'
      GC: '127'
```

If you just want to display the locations of the stars, you can find position and visual magnitude data in the `stars` workspace as part of the Mapping Toolbox atlas data. The remainder of this section illustrates the use of Mapping Toolbox functions to manipulate and transform star data for display.

The star positions are given in terms of right ascension and declination. You can display a star map by converting the positions to latitude and longitude coordinates. Right ascension is given in hours, minutes, and seconds, while declination is in degrees, minutes, and seconds.

```
hRA = [fk5.RAh fk5e.RAh]'; dDE = [fk5.DEd fk5e.DEd]';
mRA = [fk5.RAm fk5e.RAm]'; mDE = [fk5.DEm fk5e.DEm]';
sRA = [fk5.RAs fk5e.RAs]'; sDE = [fk5.DEs fk5e.DEs]';
lat = dms2deg(dDE,mDE,sDE);
lon = hms2hr(hRA,mRA,sRA)*360/24;
```

You probably want the map to look like you are inside the celestial sphere looking out, so switch the sign of the longitudes:

```
lon = zero22pi(-lon);
```

It is customary to display the stars with sizes proportional to the visual magnitudes. Extract the magnitudes from the data structures:

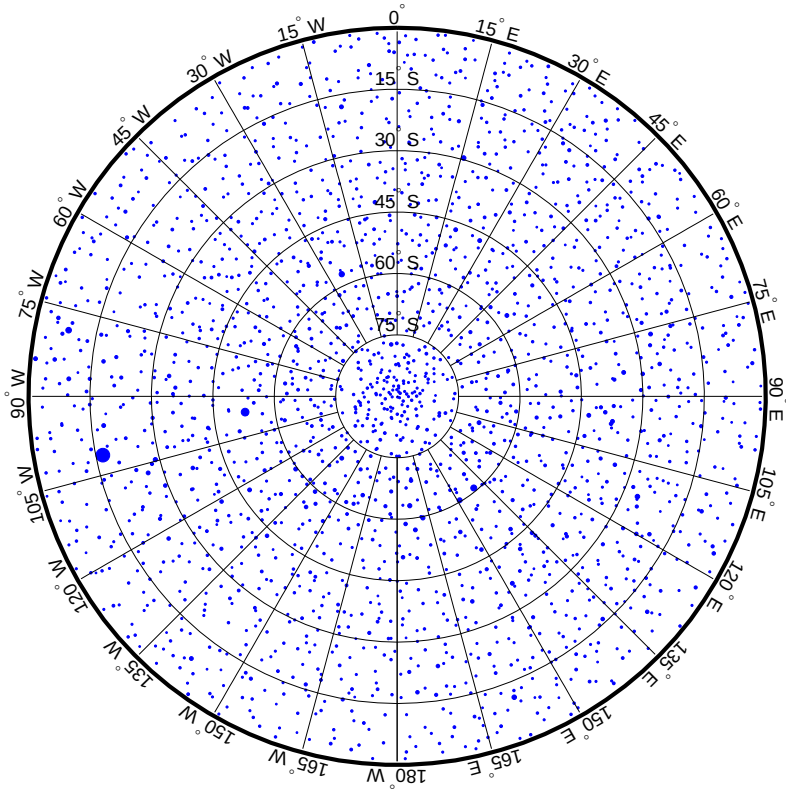
```
magn = [fk5.Vmag fk5e.Vmag]';
```

Faint stars have large visual magnitude numbers, while bright ones have smaller values. The magnitudes of particular bright stars were used to define the visual magnitude scale. It was later determined that certain stars were even brighter than the reference stars, resulting in negative visual magnitudes. You will need to convert the values to relative intensities for plotting.

```
relintensity = (magn+2).^(-100^(1/5));
clear fk5 fk5e
```

Stars in equatorial coordinates are typically displayed in the Equidistant Azimuthal projection. Here is the southern sky. You can see the southern cross at about 60 S, 175 E.

```
axesm eqdazim
framem; gridm; mlabel; plabel
setm(gca,'Origin',[-90 180 0],'FLatLimit',[-inf 90],...
   'MLineLocation',15,'MLabelLocation',-180:15:165,...
   'MLabelParallel','equator','MLineLimit',[-75 75],...
   'PLabelLocation',-15:-15:-75,'PlabelMeridian',...
   'LabelRotation','on','GLineWidth',.01,'GLineStyle','-')
set(handlem('alltext'),'HorizontalAlignment','center',...
  'VerticalAlignment','bottom')
scatterm(lat,long,sqrt(relintensity)*20,'filled')
```



The sky is often depicted in galactic coordinates, with the center of our galaxy as the origin of a polar coordinate system. You can use the Mapping Toolbox to transform the star positions from equatorial coordinates to galactic coordinates. First define the location of the galactic pole and the galactic center.

```
% galactic pole
poleRAhms = mat2hms([12 49 0]);
poleDEhms = mat2hms([27 24 0]);
polelat = dms2deg(poleDEhms);
polelon = zero22pi(-hms2hr(poleRAhms)*360/24);
```

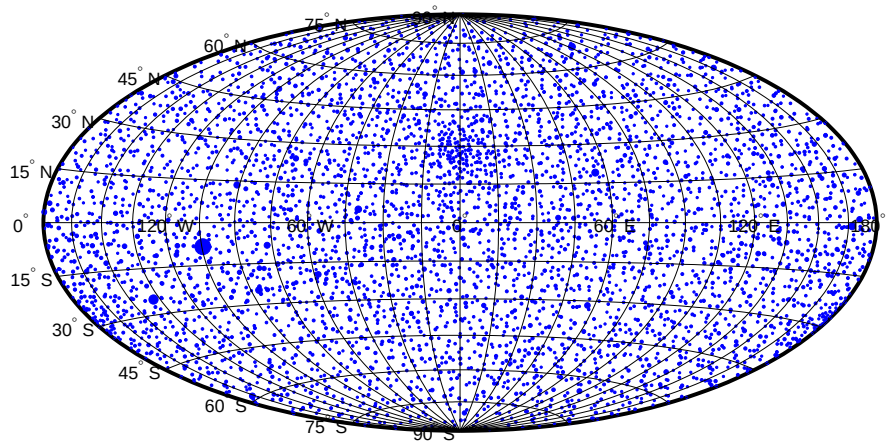
```
% galactic center
cntrRAhms = mat2hms([17 42 0.4]);
cntrDEhms = mat2hms([-28 55 0]);
cntrlat = dms2deg(cntrDEhms);
cntrlon = zero22pi(-hms2hr(cntrRAhms)*360/24);
```

Next create a new origin vector based on the location of the new pole in the old equatorial coordinate system. Use it to rotate the locations of the pole and the center of the galactic coordinates. Note that the origin provided by the `newpole` function results in a system centered on the location of the new pole in the old coordinate system. Rotate the star positions and bring the galactic center to zero longitude.

```
% new origin vector
origin = newpole(polelat,polelon);

% rotate our known points
[newpolelat,newpolelon] = rotatem(polelat,polelon,...
                                  origin,'forward','degrees');
[newcntrlat,newcntrlon] = rotatem(cntrlat,cntrlon,...
                                  origin,'forward','degrees');

% rotate the stars
[glat,glon] = rotatem(lat,lon,origin,'forward','degrees');
glon = glon - newcntrlon;
```

The stars in galactic coordinates are commonly displayed in a pseudo-cylindrical projection like the Hammer.

```
figure
axesm hammer
framem; gridm; mlabel; plabel
setm(gca,'MLabelLocation',-120:60:180,...
   'MLineLocation',15,'MLabelParallel','equator',...
   'GLineWidth',.01,'GLineStyle','-')
scatterm(glat,glon,sqrt(relintensity)*30,'filled')
```


75° N
60° N
45° N
30° N
15° N
0°
15° S
30° S
45° S
60° S
75° S
90° S
120° W
60° W
0°
60° E
120° E
180° E

Importing Other Data

With the huge quantities of geographic information available, it is impossible to provide interface functions for every file format. To help you read data in other formats, the Mapping Toolbox provides a set of high–level file import functions. These allow you to read data from binary or text files by simply describing the contents of the file. The files may contain matrices or fixed length records of geographic data. MATLAB also provides some file import functions. Type `help iofun` for more information on the MATLAB functions.

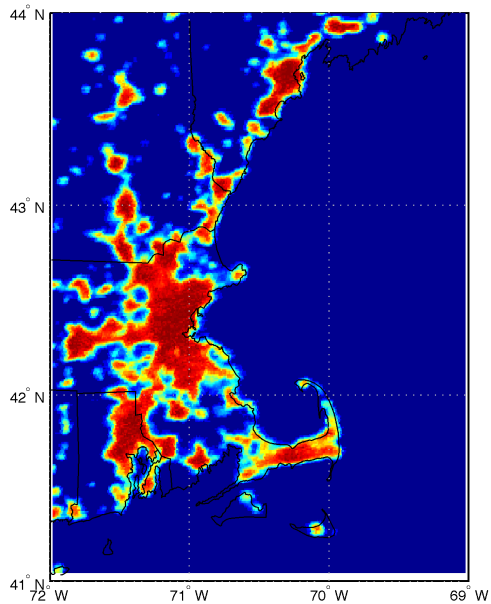
Much geographic data is provided as very large files containing matrices of values. You can read the part of the matrix of interest to you using the `readmtx` function. To illustrate the use of `readmtx`, read in data giving the intensity of stable light sources collected from the Defense Meteorological Satellite program. The data files are available from <ftp://ftp.ngdc.noaa.gov/DMSP/lights/>. We will read the 24 megabyte 'usa_pct.latlon.dat' file for the Cape Cod region. The README file describes the format and contents of the files. From this you need the following minimum information: the number of rows and columns in the matrix, the location of the upper left corner, the grid spacing, and the storage format of the numbers.

Most matrix files can be read using the same procedure. First you map your geographic limits into matrix row and column limits and construct vectors of row and column indices to be read. Then you read the submatrix using `readmtx`, and construct the `maplegend`.

Here is an example:

```
% we want to read
latlim = [41 44]; lonlim = [-72 -69]; scalefactor = 1;

% file format and contents
nrows = 3240; ncols = 7800;
lat1 = 51; lon1 = -129;
yperrow = -dms2deg(mat2dms(0,0,30));
xpercol =  dms2deg(mat2dms(0,0,30));

% map latlim and lonlim to row and column limits
[rlim,clim] = yx2rc(latlim,lonlim,lat1,lon1,yperrow,xpercol)
rlim = sort(rlim);

readrows = rlim(1):scalefactor:rlim(2);
readcols = clim(1):scalefactor:clim(2);
```

```
% extract the map matrix
map = flipud(readmtx('usa_lights.latlon.dat',...
             nrows,ncols,'int8',readrows,readcols));

% construct maplegend
cellsize = scalefactor*xpercol;
maplegend = [1/cellsize, latlim(2)-cellsize/2, ...
             lonlim(1)-cellsize/2];

% display result
usamap(latlim,lonlim,'lineonly')
setm(gca,'mapproj','mercator'); tightmap
meshm(map,maplegend)
```



Boston appears to have grown since the data in the VMAP0 was compiled. The VMAP0 also missed the heavily populated part of Cape Cod.

Some data is stored on regular grids in projected coordinates. In that case, you need to define a map projection that matches the one used for the data. Because

the grid is in projected coordinates rather than latitude and longitude, the map is a general matrix map and needs a graticule. To construct one, build a matrix of row and column indices, convert them to projected coordinates, and carry out the inverse projection. The resulting latitude, longitude, and map matrices can be displayed in any projection. Edit the `avhrrlambert` interface function to see an example.

Another common type of geographic data file format is fixed length records with fields of information. Examples of such data include geographic names, hydrographic soundings, ship track lines, and astronomical catalogs of stars. Use the `readfields` function to read selected fields or records from such files.

As an example, retrieve the U. S. Geological Survey Geographic Names Information System (GNIS) file containing a concise list of names and locations within the United States. The file can be found at `<http://mapping.usgs.gov/pub/gnis/US_concise.gz>`.

The README file describes the file format and contents. To read this data, create a structure with this information.

```
filestruct(1).length = 51;
filestruct(1).name = 'Feature Name';
filestruct(1).type = 'char';

filestruct(2).length = 10;
filestruct(2).name = 'Feature Type';
filestruct(2).type = 'char';

filestruct(3).length = 32;
filestruct(3).name = 'County';
filestruct(3).type = 'char';

filestruct(4).length = 17;
filestruct(4).name = 'State';
filestruct(4).type = 'char';

filestruct(5).length = 16;
filestruct(5).name = 'Geographic Coordinates';
filestruct(5).type = 'char';

filestruct(6).length = 6;
filestruct(6).name = 'Feature Elevation';
filestruct(6).type = 'char';
```

```
filestruct(7).length = 1;
filestruct(7).name = ''; %line ending
filestruct(7).type = 'char';
```

You can identify records that match a search string using the `grepfields` function. With no output arguments, records matching the string are displayed onscreen. When an output argument is provided, the record numbers of matching records are returned.

Search for the locations of The MathWorks and Systems Planning and Analysis.

```
grepfields('us_concise','natick')
Natick                          ppl    Middlesex
Massachusetts    421700N0712100W 180

indx = grepfields('us_concise','alexandria');
```

Use the `readfields` function to read the selected records into a structure.

```
s = readfields('us_concise',filestruct,indx)
s =
1x12 struct array with fields:
    FeatureName
    FeatureType
    County
    State
    GeographicCoordinates
    FeatureElevation

s(7)
ans =
              FeatureName: 'Alexandria'
              FeatureType: 'ppl'
                   County: 'Independent City'
                    State: 'Virginia'
    GeographicCoordinates: '384817N0770250W'
          FeatureElevation: '32'
```

If you are having trouble getting the file format right, you can check the record length of a file with line ending characters. Remember that DOS-based computers use two line ending characters.

```
fid = fopen('us_concise','r');
str = fgets(fid)
str =
Aasu                                ppl     Western
American Samoa   141751S1704530W

recordlength = length(str)
recordlength =
        133.00

fclose(fid);
```

If the file does not have line endings, you can use `fread` to read a few characters at a time.

9

Mapping Reference

Function Tables

Map Definition and Display

Table 9-1: 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 9-2 Projection Calculations

Function	Description
`distortcalc`	3-D contour plot of matrix map data.
`flatearthpoly`	Cuts polygons at dateline.
`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 9-3 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 9-4 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 9-5 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 9-6 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 9-7 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 9-8 Annotation

Function	Description
`clabelm`	Map contour labeling.
`clegendm`	Add legend labels to map contour plot.
`contourcmap`	Contour-like color jumps on surface objects
`demcmap`	Generation of color maps.
`framem`	Toggle display of map frame.
`gridm`	Grid lines projected on map axes.
`lcolorbar`	Labeled colorbar.
`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 9-9 Interaction

Function	Description
gcpmap	Get current mouse point from map.
gtextm	Mouse placement of text on map.
scircleg	Interactive small circles.
sectorg	Interactive sectors of small circles.
trackg	Interactive tracks.
inputm	Latitude and longitude selected by mouse.

Table 9-10 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 9-11 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 9-12 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 9-13 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 9-14 Points

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

Table 9-15 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.
sectorg	Display of sectors of small circles 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 9-16 Polygons

Function	Description
bufferm	Polygon buffer regions.
polybool	Polygon boolean operations.
polyjoin	Combine polygon segments to a NaN-clipped polygon.
polymerge	Merge polygon segments with abutting ends.
polysplit	Separate NaN-clipped polygon segments into cell arrays.
polyxpoly	Polygon intersections.

Table 9-17 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 9-18 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 9-19 Navigation

Function	Description
`driftcorr`	Compute correction for drift.
`driftvel`	Compute drift speed and direction.
`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 9-20 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 9-21 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.
polyxpoly	Polygon intersections.
rhxrh	Find intersection point between two rhumb lines.
scxsc	Calculate intersection points for pairs of small circles.

Table 9-22 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 9-23 Data Precision

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

Matrix Map Data Operations

Table 9-24 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 9-25 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.
gradientm	Gradient, aspect and slope of regular or general matrix map.
imbedm	Encode vector data points into regular matrix map.
limitm	Latitude and longitude limits for regular matrix map.
los2	Visibility between two points on a Digital Elevation Map.
mapprofile	Matrix map values along a path
maptrims	Trim matrix map to a region
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.
viewshed	Area visible from a point on a Digital Elevation map.

Table 9-26 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 9-27 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.
worldhi	High-resolution world vector data.
worldlo	Low-resolution world vector data.
worldmtx	World matrix map.

Table 9-28 Interface

Function	Description
`grepfields`	Identify matching fields in fixed record length files.
`nanclip`	Clip vector data with `NaN`s 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 9-29 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 9-30 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 9-31 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.
zero22pi	Truncate angles into [0 360] range.

Table 9-32 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 9-33 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 9-34 String Manipulation

Function	Description
`angl2str`	Angle conversion to string.
`dist2str`	Distance conversion to string.
`str2angle`	String conversion to angle.
`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.

Alphabetical List of Functions

This section provides reference entries with detailed descriptions of all the Mapping Toolbox functions, listed alphabetically.

Following is a list of headings used in this section. Not every function will have all this material, but the information that is included will be ordered as shown.

Purpose

Syntax

Background

Description

Axes Definition

Examples

Object Properties

Limitations

Remarks

See Also

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

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 $10^{\text{digits}-1}$ 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

`str2angle`	Convert strings containing DMS formatted angles to numbers
`angledim`	Convert angle units
`dist2str`	Convert distance values to strings
`time2str`	Convert time values to strings

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

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.

areaint

Purpose

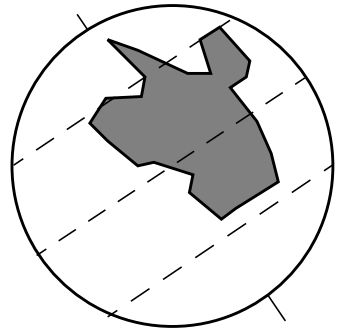
Calculate spherical surface area enclosed by a polygon

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 `NaNs`. 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

areamat

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

Purpose

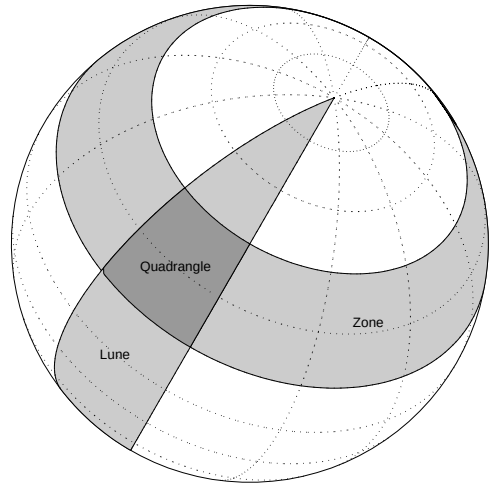
Compute area of a latitude-longitude quadrangle

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

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

axes2ecc

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

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` reference page in Chapter 11, "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 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 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 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 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

axesscale

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.

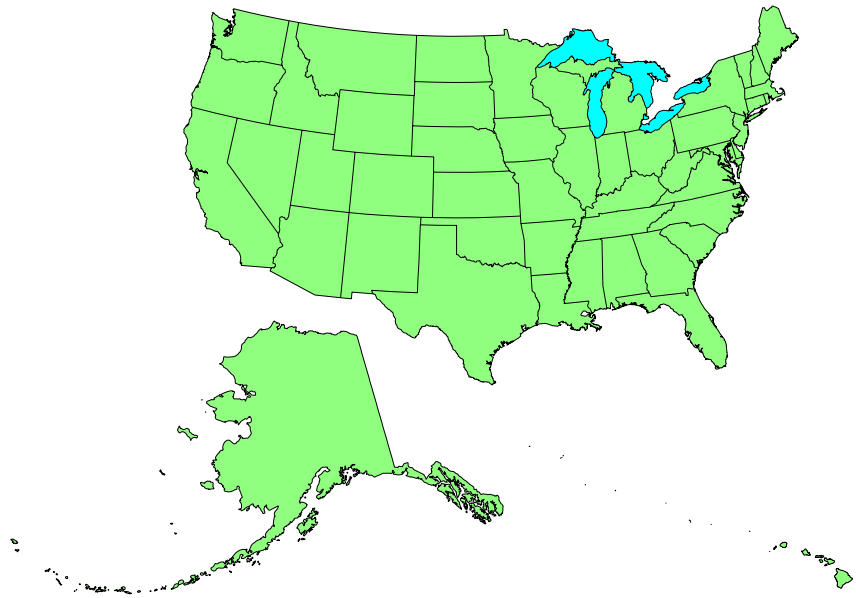
Examples

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

azimuth

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 *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
`elevation`	Elevation angle to a point

`reckon`	New point from an azimuth and

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

Purpose

Compute buffer zones for vector data.

Syntax

```
[latb,lonb] = bufferm(lat,lon,dist,direction)
[latb,lonb] = bufferm(lat,lon,dist,direction,npts)
[latb,lonb] = bufferm(lat,lon,dist,direction,npts,outputformat)
```

Description

[latb,lonb] = bufferm(lat,lon,dist,*direction*) computes the buffer zone around a polygon. A buffer zone for a closed polygon is defined as the locus of points that are a certain distance in or out of the polygon. A buffer zone for an open polygon is the locus of points a certain distance out from the polygon. The polygon is specified as vectors of latitude and longitude in units of degrees. The distance is a scalar specified in degrees of arc along the surface. Valid direction strings are 'in' and 'out'. The result is returned as NaN-clipped vectors in units of degrees.

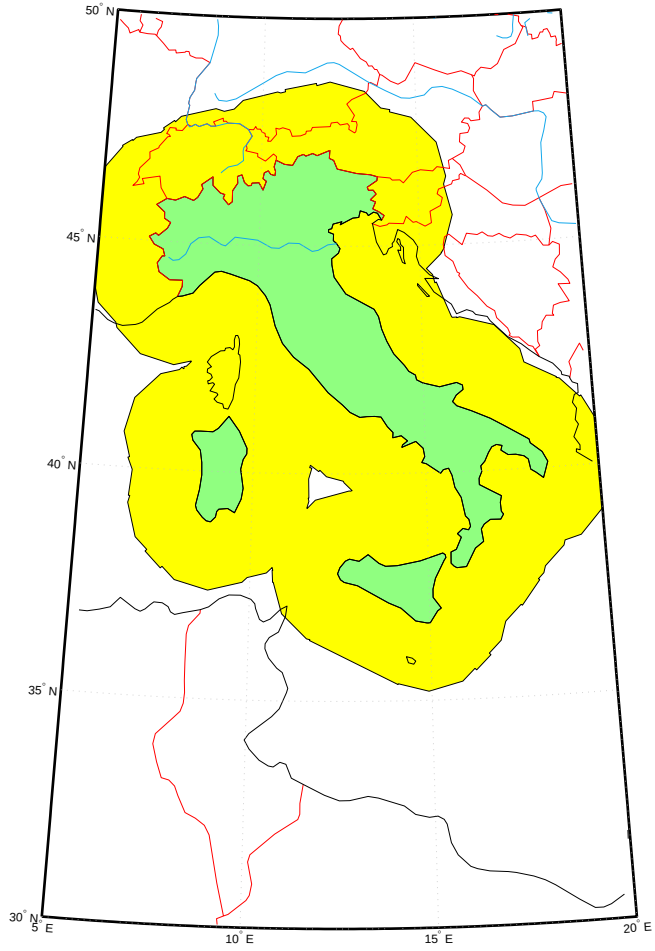
[latb,lonb] = bufferm(lat,lon,dist,*direction*,npts) controls the number of points used to construct circles about the vertices of the polygon. A larger number of points produces smoother buffers, but requires more time. If omitted, 13 points per circle are used.

[latb,lonb] = bufferm(lat,lon,dist,*direction*,npts,*outputformat*) controls the format of the returned buffer zones. Outputformat 'vector' returns NaN- clipped vectors. Outputformat 'cutvector' returns NaN-clipped vectors with cuts connecting holes to the exterior of the polygon. Outputformat 'cell' returns cell arrays in which each element of the cell array is a separate polygon. Each polygon may consist of an outer contour followed by holes separated with NaNs.

Examples

Extract the coastline for Italy from the low resolution database and reduce the polygon. Construct construct a buffer polygon 1.5 degrees of arc along the earth's surface out from the italian coast. Display the result on a basemap of the italian peninsula along with the original polygon.

```
[lat,lon]=extractm(worldlo('POpatch'),'italy');
[lat,lon]=reducem(lat,lon);

[latb,lonb] = bufferm(lat,lon,1.5,'out')

worldmap('lo','italy','lineonly')
hp = patchesm(latb,lonb,'y');              zdatam(hp,-2)
set(hp(2),'facecolor','w');                zdatam(hp(2),1) % hole
hp2=displaym(worldlo('POpatch'),'italy'); zdatam(hp2,-1)
```



See Also `polybool` Perform polygon boolean operations

Purpose

Camera position in geographic coordinates

Syntax

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

Description

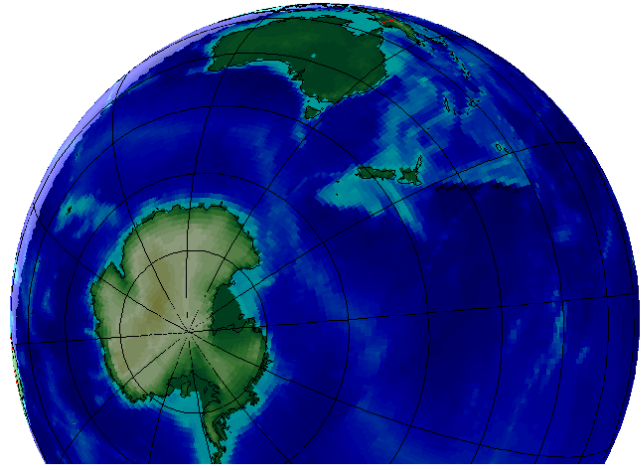
camposm(lat,long,alt) sets the axes `CameraPosition` property of the current map axes to the position specified in geographic coordinates. The inputs `lat` and `long` are assumed to be in the angle units of the current map axes.

[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 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

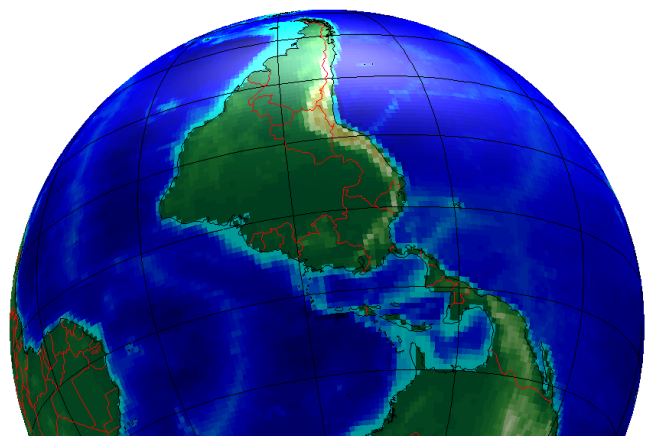
camtargm(lat,long,alt) sets the axes `CameraTarget` property of the current map axes to the position specified in geographic coordinates. The inputs `lat` and `long` are assumed to be in the angle units of the current map axes.

[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

`camposm`	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

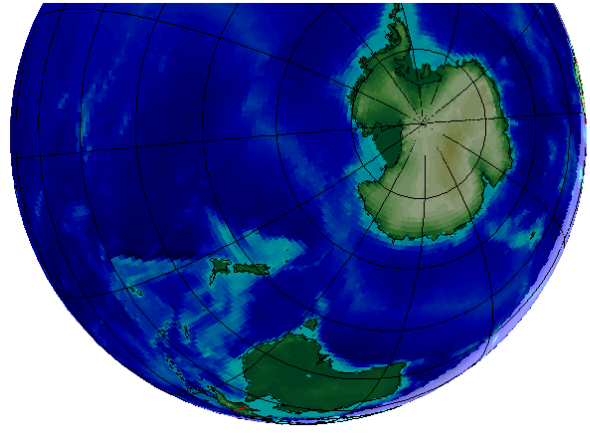
camtargm(lat,long) sets the axes `CameraUpVector` property of the current map axes to the position specified in geographic coordinates. The inputs `lat` and `long` are assumed to be in the angle units of the current map axes.

[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`.

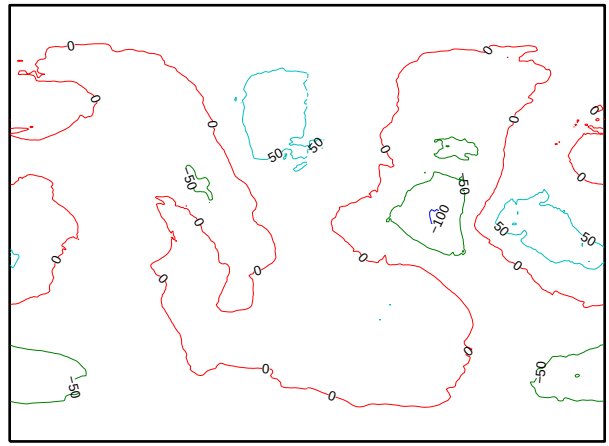
`h1 = clabelm(c,h,v)` creates in line labels only for those levels specified in the vector `v`.

`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

```
axesm miller
framem
tightmap
[c,h] = contourm(geoid,geoidlegend,-100:50:80);
clabelm(c,h)
```


0
0
0
0
50
50
50
50
0
0
0

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`.

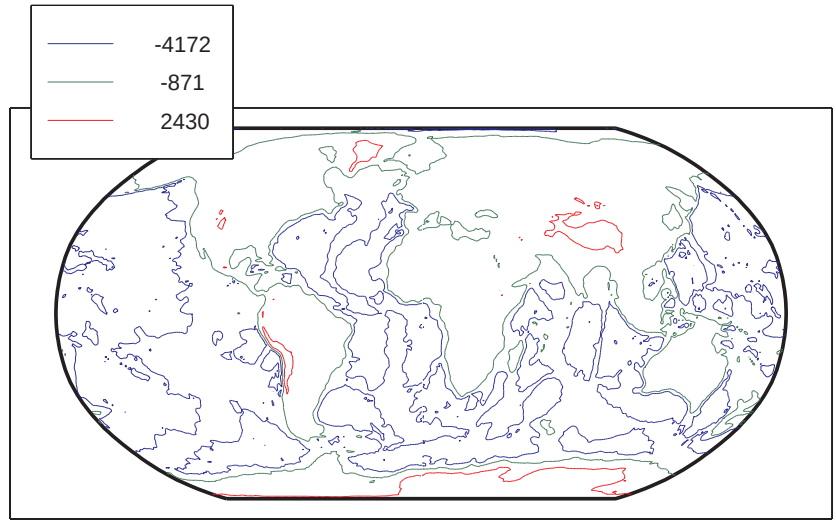
clegendm(cs,h,pos) allows you to specify the position of the legend in the display. The input `pos` can be any of the following integers, with the indicated result:

`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
```

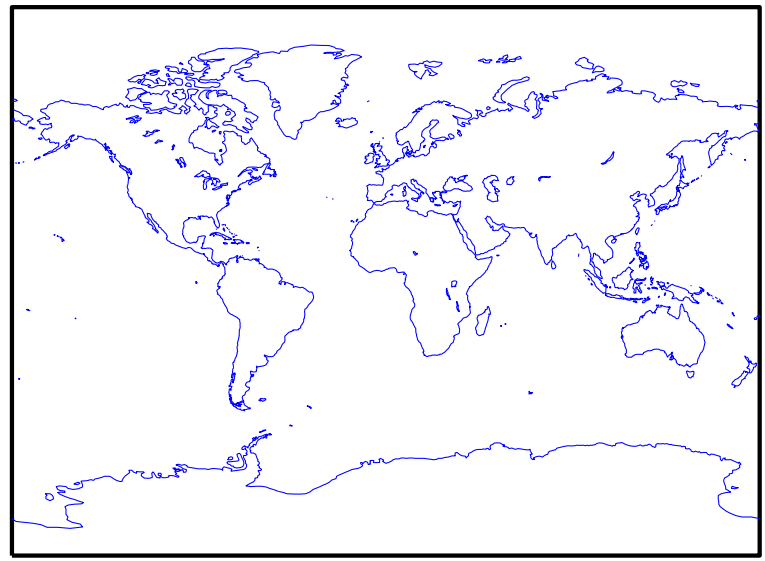
Description

`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.

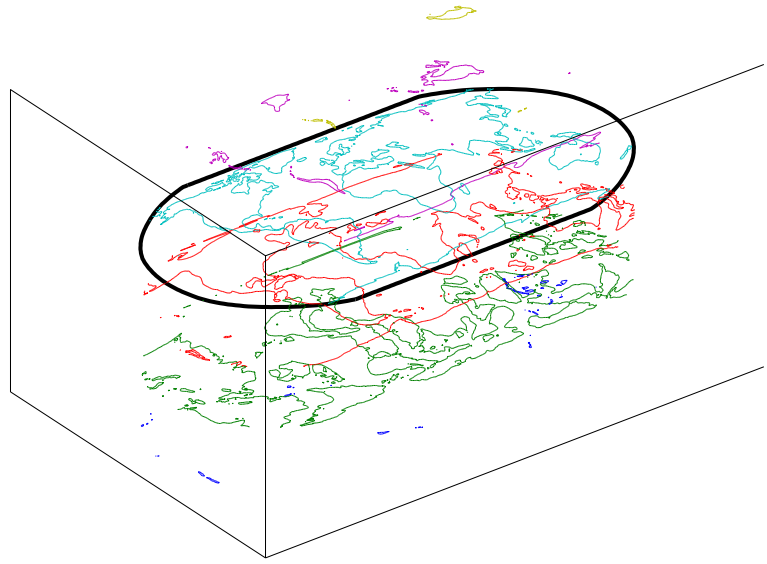
[c,h] = contor3m(...) returns the contour matrix and the handles to the contour lines drawn.

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.

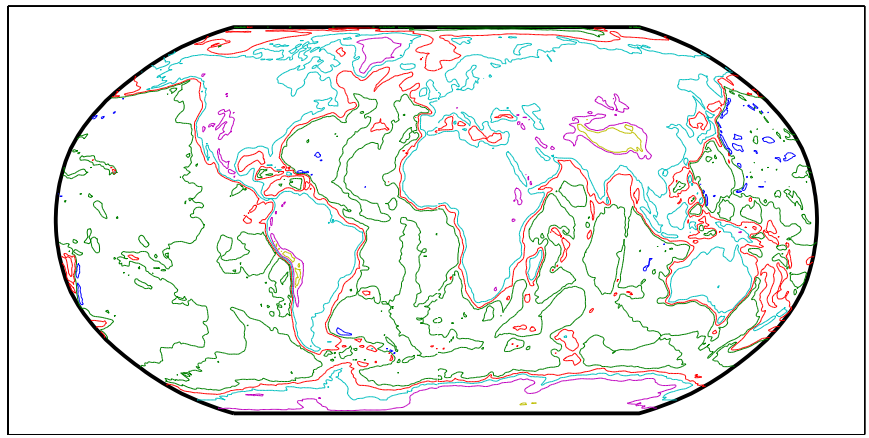
[c,h] = contorm(...) returns the contour matrix and the handles to the contour lines drawn.

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

Contour colormap and colorbar for surfaces

Syntax

```
contourcmap(cdelta,cmap)
contourcmap(cdelta,cmap,property,value,...)
hcb = contourcmap(...)
```

Description

contourcmap(cdelta,*cmap*) creates a contour colormap for the current axes. A contour colormap is a colormap with color changes aligned to the color data. If cdelta is a scalar, contours are generated at multiples of cdelta. If cdelta is a vector of evenly spaced values, contours are generated at those values. The string input cmap is the name of the colormap function used in the surface. Valid entries for cmap include 'pink', 'hsv', 'jet', or any similar colormap function.

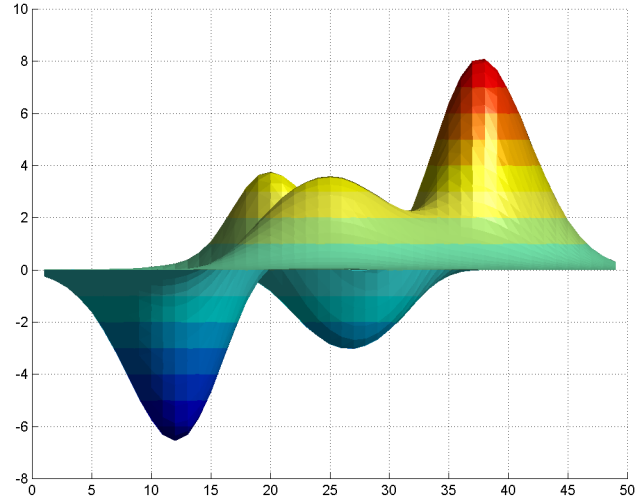
contourcmap(cdelta,*cmap*,*property*,value,...) allows you to add a colorbar and control the colorbar's properties. You turn the colorbar on with the property-value pair 'Colorbar' and 'on'. The location of the colorbar is controlled by the 'Location' property. Valid entries for Location are 'vertical' (the default) or 'horizontal'. Properties 'TitleString', 'XLabelString', 'YLabelString' and 'ZLabelString' set the respective strings. Property 'ColorAlignment' controls whether the colorbar labels are centered on the color bands or the color breaks. Valid values for ColorAlignment are 'center' or 'ends'. Property 'SourceObject' controls which object is used to determine the color limits for the colormap. The SourceObject value is the handle of a currently displayed object. If omitted, gca is used. Other valid property-value pairs are any properties and values that can be applied to the title and labels of the colorbar axes.

hcb = contourcmap(...) returns a handle to the colorbar.

Example

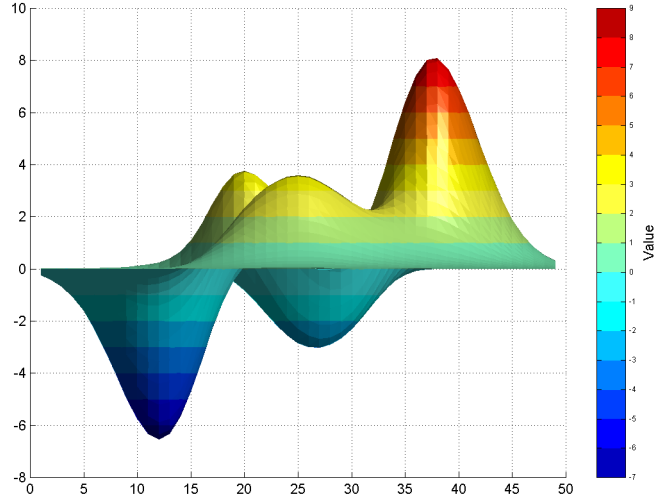
Create a colormap and set color limits to make the color changes occur at multipled of one for the peaks surface.

```
surf(peaks)
contourcmap(1,'jet')
view(90,0)
shading interp
camlight
```



Add a colorbar, controlling the labels and font properties.

```
h =
contourcmap(1,'jet','colorbar','on','YLabelString','Value','Fon
tSize',6)
```



Delete the colorbar.

```
delete(h)
```

See Also

`contourfm`	Filled contour map.
`lcolorbar`	Labeled color bar.
`demcmap`	Digital elevation data color map.

contourfm

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.

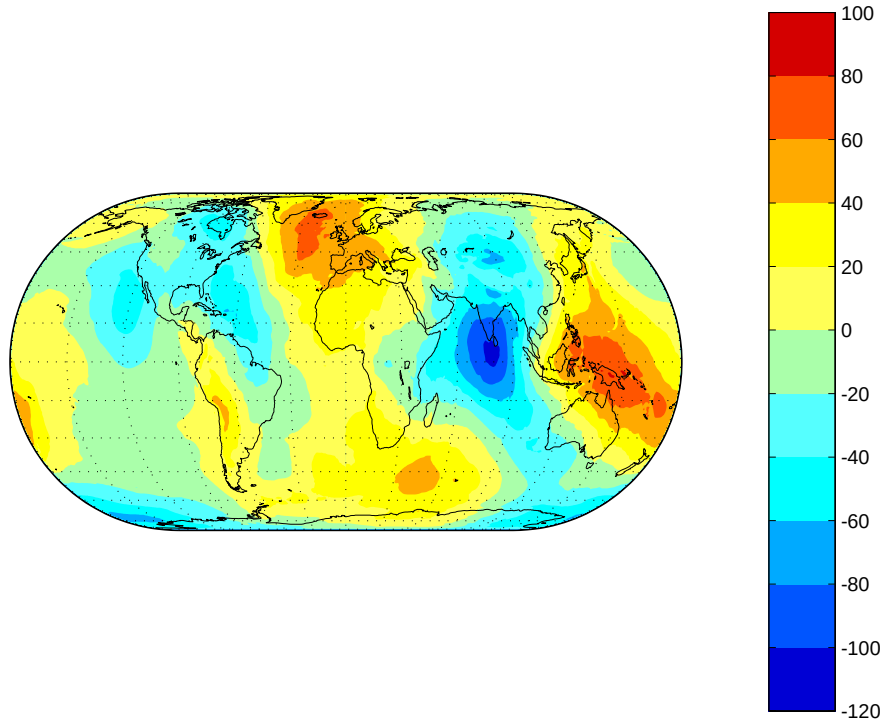
[c,h] = contourfm(...) returns the contour matrix and the handles to the contour lines drawn.

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

Examples

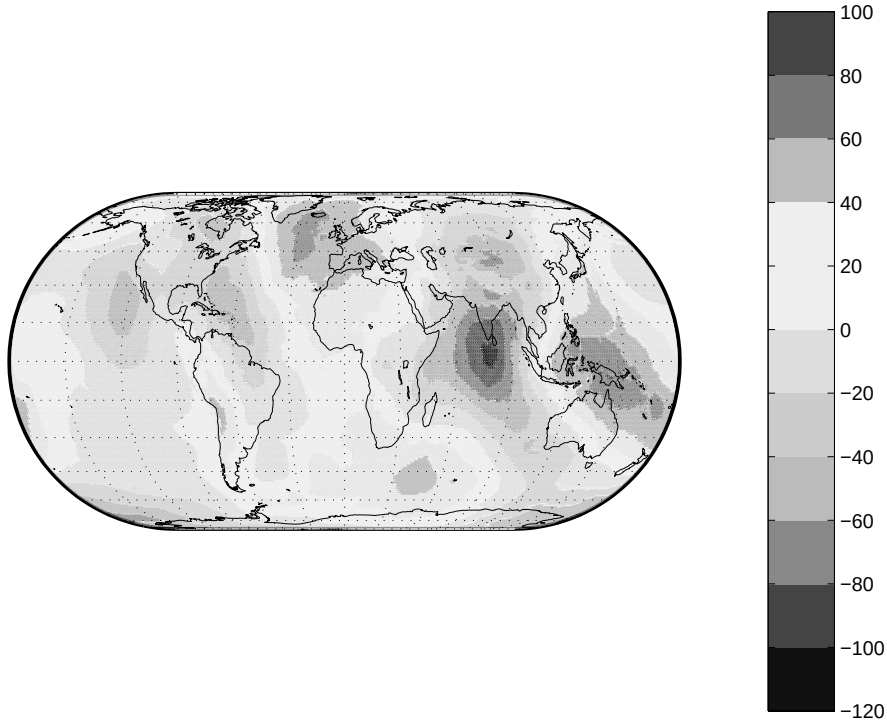
Plot the Earth's geoid with filled contours. The data is in meters.

```
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`.

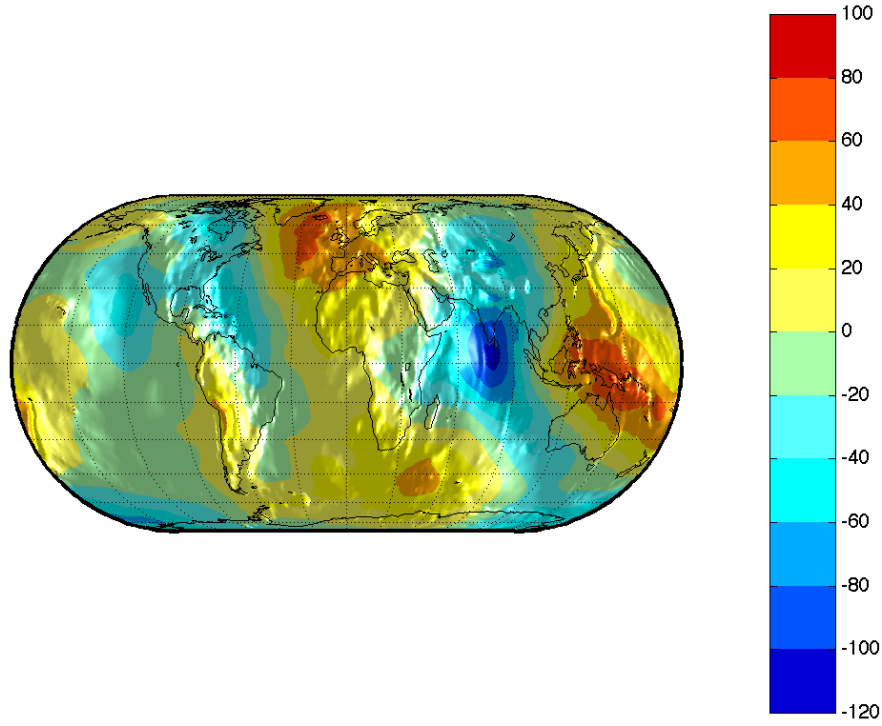
```
figure
axesm eckert4
framem;gridm
plotm(coast,'k')

meshm(geoid,geoidlegend,size(geoid),'Facecolor','interp')
contourcmap(20,’jet’);colorbar
```



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

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.

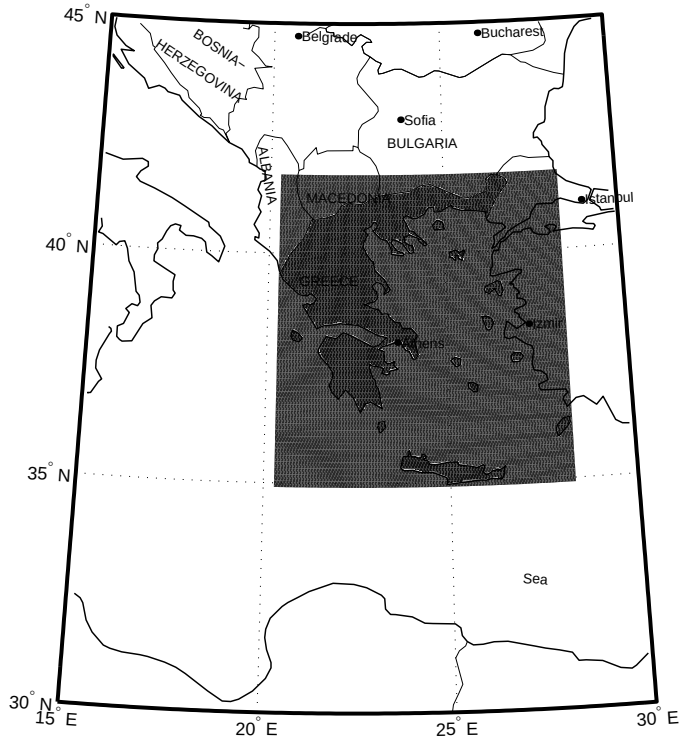
`[map,maplegend] = country2mtx(countryname,scalelatlim,lonlim)` uses the two-element vector latitude and longitude limits to define the extent of the map. If omitted, the limits are computed automatically.

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

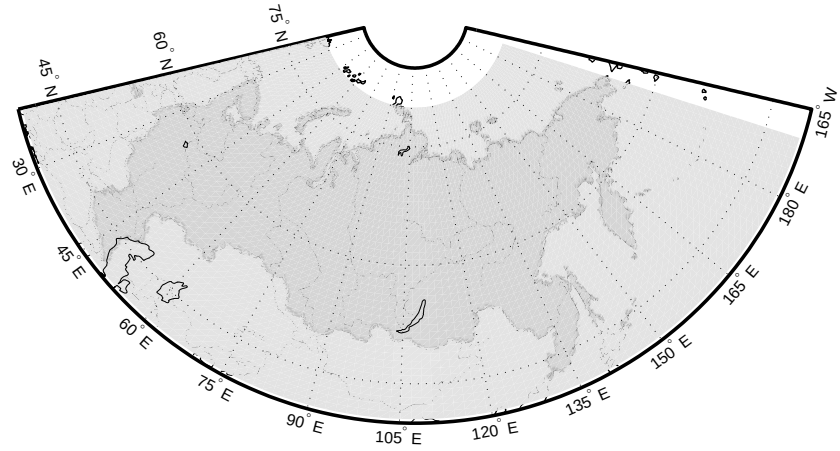
Examples

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

country2mtx



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



See Also

`vec2mtx`	Regular matrix map from vector data
`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

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'`.

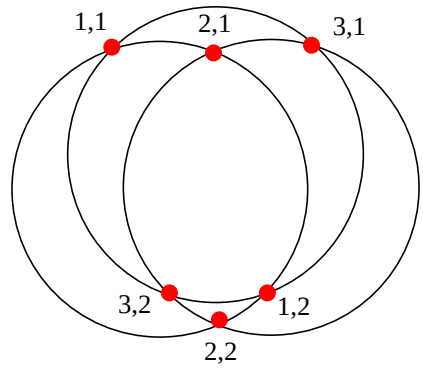
mat = crossfix(...) returns the output in a two- or four-column matrix `mat`.

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 polyxpoly	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.

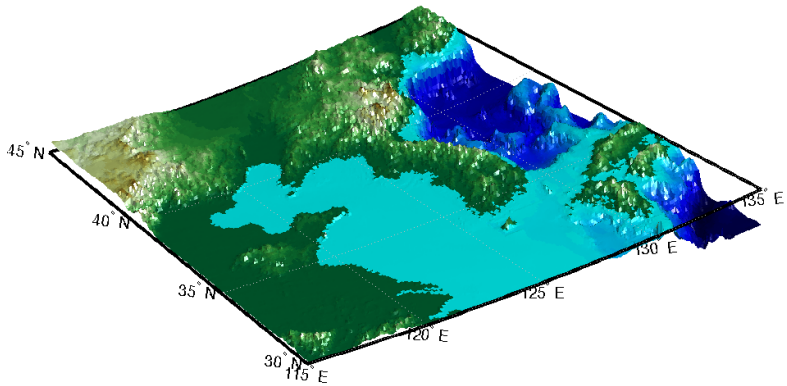
daspectm(*zunits*,vfac,lat,long,az,*gunits*,radius) uses the last input to determine the radius of the sphere. If `radius` is a string, then it is evaluated as an `almanac` body to determine the spherical radius. If numerical, it is the radius of the desired sphere in `zunits`. If omitted, the default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

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
mat2dms	Convert from separated matrices input to *dms*

deg2km, deg2nm, deg2sm

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

deg2rad

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 2 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

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.

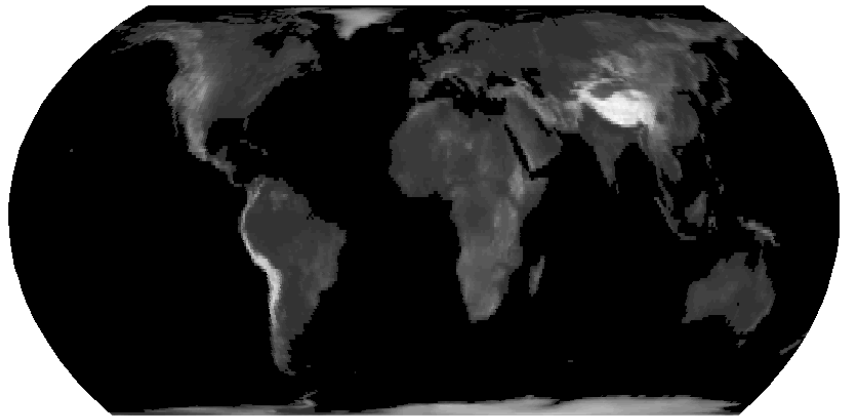
demcmap(color,map,spec) uses the *color* string to define a colormap. If the string is set to `'size'`, spec is the length of the colormap. If it is set to `'inc'`, spec is the size of the altitude range assigned to each color. If omitted, *color* is `'size'` by default.

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

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

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.

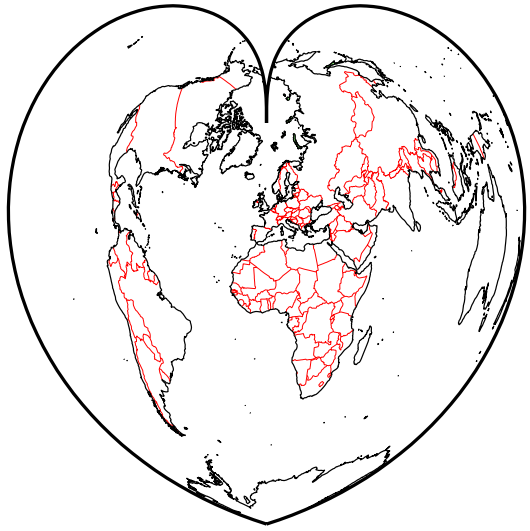
[lat,lon] = displaym(mstruct,objects,'exact') requires an exact match to extract data.

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

Examples

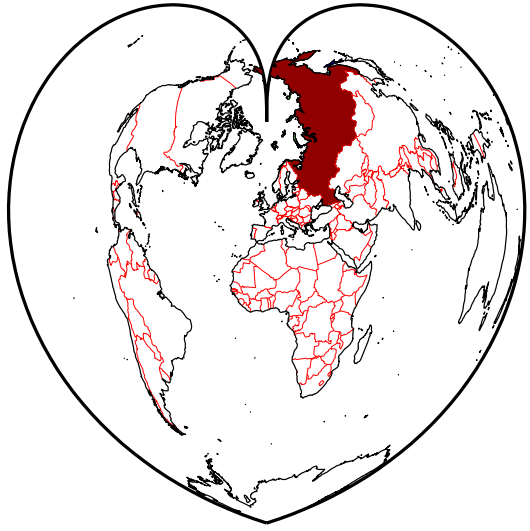
Display political and coast lines contained in structure format:

```
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

dist2str

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 $10^{\text{digits}-1}$ 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

distance

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

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

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

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

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

dms2mat

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

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*

dreckon

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 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

driftcorr

Purpose

Heading to correct for wind or current drift.

Syntax

```
[heading,groundspeed,windcorrangle] =
   driftcorr(course,airspeed,windfrom,windspeed)
```

Description

heading = driftcorr(course,airspeed,windfrom,windspeed) computes the heading which corrects for drift due to wind (for aircraft) or current (for watercraft). Course is the desired direction of movement (in degrees), speed is the speed of the vehicle relative to the moving air or water mass, windfrom is the direction facing into the wind or current (in degrees), and windspeed is the speed of the wind or current (in the same units as airspeed).

[heading,groundspeed,windcorrangle] = driftcorr(...) also returns the groundspeed and wind correction angle. The wind correction angle is positive to the right, and negative to the left.

Example

An aircraft with a cruising at a speed of 160 knots plans to fly to an airport due north of its current position. If the wind is blowing from 310 degrees at 45 knots, what heading should the aircraft fly to remain on course?

```
course=0; airspeed=160;windfrom=310; windspeed = 45;
[heading,groundspeed,windcorrangle] =
driftcorr(course,airspeed,windfrom,windspeed)

heading =

       347.56


groundspeed =

       127.32


windcorrangle =

      -12.442
```

The required heading is 348 degrees, which amounts to a wind correction angle of 12 degrees to the left of course. The headwind component reduces the aircraft's groundspeed to 127 knots.

See Also

`driftvel`	Wind or current from heading, course, and speeds

driftvel

Purpose

Wind or current from heading, course and speeds.

Syntax

```
[windfrom,windspeed] =
   driftvel(course,groundspeed,heading,airspeed)
```

Description

[windfrom,windspeed] = driftvel(course,groundspeed,heading,airspeed) computes the wind (for aircraft) or current (for watercraft) from course, heading and speeds. Course and groundspeed are the direction and speed of movement relative to the ground (in degrees), Heading is the direction in which the vehicle is steered, and airspeed is the speed of the vehicle relative to the airmass or water. The output windfrom is the direction facing into the wind or current (in degrees), and windspeed is the speed of the wind or current (in the same units as airspeed and groundspeed).

Example

An aircraft is cruising at a true airspeed of 160 knots and a heading of 10 degrees. From the Global Positioning System (GPS) reciever, the pilot determines that the aircraft is progressing over the ground at 155 knots in a northerly direction. What is the wind aloft?

```
course = 0; groundspeed = 155; heading = 10; airspeed = 160;
[windfrom,windspeed] =
driftvel(course,groundspeed,heading,airspeed)

windfrom =

       84.717


windspeed =

       27.902
```

The wind is blowing from the right, 085 degrees at 28 knots.

See Also

`driftcorr` Heading to correct for wind or current drift

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

elevation

Purpose

Calculates elevation angle between points on a geoid.

Syntax

```
elevang = elevation(lat1,lon1,alt1,lat2,lon2,alt2)
[elevang,slantrange] = elevation(...)
elevang,slantrange] =
   elevation(lat1,lon1,alt1,lat2,lon2,alt2,angleunits)
elevang,slantrange] =
   elevation(lat1,lon1,alt1,lat2,lon2,alt2,angleunits,zunits)
elevang,slantrange] =
   elevation(lat1,lon1,alt1,lat2,lon2,alt2,angleunits,ellipsoid)
```

Description

elevang = elevation(lat1,lon1,alt1,lat2,lon2,alt2) computes the elevation angle of point 2 as viewed from point 1. The elevation angle is the angle of the line of sight above the local horizontal at point 1. Points are specified as latitude, longitude and altitude above the surface. Angles are in units of degrees, altitudes are in meters. The figure of the earth is assumed to the default ellipsoid. Inputs can be vectors of points. Elevations from a point to a line can be computed by providing a scalar point 1 and vector point 2.

[elevang,slantrange] = elevation(...) also returns the slant range between the points in units of meters.

[elevang,slantrange] = elevation(lat1,lon1,alt1,lat2,lon2,alt2,*angleunits*) uses the inputs angleunits to define the angle units of the inputs and outputs. If omitted, 'degrees' are assumed.

[elevang,slantrange] = elevation(lat1,lon1,alt1,lat2,lon2,alt2,*angleunits*,*zunits*) uses the inputs zunits to define the units of the altitude inputs and slant range outputs. If omitted, 'meters' are assumed. All distance units recognized by `distdim` are valid inputs.

[elevang,slantrange] = elevation(lat1,lon1,alt1,lat2,lon2,alt2,*angleunits*,ellipsoid) computes the elevation and slant range on the ellipsoid defined by the input ellipsoid. The ellipsoid vector is of the form [semimajor axes, eccentricity]. If omitted, the default ellipsoid for the earth is assumed. The altitudes are input and the slant range is returned in the units of ellipsoid(1).

Example

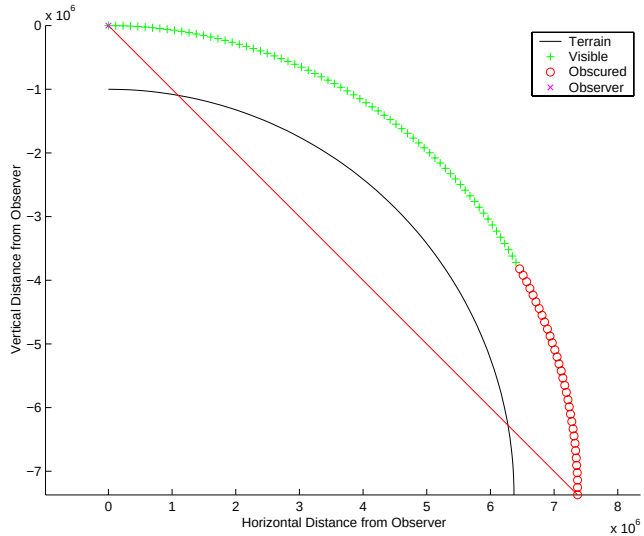
What is the elevation angle of a point 90 degrees distant when both the observer and target are 1000 km altitude above the Earth?

```
lat1 = 0; lon1 = 0; alt1 = 1000*1000;
lat2 = 0; lon2 = 90;alt2 = 1000*1000;
```

```
elevang = elevation(lat1,lon1,alt1,lat2,lon2,alt2)

elevang =

   -45
```

Visually check the result using the `los2` line of sight command. Construct a matrix map of zeros to represent the Earth's surface. The `los2` command with no output arguments creates a figure displaying the geometry.

```
map = zeros(180,360); maplegend = [1 90 -180];
los2(map,maplegend,lat1,lon1,lat2,lon2,alt1,alt1);
```



See Also

`azimuth` Calculates azimuth between points on a geoid

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 .

