Package ‘geosphere’

April 11, 2015

Type Package
Title Spherical Trigonometry
Version 1.3-13
Date 2015-3-17
Depends sp, R (>= 2.10.0)
Suggests methods, raster
Description Spherical trigonometry for geographic applications. That is, compute distances and related measures for angular (longitude/latitude) locations.
License GPL (>= 3)
LazyLoad yes
NeedsCompilation no
Author Robert J. Hijmans [cre, aut],
   Ed Williams [ctb],
   Chris Vennes [ctb]
Maintainer Robert J. Hijmans <rNhijmans@gmail.com>
Repository CRAN
Date/Publication 2015-04-11 07:48:26

R topics documented:

  geosphere-package ........................................ 2
  alongTrackDistance ...................................... 3
  antipode .................................................. 4
  areaPolygon .............................................. 5
  bearing ................................................... 6
  bearingRhumb ............................................. 7
  centroid .................................................. 8
  daylength ................................................ 9
  destPoint ................................................. 10
  destPointRhumb ......................................... 11
  dist2ge .................................................. 12
This package implements spherical trigonometry functions for geographic applications. There are functions to compute distance and direction (bearing, azimuth, course) along Great Circles (= shortest distance on a sphere, or "as the crow flies") and along rhumb lines (lines of constant direction) as well as functions to compute intersections of great circles, and of rhumb lines. There are also functions to compute the distance between points and polylines, and to characterize spherical polygons; for random sampling on a sphere, and to compute daylength. See the vignette `vignette('geosphere')` for examples.

Geographic locations must be specified in latitude and longitude in degrees (NOT radians). Degrees are (obviously) in decimal notation. Thus 12 degrees, 30 minutes, 10 seconds = 12 + 10/60 + 30/3600 = 12.175 degrees. The Southern and Western hemispheres have a negative sign.
The default unit of distance is meter; but this can be adjusted by supplying a different radius $r$ to functions.

Directions are expressed in degrees ($N = 0$ and $360$, $E = 90$, $S = 180$, and $W = 270$ degrees).

Acknowledgements

David Purdy, Bill Monahan, and George Wang for suggestions to improve the package.

Author(s)

Robert Hijmans, Ed Williams, Chris Veness

Maintainer: Robert J. Hijmans <r.hijmans@gmail.com>

References

http://williams.best.vwh.net/avform.htm
http://www.movable-type.co.uk/scripts/latlong.html
http://en.wikipedia.org/wiki/Great_circle_distance
http://mathworld.wolfram.com/SphericalTrigonometry.html

alongTrackDistance Along Track Distance

Description

The "along track distance" is the distance from the start point ($p_1$) to the closest point on the path to a third point ($p_3$), following a great circle path defined by points $p_1$ and $p_2$. See distgc for the "cross track distance".

Usage

alongTrackDistance($p_1$, $p_2$, $p_3$, $r=6378137$)

Arguments

$p_1$ longitude/latitude of point(s). Can be a vector of two numbers, a matrix of 2 columns (first one is longitude, second is latitude) or a SpatialPoints* object

$p_2$ as above

$p_3$ as above

$r$ radius of the earth; default = 6378137m

Value

A distance in units of $r$ (default is meters)
Author(s)
Ed Williams and Robert Hijmans

See Also
dist2gc

Examples
alongTrackDistance(c(0,0),c(60,60),c(50,40))

Description
Compute an antipode, or check whether two points are antipodes. Antipodes are places on Earth that are diametrically opposite to one another; and could be connected by a straight line through the centre of the Earth.

Antipodal points are connected by an infinite number of great circles (e.g. the meridians connecting the poles), and can therefore not be used in some great circle based computations.

Usage
antipode(p)
antipodal(p1, p2, tol=1e-9)

Arguments
p Longitude/latitude of a single point, in degrees; can be a vector of two numbers, a matrix of 2 columns (first one is longitude, second is latitude) or a SpatialPoints* object
p1 as above
p2 as above
tol tolerance for equality

Value
antipodal points or a logical value (TRUE if antipodal)

Author(s)
Robert Hijmans

References
http://en.wikipedia.org/wiki/Antipodes
areaPolygon

Examples


tantipode(rbind(c(5,52), c(-120,37), c(-60,0), c(0,70)))
tantipodal(c(0,0), c(180,0))

areaPolygon(x, ...)

Arguments

x Longitude/latitude of the points forming a polygon; Must be a matrix of 2
columns (first one is longitude, second is latitude) or a SpatialPolygons* ob-
ject

... Additional arguments. One implemented r=6378137, the radius of the earth;
default = 6378137 m

Value

area. Unit is the square of the unit of r (default is square meter).

Author(s)

Robert J. Hijmans. Based on code by Jason_Steven (http://forum.worldwindcentral.com/
showthread.php?p=69704)

References

Mathematical Geology 19: 335-346

See Also

centroid, perimeter

Examples

p <- rbind(c(-180, -20), c(-140, 55), c(10, 0), c(-140, -60), c(-180, -20))
areaPolygon(p)
Description

Get the initial bearing (direction; azimuth) to go from p1 to p2 following the shortest path (a Great Circle). Note that the bearing of travel changes continuously while going along a great circle. A route with constant bearing is a rhumb line (see bearingRhumb).

