Package ‘geosphere’

October 12, 2009

Type Package

Title Spherical Trigonometry

Version 0.2

Date 2009-10-12

Suggests sp

Author Robert J. Hijmans

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

Description Spherical trigonometry for geographic applications

License GPL

LazyLoad yes

R topics documented:

`geosphere-package` *Spherical Trigonometry*

Description

Spherical Trigonometry for geographic applications, such as great circle distance and distance along a rhumb line.

Details

Package:	geosphere
Type:	Package
Version:	1.0
Date:	2009-10-10
License:	GPL3
LazyLoad:	yes

Author(s)

Robert Hijmans, Chris Veness and Ed Williams

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

References

`http://williams.best.vwh.net/ftp/avsig/avform.txt`

`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 (p1) to the closest point on the path to a third point (p3), following a great circle path defined by points p1 and p2

Usage

```
alongTrackDistance(p1, p2, p3, 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. Should have same length as p1, or a single point (or vice versa when p1 is a single point
`p3`	as above
`r`	radius of the earth; default = 6378137m

Value

A distance in units of r (default is meters)

Author(s)

Chris Veness and Robert Hijmans

See Also

`alongTrackDistance`

Examples

```
alongTrackDistance(c(0,0),c(90,90),c(80,80))
```

`antipode` *Antipodes*

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)
isAntipodal(p1, p2)
```

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
`p1`	as above.
`p2`	as above. Should have same length as p1, or a single point (or vice versa when p1 is a single point

Value

antipodal points or a logical value

Author(s)

Robert Hijmans

References

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

Examples

```
antipode(rbind(c(5,52), c(-120,37), c(-60,0), c(0,70)))
isAntipodal(c(0,0), c(180,0))
```

`bearing` *Bearing*

Description

Get the initial bearing (direction to travel in) to go from point1 to point2 following the shortest path (a great circle). Note that bearings change continuously while traveling along a great circle. A route with constant bearing is a rhumb line (see `brngRhumb`.

Usage

```
bearing(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. Should have same length as p1, or a single point (or vice versa when p1 is a single point

Value

A bearing in degrees

Author(s)

Chris Veness; ported to R by Robert Hijmans;

References

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

http://williams.best.vwh.net/ftp/avsig/avform.txt

See Also

`brngRhumb`

Examples

```
bearing(c(0,0),c(90,90))
```

---

brngRhumb *Rhumbline bearing*

Description

Bearing (direction of travel) along a rhumb line. Unlike a great circle, a rhumb line is a line of constant bearing.

Usage

```
brngRhumb(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. Should have same length as p1, or a single point (or vice versa when p1 is a single point

Value

A bearing in degrees

Author(s)

Chris Veness; ported to R by Robert Hijmans

References

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

See Also

`bearing`, `bearing`

Examples

```
brngRhumb(c(0,0),c(90,90))
```

`crossingParallels` *Crossing parellels*

Description

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

Usage

```
crossingParallels(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. Should have same length as p1, or a single point (or vice versa when p1 is a single point
`lat`	a latitude

Value

two points (longitudes)

Author(s)

Robert Hijmans based on code by Ed Williams

References

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

Examples

```
crossingParallels(c(5,52), c(-120,37), 40)
```

`crossTrackDistance` *Cross Track Distance*

Description

The cross track distance (or cross track error) is the distance of a point from a great-circle path. The great circle path is defined by `p1` and `p2`, while `p3` is the point away from the path.

Usage

```
crossTrackDistance(p1, p2, p3, r=6378137)
```

Arguments

`p1`	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
`p2`	End of great circle path. As above. Should have same length as p1, or a single point (or vice versa when p1 is a single point
`p3`	Point away from the great cricle path. As for p2
`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 p3 is on. Positive means right of the course from p1 to p2, negative means left.

Author(s)

Chris Veness and Robert Hijmans

References

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

http://williams.best.vwh.net/ftp/avsig/avform.txt

See Also

`alongTrackDistance`

Examples