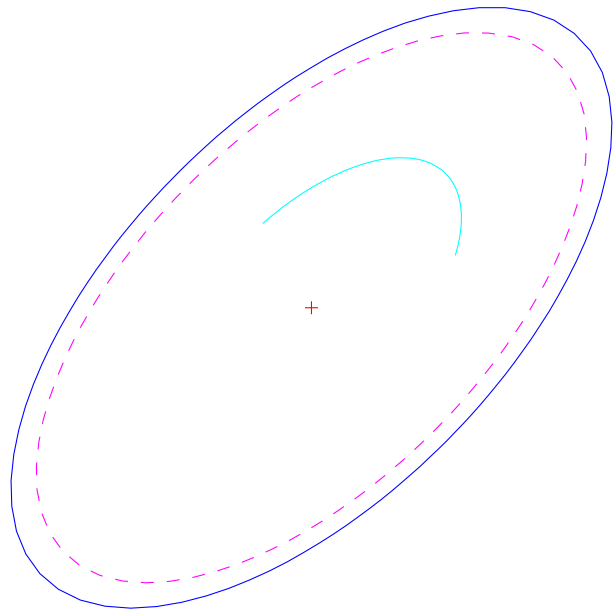
```
axesm mercator
ecc = axes2ecc(10,5);
plotm(0,0,'r+')
[elat,elon] = ellipse1(0,0,[10 ecc],45);
plotm(elat,elon)
```

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

encodem

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

epsm

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

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

extractm

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

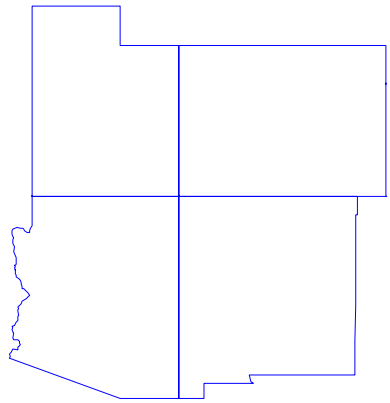
Extract the District of Columbia from the low resolution U.S. vector data:

```
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 them:

```
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,...)
```

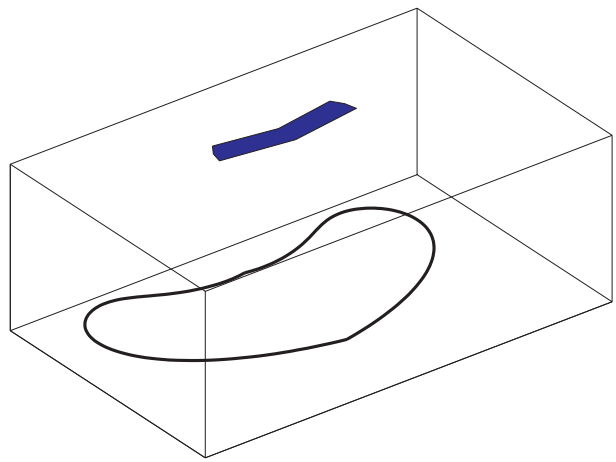
Description

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,...)
```

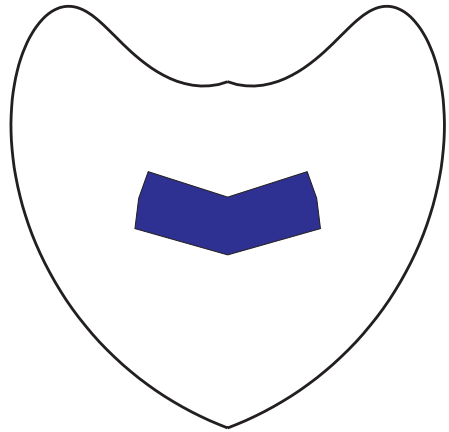
Description

`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

flatearthpoly

Purpose

Inserts points along the dateline to the pole.

Syntax

```
[lat2,lon2] = fleatearthpoly(lat,lon)
[lat2,lon2] = fleatearthpoly(lat,lon,origin)
```

Description

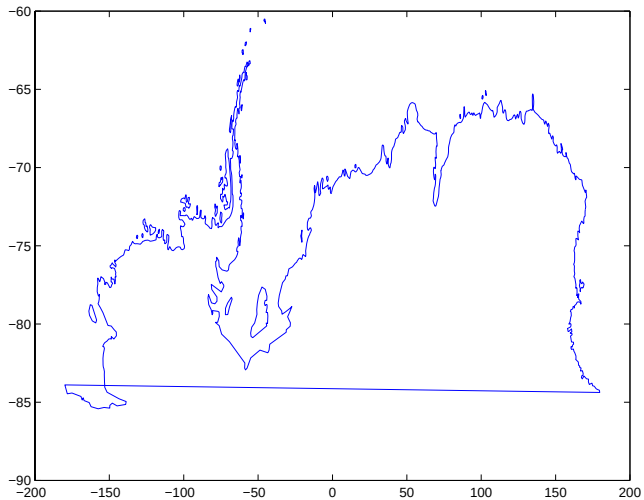
[lat2,lon2] = fleatearthpoly(lat,lon) inserts points in the input latitude and longitude vectors at +/- 180 longitude and to the poles. The resulting vectors look like the result of `patchm` on a cylindrical projection and do not encompass the poles. Inputs and outputs are in degrees.

[lat2,lon2] = fleatearthpoly(lat,lon,origin) centers the polygon on the provided origin. The origin is a scalar longitude or a three element vector containing latitude, longitude and orientation in units of degrees.

Example

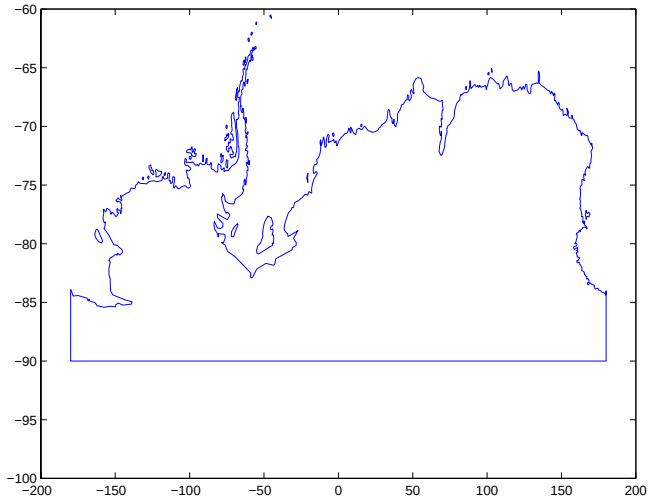
Vector data for geographic objects that encompass a pole will inevitably encounter or cross the dateline. While the Mapping Toolbox properly displays such polygons, they can cause problems for commands like the polygon intersection and boolean operations that work with cartesian coordinates. When these polygons are treated as cartesian coordinates the dateline crossing will result in a spurious line segment, and the polygon displayed as a patch will not have the interior filled correctly.

```
[lat,long]=extractm(worldlo('POpatch'),'Antarctica');
plot(long,lat)
```



The polygons can be reformatted more appropriately for cartesian coordinates using the `flatearthpoly` command. The result resembles a map display on a cylindrical projection. The polygon meets the dateline, drops down to the pole, sweeps across the longitudes at the pole, and follows the the dateline up to the other side of the dateline crossing.

```
[lat2,lon2]=flatearthpoly(lat,long);
plot(lon2,lat2)
ylim([-100 -60])
```



See Also

`polybool`	Polygon boolean operations
`polyxpoly`	Polygon intersections
`mfwdtran`	Process the map forward transformations

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

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

gcwaypts

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.

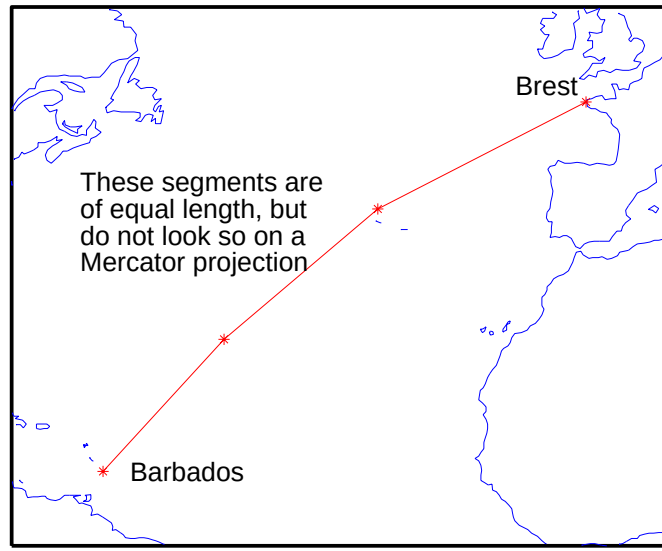
`pts = gcwaypts(lat1,lon1,lat2,lon2...)` packs the outputs, which are otherwise two-column vectors, into a two-column matrix of the form `[latitude longitude]`. This format for successive waypoints along a navigational track is called *navigational track format* in this guide. See the `navigational track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

gradientm

Purpose

Numerical gradient, slope and aspect of matrix maps.

Syntax

```
[aspect,slope,gradN,gradE] = gradientm(map,maplegend)
[...] = gradientm(lat,lon,map)
[...] = gradientm(map,maplegend,ellipsoid)
[...] = gradientm(lat,lon,map,ellipsoid)
[...] = gradientm(lat,lon,map,ellipsoid,units)
```

Description

[aspect,slope,gradN,gradE] = gradientm(map,maplegend) computes the slope, aspect and north and east components of the gradient for a regular matrix map. If the map contains elevations in meters, the resulting aspect and slope are in units of degrees clockwise from north and up from the horizontal. The north and east gradient components are the change in the map variable per meter of distance in the north and east directions. The computation uses finite differences for the map variable on the default earth ellipsoid.

[aspect,slope,gradN,gradE] = gradientm(lat,lon,map) does the computation for a general matrix map.

[aspect,slope,gradN,gradE] = gradientm(map,maplegend,ellipsoid) and [aspect,slope,gradN,gradE] = gradientm(lat,lon,map,ellipsoid) use the provided ellipsoid definition. The ellipsoid vector is of the form [semimajor axes, eccentricity]. If the map contains elevations in the same units as ellipsoid(1), the slope and aspect are in units of degrees. This calling form is most useful for computations on bodies other than the earth.

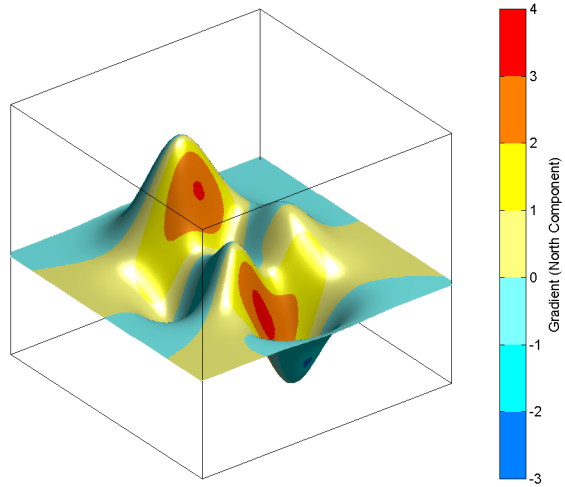
[aspect,slope,gradN,gradE] = gradientm(lat,lon,map,ellipsoid,*units*) specifies the angle units of the latitude and longitude inputs. If omitted, 'degrees' are assumed. For elevation maps in the same units as ellipsoid(1), the resulting slope and aspect are in the specified units. The components of the gradient are the change in the map variable per unit of ellipsoid(1).

Example

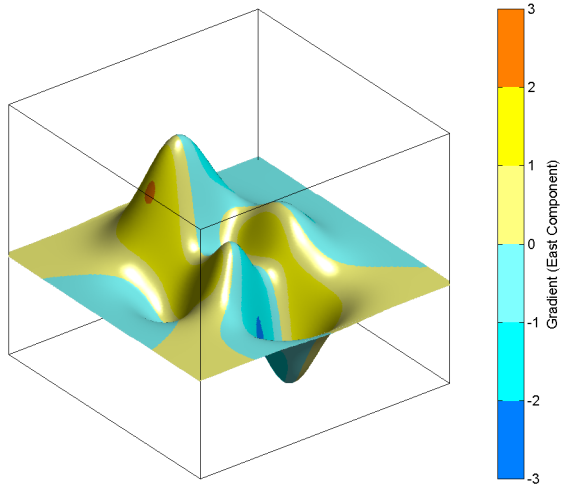
Compute the gradient, slope, and aspect for a regular matrix map based on the MATLAB peaks matrix. Show the `gradientm` outputs as colors on a three-dimensional surfaces of elevation. The surfaces are shown with no vertical exaggeration.

```
map = 500*peaks(100);
maplegend = [ 1000 0 0];
```

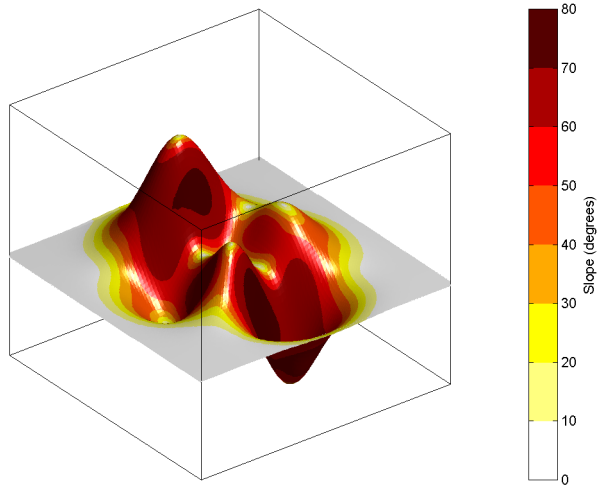
```
[aspect,slope,gradN,gradE] = gradientm(map,maplegend);

figure
worldmap(gradN,maplegend,'none')
meshm(gradN,maplegend,size(map),map)
hcb =
contourcmap(1,'jet','colorbar','on','ylabelstring','Gradient
(North Component)');
view(3)
daspectm meters
camlight
shading interp; lighting phong
framem off
```



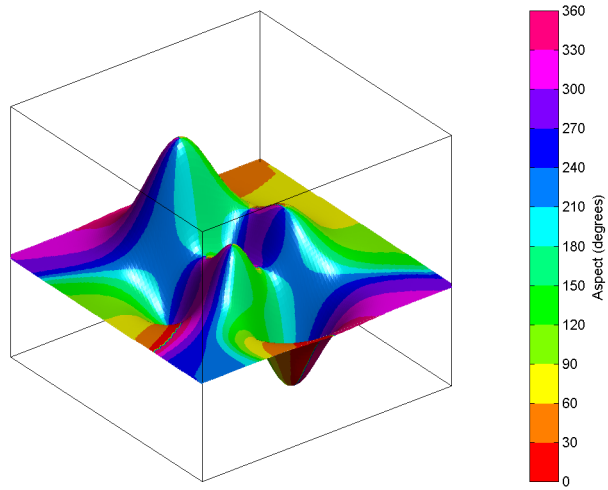

```
clmo surface
meshm(gradE,maplegend,size(map),map)
set(get(hcb,'ylabel'),'String','Gradient (East Component)');
shading interp; lighting phong
```




```
clmo surface
meshm(slope,maplegend,size(map),map)
contourcmap(10,'hot','colorbar','on','ylabelstring','Sloep
(degrees)')
colormap(flipud(colormap))
shading interp; lighting phong
```

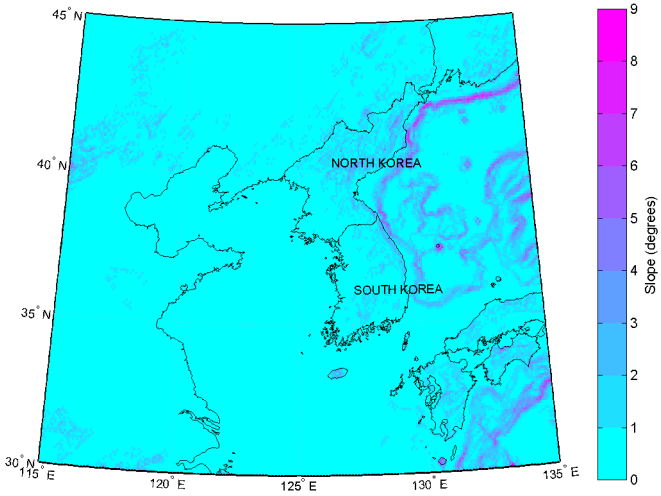


```
clmo surface
meshm(aspect,maplegend,size(map),map)
contourcmap(30,'hsv','colorbar','on','ylabelstring','Aspect
(degrees)')
shading interp; lighting phong
```




```
load korea
[aspect,slope,gradN,gradE] = gradientm(map,maplegend);
worldmap(slope,maplegend);
contourcmap(1,'cool','colorbar','on','ylabelstring','Slope
(degrees)')
hidem(gca)
```

gradientm



Remarks

Coarse digital elevation models may considerably underestimate the local slope. For the above map, the elevation points are separated by about 10 kilometers. The terrain between two adjacent points is modelled as a linear variation, while actual terrain may vary much more abruptly over such a distance.

See Also

gradient	Approximate gradient
viewshed	Visible areas from a point on a digital elevation map

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

gridm

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

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.

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.

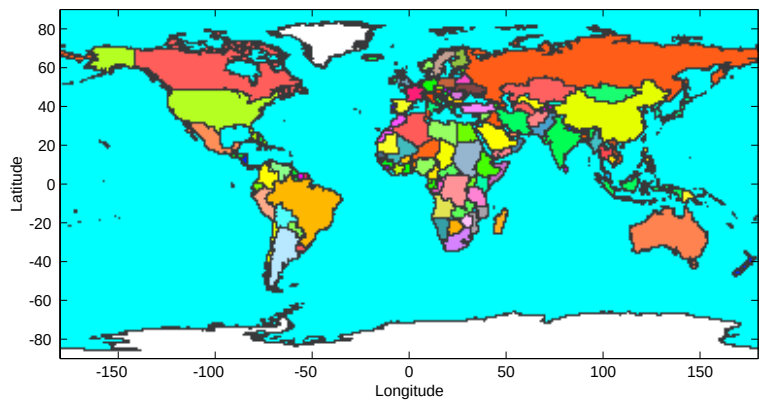
Note that `imagem` uses standard MATLAB axes, *not* map axes.

`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

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*)

interpm

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

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

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

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

ismapped

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

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` `par2geod`	Other auxiliary latitude functions

km2deg, km2nm, km2rad, km2sm

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

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

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

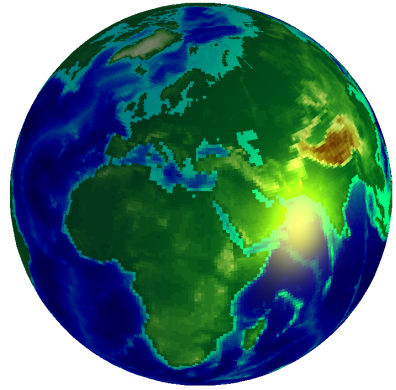
h = lightm(lat,lon) projects a light object at the coordinates `lat` and `lon`. The handle, `h`, of the object can be returned.

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*)

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

linem

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`.

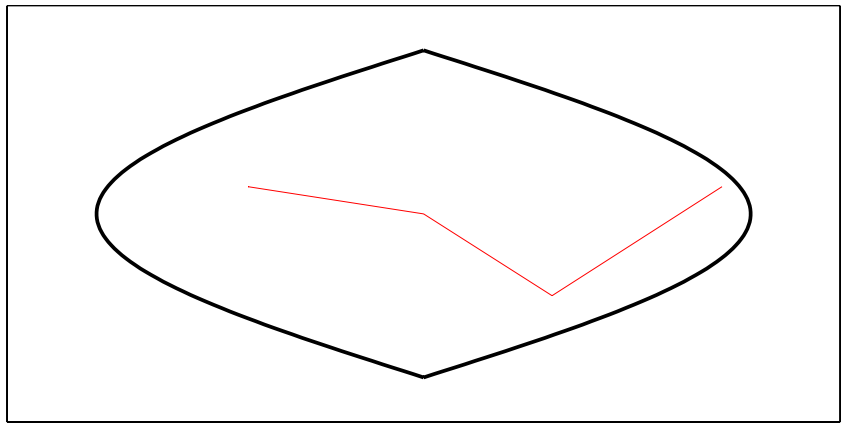
h = linem(lat,lon,*linetype*) allows the specification of the line style, where *linetype* is any string recognized by the MATLAB `line` command.

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

los2

Purpose

Line of sight visibility between two points in terrain

Syntax

```
los2
vis = los2(map,maplegend,lat1,lon1,lat2,lon2)
vis = los2(map,maplegend,lat1,lon1,lat2,lon2,alt1)
vis = los2(map,maplegend,lat1,lon1,lat2,lon2,alt1,alt2)
vis = los2(map,maplegend,lat1,lon1,lat2,lon2,alt1,alt2,alt1opt)
vis =
   los2(map,maplegend,lat1,lon1,lat2,lon2,alt1,alt2,alt1opt,alt2op
   t)
vis =
   los2(map,maplegend,lat1,lon1,lat2,lon2,alt1,alt2,alt1opt,alt2op
   t,actualradius)
vis =
   los2(map,maplegend,lat1,lon1,lat2,lon2,alt1,alt2,alt1opt,alt2op
   t,actualradius,effectiveradius)
[vis,visprofile,dist,z,lattrk,lontrk] = los2(...)
los2(...)
```

Description

los2 computes the mutual visibility between two points on a displayed digital elevation map. los2 uses the current object if it is a regular matrix map, or the first regular matrix map found on the current axes. The map's zdata is used for the profile. The color data is used in the absence of data in z. The two points are selected by clicking on the map. The result is displayed in a new figure. Markers indicate visible and obscured points along the profile. The profile is shown in a cartesian coordinate system with the origin at the observer's location. The displayed z coordinate accounts for the elevation of the terrain and the curvature of the body.

vis = los2(map,maplegend,lat1,lon1,lat2,lon2) computes the mutual visibility between pairs of points on a digital elevation map. The elevations are provided as a regular matrix map containing elevations in units of meters. The two points are provided as vectors of latitudes and longitudes in units of degrees. The resulting logical variable vis is equal to one when the two points are visible to each other, and zero when the line of sight is obscured by terrain. If any of the input arguments are empty, los2 attempts to gather the data from the current axes. With one or more output arguments, no figures are created and only the data is returned.

vis = los2(map,maplegend,lat1,lon1,lat2,lon2,alt1) places the first point at the specified altitude in meters above the surface. This is equivalent to putting the point on a tower. If omitted, the point is assumed to be on the surface.

vis = los2(map,maplegend,lat1,lon1,lat2,lon2,alt1,alt2) also places the second point at a specified altitude in meters above the surface. If omitted, the point is assumed to be on the surface.

vis = los2(map,maplegend,lat1,lon1,lat2,lon2,alt1,alt2,*alt1opt*) controls whether the first point is at a relative or absolute altitude. If the altitude option is 'AGL', the point's altitude is in meters above the terrain ground level. If alt1opt is 'MSL', alt1 is interpreted as altitude above zero, or mean sea level. If omitted, 'AGL' is assumed.

vis = los2(map,maplegend,lat1,lon1,lat2,lon2,alt1,alt2,*alt1opt*,*alt2opt*) also controls the interpretation of the second point's altitude.

vis = los2(map,maplegend,lat1,lon1,lat2,lon2,alt1,alt2,alt1opt,alt2opt,actualradius) does the visibility calculation on a sphere with the specified radius. If omitted, the radius of the earth in meters is assumed. The altitudes, elevations and the radius should be in the same units. This calling form is most useful for computations on bodies other than the Earth.

vis = los2(map,maplegend,lat1,lon1,lat2,lon2,alt1,alt2,*alt1opt*,*alt2opt*,actualradius,effectiveradius) assumes a larger radius for propagation of the line of sight. This can account for the curvature of the signal path due to refraction in the atmosphere. For example, radio propagation in the atmosphere is commonly treated as straight line propagation on a sphere with 4/3rds the radius of the earth. In that case the last two arguments would be R_e and 4/3*R_e, where R_e is the radius of the Earth. Use Inf as the effective radius for flat Earth visibility calculations. The altitudes, elevations and the radii should be in the same units.

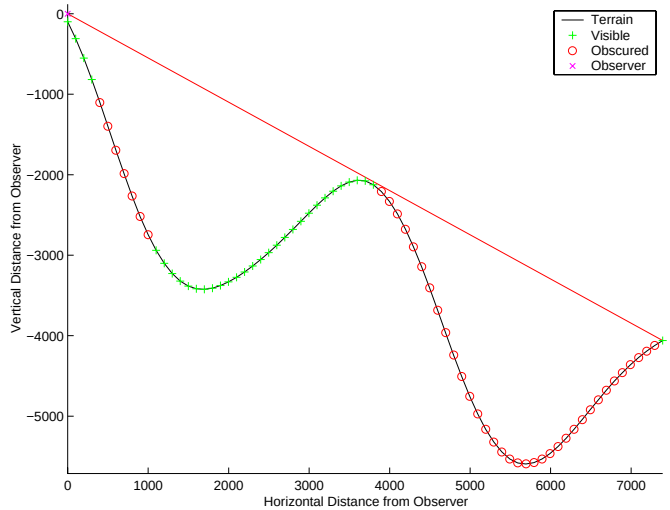
[vis,visprofile,dist,z,lattrk,lontrk] = los2(...) also returns vectors of points along the path between the two points. The visprofile is a vector containing ones where the intermediate points are visible. Dist is the distance along the path (in meters or the units of the radius). Z contains the terrain profile relative to the z datum along the path. Lattrk and lontrk are the latitudes longitudes of the the points along the path.

los2(...), with no output arguments, displays the visibility profile between the two points in a new figure.

Example

Check the visibility between two points on the peaks map.

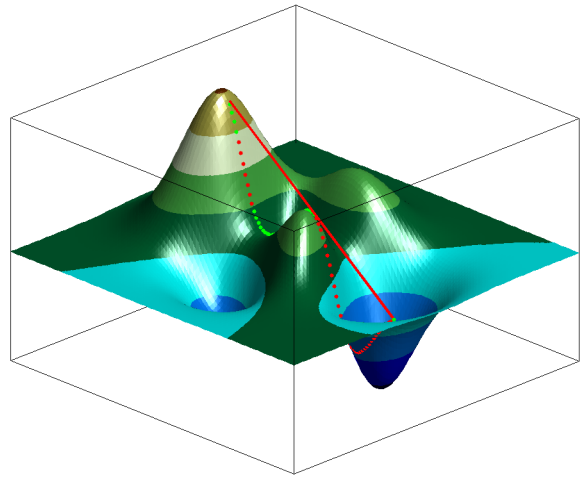
```
map = 500*peaks(100);
maplegend = [ 1000 0 0];
[lat1,lon1,lat2,lon2]=deal(-0.027,0.05,-0.093,0.042);

los2(map,maplegend,lat1,lon1,lat2,lon2,100)

ans =

     1
```



Display the same data together with the peaks surface.

```
figure
axesm('globe','geoid',almanac('earth','sphere','meters'))
meshm(map,maplegend,size(map),map); axis tight
camposm(-10,-10,1e6); camupm(0,0)
demcmap('inc',map,1000); shading interp; camlight

[vis,visprofile,dist,z,lattrk,lontrk] =
los2(map,maplegend,lat1,lon1,lat2,lon2,100);
```

```
plot3m(lattrk([1;end]),lontrk([1; end]),z([1; end])+[100;
0],'r','linewidth',2)
plotm(lattrk(~visprofile),lontrk(~visprofile),z(~visprofile),'r
.','markersize',10)
plotm(lattrk(visprofile),lontrk(visprofile),z(visprofile),'g.',
'markersize',10)
```



See Also

`viewshed`	Compute visible areas from a point on a digital elevation map
`mapprofile`	Interpolate values between waypoints on a regular matrix map

ltln2val

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 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 cells, 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 =
   313
   1660
   297
```

See Also

`findm` Coordinates of nonzero map entries

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

makemapped

Purpose

Make an object a mapped object

Syntax

```
makemapped(h)
```

Background

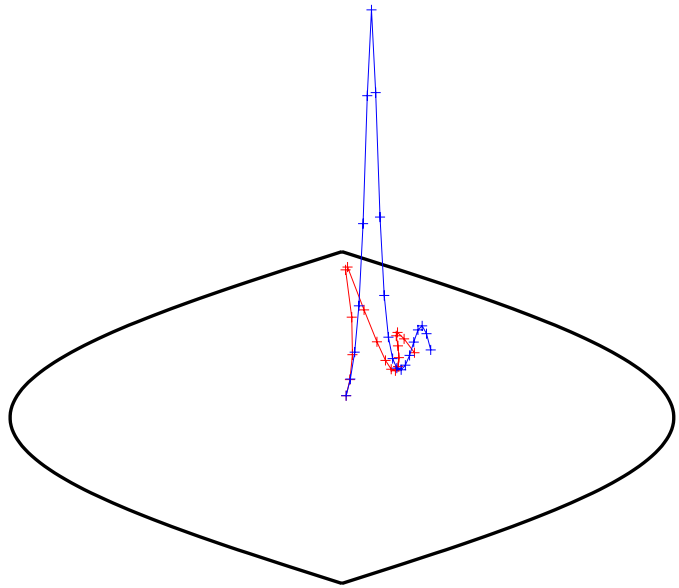
The Mapping Toolbox identifies displayed objects that have been projected from geographic coordinates by a special structure in that object's `UserData` property. Objects created using standard MATLAB display commands lack this structure, and retain the same Cartesian coordinates, regardless of the projection. 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

map legend vector

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

Purpose

Values between waypoints on a regular matrix map.

Syntax

```
[z,rng,lat,lon] = mapprofile
[z,rng,lat,lon] = mapprofile(map,maplegend,lat,lon)
[z,rng,lat,lon] = mapprofile(map,maplegend,lat,lon,rngunits)
[z,rng,lat,lon] = mapprofile(map,maplegend,lat,lon,ellipsoid)
[z,rng,lat,lon] = mapprofile(map,maplegend,lat,lon,rngunits,trackmethod,interpmethod)
[z,rng,lat,lon] =
   mapprofile(map,maplegend,lat,lon,ellipsoid,trackmethod,interpmethod)
```

Description

mapprofile plots a profile of values between waypoints on a displayed regular matrix map. `mapprofile` uses the current object if it is a regular matrix map, or the first regular matrix map found on the current axes. The map's zdata is used for the profile. The color data is used in the absence of data in z. The result is displayed in a new figure.

[z,rng,lat,lon] = mapprofile returns the values of the profile without displaying them. The output z contains interpolated values from map along great circles between the waypoints. Rng is a vector of associated distances from the first waypoint in units of degrees of arc along the surface. Lat and lon are the corresponding latitudes and longitudes.

[z,rng,lat,lon] = mapprofile(map,maplegend,lat,lon) uses the provided regular matrix map and waypoint vectors. No displayed map is required. Sets of waypoints may be separated by NaNs into line sequences. The output ranges are measured from the first waypoint within a sequence.

[z,rng,lat,lon] = mapprofile(map,maplegend,lat,lon,rngunits) specifies the units of the output ranges along the profile. Valid range units inputs are any distance string recognized by DISTDIM. Surface distances are computed using the default radius of the earth. If omitted, 'degrees' are assumed.

[z,rng,lat,lon] = mapprofile(map,maplegend,lat,lon,ellipsoid) uses the provided ellipsoid definition in computing the range along the profile. The ellipsoid vector is of the form [semimajor axes, eccentricity]. The output range is reported in the same distance units as the semimajor axes of the ellipsoid vector. If omitted, the range vector is for a sphere.

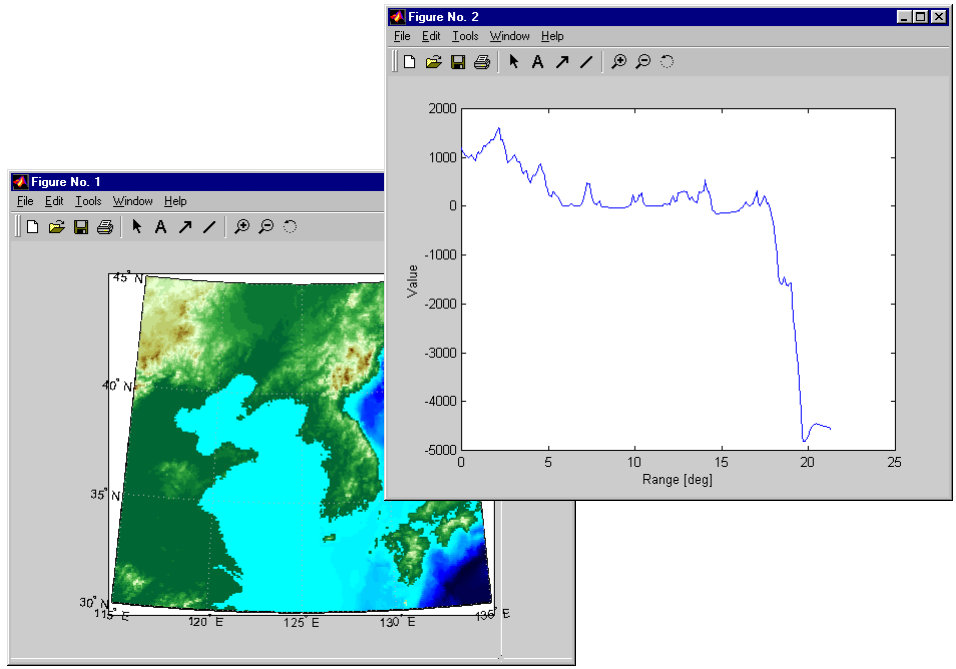
[z,rng,lat,lon] = mapprofile(map,maplegend,lat,lon,rngunits,*trackmethod*,*interpmethod*) and [z,rng,lat,lon] = mapprofile(map,maplegend,lat,lon,ellipsoid,*trackmethod*,*interpmethod*) control the interpolation methods used. Valid trackmethods are 'gc' for great circle tracks

between waypoints, and 'rh' for rhumb lines. Valid interpmethods for interpolation within the matrix are 'bilinear' for linear interpolation, 'bicubic' for cubic interpolation, and 'nearest' for nearest neighbor interpolation. If omitted, 'gc' and 'bilinear' are assumed.

Example

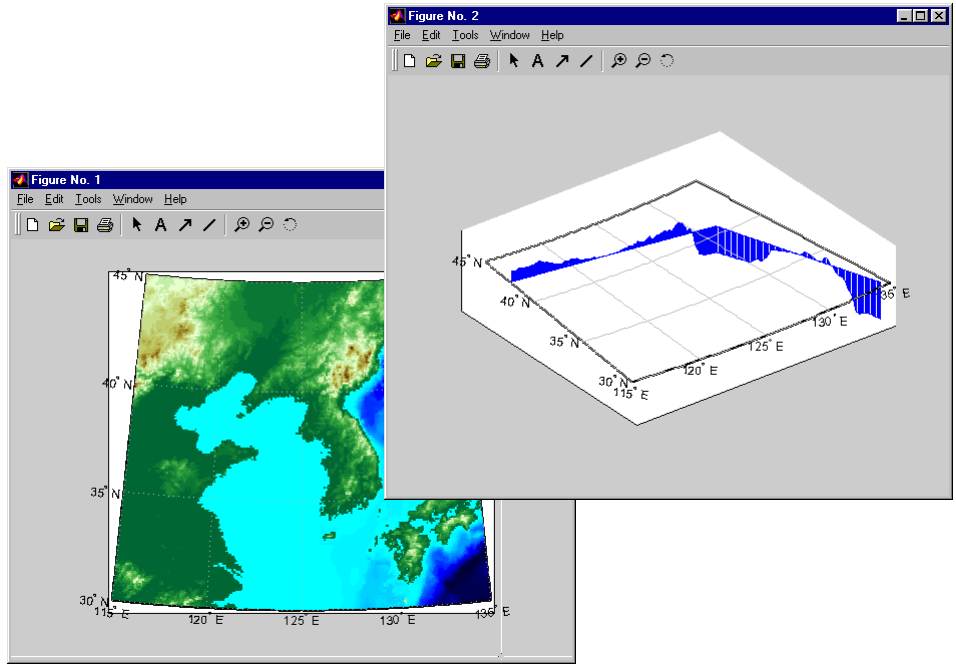
What is the elevation profile across the sample Korean digital elevation data? Take the data and the way points from a map display. Click on the upper left and the lower right corners of the map, and then press the **Enter** key.

```
load korea
worldmap(map,maplegend,'dem')
mapprofile
```



When you select more than two wayppoints, the automatically generated figure displays the result in three dimensions. The following example shows the relative sizes of the mountains in northern China compared to the depths of the Sea of Japan.

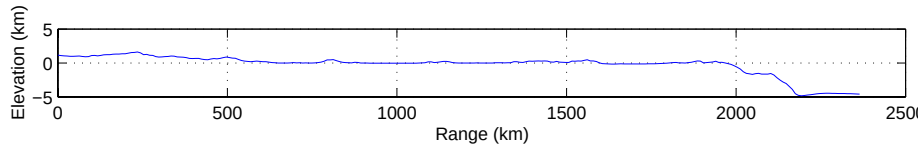
```
close
mapprofile
```



Adding output arguments supresses the display of the results in a new figure. You can then use the results in furthur calculations, or display the results yourself. Here the profile from the upper left to lower right is computed from waypoints selected on the map. The ranges and elevations are converted to kilometers and displayed in a new figure. The vertical exaggeration factor is set to 20. With no vertical exaggeration, the changes in elevation would be to almost to small to see.

```
[z,rng,lat,lon] = mapprofile;
figure
plot(deg2km(rng),z/1000)
daspect([ 1 1/20 1 ]); grid
xlabel 'Range (km)'
```

```
ylabel 'Elevation (km)'
```



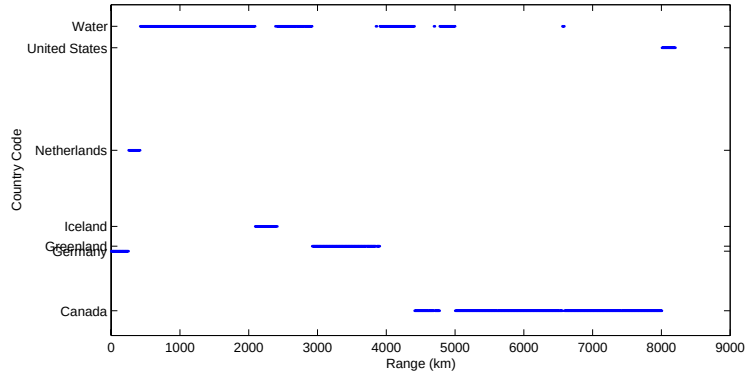
You can compute values along a path without reference to an existing figure by providing a regular matrix map and vectors of waypoint coordinates. Optional arguments allow control over the units of the range output and interpolation methods between waypoints and matrix elements.

Find the countries that lie under a great circle track from Frankfurt to Seattle. Use the 15 minute (25 kilometer) `worldmtxmed` political matrix map data.

```
load worldmtxmed
[lat,lon] =
extractm(worldlo('PPpoint'),{'Seattle','Frankfurt'});
lat(isnan(lat)) = []; lon(isnan(lon)) = [];
[lat lon]

ans =

   50.1082    8.6732
   47.5070 -122.3589


[valp,rngp,latp,lonp] =
mapprofile(double(map),maplegend,lat,lon,'km','gc','nearest');

ucodes = unique(valp)

ucodes =

    35
    71
    74
    86
   132
   194
```

```
   207

unames = names(ucodes)

unames =

    'Canada'
    'Germany'
    'Greenland'
    'Iceland'
    'Netherlands'
    'United States'
    'Water'
```

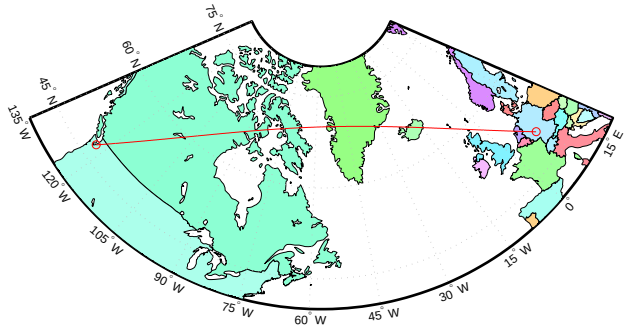
```
plot(rngp,valp,'.')
xlabel 'Range (km)'
ylabel 'Country Code'
set(gca,'ytick',ucodes)
set(gca,'yticklabel',unames)
```



Verify the result by displaying the track on a map.

```
worldmap([40 80],[-135 20],'patch')
plotm(latp,lonp,'r')
plotm(lat,lon,'ro')
```

```
hidem(gca)
```



See Also

`ltln2val`	Values for position on a regular matrix map.
`mlos2`	Line of sight between two points.

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

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 NaNs inserted at trim points.

See Also

`maptrimp`	Trim patch map data to specified region
`maptrims`	Trim surface matrix map data to specified region

maptrimp

Purpose

Trim patch map data to a specified region

Syntax

```
[lat,lon] = maptrimp(lat0,lon0,latlim,lonlim)
```

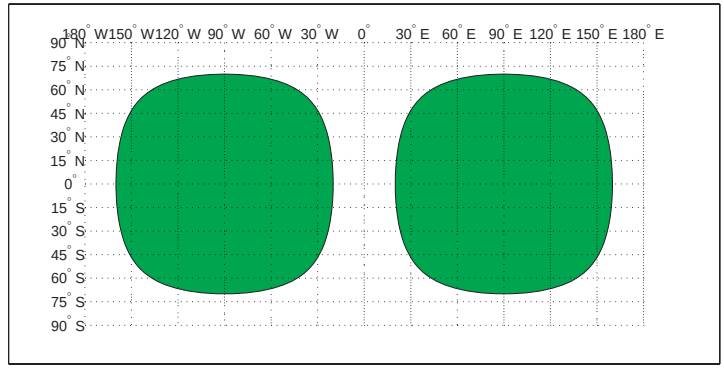
Description

[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:

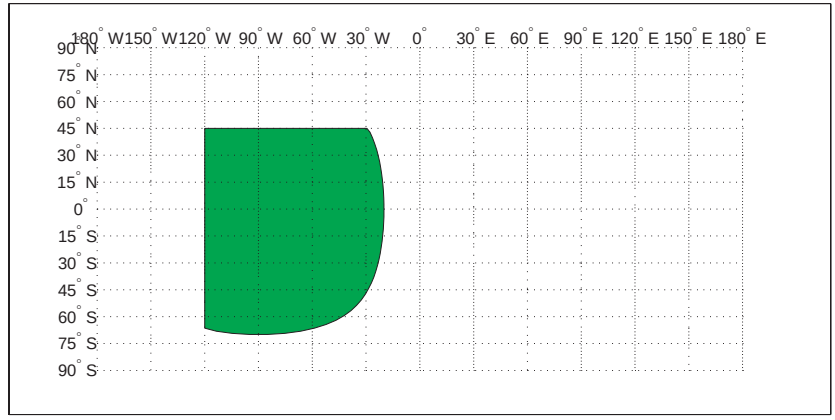
```
[lat0,lon0] = scircle1([0 0]',[-90 90]',[70 70]');
axesm('pcarree','Grid','on',...
      'MeridianLabel','on','ParallelLabel','on')
h = fillm(lat0,lon0,'green');
```



Now, trim the patch data to lie between 80 Sand 45 Nlatitude, and 120 Wand 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

maptrims

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,...
                              [80.25 85.3],[165.2 170.7])

submap =

       -2826       -2810       -2802       -2793
       -2915       -2913       -2905       -2884
       -3192       -3186       -3165       -3122
       -3399       -3324       -3273       -3214


sublegend =

     1    85   166
```

The upper left corner of the map may differ slightly from that of the requested region. `maptrims` uses the corner coordinates of the first cell inside the limits.

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

maskm

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, NaNs 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

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

mat2hms

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

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.

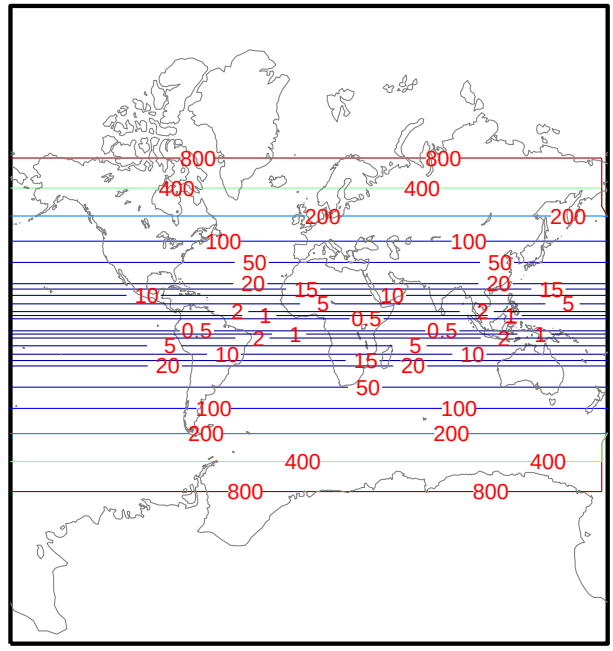
mdistort(parameter,levels,gsize) controls the size of the underlying graticule matrix used to compute the contours. gsize is a two-element vector containing the number of rows and columns. If omitted, the default Mapping Toolbox graticule size of [50 100] is assumed.

[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.

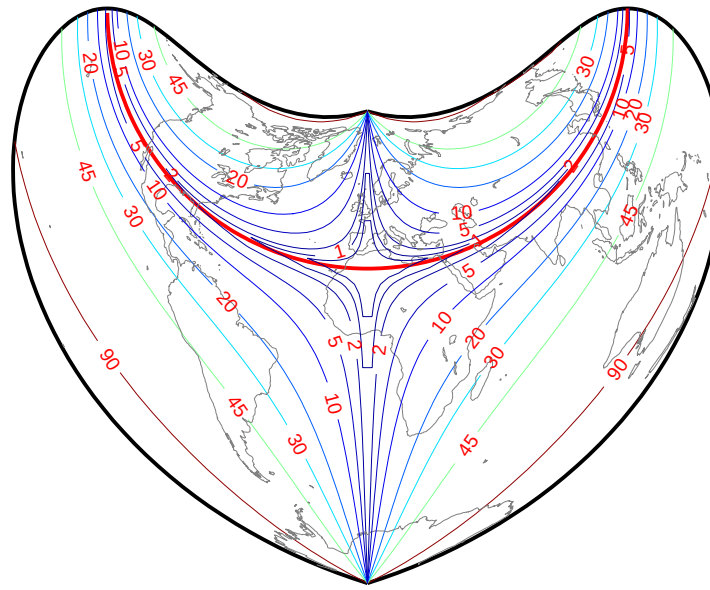
```
figure
axesm mercator
framem;plotm(coast,'color',.5*[1 1 1])

mdistort area
```



The lines of zero distortion for the Bonne projection follow the central meridian and the standard parallel.

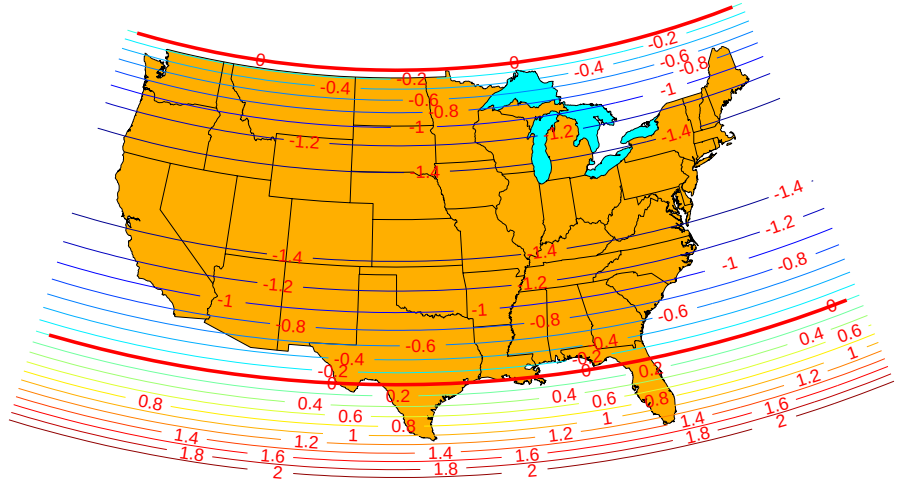
```
figure
axesm bonne
framem;plotm(coast,'color',.5*[1 1 1])

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

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 the "Mapping Applications" chapter 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

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

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

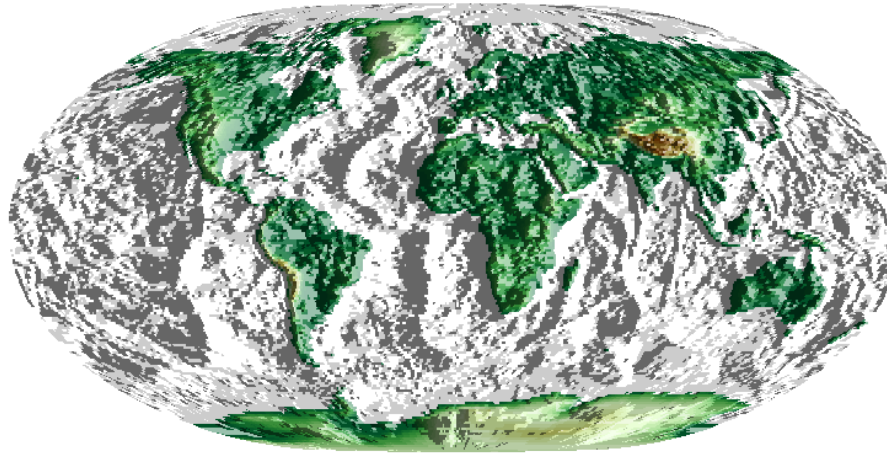
This function effectively multiplies two colormaps, one with color based on elevation, the other with a grayscale based on the slope of the surface, to create a new colormap. This produces an effect similar to using a light on a surface, but with all of the visible colors actually in the colormap. Lighting 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

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.

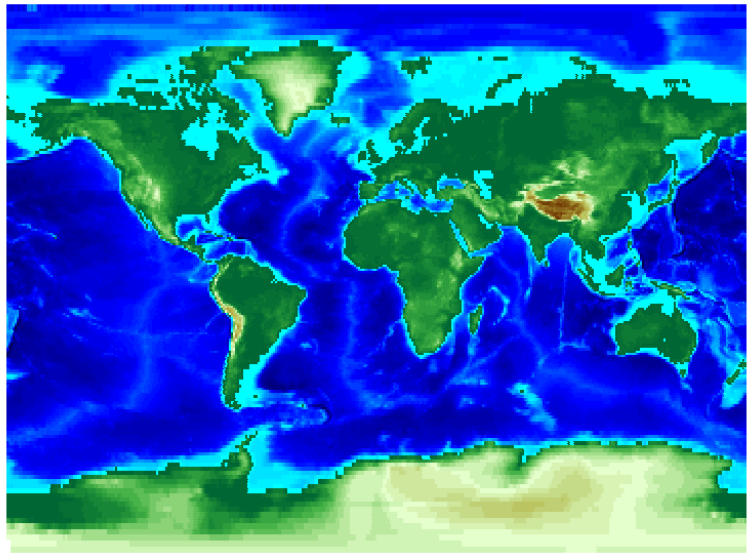
`h = meshm(map,maplegend,npts)` specifies the resolution of the graticule grid. The input `npts` is of the form `[latitude-points longitude-points]`. The default value of `npts` is `[50 100]` (the graticule has 50 vertices in the latitude direction and 100 vertices in the longitude direction).

`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

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 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.

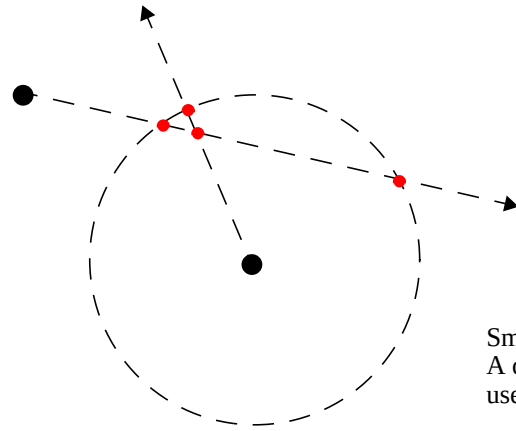
Examples

For a fully illustrated example of the application of this function, please refer to the "Navigation" section in 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`
`polyxpoly`
	Other intersection functions

navigational track format

Purpose

Describe waypoints of a navigational track

Description

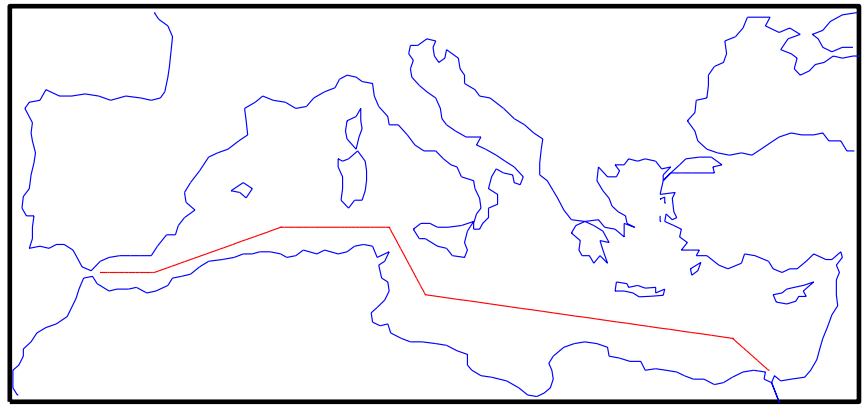
Track waypoints put into latitude and longitude column-vectors are said to be in *navigational track format*. *Waypoints* are points through which a track passes, usually corresponding to course or speed changes. A navigational track is made up of the line segments connecting these waypoints, which are 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 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

neworig

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'`.

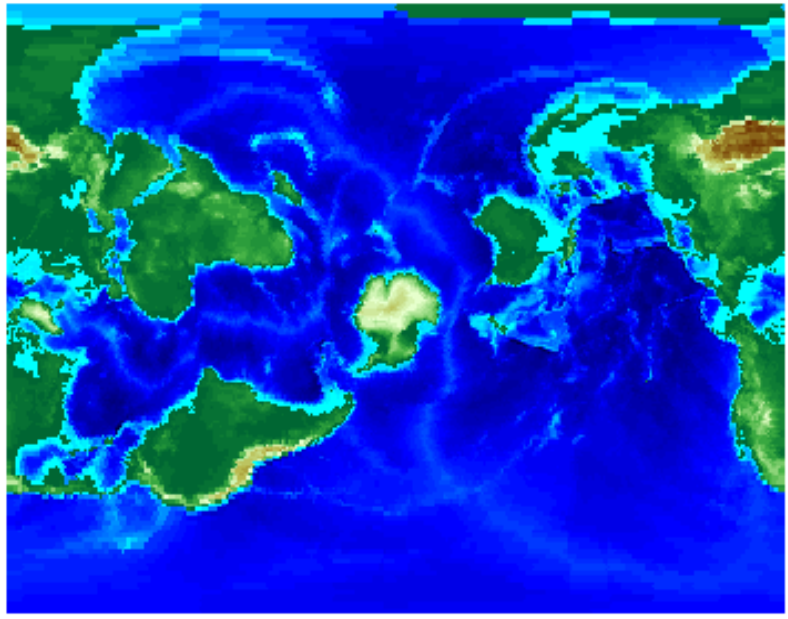
Limitations

`neworig` only support matrix maps that cover the entire globe.

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

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

nm2deg, nm2km, nm2rad, nm2sm

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

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

onem

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

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

paperscale

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

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

patchesm

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.

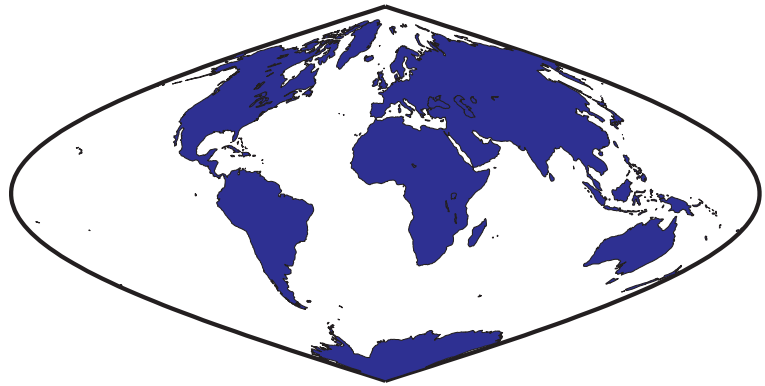
`h = patchesm(lat,lon,cdata)` projects and displays patch (polygon) objects defined by their vertices given in `lat` and `lon` on the current map axes. `lat` and `lon` must be vectors. The color data, `cdata`, can be any color data designation supported by the standard MATLAB `patch` command. The object 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

patchm

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`.

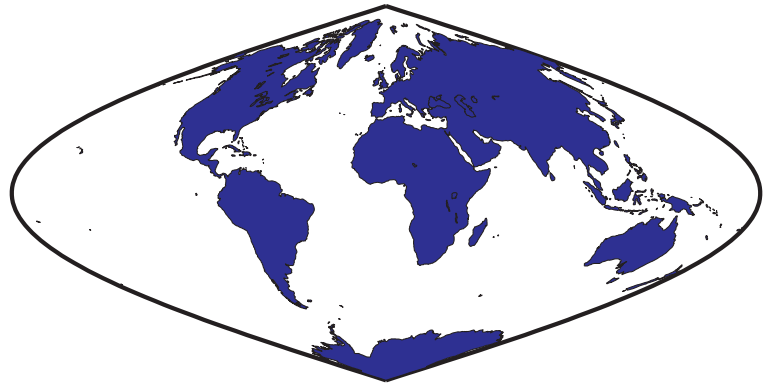
h = patchm(lat,lon,cdata) projects and displays patch (polygon) objects defined by their vertices given in `lat` and `lon` on the current map axes. `lat` and `lon` must be vectors. The color data, `cdata`, can be any color data designation supported by the standard MATLAB `patch` command. The object 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

pcolorm

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.

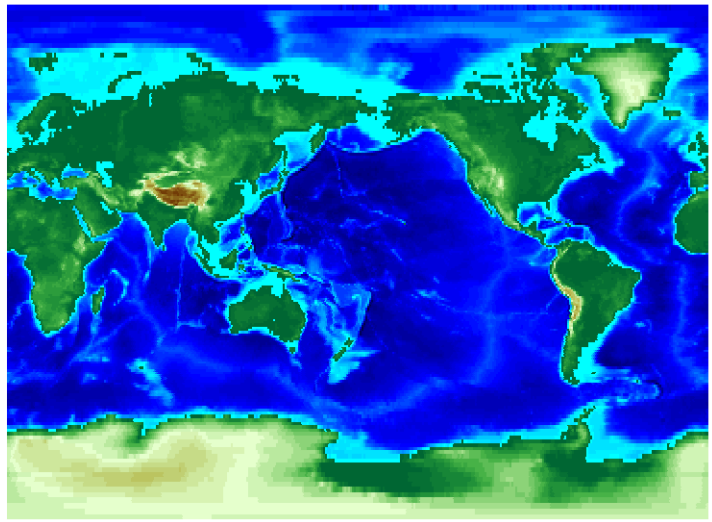
`h = pcolorm(map,npts)` results in a graticule grid defined by `npts`, which is a two-element vector of the form `[latitude-points longitude-points]`. The default `npts` is `[50 100]`.

`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

plabel

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

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.

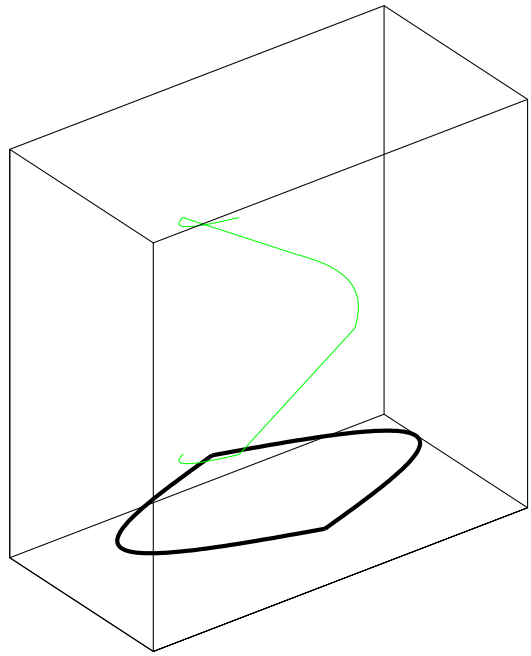
`h = plot3m(lat,lon,z)` displays projected line objects on the current map axes. `lat` and `lon` are the latitude and longitude coordinates, respectively, of the line object to be projected. Note that this ordering is conceptually reversed from the MATLAB `line` command, because the *vertical* (*y*) 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

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`.

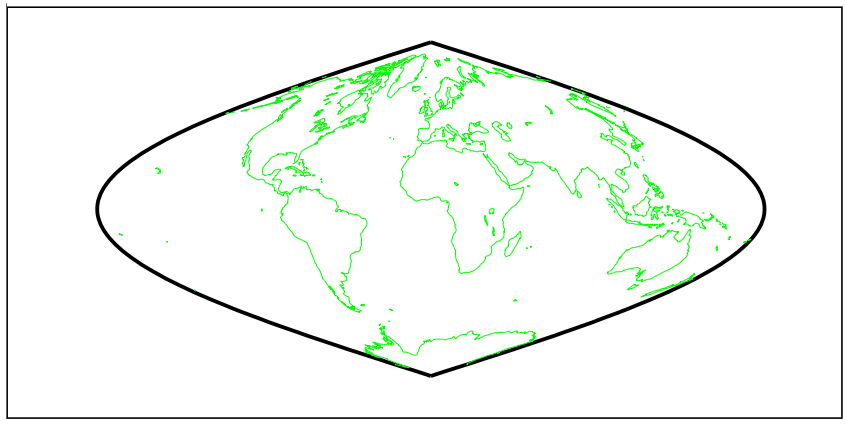
h = plotm(lat,lon,*linetype*) allows the specification of the line style, where *linetype* is any string recognized by the MATLAB `line` command.

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

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

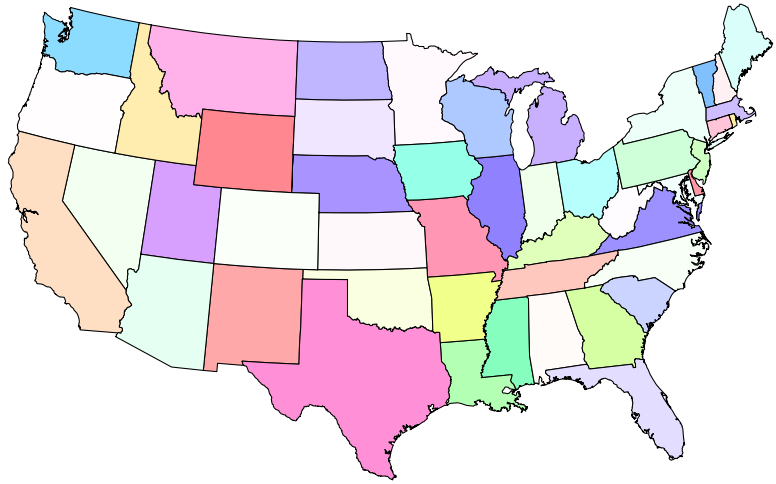
polcmap(ncolors,maxsat) controls the maximum saturation of the colors. Larger maximum saturation values produce brighter, more saturated colors. If omitted, the default is 0.5.

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

Purpose

Perform polygon boolean operations.

Syntax

