

Package ‘mapproj’

January 12, 2023

Title Map Projections

Version 1.2.11

Date 2023-01-12

Author Doug McIlroy. Packaged for R by Ray Brownrigg and Thomas P Minka, transition to Plan 9 codebase by Roger Bivand.

Description Converts latitude/longitude into projected coordinates.

Depends R (>= 3.0.0), maps (>= 2.3-0)

Imports stats, graphics

License Lucent Public License

Maintainer Alex Deckmyn <alex.deckmyn@meteo.be>

NeedsCompilation yes

Repository CRAN

Date/Publication 2023-01-12 20:20:02 UTC

R topics documented:

map.grid	1
mapproject	3
Index	7

map.grid	<i>Draw a latitude/longitude grid on a projected map</i>
----------	--

Description

Draws a grid on an existing map.

Usage

```
map.grid(lim, nx=9, ny=9, labels=TRUE, pretty=TRUE, cex, col, lty,  
font, ...)
```

Arguments

lim	a vector of 4 numbers specifying limits: c(lon.low, lon.high, lat.low, lat.high). lim can also be a list with a component named range, such as the result of map, from which limits are taken.
nx, ny	the desired number of equally-spaced longitude and latitude lines
labels	logical to indicate if grid lines should be labeled with longitude/latitude values.
pretty	If TRUE, grid lines will be placed at round numbers.
cex, col, lty, font	passed to arguments to par
...	additional arguments passed to lines and text, e.g. col to change the color of the grid and lty to change the line type.

Value

Equally-spaced lines of constant longitude and lines of constant latitude are superimposed on the current map, using the current projection. These lines will appear curved under most projections, and give an idea of how the projection works.

See Also

[map](#)

Examples

```
library(maps)
m <- map("usa", plot=FALSE)
map("usa", project="albers", par=c(39, 45))
map.grid(m)

# get unprojected world limits
m <- map('world', plot=FALSE)

# center on NYC
map('world', proj='azequalarea', orient=c(41, -74, 0))
map.grid(m, col=2)
points(mapproject(list(y=41, x=-74)), col=3, pch="x", cex=2)

map('world', proj='orth', orient=c(41, -74, 0))
map.grid(m, col=2, nx=6, ny=5, label=FALSE, lty=2)
points(mapproject(list(y=41, x=-74)), col=3, pch="x", cex=2)

# center on Auckland
map('world', proj='orth', orient=c(-36.92, 174.6, 0))
map.grid(m, col=2, label=FALSE, lty=2)
points(mapproject(list(y=-36.92, x=174.6)), col=3, pch="x", cex=2)

m <- map('nz')
# center on Auckland
map('nz', proj='azequalarea', orient=c(-36.92, 174.6, 0))
```

```
points(mapproject(list(y=-36.92, x=174.6)), col=3, pch="x", cex=2)
map.grid(m, col=2)
```

mapproject

Apply a Map Projection

Description

Converts latitude and longitude into projected coordinates.

Usage

```
mapproject(x, y, projection="", parameters=NULL, orientation=NULL)
```

Arguments

<code>x,y</code>	two vectors giving longitude and latitude coordinates of points on the earth's surface to be projected. A list containing components named <code>x</code> and <code>y</code> , giving the coordinates of the points to be projected may also be given. Missing values (NAs) are allowed. The coordinate system is degrees of longitude east of Greenwich (so the USA is bounded by negative longitudes) and degrees north of the equator.
<code>projection</code>	optional character string that names a map projection to use. If the string is "" then the previous projection is used, with parameters modified by the next two arguments.
<code>parameters</code>	optional numeric vector of parameters for use with the <code>projection</code> argument. This argument is optional only in the sense that certain projections do not require additional parameters. If a projection does require additional parameters, these must be given in the <code>parameters</code> argument.
<code>orientation</code>	An optional vector <code>c(latitude, longitude, rotation)</code> which describes where the "North Pole" should be when computing the projection. Normally this is <code>c(90, 0)</code> , which is appropriate for cylindrical and conic projections. For a planar projection, you should set it to the desired point of tangency. The third value is a clockwise rotation (in degrees), which defaults to the midrange of the longitude coordinates in the map. This means that two maps plotted with their own default orientation may not line up. To avoid this, you should not specify a projection twice but rather default to the previous projection using <code>projection=""</code> . See the examples.