```
crossTrackDistance(c(0,0),c(90,90),c(80,80))
```

`destPoint` *Destination given bearing and distance, when following a great circle*

Description

Calculate the destination point travelling along a (shortest distance) great circle arc, given a start point, initial bearing, and distance.

Usage

```
destPoint(p, brng, 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
`brng`	bearing
`d`	distance
`r`	radius of the earth; default = 6378137 m

Value

A pair of coordinates (longitude/latitude)

Note

The bearing changes continuously when traveling along a great circle line. Therefore, thbe final bearing is not the same as the initial bearing. You can comute the final bearing with `finalBearing` (see examples, below)

Author(s)

Chris Veness; ported to R by Robert Hijmans

References

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

http://williams.best.vwh.net/ftp/avsig/avform.txt

Examples

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

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

`destPointRhumb` *Destination along a rhumb line*

Description

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

Usage

```
destPointRhumb(p, brng, dist, 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
`brng`	bearing in degrees
`dist`	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://www.movable-type.co.uk/scripts/latlong.html

See Also

`destPoint`

Examples

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

`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. Should have same length as p1, or a single point (or vice versa when p1 is a single point
`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

`distHaversine`, `distHaversine`, `distHaversine`

Examples

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

`distHaversine` *'Havesine' great circle distance*

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. Should have same length as p1, or a single point (or vice versa when p1 is a single point
`r`	radius of the earth; default = 6378137 m

Details

The Haversine ('half-versed-sine') formula 'remains particularly well-conditioned for numerical computation even at small distances' – unlike calculations based on the spherical law of cosines. It was published by r.W. Sinnott in 1984, although it has been known for much longer. When Sinnott devised the Haversine formula, computational precision was limited. Nowadays, computers can use 15 significant figures of precision. With this precision, the simple spherical law of cosines formula gives good results down to distances as small as around 1 meter. In view of this it is probably worth, in most situations, using either the simpler law of cosines or the more accurate ellipsoidal `distVincenty` formula in preference to haversine!

Since the earth is not quite a sphere, there are small errors in using spherical geometry; the earth is actually roughly ellipsoidal (or more precisely, oblate spheroidal) with a radius varying between about 6378km (equatorial) and 6357km (polar), and local radius of curvature varying from 6336km (equatorial meridian) to 6399 km (polar). This means that errors from assuming spherical geometry might be up to 0.55% crossing the equator, though generally below 0.3%, depending on latitude and direction of travel. An accuracy of better than 3m in 1km is often good enough, but if you want greater accuracy, you could use the `distVincenty` method for calculating geodesic distances on ellipsoids, which gives results accurate to within 1mm.

Value

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

Author(s)

Chris Veness and Robert Hijmans

References

Sinnott, R.W, 1984. Virtues of the Haversine. Sky and Telescope 68(2): 159

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

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

See Also

`distCosine`, `distCosine`, `distCosine`

Examples

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

`distRhumb` *Distance along a 'rhumb line'*

Description

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

Usage

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

Arguments

`p1`	longitude/latitude of point(s) 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. Should be of same length of p1, or a single point (or vice versa when p1 is a single point
`r`	radius of the earth; default = 6378137 m

Details

Sailors used to (and sometimes still) navigate along rhumb lines since it is easier to follow a constant compass bearing than to continually adjust the bearing as is needed to follow a great circle, though they are normally longer than great-circle (orthodrome) routes. Rhumb lines are straight lines on a Mercator Projection map. If you maintain a constant bearing along a rhumb line, you will gradually spiral in towards one of the poles.

Value

distance value in units of r (default=meters)

Author(s)

Chris Veness; ported to R by Robert Hijmans

References

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

See Also

distCosine, distCosine, distCosine

Examples

```
distRhumb(c(0,0),c(90,90))
```

---

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) 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. Should have same length as p1, or a single point (or vice versa when p1 is a single point
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	1/298.257223563
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

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

Vincenty, T. 1975. Direct and inverse solutions of geodesics on the ellipsoid with application of nested equations. Survey Review Vol. 23, No. 176, pp88-93. Available here: http://www.movable-type.co.uk/scripts/latlong-vincenty.html

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

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

See Also

`distVincentySphere`, `distVincentySphere`, `distVincentySphere`

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 then the `distVicentyEllipsoid` method.

Usage

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

Arguments

`p1`	longitude/latitude of point(s) 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. Should have same length as p1, or a single point (or vice versa when p1 is a single point
`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

distVincentySphere, distVincentySphere, distVincentySphere

Examples