```
[lat,lon] = polybool(flag,x1,y1,x2,y2)
[lat,lon] = polybool(flag,x1,y1,x2,y2,outputformat)
```

Description

[lat,lon] = polybool(*flag*,x1,y1,x2,y2) performs the polygon boolean operation identified by *flag*. Valid flags strings are one of the following alternatives: ('intersection','and','&'), ('union','or','|','+','plus'), ('exclusiveor','xor') and ('subtraction','minus','-'). The polygon inputs are NaN-delimited vectors, or cell arrays containing individual polygons in each element with the outer face separated from the subsequent inner faces by NaNs. The result is output as NaN-delimited vectors.

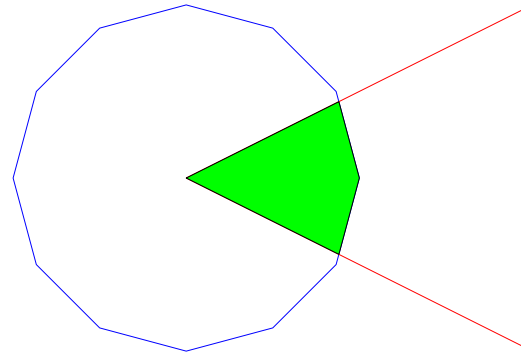
[lat,lon] = polybool(*flag*,x1,y1,x2,y2,*outputformat*) controls the format of the resulting polygons. If *outputformat* is 'vector', the result is returned as vectors with NaNs separating the faces. No distinction is made between outer and inner faces of polygons. If *outputformat* is 'cutvector', inner faces are connected to the enclosing polygon face by inserting a cut. If *outputformat* is 'cell', the result is returned as cell arrays containing individual polygons in each element, with the outer face separated from the subsequent inner faces by NaNs. If omitted, 'vector' is assumed.

Limitations

Polygons are assumed to be in a cartesian coordinate system. As such, geographic data that encompasses a pole cannot be used directly. Use flatearthpoly to convert polygons to cartesian coordinates.

Example

```
theta = (0:pi/6:2*pi)';
lat1 = sin(theta);
lon1 = cos(theta);
lat2 = [0 1 -1 0]';
lon2 = [0 2 2 0]';
[latb,lonb] = polybool('intersection',lat1,lon1,lat2,lon2);

axesm miller
plotm(lat1,lon1,'b')
plotm(lat2,lon2,'r')
patchm(latb,lonb,'g')
```



```
[latb2,lonb2] = polybool('xor',lat1,lon1,lat2,lon2,'cell')

latb2 =

    [ 6x1 double]
    [15x1 double]


lonb2 =

    [ 6x1 double]
    [15x1 double]
```

See Also

`bufferm`	Compute buffer zones for vector data
`polyjoin`	Convert polygon segments from cell array to vector format
`polysplit`	Extract segments of `NaN`-delimited polygon vectors to cell arrays

Purpose

Convert polygon segments from cell array to vector format.

Syntax

```
[lat,lon] = polyjoin(latcells,loncells)
```

Description

[lat,lon] = polyjoin(latcells,loncells) converts polygons from cell array format to vector format. In cell array format, each element of the cell array is a separate polygon. Each polygon may consist of an outer contour followed by holes separated with NaNs. In vector format, each vector may contain multiple faces separated by NaNs. There is no distinction between outer contours and holes.

Example

```
latcells = {[1 2 3]'; 4; [5 6 7 8 NaN 9]'};
loncells = {[9 8 7]'; 6; [5 4 3 2 NaN 1]'};
[lat,lon] = polyjoin(latcells,loncells);

[lat lon]
ans =
     1     9
     2     8
     3     7
   NaN   NaN
     4     6
   NaN   NaN
     5     5
     6     4
     7     3
     8     2
NaN   NaN
     9     1
```

See Also

polybool	Perform polygon boolean operations
polycut	Implement polygon branch cuts for holes
polysplit	Extract segments of NaN-delimited polygon vectors to cell arrays

polymerge

Purpose

Merge line segments with matching endpoints.

Syntax

```
[lat2,lon2] = polymerge(lat,lon)
[lat2,lon2] = polymerge(lat,lon,tol)
[lat2,lon2] = polymerge(lat,lon,outputformat)
```

Description

[lat2,lonc2 = polymerge(lat,lon) combines vector line segments with identical endpoints. `polymerge` compares the endpoints of all line segments and combines those which match. The line may be input as vectors of latitude and longitude with `NaN`s delimiting segments. The line may also be input as cell arrays, with each element of a cell array containing a line segment. The resulting line is in the same format as the input.

[lat2,lonc2 = polymerge(lat,lon,tol) combines line segments whose endpoints are separated by less than the circular tolerance. If omitted, `tol= 0` is assumed. The tolerance is in the same units as the polygon input.

[lat2,lonc2 = polymerge(lat,lon,tol,*outputformat*) controls the format of the resulting polygons. If outputformat is '`vector`', the result is returned as vectors with `NaN`s separating the segments. If outputformat is '`cell`', the result is returned as cell arrays containing segments in each element. If omitted, '`vector`' is assumed.

Example

```
lat = [1 2 3 NaN 6 7 8 9 NaN 6 5 4 3 NaN 12 13 14 NaN 9 10 11 12]';
lon = lat;
[lat2,lon2] = polymerge(lat,lon);

[lat2 lon2]
ans =
     1     1
     2     2
     3     3
     3     3
     4     4
     5     5
     6     6
     6     6
     7     7
     8     8
     9     9
     9     9
```

```
    10    10
    11    11
    12    12
    12    12
    13    13
    14    14
```

See Also

`polyjoin`	Convert polygon segments from cell array to vector format
`polysplit`	Extract segments of `NaN`-delimited polygon vectors to cell arrays

polysplit

Purpose

Extract segments of NaN-delimited polygon vectors to cell arrays.

Syntax

```
[latcells,loncells] = polysplit(lat,lon)
```

Description

[latcells,loncells] = polysplit(lat,lon) returns the NaN-delimited segments of the vectors lat and lon as cell arrays. Each element of the cell array contains one segment.

Example

```
lat = [1 2 3 NaN 4 NaN 5 6 7 8 9]';
lon = [9 8 7 NaN 6 NaN 5 4 3 2 1]';
[latcells,loncells] = polysplit(lat,lon);

[latcells' loncells']
ans =
    [3x1 double]    [3x1 double]
    [         4]    [         6]
    [5x1 double]    [5x1 double]
```

See Also

polybool	Perform polygon boolean operations
polyjoin	Convert polygon segments from cell array to vector format

Purpose

Compute line or polygon intersection points.

Syntax

```
[xi,yi] = polyxpoly(x1,y1,x2,y2)
[xi,yi] = polyxpoly(...,'unique')
[xi,yi,ii] = polyxpoly(...)
```

Description

`[xi,yi] = polyxpoly(x1,y1,x2,y2)` returns the intersection points of two sets of lines and/or polygons.

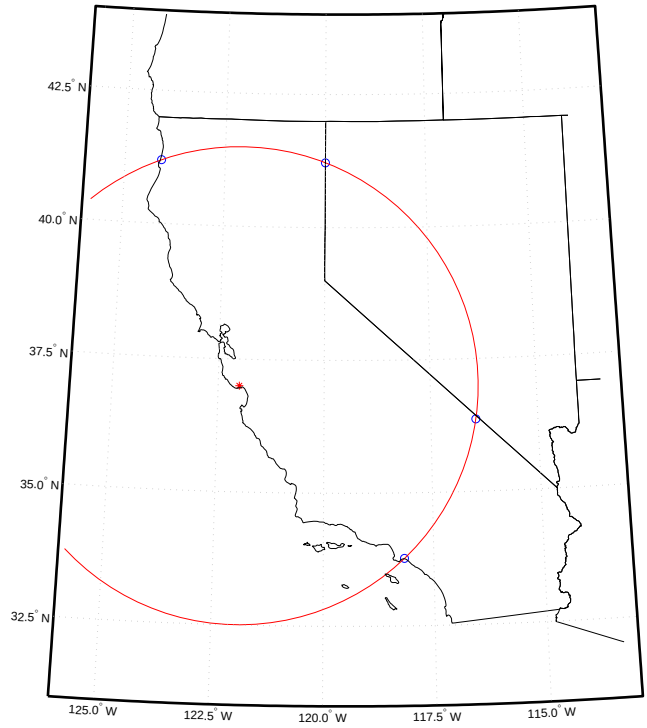
`[xi,yi] = polyxpoly(...,'unique')` returns only unique intersections.

`[xi,yi,ii] = polyxpoly(...)` also returns a two-column index of line segment numbers corresponding to the intersection points.

Example

```
[lat,lon]=extractm(usalo('state'),'california');
[lat,lon]=reducem(lat,lon);

lat0 = 37; lon0 = –122; rad = 500;
[latc,lonc] = scircle1(lat0,lon0,km2deg(rad));

usamap('california','lineonly')
plotm(lat0,lon0,'r*')
plotm(latc,lonc,'r')

[loni,lati] = polyxpoly(lon,lat,lonc,latc);
plotm(lati,loni,'bo')
```

scxsc	Compute the intersection points for pairs of small circles



See Also

crossfix	Compute cross fix positions for bearings and ranges
gcxgc	Compute the intersection points between two great circles
gcxsc	Compute the intersection points between a great and a small circle
navfix	Perform mercator-based navigational fixing
rhxrh	Compute the intersection point between two rhumb lines
scxsc	Compute the intersection points for pairs of small circles

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

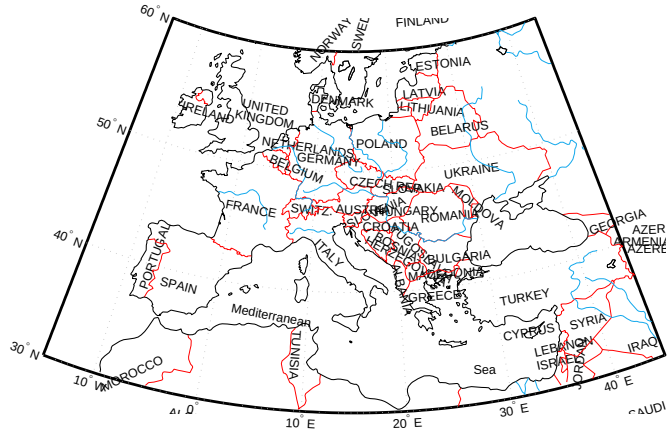
previewmap changes the size of the current figure to match the printed output. If the resulting figure size exceeds the screen size, the figure will be enlarged as much as possible.

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
```

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

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!

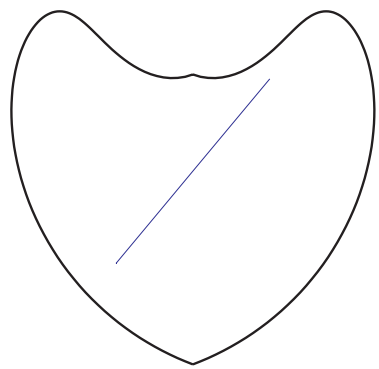
project(h,'xy') specifies that the XData of the unprojected objects correspond to longitudes and the YData to latitudes. This is the default assumption.

project(h,'yx') specifies that the XData of the unprojected objects correspond to latitudes and the YData to longitudes.

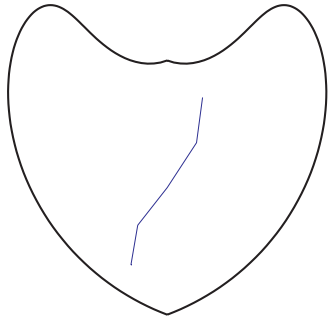
Examples

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

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

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`).

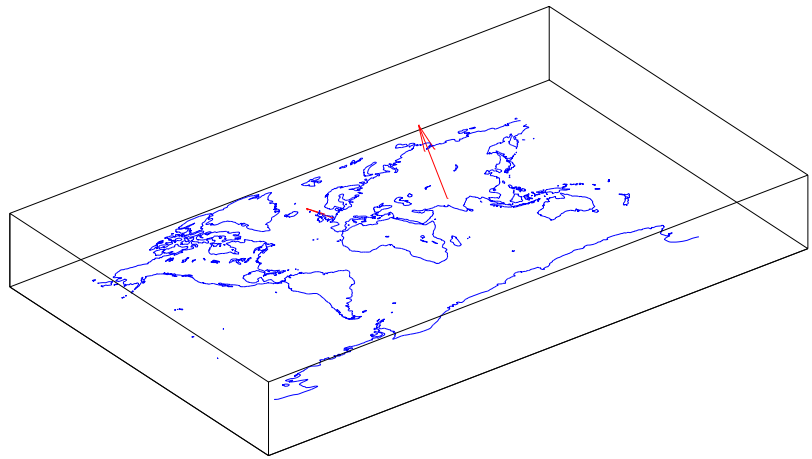
h = quiver3m(lat,lon,alt,u,v,w,*linespec*,'filled') results in any symbols specified by *linespec* being filled in.

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*)

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.

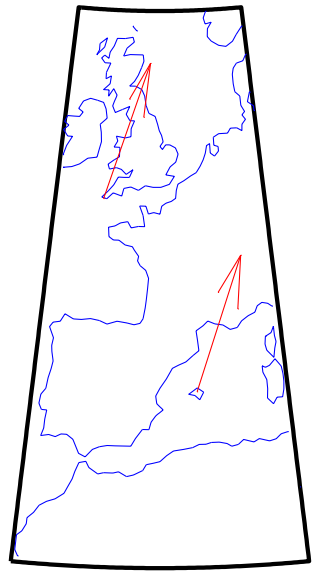
h = quiverm(lat,lon,u,v,scale) alters the automatically calculated vector lengths by multiplying them by the scalar value scale. For example, if scale is 2, the displayed vectors are twice as long as they would be if scale were 1 (the default). When scale is set to 0, the automatic scaling is suppressed, 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*)

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 inπ 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

rad2dms, rad2dm

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*

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

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

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

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

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

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

reckon

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

reducem

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.

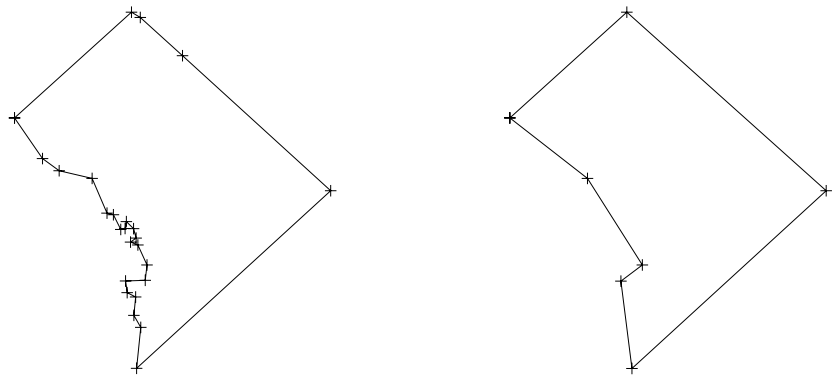
[latout,lonout,cerr] = reducem(...) in addition returns a measure of the error introduced by the simplification. The output `cerr` is the difference in the arc length of the original and reduced data, normalized by the original length.

[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

resizem

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
```

restack

Purpose

Re-stack objects within the axes

Syntax

```
restack(h,position)
```

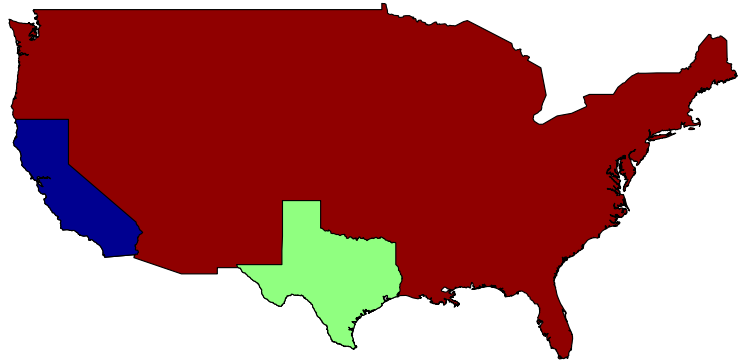
Description

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

rhxrh

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`
`polyxpoly`
	Other intersection functions

`navfix`	Navigational fixing

rootlayr

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 11, “GUI Reference”)

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

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.

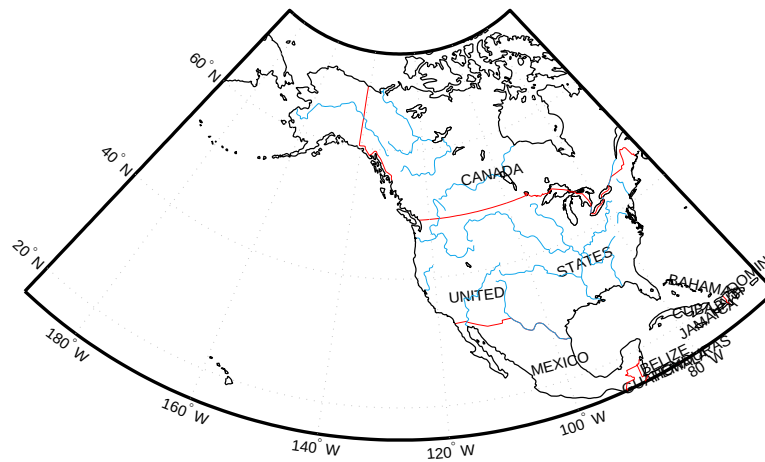
rotatetext(objects) rotates the selected objects. `objects` may be a name string recognized by `handlem`, or a vector of handles to displayed text objects.

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

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

rsphere

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

scaleruler

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.

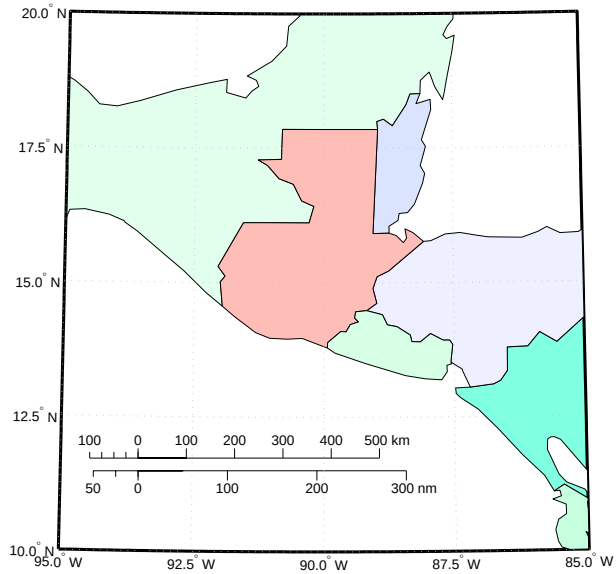
scaleruler off removes any currently displayed graphic scales.

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

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.

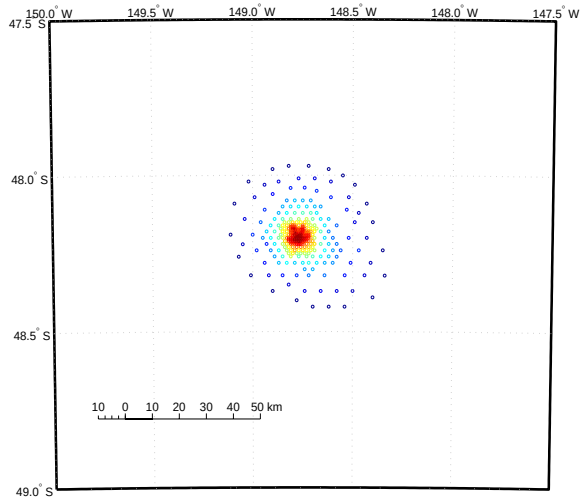
scatterm, without any inputs, will activate a GUI to project point data onto the current map axes.

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

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°:

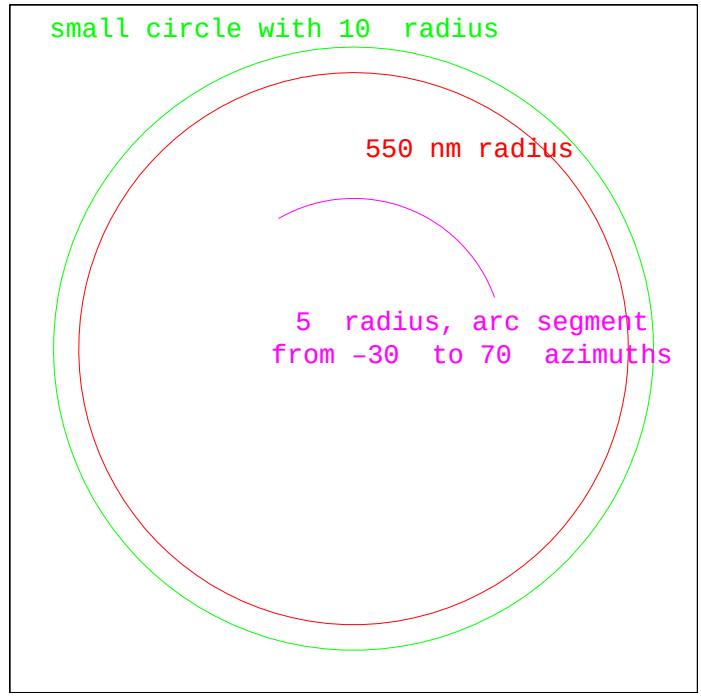
```
axesm('mercator','MapLatLimit',[30 -30],'MapLonLimit',[-30 30]);
[latc,longc] = scircle1(0,0,10);
plotm(latc,longc,'g')
```

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
`scircleg`	Interactive small circles
`trackg`	Interactive tracks
`track1` `track2`	Great circles and rhumb lines

scircle2

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.

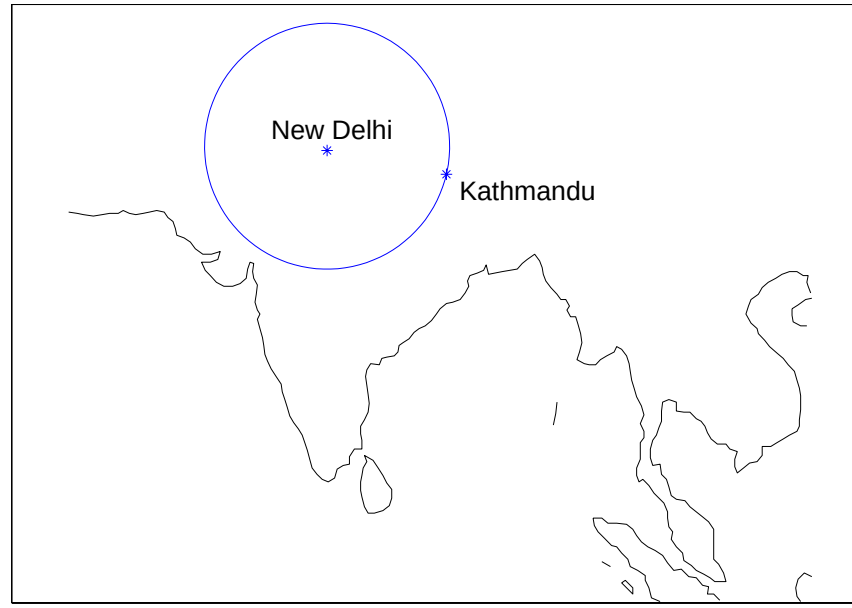
pts = scircle2(lat1,lon1,lat2,lon2) returns the points in a two-column output `pts`.

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

scircleg

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. You can modify the circle after creation by shift-clicking on it. The circle is then in a edit mode, during which you can change the size and position by dragging control points, or by entering values into a control panel. Shift-clicking again exits the edit mode.

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`
`polyxpoly`
	Other intersection functions

sec2hms, sec2hm

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

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

setltln

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

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.

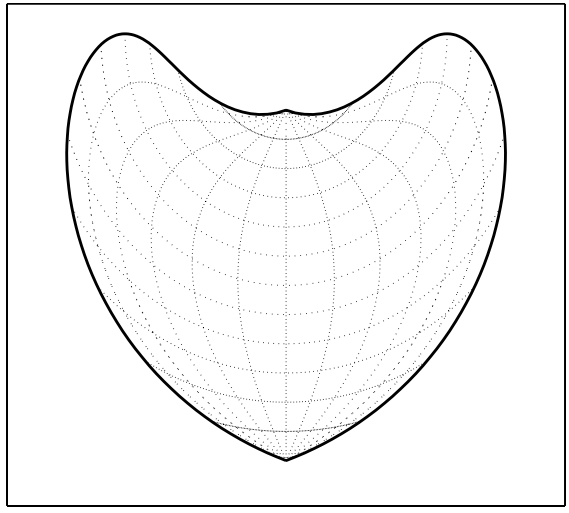
setm(surfhndl,'MeshGrat',npts,alt) alters the mesh graticule of projected surface objects displayed using the `meshm` command. In this case, the two-element vector `npts` specifies the graticule size in the manner described under `meshm`. The altitude can be a scalar or a matrix with a size corresponding to `npts`.

Examples

Display a map axes and alter it:

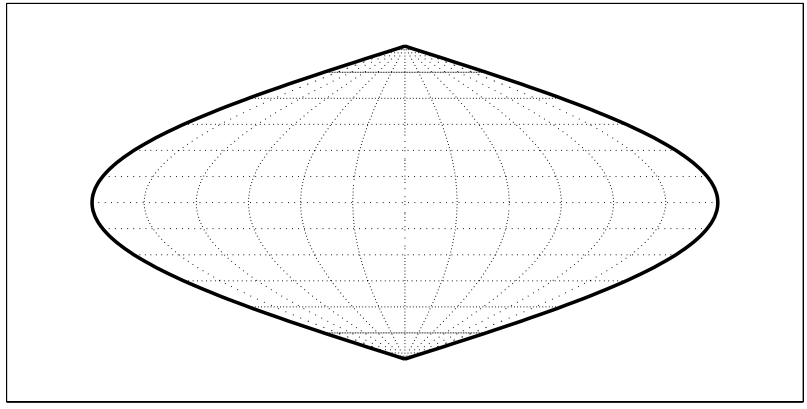
```
axesm('bonne','Frame','on','Grid','on')
```

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

setpostn

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

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

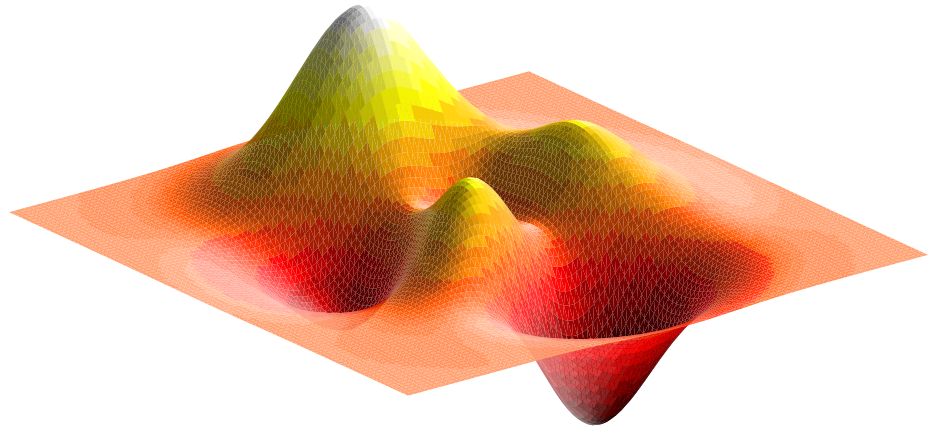
This function effectively multiplies two colormaps, one with color based on elevation, the other with a grayscale based on the slope of the surface, to create a new color map. This produces an effect similar to using a light on a surface, but with all of the visible colors actually in the colormap. Lighting 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

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

showm

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

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=4.05×10^7 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

sm2deg, sm2km, sm2nm, sm2rad

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

Purpose

Remove discontinuities in longitudes

Syntax

```
ang = smoothlong(angin)
ang = smoothlong(angin,units)
```

Description

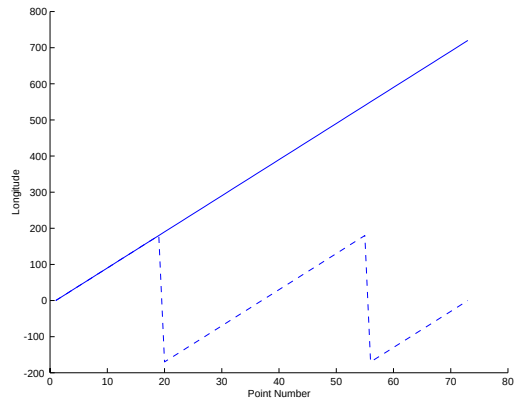
ang = smoothlong(angin) removes discontinuities in longitude data. The resulting angles may cover more than one revolution.

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

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

spzerom

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

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

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 the "Mapping Applications" chapter 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

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`.

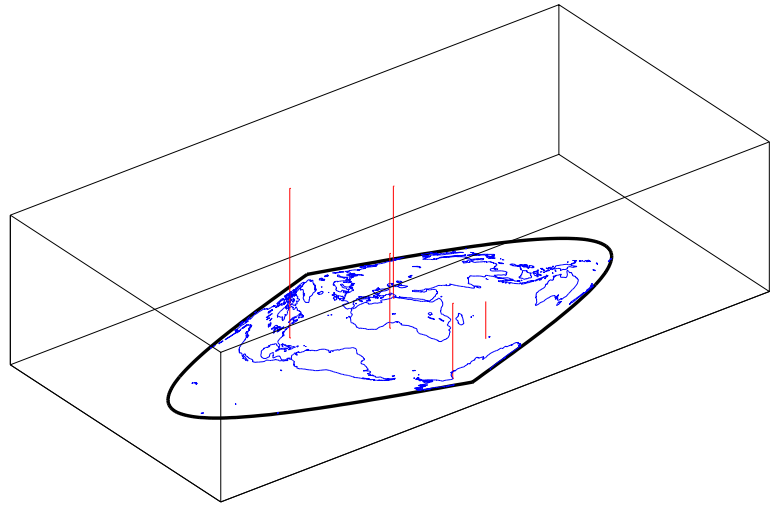
h = stem3m(lat,lon,z,*LineType*) allows the style of the stem plot's lines to be specified with any string *LineType* recognized by the MATLAB `line` command.

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

Purpose

Converts formatted DMS angle strings to numbers.

Syntax

```
angles = str2angle(strings)
```

Description

angles = str2angle(strings) converts the formatted Degrees-Minutes-Seconds strings to numeric angles in units of degrees. Examples of recognized formats are 123˚30'00"S, 123-30-00S, 123d30m00sS and 1233000S. The seconds field may contain a fractional component in all but the last form. Strings may be a character matrix or a cell array vector of strings.

Example

```
strs = {'23˚30''00"N', '23-30-00S', '123d30m00sE', '1233000W'}

strs =

    '23˚30'00"N'    '23-30-00S'    '123d30m00sE'    '1233000W'

str2angle(strs)

ans =

         23.5
        -23.5
        123.5
       -123.5

strs = strvcat(strs{:})

strs =

23˚30'00"N
23-30-00S
123d30m00sE
1233000W

str2angle(strs)

ans =

         23.5
        -23.5
```

```
 123.5
-123.5
```

See Also

`angl2str`	Angle conversion to a string

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).

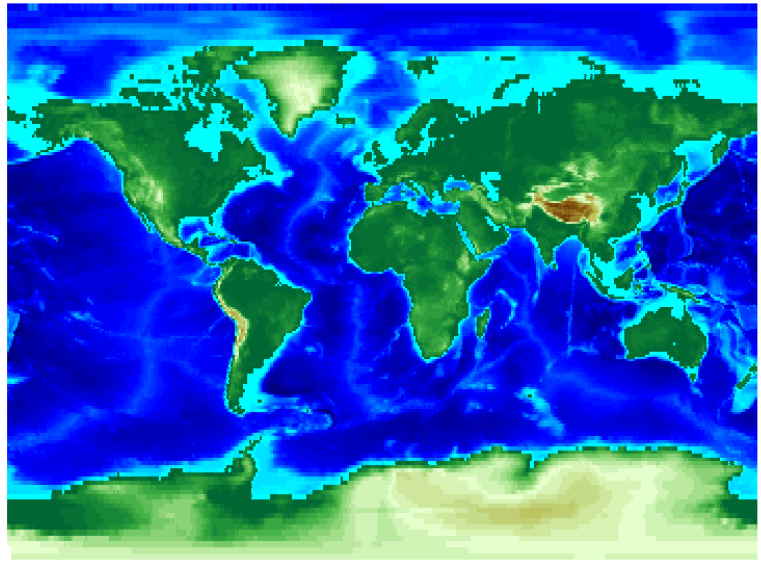
`h = surfacem(lat,lon,map)` allows greater graticule control through the inputs `lat` and `lon`. If matrices, they are the graticule vertex coordinates as might be returned by `meshgrat`. If vectors, they are the representative coordinates of the rows and columns, respectively, of such a grid. If they 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

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.

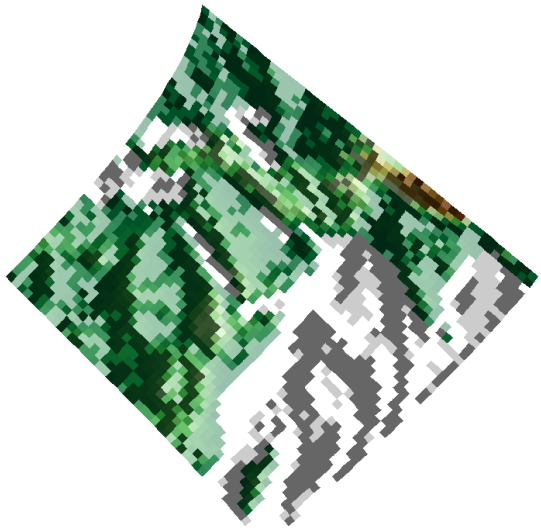
Remarks

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.

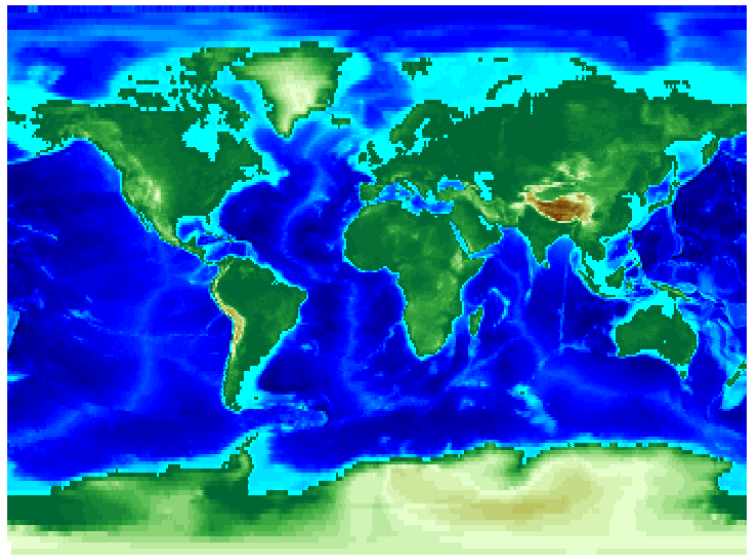
`h = surfm(lat,lon,map,alt)` sets the *z*-axis altitude of the graticule mesh. `alt` must be the same size as `lat`. If no `alt` is supplied, the mesh is plotted at z=0, unless `lat` is the same size as `map`, in which case `zdata=map`, and a 3-D topological map results. Since the default graticule is 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.

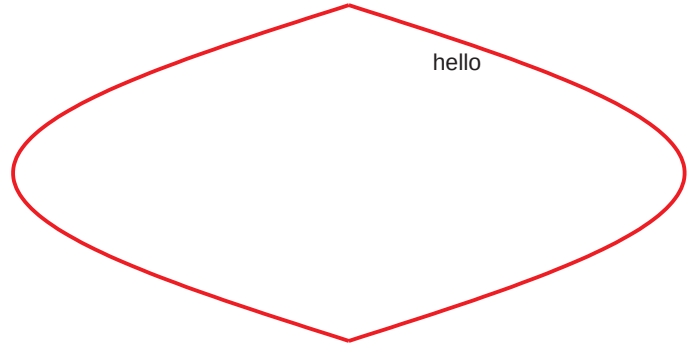
h = textm(lat,lon,z,*string*) displays the strings at the *z*-axis altitudes `z`. The default altitude is 0.

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`.

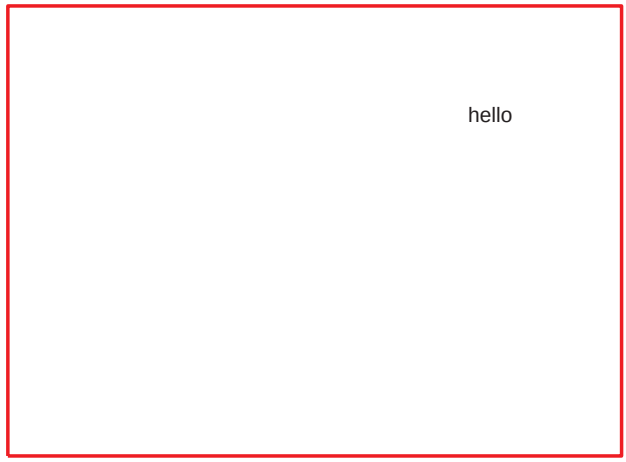
Example

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

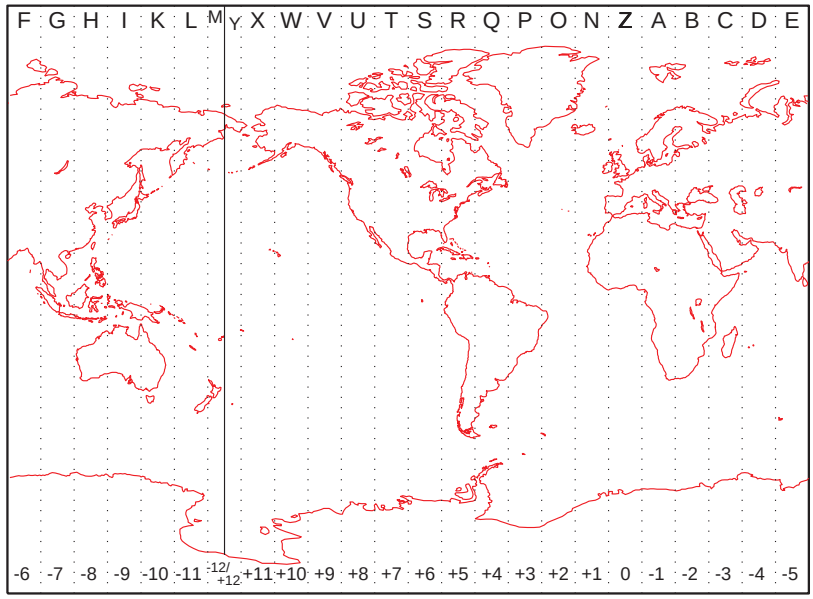
Given that it is locally 1330 (1:30 p.m.) at a longitude of 75°W, determine GMT:

```
[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 10, "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.

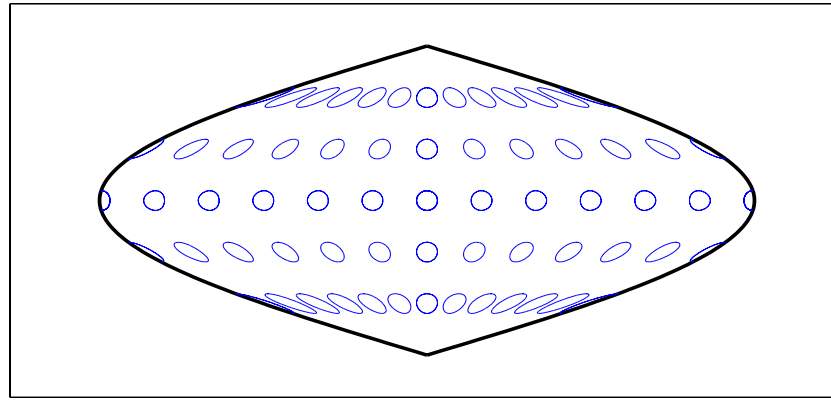
h = tissot(spec) allows you to specify plotting parameters of the displayed Tissot diagram as described above.

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.

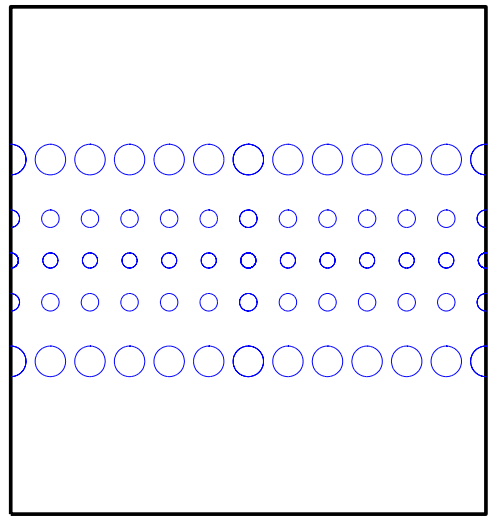
Examples

```
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 10, “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')
```

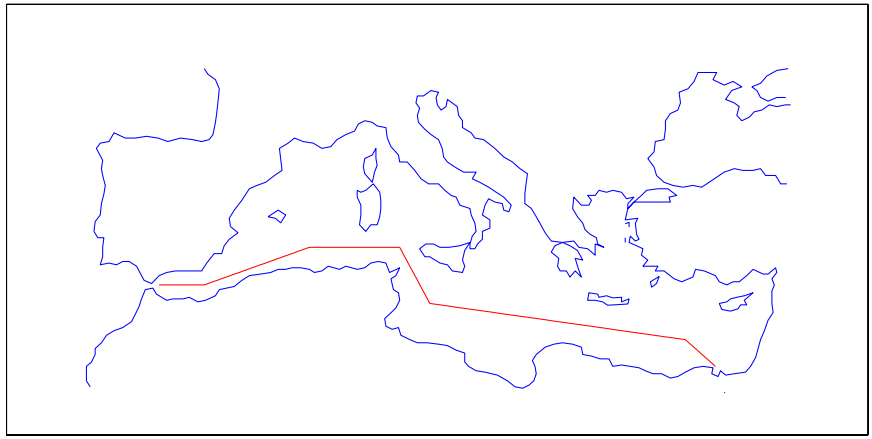
These are the waypoints Lt. Sextant has selected:

```
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.

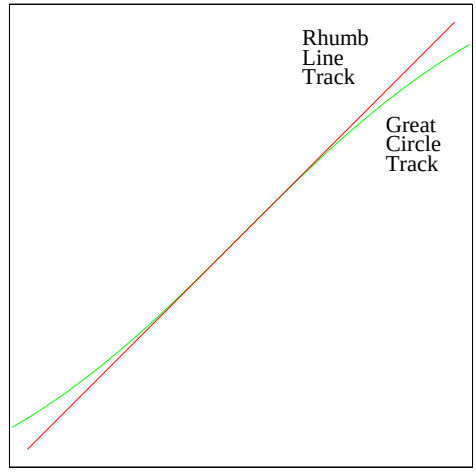
[lattrk,lontrk] = track1(track,lat,lon,az,rng,geoid,*units*) specifies the elliptical definition of the Earth with a two-element geoid model vector `geoid`. The default geoid is a spherical Earth, which is sufficient for most applications. If used, the units of the semimajor axis of the geoid vector 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
`trackg`	Interactive tracks

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.

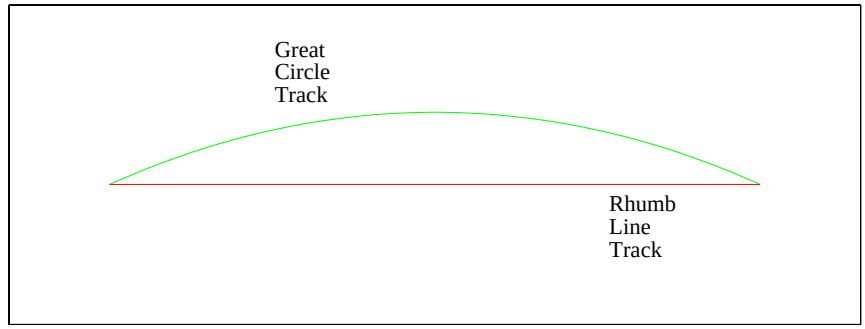
[lattrk,lontrk] = track2(lat1,lon1,lat2,lon2,geoid,*units*,npts) specifies the number of points, `npts`, per output track. `npts` is 100 by default.

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`	Interactive tracks

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. You can modify the track after creation by shift-clicking on it. The track is then in a edit mode, during which you can change the length and position by dragging control points, or by entering values into a control panel. Shift-clicking again exits the edit mode

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
`scircleg`	Interactive small circles

trimcart

Purpose

Trim graphic objects to the map frame

Syntax

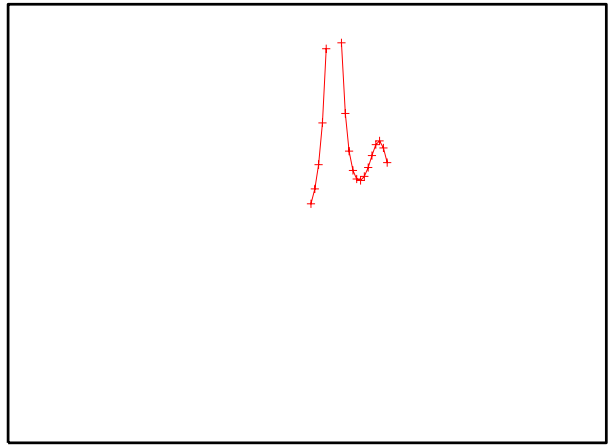
```
trimcart(h)
```

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

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

usahi

Purpose

Return data from the USAHI atlas data file

Syntax

```
usahi
s = usahi(request)
```

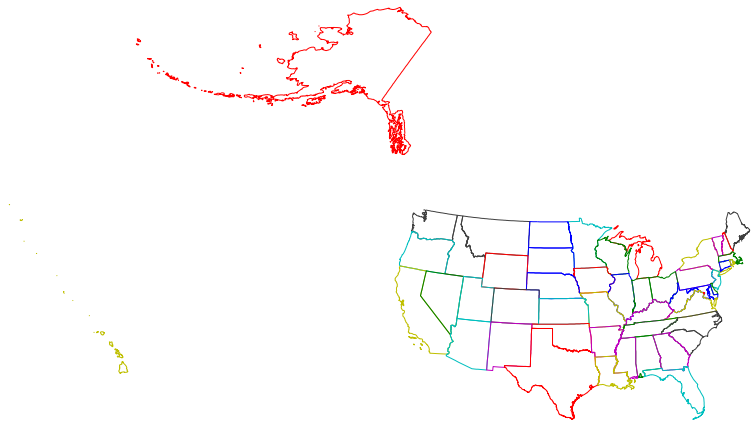
Description

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

Purpose

Return data from the USALO atlas data file

Syntax

```
usalo
s = usahlo(request)
```

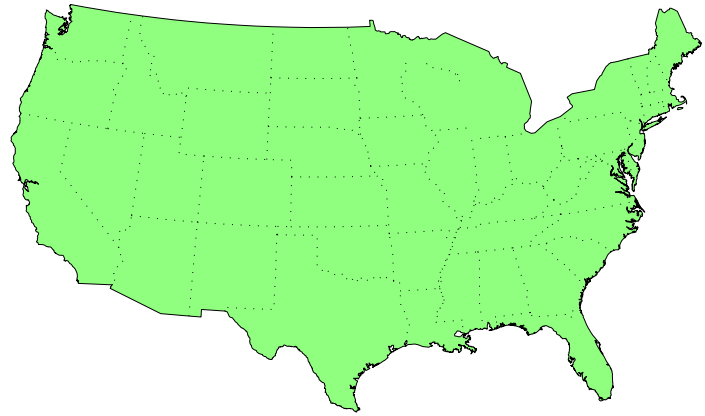
Description

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

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.

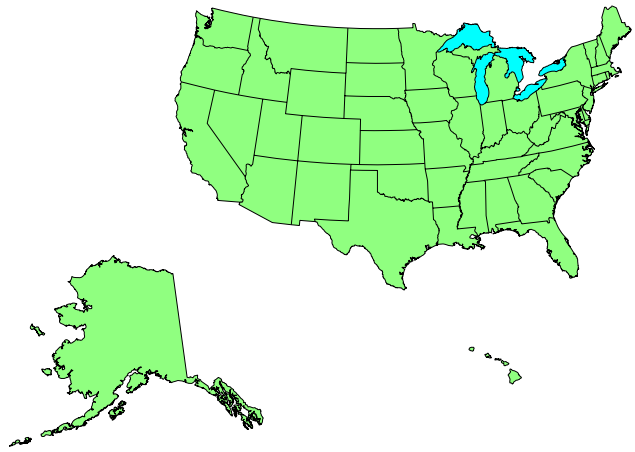
usamap(latlim,lonlim,*type*) is also a valid calling form.

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

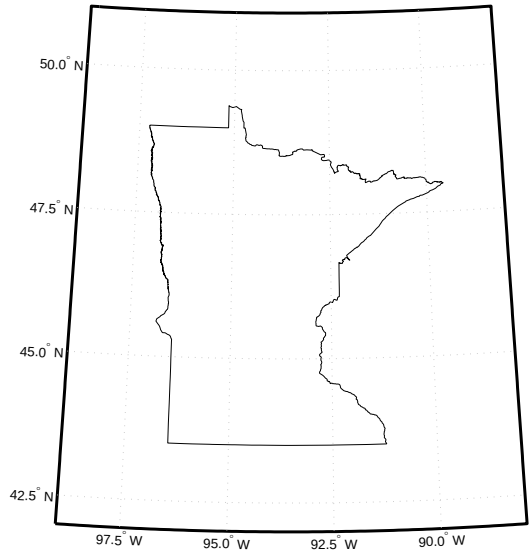
h = usamap(...) returns the handle or handles of the axes containing the map.

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

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

`utmgeoid` Universal Transverse Mercator suggested ellipsoids

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

vec2mtx

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.

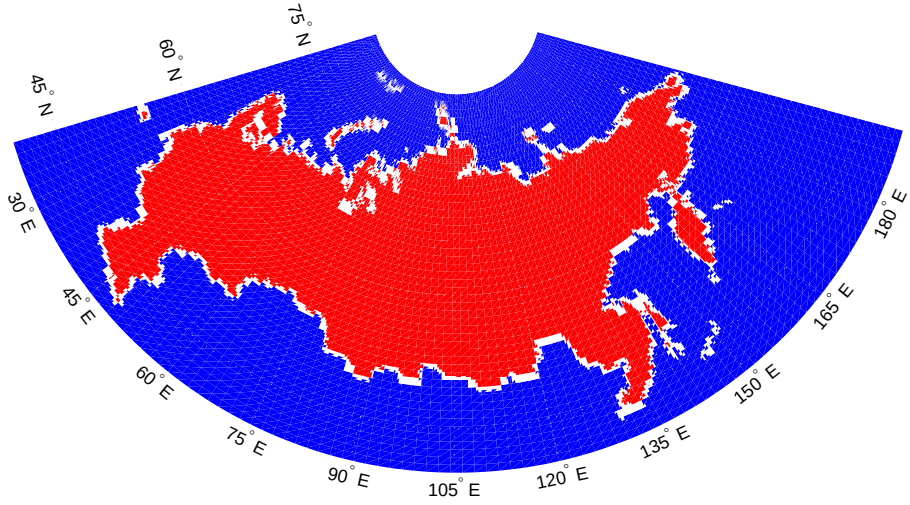
[map,maplegend] = vec2mtx(lat,lon,scale,latlim,lonlim) uses the two element vector latitude and longitude limits to define the extent of the map. If omitted, the limits are computed automatically.

[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

vfwdtran

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.

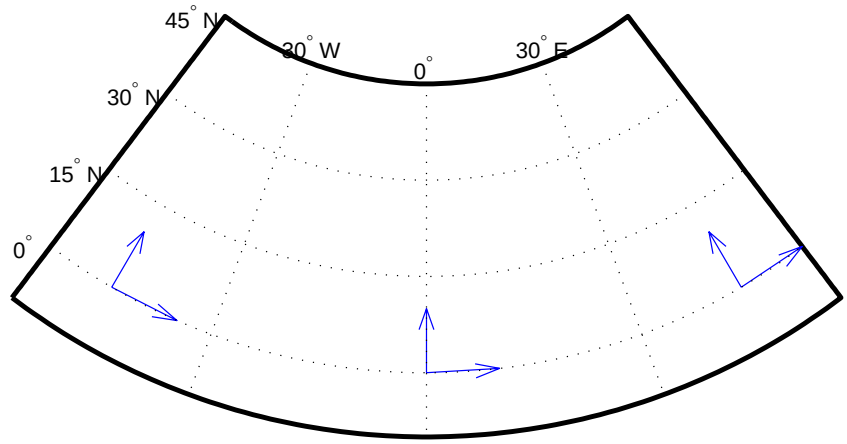
th = vfwdtran(mstruct,lat,lon,az) uses the map projection defined by the input `mstruct` to compute the map projection.

[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

Purpose

Compute areas visible from a point on a digital elevation map.

Syntax

```
[vismap,vismaplegend] = viewshed(map,maplegend,lat1,lon1)
[..] = viewshed(map,maplegend,lat1,lon1,oalt)
[..] = viewshed(map,maplegend,lat1,lon1,oalt,talt)
[..] = viewshed(map,maplegend,lat1,lon1,oalt,talt,oaltopt)
[..] = viewshed(map,maplegend,lat1,lon1,oalt,talt,oaltopt,taltopt)
[..] =
   viewshed(map,maplegend,lat1,lon1,oalt,talt,oaltopt,taltopt,actu
   alradius)
[..] =
   viewshed(map,maplegend,lat1,lon1,oalt,talt,oaltopt,taltopt,actu
   alradius,effectiveradius)
```

Description

[vismap,vismaplegend] = viewshed(map,maplegend,lat1,lon1) computes areas visible from a point on a digital elevation map. The elevations are provided as a regular matrix map containing elevations in units of meters. The observer location is provided as scalar latitude and longitude in units of degrees. The resulting vismap contains ones at the surface locations visible from the observer location, and zeros where the line of sight is obscured by terrain.

viewshed(map,maplegend,lat1,lon1,oalt) places the observer at the specified altitude in meters above the surface. This is equivalent to putting the observer on a tower. If omitted, the observer is assumed to be on the surface.

viewshed(map,maplegend,lat1,lon1,oalt,talt) checks for visibility of target points a specified distance above the terrain. This is equivalent to putting the target points on towers which do not obstruct the view. if omitted, the target points are assumed to be on the surface.

viewshed(map,maplegend,lat1,lon1,oalt,talt,oaltopt) controls whether the observer is at a relative or absolute altitude. If the observer altitude option is 'AGL', the observer altitude oalt is in meters above ground level. If oaltopt is 'MSL', oalt is interpreted as altitude above zero, or mean sea level. If omitted, 'AGL' is assumed.

viewshed(map,maplegend,lat1,lon1,oalt,talt,oaltopt,taltopt) controls whether the target points are at a relative or absolute altitude. If the target altitude option is 'AGL', the target altitude talt is in meters above ground level. If taltopt is 'MSL', talt is interpreted as altitude above zero, or mean sea level. If omitted, 'AGL' is assumed.

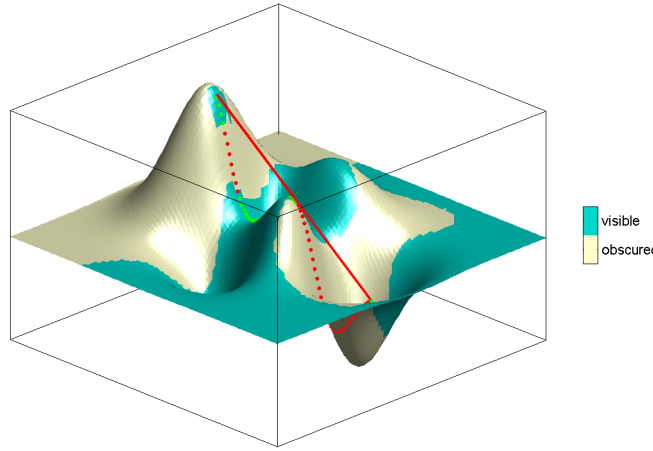
viewshed(map,maplegend,lat1,lon1,oalt,talt,oaltopt,taltopt,actualradius) does the visibility calculation on a sphere with the specified radius. If omitted, the radius of the earth in meters is assumed. The altitudes, elevations and the radius should be in the same units. This calling form is most useful for computations on bodies other than the Earth.

viewshed(map,maplegend,lat1,lon1,oalt,talt,oaltopt,taltopt,actualradius,effectiveradius) assumes a larger radius for propagation of the line of sight. This can account for the curvature of the signal path due to refraction in the atmosphere. For example, radio propagation in the atmosphere is commonly treated as straight line propagation on a sphere with 4/3rds the radius of the Earth. In that case the last two arguments would be R and 4/3*R_e, where R is the radius of the earth. Use Inf for flat Earth viewshed calculations. The altitudes, elevations and the radii should be in the same units.

Example

Compute visibility for point on the peaks map. Add the detailed information for the line of sight calculation between two point from `los2`.

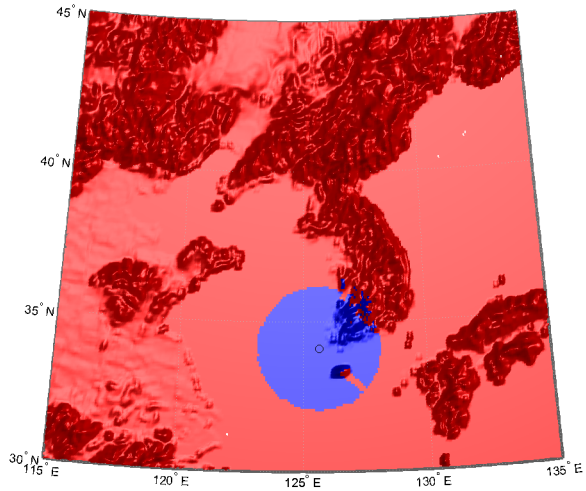
```
map = 500*peaks(100);
maplegend = [ 1000 0 0];
[lat1,lon1,lat2,lon2]=deal(-0.027,0.05,-0.093,0.042);

[vismap,vismapleg] = viewshed(map,maplegend,lat1,lon1,100);
[vis,visprofile,dist,z,lattrk,lontrk] =
los2(map,maplegend,lat1,lon1,lat2,lon2,100);

axesm('globe','geoid',almanac('earth','sphere','meters'))
meshm(vismap,vismapleg,size(map),map); axis tight
camposm(-10,-10,1e6); camupm(0,0)
colormap(flipud(summer(2))); brighten(0.75);
shading interp; camlight
h = lcolorbar({'obscured','visible'});
set(h,'Position',[.875 .45 .02 .1])

plot3m(lattrk([1;end]),lontrk([1; end]),z([1; end])+[100;
0],'r','linewidth',2)
plotm(lattrk(~visprofile),lontrk(~visprofile),z(~visprofile),'r
.','markersize',10)
plotm(lattrk(visprofile),lontrk(visprofile),z(visprofile),'g.',
'markersize',10)
```



Compute the surface areas visible by radar from an aircraft 3000 meters above the Yellow Sea. Assume that radio wave propagation in the atmosphere can be modeled as straight lines on a 4/3rds radius Earth. Display the visible areas as blue and the obscured areas as red. Drape the visibility colors on an elevation map, and use lighting to bring out the surface topography. The aircraft's radar can see out a certain radius on the surface of the ocean, but some ocean areas are shadowed by the island of Jeju-Do. Also some mountain valleys closer than the ocean horizon are obscured, while some mountain tops further away are visible.

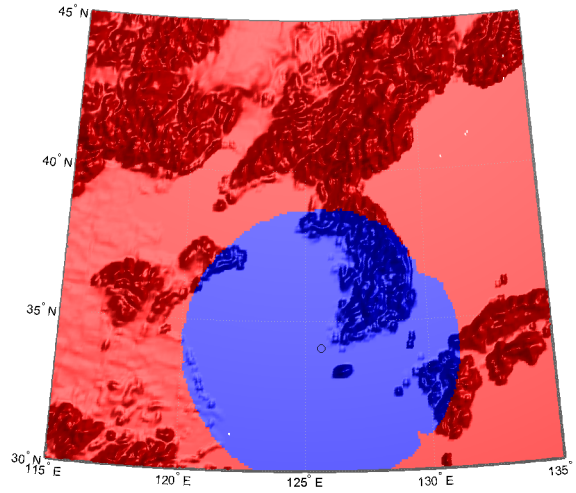
```
load korea
map(map<0) = -1;
figure
worldmap(map,maplegend,'ldem3d')
lat = 34.0931; lon = 125.6578; altobs = 3000; alttarg = 0;
plotm(lat,lon,'wo')
Re = almanac('earth','radius','m');
[vmap,vmapl] =
viewshed(map,maplegend,lat,lon,altobs,alttarg,'MSL','AGL',Re,4/
3*Re);
clmo surface
meshm(vmap,vmapl,size(map),map)
caxis auto; colormap([1 0 0; 0 0 1])
lighting phong; material metal
```

```
hidem(gca)
```



Over what area can the radar plane flying at an altitude of 3000 meters have line-of-sight to other aircraft flying at 5000 meters? Now the area is much larger. Some edges of the area are reduced by shadowing from Jeju-Do and the mountains on the Korean peninsula.