Details

Each standard projection is displayed with the Prime Meridian (longitude 0) being a straight vertical line, along which North is up. The orientation of nonstandard projections is specified by the three `parameters=c(lat, lon, rot)`. Imagine a transparent gridded sphere around the globe. First turn the overlay about the North Pole so that the Prime Meridian (longitude 0) of the overlay coincides with meridian `lon` on the globe. Then tilt the North Pole of the overlay along its Prime Meridian to latitude `lat` on the globe. Finally again turn the overlay about its "North Pole" so that its Prime

Meridian coincides with the previous position of (the overlay) meridian rot. Project the desired map in the standard form appropriate to the overlay, but presenting information from the underlying globe.

In the descriptions that follow each projection is shown as a function call; if it requires parameters, these are shown as arguments to the function. The descriptions are grouped into families.

Equatorial projections centered on the Prime Meridian (longitude 0). Parallels are straight horizontal lines.

mercator() equally spaced straight meridians, conformal, straight compass courses

sinusoidal() equally spaced parallels, equal-area, same as **bonne(0)**

cylequalarea(lat0) equally spaced straight meridians, equal-area, true scale on **lat0**

cylindrical() central projection on tangent cylinder

rectangular(lat0) equally spaced parallels, equally spaced straight meridians, true scale on **lat0**

gall(lat0) parallels spaced stereographically on prime meridian, equally spaced straight meridians, true scale on **lat0**

mollweide() (homalographic) equal-area, hemisphere is a circle

gilbert() sphere conformally mapped on hemisphere and viewed orthographically

Azimuthal projections centered on the North Pole. Parallels are concentric circles. Meridians are equally spaced radial lines.

azequidistant() equally spaced parallels, true distances from pole

azequalarea() equal-area

gnomonic() central projection on tangent plane, straight great circles

perspective(dist) viewed along earth's axis **dist** earth radii from center of earth

orthographic() viewed from infinity

stereographic() conformal, projected from opposite pole

laue() radius = $\tan(2 * \text{colatitude})$ used in xray crystallography

fisheye(n) stereographic seen through medium with refractive index **n**

newyorker(r) radius = $\log(\text{colatitude}/r)$ map from viewing pedestal of radius **r** degrees

Polar conic projections symmetric about the Prime Meridian. Parallels are segments of concentric circles. Except in the Bonne projection, meridians are equally spaced radial lines orthogonal to the parallels.

conic(lat0) central projection on cone tangent at **lat0**

simpleconic(lat0,lat1) equally spaced parallels, true scale on **lat0** and **lat1**

lambert(lat0,lat1) conformal, true scale on **lat0** and **lat1**

albers(lat0,lat1) equal-area, true scale on **lat0** and **lat1**

bonne(lat0) equally spaced parallels, equal-area, parallel **lat0** developed from tangent cone

Projections with bilateral symmetry about the Prime Meridian and the equator.

polyconic() parallels developed from tangent cones, equally spaced along Prime Meridian

aitoff() equal-area projection of globe onto 2-to-1 ellipse, based on azequalarea
lagrange() conformal, maps whole sphere into a circle
bicentric(lon0) points plotted at true azimuth from two centers on the equator at longitudes +lon0 and -lon0, great circles are straight lines (a stretched gnomonic projection)
elliptic(lon0) points are plotted at true distance from two centers on the equator at longitudes +lon0 and -lon0
globular() hemisphere is circle, circular arc meridians equally spaced on equator, circular arc parallels equally spaced on 0- and 90-degree meridians
vandergrinten() sphere is circle, meridians as in globular, circular arc parallels resemble mercator
eisenlohr() conformal with no singularities, shaped like polyconic

Doubly periodic conformal projections.

