The ”geosphere” package (Version 1.1.2)

Robert J. Hijmans

April 18, 2010

1 Introduction

This (early draft) vignette describes the R package ’`geosphere`’. The package implements spherical trigonometry functions for geographic applications. Most functions have applications in navigation, but others are more general. For example, to estimate the location of an object at a given direction and distance.

There are a number of 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). There are also functions to compute intersections of great circles, and of rhumb lines, and daylength.

Geographic locations must be specified in latitude and longitude in degrees (and NOT in 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).

This is one of the first versions of this package, so please look out for bugs and let us know if you find any.

The functions in this package are mostly based on formulae provided by Ed Williams (`http://williams.best.vwh.net/ftp/avsig/avform.txt`, and partly on javascript implementations of these formulae by Chris Veness (`http://www.movable-type.co.uk/scripts/latlong.html`)

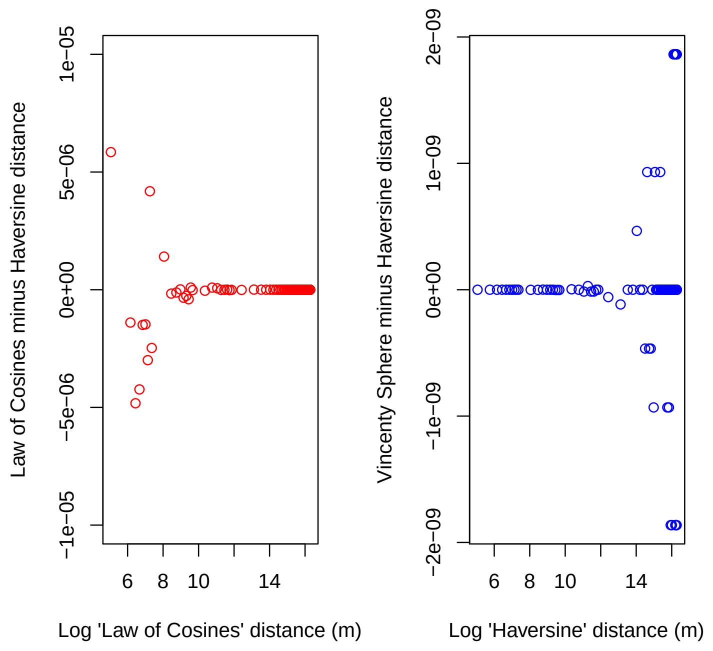
2 Great Circle Distance

There are four different functions to compute distance between two points. These are, in order of increasing complexity of the algorithm, the ’Spherical law of cosines’, ’Haversine’, ’Vincenty Sphere’ and ’Vincenty Ellipsoid’ methods. The first three assume the earth to be a sphere, while the ’Vincenty Ellipsoid’ (Vincenty, 1975) assumes it is an ellipsoid (which is closer to the truth).

The results from the first three functions are identical for practical purposes. 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.