```
[vmap,vmapl] =
viewshed(map,maplegend,lat,lon,3000,5000,'MSL','MSL',Re,4/
3*Re);
clmo surface
meshm(vmap,vmapl,size(map),map)
material metal
lighting phong
```



See Also

`los2`	Line of sight visibility between two points in terrain

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.

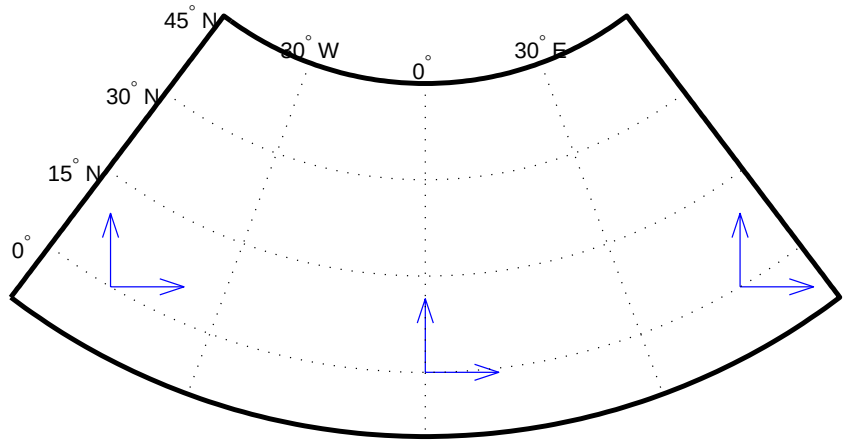
az = vinvtran(mstruct,x,y,th) uses the map projection defined by the input struct to compute the map projection.

[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

worldhi

Purpose

Return data from the 1:1,000,000 WORLDHI atlas data file.

Syntax

```
worldhi
s = worldhi(request)
s = worldhi(countryname,minarea)
s = worldhi(countryname,searchmethod)
s = worldhi(countryname,searchmethod,minarea)
s = worldhi(latlim,lonlim)
s = worldhi(latlim,lonlim,minarea)
```

Description

worldhi types a list of the atlas data variables in the `worldhi` MAT-file to the screens.

s = worldhi(*request*) returns the requested variable. Valid requests are '`POpatch`', '`POtext`', '`PPpoint`', '`PPtext`', and any country name in the `worldhi` database. Multiple countries can be requested by a string matrix or cell array of strings. The result is returned as a geographic data structure. This command can be used as an argument to other commands, such as `displaym(worldhi('POtext'))` or `displaym(worldhi('france'))`. The `worldhi` database contains country outlines and text originally compiled at a scale of 1:1,000,000.

s = worldhi(*countryname*,minarea) removes small islands from the output. Only polygon faces with an area greater than `minarea` (in units of square kilometers) are returned.

s = worldhi(*countryname*,*searchmethod*) controls the method used to match the country name input. Search method '`strmatch`' searches for matches at the beginning of the name, similar to the MATLAB `strmatch` function. Search method '`findstr`' searches within the name, similar to the MATLAB `findstr` function. Search method '`exact`' requires an exact match. If omitted, search method '`strmatch`' is assumed. The search is not case-sensitive.

s = worldhi(*countryname*,*searchmethod*,minarea) controls both the minumum area and the string matching method.

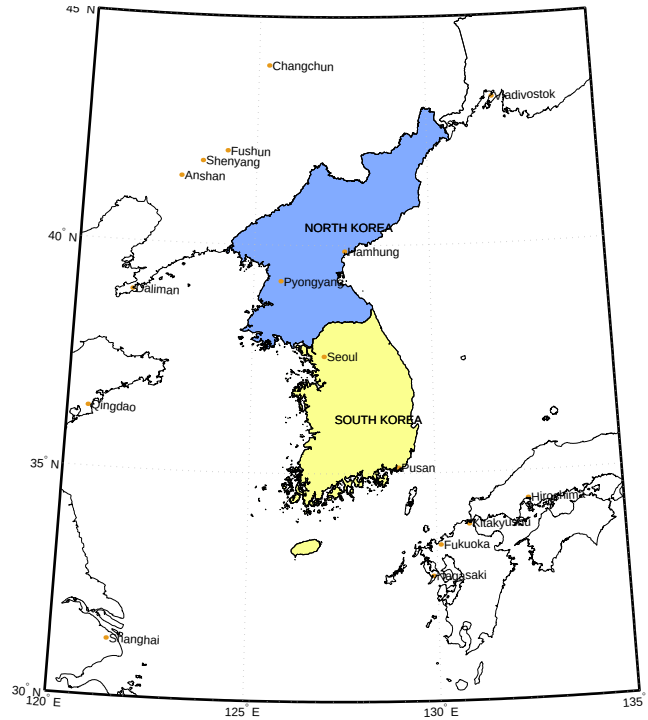
s = worldhi(latlim,lonlim) returns all countries within a quadrangle. `latlim` and `lonlim` are two-element vectors in units of degrees positive or negative from the prime meridian. Islands or noncontiguous parts of the countries whose bounding boxes fall completely outside the requested region are omitted from the output.

s = worldhi(latlim,lonlim,minarea) also removes small islands from the output. Only polygon faces with an area greater than `minarea` (in units of square kilometers) are returned.

Example

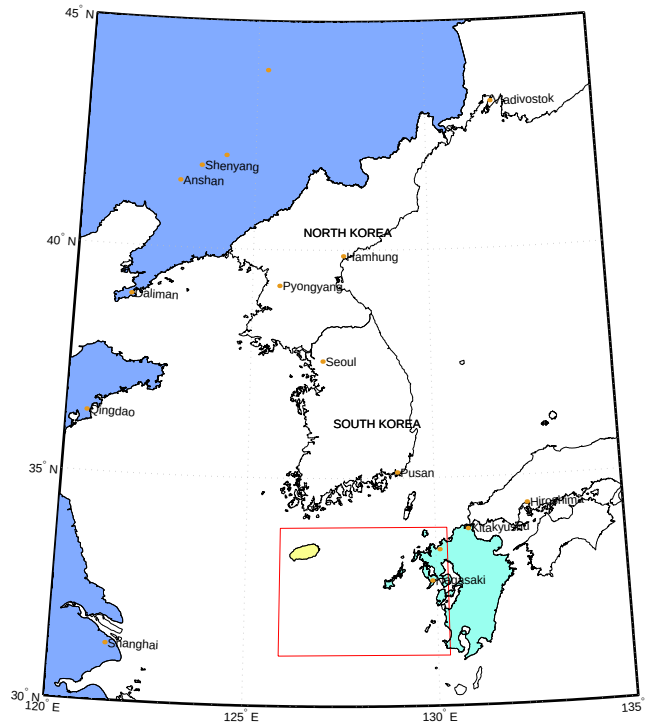
Make a basemap map of the Korean peninsula using lines and display the two Koreas from the worldhi database.

```
worldmap korea
s = worldhi('korea');
h = displaym(s);
polcmap
zdatam('allpatch',-1)
```



Delete the Korean patches, and display data from the worldhi database that falls within some geographic limits. Mainland China is displayed because its bounding box encompasses the requested quadrangle.

```
delete(h)

latlim = [31.1 33.9]';
lonlim = [125.9 130.3]';
lat = [latlim; flipud(latlim); latlim(1)];
lon = [lonlim(1); lonlim; flipud(lonlim)];
plotm(lat,lon,'r')

s = worldhi(latlim,lonlim,100);
h = displaym(s);
```



See Also

`worldlo`	Return data from the `worldlo` atlas data file
`coast`	Return vector coastline data
`usahi`	Return data from the `usahi` atlas data file
`usalo`	Return data from the `usalo` atlas data file
`worldhi.mat`	`worldhi` atlas data MAT-file
`displaym`	Project data in a geographic data structure

worldlo

Purpose

Returns data from the `worldlo` atlas data file

Syntax

```
worldlo
s = worldlo(request)
```

Description

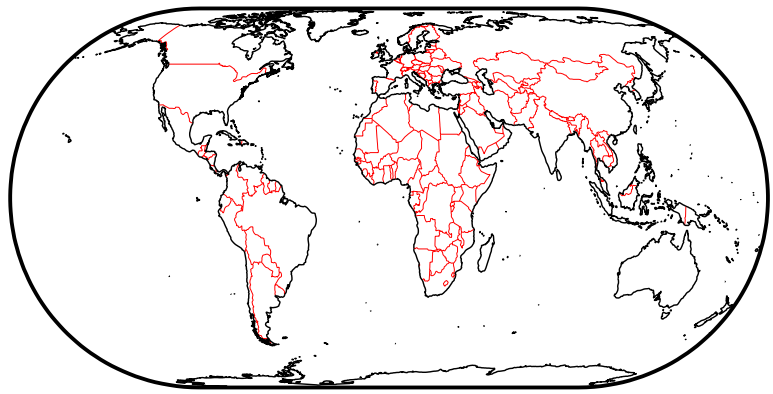
worldlo types a list of the atlas data variables in the `worldlo` MAT-file to the screen.

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 regionony
worldmap(region,type)
worldmap(latlim,lonlim)
worldmap(latlim,lonlim,type)
worldmap(map,maplegend)
worldmap(map,maplegend,type)
worldmap(datasource,...)
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 *regionony* 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(*regionony*,*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. Type 'mesh' and 'meshonly' display the matrix map as a colored surface in the z=0 plane with the default colormap.

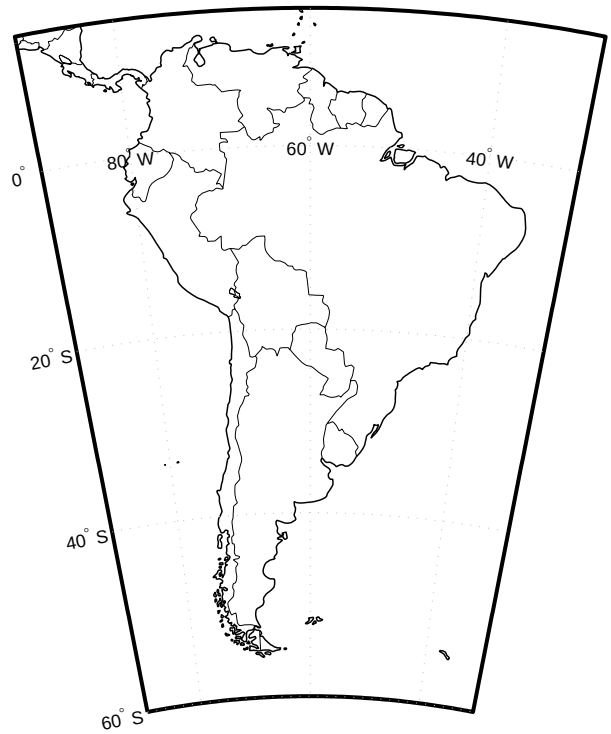
Type 'dem' and 'demonly' apply the digital elevation colormap. Type 'dem3d' and 'dem3donly' display a 3-dimensional surface. Type 'lmesh3d', 'lmesh3donly', 'ldem3d' and 'ldem3donly' apply lighting effects to the surfaces.

worldmap('hi',...) or worldmap('lo',...) enforces the use of high or low resolution data in the map. If omitted, `worldmap` will select the appropriate data set. For a paper scale of 1:30,000,000 or greater, the `worldmap` database is used. For smaller scales, the `worldmap` database is used. When using the high resolution data, `worldmap` will not include islands smaller than an appropriate threshhold size. Use worldmap('allhi',...) to ensure that all islands are included, regardless of size.

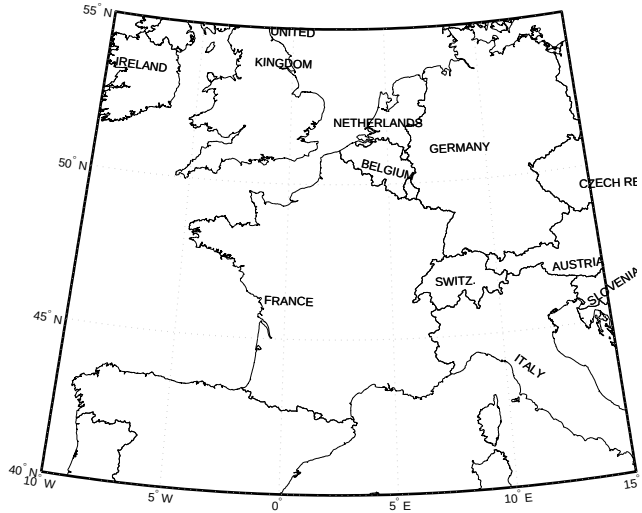
h = worldmap(...) returns the handle of the map axes.

Example

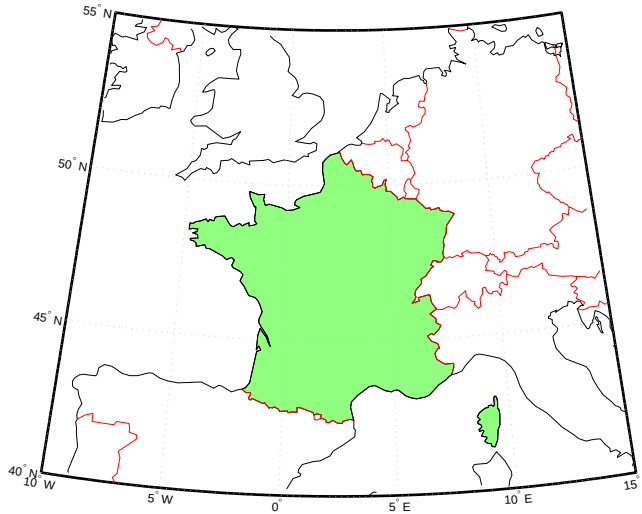
```
worldmap 'south america'
```



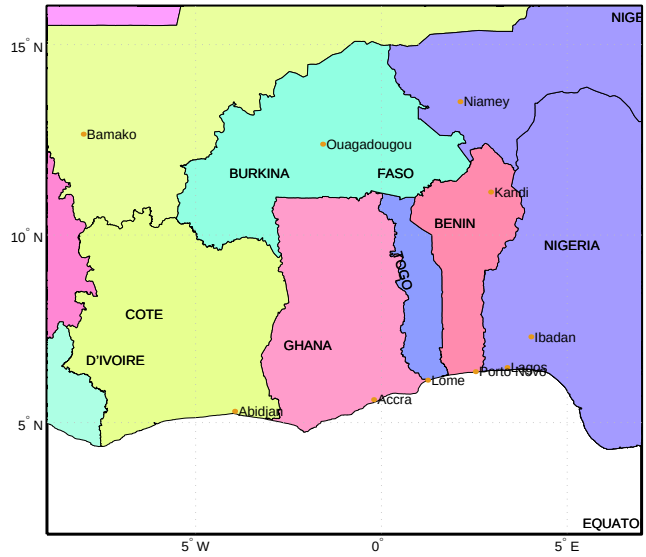
```
figure; worldmap france
```



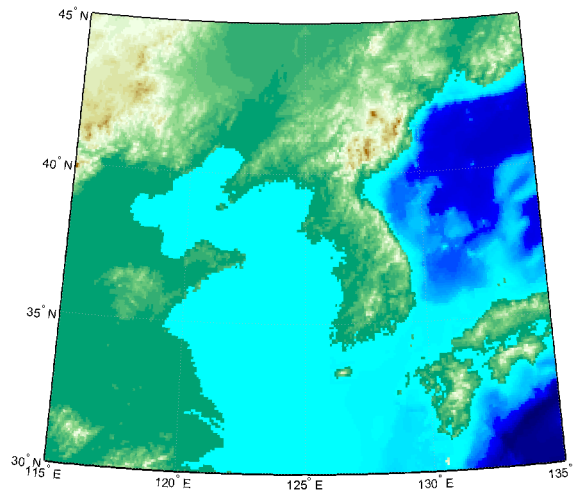