Usage

bearing(p1, p2)

Arguments

<table>
<thead>
<tr>
<th>p1</th>
<th>longitude/latitude of point(s). Can be a vector of two numbers, a matrix of 2 columns (first one is longitude, second is latitude) or a SpatialPoints* object</th>
</tr>
</thead>
<tbody>
<tr>
<td>p2</td>
<td>as above</td>
</tr>
</tbody>
</table>

Value

Bearing in degrees

Author(s)

Robert Hijmans

References

http://williams.best.vwh.net/ftp/avsig/avform.txt
http://www.movable-type.co.uk/scripts/latlong.html

See Also

bearingRhumb

Examples

bearing(c(10,10),c(20,20))
**bearingRhumb**

---

**bearingRhumb**  
*Rhumbline direction*

**Description**

Bearing (direction of travel; true course) along a rhumb line (loxodrome) between two points.

**Usage**

`bearingRhumb(p1, p2)`

**Arguments**

- `p1` longitude/latitude of point(s). Can be a vector of two numbers, a matrix of 2 columns (first one is longitude, second is latitude) or a SpatialPoints* object
- `p2` as above

**Value**

A direction (bearing) in degrees

**Note**

Unlike most great circles, a rhumb line is a line of constant bearing (direction), i.e. tracks of constant true course. The meridians and the equator are both rhumb lines and great circles. Rhumb lines approaching a pole become a tightly wound spiral.

**Author(s)**

Chris Veness and Robert Hijmans, based on formulae by Ed Williams

**References**

http://williams.best.vwh.net/avform.htm#Rhumb

http://en.wikipedia.org/wiki/Rhumb_line

**See Also**

`bearing`, `distRhumb`

**Examples**

`bearingRhumb(c(10,10),c(20,20))`
### Description

Compute the centroid of longitude/latitude polygons. Unlike other functions in this package, there is no spherical trigonometry involved in the implementation of this function. Instead, the function projects the polygon to the (conformal) Mercator coordinate reference system, computes the centroid, and then inversely projects it to longitude and latitude. This approach fails for polygons that include one of the poles. The function should work for polygons that cross the -180/180 meridian (date line).

### Usage

```r
centroid(x, ...)
```

### Arguments

- `x` a 2-column matrix (longitude/latitude)
- `...` Additional arguments. None implemented

### Value

A matrix (longitude/latitude)

### Note

For multi-part polygons, the centroid of the largest part is returned.

### Author(s)

Robert J. Hijmans

### See Also

- `area`, `perimeter`

### Examples

```r
pol <- rbind(c(-180,-20), c(-160,5), c(-60, 0), c(-160,-60), c(-180,-20))
centroid(pol)
```
daylength

| daylength | Daylength |

Description

Compute daylength (photoperiod) for a latitude and date.

Usage

daylength(lat, doy)

Arguments

- lat: latitude, in degrees. I.e. between -90.0 and 90.0
- doy: Integer, day of the year (1..365) for leap years; or an object of class Date; or a character that can be coerced into a date, using `yyyy-mm-dd` format, e.g. '1982-11-23'

Value

Daylength in hours

Author(s)

Robert J. Hijmans

References


Examples

daylength(-25, '2010-10-10')
daylength(45, 1:365)

# average monthly daylength
dl <- daylength(45, 1:365)
tapply(dl, rep(1:12, c(31,28,31,30,31,30,31,31,30,31,30,31)), mean)
**destPoint**  
*Destination given bearing (direction) and distance, when following a Great Circle*

**Description**

Calculate the destination point travelling along a (shortest distance) Great Circle, given a start point, initial direction (bearing), and distance.

**Usage**

```r
destPoint(p, b, d, r = 6378137)
```

**Arguments**

- **p**: Longitude and Latitude of point(s), in degrees. Can be a vector of two numbers, a matrix of 2 columns (first one is longitude, second is latitude) or a SpatialPoints* object.
- **b**: Numeric. Bearing (direction) in degrees.
- **d**: Numeric. Distance in the same unit as **r** (default is meters).
- **r**: Numeric. Radius of the earth; default = 6378137 m.

**Value**

A pair of coordinates (longitude/latitude)

**Note**

Direction changes continuously when traveling along a Great Circle. Therefore, the final direction is not the same as the initial direction. You can compute the final direction with `finalBearing` (see examples, below).

**Author(s)**

Robert Hijmans, based on code by Chris Veness

**References**

- [http://www.movable-type.co.uk/scripts/latlong.html](http://www.movable-type.co.uk/scripts/latlong.html)

**Examples**

```r
p <- c(5, 52)
d <- destPoint(p, 30, 10000)

# final direction, when arriving at endpoint:
finalBearing(d, p)
```
Description

Calculate the destination point when travelling along a 'rhumb line' (loxodrome), given a start point, direction, and distance.

Usage

destPointRhumb(p, b, d, r = 6378137)

Arguments

p longitude/latitude of point(s). Can be a vector of two numbers, a matrix of 2 columns (first one is longitude, second is latitude) or a SpatialPoints* object
b bearing (direction) in degrees
d distance; in the same unit as r (default is meters)
r radius of the earth; default = 6378137 m

Value

Coordinates (longitude/latitude) of a point

Author(s)

Chris Veness; ported to R by Robert Hijmans

References

http://williams.best.vwh.net/avform.htm#Rhumb
http://www.movable-type.co.uk/scripts/latlong.html
http://en.wikipedia.org/wiki/Rhumb_line

See Also

destPoint

Examples

destPointRhumb(c(0,0), 30, 100000, r = 6378137)
dist2gc

Cross Track Distance

Description

Compute the distance of a point to a great-circle path (also referred to as the cross track distance or cross track error). The great circle is defined by \( p_1 \) and \( p_2 \), while \( p_3 \) is the point away from the path.