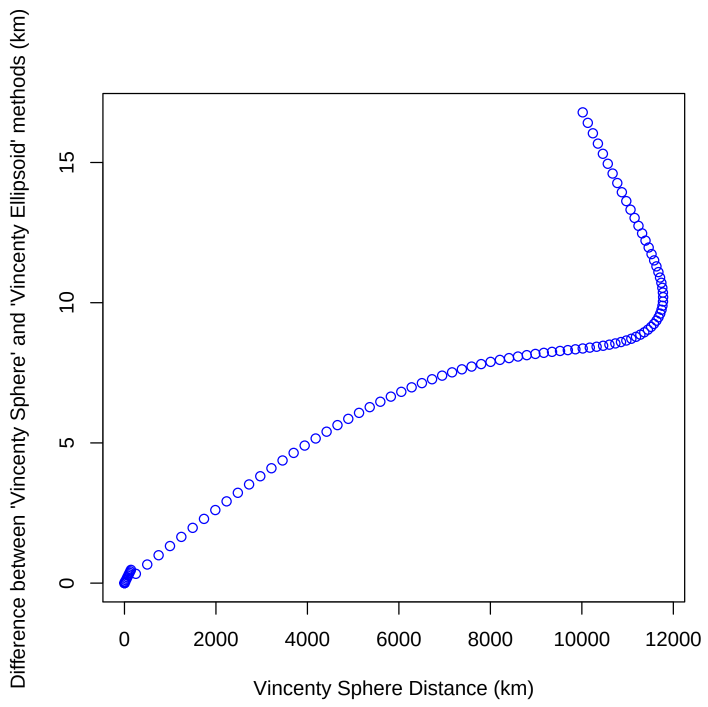
Below the differences between the three spherical methods are illustrated. At very short distances, there are small differences between the 'law of the Cosine' and the other two methods. There are even smaller differences between the 'Haversine' and 'Vincenty Sphere' methods at larger distances.

```
> library(geosphere)
> Lon = c(1:9/1000, 1:9/100, 1:9/10, 1:90*2)
> Lat = c(1:9/1000, 1:9/100, 1:9/10, 1:90)
> dcos = distCosine(c(0,0), cbind(Lon, Lat))
> dhav = distHaversine(c(0,0), cbind(Lon, Lat))
> dvsp = distVincentySphere(c(0,0), cbind(Lon, Lat))
> par(mfrow=(c(1,2)))
> plot(log(dcos), dcos-dhav, col='red', ylim=c(-1e-05, 1e-05),
+             xlab="Log 'Law of Cosines' distance (m)",
+             ylab="Law of Cosines minus Haversine distance")
> plot(log(dhav), dhav-dvsp, col='blue',
+             xlab="Log 'Haversine' distance (m)",
+             ylab="Vincenty Sphere minus Haversine distance")
```



The difference with the 'Vincenty Ellipsoid' method is more pronounced. In the example below (using the default WGS83 ellipsoid), the difference is about 0.3

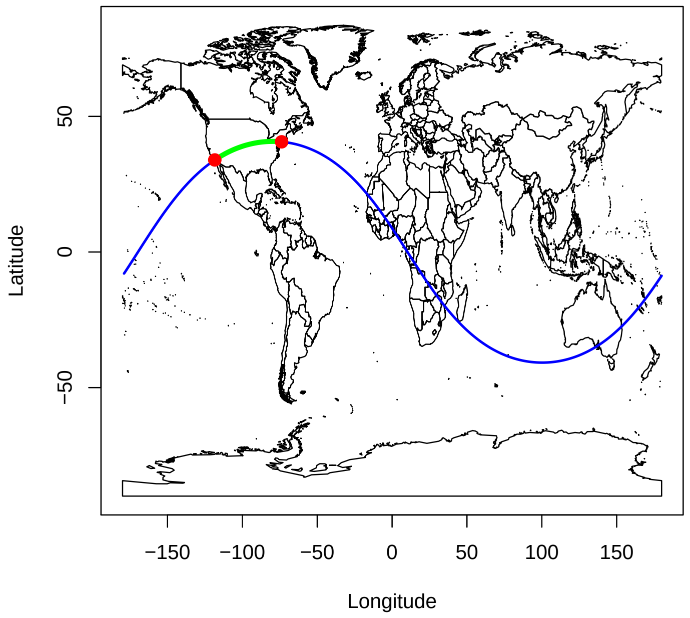
```
> dvse = distVincentyEllipsoid(c(0,0), cbind(Lon, Lat))
> plot(dvsp/1000, (dvsp-dvse)/1000, col='blue', xlab='Vincenty Sphere Distance (km)',
+         ylab="Difference between 'Vincenty Sphere' and 'Vincenty Ellipsoid' methods (km)")
```