```
figure;
worldmap('lo','franceonly','patch')
displaym(worldlo('POline'))
```



```
figure; worldmap([2 16],[-9 7],'patch');
```




```
figure; load korea; worldmap(map,maplegend,'dem');
brighten(.5)
```



Remarks

`worldmap` 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 `tightmap` again or `axis auto`.

Use `hidem(gca)` to remove the axes box and `hidem` or `clmo` to remove other unwanted graphic objects. `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
`worldhi`	Returns data from the `worldhi` 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

10

Projections Reference

Alphabetical List of Map Projections

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.

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.

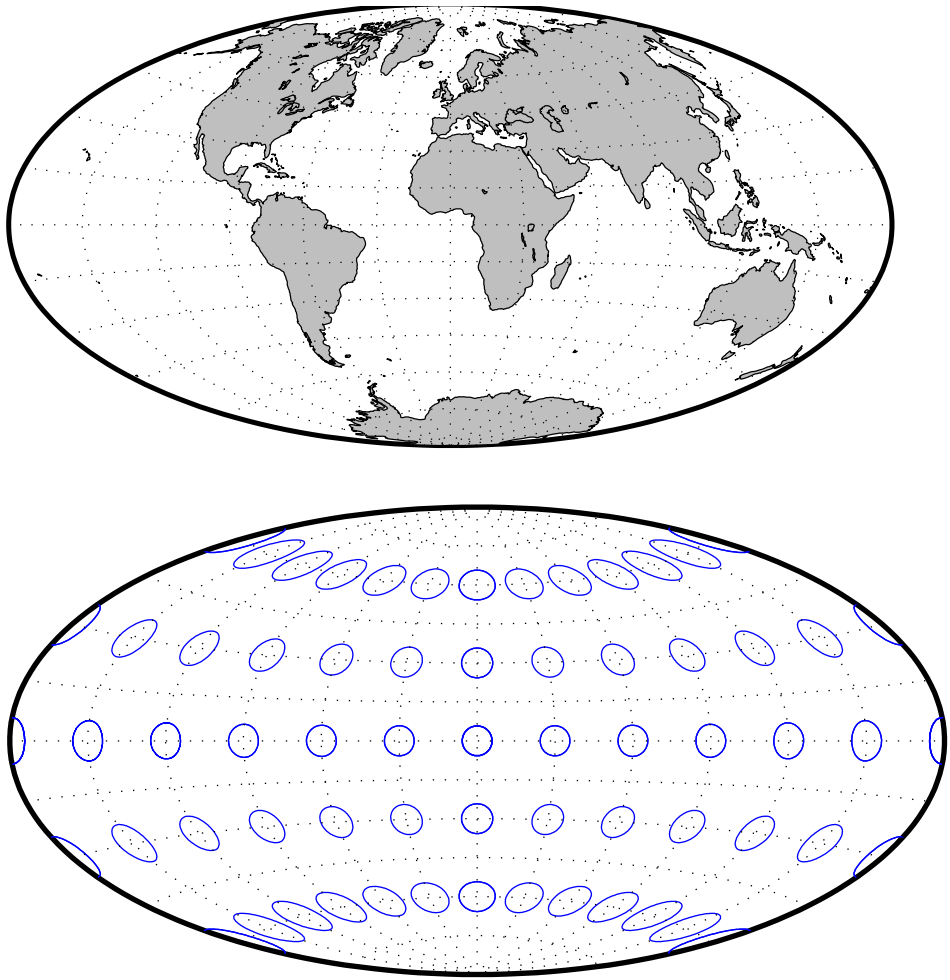
Symmetry: About the Equator and central meridian.

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.

Aitoff Projection



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.

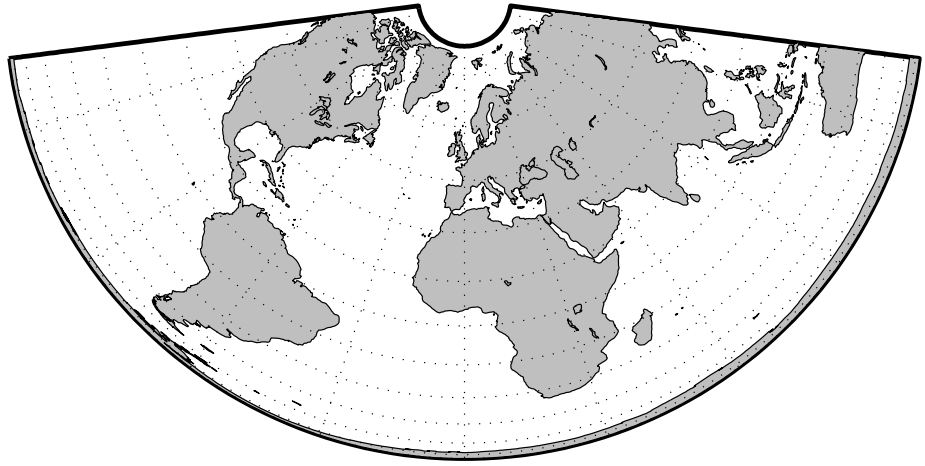
Features This is an equal-area projection. Scale is true along the one or two selected standard parallels. Scale is constant along any parallel; the scale factor of a meridian at any given point is the reciprocal of that along the parallel to preserve equal-area. This projection is free of distortion along the standard parallels. Distortion is constant along any other parallel. This projection is neither conformal nor equidistant.

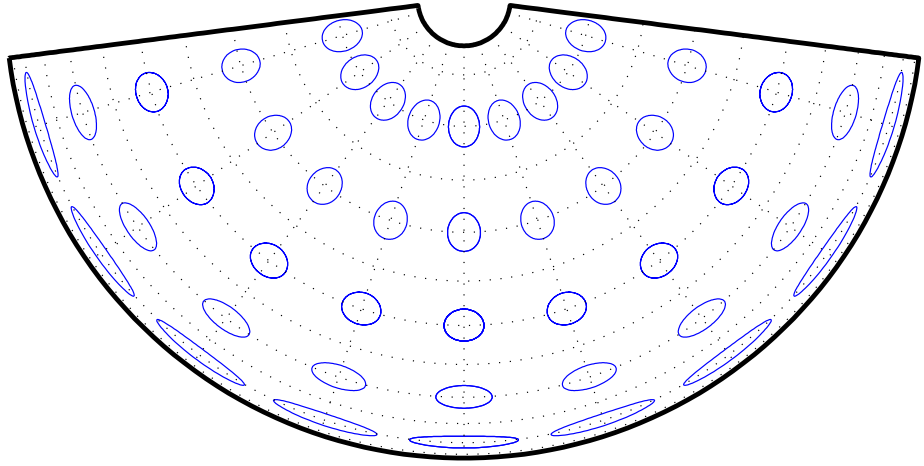
Parallels The cone of projection has interesting limiting forms. If a pole is selected as a single standard parallel, the cone is a plane and a Lambert Azimuthal Equal-Area projection results. If two parallels are chosen, not symmetric about the Equator, then a Lambert Equal-Area Conic projection results. 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.

Albers Equal-Area Conic Projection





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.

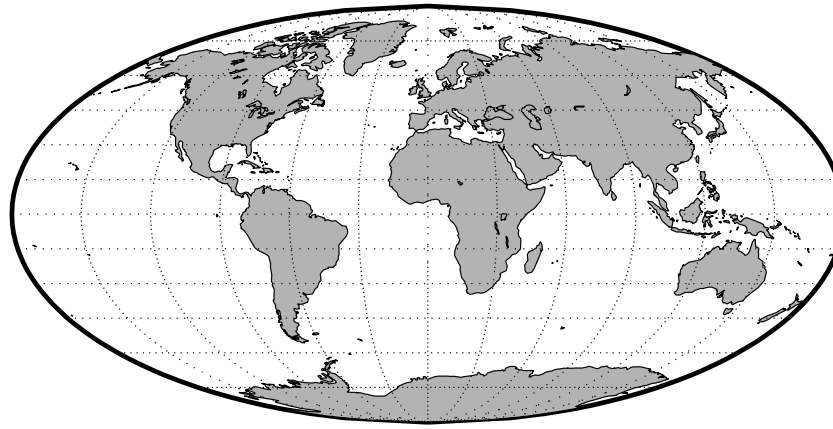
Features Scale is constant along any parallel or pair of parallels equidistant from the Equator, as well as along the central meridian. The Equator is free of angular distortion. This projection is not equal-area, equidistant, or conformal.

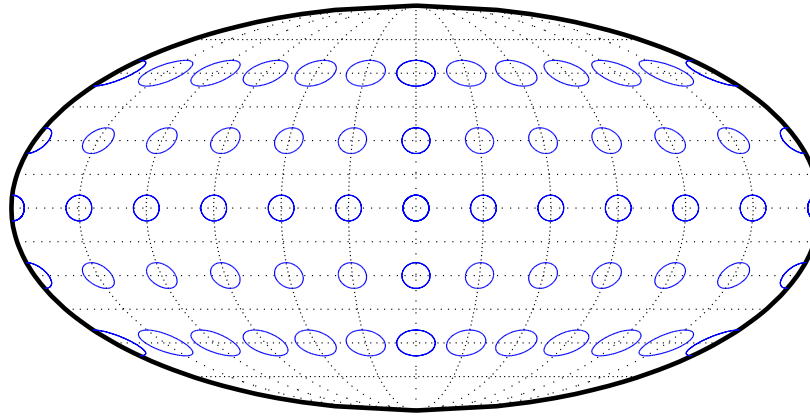
Parallels There is no standard parallel for this projection.

Remarks This projection was first described in 1524 by Peter Apian (or Bienewitz).

Limitations This projection is available for the spherical geoid only.

Apianus II Projection





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.

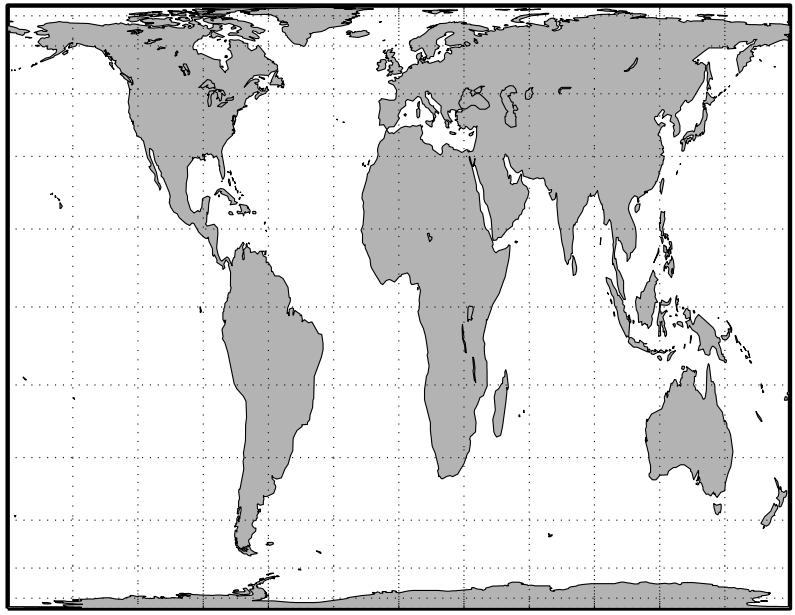
Symmetry: About any meridian or the Equator.

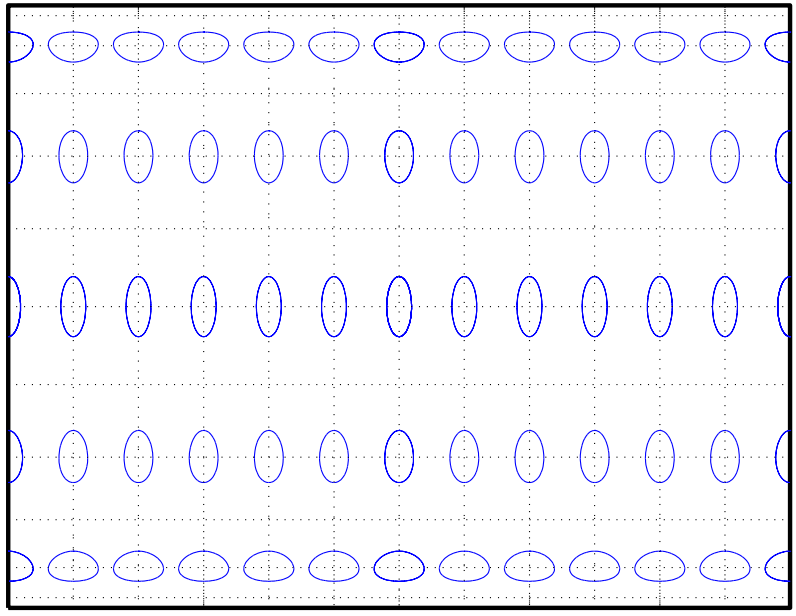
Features This is an orthographic projection onto a cylinder secant at the 50° parallels. It is equal-area, but distortion of shape increases with distance from the standard parallels. Scale is true along the standard parallels and constant between two parallels equidistant from the Equator. This projection 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.

Balthasart Cylindrical Projection





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.

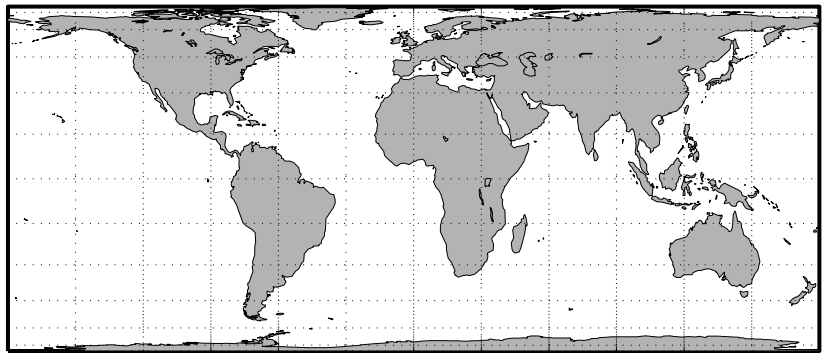
Symmetry: About any meridian or the Equator.

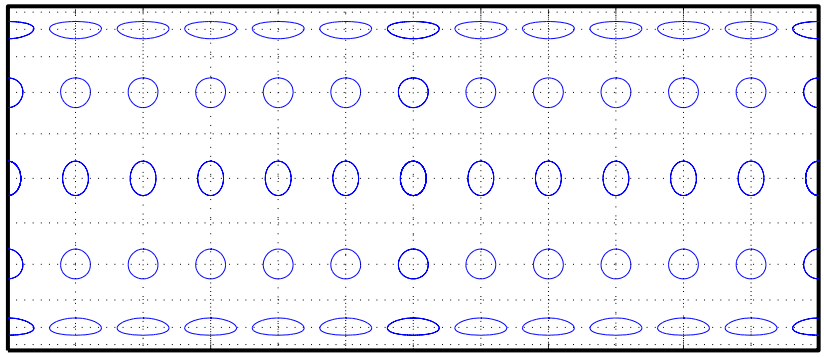
Features This is an orthographic projection onto a cylinder secant at the 30° parallels. It is equal-area, but distortion of shape increases with distance from the standard parallels. Scale is true along the standard parallels and constant between two parallels equidistant from the Equator. This projection 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.

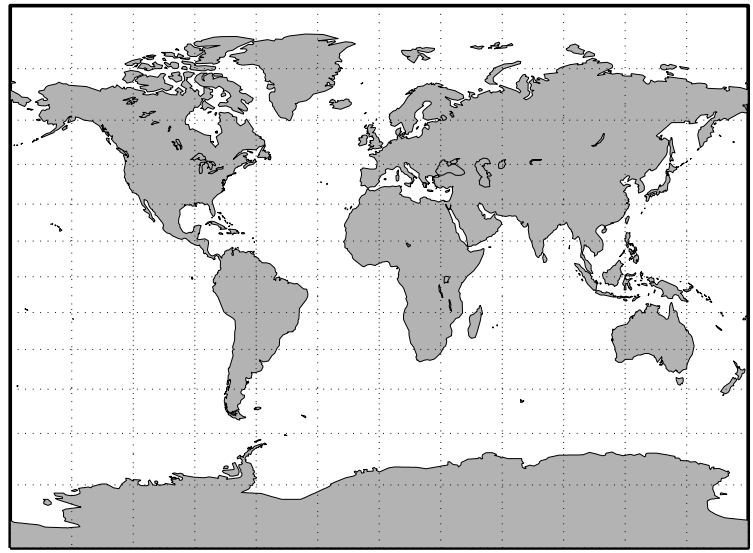
Features This is a perspective projection from a point on the Equator opposite a given meridian onto a cylinder secant at the 30° parallels. It is not equal-area, equidistant, or conformal. Scale is true along the standard parallels and constant between two parallels equidistant from the Equator. There is no distortion along the standard parallels, but it increases moderately away from these parallels, becoming severe at the poles.

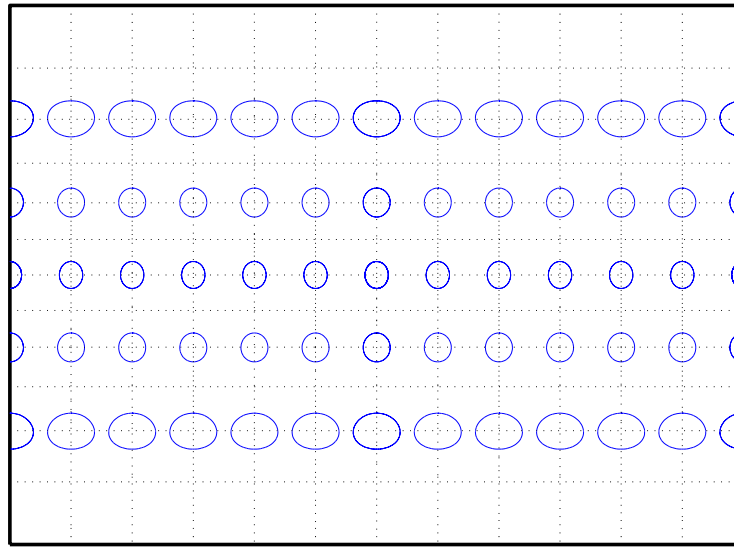
Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, the standard parallel is by definition fixed at 30°.

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.

Bolshoi Sovietskii Atlas Mira Projection





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.

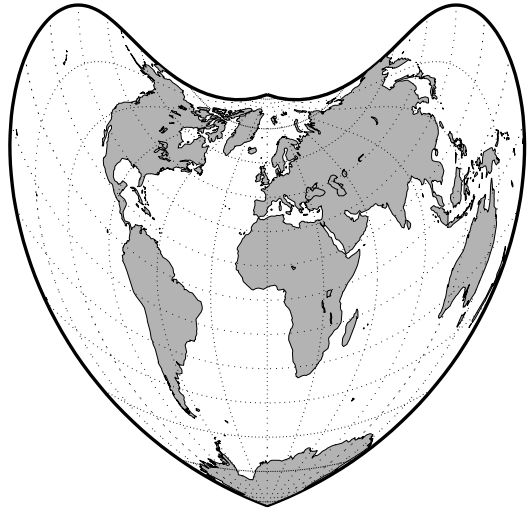
Symmetry: About the central meridian.

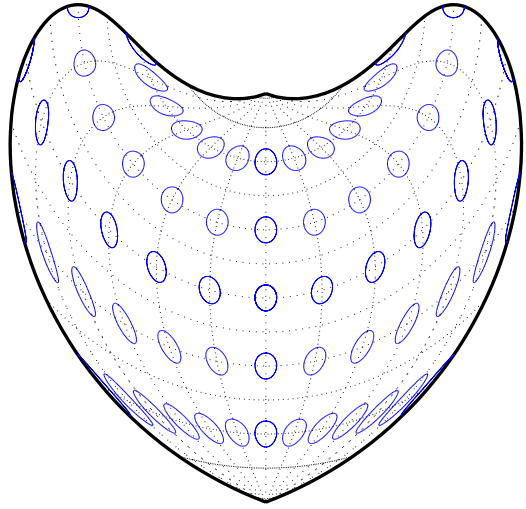
Features This is an equal-area projection. The curvature of the standard parallel is identical to that on a cone tangent at that latitude. The central meridian and the central parallel are free of distortion. This projection is not conformal.

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.

Bonne Projection





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.

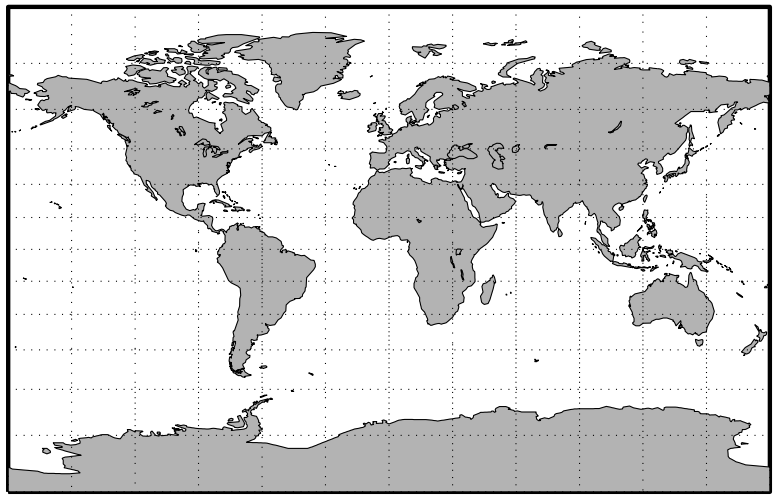
Features This is an perspective projection from a point on the Equator opposite a given meridian onto a cylinder secant at standard parallels. It is not equal-area, equidistant, or conformal. Scale is true along the standard parallels and constant between two parallels equidistant from the Equator. There is no distortion along the standard parallels, but it increases moderately away from these parallels, becoming severe at the poles.

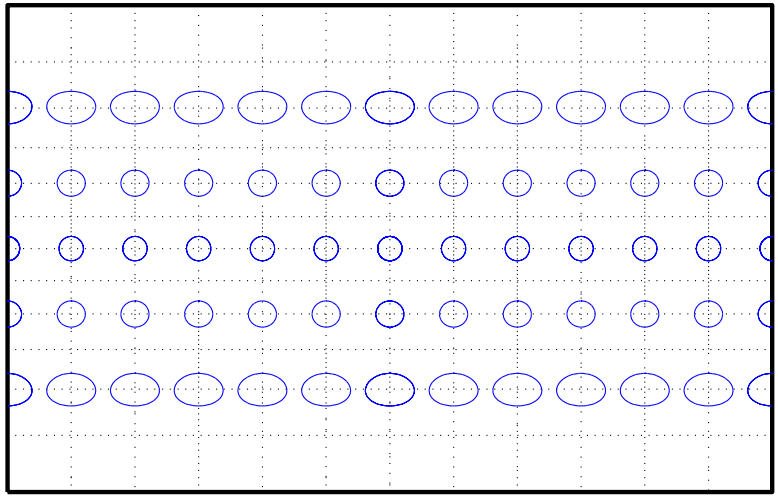
Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, any latitude may be chosen; the default is arbitrarily set to 0°.

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.

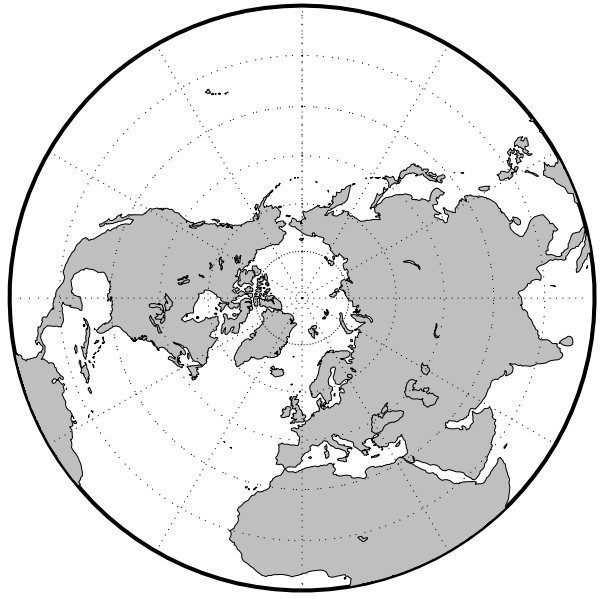
Features This is a harmonic mean between a Stereographic and Lambert Equal-Area Azimuthal projection. It is not equal-area, equidistant, or conformal. There is no point at which scale is accurate in all directions. The primary feature of this projection is that it is minimum error – distortion is moderate throughout.

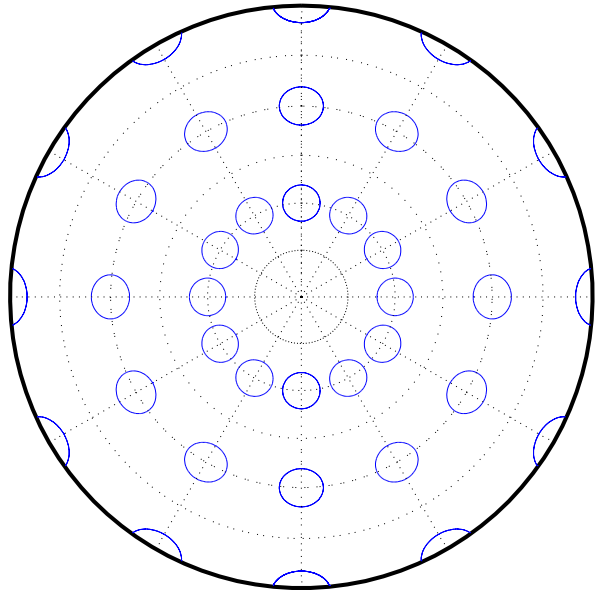
Parallels There are no standard parallels for azimuthal projections.

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.

Breusing Harmonic Mean Projection





Briesemeister Projection

Classification Modified Azimuthal

Syntax `bries`

Graticule Meridians: Central meridian is straight. Other meridians are complex curves.

Parallels: Complex curves.

Poles: Points.

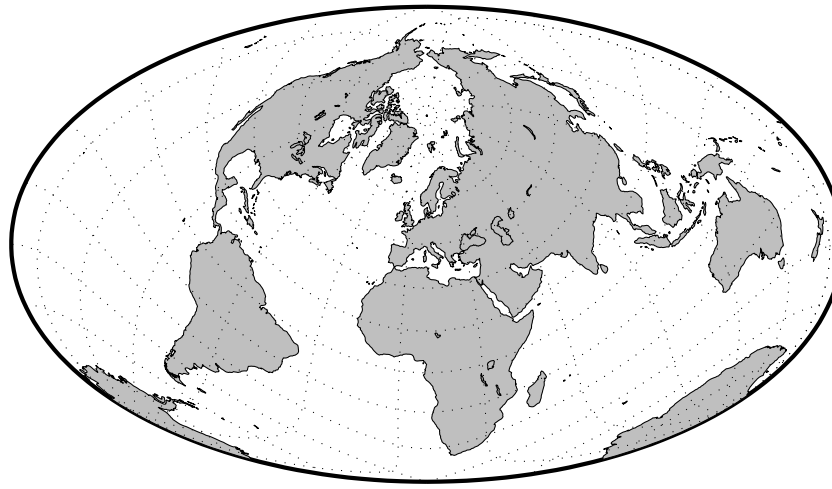
Symmetry: About the central meridian.

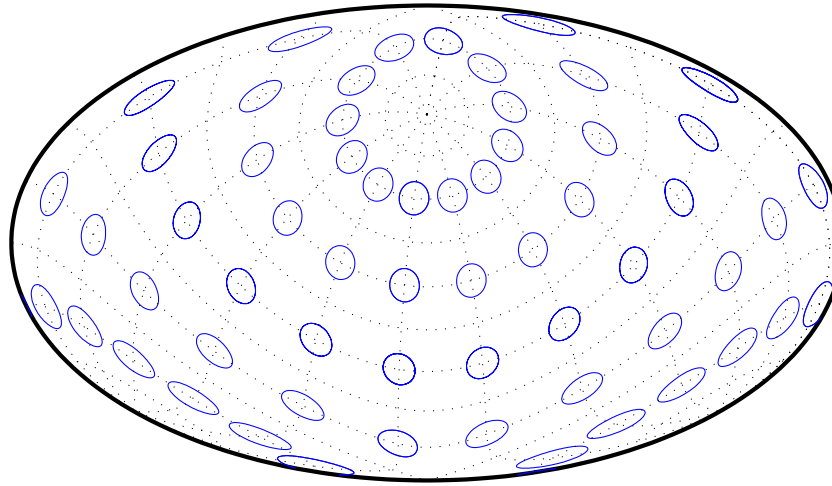
Features This equal-area projection groups the continents about the center of the projection. The only point free of distortion is the center point. Distortion of shape and area are moderate throughout.

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.

Briesemeister Projection





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.

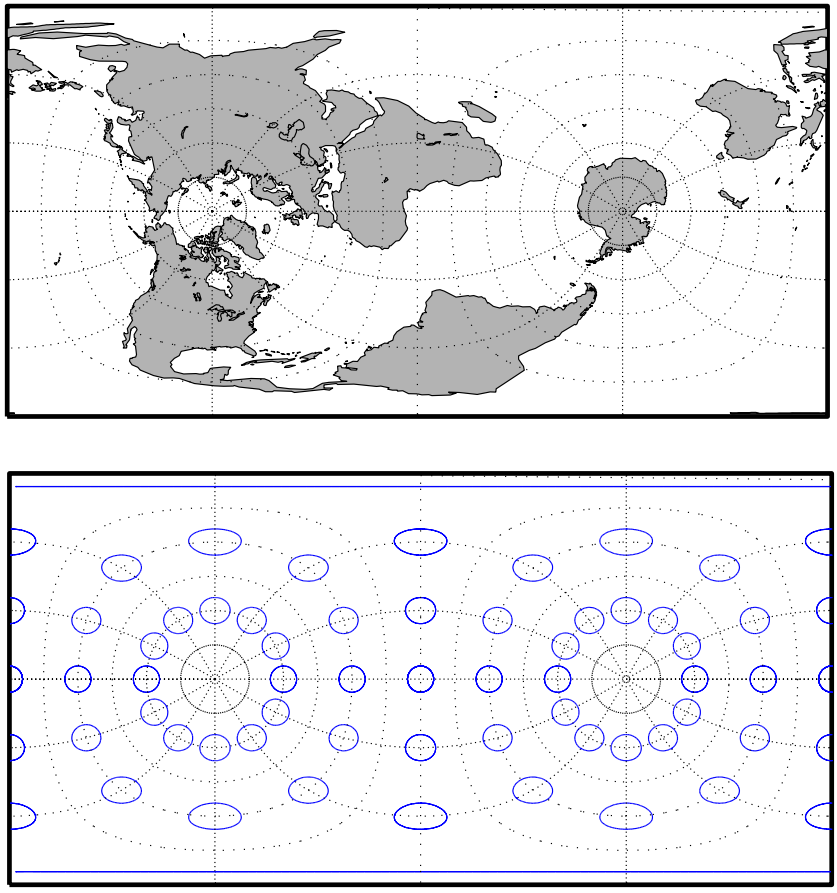
Symmetry: About any straight meridian or the Equator.

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.

Cassini Cylindrical Projection



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.

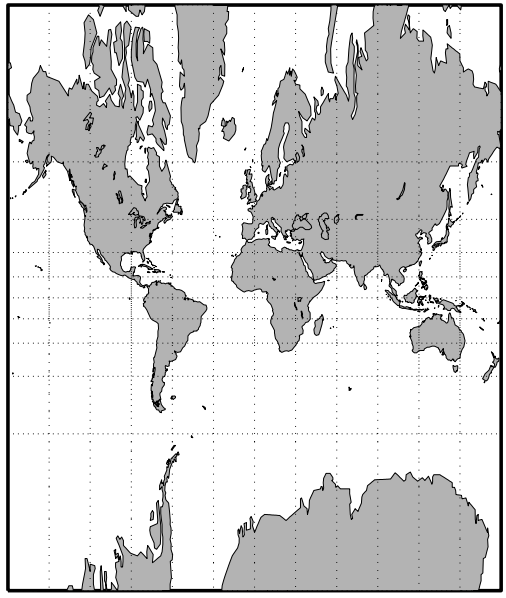
Features This is a perspective projection from the center of the Earth onto a cylinder tangent at the Equator. It is not equal-area, equidistant, or conformal. Scale is true along the Equator and constant between two parallels equidistant from the Equator. Scale becomes infinite at the poles. There is no distortion along the Equator, but it increases rapidly away from the Equator.

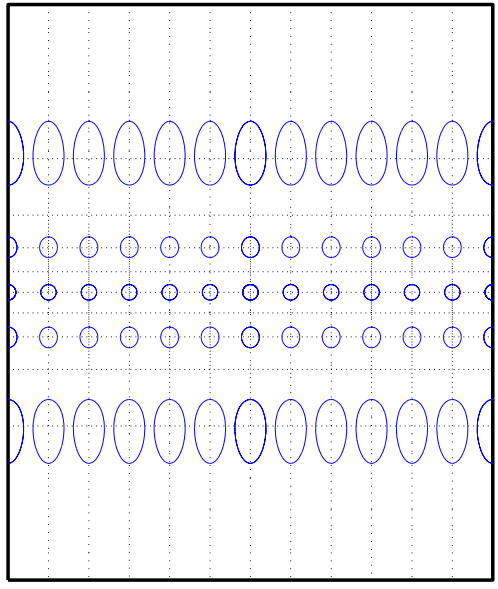
Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, the standard parallel is by definition fixed at 0°.

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.

Central Cylindrical Projection





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.

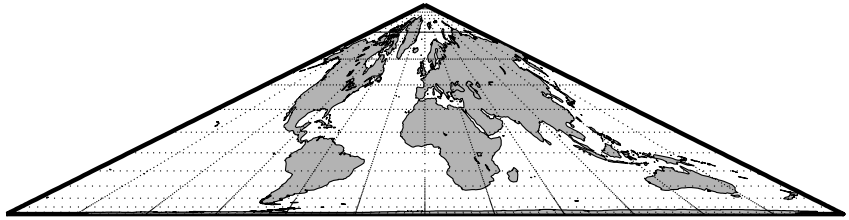
Symmetry: About the central meridian.

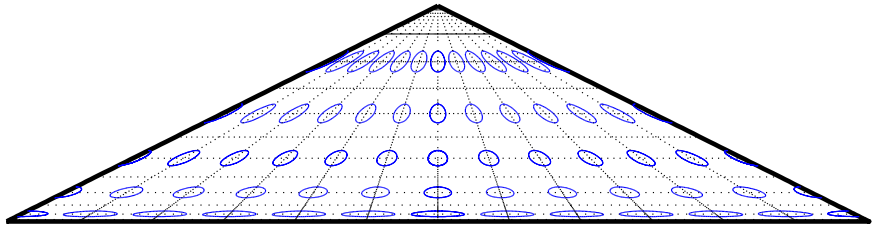
Features This is a novelty projection showing a straight-line, equal-area graticule. Scale is true along the 15 51'Nparallel, constant along any parallel, and *different* for any pair of parallels. Distortion is severe in many regions, and is only absent at 15 51'N on the central meridian. This projection 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.

Collignon Projection





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.

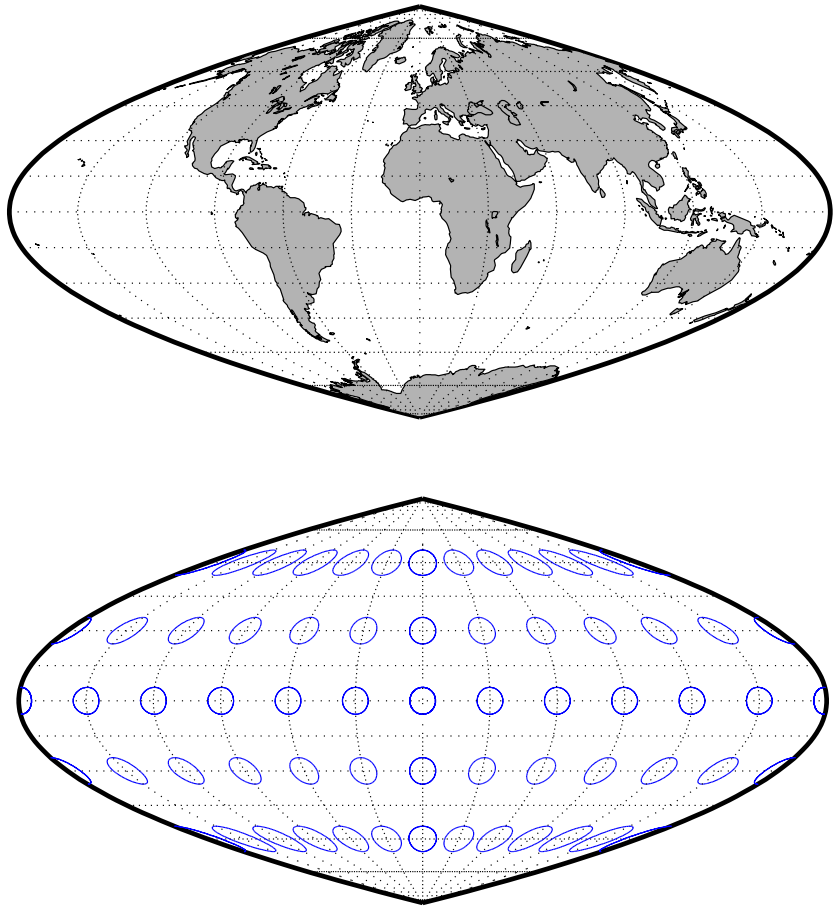
Symmetry: About the central meridian or the Equator.

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.

Craster Parabolic 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.

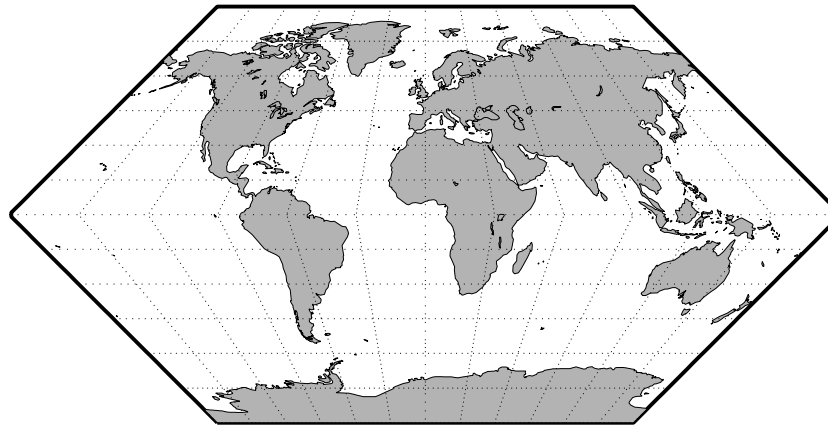
Features Scale is true along the 47 10' parallels and is constant along any parallel, between any pair of parallels equidistant from the Equator, and along any given meridian. It is not free of distortion at any point, and the break at the Equator introduces excessive distortion there; regardless of the appearance here, the Tissot indicatrices are of indeterminate shape along the Equator. This novelty projection is not equal-area or conformal.

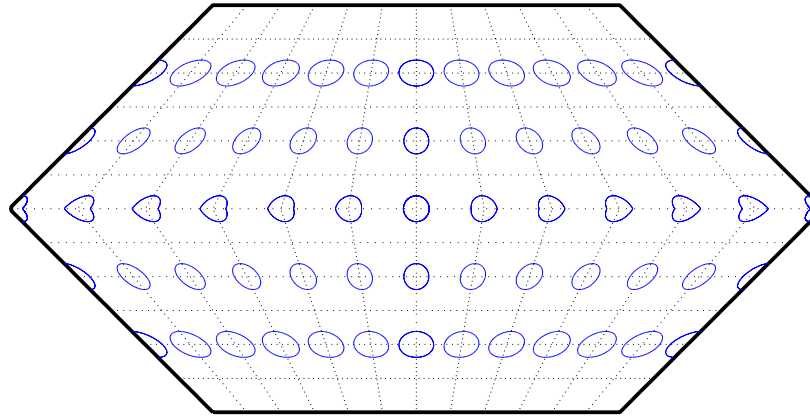
Parallels For this projection, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. The standard parallel is by definition fixed at 47 10'.

Remarks This projection was presented by Max Eckert in 1906.

Limitations This projection is available for the spherical geoid only.

Eckert I 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.

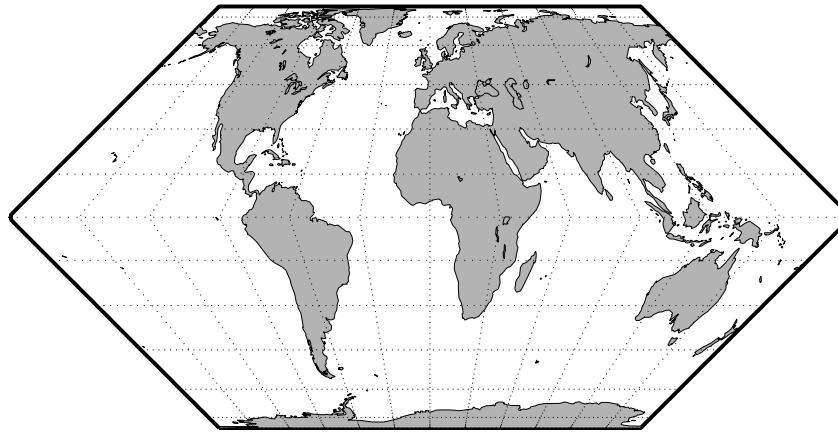
Symmetry: About the central meridian or the Equator.

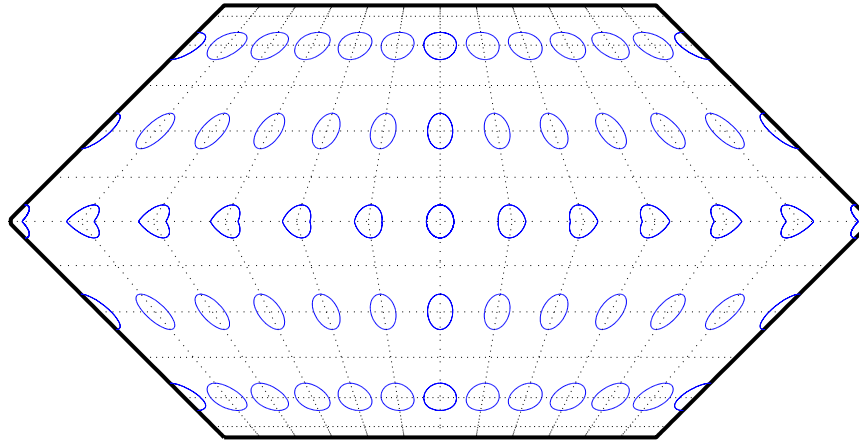
Features This is an equal-area projection. Scale is true along the 55 10' parallels and is constant along any parallel and between any pair of parallels equidistant from the Equator. It is not free of distortion at any point except at 55 10'N and S along the central meridian; the break at the Equator 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 II 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.

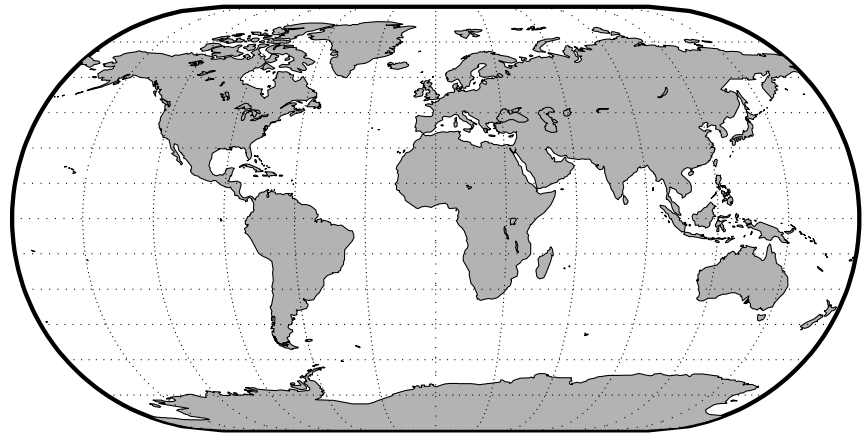
Features Scale is true along the 35 58' parallels and is constant along any parallel and between any pair of parallels equidistant from the Equator. No point is free of all scale distortion, but the Equator is free of angular distortion. This projection is not equal-area, conformal, or equidistant.

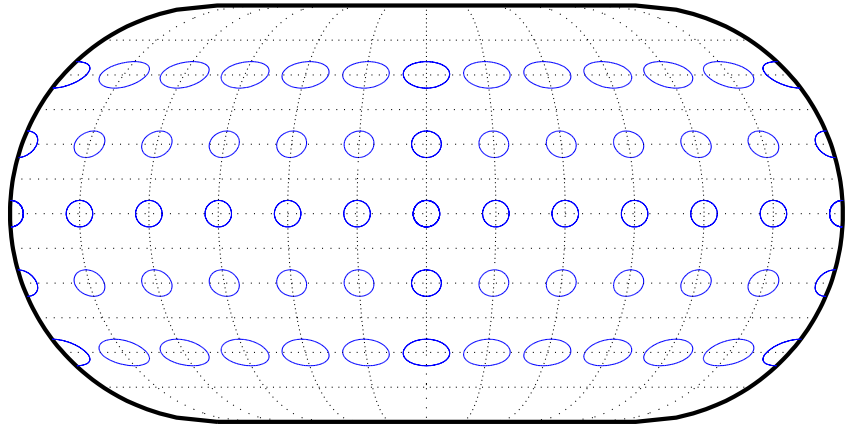
Parallels For this projection, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. The standard parallel is by definition fixed at 35 58'.

Remarks This projection was presented by Max Eckert in 1906.

Limitations This projection is available for the spherical geoid only.

Eckert III 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.

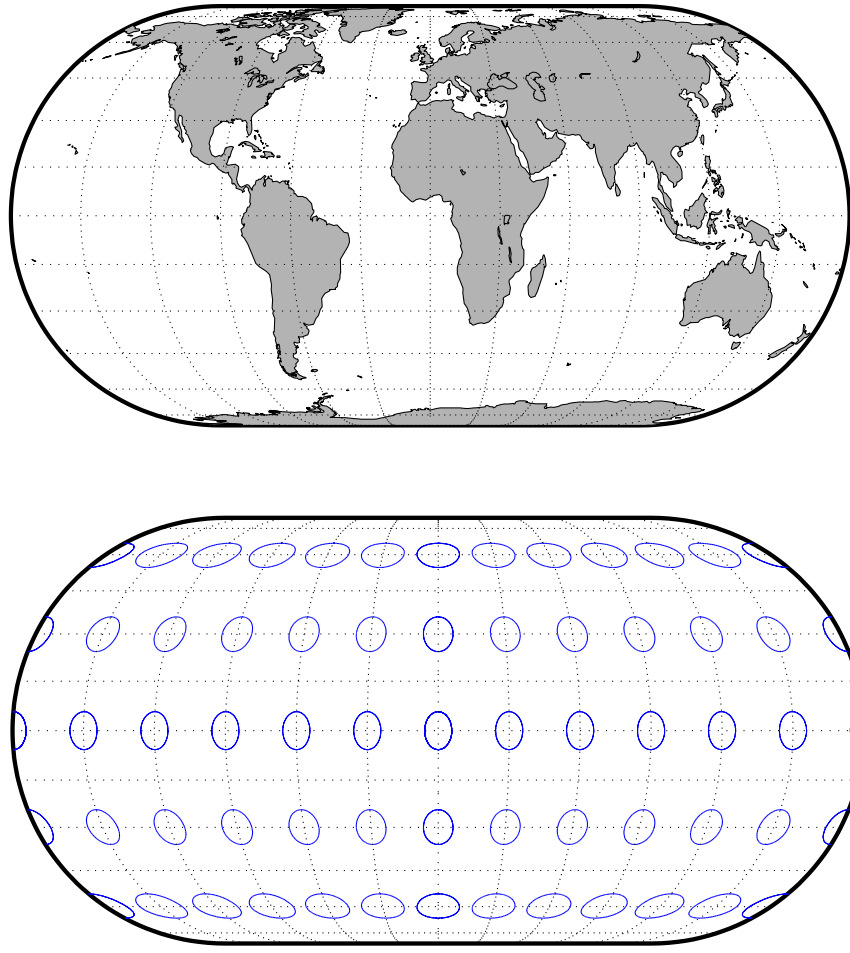
Symmetry: About the central meridian or the Equator.

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 IV 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.

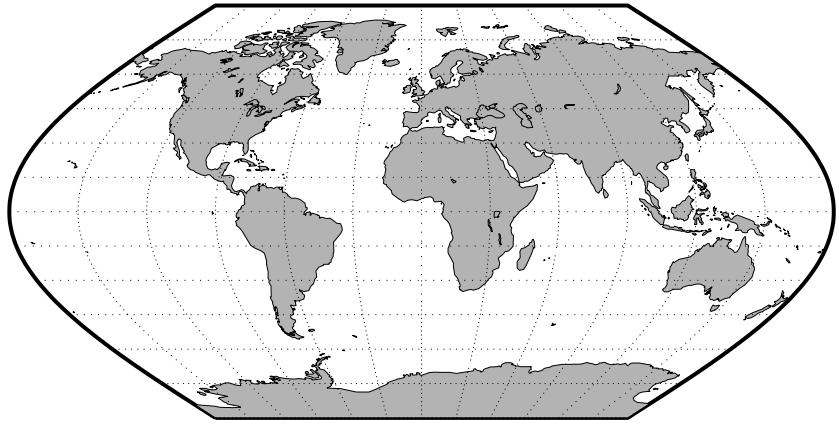
Features This projection is an arithmetic average of the *x* and *y* coordinates of the Sinusoidal and Plate Carrée projections. Scale is true along latitudes 37°55'N and S, and is constant along any parallel and between any pair of parallels equidistant from the Equator. There is no point free of all distortion, but the Equator is free of angular distortion. This projection is not equal-area, conformal, or equidistant.

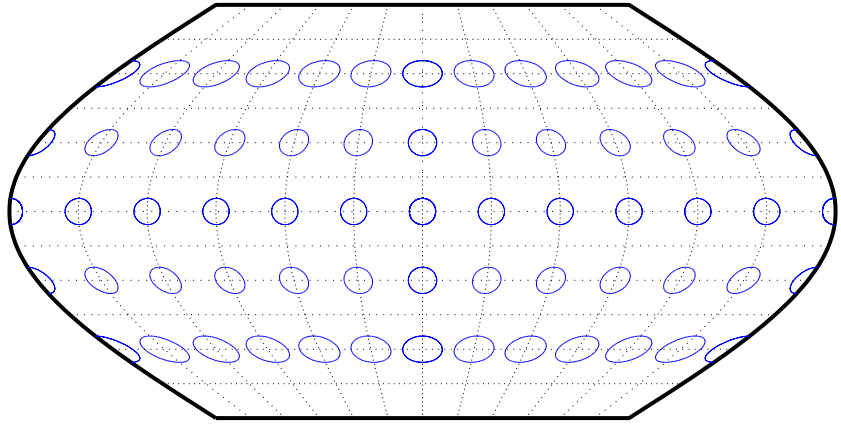
Parallels This projection has one standard parallel, which is by definition fixed at 0°.

Remarks This projection was presented by Max Eckert in 1906.

Limitations This projection is available for the spherical geoid only.

Eckert V 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.

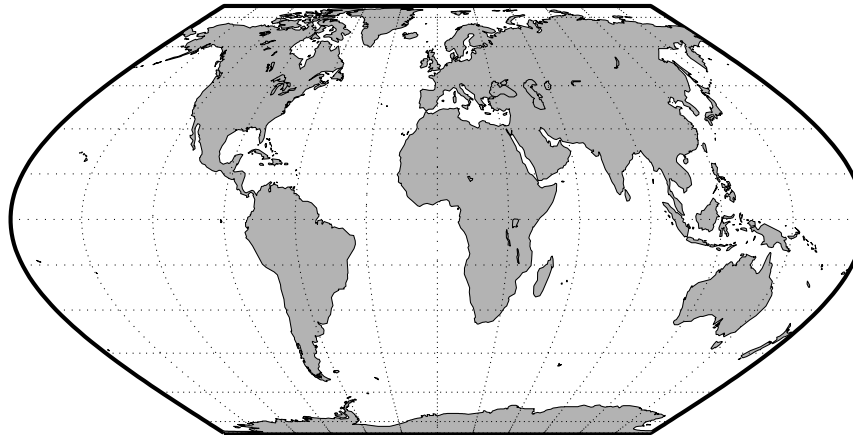
Symmetry: About the central meridian or the Equator.

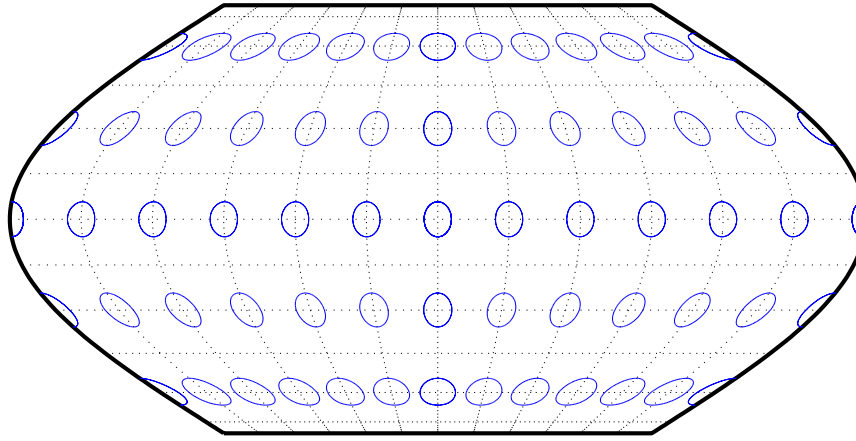
Features This is an equal-area projection. Scale is true along the 49 16' parallels and is constant along any parallel and between any pair of parallels equidistant from the Equator. It is free of distortion only at the two points where the 49 16' parallels intersect the central meridian. This projection 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.

Eckert VI Projection





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.

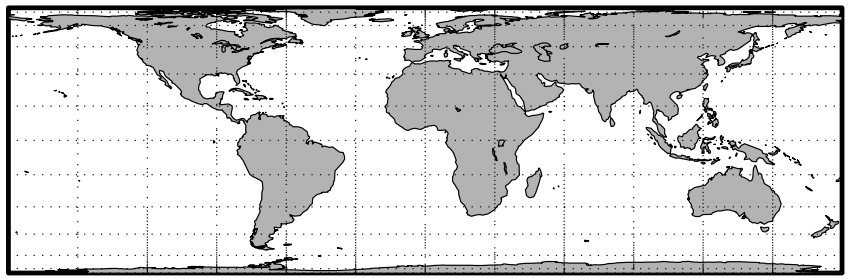
Symmetry: About any meridian or the Equator.

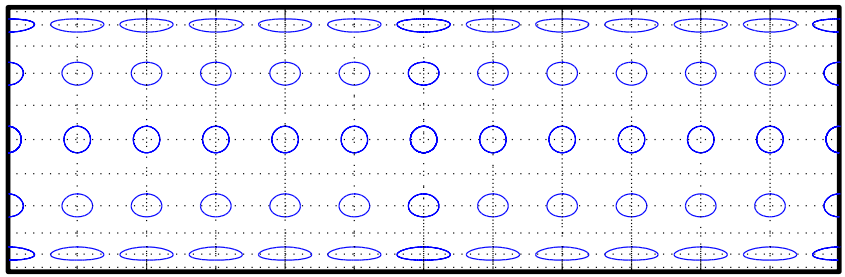
Features This is an orthographic projection onto a cylinder secant at the standard parallels. It is equal-area, but distortion of shape increases with distance from the standard parallels. Scale is true along the standard parallels and constant between two parallels equidistant from the Equator. This 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.

Equal-Area Cylindrical Projection





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.

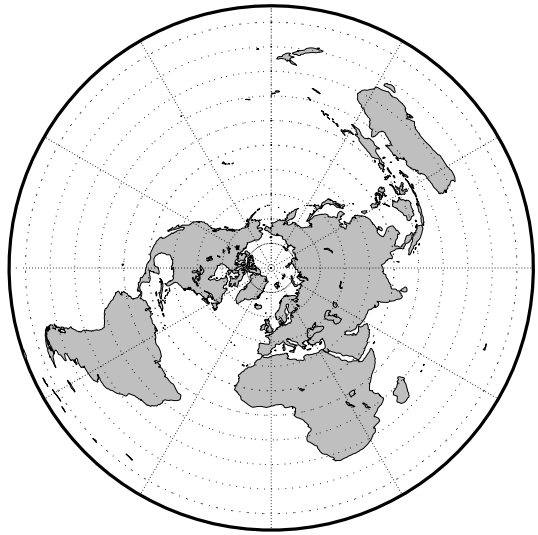
Features This is an equidistant projection. It is neither equal-area nor conformal. In the polar aspect, scale is true along any meridian. The projection is distortion free only at the center point. Distortion is moderate for the inner hemisphere, but it becomes extreme in the outer hemisphere.

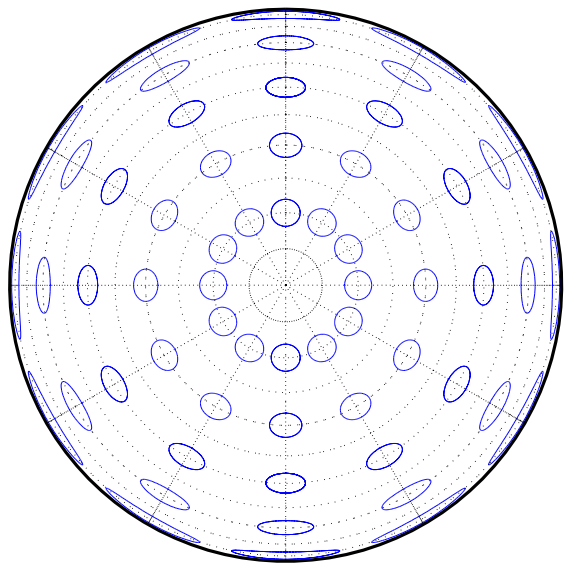
Parallels There are no standard parallels for azimuthal projections.

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 Azimuthal 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.

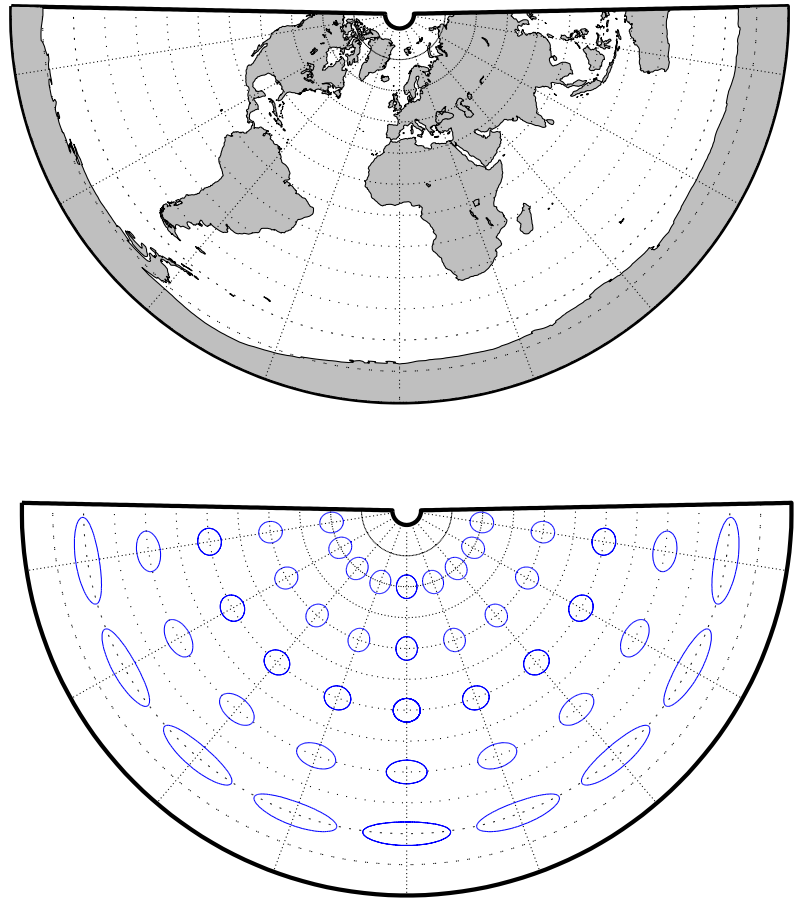
Features Scale is true along each meridian and the one or two selected standard parallels. Scale is constant along any parallel. This projection is free of distortion along the two standard parallels. Distortion is constant along any other parallel. This projection provides a compromise in distortion 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 Conic Projection



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.

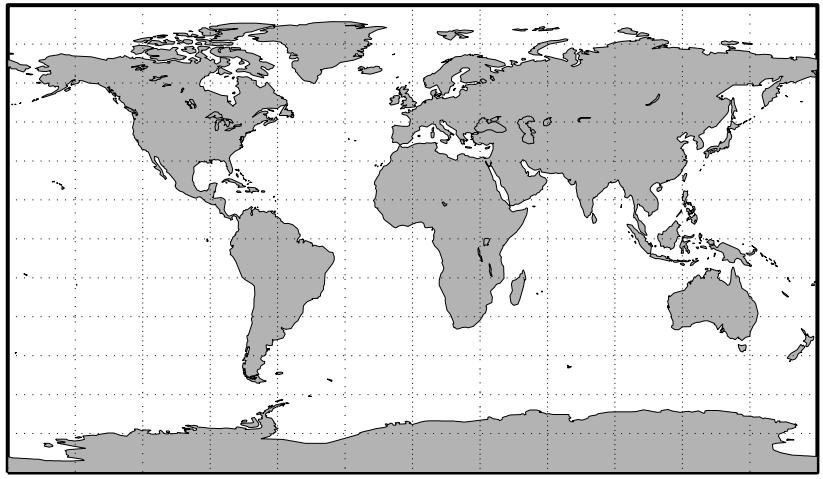
Symmetry: About any meridian or the Equator.

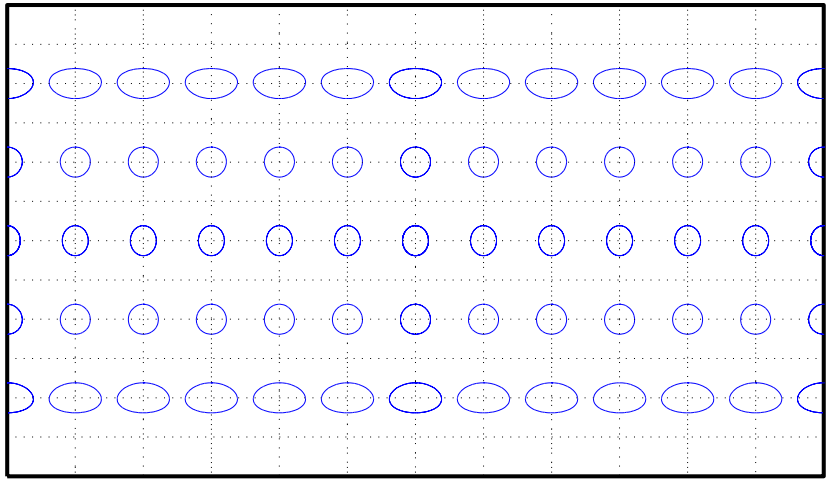
Features This is a projection onto a cylinder secant at the standard parallels. Distortion of both shape and area increase with distance from the standard parallels. Scale is true along all meridians (i.e., it is equidistant) and the standard parallels and is constant along any parallel and along the 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*.

Equidistant Cylindrical Projection





Classification Pseudocylindrical

Syntax `fournier`

Graticule Meridians: Equally spaced elliptical curves converging at the poles.

Parallels: Straight lines.

Poles: Points.

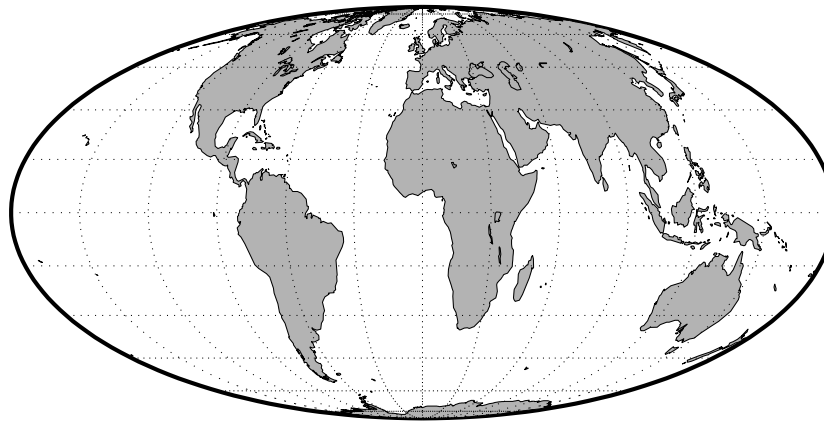
Symmetry: About the Equator and central meridian.

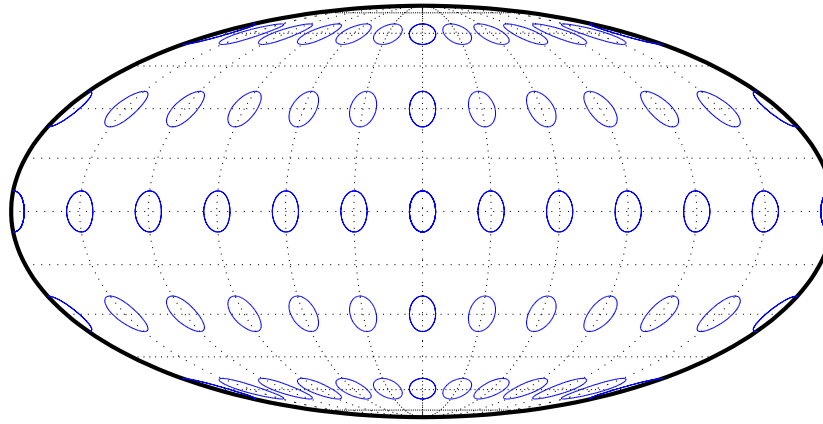
Features This projection is equal-area. Scale is constant along any parallel or pair of parallels equidistant from the Equator. This projection is neither equidistant nor conformal.

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.

Fournier Projection





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.

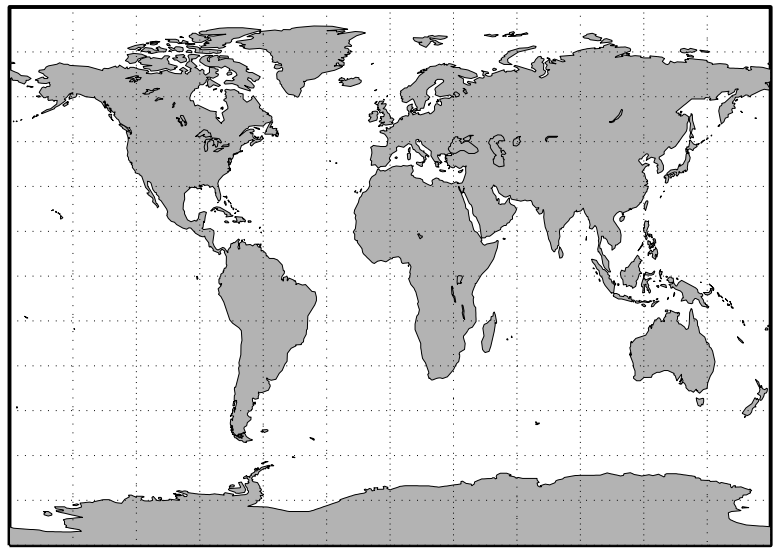
Symmetry: About any meridian or the Equator.

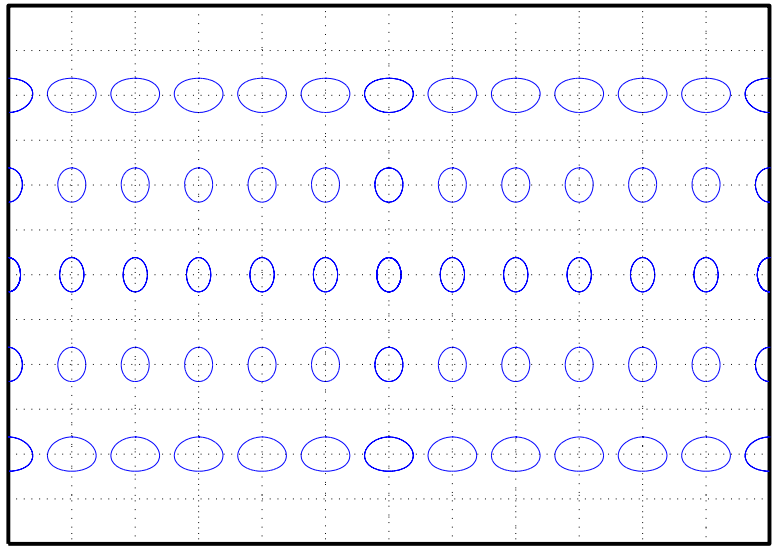
Features This is a projection onto a cylinder secant at the 45° parallels. Distortion of both shape and area increase with distance from the standard parallels. Scale is true along all meridians (i.e., it is equidistant) and the two standard parallels, and is constant along any parallel and along the 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 Isographic Projection





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.

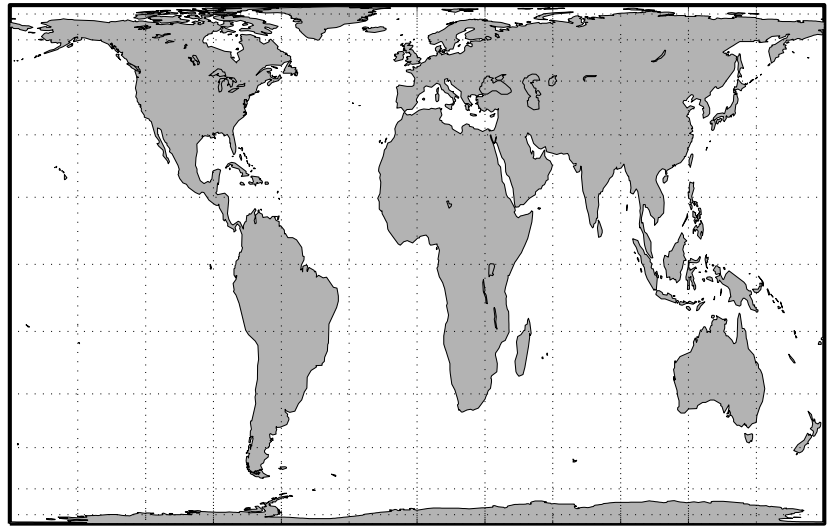
Symmetry: About any meridian or the Equator.

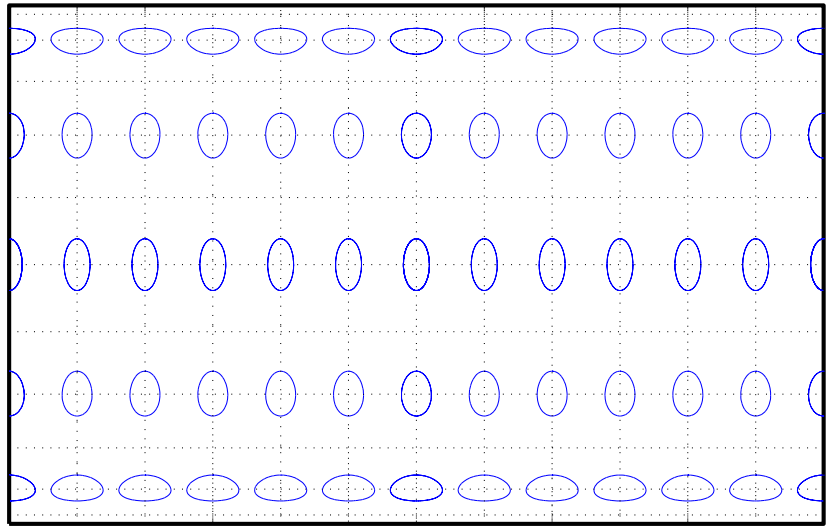
Features This is an orthographic projection onto a cylinder secant at the 45° parallels. It is equal-area, but distortion of shape increases with distance from the standard parallels. Scale is true along the standard parallels and constant between two parallels equidistant from the Equator. This projection 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 Orthographic 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.

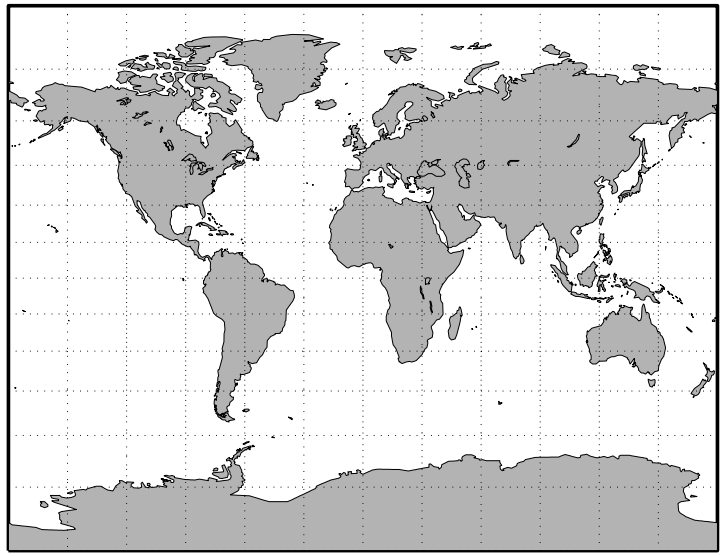
Features This is a perspective projection from a point on the Equator opposite a given meridian onto a cylinder secant at the 45° parallels. It is not equal-area, equidistant, or conformal. Scale is true along the standard parallels and constant between two parallels equidistant from the Equator. There is no distortion along the standard parallels, but it increases moderately away from these parallels, becoming severe at the poles.

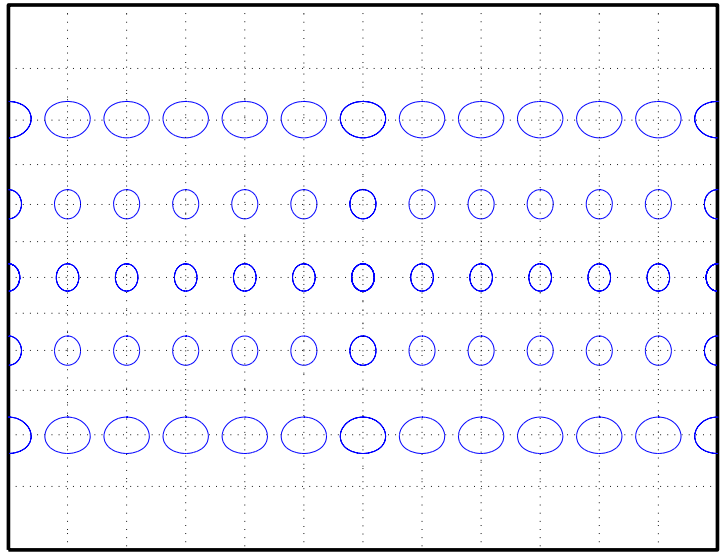
Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, the standard parallel is by definition fixed at 45°.

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.

Gall Stereographic Projection





Classification Spherical

Syntax `globe`

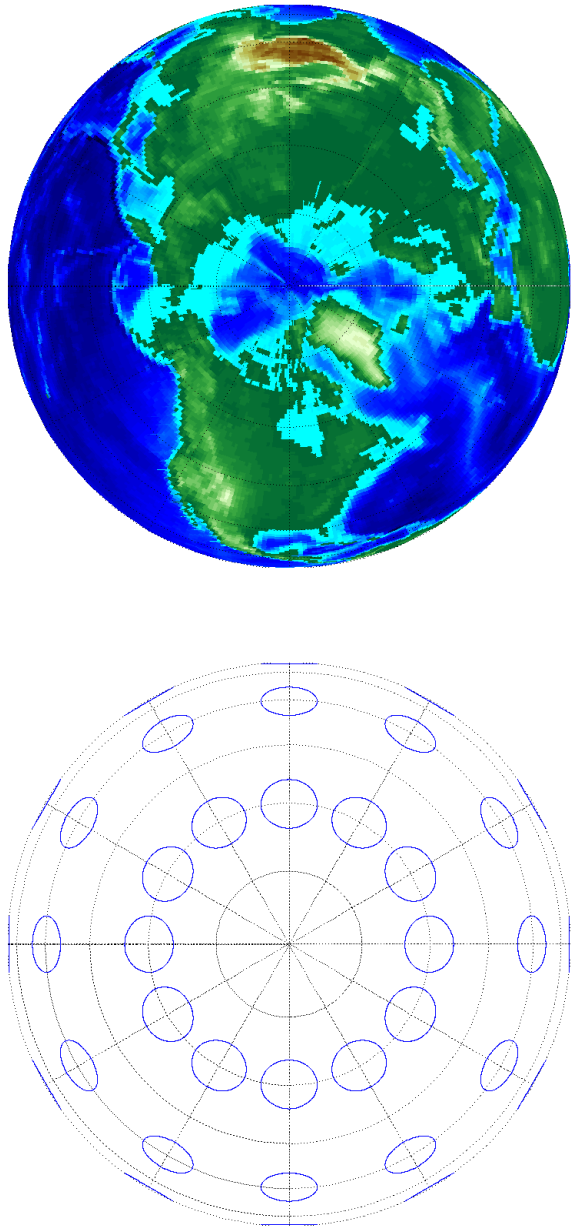
Graticule This *projection* is constructed by calculating a three-dimensional frame and displaying the map objects on the surface of this frame.

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.

Globe



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.

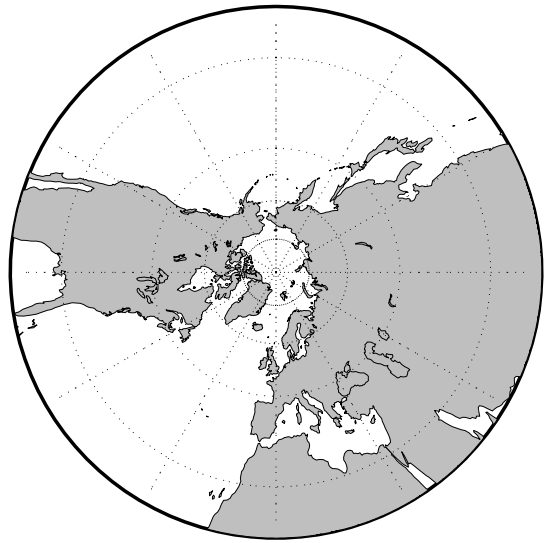
Features This is a perspective projection from the center of the globe on a plane tangent at the center point, which is a pole in the common polar aspect, but can be any point. Less than one hemisphere can be shown with this projection, regardless of its center point. The significant property of this 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.

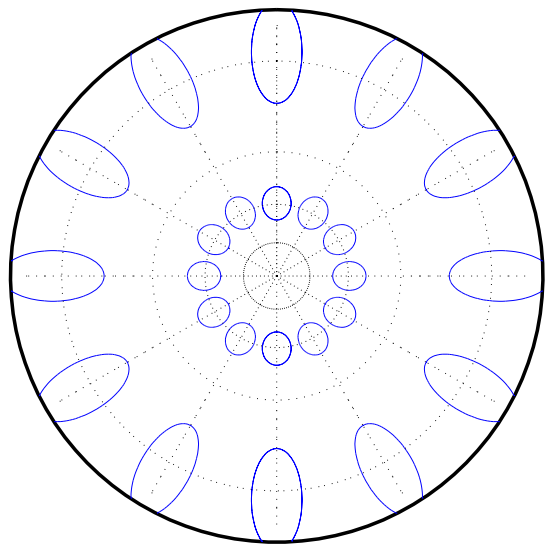
Parallels There are no standard parallels for azimuthal projections.

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.

Gnomonic Projection





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.

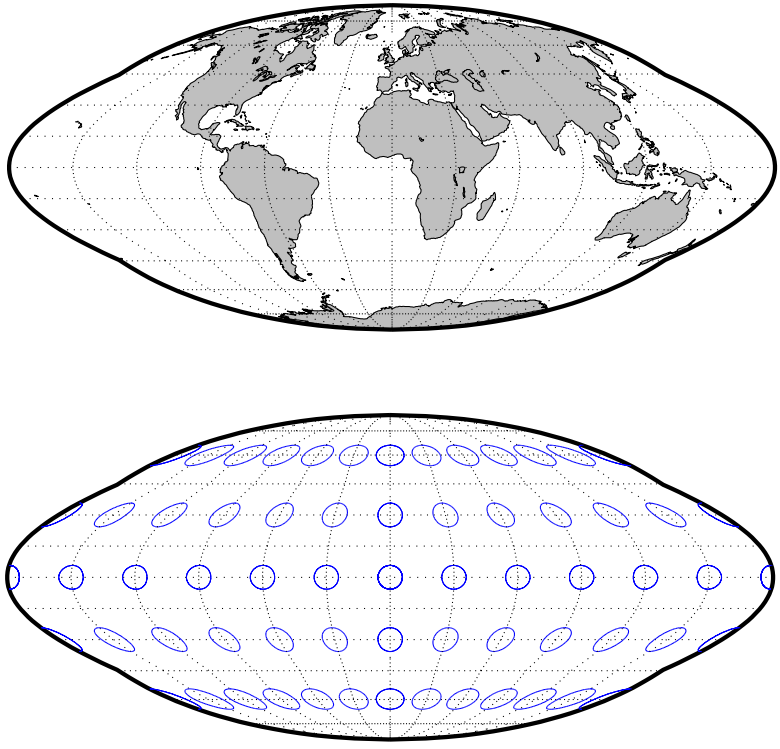
Features This is an equal-area projection. Scale is true along all parallels and the central meridian between 40 44'11.8" N and S, and is constant along any parallel and between any pair of parallels equidistant from the Equator for all latitudes. Its distortion is identical to that of the Sinusoidal 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.

Goode Homolosine 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.

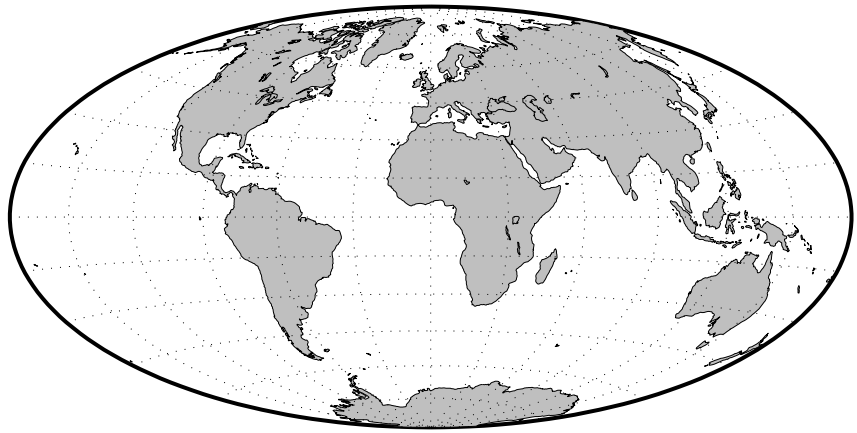
Symmetry: About the Equator and central meridian.

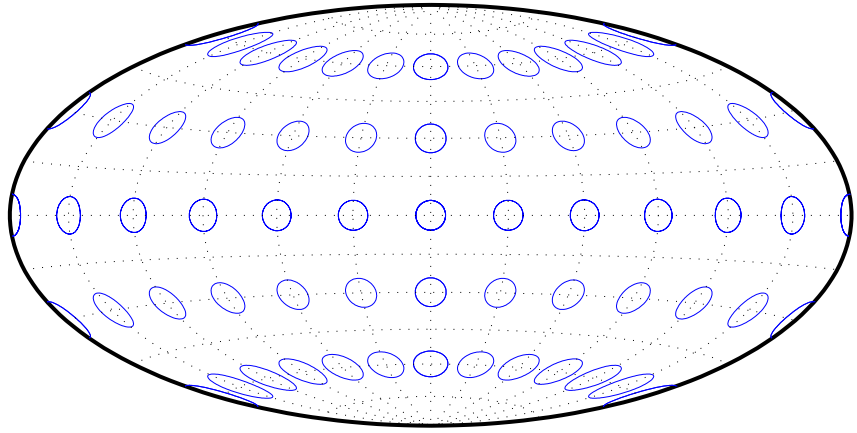
Features This projection is equal-area. The only point free of distortion is the center point. Distortion of shape is moderate throughout. This projection has less angular distortion on the outer meridians near the poles than pseudoazimuthal projections

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.

Hammer Projection





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.

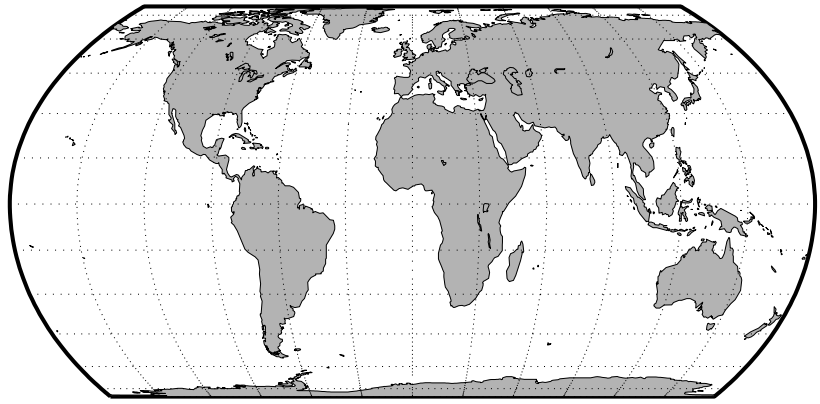
Symmetry: About the central meridian but *not* the Equator.

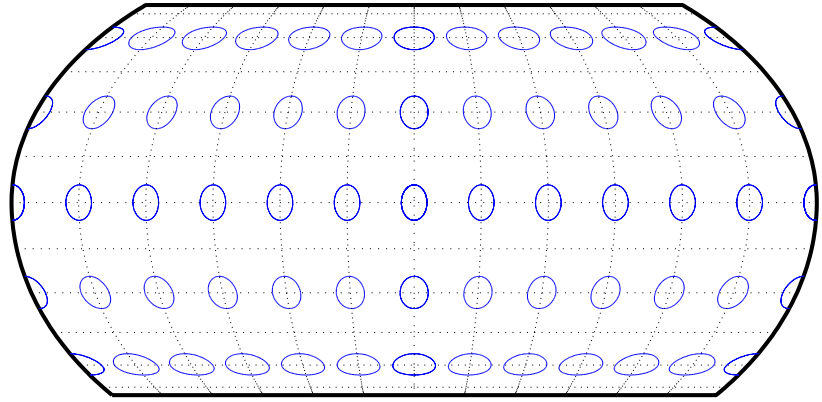
Features This is an equal-area projection. Scale is true along 40 42'N and 38 27'S, and is constant along any parallel but generally *not* between pairs of parallels equidistant from the Equator. It is free of distortion only along the central meridian at 40 42'N and 38 27'S. This projection is not conformal 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'Nand 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.

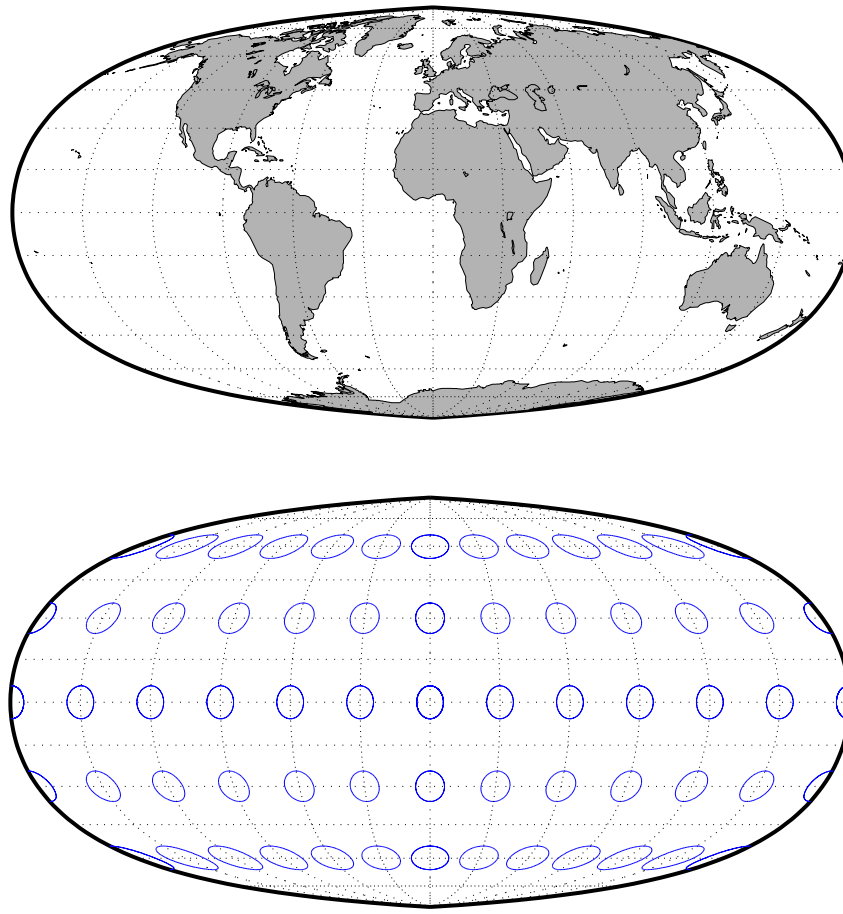
Symmetry: About the Equator and the central meridian.

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 V Projection



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.

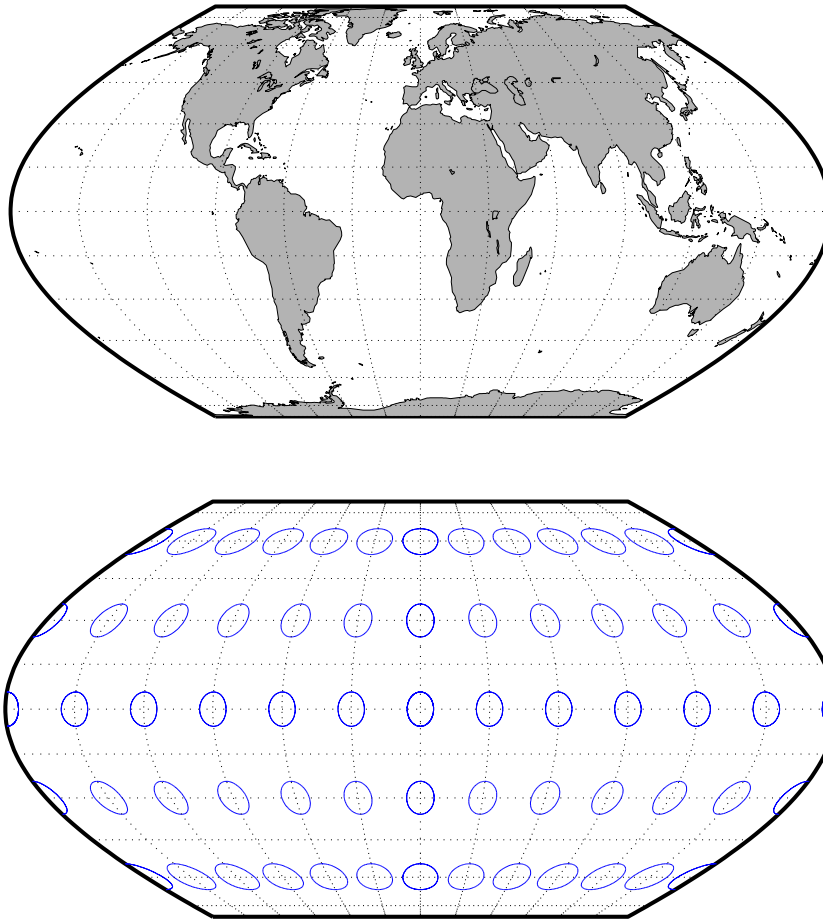
Symmetry: About the Equator and the central meridian.

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.

Kavraisky VI Projection



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.

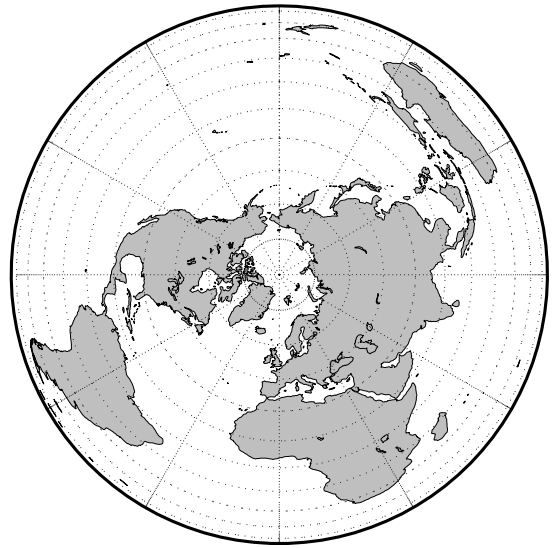
Features This nonperspective projection is equal-area. Only the center point is free of distortion, but distortion is moderate within 90 of this point. Scale is true only at the center point, increasing tangentially and decreasing radially with distance from the center point. This projection is neither conformal nor equidistant.

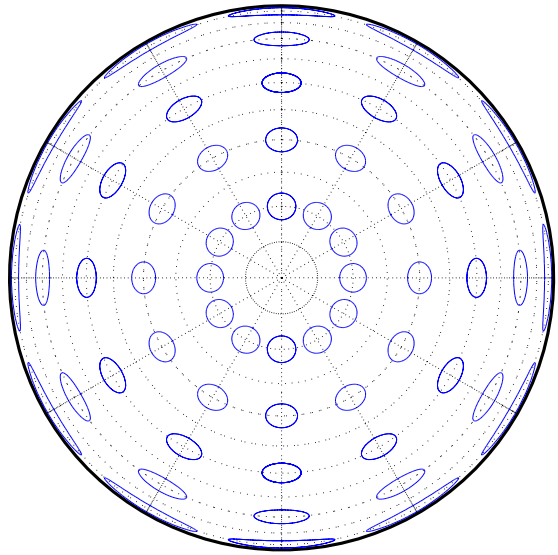
Parallels There are no standard parallels for azimuthal projections.

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 Azimuthal Equal-Area 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.

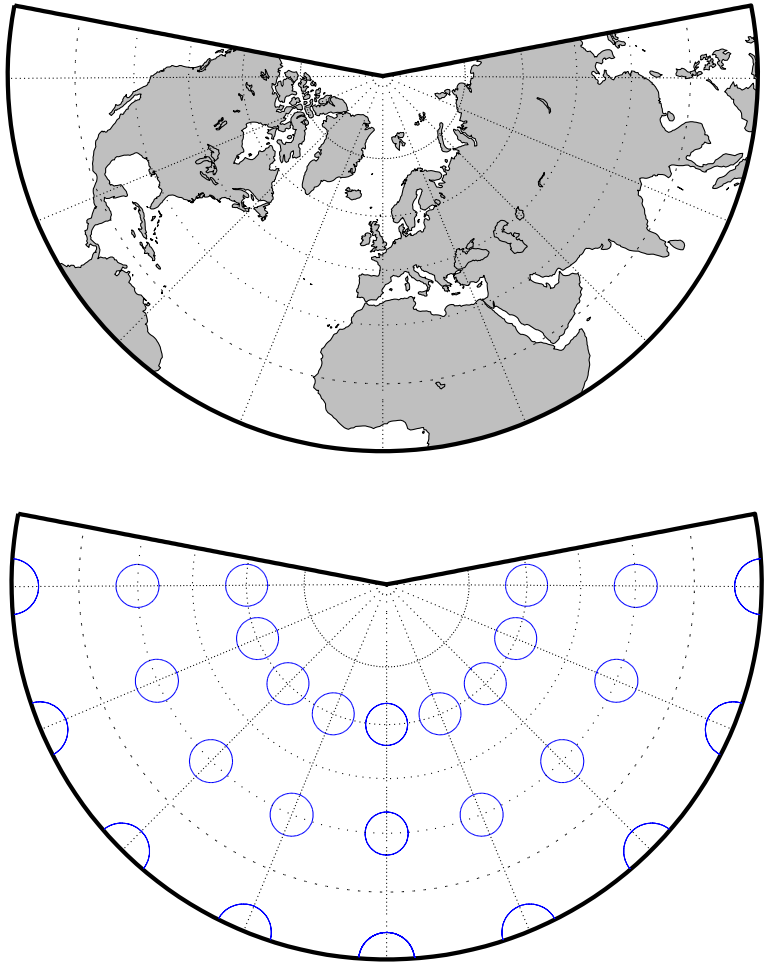
Features Scale is true along the one or two selected standard parallels. Scale is constant along any parallel and is the same in every direction at any point. This projection is free of distortion along the standard parallels. Distortion is constant along any other parallel. This projection is conformal 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 Conformal Conic Projection



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.

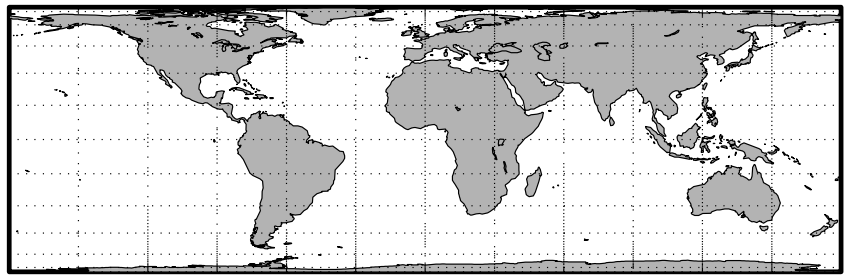
Symmetry: About any meridian or the Equator.

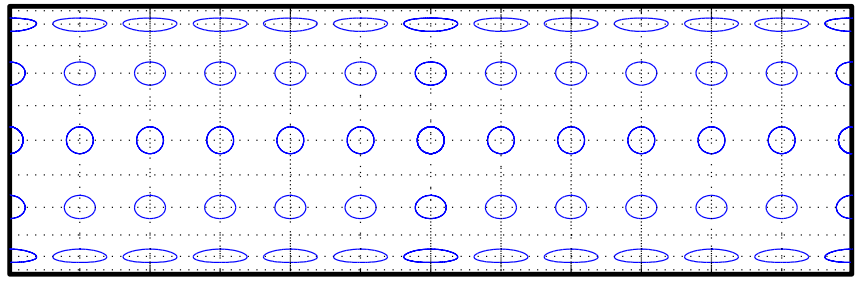
Features This is an orthographic projection onto a cylinder tangent at the Equator. It is equal-area, but distortion of shape increases with distance from the Equator. Scale is true along the Equator and constant between two parallels equidistant from the Equator. This projection is not equidistant.

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.

Lambert Equal-Area Cylindrical 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.

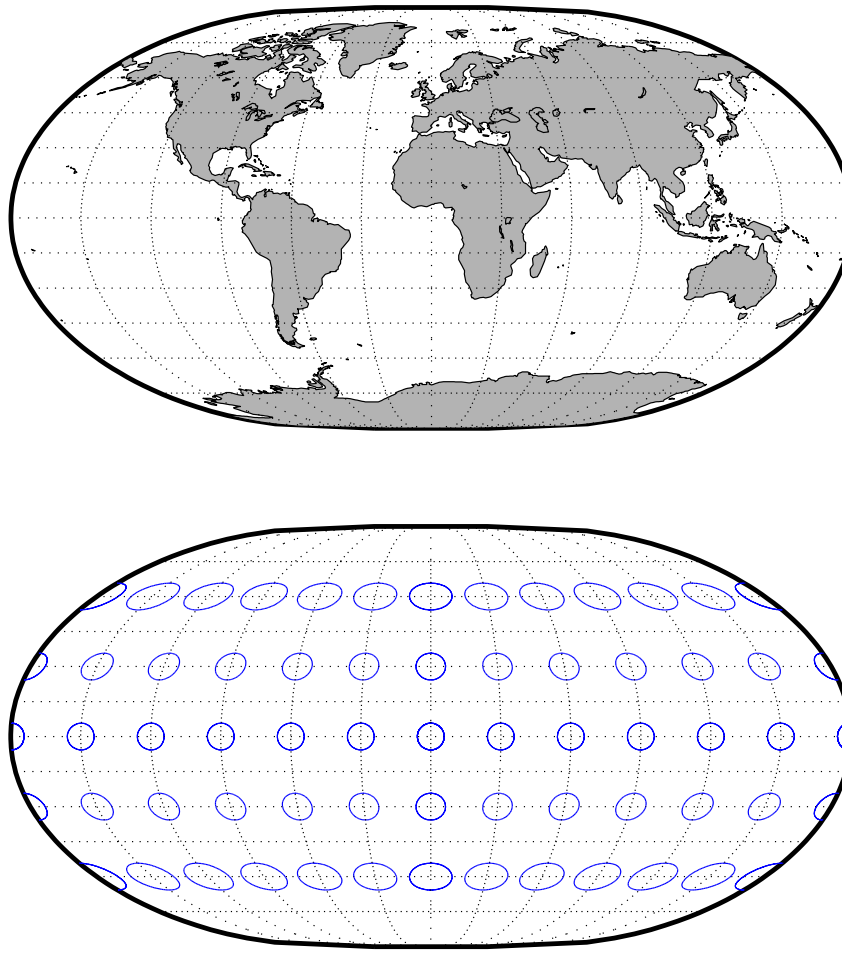
Features This projection has the special property that from the central point (the intersection of the central latitude with the central meridian), rhumb lines (loxodromes) are shown as straight, true to scale, and correct in azimuth from the center. This differs from the Mercator projection, in that 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.

Loximuthal Projection



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.