Usage

\[
\text{dist2gc}(p_1, p_2, p_3, r=6378137)
\]

Arguments

- \( p_1 \): Start of great circle path. longitude/latitude of point(s). Can be a vector of two numbers, a matrix of 2 columns (first one is longitude, second is latitude) or a SpatialPoints* object
- \( p_2 \): End of great circle path. As above
- \( p_3 \): Point away from the great circle path. As for \( p_2 \)
- \( r \): radius of the earth; default = 6378137

Value

A distance in units of \( r \) (default is meters)

The sign indicates which side of the path \( p_3 \) is on. Positive means right of the course from \( p_1 \) to \( p_2 \), negative means left.

Author(s)

Ed Williams and Robert Hijmans

References

http://www.movable-type.co.uk/scripts/latlong.html
http://williams.best.vwh.net/ftp/avsig/avform.txt

See Also

dist2Line, alongTrackDistance

Examples

\[
\text{dist2gc}(c(0,0),c(90,90),c(80,80))
\]
dist2Line

Distance between points and lines or the border of polygons.

Description

The shortest distance between points and polylines or polygons.

Usage

dist2Line(p, line, distfun=distHaversine)

Arguments

- **p**: longitude/latitude of point(s). Can be a vector of two numbers, a matrix of 2 columns (first one is longitude, second is latitude) or a SpatialPoints* object
- **line**: longitude/latitude of line as a matrix of 2 columns (first one is longitude, second is latitude) or a SpatialLines* or SpatialPolygons* object
- **distfun**: A distance function, such as distHaversine

Value

matrix with distance and lon/lat of the nearest point on the line. Distance is in the same unit as \( r \) in the distfun(default is meters). If line is a Spatial* object, the ID (index) of (one of) the nearest objects is also returned. Thus if the objects are polygons and the point is inside a polygon the function may return the ID of a neighboring polygon that shares the nearest border. You can use the over functions in packages sp or rgeos for point-in-polygon queries.

Author(s)

George Wang and Robert Hijmans

See Also

dist2gc, alongTrackDistance

Examples

```r
line <- rbind(c(-180,-20), c(-150,-10), c(-140,55), c(10, 0), c(-140,-60))
pnts <- rbind(c(-170,0), c(-75,0), c(-70,-10), c(-80,20), c(-100,-50),
             c(-100,-60), c(-100,-40), c(-100,-20), c(-100,-10), c(-100,0))
d = dist2Line(pnts, line)
plot(makeLine(line), type='l')
points(line)
points(pnts, col='blue', pch=20)
points(d[,2], d[,3], col='red', pch='x')
for (i in 1:nrow(d)) lines(gcIntermediate(pnts[i,], d[i,2:3], 10), lwd=2)
```
distCosine

‘Law of cosines’ great circle distance

Description
The shortest distance between two points (i.e., the 'great-circle-distance' or 'as the crow flies'), according to the 'law of the cosines'. This method assumes a spherical earth, ignoring ellipsoidal effects.

Usage
`distCosine(p1, p2, r=6378137)`

Arguments
- `p1` longitude/latitude of point(s). Can be a vector of two numbers, a matrix of 2 columns (first one is longitude, second is latitude) or a SpatialPoints* object
- `p2` as above
- `r` radius of the earth; default = 6378137 m

Value
Vector of distances in the same unit as `r` (default is meters)

Author(s)
Robert Hijmans

References
http://en.wikipedia.org/wiki/Great_circle_distance

See Also
`distHaversine, distVincentySphere, distVincentyEllipsoid, distMeeus`

Examples
`distCosine(c(0,0),c(90,90))`
Description

The shortest distance between two points (i.e., the 'great-circle-distance' or 'as the crow flies'), according to the 'haversine method'. This method assumes a spherical earth, ignoring ellipsoidal effects.

Usage

```
distHaversine(p1, p2, r=6378137)
```

Arguments

- **p1**: longitude/latitude of point(s). Can be a vector of two numbers, a matrix of 2 columns (first one is longitude, second is latitude) or a SpatialPoints* object
- **p2**: as above
- **r**: radius of the earth; default = 6378137 m

Details

The Haversine ('half-versed-sine') formula was published by R.W. Sinnott in 1984, although it has been known for much longer. At that time computational precision was lower than today (15 digits precision). With current precision, the spherical law of cosines formula appears to give equally good results down to very small distances. If you want greater accuracy, you could use the `distVincentyEllipsoid` method.

Value

Vector of distances in the same unit as `r` (default is meters)

Author(s)

Chris Veness and Robert Hijmans

References


http://www.movable-type.co.uk/scripts/latlong.html

http://en.wikipedia.org/wiki/Great_circle_distance

See Also

`distCosine, distVincentySphere, distVincentyEllipsoid, distMeeus`
**Examples**

```r
distHaversine(c(0,0), c(90,90))
```

---

**distm**

*Distance matrix*

**Description**

Distance matrix of a set of points, or between two sets of points

**Usage**

```r
distm(x, y, fun=distHaversine)
```

**Arguments**

- **x**: longitude/latitude of point(s). Can be a vector of two numbers, a matrix of 2 columns (first one is longitude, second is latitude) or a SpatialPoints* object
- **y**: Same as x. If missing, y is the same as x
- **fun**: A function to compute distances (e.g., distCosine, distHaversine, distVincenty*)

**Value**

Matrix of distances

**Author(s)**

Robert Hijmans

**References**


**See Also**

*distCosine, distHaversine, distVincentySphere, distVincentyEllipsoid*

**Examples**

```r
xy <- rbind(c(0,0), c(90,90), c(10,10), c(-120,-45))
distm(xy)
xy2 <- rbind(c(0,0), c(10,-10))
distm(xy, xy2)
```
distMeeus

'distMeeus' great circle distance

Description

The shortest distance between two points (i.e., the 'great-circle-distance' or 'as the crow flies'), according to the 'Meeus' method. This method uses an ellipsoid, and the results should be very accurate.