```
distVincentySphere(c(0,0),c(90,90))
```

---

finalBearing *Final bearing*

Description

Get the final bearing when arriving at point2 after starting from p1 and following the shortest path (a great circle).

Usage

```
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. Should have same length as p1, or a single point (or vice versa when p1 is a single point

Value

A 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

```
bearing(c(0,0),c(90,90))
finalBearing(c(0,0),c(90,90))
```

`greatCircle` *Intersecting radials*

Description

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

Usage

```
greatCircle(p1, p2, n=360)
```

Arguments

`p1`	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
`p2`	As above
`n`	The requested number of points on the Great Circle

Value

a matrix of points

Author(s)

Robert Hijmans based on a formula by Ed Williams

References

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

Examples

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

greatCircleIntermediat

Intermediate points on a great circle

Description

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

Usage

```
greatCircleIntermediate(p1, p2, n=50)
```

Arguments

p1	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
p2	As above
n	The requested number of points on the Great Circle

Value

a matrix of points

Author(s)

Robert Hijmans based on code by Ed Williams

References

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

Examples

```
greatCircleIntermediate(c(5,52), c(-120,37), n=10)
```

greatCircleIntersect

Intersections of two great circles

Description

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

Usage

```
greatCircleIntersect(p1, p2, p3, p4)
```

Arguments

`p1`	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
`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

Examples

```
p1 <- c(5,52); p2 <- c(-120,37); p3 <- c(-60,0); p4 <- c(0,70)
greatCircleIntersect(p1,p2,p3,p4)
```

`midPoint` *Mid-point*

Description

Mid-point between two points along a great circle

Usage

```
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. Should be of same length of p1, or a single point (or vice versa when p1 is a single point

Details

Just as the initial bearing may vary from the final bearing, the midpoint may not be located half-way between latitudes/longitudes; the midpoint between 35N,45E and 35N,135E is around 45N,90E.

Value

A pair of coordinates (longitude/latitude)

Author(s)

Chris Veness; ported to R by Robert Hijmans

References

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

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

Examples

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

---

polePoint *Highest latitude on a great circle*

Description

Given a latitude and an initial bearing, what is the polar-most point that will be reached when following a great circle? Computed with Clairaut's formula.

Usage

```
polePoint(lat, brng)
```

Arguments

`lat`	latitude of point(s)
`brng`	bearing

Value

A pair of coordinates (longitude/latitude)

Author(s)

Chris Veness; ported to R by Robert Hijmans

References

http://williams.best.vwh.net/ftp/avsig/avform.txt

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

Examples

```
polePoint(c(5,52),30)
```

`radialIntersect` *Intersecting radials*

Description

Intersection between two lines defined by their point of origin and true bearing

Usage

```
radialIntersect(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`	True 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`	True bearing from p2

Value

a point

Author(s)

Robert Hijmans based on code by Ed Williams

References

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

Examples

```
radialIntersect(c(10,0), 10, c(-10,0), 10)
```

`utilities` *Internal utility functions*

Description

pointsToMatrix is a simple helper function. Point input to all other functions is processed by this function to coerce points into a two column matrix of longitude / latitude.

compareDim compares two or three points objects to check if they have the same length or have lenght 1 (point).

Usage

```
pointsToMatrix(p)
compareDim(p1,p2,p3)
```

Arguments

`p`	a vector of two numbers, a matrix of 2 columns (first one is longitude, second is latitude) or a spatialPoints* object
`p1`	a matrix of 2 columns
`p2`	a matrix of 2 columns
`p3`	a matrix of 2 columns

Value

pointsToMatrix: a matrix with two columns representing longitude and latitude.

compareDim: `TRUE` (invisible) or an error

Author(s)

Robert Hijmans and Jacob van Etten

Index