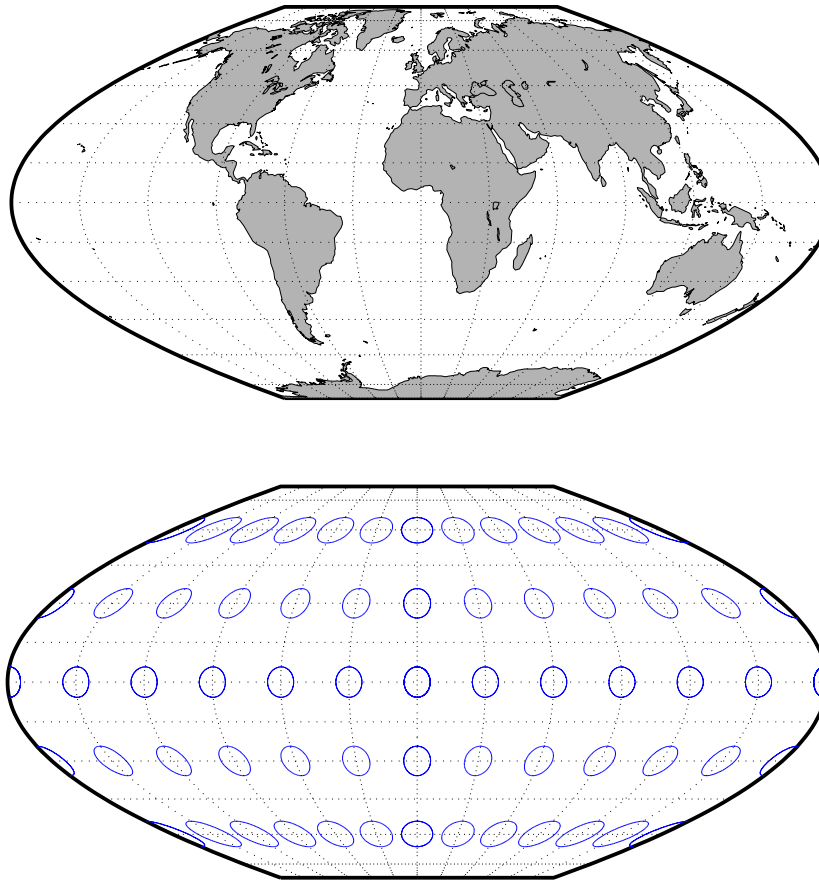
Symmetry: About the central meridian or the Equator.

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.

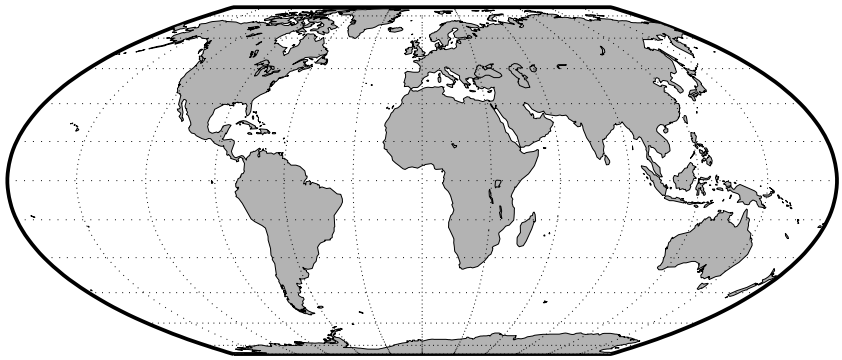
Symmetry: About the central meridian or the Equator.

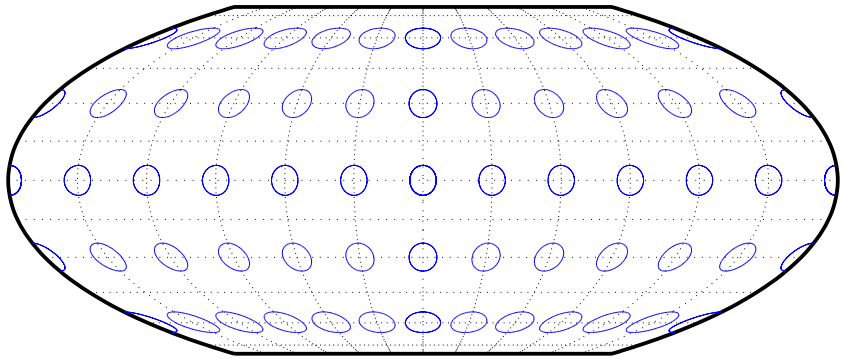
Features This is an equal-area projection. Scale is true along the 33 45' parallels and is constant along any parallel and between any pair of parallels equidistant from the Equator. Distortion is severe near the outer meridians at high latitudes, but less so than on the pointed-polar projections. It 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.

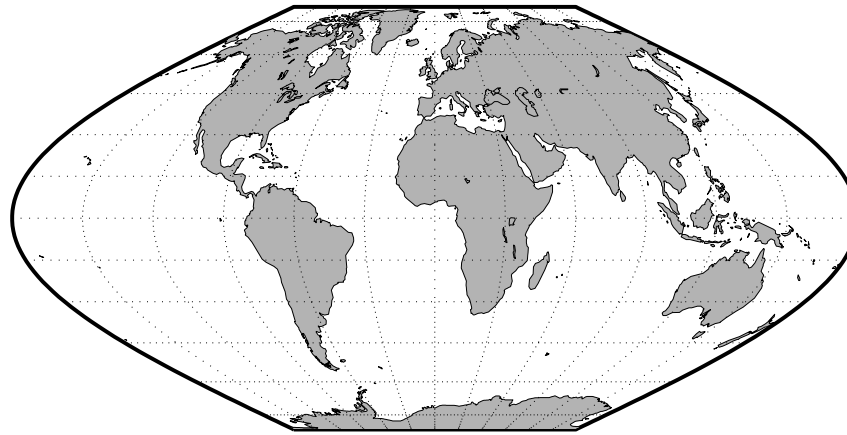
Symmetry: About the central meridian or the Equator.

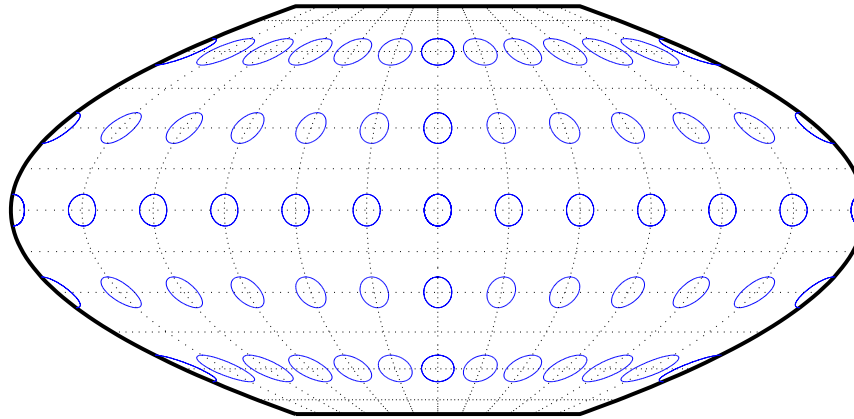
Features This projection is equal-area. Scale is true along the 55 51' parallels and is constant along any parallel and between any pair of parallels equidistant from the Equator. It is free of distortion only at the two points where the central meridian intersects the 55 51' parallels. This projection 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





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.

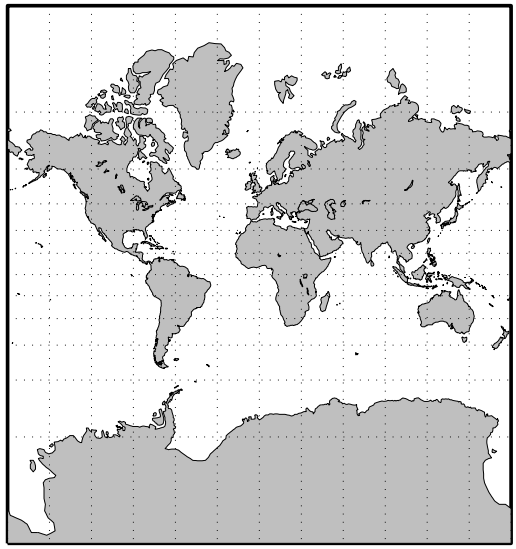
The Mercator, which may be the most famous of all projections, has the special feature that all rhumb lines, or loxodromes (lines that make equal angles with all meridians, i.e., lines of constant heading), are straight lines. This makes it an excellent projection for navigational purposes. However, the extreme area distortion makes it unsuitable for general maps of large areas.

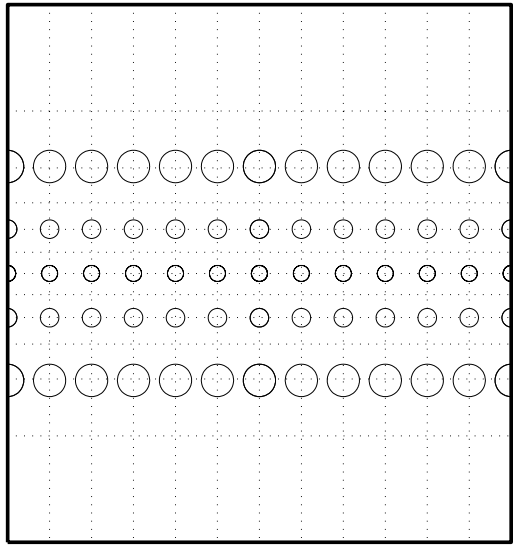
Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, any latitude less than 86° may be chosen; the default is arbitrarily set to 0°.

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.

Mercator Projection





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.

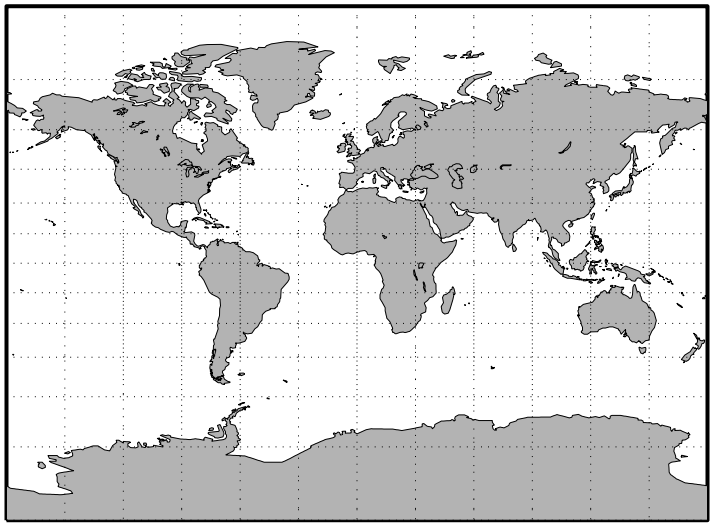
The Miller Cylindrical projection is derived from the Mercator projection; parallels are spaced from the Equator by calculating the distance on the Mercator for a parallel at 80% of the true latitude and dividing the result by 0.8. The result is that the two projections are almost identical near the Equator.

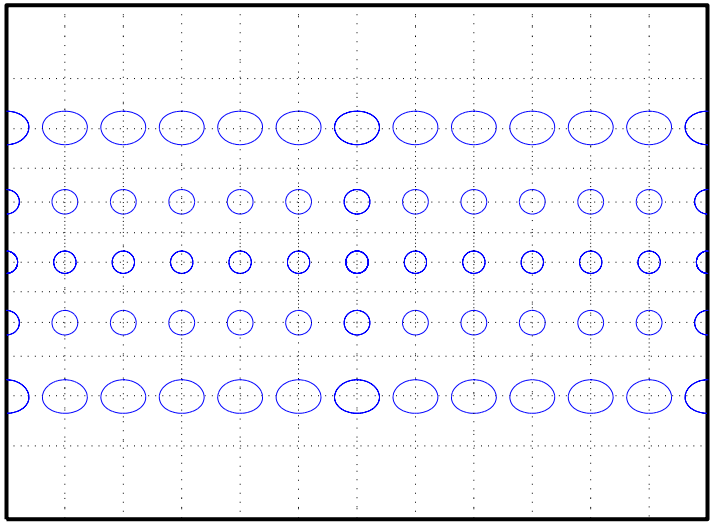
Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, the standard parallel is by definition fixed at 0°.

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.

Miller Cylindrical 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.

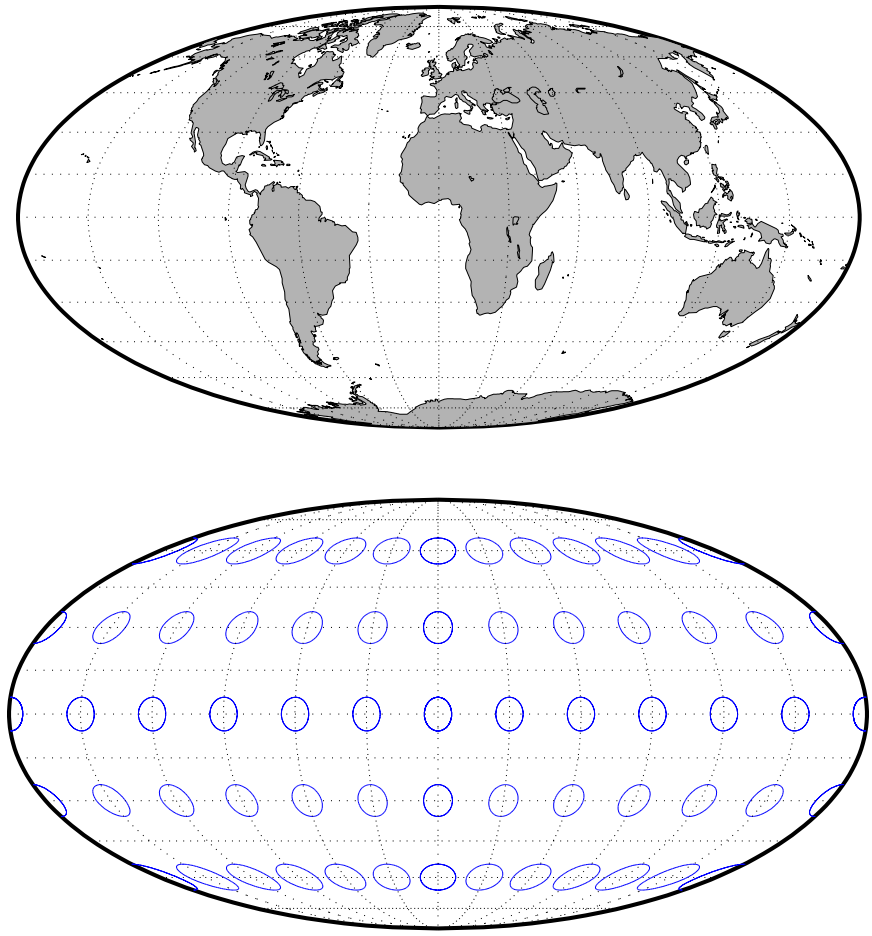
Symmetry: About the central meridian or the Equator.

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.

Mollweide 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.

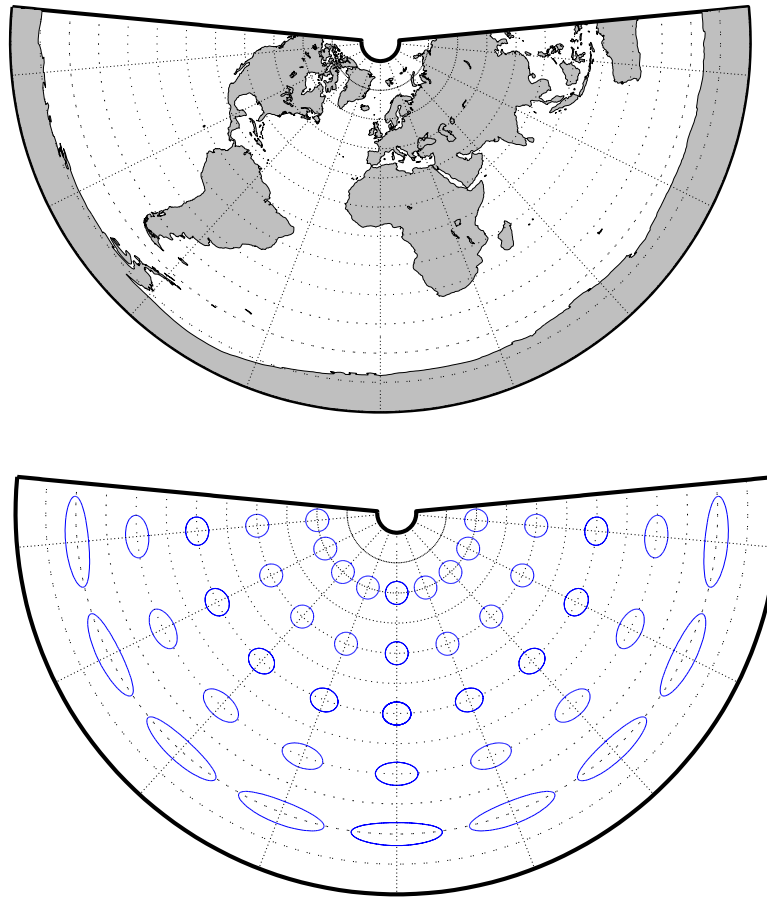
Features This is an equidistant projection that is nearly minimum-error. Scale is true along any meridian and is constant along any parallel. Scale is also true along two standard parallels. These must be calculated, however (see below). The total area of the mapped area is correct, but it is not equal-area 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 I Conic Projection



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.

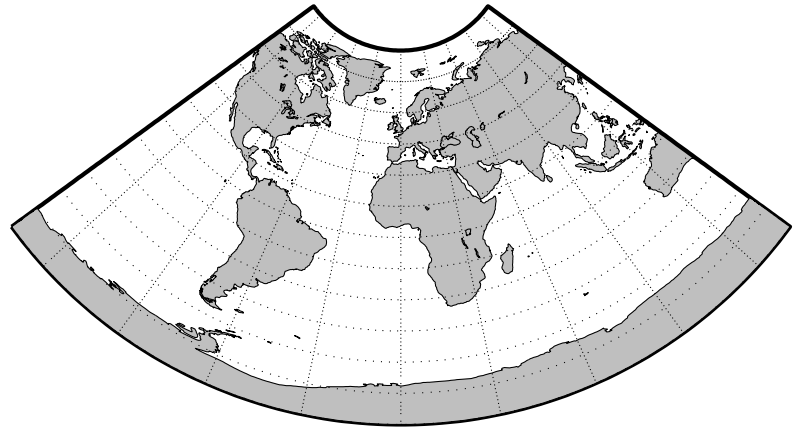
Features This is an equidistant projection that is minimum-error. Scale is true along any meridian and is constant along any parallel. Scale is also true along two standard parallels. These must be calculated, however (see below). The total area of the mapped area is correct, but it is not equal-area everywhere.

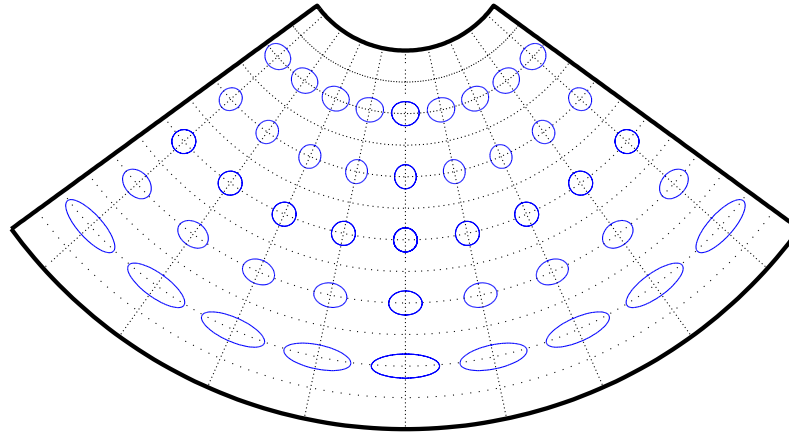
Parallels The parallels for this projection are not standard parallels, but rather limiting parallels. The special feature of this map, correct total area, holds between these parallels. The default parallels are [15 75].

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.

Murdoch III Minimum Error Conic 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.

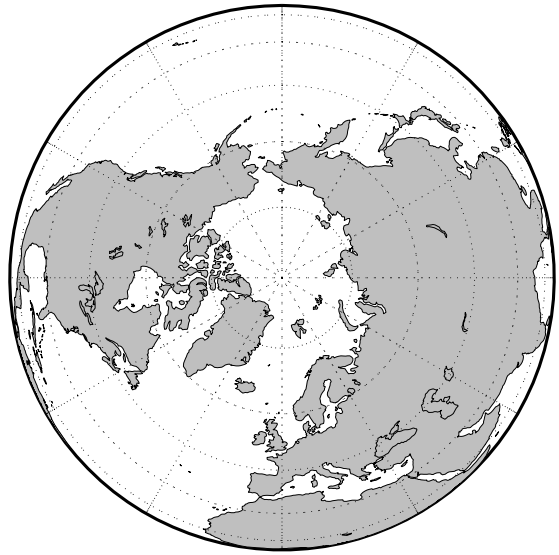
Features This is a perspective projection on a plane tangent at the center point from an infinite distance (that is, orthogonally). The center point is a pole in the common polar aspect, but can be any point. This projection has two significant properties. It looks like a globe, providing views of the 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.

Orthographic Projection



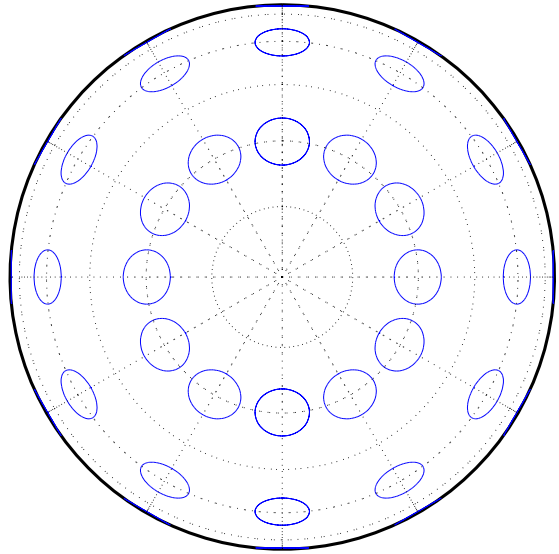


Plate Carrée Projection

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.

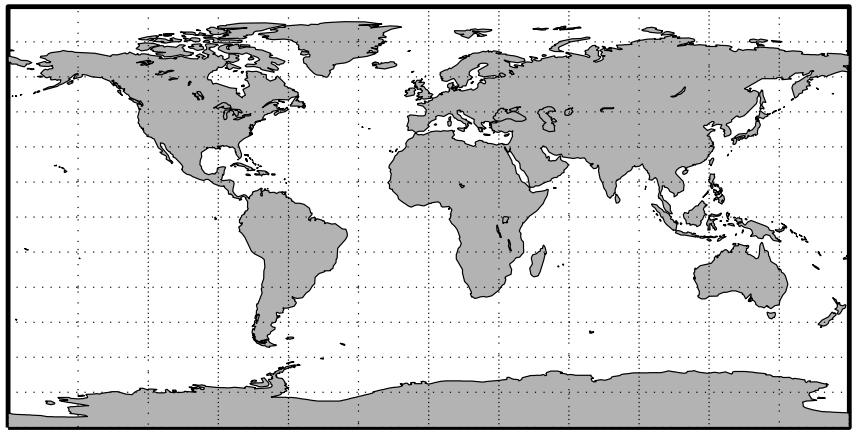
Symmetry: About any meridian or the Equator.

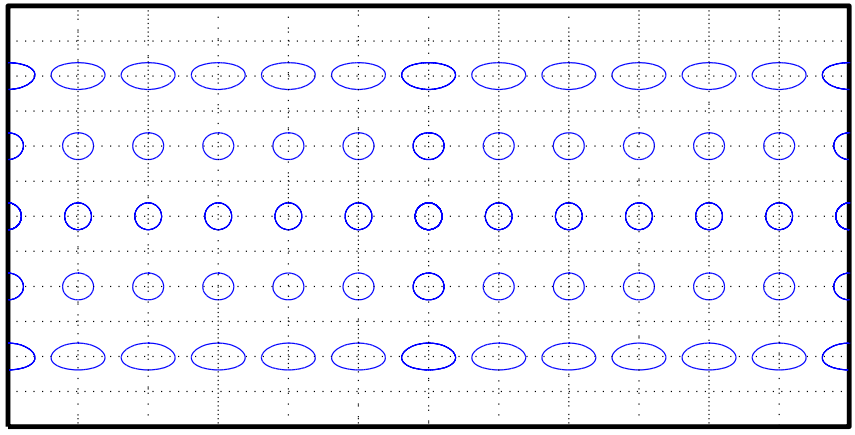
Features This is a projection onto a cylinder tangent at the Equator. Distortion of both shape and area increases with distance from the Equator. Scale is true along all meridians (i.e., it is equidistant) and the Equator and is constant along any parallel and along the parallel of opposite sign.

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.

Plate Carrée 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.

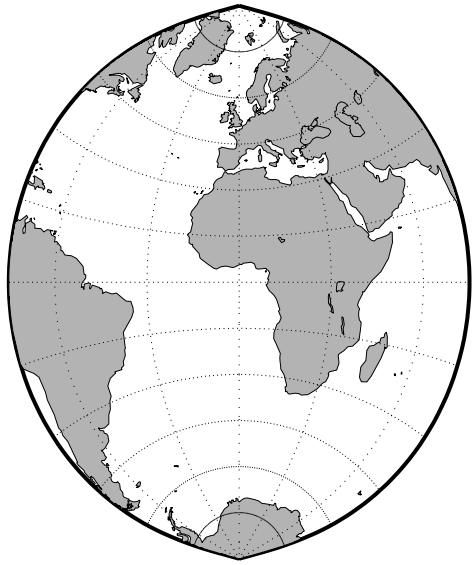
Features For this projection, each parallel has a curvature identical to its curvature on a cone tangent at that latitude. Since each parallel has its own cone, this is a "polyconic" projection. Scale is true along the central meridian and along each parallel. This projection is free of distortion only along the central meridian; distortion can be severe at extreme longitudes. This projection is neither conformal nor equal-area.

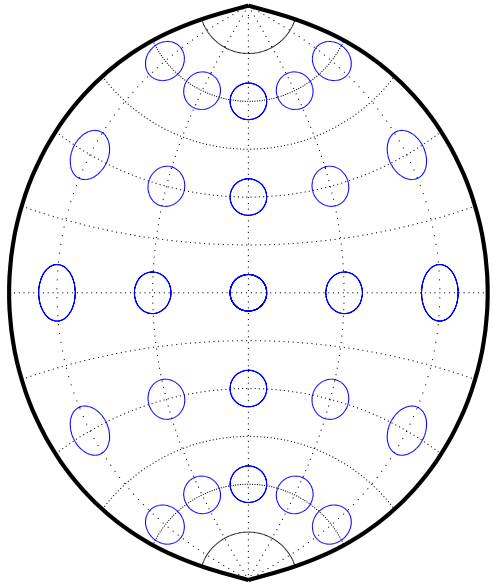
Parallels By definition, this projection has no standard parallels, since every parallel is a *standard parallel*.

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.

Polyconic Projection





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.

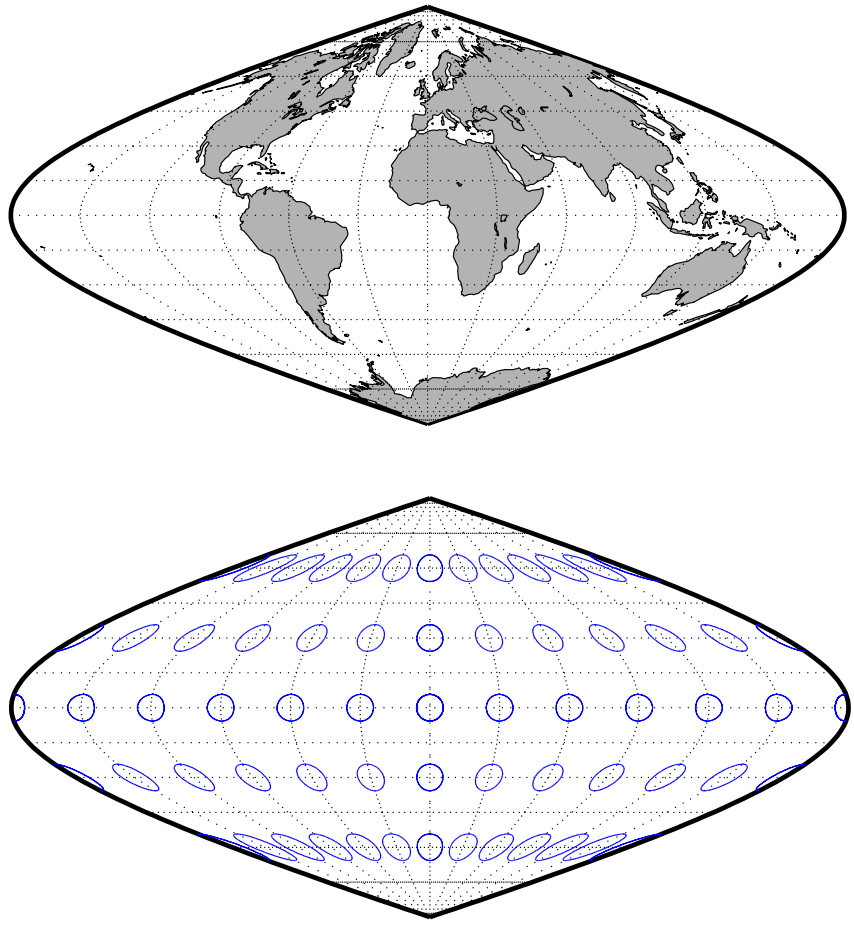
Features Scale is true along the 21 14' parallels and is constant along any parallel, between any pair of parallels equidistant from the Equator, and along the central meridian. It is not free of distortion at any point. This projection is not equal-area, conformal, or equidistant.

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.

Putnins P5 Projection



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.

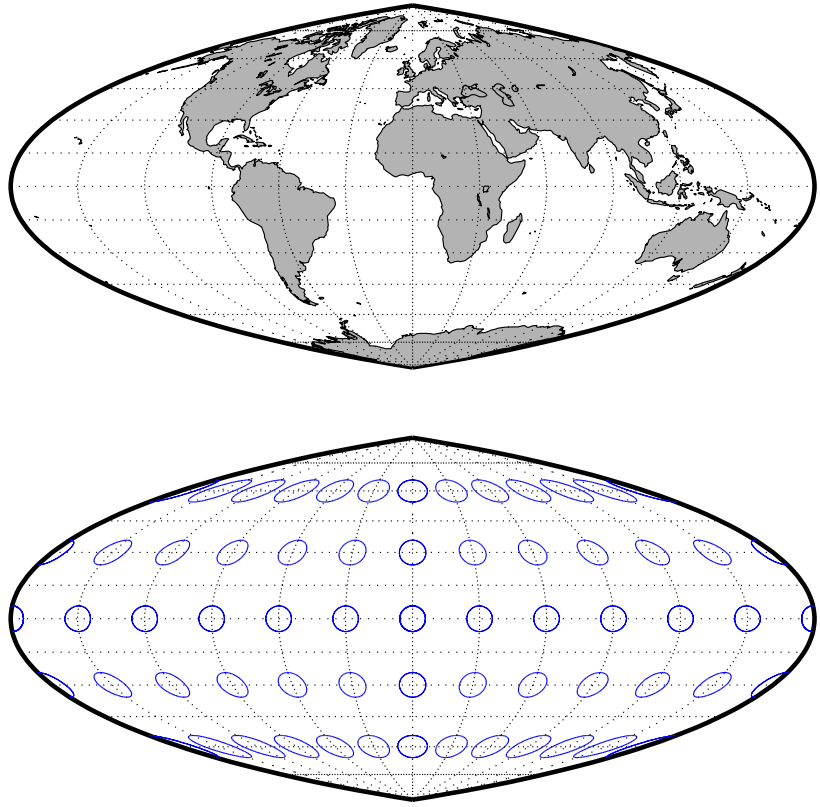
Symmetry: About the central meridian or the Equator.

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.

Quartic Authalic 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.

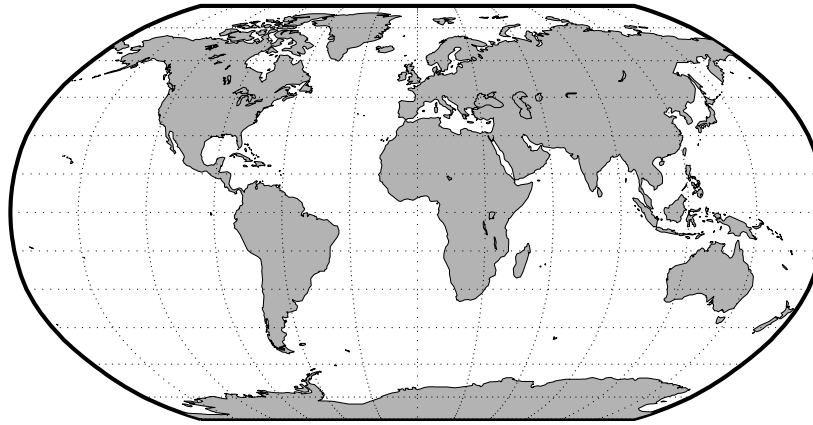
Features Scale is true along the 38 parallels and is constant along any parallel or between any pair of parallels equidistant from the Equator. It is not free of distortion at any point, but distortion is very low within about 45 of the center and along the Equator. This projection is not equal-area, 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.

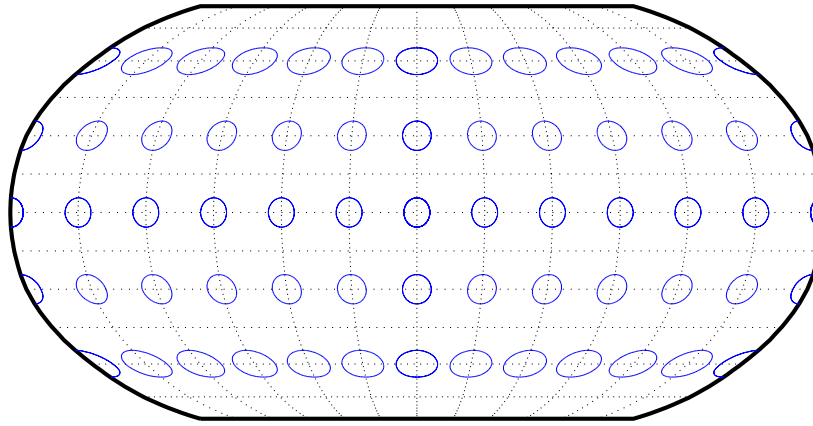
Parallels For this projection, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. The standard parallel is by definition fixed at 38 .

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.

Robinson 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.

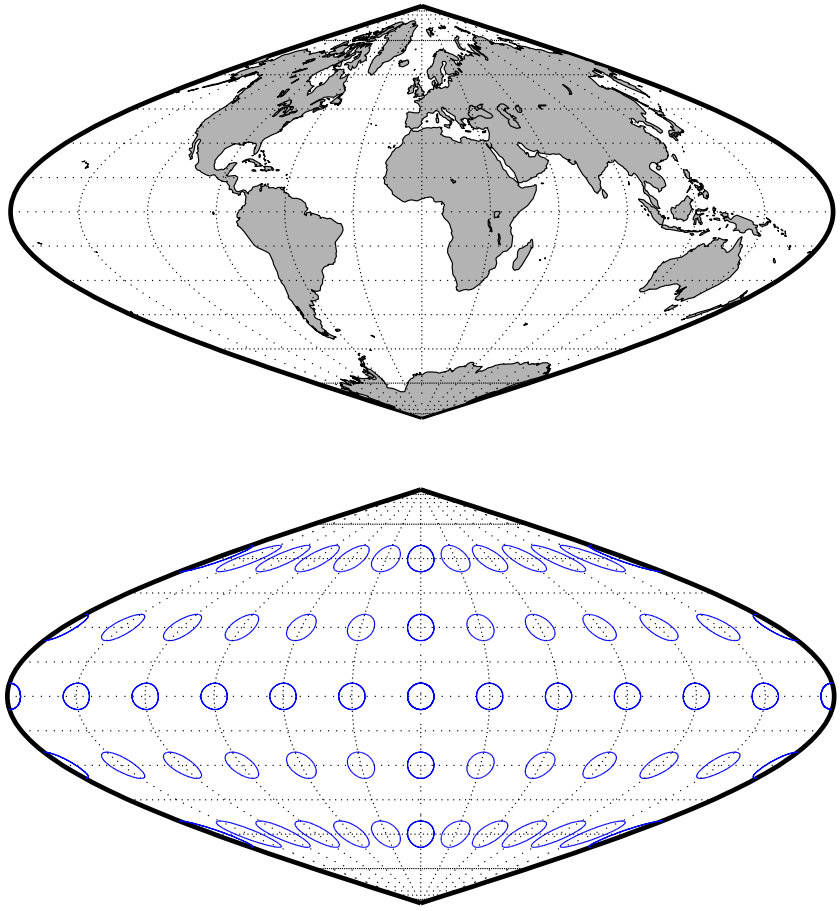
Symmetry: About the central meridian or the Equator.

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.

Sinusoidal 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.

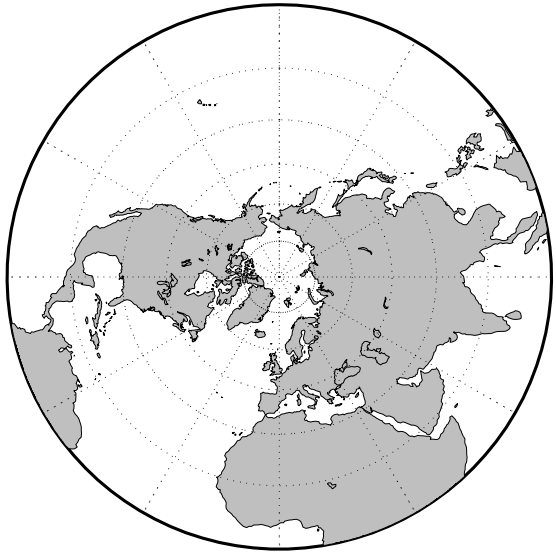
Features This is a perspective projection on a plane tangent at the center point from the point antipodal to the center point. The center point is a pole in the common polar aspect, but can be any point. This projection has two significant properties. It is conformal, being free from angular distortion. 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.

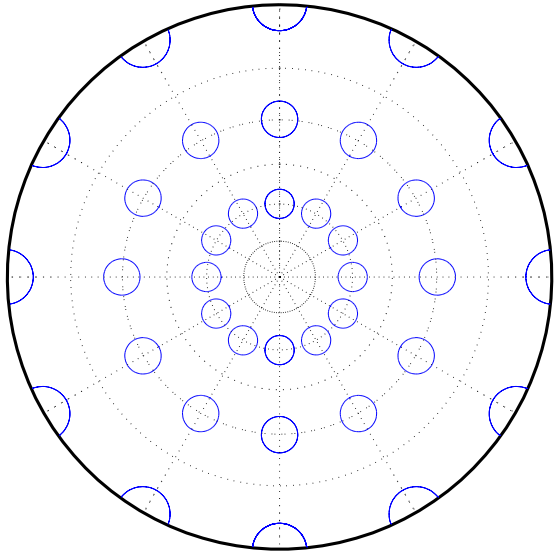
Parallels There are no standard parallels for azimuthal projections.

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.

Stereographic Projection





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

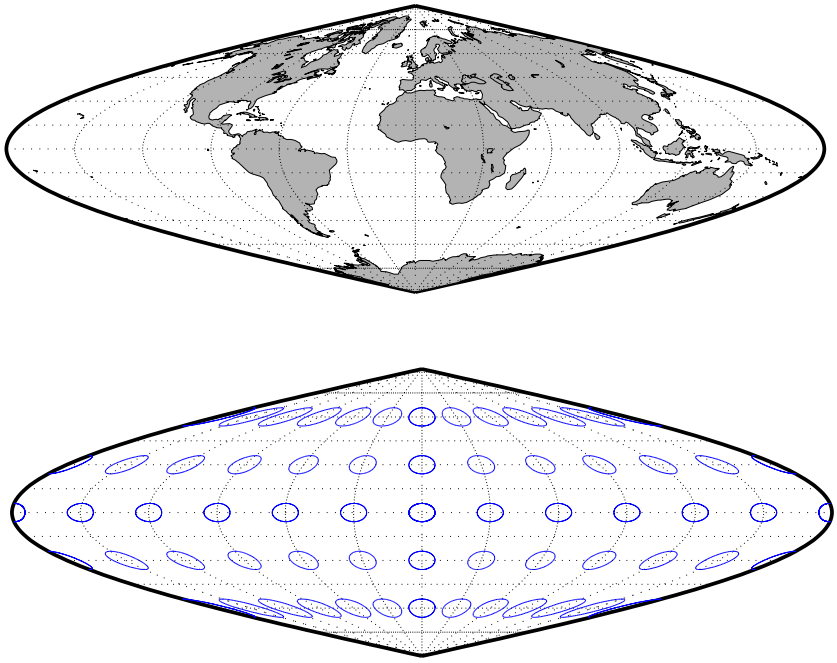
Features This is an equal-area projection. Scale is constant along any parallel or any pair of equidistant parallels, and along the central meridian. It is not equidistant or conformal.

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.

Tissot Modified Sinusoidal Projection



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.

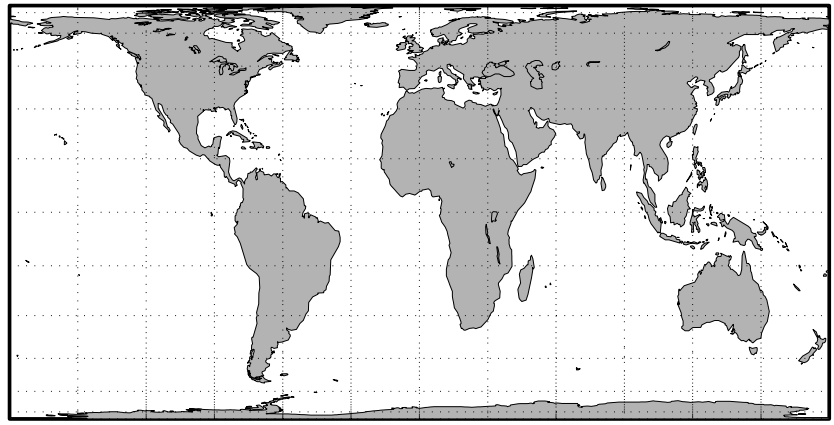
Symmetry: About any meridian or the Equator.

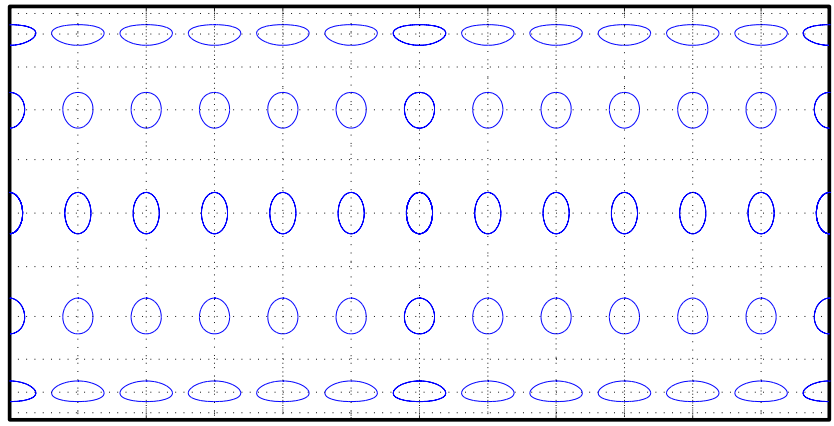
Features This is an orthographic projection onto a cylinder secant at the 37°24' parallels. It is equal-area, but distortion of shape increases with distance from the standard parallels. Scale is true along the standard parallels and constant between two parallels equidistant from the Equator. This projection 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.

Trystan Edwards Cylindrical Projection





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.

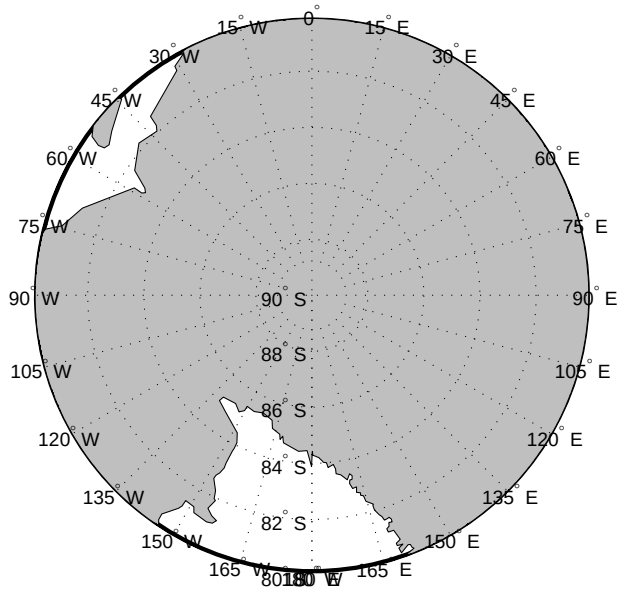
Symmetry: About any meridian.

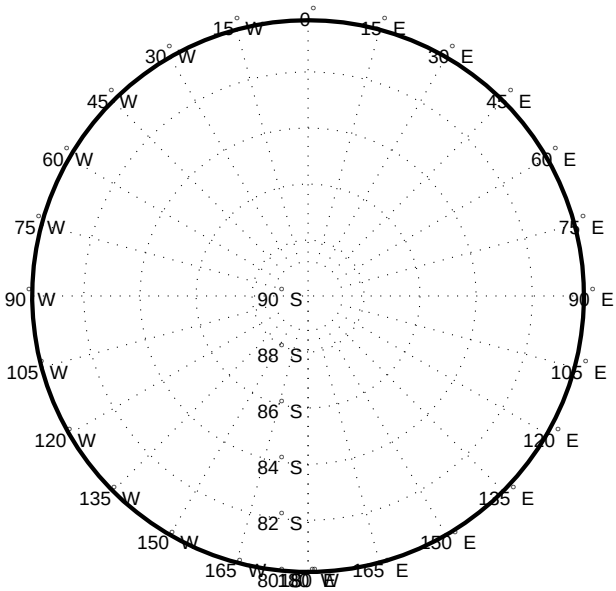
Features This is a perspective projection on a plane tangent to either the North or South Pole. It is conformal, being free from angular distortion. Additionally, all great and small circles are either straight lines or circular arcs on this projection. Scale is true along latitudes 87 degrees, 7 minutes 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.

Universal Polar Stereographic Projection





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.

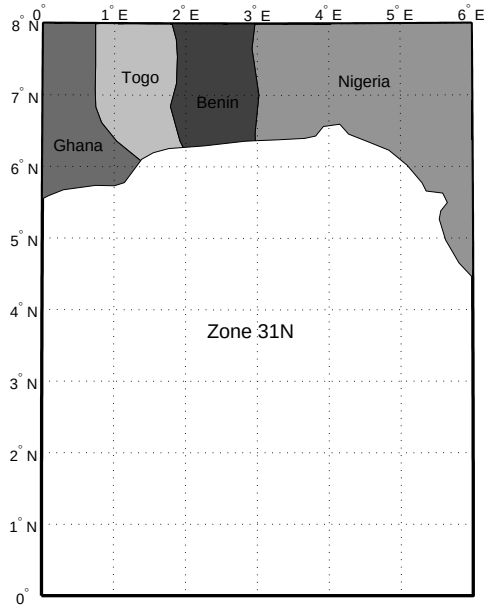
Features This is a conformal projection with parameters chosen to minimize distortion over a defined set of small areas. It is not equal area, equidistant, or perspective. Scale is true along two straight lines on the map approximately 180 kilometers east and west of the central meridian, and is constant 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.

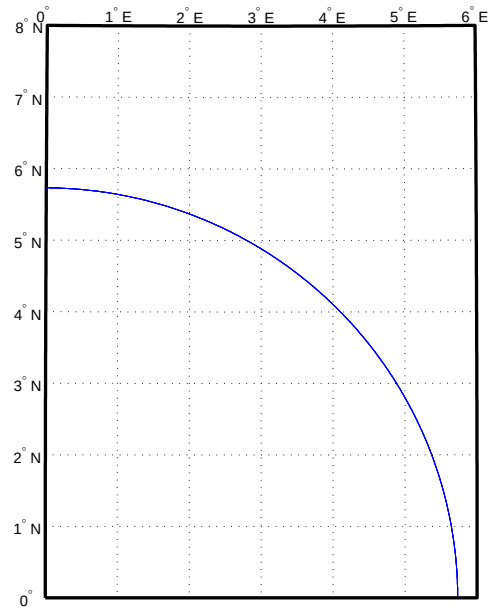
Parallels There are no standard parallels for this projection. There are two lines of zero distortion by virtue of the scale factor.

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.

Universal Transverse Mercator Projection





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.

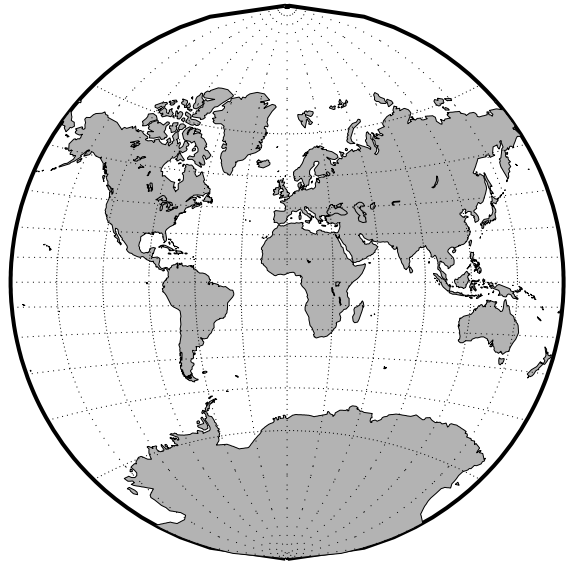
Features In this projection, the world is enclosed in a circle. Scale is true along the Equator and increases rapidly away from the Equator. Area distortion is extreme near the poles. This projection is neither conformal nor equal-area.

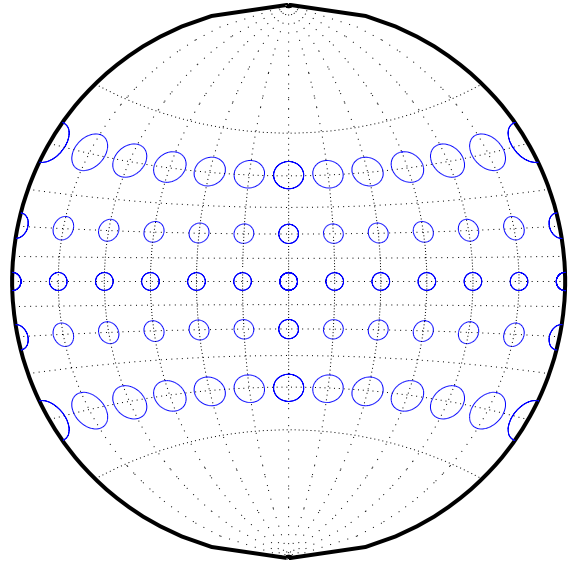
Parallels There are no standard parallels for this projection.

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.

Van der Grinten I Projection





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.

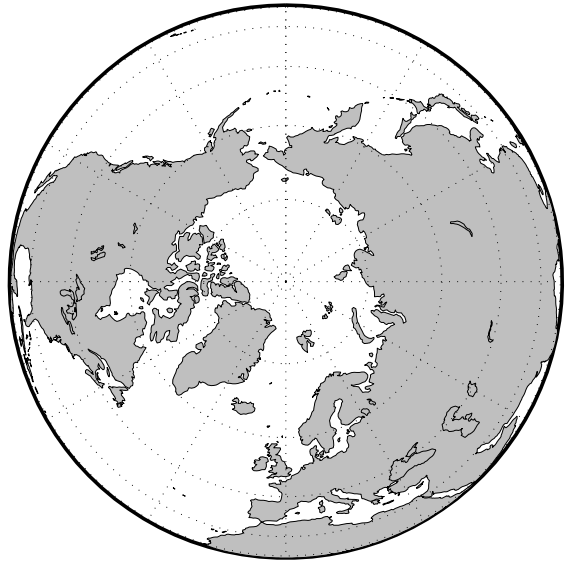
Features This is a perspective projection on a plane tangent at the center point from a finite distance. Scale is true only at the center point, and is constant in the circumferential direction along any circle having the center point as its center. Distortion increases rapidly away from the center point, the only point which is distortion free. This projection is neither conformal nor equal area.

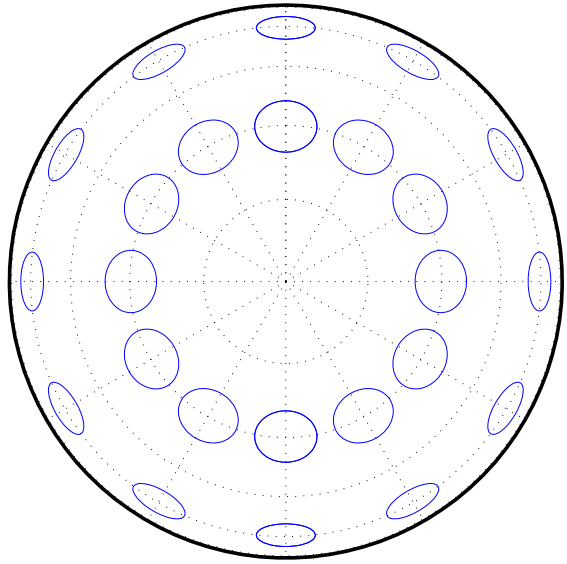
Parallels The standard parallel contains the observation altitude above the surface in the same units as the geoid semi-major axis.

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.

Vertical Perspective Azimuthal 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.

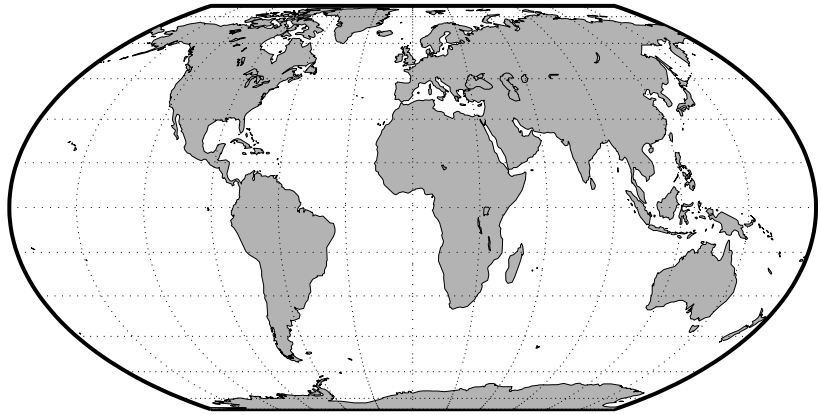
Symmetry: About the central meridian or the Equator.

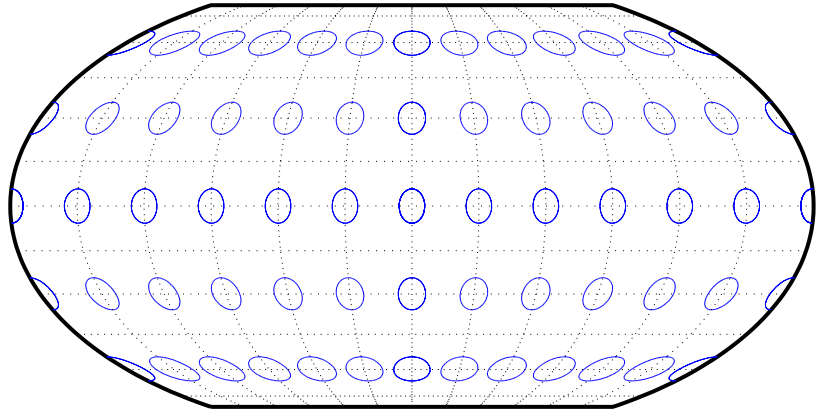
Features This is an equal-area projection. Scale is true along the 42 59' parallels and is constant along any parallel and between any pair of parallels equidistant from the Equator. Distortion is not as extreme near the outer meridians at high latitudes as for pointed-polar pseudocylindrical projections, 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.

Wagner IV Projection





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.

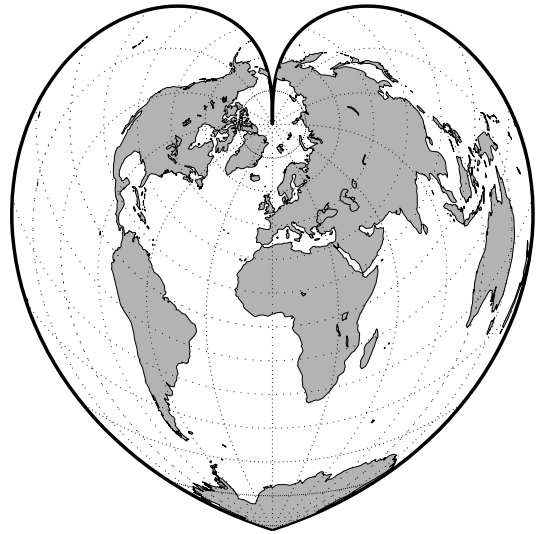
Symmetry: About the central meridian.

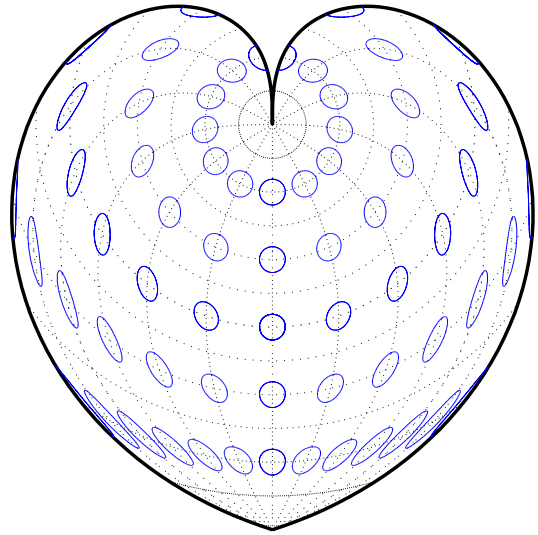
Features This is an equal-area projection. It is a Bonne projection with one of the poles as its standard parallel. The central meridian is free of distortion. This projection is not conformal. Its heart shape gives it the additional descriptor *cordiform*.

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.

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.

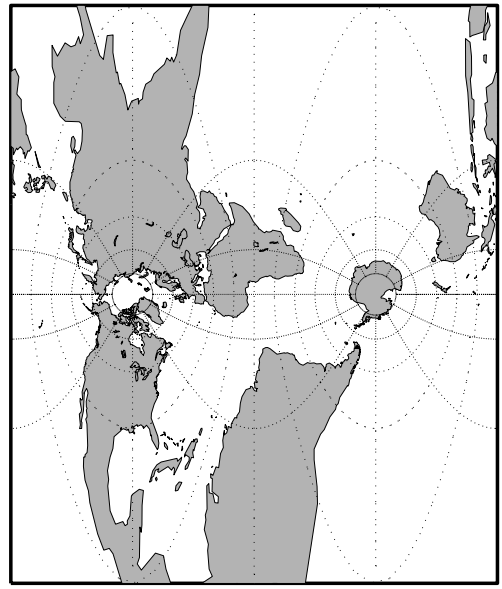
Features This is a perspective projection from the center of the Earth onto a cylinder tangent to the central meridian. It is not equal-area, equidistant, or conformal. Scale is true along the central meridian and constant between two points equidistant in *x* and *y* from the central meridian. There is no distortion along the central meridian, but it increases rapidly away from the central meridian in the *y*-direction.

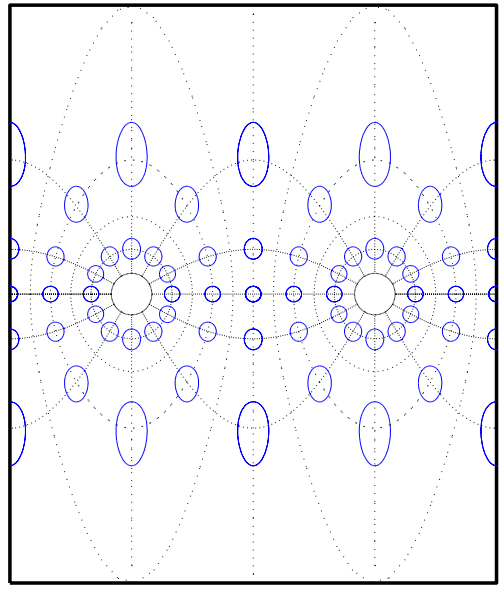
Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, which is the transverse aspect of the Central Cylindrical, the standard parallel *of the base projection* is by definition fixed 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.

Wetch Cylindrical Projection





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.

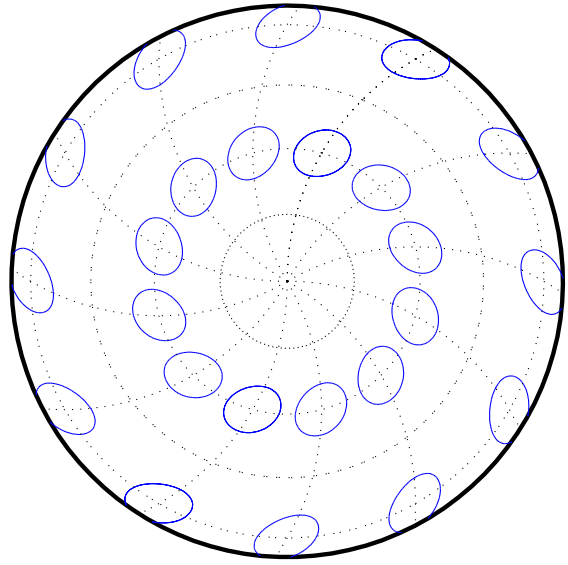
Features This equal-area projection is a novelty map, usually centered at a pole, showing semicircular meridians in a pinwheel arrangement. Scale is correct along the meridians. This projection is not conformal.

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.

Wiechel Projection



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.

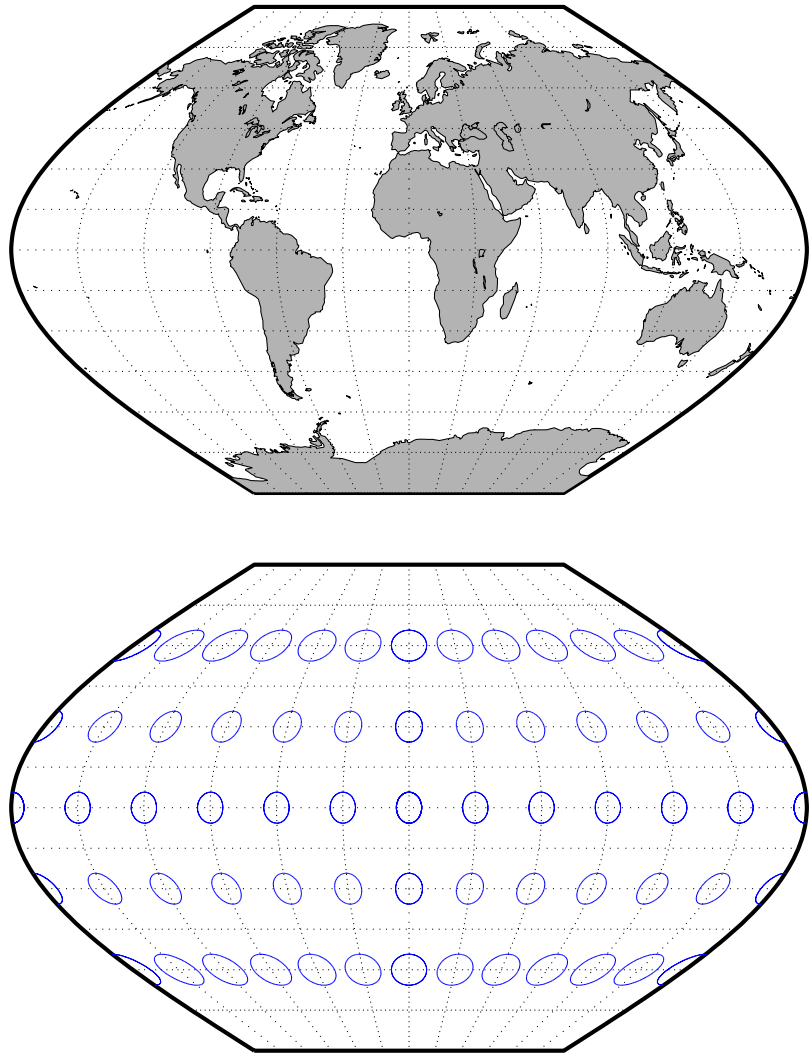
Features This projection is an arimethical average of the *x* and *y* coordinates of the Sinusoidal and Equidistant Cylindrical projections. Scale is true along the standard parallels and is constant along any parallel and between any pair of parallels equidistant from the Equator. There is no point 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



11

GUI Reference

=

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`, `patchesm`	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`, `surfacem`, `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.
`cmapui`	Create custom colormap.
`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
`scircleg`	Manipulate small circles on map axes.
`scirclui`	Display small circles on map axes.
`sectorg`	Manipulate sectors of small circles on map axes.
`surfdist`	Calculate distance, azimuth, and reckoning.
`trackg`	Manipulate great circle and rhumb line tracks on map axes.
`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.

Alphabetical List of GUI Functions and Tools

The following 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 9, "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.

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`.

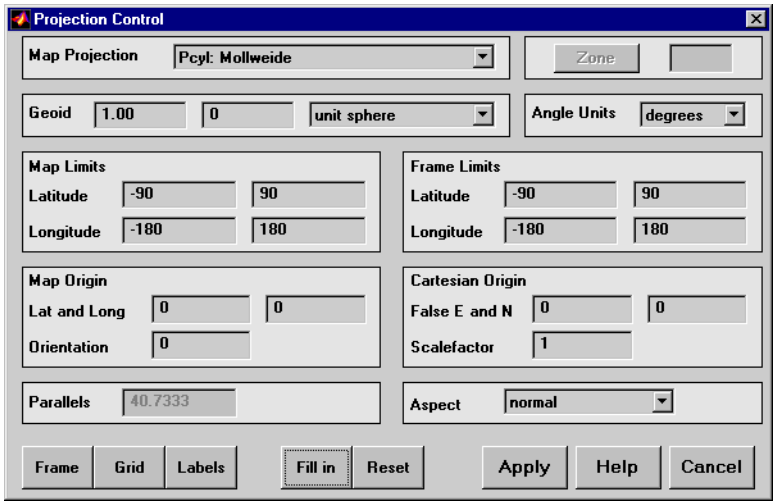
`axesmui` activates the **Projection Control** box for the current map axes.

`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 *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'`.

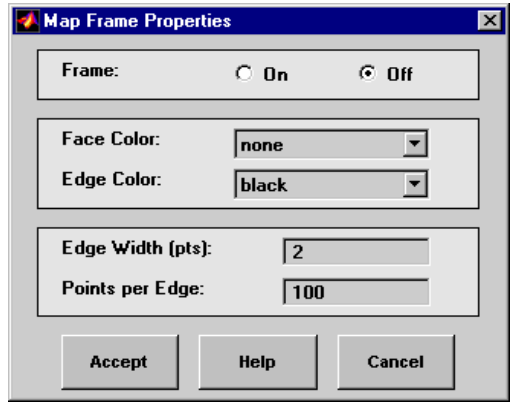
The **Apply** button is used to apply the projection and display settings to the current map, which results in the map being reprojected.

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.

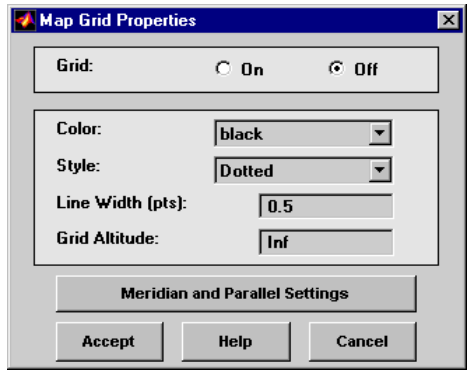
The **Points per Edge** edit box is used to enter the number of points used to display each edge of the map frame.

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.

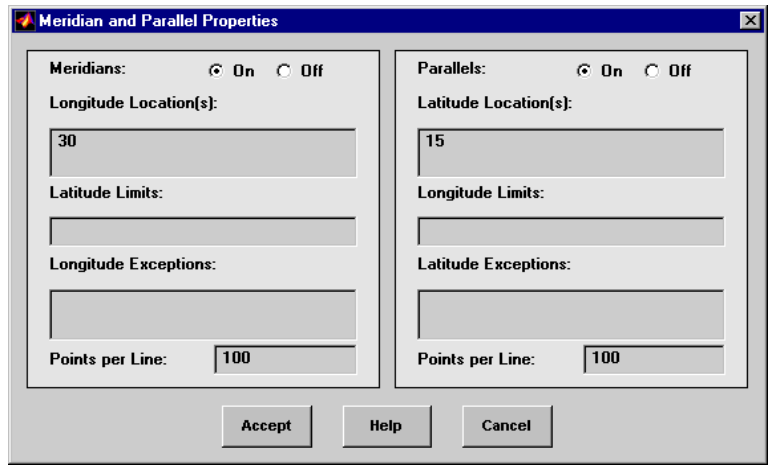
The **Meridian and Parallel Settings** button brings up the **Meridian and Parallel Properties** dialog box, which allows the properties of the meridian and parallel grid lines to be modified.

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.

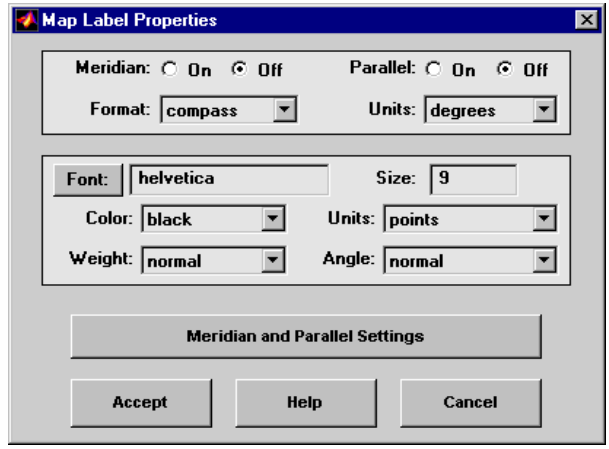
The **Points per Line** edit boxes are used to enter the number of points used to plot each meridian and each parallel grid line. The default value is 100 points.

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.

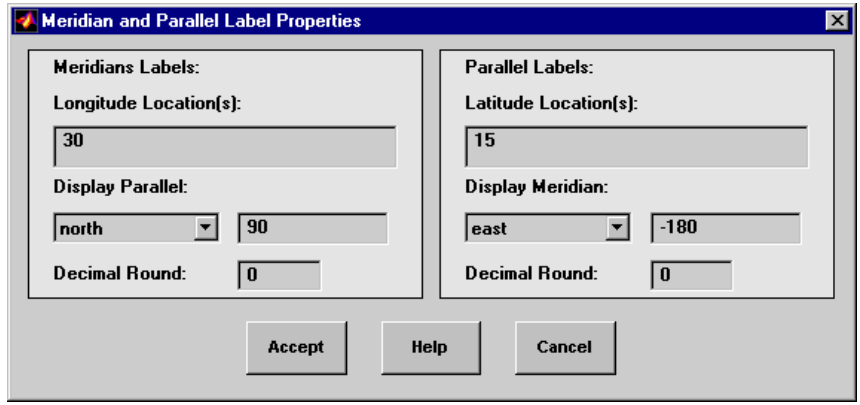
The **Meridian and Parallel Settings** button brings up the **Meridian and Parallel Label Properties** dialog box, which allows modification of properties specific to the meridian and parallel grid labels.

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

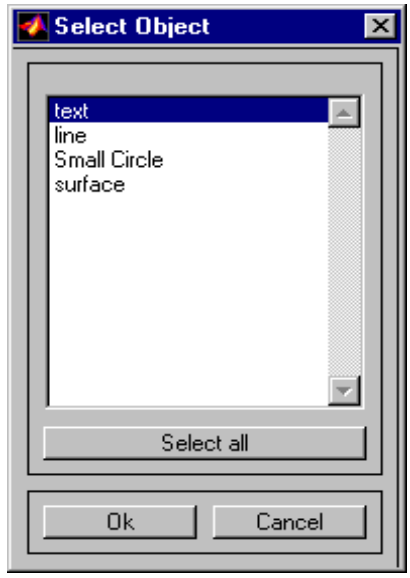
Command Line	Maptool
clmo	**Tools⇒Delete⇒Object**

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

`colorm`

Purpose

Create colormaps for an indexed regular matrix map

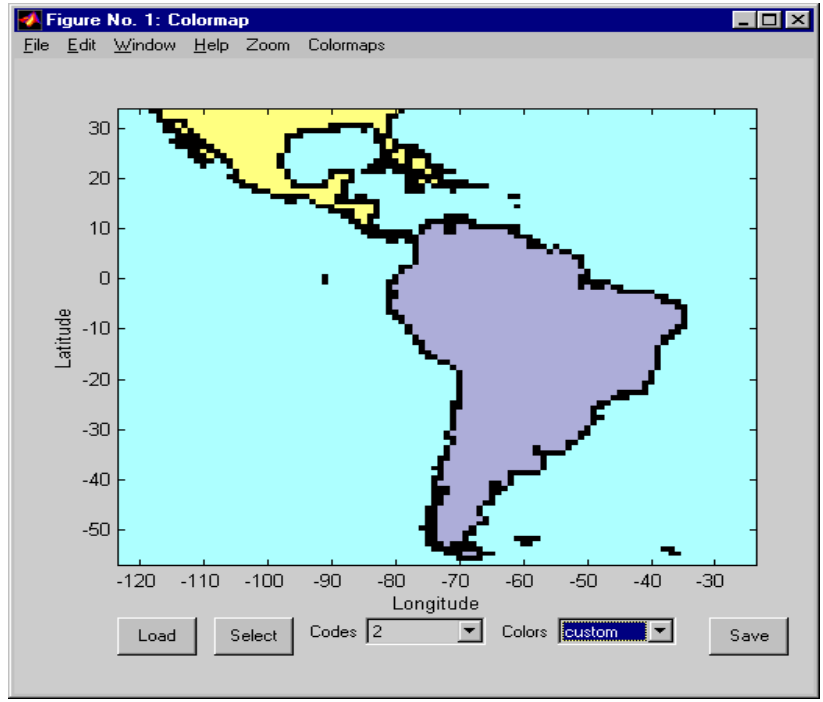
Activation

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

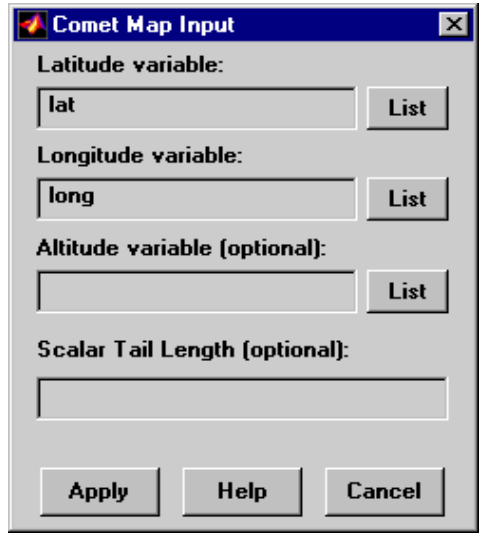
Command Line	Maptool
cometm comet3m	**Map⇒Comet**

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

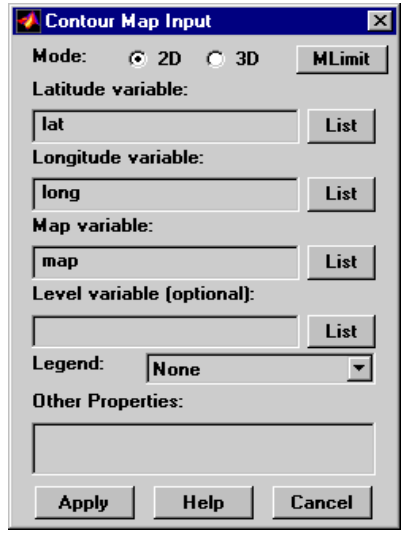
Command Line	Maptool
contorm contor3m	**Map⇒Contours**

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

`contorm, contor3m`

Purpose

Create and assign a colormap to a digital elevation matrix map

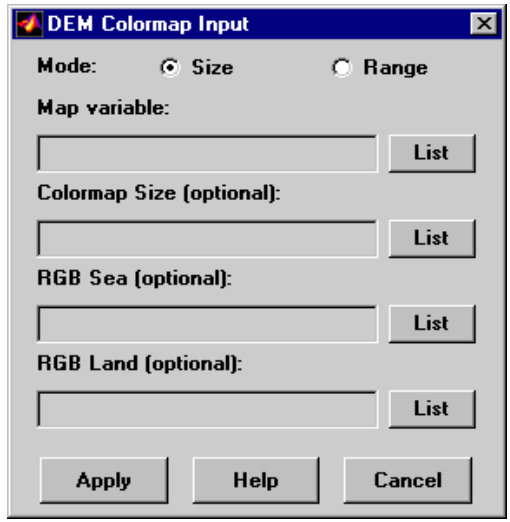
Activation

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

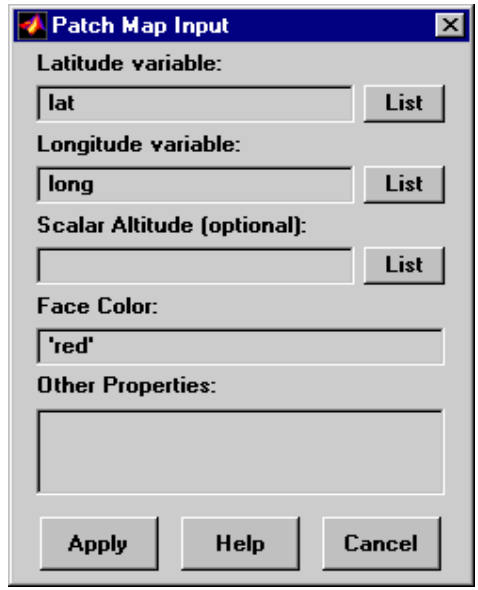
Command Line	Maptool
fillm fill3m patchm patchesm	**Map⇒Patch**

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')

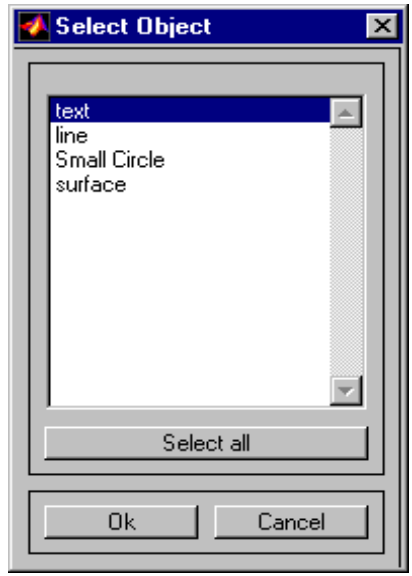
Description

`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.

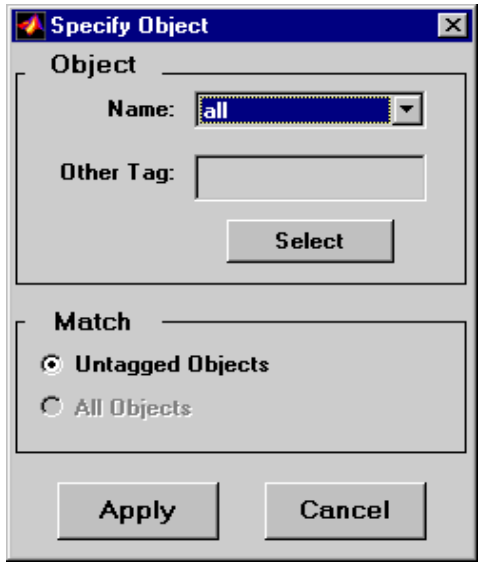
Controls

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

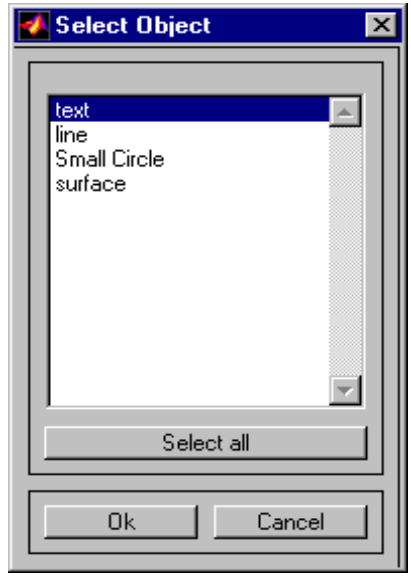
Activation

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

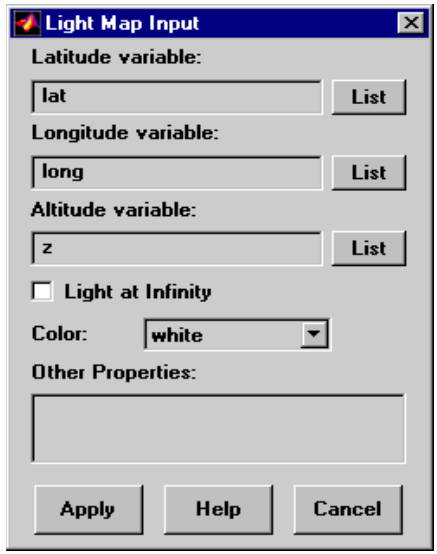
Command Line	Maptool
lightm	**Map⇒Light**

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

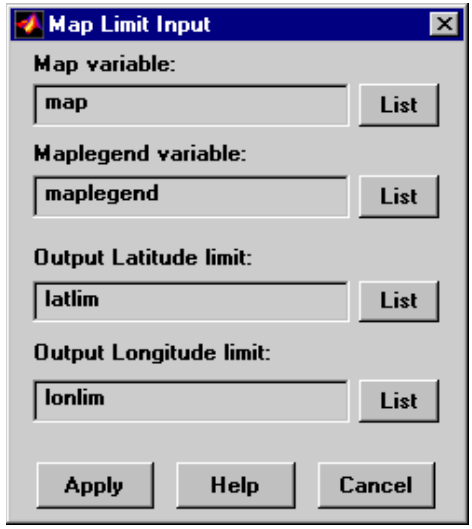
Activation

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

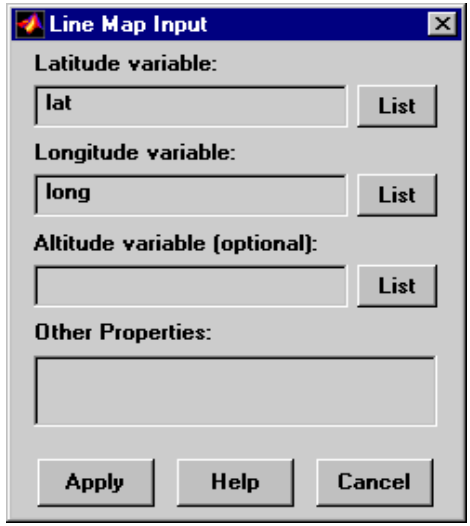
Command Line	Maptool
linem plotm plot3m	**Map⇒Lines**

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
maptoolmaptool(*PropertyName*,PropertyValue) maptool(*ProjectionFile*,...) h = maptool(...)

Description

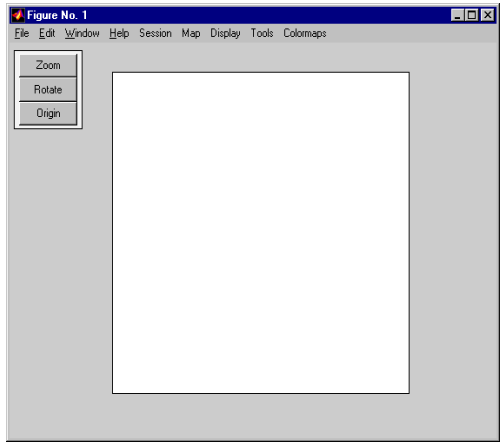
`maptool` creates a figure window with a map axes and activates the **Projection Control** dialog box for defining map projection and display properties. The figure window features a special menu bar that provides access to most of the Mapping Toolbox GUIs.

`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

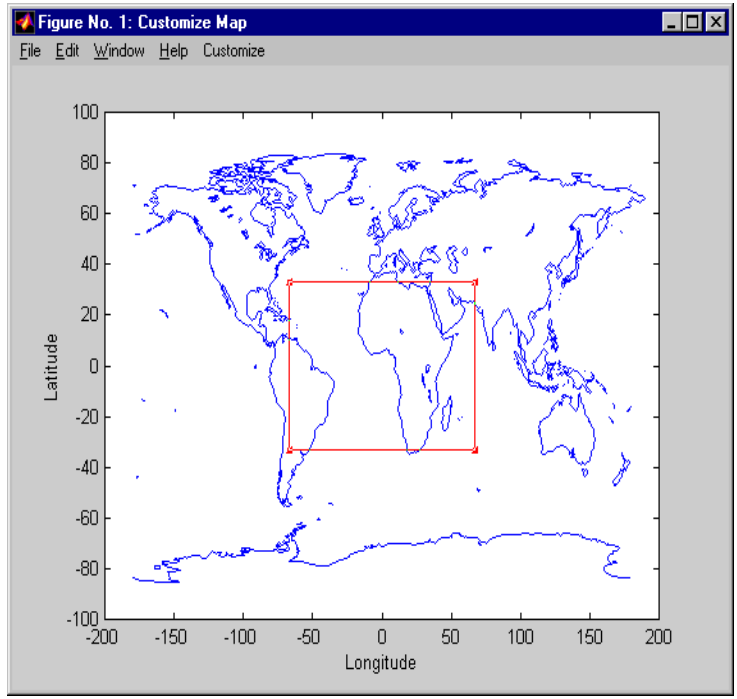
`maptrim(lat,lon)` displays the supplied map data in a new figure window and allows a region of the map to be selected and saved in the workspace. `lat` and `lon` must be vector map data. The output can be line, patch, or regular surface (matrix) data. If patch map output is selected, the inputs `lat` 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.

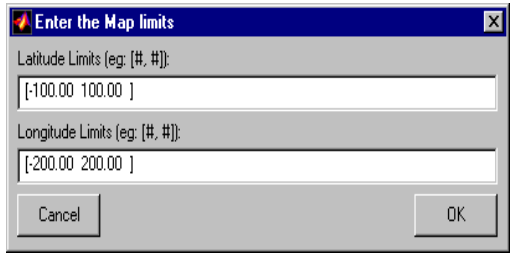
Controls



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

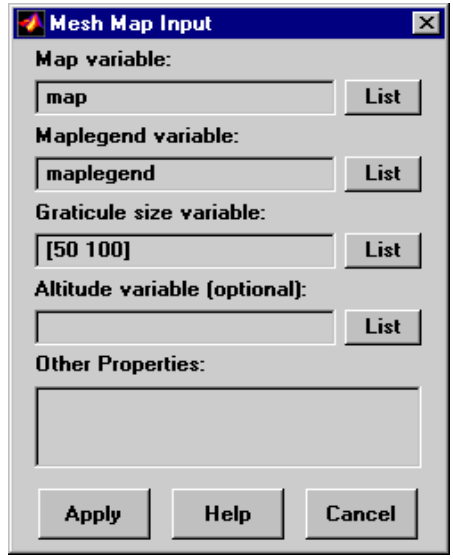
Command Line	Maptool
meshm	**Map⇒Regular Surfaces**

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)`.

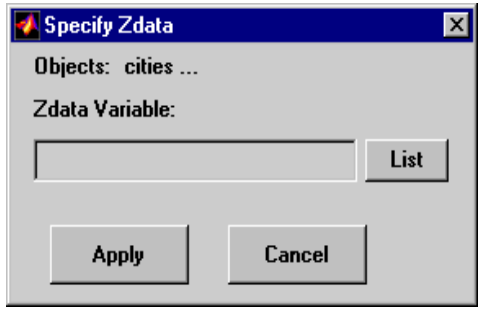
`mlayers(cellarray,h)` assigns the layers specified by `cellarray` to the map axes specified by the handle `h`.

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.

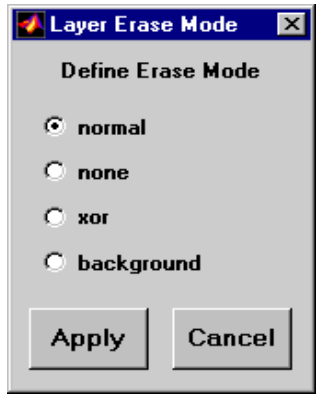


The **Highlight** button is used to toggle the selected map layer between highlighted and normal display.

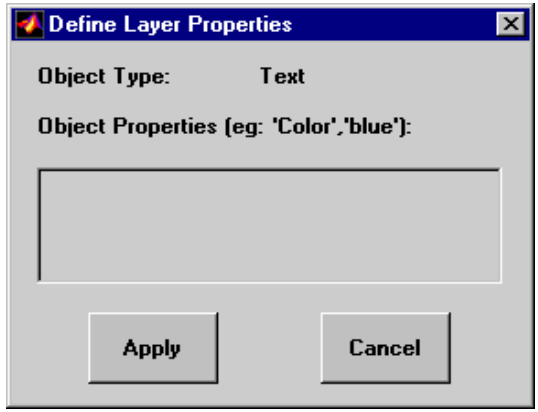
The **Members** button brings up a list of members of the selected map layer. Members of a layer are defined by their `Tag` property.

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`.

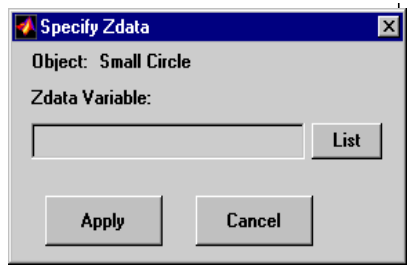
Controls

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.

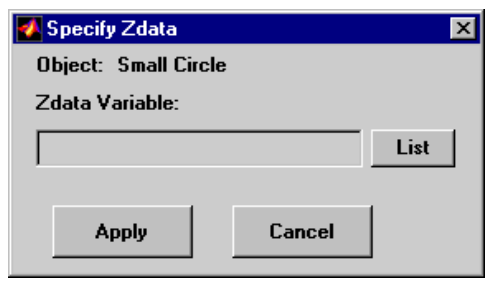


The **Highlight** button highlights all objects belonging to the selected object set.

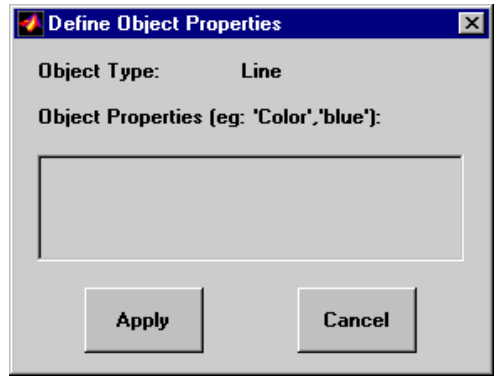
The **Tag** button brings up an **Edit Tag** dialog box, which allows the tag of all members of the selected object set to be modified.

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

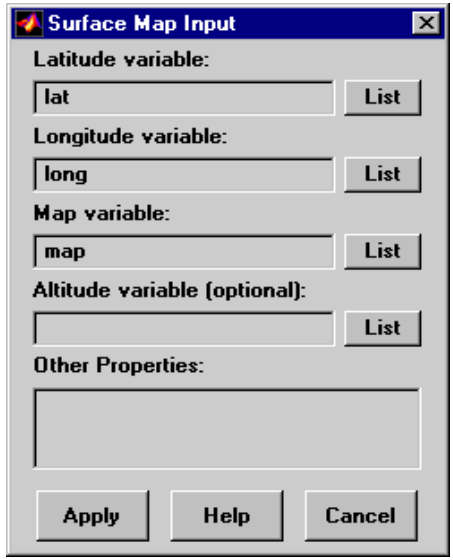
Command Line	Maptool
pcolorm surfacem surfm	**Map⇒General Surfaces**

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

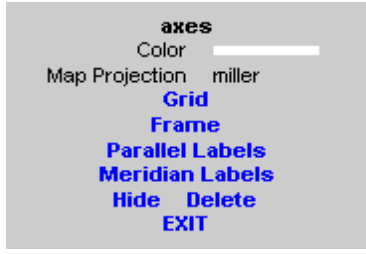
Alternate-clicking on a mapped object activates a property editor, which allows modification of some basic properties of the object through simple mouse clicks and drags. The objects supported by this editor are map axes, lines, text, patches, and surfaces, and the properties supported for each object 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

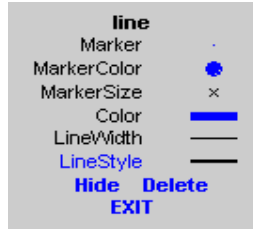
Click-and-Drag Property Editor

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.

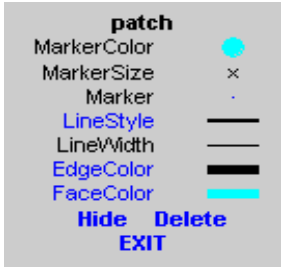


Click-and-Drag Editor for a line object

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.



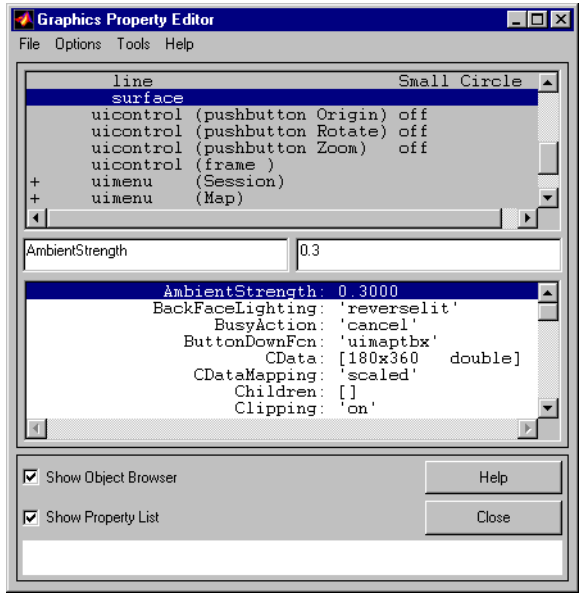
Click-and-Drag Editor for a patch object

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.

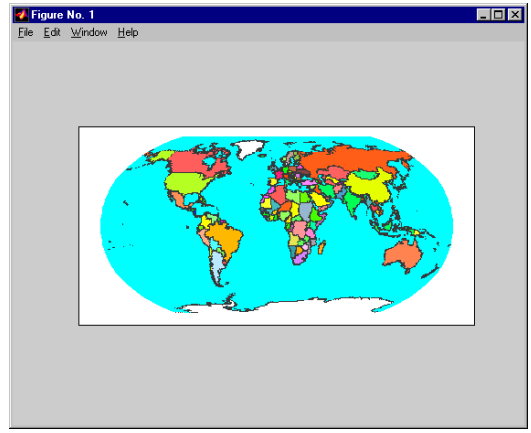
The **Process** button appears if the handle `h` is not provided. In this case, the `(x,y)` coordinates of the desired point are entered into the edit boxes. Pressing the **Process** button performs the data queries and displays the results.

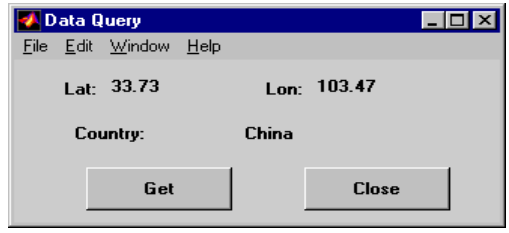
Pressing the **Close** button closes the query dialog box.

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)})
```