Usage

distMeeus(p1, p2, a=6378137, f=1/298.257223563)

Arguments

- **p1**: longitude/latitude of point(s), in degrees 1; can be a vector of two numbers, a matrix of 2 columns (first one is longitude, second is latitude) or a SpatialPoints* object
- **p2**: as above
- **a**: Equatorial axis of ellipsoid
- **f**: Inverse flattening of ellipsoid

Details

Parameters from the WGS84 ellipsoid are used by default. It is the best available global ellipsoid, but for some areas other ellipsoids could be preferable, or even necessary if you work with a printed map that refers to that ellipsoid. Here are parameters for some commonly used ellipsoids:

<table>
<thead>
<tr>
<th>ellipsoid</th>
<th>a</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>WGS84</td>
<td>6378137</td>
<td>1/298.257223563</td>
</tr>
<tr>
<td>GRS80</td>
<td>6378137</td>
<td>1/298.257222101</td>
</tr>
<tr>
<td>GRS67</td>
<td>6378160</td>
<td>1/298.25</td>
</tr>
<tr>
<td>Airy 1830</td>
<td>6377563.396</td>
<td>1/299.3249646</td>
</tr>
<tr>
<td>Bessel 1841</td>
<td>6377397.155</td>
<td>1/299.1528434</td>
</tr>
<tr>
<td>Clarke 1880</td>
<td>6378249.145</td>
<td>1/293.465</td>
</tr>
<tr>
<td>Clarke 1866</td>
<td>6378206.4</td>
<td>1/294.9786982</td>
</tr>
<tr>
<td>International 1924</td>
<td>6378388</td>
<td>1/297</td>
</tr>
<tr>
<td>Krasovsky 1940</td>
<td>6378245</td>
<td>1/298.2997381</td>
</tr>
</tbody>
</table>


Value

Distance value in the same units as parameter a of the ellipsoid (default is meters)
**Note**

This algorithm is also used in the `spDist`s function in the sp package.

**Author(s)**

Robert Hijmans, based on a script by Stephen R. Schmitt

**References**


**See Also**

`distVincentyEllipsoid`, `distVincentySphere`, `distHaversine`, `distCosine`

**Examples**

```r
distMeeus(c(0, 0), c(90, 90))
# on a 'Clarke 1880' ellipsoid
distMeeus(c(0, 0), c(90, 90), a=6378249.145, f=1/293.465)
```

---

**distRhumb**

*Distance along a rhumb line*

**Description**

A rhumb line (loxodrome) is a path of constant bearing (direction), which crosses all meridians at the same angle.

**Usage**

`distRhumb(p1, p2, r=6378137)`

**Arguments**

- `p1` longitude/latitude of point(s). Can be a vector of two numbers, a matrix of 2 columns (first one is longitude, second is latitude) or a SpatialPoints* object
- `p2` as above
- `r` radius of the earth; default = 6378137 m

**Details**

Rhumb (from the Spanish word for course, 'rumbo') lines are straight lines on a Mercator projection map. They were used in navigation because it is easier to follow a constant compass bearing than to continually adjust the bearing as is needed to follow a great circle, even though rhumb lines are normally longer than great-circle (orthodrome) routes. Most rhumb lines will gradually spiral towards one of the poles.
Value
distance in units of r (default=meters)

Author(s)
Robert Hijmans and Chris Veness

References
http://www.movable-type.co.uk/scripts/latlong.html

See Also
distCosine, distHaversine, distVincentySphere, distVincentyEllipsoid

Examples
distRhumb(c(10,10),c(20,20))

distVincentyEllipsoid  'Vincenty' (ellipsoid) great circle distance

Description
The shortest distance between two points (i.e., the 'great-circle-distance' or 'as the crow flies'),
according to the 'Vincenty (ellipsoid)' method. This method uses an ellipsoid and the results are
very accurate. The method is computationally more intensive than the other great-circled methods
in this package.

Usage
distVincentyEllipsoid(p1, p2, a=6378137, b=6356752.3142, f=1/298.257223563)

Arguments
p1 longitude/latitude of point(s), in degrees; can be a vector of two numbers, a
matrix of 2 columns (first one is longitude, second is latitude) or a SpatialPoints* object
p2 as above
a Equatorial axis of ellipsoid
b Polar axis of ellipsoid
f Inverse flattening of ellipsoid

Details
The WGS84 ellipsoid is used by default. It is the best available global ellipsoid, but for some areas
other ellipsoids could be preferable, or even necessary if you work with a printed map that refers to
that ellipsoid. Here are parameters for some commonly used ellipsoids:
ellipsoid a b f
WGS84 6378137 6356752.3142 1/298.25723563
GRS80 6378137 6356752.3141 1/298.257222101
GRS67 6378160 6356774.719 1/298.25
Airy 1830 6377563.396 6356256.909 1/299.3249646
Bessel 1841 6377397.155 6356078.965 1/299.1528434
Clarke 1880 6378249.145 6356514.86955 1/293.465
Clarke 1866 6378206.4 6356583.8 1/294.9786982
International 1924 6378388 6356911.946 1/297
Krasovsky 1940 6378245 6356863 1/298.2997381

a is the ‘semi-major axis’, and b is the ‘semi-minor axis’ of the ellipsoid. f is the flattening. Note that f = (a−b)/a

more info: http://en.wikipedia.org/wiki/Reference_ellipsoid

Value

Distance value in the same units as the ellipsoid (default is meters)