3 Points on Great Circles

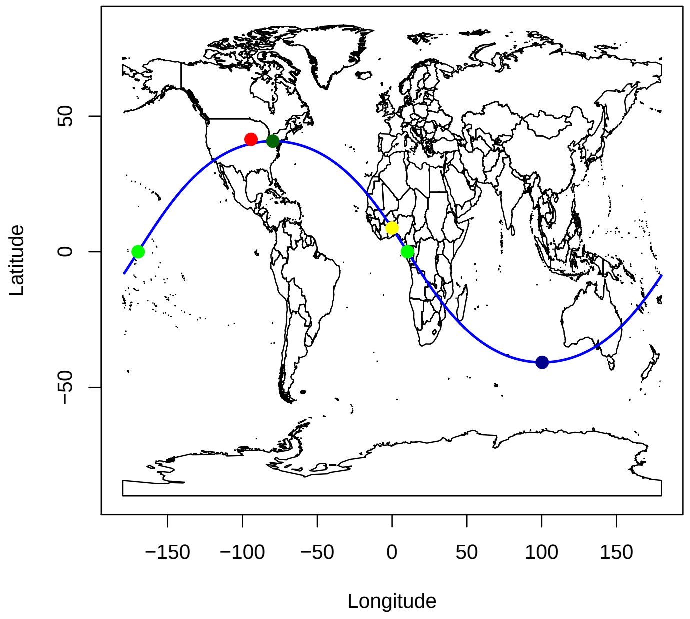
Great Circles can be defined with the function 'greatCircle', using two points on the Great Circle to define it, and an additional argument to indicate how many points on the great circle should be returned. gcIntermediate only returns points on the great circle that are on the track of shortest distance between the two points defining the great circle.

```
> LA <- c(-118.40, 33.95)
> NY <- c(-73.78,  40.63)
> data(wrld)
> plot(wrld, type='l')
> gc <- greatCircle(LA, NY)
> lines(gc, lwd=2, col='blue')
> gci <- gcIntermediate(LA, NY)
> lines(gci, lwd=4, col='green')
> points(rbind(LA, NY), col='red', pch=20, cex=2)
```



You can use the functions illustrated below to find out what the maximum latitude is that a great circle will reach; at what latitude it crosses a specified longitude; or at what longitude it crosses a specified latitude. From the map below it appears that Clairaut's formula, used in gcMaxLat is not very accurate. Through optimization, and using greatCircle, a more accurate value was found. The southern-most point is the antipode of the northern-most point.

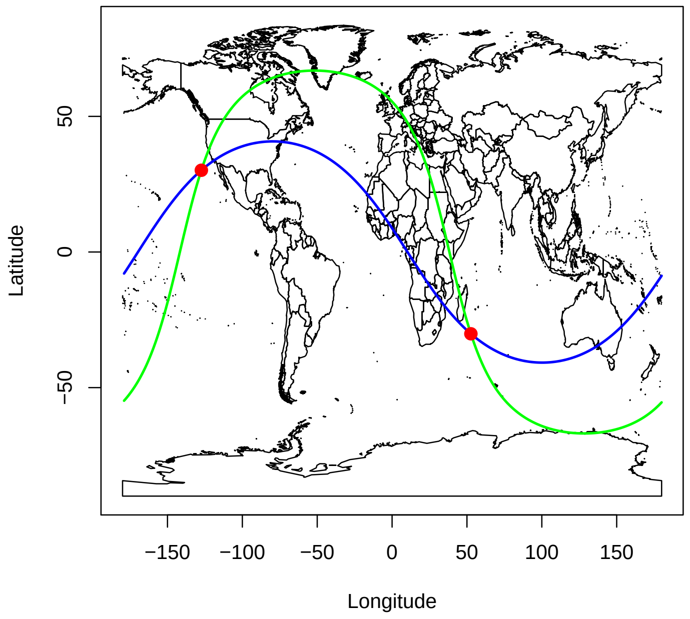
```
> ml <- gcMaxLat(LA, NY)
> lat0 <- gcLat(LA, NY, lon=0)
> lon0 <- gcLon(LA, NY, lat=0)
> plot(wrld, type='l')
> lines(gc, lwd=2, col='blue')
> points(ml, col='red', pch=20, cex=2)
> points(cbind(0, lat0), pch=20, cex=2, col='yellow')
> points(t(rbind(lon0, 0)), pch=20, cex=2, col='green' )
> f <- function(lon){gcLat(LA, NY, lon)}
> opt <- optimize(f, interval=c(-180, 180), maximum=TRUE)
> points(opt$maximum, opt$objective, pch=20, cex=2, col='dark green' )
> anti <- antipode(c(opt$maximum, opt$objective))
> points(anti, pch=20, cex=2, col='dark blue' )
```