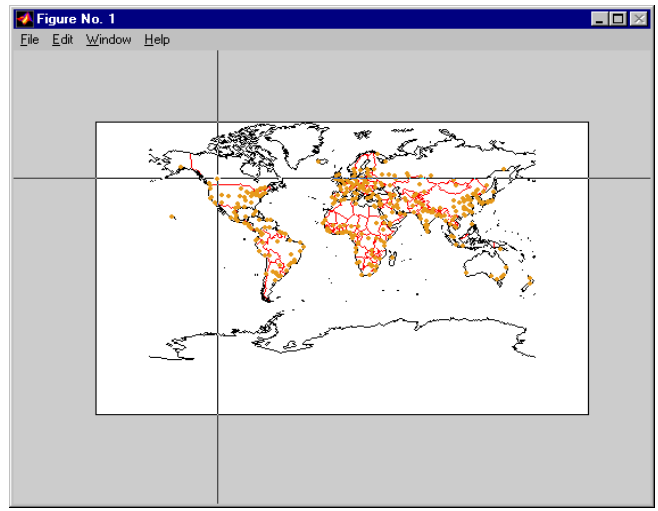
The next example makes use of a user-defined query to display city names for map points specified by a mouse click.

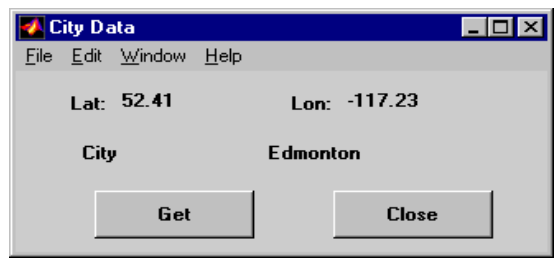
```
axesm miller
load worldlo
lat = [PPtext.lat]';
lon = [PPtext.long]';
mat = strvcat(PPtext.string);
displaym(POline)
displaym(PPpoint)
qrydata(gca,'City Data',{'City','qrytest',lat,lon,mat})
```

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

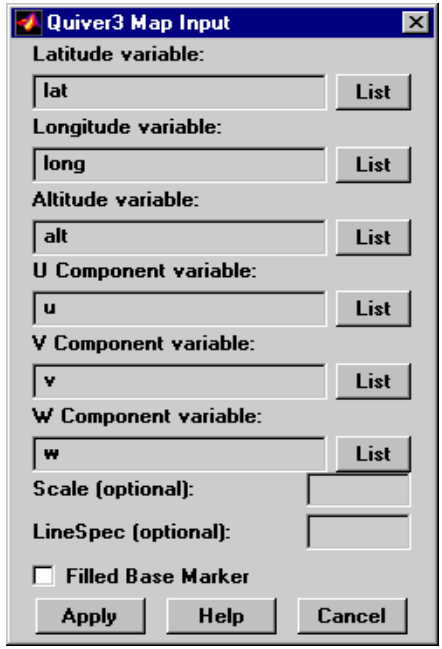
Command Line	Maptool
quiver3m	**Map⇒Quiver 3D**

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`

Purpose

Project a 2-D quiver plot onto the current map axes

Activation

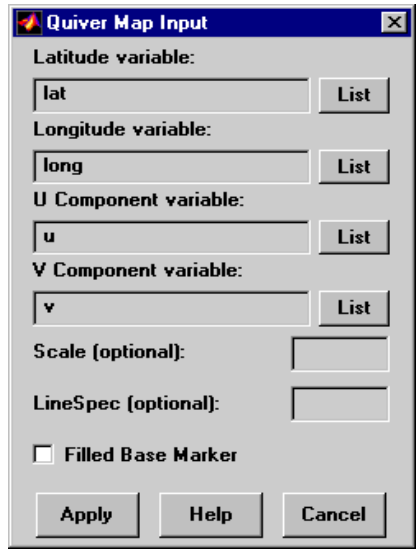
Command Line	Maptool
quiverm	**Map⇒Quiver 2D**

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`

Purpose

Project a symbol map on the current map axes

Activation

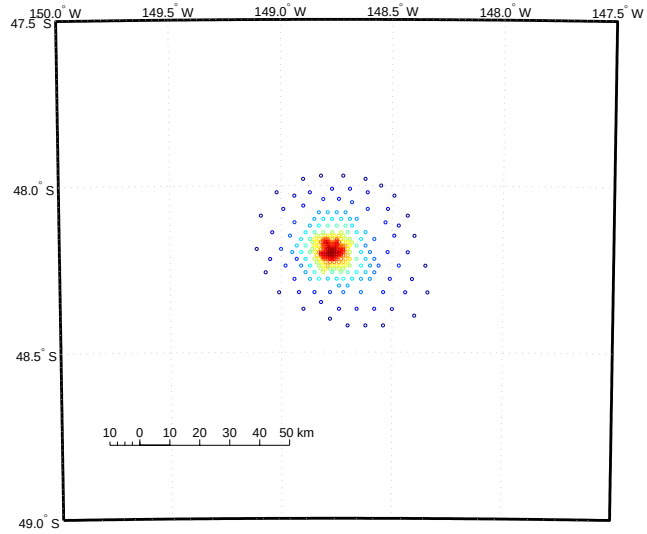
Command Line	Maptool
scatterm	**Map⇒Scatter**

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`

Purpose

Display small circles on a map axes

Activation

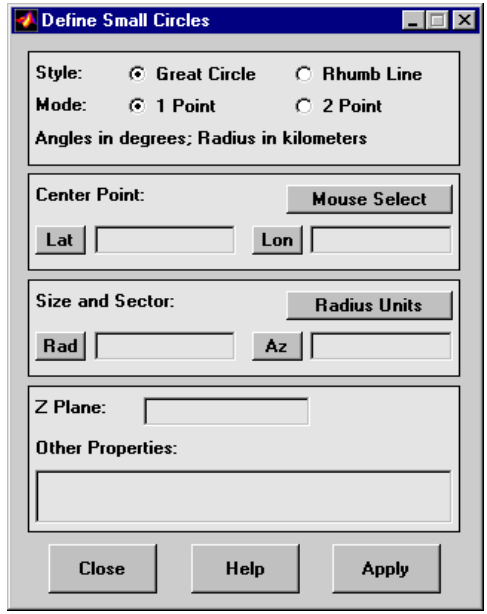
Command Line	Maptool
scirclui scirclui(h)	**Display⇒Small Circles**

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.

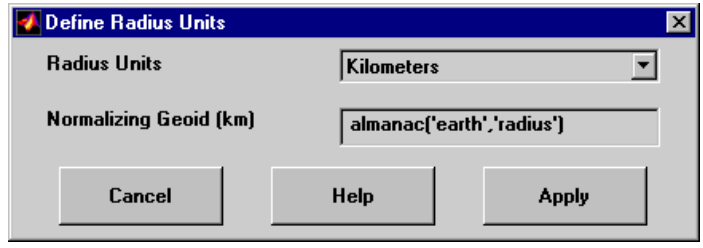
The **Other Properties** edit box is used to specify additional properties of the small circles to be projected, such as `'Color','b'`. String entries must be enclosed in quotes.

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`

Purpose

Encode a regular surface map

Activation

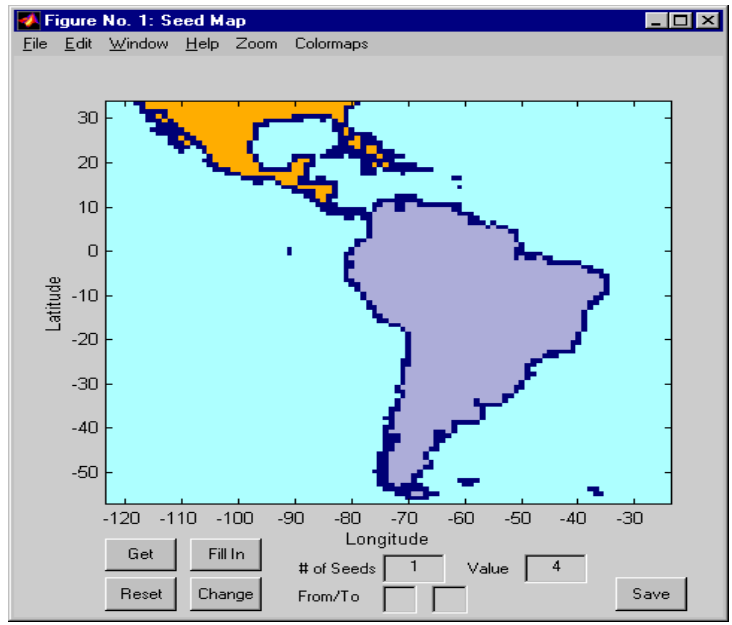
Command Line
seedm(map,maplegend)

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`

Purpose

Show mapped objects

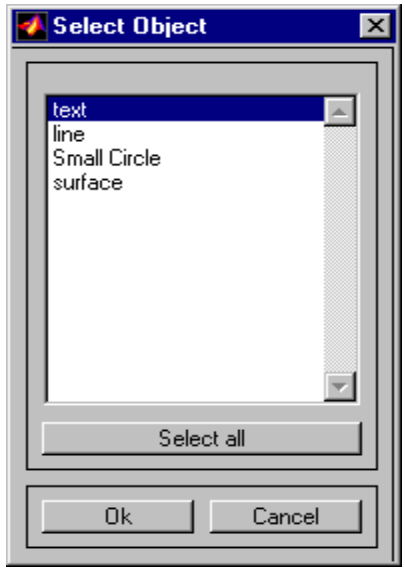
Activation

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

stem3m

Purpose

Project a stem plot onto the current map axes

Activation

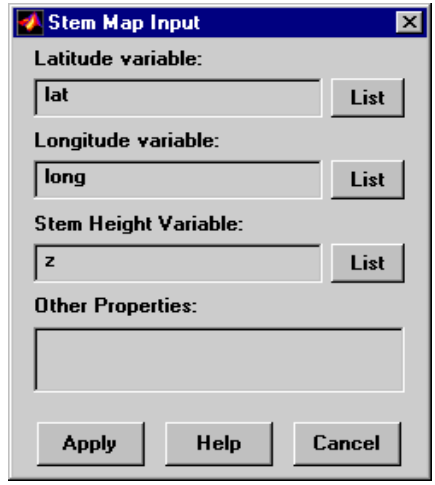
Command Line	Maptool
stem3m	**Map⇒Stem**

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



The **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`

surfdist

Purpose

Interactively calculate distance, azimuth, and reckoning

Activation

Command Line	Maptool
surfdist surfdist(h) surfdist([])	**Display⇒Surface⇒Distances**

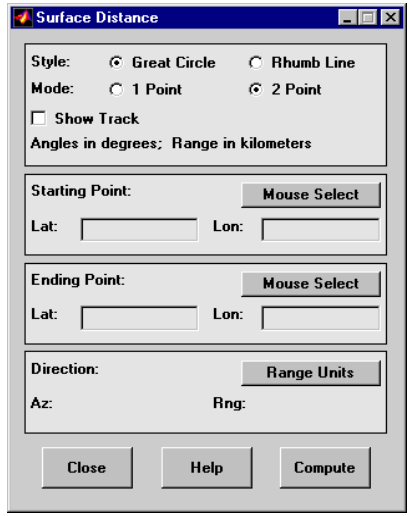
Description

`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

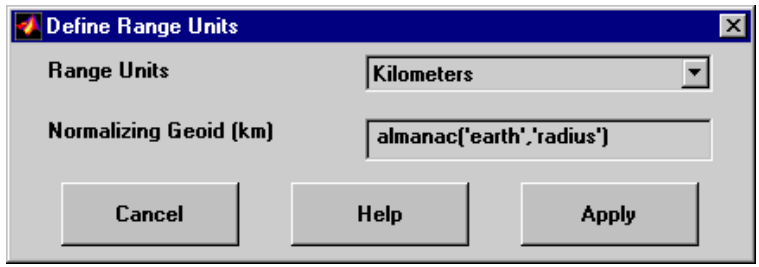
used to compute the ending point of the track in one-point mode. During two-point mode, the **Direction** controls are disabled, but the azimuth and range values resulting from the surface distance calculation are displayed.

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.

surflm

Purpose

Display a lighted matrix map warped to a projected graticule

Activation

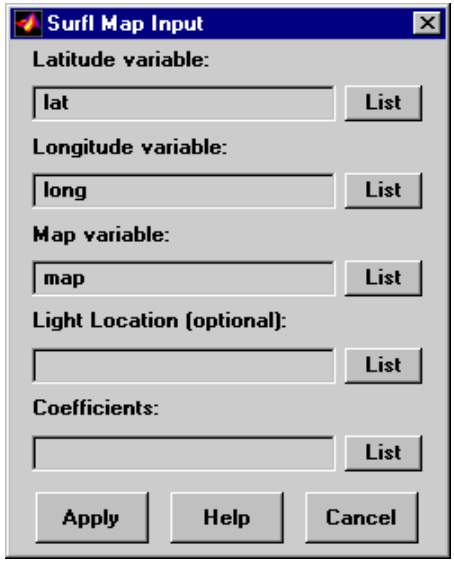
Command Line
surflm

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`

tagm

Purpose

Edit the tag of mapped objects

Activation

Command Line
tagm
tagm(h)

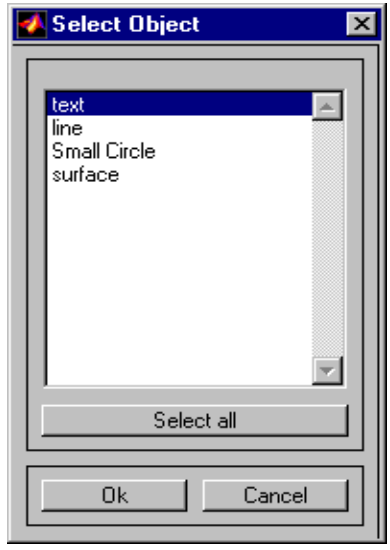
Description

`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`

textm

Purpose

Project text on the current map axes

Activation

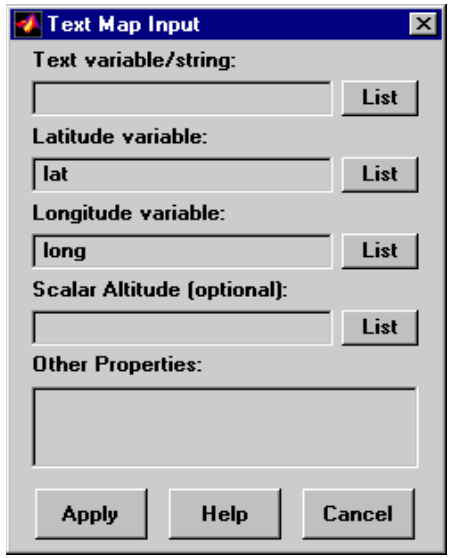
Command Line	Maptool
textm	**Map⇒Text**

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`

trackui

Purpose

Display great circles and rhumb lines on a map

Activation

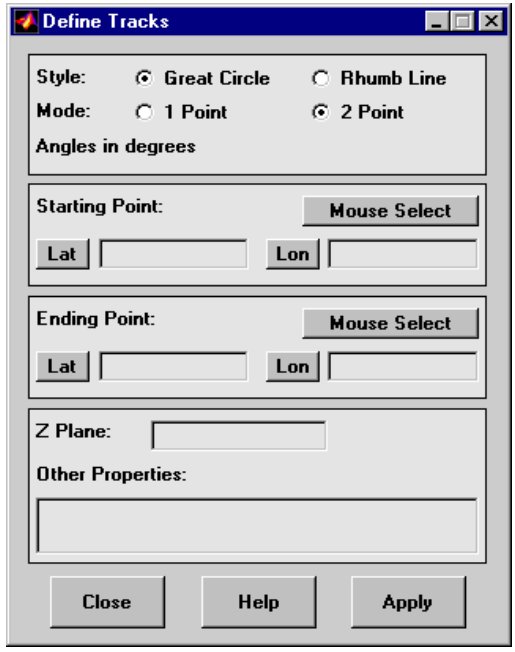
Command Line	Maptool
trackui trackui(h)	**Display⇒Tracks**

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.

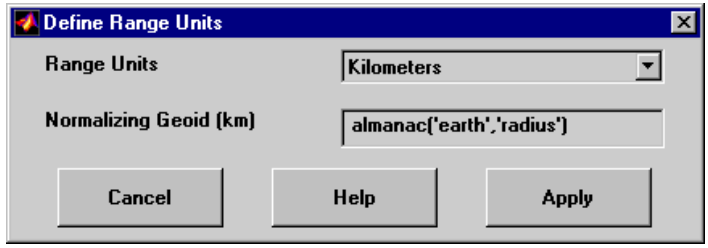
The **Other Properties** edit box is used to specify additional properties of the tracks to be projected, such as `'Color','b'`. String entries must be enclosed in quotes.

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

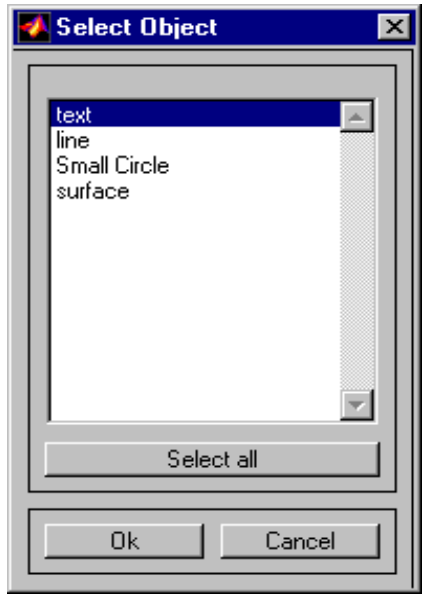
`zdatam` brings up a **Select Object** dialog box for selecting mapped objects and adjusting their `ZData` property. Upon selecting the objects, the **Specify Zdata** dialog box is activated, in which the new `ZData` variable is entered. Note that not all mapped objects have the `ZData` property (for 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`.

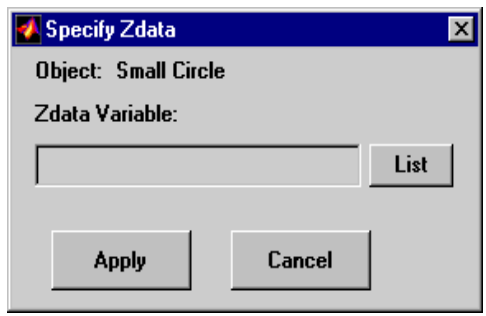
Controls

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

12

External Data Reference

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.
demdataui	Digital Elevation Map Data User Interface.
dted	Read U. S. Department of Defense Digital Terrain Elevation Data (DTED) data
dteds	U. S. Department of Defense Digital Terrain Elevation Data (DTED) data filenames
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.
globedem	Read GLOBE 30-arc-second (1 km) Digital Elevation Map.
globedems	GLOBE 30-arc-second (1 km) Digital Elevation Map filenames.
gshhs	Read Global Self-consistent Hierarchical High-resolution Shoreline data.

Function	Description
gtopo30	Read GTOPO30 30-arc-second (1 km) global elevation data.
gtopo30s	GTOPO30 30-arc-second (1 km) global elevation data filenames.
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.
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.
vmap0ui	Vector Map Level 0 User Interface.

Alphabetical listing of External Interface Functions

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.

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,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,-,-,:;
```

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

Digital elevation map data user interface

Activation

demdataui

Description

demdataui is a Graphical User Interface to extract Digital Elevation Map data from a number of external data files.

The `demdataui` panel lets you read data from a variety of high-resolution Digital Elevation Maps (DEMs). These DEMs range in resolution from about 10 kilometers to 100 meters or less. The data files are available over the internet at no cost, or (in some cases) on CD-ROMs for varying fees. demdataui reads ETOPO5, TerrainBase, GTOPO30, GLOBE, satellite bathymetry and DTED data. See the links under “See also” for more information on these datasets. `demdataui` looks for these external data files on the Matlab path and, for some operating systems, on CD-ROM disks.

You use the list to select the source of data and the map to select the region of interest. When you click the **Get** button, data is extracted and displayed on the map. Use the **Save** button to save the data in a MAT-file or to the base workspace for later display. The **Close** button closes the window.

Controls

The Map

The map controls the geographic extent of the data to be extracted. `demdataui` extracts data for areas currently visible on the map. Use the mouse to zoom in or out to the area of interest. See `zoom` for more on zooming.

Some data sources divide the world up into tiles. When extracting, data is concatenated across all visible tiles. The map shows the tiles in light yellow with light gray edges. When data resolution is high, extracting data for large area may take much time and memory.An approximate count of the number of points is shown above the map. Use the **Samplefactor** slider to reduce the amount of data.

The List

The list controls the source of data to be extracted. Click on a name to see the geographic coverage in light yellow. The sources list shows which data sources were found when `demdataui` started.

demdataui searches for data files on the Matlab path. On some computers, demdataui will also check for data files on the root level of letter drives. demdataui looks for the following data: etopo5: new_etopo5.bil or etopo5.northern.bat and etopo5.southern.bat files. tbase: tbase.bin file. satbath: topo_6.2.img file. gtopo30: a directory that contains subdirectories with the datafiles. For example, demdataui would detect gtopo30 data if a directory on the path contained the directories E060S10 and E100S10, each of which holds the uncompressed data files. globedem: a directory that contains data files and in the subdirectory "/esri/hdr" the "*.hdr" header files. dted: a directory that has a subdirectory named DTED. The contents of the DTED directory are more subdirectories organized by longitude and, below that, the DTED data files for each latitude tile. See the help for functions with the data source names for more on the data attributes and internet locations.

The Samplefactor Slider

The **Sample Factor** slider allows you to reduce the density of the data. A sample factor of 2 returns every second point. The current sample factor is shown on the slider.

The Get Button

The **Get** button reads the currently selected data and displays it on the map. Press the standard interrupt key combination for your platform to interrupt the process.

The Clear Button

The **Clear** button removes any previously read data from the map.

The Save Button

The **Save** button saves the currently displayed data to a Mat-file or the base workspace. If you choose to save to a file, you will be prompted for a file name and location. If you choose to save to the base workspace, you you can choose the variable name under which the data will be stored. The results are stored as a geographic data structure. Use load and displaym to redisplay the data from a file on a map axes. To display the data in the base workspace, use displaym. To gain access to the data matrices, subscript into the structure (e.g. map = demdata(1).map; maplegend = demdata(1).maplegend). Use worldmap to create easy displays of the elevation data (e.g. worldmap(map,maplegend,'ldem3d')). Use meshm to add regular matrix maps

to existing displays, or `surfm` or a similar function for general matrix maps (e.g. `meshm(map,maplegend)` or `surfm(latgrat,longrat,map)`).

The Close Button

The **Close** button closes the `demdataui` panel.

See Also

`etopo5` `tbase` `gtopo30` `globedem` `dted` `satbath` `vmap0ui`

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] = dted(*dirname*,scalefactor,latlim,lonlim) reads and concatenates data from multiple files within a DTED CD-ROM or directory structure. The dirname input is a string with the name of a directory containing the DTED directory. Within the DTED directory are subdirectories for each degree of longitude, each of which contain files for each degree of latitude. For DTED CD-ROMs, dirname is the device name of the CD-ROM drive.

[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]);

[map,maplegend,UHL,DSI,ACC] = dted('f:',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

dteds

Purpose

DTED file names

Syntax

```
fname = dteds(latlim,lonlim)
fname = dteds(latlim,lonlim,level)
```

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.

Determining which files are needed to cover a particular region requires knowledge of the DTED database naming conventions. This function constructs the file names for a given geographic region based on these conventions.

Description

fname = dteds(latlim,lonlim) returns 'Level 0' DTED file names (directory and name) required to cover the geographic region specified by latlim and lonlim.

fname = dteds(latlim,lonlim,level) controls the level for which the file names are generated. Valid inputs for the LEVEL of the DTED files include 0, 1, or 2.

Examples

Which files are needed for Cape Cod.

```
latlim = [ 41.15 42.22]; lonlim = [-70.94 -69.68];
dteds(latlim,lonlim,1)

ans =

    '\DTED\W071\N41.dt1'
    '\DTED\W070\N41.dt1'
    '\DTED\W071\N42.dt1'
    '\DTED\W070\N42.dt1'
```

See Also

`dted` Read DTED digital elevation map data

egm96geoid

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 =
   53.4809
```

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

etopo5

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.

fipsname

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

Purpose

Read elevation data from GLOBE Digital Elevation Map files into a regular matrix map

Syntax

[map,maplegend] = globedem(*filename*,scalefactor)

[map,maplegend] = globedem(*filename*,scalefactor,latlim,lonlim)

[map,maplegend] = globedem(*dirname*,scalefactor,latlim,lonlim)

Background

GLOBE, the Global Land One-km Base Elevation data, was compiled by the National Geophysical Data Center from more than 10 different sources of gridded elevation data. GLOBE may be considered a higher resolution sucessor to TerrainBase. The dataset consists of 16 tiles, each covering 50 by 90 degrees. Tiles require as much as 60 MB of storage. Uncompressed tiles take between 100 and 130 MB.

Description

[map,maplegend] = globedem(*filename*,scalefactor) reads the GLOBE DEM files and returns the result as a regular matrix map. The filename is given as a string which does not include an extension. GLOBEDEM first reads the ESRI header file found in the subdirectory '/esri/hdr/' and then the binary data file filename. If the files are not found on the Matlab path, they may be selected interactively. Scalefactor is an integer, which when equal to 1 gives the data at its full resolution. When scalefactor is an integer n larger than one, every nth point is returned. The map data is returned as an array of elevations and associated regular matrix map legend. Elevations are given in meters above mean sea level using WGS 84 as a horizontal datum.

[map,maplegend] = globedem(*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 latitude and longitude in degrees. The elements of latlim and lonlim must be in ascending order.

[map,maplegend] = globedem(*dirname*,scalefactor,latlim,lonlim) reads and concatenates data from multiple files within a GLOBE directory tree. The dirname input is a string with the name of the directory which contains both the uncompressed files data files and the ESRI header files.

Remarks

The Mapping Toolbox reads data from GLOBE version 1.0. 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://ftp.ngdc.noaa.gov/GLOBE_DEM/data/elev/>
```

The data and some documentation is also available over the World-Wide-Web from:

```
<http://www.ngdc.noaa.gov/seg/topo/globe.shtml>
```

Examples

Determine which file contains the area around Cape Cod.

```
latlim = [41 42.5]; lonlim = [-73 -69.9];
globedems(latlim,lonlim)

ans =

    'f10g'
```

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] = globedem([],20);
size(map)
ans =
   300   540
```

Extract a subset of the data for Massachusetts at the full resolution.

```
latlim = [41 42.5]; lonlim = [-73 -69.9];
[map,maplegend] = globedem('f10g',1,latlim,lonlim);
size(map)
ans =
   181 373
```

Replace the `NaN`s in the ocean with –1 to color them blue.

```
map(isnan(map)) = -1;
```

Extract some data for southern Louisiana in an area that straddles two tiles. Provide the name of the directory containing the data files, and let `globedem` determine which files are required, read from the files, and concatenate the data into a single regular matrix map.

```
latlim =[28.61 31.31]; lonlim = [-91.24 -88.62];
globedems(latlim,lonlim)

ans =

    'e10g'
    'f10g'

[map,maplegend] =
globedem('d:\externalData\globe\elev',1,latlim,lonlim);
size(map)

ans =

        325.00        315.00
```

See Also

demdataui	Interactive tool to read digital elevation data
dted	Read Digital Terrain Elevation Data (DTED) data
gtopo30	Read GTOPO30 digital elevation data
satbath	Read global 2-minute topography from satellite bathymetry
tbase	Read data from the TerrainBase dataset
usgsdem	Read USGS digital elevation maps

References

<http://www.ngdc.noaa.gov/seg/topo/globe.shtml>

globedems

Purpose

GLOBE DEM file names

Syntax

```
fnames = globedems(latlim,lonlim)
```

Background

GLOBE, the Global Land One-km Base Elevation data, was compiled by the National Geophysical Data Center from more than 10 different sources of gridded elevation data. The dataset consists of 16 tiles, each covering 50 by 90 degrees. Determining which files are needed to cover a particular region generally requires consulting an index map. This function takes the place of such a reference by returning the file names for a given geographic region.

Description

fname = globedems(latlim,lonlim) returns a cellarray of the file names covering the geographic region for GLOBEDEM digital elevation maps. The region is specified by scalar latitude and longitude points, or two element vectors of latitude and longitude limits in units of degrees.

Remarks

The Mapping Toolbox reads data from GLOBE version 1.0. GLOBEDEM first reads the corresponding ESRI header file found in the subdirectory '/esri/hdr/' and then the binary data file (with no extension)

Examples

Which files are needed for southern Louisiana.

```
latlim =[28.61 31.31]; lonlim = [-91.24 -88.62];
globedems(latlim,lonlim)

ans =

    'e10g'
    'f10g'
```

See Also

`globedem` Read GLOBE digital elevation map data

References

<http://www.ngdc.noaa.gov/seg/topo/globe.shtml>

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)

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 `NaN`s in the ocean with –1 to color them blue.

```
map(isnan(map)) = -1;
```

Extract the data for Thailand. This area straddles two tiles. The data is on CD number 3 distributed by the USGS.

```
latlim = [5.22 20.90]; lonlim = [96.72 106.38];
gtopo30s(latlim,lonlim)

ans =

    'e060n40'
    'e100n40'

[map,maplegend] = gtopo30('f:\',3,latlim,lonlim);
size(map)

ans =
```

```
628 388
```

See Also

`gtopo30s`	Filenames for GTOPO30 data from geographic area.
`demdataui`	Digital elevation map data user interface
`dted`	Read Digital Terrain Elevation Data (DTED) data
`globedem`	Read 1 km GLOBE DEM 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 8 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.

satbath

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

tbase

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

tgrline

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

tigermif

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

tigerp

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

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 30meters. 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

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

Purpose

Vector Map Level 0 (VMap0) user interface

Activation

vmap0ui
vmap0ui(*devicename*) or vmap0ui devicename

Description

vmap0ui is a graphical user interface to read Vector Map Level 0 CD-ROMs. The VMAP0 is the most detailed global database of vector map data available to the public. `vmap0ui` attempts to automatically detect which drive contains a VMAP0 CD-ROM. If `vmap0ui` can't find the CD, use the following calling form.

vmap0ui(*devicename*) or vmap0ui devicename uses the specified logical device name for the CD-ROM drive containing the VMAP0 CD-ROM. Under the Macintosh operating system, it would be 'VMAP'. Under the Windows OS, it could be 'f:' or 'g:' or some other letter. Under UNIX, it could be '\cdrom\'. Check your computer's documentation for the correct device name.

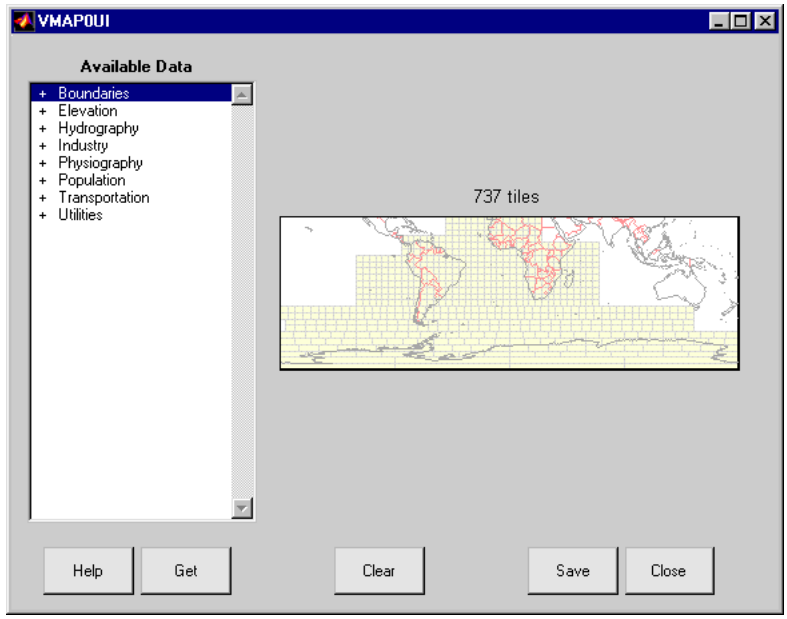
VMAP0 CD-ROMs are available from

USGS Information Services (Map and Book Sales)
Box 25286
Denver Federal Center
Denver, CO 80225
Telephone: (303) 202-4700
Fax: (303) 202-4693

The price as of early 1998 was $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>

Controls



The `vmap0ui` screen lets you read data from the Vector Map Level 0 (VMAP0). The VMAP0 is the most detailed map database available to the public.

You use the list to select the type of data and the map to select the region of interest. When you click the **Get** button, data is extracted and displayed on the map. Use the **Save** button to save the data in a MAT-file or to the base workspace for later display. The *Close* button closes the window.

The Map

The **Map** controls the geographic extent of the data to be extracted. `vmap0ui` extracts data for areas currently visible on the map. Use the mouse to zoom in or out to the area of interest. Type HELP ZOOM for more on zooming.

The VMAP0 divides the world up into tiles of about 5 by 5 degrees. When extracting, data is returned for all visible tiles, including those parts of the tile that are outside of the current view. The map shows the VMAP0 tiles in light yellow with light gray edges. The data density is high, so extracting data for a large number of tiles may take much time and memory. A count of the number of visible tiles is above the map.

The List

The **List** controls the type of data to be extracted. The tree structure of the list reflects the structure of the VMAP0 database. Upon starting `vmap0ui`, the list shows the major categories of VMAP data, called themes.Themes are subdivided into features, which consist of data of common graphic type (patch, line, point or text) or cultural type (airport, roads, railroads). Double-click on a theme to see the associated features. Features may have properties and values, for example a railroad tracks property, with values single or multiple. Double-click on a feature to see the associated properties and values. Double clicking on an open theme or feature will close it. When a theme is selected, `vmap0ui` will get all of the associated features. When a feature is selected, `vmap0ui` will get all of that feature data. When properties and values are selected, `vmap0ui` will get the data for any of the properties and values match (i.e., the union operation).

The Get Button

The **Get** button reads the currently selected VMAP0 data and displays it on the map. Use the **Cancel** button on the progress bar to interrupt the process. For a quicker response, press the standard interrupt key combination for your platform.

The Clear Button

The **Clear** button removes any previously read data from the map.

The Save Button

The **Save** button saves the currently displayed VMAP0 data to a Mat-file or the base workspace. If you choose to save to a file, you will be prompted for a file name and location. If you choose to save to the base workspace, you will be notified of the variable names that will be overwritten. The results are stored as geographic data structures with variables names based on theme and feature names. Use `load` and `displaym` to redisplay the data from a file on a map axes. You can also use the `mlayers` GUI to read and display the data from a file. To display the data in the base workspace, use `displaym`. To display all of the geographic data structures, use `rootlayr; displaym(ans)`. To display all of the geographic data structures using the `mlayers` GUI, type `rootlayr; mlayers(ans)`.