Author(s)

Chris Veness and Robert Hijmans

References


http://www.movable-type.co.uk/scripts/latlong-vincenty.html

http://en.wikipedia.org/wiki/Great_circle_distance

See Also

distVincentySphere, distHaversine, distCosine, distMeeus

Examples

distVincentyEllipsoid(c(0,0),c(90,90))
# on a 'Clarke 1880' ellipsoid
distVincentyEllipsoid(c(0,0),c(90,90), a=6378249.145, b=6356514.86955, f=1/293.465)
distVincentySphere

‘Vincenty’ (sphere) great circle distance

Description

The shortest distance between two points (i.e., the ‘great-circle-distance’ or ‘as the crow flies’), according to the ‘Vincenty (sphere)’ method. This method assumes a spherical earth, ignoring ellipsoidal effects and it is less accurate than the distVincentyEllipsoid method.

Usage

distVincentySphere(p1, p2, r=6378137)

Arguments

- **p1**: longitude/latitude of point(s). Can be a vector of two numbers, a matrix of 2 columns (first one is longitude, second is latitude) or a SpatialPoints* object
- **p2**: as above
- **r**: radius of the earth; default = 6378137 m

Value

Distance value in the same unit as r (default is meters)

Author(s)

Robert Hijmans

References

http://en.wikipedia.org/wiki/Great_circle_distance

See Also

distVincentyEllipsoid, distHaversine, distCosine, distMeeus

Examples

distVincentySphere(c(0,0),c(90,90))
finalBearing  

**Final direction**

**Description**

Get the final direction (bearing) when arriving at p2 after starting from p1 and following the shortest path (a great circle).

**Usage**

```r
finalBearing(p1, p2)
```

**Arguments**

- `p1`  
  longitude/latitude of point(s). Can be a vector of two numbers, a matrix of 2 columns (first one is longitude, second is latitude) or a SpatialPoints* object
- `p2`  
  as above

**Value**

A direction (bearing) in degrees

**Author(s)**

Robert Hijmans

**References**

- http://www.movable-type.co.uk/scripts/latlong.html
- http://williams.best.vwh.net/ftp/avsig/avform.txt

**See Also**

- `bearing`

**Examples**

```r
bearing(c(10, 10), c(20, 20))
finalBearing(c(10, 10), c(20, 20))
```
gcIntermediate

Intermediate points on a great circle

Description

Get intermediate points on a great circle inbetween the two points used to define the great circle.

Usage

gcIntermediate(p1, p2, n=50, breakAtDateLine=FALSE, addStartEnd=FALSE, sp=FALSE, sepNA)

Arguments

- p1: Longitude/latitude of a single point, in degrees; can be a vector of two numbers, a matrix of 2 columns (first one is longitude, second is latitude) or a SpatialPoints* object
- p2: As above
- n: The requested number of points on the Great Circle
- breakAtDateLine: Logical. Return two matrices if the dateline is crossed?
- addStartEnd: Logical. Add p1 and p2 to the result?
- sp: Logical. Return a SpatialLines object?
- sepNA: Logical. Rather than as a list, return the values as a two column matrix with lines seperated by a row of NA values? (for use in 'plot')

Value

matrix or list

Author(s)

Robert Hijmans based on code by Ed Williams

References

http://williams.best.vwh.net/avform.htm#Intermediate

Examples

gcIntermediate(c(5,52), c(-120,37), n=6, addStartEnd=TRUE)
gcIntersect

Intersections of two great circles

Description

Get the two points where two great circles cross each other. Great circles are defined by two points on it.

Usage

gcIntersect(p1, p2, p3, p4)

Arguments

- **p1**: Longitude/latitude of a single point, in degrees; can be a vector of two numbers, a matrix of 2 columns (first one is longitude, second is latitude) or a SpatialPoints* object
- **p2**: As above
- **p3**: As above
- **p4**: As above

Value

two points for each pair of great circles

Author(s)

Robert Hijmans, based on equations by Ed Williams (see reference)

References

http://williams.best.vwh.net/intersect.htm

See Also

gcIntersectBearing

Examples

p1 <- c(5, 52); p2 <- c(-120, 37); p3 <- c(-60, 0); p4 <- c(0, 70)
gcIntersect(p1, p2, p3, p4)
**gcIntersectBearing**  
*Intersections of two great circles*

**Description**
Get the two points where two great circles cross each other. In this function, great circles are defined by a point and an initial bearing. In function `gcIntersect` they are defined by two sets of points.