Points of intersection of two Great Circles. Our second great circle is the one that connects San Francisco with Amsterdam. The two points where the Great Circles cross are antipodes. Antipodes are connected with an infinite number of Great Circles.

```
> SF <- c(-122.44, 37.74)
> AM <- c(4.75, 52.31)
> gc2 <- greatCircle(AM, SF)
> plot(wrld, type='l')
> lines(gc, lwd=2, col='blue')
> lines(gc2, lwd=2, col='green')
> int <- gcIntersect(LA, NY, SF, AM)
> points(rbind(int[,1:2], int[,3:4]), col='red', pch=20, cex=2)
> antipodal(int[,1:2], int[,3:4])
```

```
[1] TRUE
```



4 Great Circle Tracks

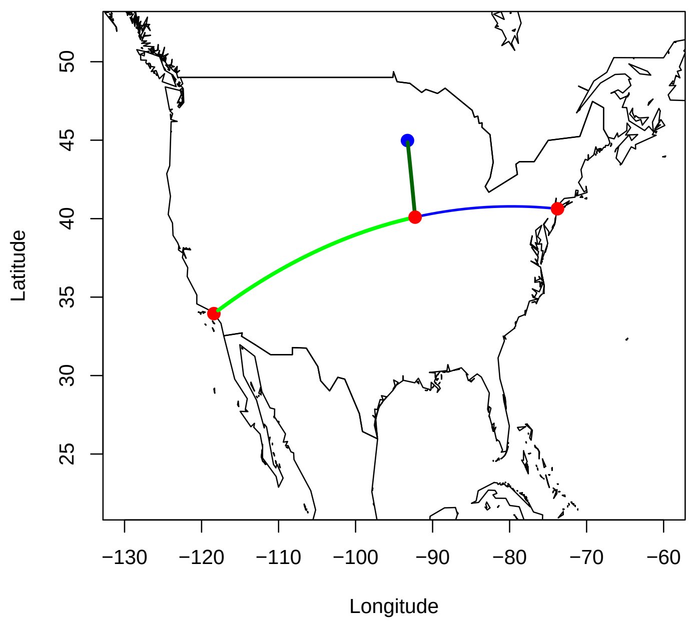
Below we first compute the distance and direction from Los Angeles (LA) to New York (NY). These are then used to compute the point from LA at that distance in that (initial) bearing (direction). Bearing changes continuously when traveling along a Great Circle. The final bearing, when approaching NY, is also given.

```
> d <- distCosine(LA, NY)
> d

     distance
[1,]  3977614

> b <- bearing(LA, NY)
> b

[1] 65.89757

> destPoint(LA, b, d)

    lon   lat
 -73.78 40.63
```

```
> NY

[1] -73.78  40.63

> finalBearing(LA, NY)

[1] 93.86472
```

What if we went off-course and were flying over Minneapolis (MS)? The closest point on the planned route (p) can be computed with the alongTrackDistance and destPoint functions. The distance from 'p' to MS can be computed with the crossTrackDistance function. The light green line represents the along-track distance, and the dark green line represents the cross-track distance.

```
> MS <- c(-93.26, 44.98)
> atd <- alongTrackDistance(LA, NY, MS)
> p <- destPoint(LA, b, atd)
> plot(wrld, type='l', xlim=c(-130,-60), ylim=c(22,52))
> lines(gci, col='blue', lwd=2)
> points(rbind(LA, NY), col='red', pch=20, cex=2)
> points(MS[1], MS[2], pch=20, col='blue', cex=2)
> lines(gcIntermediate(LA, p), col='green', lwd=3)
> lines(gcIntermediate(MS, p), col='dark green', lwd=3)
> points(p, pch=20, col='red', cex=2)
> crossTrackDistance(LA, NY, MS)

      distance
[1,] -549733.7

> distCosine(p, MS)

     distance
[1,] 549733.7
```

