# Package 'geosed'

October 13, 2022

**Type** Package

**Title** Smallest Enclosing Disc for Latitude and Longitude Points

**Version** 0.1.1

**Date** 2019-08-17

**Author** Shant Sukljian <asshtnt@gmail.com>

**Maintainer** Shant Sukljian <asshtnt@gmail.com>

**Description**

Find the smallest circle that contains all longitude and latitude input points. From the generated center and radius, variable side polygons can be created, navigation based on bearing and distance can be applied, and more. Based on a modified version of Welzl's algorithm for smallest circle. Distance calculations are based on the haversine formula. Calculations for distance, midpoint, bearing and more are derived from <https://www.movable-type.co.uk>.

**Imports** grDevices

**License** MIT + file LICENSE

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 6.1.1

**Suggests** sp, mapview

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2019-09-03 07:50:02 UTC

## R topics documented:

geo_midpoint . . . . . 2
geo_move_point . . . . . 3
geo_points_bearing . . . . . 4
geo_point_dist . . . . . 5
geo_sed . . . . . 7
geo_surround_poly . . . . . 8
geo_trivial_circle . . . . . 9

**Index** **12**

---

`geo_midpoint` *Point of Equidistance to Up to Three Longitude, Latitude Points*

---

## Description

Generates a latitude and longitude point that is equidistant to up to three latitude and longitude points

## Usage

```
geo_midpoint(coordinate_matrix, alternative = FALSE)
```

## Arguments

| | |
|---|---|
| `coordinate_matrix` | A matrix of latitude and longitude columns and up to three rows |
| `alternative` | Whether to use alternative line creation method. Could be needed when nearly inverse angles cause intersections to be ambiguous. |

## Value

Returns a vector of length 2 containing a latitude and longitude point.

## Author(s)

Shant Sukljian

## See Also

`geo_sed` `geo_point_dist`

## Examples

```
# Load required packages
require(mapview)
require(sp)

# Create sample geo dataset
sample_coord <-
   matrix(
        c(
            sample(327131680:419648450, 3) / 10000000,
            sample(-1147301410:-1241938690, 3) / 10000000
        ),
        ncol = 2
    )
```

```
# Generate circumcenter and radius
gmp <- geo_midpoint(sample_coord)

# Find distance to circumcenter
radius <- geo_point_dist(rbind(sample_coord[1, ], gmp))

# Create 80 sided polygon based on gmp's center and radius
gmp_poly <- geo_surround_poly(gmp, radius, 80)

# Join all the points into a single matrix
bound_poly <- rbind(sample_coord, as.vector(gmp), gmp_poly)

# Create SpacialPoints object and pass to mapview for visualization
mapview(
    SpatialPoints(
        bound_poly[,c(2, 1)],
        proj4string = CRS("+proj=longlat +datum=WGS84")
    )
)
```

---

`geo_move_point` *New Latitude and Longitude Points from Point, Bearing and Distance*

---

## Description

Creates a new latitude, longitude point based on an origin point, bearing and distance

## Usage

```
geo_move_point(coordinates, bearing, distance)
```

## Arguments

| | |
|---|---|
| `coordinates` | A vector contaning one latitude and longitude point |
| `bearing` | The angle relative to north to move towards |
| `distance` | The distance in kilometers to move away from the origin point |

## Value

Returns a vector of length 2 containing a latitude and longitude point.

## Author(s)

Shant Sukljian

## See Also

`geo_sed` `geo_point_dist`

## Examples

```
# Load required packages
require(mapview)
require(sp)

# Create sample geo dataset
sample_coord <-
   matrix(
        c(
            sample(327131680:419648450, 1) / 10000000,
            sample(-1147301410:-1241938690, 1) / 10000000
        ),
        ncol = 2
    )

# Create new point
(gmp <- geo_move_point(sample_coord, sample(0:359, 1), 500))

# Join all the points into a single matrix
bound_poly <- rbind(sample_coord, gmp)

# Create SpacialPoints object and pass to mapview for visualization
mapview(
    SpatialPoints(
        bound_poly[,c(2, 1)],
        proj4string = CRS("+proj=longlat +datum=WGS84")
    )
)
```

---

`geo_points_bearing` *Bearing Between Two Latitude and Longitude Points*

---

## Description

Calculates the bearing angle in degrees between two latitude, longitude points

## Usage

```
geo_points_bearing(coordinate_pair)
```

## Arguments

`coordinate_pair`
: A matrix of latitude and longitude columns and two rows of points

## Value

A vector of length 1 containing a bearing

## Author(s)

Shant Sukljian

## See Also

geo_sed geo_point_dist

## Examples

```
# Load required packages
require(mapview)
require(sp)

# Create sample geo dataset
sample_coord <-
   matrix(
        c(
            sample(327131680:419648450, 2) / 10000000,
            sample(-1147301410:-1241938690, 2) / 10000000
        ),
        ncol = 2
    )

# Calculate bearing
(gpb <- geo_points_bearing(sample_coord))

# Create SpacialPoints object and pass to mapview for visualization
mapview(
    SpatialPoints(
        sample_coord[,c(2, 1)],
        proj4string = CRS("+proj=longlat +datum=WGS84")
    )
)
```