**Usage**
```r
gcIntersectBearing(p1, brng1, p2, brng2)
```

**Arguments**
- `p1`: longitude/latitude of point(s). Can be a vector of two numbers, a matrix of 2 columns (first one is longitude, second is latitude) or a SpatialPoints* object
- `brng1`: Bearing from `p1`
- `p2`: As above. Should have same length as `p1`, or a single point (or vice versa when `p1` is a single point)
- `brng2`: Bearing from `p2`

**Value**
a matrix with four columns (two points)

**Author(s)**
Chris Veness and Robert Hijmans based on code by Ed Williams

**References**
- [http://williams.best.vwh.net/avform.htm#Intersection](http://williams.best.vwh.net/avform.htm#Intersection)
- [http://www.movable-type.co.uk/scripts/latlong.html](http://www.movable-type.co.uk/scripts/latlong.html)

**See Also**
- `gcIntersect`

**Examples**
```r
gcIntersectBearing(c(10,0), 10, c(-10,0), 10)
```
**gcLat**

*Latitude on a Great Circle*

**Description**

Latitude at which a great circle crosses a longitude

**Usage**

`gcLat(p1, p2, lon)`

**Arguments**

- `p1`: Longitude/latitude of a single point, in degrees; can be a vector of two numbers, a matrix of 2 columns (first one is longitude, second is latitude) or a SpatialPoints* object
- `p2`: As above
- `lon`: Longitude

**Value**

A numeric (latitude)

**Author(s)**

Robert Hijmans based on a formula by Ed Williams

**References**

[http://williams.best.vwh.net/avform.htm#Int](http://williams.best.vwh.net/avform.htm#Int)

**See Also**

`gcLon`, `gcMaxLat`

**Examples**

`gcLat(c(5,52), c(-120,37), lon=-120)`
gcLon

Longitude on a Great Circle

Description

Longitudes at which a great circle crosses a latitude (parallel)

Usage

gcLon(p1, p2, lat)

Arguments

p1          longitude/latitude of point(s). Can be a vector of two numbers, a matrix of 2 columns (first one is longitude, second is latitude) or a SpatialPoints* object
p2          as above
lat         a latitude

Value

vector of two numbers (longitudes)

Author(s)

Robert Hijmans based on code by Ed Williams

References

http://williams.best.vwh.net/avform.htm#Intersection

See Also

gcLat, gcMaxLat

Examples

gcLon(c(5,52), c(-120,37), 40)
**Description**

What is the northern most point that will be reached when following a great circle? Computed with Clairaut's formula. The southern most point is the antipode of the northern-most point. This does not seem to be very precise; and you could use optimization instead to find this point (see examples).

**Usage**

gcMaxLat(p1, p2)

**Arguments**

- `p1`: longitude/latitude of point(s). Can be a vector of two numbers, a matrix of 2 columns (first one is longitude, second is latitude) or a SpatialPoints* object
- `p2`: as above

**Value**

A matrix with coordinates (longitude/latitude)

**Author(s)**

Ed Williams, Chris Veness, Robert Hijmans

**References**

http://williams.best.vwh.net/ftp/avsig/avform.txt
http://www.movable-type.co.uk/scripts/latlong.html

**See Also**

gclat, gclon

**Examples**

gcMaxLat(c(5, 52), c(-120, 37))

# this seems to be more accurate:
f <- function(lon){gclat(c(5, 52), c(-120, 37), lon)}
optimize(f, interval=c(-180, 180), maximum=TRUE)
geomean

Mean location of spherical coordinates

Description

Mean location for spherical (longitude/latitude) coordinates that deals with the angularity. I.e., the mean of longitudes -179 and 178 is 179.5

Usage

geomean(xy, w)

Arguments

xy matrix with two columns (longitude/latitude), or a SpatialPoints or SpatialPolygons object with a longitude/latitude CRS

w weights (vector of numeric values, with a length that is equal to the number of spatial features in x

Value

Coordinate pair (numeric)

Author(s)

Robert J. Hijmans

Examples

xy <- cbind(x=c(-179, 179, 177), y=c(12, 14, 16))
xy <- geomean(xy)

greatCircle

Great circle

Description

Get points on a great circle as defined by the shortest distance between two specified points

Usage

greatCircle(p1, p2, n=360, sp=FALSE)
greatCircleBearing

Arguments

- **p1**: longitude/latitude of point(s). Can be a vector of two numbers, a matrix of 2 columns (first one is longitude, second is latitude) or a SpatialPoints* object.
- **p2**: as above.
- **n**: The requested number of points on the Great Circle.
- **sp**: Logical. Return a SpatialLines object?

Value

A matrix of points, or a list of such matrices (e.g., if multiple bearings are supplied).

Author(s)

Robert Hijmans, based on a formula provided by Ed Williams.

References

http://williams.best.vwh.net/avform.htm

Examples

greatCircle(c(5,52), c(-120,37), n=36)

greatCircleBearing  Great circle

Description

Get points on a great circle as defined by a point and an initial bearing.

Usage

greatCircleBearing(p, brng, n=360)

Arguments

- **p**: longitude/latitude of a single point. Can be a vector of two numbers, a matrix of 2 columns (first one is longitude, second is latitude) or a SpatialPoints* object.
- **brng**: bearing.
- **n**: The requested number of points on the great circle.

Value

A matrix of points, or a list of matrices (e.g., if multiple bearings are supplied).
**horizon**

**Author(s)**
Robert Hijmans based on formulae by Ed Williams

**References**
http://williams.best.vwh.net/avform.htm#Int

**Examples**
greatCircleBearing(c(5,52), 45, n=12)

<table>
<thead>
<tr>
<th>horizon</th>
<th>Distance to the horizon</th>
</tr>
</thead>
</table>

**Description**
Empirical function to compute the distance to the horizon from a given altitude. The earth is assumed to be smooth, i.e. mountains and other obstacles are ignored.