guyou W and E hemispheres are square
square world is square with Poles at diagonally opposite corners
tetra map on tetrahedron with edge tangent to Prime Meridian at S Pole, unfolded into equilateral triangle
hex world is hexagon centered on N Pole, N and S hemispheres are equilateral triangles

Miscellaneous projections.

harrison(dist,angle) oblique perspective from above the North Pole, dist earth radii from center of earth, looking along the Date Line angle degrees off vertical
trapezoidal(lat0,lat1) equally spaced parallels, straight meridians equally spaced along parallels, true scale at lat0 and lat1 on Prime Meridian
lune(lat,angle) conformal, polar cap above latitude lat maps to convex lune with given angle at 90E and 90W

Retroazimuthal projections. At every point the angle between vertical and a straight line to "Mecca", latitude lat0 on the prime meridian, is the true bearing of Mecca.

mecca(lat0) equally spaced vertical meridians
homing(lat0) distances to Mecca are true

Maps based on the spheroid. Of geodetic quality, these projections do not make sense for tilted orientations.

sp_mercator() Mercator on the spheroid.
sp_albers(lat0,lat1) Albers on the spheroid.

Value

list with components named x and y, containing the projected coordinates. NAs project to NAs. Points deemed unprojectable (such as north of 80 degrees latitude in the Mercator projection) are returned as NA. Because of the ambiguity of the first two arguments, the other arguments must be given by name.

Each time mapproject is called, it leaves on frame 0 the dataset `.Last.projection`, which is a list with components `projection`, `parameters`, and `orientation` giving the arguments from the

call to `mapproject` or as constructed (for orientation). Subsequent calls to `mapproject` will get missing information from `.Last.projection`. Since `map` uses `mapproject` to do its projections, calls to `mapproject` after a call to `map` need not supply any arguments other than the data.

References

Richard A. Becker, and Allan R. Wilks, "Maps in S", *AT&T Bell Laboratories Statistics Research Report, 1991*. <https://web.archive.org/web/20070824013345/http://www.research.att.com/areas/stat/doc/93.2.ps>

M. D. McIlroy, Documentation from the *Tenth Edition UNIX Manual, Volume 1*, Saunders College Publishing, 1990.

Examples

```
library(maps)
# Bonne equal-area projection with state abbreviations
map("state",proj='bonne', param=45)
data(state)
text(mapproject(state.center), state.abb)

# this does not work because the default orientations are different:
map("state",proj='bonne', param=45)
text(mapproject(state.center,proj='bonne',param=45),state.abb)

map("state",proj="albers",par=c(30,40))
map("state",par=c(20,50)) # another Albers projection

map("world",proj="gnomonic",orient=c(0,-100,0)) # example of orient
# see map.grid for more examples

# tests of projections added RSB 091101
projlist <- c("aitoff", "albers", "azequalarea", "azequidist", "bicentric",
"bonne", "conic", "cylequalarea", "cylindrical", "eisenlohr", "elliptic",
"fisheye", "gall", "gilbert", "guyou", "harrison", "hex", "homing",
"lagrange", "lambert", "laue", "lune", "mercator", "mollweide", "newyorker",
"orthographic", "perspective", "polyconic", "rectangular", "simpleconic",
"sinusoidal", "tetra", "trapezoidal")
x <- seq(-100, 0, 10)
y <- seq(-45, 45, 10)
xy <- expand.grid(x=x, y=y)
pf <- c(0, 2, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 2, 0, 1, 0, 2, 0, 2,
0, 0, 1, 0, 1, 0, 1, 2, 0, 0, 2)
res <- vector(mode="list", length=length(projlist))
for (i in seq(along=projlist)) {
  if (pf[i] == 0) res[[i]] <- mapproject(xy$x, xy$y, projlist[i])
  else if (pf[i] == 1) res[[i]] <- mapproject(xy$x, xy$y, projlist[i], 0)
  else res[[i]] <- mapproject(xy$x, xy$y, projlist[i], c(0,0))
}
names(res) <- projlist
lapply(res, function(p) rbind(p$x, p$y))
```

Index

* **aplot**

map.grid, 1

* **dplot**

mapproject, 3

map, 2

map.grid, 1

mapproject, 3