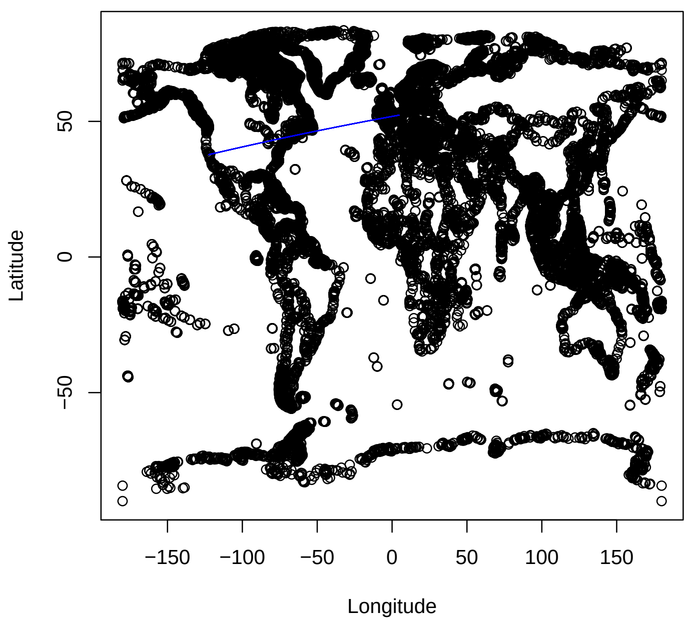


5 Rhumb lines

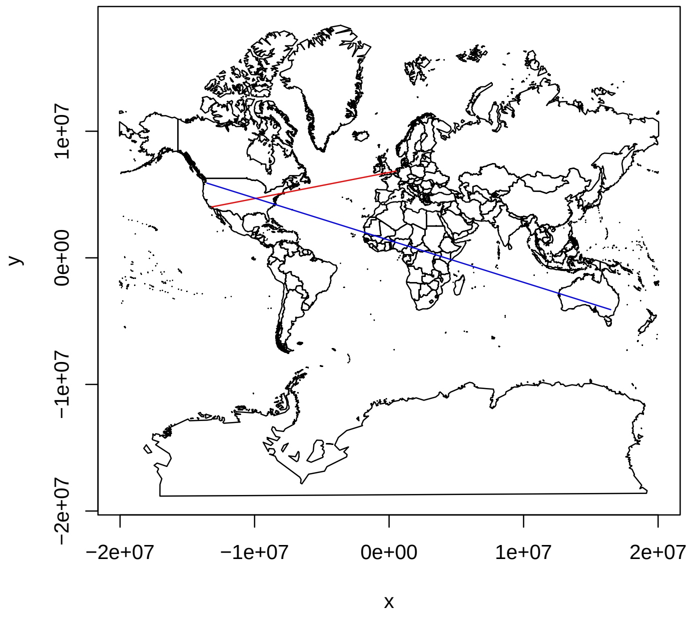
Rhumb (from the Spanish word for course, ’rumbo’) lines are straight lines on a Mercator projection map. They were used in navigation because it is easier to follow a constant compass bearing than to continually adjust direction as is needed to follow a great circle, even though rhumb lines are normally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

- to be continued -

```
> b = bearingRhumb(SF, AM)
> b

[1] 80.71958

> d = distRhumb(SF, AM)
> d

[1] 10057420

> p = destPointRhumb(SF, b, d=1:round(d/100) * 100)
> plot(wrld)
> lines(p, col='blue')
```



```
> VA <- c(-121.73, 46.78)
> SY <- c(148.04, -34.48)
> b1 <- bearingRhumb(LA, AM)
> b2 <- bearingRhumb(VA, SY)
> int <- rhumbIntersect(LA, b1, VA, b2)
> r1 <- mercator(rbind(LA, AM))
> r2 <- mercator(rbind(VA, SY))
> data(merc)
> plot(merc, type='l')
> lines(r1, col='red')
> lines(r2, col='blue')
```



6 References

Vincenty, T. 1975. Direct and inverse solutions of geodesics on the ellipsoid with application of nested equations. Survey Review 23(176): 88-93. Available here: `http://www.movable-type.co.uk/scripts/latlong-vincenty.html`