**Usage**
horizon(h, r=6378137)

**Arguments**

- **h** altitude, numeric >= 0. Should have the same unit as r
- **r** radius of the earth; default value is 6378137 m

**Value**
Distance in units of h (default is meters)

**Author(s)**
Robert J. Hijmans

**References**
http://williams.best.vwh.net/avform.htm#Horizon

**Examples**

- horizon(1.80) # me
- horizon(324) # Eiffel tower
**makePoly**

*Add vertices to a polygon or line*

**Description**

Make a polygon or line by adding intermediate points (vertices) on the great circles inbetween the points supplied. This can be relevant when vertices are relatively far apart. It can make the shape of the object to be accurate, when plotted on a plane. `makePoly` will also close the polygon if needed.

**Usage**

```r
makePoly(p, interval=10000, r=6378137, sp=FALSE)
makeline(p, interval=10000, r=6378137, sp=FALSE)
```

**Arguments**

- `p`: a 2-column matrix (longitude/latitude) or a SpatialPolygons or SpatialLines object
- `interval`: maximum interval of points, in units of `r`
- `r`: radius of the earth; default = 6378137
- `sp`: Logical. If `TRUE`, a SpatialPolygons object is returned (depends on the `sp` package)

**Value**

A matrix

**Author(s)**

Robert J. Hijmans

**Examples**

```r
pol <- rbind(c(-180,-20), c(-160,5), c(-60, 0), c(-160,-60), c(-180,-20))
plot(pol)
lines(pol, col='red', lwd=3)
pol2 = makePoly(pol, interval=100000)
lines(pol2, col='blue', lwd=2)
```
### mercator

**Description**

Transform longitude/latitude points to the Mercator projection. The main purpose of this function is to compute centroids, and to illustrate rhumb lines in the vignette.

**Usage**

```r
mercator(p, inverse=FALSE, r=6378137)
```

**Arguments**

- `p`: longitude/latitude of point(s). Can be a vector of two numbers, a matrix of 2 columns (first one is longitude, second is latitude) or a SpatialPoints* object
- `inverse`: Logical. If TRUE, do the inverse projection (from Mercator to longitude/latitude
- `r`: Numeric. Radius of the earth; default = 6378137 m

**Value**

matrix

**Author(s)**

Robert Hijmans

**Examples**

```r
a <- mercator(c(5,52))
a
mercator(a, inverse=TRUE)
```

### midPoint

**Description**

Find the point half-way between two points along a great circle

**Usage**

```r
midPoint(p1, p2)
```
Arguments

p1  longitude/latitude of point(s). Can be a vector of two numbers, a matrix of 2 columns (first one is longitude, second is latitude) or a SpatialPoints* object

p2  As above

Value

A pair of coordinates (longitude/latitude)

Author(s)

Robert Hijmans, based on code by Chris Vennes

References

http://mathforum.org/library/drmath/view/51822.html
http://en.wikipedia.org/wiki/Great_circle_distance
http://www.movable-type.co.uk/scripts/latlong.html

Examples

midPoint(c(0,0),c(90,90))

---

onGreatCircle Is a point on a given great circle?

Description

Test if a point is on a great circle defined by two other points.

Usage

onGreatCircle(p1, p2, p3)

Arguments

p1  Longitude/latitude of the first point defining a great circle, in degrees; can be a vector of two numbers, a matrix of 2 columns (first one is longitude, second is latitude) or a SpatialPoints* object

p2  as above for the second point

p3  the point(s) to be tested if they are on the great circle or not

Value

logical
perimeter

Author(s)

Robert Hijmans

Examples

onGreatCircle(c(0, 0), c(30, 30), rbind(c(-10, -11.33812), c(10, 20)))

perimeter Compute the perimeter of a polygon

Description

Compute the perimeter of a polygon (or the length of a line) on a sphere

Usage

perimeter(x, ...)

Arguments

x Longitude/latitude of the points forming a polygon; Must be a matrix of 2 columns (first one is longitude, second is latitude) or a SpatialPolygons* orSpatialLines* object

... Additional arguments. One implemented r=6378137, the radius of the earth; default = 6378137 m

Value

Numeric. The perimeter or length. Unit is that of r (default is m).

Author(s)

Robert Hijmans

See Also

area, centroid

Examples

xy <- rbind(c(-180, -20), c(-140, 55), c(10, 0), c(-140, -60), c(-180, -20))

perimeter(xy)
plotArrows

Description

Plot polygons with arrow heads on each line segment, pointing towards the next vertex. This shows the direction of each line segment.

Usage

plotArrows(p, fraction=0.9, length=0.15, first=' ', add=FALSE, ...)

Arguments

p Polygons (either a 2 column matrix or data.frame; or a SpatialPolygons* object
fraction numeric between 0 and 1. When smaller then 1, interrupted lines are drawn
length length of the edges of the arrow head (in inches)
first Character to plot on first (and last) vertex
add Logical. If TRUE, the plot is added to an existing plot
... Additional arguments, see Details

Note

Based on an example in Software for Data Analysis by John Chambers (pp 250-251) but adjusted such that the line segments follow great circles between vertices.

Author(s)

Robert J. Hijmans

Examples

pol <- rbind(c(-180,-20), c(-160,5), c(-60, 0), c(-160,-60), c(-180,-20))
plotArrows(pol)
**randomCoordinates**

*Random or regularly distributed coordinates on the globe*

**Description**

`randomCoordinates` returns a 'uniform random sample' in the sense that the probability that a point is drawn from any region is equal to the area of that region divided by the area of the entire sphere. This would not happen if you took a random uniform sample of longitude and latitude, as the sample would be biased towards the poles.