The Close Button

The **Close** button closes the vmap0ui panel.

See Also

vmap0data mlayers displaym extractm

[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.

Bibliography

A

Geographic Terms

Glossary Notes

This glossary of geographical terms is drawn extensively from *An Album of Map Projection, U.S. Geological Survey Professional Paper 1453*, by John P. Snyder and Philip M. Voxland.

The purpose of this glossary is to assist the user in understanding the Mapping Toolbox. For this reason, some terms in this glossary are specifically defined as they are used in this guide rather than as they might be more generally used in the geographic community.

Glossary

Antipodes - Two points on opposite sides of a planet.

Arc-second - 1/3600th of a degree (1 second) of latitude or longitude.

ARC/Info - A largely UNIX-oriented GIS developed and distributed by the Environmental Systems Research Institute, Inc.

Aspect - The conceptual placement of a projection system in relation to the Earth's axis (direct, normal, polar, equatorial, oblique, and so on).

Authalic projection - *See* Equal-area projection.

Axes - *See* Map axes.

Azimuth - The angle a line makes with a meridian, taken clockwise from north.

Azimuthal projection - A projection on which the azimuth or direction from a given central point to any other point is shown correctly. When a pole is the central point, all meridians are spaced at their true angles and are straight radii of concentric circles that represent the parallels. Also called a zenithal projection.

Bathymetry - The measurement of water depths of oceans, seas, lakes, and other bodies of water.

Bowditch, Nathaniel - A late 18th/early 19th century mathematician, astronomer, and sailor who "wrote the book" on navigation. John Hamilton Moore's *The Practical Navigator* was the leading navigational text when Bowditch first went out to sea, and had been for many years. Early in his first voyage, however, Bowditch began noticing errors in Moore's book, which he recorded and later used in preparing an American edition of Moore's work. The revisions were to such an extent that Bowditch was named the principal author, and the title was changed to *The New American Practical Navigator*, published in 1802. In 1868 the U.S. Navy bought the copyright to the book, which is still commonly referred to as "Bowditch" and considered the "bible" of navigation.

Cartography - The art or practice of making charts or maps.

Central meridian - The meridian passing through the center off a projection, often a straight line about which the projection is symmetrical.

Central projection - A projection in which the Earth is projected geometrically from the center of the Earth onto a plane or other surface. The Gnomonic and Central Cylindrical projections are examples.

Choropleth - A map consisting of areas of equal value separated by abrupt boundaries and colored or shaded according to those values.

Complex curves - Curves that are not elementary forms such as circles, ellipses, hyperbolas, parabolas, and sine curves.

Composite projection - A projection formed by connecting two or more projections along common lines such as parallels of latitude, necessary adjustments being made to achieve fit. The Goode Homolosine projection is an example.

Conformal projection - A projection on which all angles at each point are preserved. Also called an orthomorphic projection.

Conceptually projected - The convenient way to visualize a projection system, although it may not correspond to the actual mathematical projection method.

Conic projection - A projection resulting from the conceptual projection of the Earth onto a tangent or secant cone, which is then cut lengthwise and laid flat. When the axis of the cone coincides with the polar axis of the Earth, all meridians are straight equidistant radii of concentric circular arcs representing the parallels, but the meridians are spaced at less than their true angles. Mathematically, the projection is often only partially geometric.

Constant scale - A linear scale that remains the same along a particular line on a map, although that scale may not be the same as the stated or nominal scale of the map.

Contour - All points that are at the same elevation above or below a specified datum.

Conventional aspect - *See* Normal aspect.

Correct scale - A linear scale having exactly the same value as the stated or nominal scale of the map, or a scale factor of 1.0. Also called true scale.

Cylindrical projection - A projection resulting from the conceptual projection of the Earth onto a tangent or secant cylinder, which is then cut lengthwise and laid flat. When the axis of the cylinder coincides with the axis of the Earth, the meridians are straight, parallel, and equidistant, while the parallels of latitude

are straight, parallel, and perpendicular to the meridians. Mathematically, the projection is often only partially geometric.

Dead reckoning - From “deduced reckoning,” the estimation of geographic position based on course, speed and time.

DEM (Digital Elevation Map/Model) - Elevation data in the form of a matrix map, generally on a regular grid. DEM also refers to the five primary types of digital elevation models produced by the U.S. Geological Survey, one of which is the 1-degree (3-arc-second resolution) model that is interfaced through the Mapping Toolbox.

Departure - The arc length distance along a parallel of a point from a given meridian.

Developable surface - A simple geometric form capable of being flattened without stretching. Many map projections can be grouped by a particular developable surface: cylinder, cone, or plane.

Direct aspect - *See* Normal aspect.

Distortion - A variation of the area or linear scale on a map from that indicated by the stated map scale, or the variation of a shape or angle on a map from the corresponding shape or angle on the Earth.

DMS - Degrees-minutes-seconds angle notation of the form ddd mm' ss". There are 60 seconds in a minute, and 60 minutes in a degree. In the Mapping Toolbox, when “dms” angles are represented by a single number, the format is dddmm.ss.

Ellipsoid - When used to represent the Earth, a solid geometric figure formed by rotating an ellipse about its minor (shorter) axis. Also called spheroid.

Equal-area projection - A projection on which the areas of all regions are shown in the same proportion to their true areas. Shapes may be greatly distorted. Also called an equivalent or authalic projection.

Equatorial aspect - An aspect of an azimuthal projection on which the center of projection or origin is some point along the Equator. For cylindrical and pseudocylindrical projections, this aspect is usually called conventional, direct, normal, or regular rather than equatorial.

Equidistant projection - A projection that maintains constant scale along all great circles from one or two points. When the projection is centered on a pole,

the parallels are spaced in proportion to their true distances along each meridian.

Equireal projection - *See* Equal-area projection.

Equivalent projection - *See* Equal-area projection.

Flat-polar projection - A projection on which, in normal aspect, the pole is shown as a line rather than as a point.

Frame - *See* Map frame.

Free of distortion - Having no distortion of shape, area, or linear scale. On a flat map, this condition can exist only at certain points or along certain lines.

General matrix map - In the Mapping Toolbox, a matrix map defined with latitude-longitude coordinate matrices, allowing irregular, non-rectangular orientations.

Geoid - The true shape of the Earth, an irregular and complex shape which is usually modeled with a sphere or an ellipsoid. In the Mapping Toolbox, this term refers to the spherical or ellipsoidal model of the Earth *or other planet* in use rather than the true shape.

Geoid vector - In the Mapping Toolbox, a vector describing the geoid, or ellipsoid, model. The geoid vector has the form:

```
geoidvec = [semimajor-axis eccentricity]
```

Geometric projection - *See* Perspective projection.

Geographic data structure - In the Mapping Toolbox, A MATLAB data structure containing the data and other information for the proper display of map objects. Valid fields in the structure include `type`, `tag`, and `altitude`.

GIS (Geographic Information System) - A system, usually computer based, for the input, storage, retrieval, analysis, and display of interpreted geographic data.

Globular projection - Generally, a nonazimuthal projection developed before 1700 on which a hemisphere is enclosed in a circle and meridians and parallels are simple curves or straight lines.

Graticule - A network of lines representing a selection of the Earth's parallels and meridians for the purpose of projection. The vertices of the graticule grid are precisely projected, and the map data contained in any grid cell is warped

to fit the resulting quadralateral. A finer graticule grid results in a higher projection fidelity at the expense of greater computational requirements.

Great circle - Any circle on the surface of a sphere, especially when the sphere represents the Earth, formed by the intersection of the surface with a plane passing through the center of the sphere. It is the shortest path between any two points along the circle and therefore important for navigation. All meridians and the Equator are great circles on the Earth taken as a sphere.

Grid - *See* Map grid.

HMS - Hours-minutes-seconds time notation of the form hh mm' ss". In the Mapping Toolbox, when "hms" times are represented by a single number, the format is hhmm.ss.

Homalographic/homolographic projection - *See* Equal-area projection.

Hydrography - The science of measurement, description, and mapping of the surface waters of the Earth, especially with reference to their use in navigation. The term also refers to those parts of a map collectively that represent surface waters.

Hydrology - The scientific study of the waters of the Earth, especially with relation to the effects of precipitation and evaporation upon the occurrence and character of ground water.

Hypsography - The scientific study of the Earth's topologic configuration above sea level, especially the measurement and mapping of land elevation.

Indexed map - A matrix map in which entries are an index value into another data source. The `worldmtx` workspace contains an example of an indexed map. Each entry in the matrix map is an index into a data structure containing the names of the world countries.

Indicatrix - A circle or ellipse having the same shape as that of an infinitesimally small circle (having differential dimensions) on the Earth when it is plotted with finite dimensions on a map projection. Its axes lie in the directions of and are proportional to the maximum and minimum scales at that point. This is useful in illustrating the distortions of a given map projection. Often called a Tissot indicatrix after the originator of the concept. In the Mapping Toolbox, Tissot indicatrices may be displayed using the `tissot` command, and indicatrices for all supported projections are provided in the "Projections Reference" chapter of the online Mapping Toolbox reference documentation.

Interrupted projection - A projection designed to reduce peripheral distortion by making use of separate sections joined at certain points or along certain lines, usually the Equator in the normal aspect, and split along lines that are usually meridians. There is normally a central meridian for each section. The Mapping Toolbox does not include interrupted projections, but the user can separate data into sections and project these independently to achieve this effect.

Large-scale mapping - Mapping at a scale larger than about 1:75,000, although this limit is somewhat flexible.

Latitude (geographic) - The angle made by a perpendicular to a given point on the surface of a sphere or ellipsoid representing the Earth and the plane of the Equator (positive if the point is north of Equator, negative if it is south). One of the two common geographic coordinates of a point on the Earth.

Latitude of opposite sign - *See* Parallel of opposite sign.

Legs - Line segments connecting waypoints.

Legend - *See* Map legend.

Limiting forms - The form taken by a system of projection when the parameters of the formulas defining that projection are allowed to reach limits that cause it to be identical with another separately defined projection.

Logical map - A binary matrix map consisting entirely of 1s and 0s. An example of a logical matrix map can be created with the `topo` map by performing a logical test for positive elevations (`topo>0`). Each entry in the matrix map contains a 1 if it is above sea level, or a 0 if it is at or below sea level.

Longitude - The angle made by the plane of a meridian passing through a given point on the Earth's surface and the plane of the (prime) meridian passing through Greenwich, England, east or west to 180 (positive if the point is east, negative if it is west). One of the two common geographic coordinates of a point on the Earth.

Loxodrome - *See* Rhumb line.

Map A representation of geographic data. In the Mapping Toolbox, a map is any variable or set of variables (electronically) representing or assigning values to a geographic location or region, from a single point to an entire planet.

Map axes - In the Mapping Toolbox, a normal MATLAB axes altered for mapping display purposes. Several map axes properties are defined and stored in the `UserData` slot of the MATLAB axes. These properties control the appearance of the map display, much like the properties of the normal MATLAB axes control the appearance of the displayed plot. A map axes must first be defined in order to display maps using the Mapping Toolbox.

Map frame - In the Mapping Toolbox, a projected "box" or quadrangle enclosing the geographic display.

Map grid - A displayed network of lines representing parallels and meridians. The grid is used for visual reference and should not be confused with the graticule.

Map legend - In the Mapping Toolbox, a vector defining the geographic placement and unit cell size of a regular matrix map. A map legend has the form:

```
maplegend = [cells/angleunit north-latitude west-longitude]
```

MapInfo - A largely PC-oriented GIS developed and distributed by the MapInfo Corporation.

Matrix map - A map consisting of a matrix (or matrices) of values corresponding to specific geographic points. In the Mapping Toolbox, matrix maps can be defined as regular or general, depending on the structure and orientation of the geographic points. *See* Regular matrix map and General matrix map.

Meridian - A reference line on the Earth's surface formed by the intersection of the surface with a plane passing through both poles and some third point on the surface. This line is identified by its longitude. On the Earth as a sphere, this line is half a great circle; on the Earth as an ellipsoid, it is half an ellipse.

Minimum-error projection - A projection having the least possible total error of any projection in the designated classification, according to a given mathematical criterion. Usually, this criterion calls for the minimum sum of squares of deviations of linear scale from true scale throughout the map ("least squares").

NGVD 29 (National Geodetic Vertical Datum of 1929) - A reference surface established by the U.S. Coast Guard and Geodetic Survey of 1929, used as the datum for which relief features and elevation data are referenced in the conterminous United States; formerly called "mean sea level 1929."

Nominal scale - The stated scale at which a map projection is constructed.

Normal aspect - A form of a projection that provides the simplest graticule and calculations. It is the polar aspect for azimuthal projections, the aspect having a straight Equator for cylindrical and pseudocylindrical projections, and the aspect showing straight meridians for conic projections. Also called conventional, direct, or regular aspect.

Oblique aspect - An aspect of a projection on which the axis of the Earth is rotated so it is neither aligned with nor perpendicular to the conceptual axis of the map projection.

Orthoapsidal projection - A projection on which the surface of the Earth taken as a sphere is transformed onto a solid other than the sphere and then projected orthographically and obliquely onto a plane for the map.

Orthographic projection - A specific azimuthal projection or a type of projection in which the Earth is projected geometrically onto a surface by means of parallel projection lines.

Orthomorphic projection - *See* Conformal projection.

Parallel - A small circle on the surface of the Earth formed by the intersection of the surface of the reference sphere or ellipsoid with a plane parallel to the plane of the Equator. This line is identified by its latitude. The Equator (a great circle) is usually also treated as a parallel.

Parallel of opposite sign - A parallel that is equally distant from but on the opposite side of the Equator. For example, for lat 30(N(or +30),the parallel of opposite sign is lat 30 S (or -30). Also called latitude of opposite sign.

Parameters - The values of constants as applied to a map projection for a specific map; examples are the values of the scale, the latitudes of the standard parallels, and the central meridian. The required parameters vary with the projection.

Perspective projection - A projection produced by projecting straight lines radiating from a selected point (or from infinity) through points on the surface of a sphere or ellipsoid and then onto a tangent or secant plane. Other perspective maps are projected onto a tangent or secant cylinder or cone by using straight lines passing through a single axis of the sphere or ellipsoid. Also called geometric projection.

Planar projection - A projection resulting from the conceptual projection of the Earth onto a tangent or secant plane. Usually, a planar projection is the same as an azimuthal projection. Mathematically, the projection is often only partially geometric.

Planimetric map - A map representing only the horizontal positions of features (without their elevations).

Polar aspect - An aspect of a projection, especially an azimuthal one, on which the Earth is viewed from the polar axis. For cylindrical or pseudocylindrical projections, this aspect is called transverse.

Pole - An extremity of a planet's axis of rotation. The North Pole is a singular point at 90 N for which longitude is ambiguous. The South Pole has the same characteristics and is located at 90 S.

Polyconic projection - A specific projection or member of a class of projections that are constructed like conic projections but with different cones for each parallel. In the normal aspect, all the parallels of latitude are nonconcentric circular arcs, except for a straight Equator, and the centers of these circles lie along a central axis.

Projection - A systematic representation of a curved 3-D surface such as the Earth onto a flat 2-D plane. Each map projection has specific properties that make it useful for specific purposes.

Pseudoconic projection - A projection that, in the normal aspect, has concentric circular arcs for parallels and on which the meridians are equally spaced along the parallels, like those on a conic projection, but on which meridians are curved.

Pseudocylindrical projection - A projection that, in the normal aspect, has straight parallel lines for parallels and on which the meridians are (usually) equally spaced along parallels, as they are on a cylindrical projection, but on which the meridians are curved.

Quadrangle - A region bounded by parallels north and south, and meridians east and west.

Raster map - *See* Matrix map.

Reckoning - The determination of geographic position by calculation.

Regional map - A small-scale map of an area covering at least 5 or 10 degrees of latitude and longitude but less than a hemisphere.

Regular aspect - *See* Normal aspect.

Regular matrix map - In the Mapping Toolbox, an equiangular (equal-angle) matrix map defined with a map legend vector, limited to a rectangular orientation.

Retroazimuthal projection - A projection on which the direction or azimuth from every point on the map to a given central point is shown correctly with respect to a vertical line parallel to the central meridian. The reverse of an azimuthal projection.

Rhumb line - A complex curve (a spherical helix) on a planet's surface that crosses every meridian at the same oblique angle; a navigator can proceed between any two points along a rhumb line by maintaining a constant heading. A rhumb line is a straight line on the Mercator projection. Also called a loxodrome.

Scale - The ratio of the distance on a map or globe to the corresponding distance on the Earth; usually stated in the form 1:5,000,000 for example.

Scale factor - The ratio of the scale at a particular location and direction on a map to the stated scale of the map. At a standard parallel, or other standard line, the scale factor is 1.0.

Secant cone, cylinder, or plane - A secant cone or cylinder intersects the sphere or ellipsoid along two separate lines; these lines are parallels of latitude if the axes of the geometric figures coincide. A secant plane intersects the sphere or ellipsoid along a line that is a parallel of latitude if the plane is at right angles to the axis.

Shaded Relief - Shading added to a map or image that makes it appear to have three-dimensional aspects. This type of enhancement is commonly done to satellite images and thematic maps utilizing digital topographic data to provide the appearance of terrain relief.

Similar (projection) - Subjective and qualitative term indicating a moderate or strong resemblance.

Singular points - Certain points on most but not all conformal projections at which conformality fails, such as the poles on the normal aspect of the Mercator projection.

Skew-oblique aspect - An aspect of a projection on which the axis of the Earth is rotated, so it is neither aligned with nor perpendicular to the conceptual axis

of the map projection, and tilted, so the poles are at an angle to the conceptual axis of the map projection.

Small circle - A circle on the surface of a sphere formed by the intersection with a plane. Parallels of latitude are small circles on the Earth taken as a sphere. In the Mapping Toolbox, great circles, including the Equator and all meridians, are treated as special, limiting cases of small circles.

Small-scale mapping - Mapping at a scale smaller than about 1:1,000,000, although the limiting scale sometimes has been made as large as 1:250,000.

Spheroid - *See* Ellipsoid.

Standard parallel - In the normal aspect of a projection, a parallel of latitude along which the scale is as stated for that map. There are one or two standard parallels on most cylindrical and conic map projections and one on many polar stereographic projections.

Stereographic projection - A specific azimuthal projection or type of projection in which the Earth is projected geometrically onto a surface from a fixed (or moving) point on the opposite face of the Earth.

Tangent cone or cylinder - A cone or cylinder that just touches the sphere or ellipsoid along a single line. This line is a parallel of latitude if the axes of the geometric figures coincide.

Thematic map - A map designed to portray primarily a particular subject, such as population, railroads, or croplands.

Tissot indicatrix - *See* Indicatrix.

Topographic map - A map that usually represents the vertical positions or elevations of features as well as their horizontal positions. The `topo` workspace contains a simple example.

Transformed latitudes, longitudes, or poles - Graticule of meridians and parallels on a projection after the Earth has been turned with respect to the projection so that the Earth's axis no longer coincides with the conceptual axis of the projection. Used for oblique and transverse aspects of many projections.

Transverse aspect - An aspect of a map projection on which the axis of the Earth is rotated so that it is at right angles to the conceptual axis of the map projection. For azimuthal projections, this aspect is usually called equatorial rather than transverse.

True scale - *See* Correct scale.

Valued map - A matrix map in which entries represent some value or measurement. The `topo` workspace contains an example of a valued map. Each entry in the matrix map is an average elevation in meters for the geographic position represented by that cell.

Vector map - A map consisting of ordered latitude-longitude points, possibly connected. In the Mapping Toolbox, such map data is often represented by two vectors, representing latitude and longitude. Segments can be separated by the insertion of `NaN`'s in both vectors.

Waypoint - Points through which a track passes, usually corresponding to course or speed changes.

WGS 72 (World Geodetic System 1972 - An Earth-centered datum, used as a definition of DMA DEMs, presently stored in the USGS data base. The WGS 72 datum was the result of an extensive effort extending over approximately three years to collect selected satellite, surface gravity, and astrogeodetic data available throughout 1972. These data were combined using a unified WGS solution (a large-scale least squares adjustment).

WGS 84 (World Geodetic System 1984) - The WGS 84 was developed as a replacement for the WGS 72 by the military mapping community as a result of new and more accurate instrumentation and a more comprehensive control network of ground stations. The newly developed satellite radar altimeter was used to deduce geoid heights from oceanic regions between 70 north and south latitude. Geoid heights were also deduced from ground-based Doppler and ground-based laser satellite-tracking data, as well as surface gravity data. The ellipsoid associated with WGS 84 is GRS 80.

Zenithal projection - *See* Azimuthal projection.

B

Bibliography

1 Snyder, J. P., *Map Projections - A Working Manual*, U.S. Geological Survey Professional Paper 1395, Washington, D.C., 1987.

2 Maling, D. H., *Coordinate Systems and Map Projections*, 2nd Edition, Pergamon Press, New York, NY, 1992.

3 Snyder, J. P. and Voxland, P. M., *An Album of Map Projections*, U.S. Geological Survey Professional Paper 1453, Washington, D.C., 1994.

4 Snynder, J. P., *Flattening the Earth - 2000 Years of Map Projections*, University of Chicago Press, Chicago, IL, 1993.

Index

A

accuracy of map computations 9-146
Adams, O. S. 10-29
Adams, Oscar Sherman 10-105
Airy Minimum Error Azimuthal projection 10-19
Airy, George 10-19
aitoff 10-3
Aitoff projection 10-3, 10-65
Aitoff, David 10-3
Albers Equal-Area Conic projection 10-5
Albers, Heinrich Christian 10-5
almanac 2-20
almanac 9-25
almanac data 2-20
American Geographical Society 10-89
American Polyconic projection 10-101
angle strings
  converting to numbers 2-23
angle units
  convention for navigation functions 6-11
  converting between formats 2-23
  description of formats 2-22
angledim 2-23
angledim 9-31
angles
  converting units 9-31, 9-107, 9-110, 9-127, 9-129, 9-323, 9-324
  normalizing range 9-285, 9-472
antipode 2-8
antipode 9-33
Apian, Peter 10-7
apianus 10-7
Apianus II projection 10-7
areaint 2-36
areaint 9-34
areamat 2-56
areamat 9-36
areaquad 2-16
areaquad 9-39
ASCII file
  converting delimiters to NaNs 9-272
  reading 9-391
aspect
  defined 2-58
aspect *See* projection aspect
astronomical data 4-40
aut2geod 9-41
auxiliary sphere
  aut2geod 9-41
  calculating radius 9-356
  cen2geod 9-70
  cnf2geod 9-78
  geod2aut 9-170
  geod2cen 9-171
  geod2cnf 9-172
  geod2iso 9-173
  geod2par 9-174
  geod2rec 9-175
  iso2geod 9-217
  par2geod 9-291
  rec2geod 9-339
avhrrgoode 12-5
avhrrlambert 12-9
axes *See* map axes
axes, Cartesian *See* Cartesian axes
axes, map *See* map axes
axes2ecc 2-18
axes2ecc 9-42
axesm 3-5, 3-13, 3-17
axesm 9-43, 11-7
axesmui 11-7
axesscale 3-58
axesscale 9-56

azimuth
between track waypoints 9-219
calculating 9-58, 11-90
defined 2-11
finding cross fix position 9-98
in projected coordinates 3-67
measuring 2-11
azimuth 2-11
azimuth 9-58
azimuthal projection 5-5

B

Babinet projection 10-91
Balthasart Cylindrical projection 10-9, 10-43
balthsrt 10-9
Bartholomew, John 10-65
base map
adding to 1-10
creating 1-3–1-9
base projection 5-14
bearing *See* azimuth
behrmann 10-11
Behrmann Cylindrical projection 10-5, 10-11, 10-43
Behrmann, Walter 10-11
Bienewitz 10-7
Bolshoi Sovietskii Atlas Mira 10-13
Bolshoi Sovietskii Atlas Mira projection 10-13, 10-17
bonne 10-15
Bonne projection 10-15
Bonne, Rigobert 10-15
Bordone Oval projection 10-79
Braun 10-17
braun 10-17
Braun Perspective Cylindrical projection 10-13, 10-17, 10-57
breusing 10-19
Breusing Harmonic Mean projection 10-19
Breusing, F. A. Arthur 10-19
bries 10-21
Briesemeister projection 10-21, 10-65
Briesemeister, William 10-21
bsam 10-13
BSAM projection 10-13
Buffer 2-42
buffer zone
defined 2-42
bufferm 2-42

C

camposm 9-63
camtargm 9-65
camupm 9-67
cape workspace 4-37
cart2grn 9-69
Cartesian axes, displaying 9-385
cassini 10-23
Cassini Cylindrical projection 10-23
Cassini de Thury, César François 10-23
Cassini projection 10-99
ccylin 10-25
cen2geod 9-70
Central Cylindrical projection 10-25, 10-129
Central projection 10-61
central projection 5-6
Ch'ien Lo-Chih 10-87
changem 2-53
changem 9-71
choropleth maps, example 3-60

circles
See great circles
See small circles
clabelm 9-72
clegendm 9-74
Click-and-Drag property editor 11-64
clma 9-76
clmo 9-77, 11-19
clrmenu 11-20
cmapui 3-74, 7-38
cnf2geod 9-78
coast 9-79
coast workspace 2-3, 4-3
collig 10-27
Collignon projection 10-27
Collignon, Édouard 10-27
colorbar 3-70
labeled 3-73
colorm 7-34
colorm 11-21
colormap
annotating 3-73
creating for digital elevation map 3-68
creating for political data 3-71
creating for surface contour maps 3-70
colormap menu in figure window 11-20
colormaps
creating interactively 3-74
creating with GUI 7-34
digital elevation map 9-111, 11-27
menu in maptool 7-5
regular matrix map 11-21
shaded relief map 9-383
combinations 9-80
combntns 9-80
comet3m 9-81, 11-23
cometm 3-62
cometm 9-82, 11-23
conformal projection 5-3
Conic projection 10-47
conic projection 5-4
Conical Orthomorphic projection 10-75
contor3m 9-83, 11-25
contorm 1-12
contorm 9-85, 11-25
contour map
adding legend 9-74
creating 9-83, 9-85, 11-25
labeling 9-72
contourcmap 3-70
contourfm 9-90
conversion
angle units 9-31, 9-107, 9-110, 9-127, 9-129, 9-323, 9-324
ASCII file delimiters 9-272
coordinate types 9-69, 9-147, 9-191
distance to string 9-118
distance units 9-108, 9-125, 9-218, 9-284, 9-325, 9-388
dms to matrix elements 9-128
ellipsoid parameters 9-42, 9-136, 9-137, 9-155, 9-270
geographic coordinates to matrix coordinates 9-382
great circles to small circles 9-159
hms to matrix elements 9-203
latitude types 9-41, 9-70, 9-78, 9-170, 9-171, 9-172, 9-173, 9-174, 9-175, 9-217, 9-291, 9-339
matrix coordinates to geographic coordinates 9-378
matrix elements to dms 9-249
matrix elements to hms 9-250
time to string 9-414

conversion *(continued)*
time units 9-201, 9-202, 9-204, 9-205, 9-376, 9-377, 9-416
coordinate system transforming 9-351
coordinate transformations 5-15
example using stars 8-76
matrix data 5-17
vector data 5-16
coordinates
converting from geographic to matrix 9-382
converting from matrix to geographic 9-378
converting types 9-69, 9-147, 9-191
selecting with mouse 9-209
Cossin, Jean 10-109
`country2mtx` 4-22, 4-23
`country2mtx` 9-95
`craster` 10-29
Craster Parabolic projection 10-29
Craster, John Evelyn Edmund 10-29
cross fix positions 9-98
`crossfix` 9-98
current point from map axes 9-163
cylindrical projection 5-4

D

`daspectm` 9-101
`dcwdata` 12-12
`dcwdem` 12-5
DCW-DEM data 12-5
`dcwgaz` 12-16
`dcwread` 12-18
`dcwrhead` 12-21
de l'Isle, Nicolas 10-47
dead reckoning 6-28, 9-130
calculating positions 6-30
rules of 6-30
Deetz, Charles H. 10-29
`defaultm` 9-103
`deg2dm` 9-107
`deg2dms` 9-107
`deg2km` 2-11
`deg2km` 9-108
`deg2nm` 2-26
`deg2nm` 9-108
`deg2rad` 2-8
`deg2rad` 9-110
`deg2sm` 9-108
degrees notation 2-22
`delaunay` 3-65
`demcmap` 3-68
`demcmap` 9-111, 11-27
`demdataui` 8-51
`depart` 9-113
departure 6-5, 9-113
Die Rechteckige Plattkarte 10-49
Digital Chart of the World 8-3
Digital Chart of the World (DCW)
reading digital elevation data 12-5
reading files 12-18
reading gazette 12-16
reading headers 12-21
reading selected data 12-12
digital elevation map, colormap 9-111
digital elevation maps
colormap for 3-68
line of sight in 2-59
reading data interactively 8-51
texture mapping color data onto 3-44
digital elevation maps (DEMs) 3-68
digital elevation maps (DEMs),defined 2-4

displaying
light objects 9-221, 11-34
lighted surfaces 9-403, 11-94
lines 9-224, 9-299, 9-301, 11-38
patches 9-150, 9-152, 9-292, 9-294, 11-29
surfaces 9-261, 9-296, 9-401, 9-408, 11-49, 11-62
text 9-192, 9-411, 11-98
displaym 3-51
displaym 9-115
dist2str 9-118
distance
converting to string 9-118
converting units 9-108, 9-125, 9-218, 9-284, 9-325, 9-388
See also surface distance
distance 2-10
distance 9-120
distance *See* surface distance
distance units
convention for navigation functions 6-11
converting between formats 2-26
description of formats 2-26
distdim 2-26
distdim 9-125
distortcalc 9-123
dms notation 2-22
dms2deg 2-23
dms2deg 9-127
dms2dm 9-129
dms2mat 9-128
dms2rad 2-23
dms2rad 9-127
dreckon 6-11, 6-30
dreckon 9-130
drift correction 6-33
driftcorr 6-34
driftvel 6-35
dted 12-27
dteds 8-49

E

Earth
default geoid 2-19
ellipsoid models 2-19
Earth *See* almanac 9-25
eastof 9-135
ecc2flat 9-136
ecc2n 9-137
eccentricity 9-42
Eckert I projection 10-31
Eckert II projection 10-33
Eckert III projection 10-35
Eckert IV projection 10-37
Eckert V projection 10-39, 10-133
Eckert VI projection 10-41
Eckert, Max 10-31, 10-33, 10-35, 10-37, 10-39, 10-41
eckert1 10-31
eckert2 10-33
eckert3 10-35
eckert4 10-37
eckert5 10-39
eckert6 10-41
Edwards, Trystan 10-115
egm96geoid 12-32
Egyptians 10-45, 10-97, 10-111
elevation
defined 2-12
measuring 2-11
elevation map *See* digital elevation map
ellipse1 9-140

ellipsoid
approximating planetary geoid *See* almanac
as a geoid model 2-17
converting parameters 2-18
models for Earth 2-19
models for planets 2-20
radius of curvature 9-329
ellipsoid parameters
conversion 9-136, 9-137, 9-155, 9-270
converting 9-42
Elliptical projection 10-91
encodem 9-144
epsm 9-146
eqa2grn 6-10
eqa2grn 9-147
eqaazim 10-73
eqaconic 10-5
eqacylin 10-43
eqdazim 10-45
eqdconic 10-47
eqdcylin 10-49
Equal-Area Cylindrical projection 10-9, 10-11, 10-43, 10-55, 10-77, 10-115
equal-area projection 5-3
Equidistant Azimuthal projection 10-3, 10-45, 10-46, 10-47
Equidistant Conic projection 10-47
Equidistant Cylindrical projection 10-47, 10-49, 10-50, 10-53, 10-99, 10-133
equidistant projection 5-3
Equirectangular projection 10-49, 10-50
Erastosthenes 10-99
etopo5 8-34
etopo5 12-34
ETOPO5 model 8-34, 12-34
Etzlaub, Erhard 10-87
Everett 10-95
external data
DCW data 12-12, 12-18
DCW gazette 12-16
DCW headers 12-21
DCW-DEM data 12-5
ETOPO5 model 12-34
Fifth Fundamental Catalog of Stars 12-47
TIGER ArcInfo files 12-60
TIGER FIPS name files 12-36
TIGER MIF files 12-56
TIGER/Line data 12-54
USGS DEM data 12-63, 12-67
USGS DEM filenames 12-69
external data sources 8-2
extractm 3-52
extractm 9-148

F

Federal Information Processing Standard (FIPS) 8-29, 8-32
Federal Information Processing System (FIPS) 8-25
Fifth Fundamental Catalog of Stars 8-74, 12-47
fill3m 9-150, 11-29
fillm 3-32, 3-33
fillm 9-152, 11-29
filtering geographic data 6-10
filterm 6-10
filterm 9-153
findm 2-53
findm 9-154
fipsname 8-30, 8-32
fipsname 12-36
fixing *See* navigational fixing
flat2ecc 9-155
flatearthpoly 2-41

flatplrp 10-81
flatplrq 10-83
flatplrs 10-85
Flat-Polar Quartic projection 10-83
fournier 10-51
Fournier II projection 10-51
Fournier projection 10-51, 10-52
Fournier, Georges 10-51
frame *See* map frame
framem 3-13, 3-16
framem 9-158

G

Gall Isographic projection 10-49, 10-53
Gall Orthographic projection 10-43, 10-55
Gall projection 10-57
Gall Stereographic projection 10-17, 10-57
Gall, James 10-55, 10-57
Gauss-Krüger 10-119
gazette 4-14
gc2sc 9-159
gcm 9-161
gcpmap 9-163
gcwaypts 6-11, 6-25
gcwaypts 9-164
gcxgc 2-38
gcxgc 9-166
gcxsc 2-38
gcxsc 9-168
general matrix map
  projecting 9-296, 9-401, 9-403, 9-408, 11-62, 11-94
  projecting shaded relief 9-405
general matrix maps
  converting to regular matrix maps 2-66
  defined 2-61
  displaying 3-34
  displaying image and surface coloring 3-37
  displaying light shading 3-39
  displaying shaded relief 3-40
  See also matrix maps
geod2aut 9-170
geod2cen 9-171
geod2cnf 9-172
geod2iso 9-173
geod2par 9-174
geod2rec 9-175
geographic data structure
  creating input mlayers 9-350
  defined 3-49
  definition 9-176
  displaying 3-51, 9-115
  extracting data 9-148
  extracting data from 4-11, 4-29
  interacting with objects 11-51
geographic mean 6-3
geographic points
  mean 9-255
  standard deviation 9-395
  standard distance 9-393
geographic standard deviation 6-4
geoid
  availability for planets 2-20
  converting ellipsoid parameters 2-18
  defined 2-17
  ellipsoid approximation 2-17
  ellipsoid models for Earth 2-19
geoid matrix 4-25

geoid vector 2-17
components 9-179
planets *See* almanac
geostatistics
calculating geographic mean 6-3
calculating geographic standard deviation 6-4
equal-area coordinate system 6-9
equirectangular binning 6-7
filtering data sets 6-10
histograms 6-7
getm 3-8, 3-59
getm 9-180
getseeds 9-181
giso 10-53
Globe 10-59, 10-60
globe 10-59
GLOBE data 8-42
globe projection 5-23
globedem 8-46
globedems 8-47
Gnomic projection 10-61
gnomonic 10-61
Gnomonic projection 10-61
gnomonic projection 5-6
goode 10-63
Goode Homolosine projection 10-63, 10-91
Goode, J. Paul 10-63
gortho 10-55
gradient
defined 2-58
gradientm 2-58
graphic scale 3-59
graphical user interfaces
activating 7-2
getting help 7-2
See also maptool
graticule
choosing resolution 3-35, 3-36
defined 2-64, 3-35
graticule mesh 9-257
great circle track
calculating from one point 9-425
calculating from two points 9-428
displaying 11-100
great circles
approximating tracks with rhumb lines 6-25
calculating points of 2-32
converting to small circles 9-159
defined 2-9
interactive 3-53
intersection 9-166
intersection with small circles 9-168
Great Soviet World Atlas 10-13
Greeks 10-97, 10-111
grepfields 9-187
grid *See* map grid
gridm 9-190
grn2eqa 6-9
grn2eqa 9-191
gshhs 12-41
gstereo 10-57
gtextm 9-192
gtopo30 12-43
GTOPO30 data 8-42
gtopo30s 8-46
Guide property editor 11-64

H

hammer 10-65
Hammer projection 10-3, 10-21, 10-65
Hammer-Aitoff projection 10-65
handlem 3-57

handlem 9-193, 11-31
Hassler, Ferdinand Rudolph 10-101
hatano 10-67
Hatano Asymmetrical Equal-Area projection 10-67
Hatano, Masataka 10-67
help
  getting more 1-18
hidem 3-57
hidem 9-196, 11-33
hista 9-197
histogram
  equal area 9-197
  equirectangular 9-199
histograms 6-7
histr 6-7
histr 9-199
hms notation 2-25
hms2hm 9-201
hms2hr 9-202
hms2mat 9-203
hms2sec 9-202
Homolographic projection 10-91
Homolosine projection 10-63
Hondius, Jodocus 10-109
hours notation 2-25
hr2hm 9-204
hr2hms 9-204
hr2sec 9-205

I

imagem 9-206
imbedm 9-208
indexed maps
  defined 2-55
  modifying colormap 7-34
  replacing values 7-27
inputm 3-53, 6-26
inputm 9-209
Inset maps
  controlling scale 3-58
  creating 3-58
interplat 2-35
interplon 2-35
interpm 2-34
interpm 9-210
interpolation
  along a path 2-57
  latitude and longitude 9-210
  latitude given longitude 9-211
  longitude given latitude 9-213
interpolation, latitude and longitude 2-33
intersection
  great circles 9-166
  great circles and small circles 9-168
  object sets 9-98
  rhumb lines 9-348
  small circles 9-374
intrplat 9-211
intrplon 9-213
ismap 9-215
ismapped 9-216
iso2geod 9-217

J

Jupiter *See* almanac 9-25

K

Kavraisky V projection 10-69
Kavraisky VI projection 10-71
Kavraisky, V. V. 10-69, 10-71
kavrsky5 10-69

kavrsky6 10-71
km2deg 9-218
km2nm 9-218
km2rad 9-218
km2sm 9-218
korea workspace 3-37

L

La Carte Parallélogrammatique 10-49
labels
  meridians and parallels 3-19
  text 3-25
lambcyln 10-77
lambert 10-75
Lambert Azimuthal Equal Area projection 10-65
Lambert Azimuthal Equal-Area projection 10-5, 10-73
Lambert Conformal Conic projection 10-75
Lambert Equal-Area Azimuthal projection 10-19
Lambert Equal-Area Conic projection 10-5
Lambert Equal-Area Cylindrical projection 10-5, 10-43, 10-77
Lambert, Johann Heinrich 10-43, 10-73, 10-75, 10-77
latitude and longitude
  converting latitude types 9-41, 9-70, 9-78, 9-170, 9-171, 9-172, 9-173, 9-174, 9-175, 9-217, 9-291, 9-339
  finding corresponding time zone 9-417
  finding for map entries 9-154
  interpolation 2-33, 9-210, 9-211, 9-213
  limits of regular matrix map 9-223, 11-36
  See also map frame, setting limits
  See also map limits
latitude, defined 2-7
lcolorbar 3-73
legs
  course and distance of 6-27
  defined 6-12
legs 6-11, 6-27
legs 9-219
light object 11-34
light objects 9-221
light objects, displaying 3-41
lightm 3-41
lightm 9-221, 11-34
limitm 2-50
limitm 9-223, 11-36
line objects 9-224, 9-299, 9-301, 11-38
  displaying 3-28
linem 9-224, 11-38
loadcape 4-38
loadmoonalb 12-46
logical maps
  calculating surface area 2-56
  defined 2-56
longitude
  converting between ranges 2-8
longitude, defined 2-7
Lorgna projection 10-73
los2 2-59
loximuth 10-79
Loximuthal projection 10-79
loxodromes *See* rhumb lines
ltln2val 2-53
ltln2val 9-230

M

majaxis 9-231
makemapped 3-66
makemapped 9-232

map
deleting 9-76
precision 9-146
map axes
accessing properties 3-8
accessing properties default values 3-10
defined 3-5
defining a map projection 3-5
defining map projection 9-43, 11-7
differences from standard axes 3-6
inset maps 3-58
modifying properties 9-379
resetting to default properties 3-24
retrieving map structure 9-161
retrieving properties 9-180
setting properties 3-5, 3-8, 9-43, 11-7
testing 9-215
map data
extracting from data structures 9-148
filtering 9-153
querying 11-68
See also matrix data
See also vector data
map definition 2-2
map frame
adjusting for a new projection 3-22
controlling appearance 3-15
defined 3-13
displaying 3-13, 9-158
modifying properties 9-379
resetting altitude 3-15
setting limits 3-13, 3-14, 3-21
setting properties 9-43, 9-158, 11-7
trimming objects 4-7
trimming objects to 3-66
map grid
controlling appearance 3-17
defined 3-17
displaying 3-17, 9-190
modifying properties 9-379
resetting altitude 3-17
setting properties 9-43, 9-190, 11-7
map grid labels
displaying 9-269, 9-298
modifying properties 9-379
setting properties 9-43, 11-7
map layers 11-51
map legend
defined 2-49
map legend vector 9-234
map limits
adjusting for a new projection 3-22
setting 3-15, 3-21
versus frame limits 3-21
map origin *See* origin
map projection
aspect 5-7
azimuthal 5-5
base 5-14
central 5-6
changing 9-379
choosing 5-31
computations 5-19
conformal 5-3
conic 5-4
cylindrical 5-4
defined 3-5, 5-2
defining 9-43, 11-7
defining for a map axes 3-5
distortion 5-3, 5-14

map projection *(continued)*
equal-area 5-3
equidistant 5-3
globe 5-23
gnomonic 5-6
identification strings 9-241
names 9-241
orthographic 5-6
polyconic 5-4
stereographic 5-6
switching 3-26
table of properties 5-31
two-dimensional vs. three-dimensional 5-23
vectors 5-21
map scale
between axes 3-58
printing 3-75
`mapmtx` workspace 2-62
mapped object
versus standard object 3-25
mapped objects
converting from standard objects 3-66
manipulating by name 3-55
trimming to map frame 3-66
`mapprofile` 2-57
maps
previewing 3-76
printing 3-75
`maps` 3-5
`maps` 9-241
`maptool` 7-2
activating 7-3
menus 7-4
mouse tool buttons 7-5
`maptool` 11-40
`maptrim` 7-25
`maptrim` 11-46
`maptriml` 2-43
`maptriml` 9-243
`maptrimp` 2-43
`maptrimp` 9-244
`maptrims` 9-246
Marinus of Tyre 10-49, 10-99
Mars *See* `almanac` 9-25
`maskm` 2-54
`maskm` 9-248
`mat2dms` 2-22
`mat2dms` 9-249
`mat2hms` 2-25
`mat2hms` 9-250
MATLAB graphics
projecting 3-65
matrix data
displaying 9-261, 9-296, 9-401, 9-408, 11-49, 11-62
displaying as lighted 9-403, 11-94
displaying as shaded relief 9-259, 9-405
resizing 9-344
trimming 9-246, 11-46
matrix map
constructing graticule mesh 9-257
encoding regions 9-144
`NaN`s 9-273
ones 9-286
replacing elements 9-71, 9-248
resizing 9-344
sparse zeros 9-392
zeros 9-473
matrix maps
coloring 3-68, 7-34
creating base map from 1-6
defined 2-4
displaying 3-34
gradient 2-58

matrix maps *(continued)*
graticules 3-35
indexed maps 2-55, 7-27, 7-34
indexed maps,creating 4-23
logical maps 2-56
replacing entries 2-53, 7-27
See also general matrix maps
See also regular matrix maps
valued maps 2-55
McBryde, F. Webster 10-81, 10-83, 10-85
McBryde-Thomas Flat-Polar Parabolic projection 10-81
McBryde-Thomas Flat-Polar Quartic projection 10-83
McBryde-Thomas Flat-Polar Sinusoidal projection 10-85
`mdistort` 5-14
`mdistort` 9-251
mean of geographic data 6-3
mean of geographic points 9-255
`meanm` 6-4
`meanm` 9-255
`mercator` 10-87
Mercator Equal-Area projection 10-109
Mercator projection 5-13, 6-12, 6-26, 10-25, 10-75, 10-87, 10-89
Mercator, Gerardus 10-47, 10-87
Mercury *See* `almanac` 9-25
meridian labels 9-269
meridians
controlling display 3-17
defined 2-7
mesh *See* graticule mesh
`meshgrat` 2-66, 3-38
`meshgrat` 9-257
`meshlsrm` 3-40, 3-43
`meshlsrm` 9-259
`meshm` 7-36
`meshm` 9-261, 11-49
`mfwdtran` 9-263
`miller` 10-89
Miller Cylindrical projection 10-89
Miller, Osborn Maitland 10-89
`minaxis` 2-18
`minaxis` 9-265
`minvtran` 9-266
`mlabel` 9-269
`mlayers` 3-55
`mlayers` 11-51
`mobjects` 11-55
`modsine` 10-113
`mollweid` 10-91
Mollweide projection 10-63, 10-91
Mollweide, Carl B. 10-91
moon
albedo data 4-40
terrain data 4-40
Moon *See* `almanac` 9-25
mouse
interaction with displayed maps 3-53
mouse interactions
defining small circles 9-372
placing text 9-192
processing button down callbacks 11-104
selecting coordinates 9-209
Murdoch I Conic projection 10-93
Murdoch III Minimum Error Conic projection 10-95
Murdoch, Patrick 10-93, 10-95
`murdoch1` 10-93
`murdoch3` 10-95

N

n2ecc 9-270
name 9-271
namem 3-56
nanclip 9-272
nanm 2-56
nanm 9-273
NaNs, matrix map 9-273
National Geographic Society 10-107
navfix 6-11, 6-17
navfix 9-274
navigation
  angular conventions 6-11
  calculating course and distance 6-27
  calculating dead reckoning positions 6-30
  calculating waypoints 6-25
  connecting waypoints 6-26
  distance conventions 6-11
  fixing position 6-12, 6-17
  retrieving time zone for longitude 6-35, 6-36
  speed conventions 6-11
navigational fixing 6-12, 6-17, 9-274
navigational track
  calculating segments between waypoints 9-422
navigational track format 9-278
navigational tracks
  connecting waypoints 6-26
  displaying 6-26
  format 6-11
Neptune *See* almanac 9-25
neworig 5-17
neworig 9-280
newpole 5-16, 5-17
newpole 9-283
nm2deg 9-284
nm2km 9-284
nm2rad 9-284
nm2sm 9-284
Nordic projection 10-65
normal aspect 5-7
npi2pi 2-8
npi2pi 9-285

O

objects
  assigning tag 9-410, 11-96
  deleting 9-77, 11-19
  displaying 9-386, 11-87
  editing properties of 11-64
  hiding 9-196, 11-33
  interacting 11-55
  modifying zdata 9-471, 11-105
  retrieving handle 9-193, 11-31
  retrieving name 9-271
  testing if mapped 9-216
oblique aspect 5-8
ocean mask 4-9
oceanlo workspace 4-9
onem 2-56
onem 9-286
ones matrix map 9-286
Ordinary Polyconic projection 10-101
org2pol 9-287
orientation *See* origin vector
origin
  computing 9-283, 9-317
  interactive modification 11-58
  transformation 9-280
origin vector 5-7
  See also projection aspect
originui 11-58
ortho 10-97

Orthographic projection 10-59, 10-97, 10-123
orthographic projection 5-6
Orthophanic projection 10-107

P

panzoom 3-75
panzoom 11-59
paperscale 3-75
paperscale 9-288
par2geod 9-291
parallel labels 9-298
parallels
  controlling display 3-17
  defined 2-7
parallelui 11-61
patch drawing functions
  differences between 3-33
patch objects 9-150, 9-152, 9-292, 9-294
  displaying 3-31
patches
  projecting 11-29
patchesm 3-33
patchesm 9-292, 11-29
patchm 3-33
patchm 9-294, 11-29
pcarree 10-99
pcolorm 9-296, 11-62
Peters projection 10-55
piloting *See* navigational fixing
plabel 9-298
planetary data 2-20, 9-25
Plate Carree projection 10-99
Plate Carrée projection 10-23, 10-39, 10-47, 10-49
plot3m 3-32
plot3m 9-299, 11-38
plotm 1-10, 3-28
plotm 9-301, 11-38
Pluto *See* almanac 9-25
polcmap 3-71
polcmap 9-303
pole transformation 9-287
polybool 2-40, 2-41
polycon 10-101
Polyconic projection 10-101
polyconic projection 5-4
polygon
  boolean operations 2-40
  buffer zones 2-42
  displaying as line object 3-28
  displaying as patch object 3-31
  eliminating dateline crossing 2-41
  extracting segments 2-29
  intersection points 2-39
  merging segments 2-30
  surface area 2-36
polygon surface area 9-34
polyjoin 2-29
polymerge 2-30
polysplit 2-29
polyxpoly 9-311
polyxpoly 2-39, 2-41
position
  dead reckoning 9-130
  reckoning 9-340
Postel projection 10-45, 10-46
Postel, Guillaume 10-45
previewmap 9-313
printing maps 3-75
project 3-65
project 9-315

projection
data 9-263, 9-266
objects 9-315
projection *See* map projection
projection aspect 5-7
normal 5-7
oblique 5-8
See also origin vector
skew-oblique 5-12
transverse 5-8
Projection of Marinus 10-49, 10-50
property editors 11-64
Click-and-Drag 11-64
Guide 11-64
Ptolemy, Claudius 10-15, 10-47
Putnins 10-29
Putnins P4 projection 10-29
Putnins P5 projection 10-103
Putnins, Reinholds V. 10-103
`putnins5` 10-103
`putpole` 9-317

Q

`qrydata` 11-68
quadrangle surface area 9-39
`quartic` 10-105
Quartic Authalic projection 10-105
querying map data 11-68
`quiver` 3-67
`quiver3m` 9-319, 11-74
`quiverm` 3-62
`quiverm` 9-321, 11-77

R

`rad2deg` 9-323
`rad2dm` 9-324
`rad2dms` 9-324
`rad2km` 9-325
`rad2nm` 9-325
`rad2sm` 9-325
radians notation 2-22
radius of auxiliary sphere 9-356
radius of curvature 9-329
radius of planets 2-20
radius of planets *See* `almanac`
Rand McNally 10-107
range
angles 9-285, 9-472
finding cross fix position 9-98
raster data *See* matrix data
raster maps *See* matrix maps
`rc2yx` 9-327
`rcurve` 9-329
`readfields` 9-331
`readFK5` 12-47
`readfk5` 8-74
reading ASCII files 9-391
`readmtx` 9-335
`rec2geod` 9-339
`reckon` 2-13
`reckon` 9-340
reckoning 9-340, 11-90
reckoning position 2-13
Rectangular projection 10-49, 10-50
`reducem` 2-45
`reducem` 9-342
reducing data *See* vector data

regular matrix map
calculating required matrix size 9-387
creating colormap 11-21
displaying as image 9-206
encoding 9-208
encoding regions 11-85
latitude and longitude limits 9-223, 11-36
projecting 9-261, 11-49
projecting shaded relief 9-259
retrieving values 9-230
seeds for encoding 9-181
surface area 9-36
transforming to new coordinate system origin 9-280
trimming 9-246
regular matrix maps
accessing elements 2-53
defined 2-48
determining limits 2-50
determining size with scaling 2-54
displaying 3-34, 7-36
displaying image and surface coloring 3-37
displaying shaded relief 3-40
encoding 7-27
See also matrix maps
`resizem` 9-344
`restack` 9-346
rhumb line track
calculating from one point 9-425
calculating from two points 9-428
displaying 11-100
rhumb lines
approximating great circle tracks with 6-25
calculating points 2-32
defined 2-9
rhumb lines intersection 9-348
`rhxrh` 2-38
`rhxrh` 9-348
`robinson` 10-107
Robinson projection 10-107
Robinson, Arthur H. 10-107
`rootlayr` 9-350
`rotatem` 5-16
`rotatem` 9-351
`rotatetext` 9-353
rounding 9-355
`roundn` 9-355
`rsphere` 9-356
`russia` workspace 2-50
Ruysch, Johannes 10-47

S

Sanson-Flamsteed projection 10-109
`satbath` 12-50
Saturn *See* `almanac` 9-25
scale
between axes 3-58
printing 3-75
`scaleruler` 3-59
`scaleruler` 9-358
`scatterm` 3-62, 6-8
`scatterm` 9-365, 11-79
`scircle1` 2-31
`scircle1` 9-367
`scircle2` 2-31
`scircle2` 9-370
`scircleg` 1-11, 3-54
`scircleg` 9-372
`scirclui` 11-81
`scxsc` 2-38
`scxsc` 9-374
`sec2hm` 9-376
`sec2hms` 9-376

sec2hr 9-377
seconds notation 2-25
seedm 7-27
seedm 11-85
semimajor axis 9-231
semiminor axis 9-265
setltln 2-51
setltln 9-378
setm 3-8, 3-13, 3-17, 3-59
setm 9-379
setpostn 2-51
setpostn 9-382
shaded relief map
  constructing cdata 9-383
  constructing colormap 9-383
  general matrix map 9-405
  regular matrix map 9-259
shaded relief maps 3-40
shaderel 9-383
showaxes 9-385
showm 3-57
showm 9-386, 11-87
Siemon, Karl 10-79, 10-105
Simple Conic projection 10-47
Simple Cylindrical projection 10-99
simplifying data *See* vector data
sinusoid 10-109
Sinusoidal projection 10-15, 10-39, 10-63, 10-91, 10-109, 10-133
sizem 2-54
sizem 9-387
skew-oblique aspect 5-12
slope
  defined 2-58
sm2deg 9-388
sm2km 9-388
sm2nm 9-388
sm2rad 9-388
small circles
  calculating 9-367, 9-370
  calculating points 2-31
  defined 2-14
  defining with mouse 9-372
  displaying 11-81
  intersection 9-374
  intersection with great circles 9-168
smoothlong 9-389
spcread 9-391
speed units
  format for navigation functions 6-11
spzerom 2-56
spzerom 9-392
Stab 10-127
Stabius, Johannes 10-127
Stab-Werner projection 10-127
standard deviation of geographic data 6-4
standard deviation of geographic points 9-395
standard distance of geographic points 9-393
standard parallels
  for conic projections 3-32
stars workspace 4-41, 8-75
stdist 6-6
stdist 9-393
stdm 6-5
stdm 9-395
stem plot, example 3-63
stem3m 3-62
stem3m 9-397, 11-88
stereo 10-111
Stereographic projection 10-19, 10-75, 10-111
stereographic projection 5-6
str2angle 2-24
Sun *See* almanac 9-25

surface area
accessing from almanac 2-20
measuring logical maps 2-56
measuring polygons 2-36
measuring quadrangles 2-16
planets *See* almanac
polygon 9-34
quadrangle 9-39
regular matrix map 9-36
surface distance
along a parallel *See* departure
between track waypoints 9-219
between two points 9-120
calculating 11-90
measuring 2-10
surface objects
constructing graticule mesh 9-257
projecting 9-261, 9-296, 9-401, 9-408, 11-49, 11-62
projecting lighted 9-403, 11-94
surface objects, displaying 3-34
surfacem 9-401, 11-62
surfdist 11-90
surflm 3-39
surflm 9-403, 11-94
surflsrm 3-40, 3-43
surflsrm 9-405
surfm 3-43
surfm 9-408, 11-62
Sylvanus, Bernardus 10-15
symbol plot, example 3-64

T

tagm 9-410, 11-96
tbase 8-35
TerrainBase model 8-34
text
interacting with 3-55
mouse placement 9-192
projecting 9-411, 11-98
text objects
trimming to map frame 4-7
textm 9-411, 11-98
texture mapping
onto digital elevation maps 3-44
tgrline 8-25
tgrline 12-54
Thales 10-61
Thomas, Paul D. 10-81, 10-83, 10-85
TIGER data
ArcInfo files 12-60
MIF files 12-56
reading FIPS name files 12-36
TIGER/Line data 12-54
TIGER Thinned data 8-29
TIGER/Line data 8-25
tigermif 8-32
tigermif 12-56
tigerp 8-30
tigerp 12-60
tightmap 3-75
tightmap 9-413
time
converting to matrix elements 9-203
converting to string 9-414
converting units 9-201, 9-202, 9-204, 9-205, 9-376, 9-377, 9-416
time units
conventions for navigation 6-37
converting between formats 2-25
description of formats 2-25
time zone
determining from longitude 9-417

time zones
navigational 6-33
time2str 6-37
time2str 9-414
timedim 2-25
timedim 9-416
timezone 6-35, 6-36
timezone 9-417
tissot 3-62, 5-14
tissot 9-419
tissot indicatrices
projecting 9-419
Tissot Modified Sinusoidal projection 10-113
Tissot, N. A. 10-113
Tobler, Waldo R. 10-79
topo workspace 2-4, 4-24
topographical maps
See digital elevation maps (DEMs)
track
See great circle track
See navigational track
See rhumb line track
track 6-26
track 9-422
track waypoints
azimuth 9-219
distance 9-219
track1 2-32
track1 9-425
track2 2-32, 3-29
track2 9-428
trackg 1-11, 3-54
trackg 9-430
tracks
See great circles
See rhumb lines
trackui 11-100
transformation of coordinate system 9-351
See coordinate transformation
transverse aspect 5-8
Transverse Mercator projection 10-119
transverse Mercator projection 5-29
trimcart 3-66
trimcart 9-432
trimming data 2-43, 7-25
trisurf 3-65
trystan 10-115
Trystan Edwards Cylindrical projection 10-43, 10-115
Tunhuang star chart 10-87

U

U.S. Army 10-119
U.S. matrix data
political 4-36
terrain 4-37
U.S. vector data
creating base maps 4-34
low resolution 4-27
medium resolution 4-32
uimaptbx 11-104
units
converting *See* conversion
See angle units
See distance units
See time units
testing for valid abbreviations 9-433
testing for valid strings 9-433
unitstr 9-433
Universal Polar Stereographic (UPS) projection 5-26
Universal Polar Stereographic projection 10-117

Universal Transverse Mercator (UTM) projection 5-26
Universal Transverse Mercator projection 10-117, 10-119
ups 10-117
UPS projection 10-117
Uranus *See* almanac 9-25
usahi 9-434
usahi function 4-34
usahi workspace 4-32
usalo 9-435
usalo workspace 3-31, 3-50, 4-27
usamap 1-7, 1-8, 4-31, 4-34
usamap 9-437
usamtx workspace 4-36
UserData property
  used by the Mapping Toolbox 3-3, 3-5
USGS DEM data 8-68
  reading files 12-63, 12-67
  returning filenames 12-69
usgs24kdem 12-63
usgsdem 8-70
usgsdem 12-63, 12-67
usgsdems 8-69
usgsdems 12-69
UTM
  ellipsoid for 5-28
  zone 5-28
utm 5-26
utm 10-119
UTM projection 10-117, 10-119
utmgeoid 9-443
utmzone 9-441

V

valued maps
  defined 2-55
Van der Grinten I projection 10-121
Van der Grinten projection 10-121
Van der Grinten, Alphons J. 10-121
vec2mtx 4-22, 4-23
vec2mtx 9-444
vector data
  calculating intersections 2-37
  converting to matrix format 11-46
  creating 2-27, 2-31
  displaying as lines 9-224, 9-299, 9-301, 11-38
  displaying as patches 9-150, 9-152, 9-292, 9-294, 11-29
  geographic interpolation 2-33
  reducing 9-342
  simplifying/reducing 2-45
  trimming 9-243, 9-244
  trimming data to a region 2-43
Vector Map Level 0 8-3
vector maps
  data format of 2-27
  defined 2-3
  delineation of objects in 2-27
  displaying as lines 3-28
  displaying as patches 3-31
vectors
  projected directions 5-21
Venus *See* almanac 9-25
Vertical Perspective Azimuthal projection 10-123
vfwdtran 3-67, 5-22

vfwdtran 9-446
vgrint1 10-121
viewshed
  defined 2-60
viewshed 2-60
vinvtran 9-454
VMAP0 8-3
  command line interface 8-12
  importing interactively 8-5
  low level access 8-19
vmap0data 12-70
vmap0read 12-74
vmap0rhead 12-77
vmap0ui 1-13, 8-5
volume of planets 2-20
volume of planets *See* almanac
von Hammer, H. H. Ernst 10-65
vperspec 10-123

W

Wagner I projection 10-71
Wagner IV projection 10-125
Wagner, Karlheinz 10-71, 10-125
wagner4 10-125
waypoints
  calculating 6-25, 9-164
  connecting 6-26
  defined 6-11
  See also track waypoints
  selecting with mouse 6-26
werner 10-127
Werner projection 10-15, 10-127
Werner, Johannes 10-127
westof 9-457
wetch 10-129
Wetch Cylindrical projection 10-129
Wetch projection 10-25
Wetch, J. 10-129
wiechel 10-131
Wiechel projection 10-131
Wiechel, H. 10-131
winkel 10-133
Winkel I projection 10-133
Winkel, Oswald 10-133
world matrix data
  political data 4-21
  terrain data 4-24
world vector data
  atlas data, high-resolution 4-15
  atlas data, low-resolution 4-5
  coastline data 4-3
  deleting data with tags 4-10
  displaying atlas data 4-5, 4-13
  extracting data with tags 4-11
worldhi 9-458
worldhi 4-18
worldhi workspace 4-15
worldlo 9-462
worldlo function 4-13, 4-15
worldlo workspace 4-5
worldmap 1-3, ??–1-7, 4-13
worldmtxmed workspace 4-21
Wright projection 10-87
Wright, Edward 10-87

Y

Young, A. E. 10-19
yx2rc 9-469

Z

zdatam 9-471, 11-105
Zenithal Equal-Area projection 10-73
Zenithal Equidistant projection 10-45
Zenithal Equivalent projection 10-73
Zenithal projection 10-45, 10-46
zero22pi 2-8
zero22pi 9-472
zerom 2-56
zerom 9-473
zeros
  creating matrix map 9-473
  creating sparse matrix map 9-392
zooming in and out of map displays 11-59