`regularCoordinates` returns a set of coordinates that are regularly distributed on the globe.

**Usage**

```r
randomCoordinates(n)
regularCoordinates(N)
```

**Arguments**

- `n` Sample size (number of points (coordinate pairs))
- `N` Number of 'parts' in which the earth is subdivided

**Value**

Matrix of lon/lat coordinates

**Author(s)**

Robert Hijmans, based on code by Nils Haeck (`regularCoordinates`), [http://mathforum.org/kb/message.jspa?messageID=3985660&tstart=0](http://mathforum.org/kb/message.jspa?messageID=3985660&tstart=0)


**Examples**

```r
randomCoordinates(3)
regularCoordinates(1)
```
Description

Compute the approximate surface span of polygons in longitude and latitude direction. Span is computed by rasterizing the polygons; and precision increases with the number of 'scan lines'. You can either use a fixed number of scan lines for each polygon, or a fixed band-width.

Usage

span(x, ...)

Arguments

x a SpatialPolygons* object or a 2-column matrix (longitude/latitude)

... Additional arguments, see Details

Details

The following additional arguments can be passed, to replace default values for this function

nbands Character. Method to determine the number of bands to 'scan' the polygon. Either 'fixed' or 'variable'
n Integer >= 1. If nbands='fixed', how many bands should be used
res Numeric. If nbands='variable', what should the bandwidth be (in degrees)?
fun Logical. A function such as mean or min. Mean computes the average span
r Numeric. Radius of the earth; default=6378137m

Value

A list, or a matrix if a function fun is specified. Values are in the units of r (default is meter)

Author(s)

Robert J. Hijmans

Examples

pol <- rbind(c(-180,-20), c(-160,5), c(-60, 0), c(-160,-60), c(-180,-20))
plot(pol)
lines(pol)
# lon and lat span in m
span(pol, fun=max)
x <- span(pol)
max(x$latspan)
mean(x$latspan)
plot(x$longitude, x$lonspan)
\texttt{wrld} \hspace{1cm} \textit{World countries}

\textbf{Description}

world coastline and country outlines in longitude/latitude (wrld) and in Mercator projection (merc).

\textbf{Usage}

\begin{verbatim}
data(wrld) 
data(merc)
\end{verbatim}

\textbf{Source}

Derived from the \texttt{wrld\_simpl} data set in package \texttt{maptools}
Index

*Topic datasets
  wrld, 39
*Topic methods
  centroid, 8
gemean, 29
makePoly, 32
plotArrows, 36
span, 38
*Topic package
  geosphere-package, 2
*Topic spatial
  alongTrackDistance, 3
antipode, 4
areaPolygon, 5
bearing, 6
bearingRhumb, 7
centroid, 8
daylength, 9
destPoint, 10
destPointRhumb, 11
dist2gc, 12
dist2Line, 13
distCosine, 14
distHaversine, 15
distm, 16
distMeeus, 17
distRhumb, 18
distVincentyEllipsoid, 19
distVincentySphere, 21
finalBearing, 22
gcIntermediate, 23
gcIntersect, 24
gcIntersectBearing, 25
gcLat, 26
gcLon, 27
gcMaxLat, 28
gemean, 29
geosphere-package, 2
greatCircle, 29
greatCircleBearing, 30
orbit, 31
makePoly, 32
mercator, 33
midPoint, 33
onGreatCircle, 34
perimeter, 35
plotArrows, 36
randomCoordinates, 37
span, 38
alongTrackDistance, 3, 12, 13
antipodal (antipode), 4
antipode, 4, 28
area, 8, 35
areaPolygon, 5
areaPolygon, data.frame-method
  (areaPolygon), 5
areaPolygon, matrix-method
  (areaPolygon), 5
areaPolygon, SpatialPolygons-method
  (areaPolygon), 5
bearing, 6, 7, 22
bearingRhumb, 6, 7
centroid, 5, 8, 35
centroid, data.frame-method (centroid), 8
centroid, matrix-method (centroid), 8
centroid, SpatialPolygons-method
  (centroid), 8
daylength, 9
destPoint, 10, 11
destPointRhumb, 11
dist2gc, 3, 4, 12, 13
dist2Line, 12, 13
distCosine, 14, 15, 16, 18–21
distHaversine, 13, 14, 15, 16, 18–21
distm, 16
INDEX

distMeeus, 14, 15, 17, 20, 21
distRhumb, 7, 18
distVincentyEllipsoid, 14–16, 18, 19, 21
distVincentySphere, 14–16, 18–20, 21
finalBearing, 10, 22
gcIntermediate, 23
gcIntersect, 24, 25
gcIntersectBearing, 24, 25
gcLat, 26, 27, 28
gcLon, 26, 27, 28
gcMaxLat, 26, 27, 28
geomean, 29
geosphere (geosphere-package), 2
geosphere-package, 2
greatCircle, 29
greatCircleBearing, 30
horizon, 31
makeLine (makePoly), 32
makePoly, 32
merc (wrld), 39
mercator, 33
midPoint, 33

onGreatCircle, 34

perimeter, 5, 8, 35
perimeter, data.frame-method
(perimeter), 35
perimeter, matrix-method (perimeter), 35
perimeter, SpatialLines-method
(perimeter), 35
perimeter, SpatialPolygons-method
(perimeter), 35
plotArrows, 36

randomCoordinates, 37
regularCoordinates (randomCoordinates), 37

span, 38
span, matrix-method (span), 38
span, SpatialPolygons-method (span), 38

wrld, 39