---

# geo_point_dist *Distance Between Two Latitude and Longitude Points*

---

## Description

Calculates the distance in kilometers between up to a combination of three latitude, longitude points

## Usage

```
geo_point_dist(coordinate_matrix, matrix = FALSE)
```

## Arguments

| | |
|---|---|
| coordinate_matrix | A matrix of latitude and longitude columns and up to three rows of points |
| matrix | Generates a matrix that shows/preseves the relationship between point combinations and the respective distance between them |

## Value

An input matrix with two rows returns a vector of length 1 containing the calculated distance. If the matrix argument is set to FALSE and an input matrix with three rows is given as the coordinate_matrix argument a vector of length 3 containing the calculated distances is returned. If the matrix argument is set to TRUE and an input matrix with three rows is given as the coordinate_matrix argument a 3 by 3 matrix of distances is returned.

## Author(s)

Shant Sukljian

## See Also

geo_sed geo_point_dist

## Examples

```
# Load required packages
require(mapview)
require(sp)

# Create sample geo dataset
sample_coord <-
   matrix(
        c(
            sample(327131680:419648450, 3) / 10000000,
            sample(-1147301410:-1241938690, 3) / 10000000
        ),
        ncol = 2
    )

# Calculate distances
(gpd <- geo_point_dist(sample_coord))

# Calculate distances and preserve relationship (Useful for three input points)
(gpd <- geo_point_dist(sample_coord, matrix = TRUE))

# Create SpacialPoints object and pass to mapview for visualization
mapview(
```

```
    SpatialPoints(
        sample_coord[,c(2, 1)],
        proj4string = CRS("+proj=longlat +datum=WGS84")
    )
)
```

---

geo_sed *Smallest circle encompassing all latitude and longitude points*

---

## Description

Generates a center point and radius that represent the smallest circle that contains all input points

## Usage

```
geo_sed(coordinate_matrix)
```

## Arguments

| | |
|---|---|
| coordinate_matrix | A matrix of latitude and longitude columns and any chosen number of rows to generate a smallest circle arround |

## Value

Returns a list of three elements named radius, center and making. Radius contains a single value representing the circle radius. Center contains a vector of length 2 representing the circle center latitude and longitude. Making contains a matrix of the latitude and longitude points that lie on the final smallest circle circumference.

## Author(s)

Shant Sukljian

## See Also

geo_trivial_circle geo_point_dist

## Examples

```
# Load required packages
require(mapview)
require(sp)

# Create sample geo dataset
sample_coord <-
```

```
   matrix(
        c(
            sample(327131680:419648450, 10) / 10000000,
            sample(-1147301410:-1241938690, 10) / 10000000
        ),
        ncol = 2
    )

# Generate sed center and radius
gsc <- geo_sed(sample_coord)

# Create 80 sided polygon based on gsc's center and radius
gsc_poly <- geo_surround_poly(gsc$center, gsc$radius, 80)

# Join all the points into a single matrix
bound_poly <- rbind(sample_coord, gsc$center, gsc_poly)

# Create SpacialPoints object and pass to mapview for visualization
mapview(
    SpatialPoints(
        bound_poly[,c(2, 1)],
        proj4string = CRS("+proj=longlat +datum=WGS84")
    )
)
```

---

`geo_surround_poly` *Geo Polygon*

---

## Description

Generates a collection of points that are equidistant to the center coordinates given and are distributed equally around the center

## Usage

```
geo_surround_poly(coordinates, distance, sides)
```

## Arguments

| | |
|---|---|
| `coordinates` | A vector of the center latitude and longitude point |
| `distance` | Distance to move away from center for each bearing |
| `sides` | Number of polygon sides |

## Value

Returns a matrix of latitude and longitude points.

## Author(s)

Shant Sukljian

## See Also

geo_sed geo_point_dist

## Examples

```
# Load required packages
library(mapview)
library(sp)

# Create sample geo dataset
sample_coord <-
   matrix(
        c(
            sample(327131680:419648450, 1) / 10000000,
            sample(-1147301410:-1241938690, 1) / 10000000
        ),
        ncol = 2
    )

# Create 80 sided polygon based on a random center and radius
geo_poly <- geo_surround_poly(sample_coord, sample(50:500, 1), 80)

# Join all the points into a single matrix
bound_poly <- rbind(sample_coord, geo_poly)

# Create SpacialPoints object and pass to mapview for visualization
mapview(
    SpatialPoints(
        bound_poly[,c(2, 1)],
        proj4string = CRS("+proj=longlat +datum=WGS84")
    )
)
```

---

# geo_trivial_circle *Circle encompassing up to three points*

---

## Description

Generates a center point and radius that represent the smallest circle that contains up to three input points

## Usage

```
geo_trivial_circle(coordinate_matrix, ...)
```

## Arguments

| | |
|---|---|
| coordinate_matrix | A matrix of latitude and longitude columns and up to three rows |
| ... | 'alternative' argument to be used when calling geo_midpoint |

## Value

Returns a list of three elements named radius, center and making. Radius contains a single value representing the circle radius. Center contains a vector of length 2 representing the circle center latitude and longitude. Making contains a matrix of the latitude and longitude points were used as the coordinate_matrix argument.

## Author(s)

Shant Sukljian

## See Also

geo_sed geo_point_dist

## Examples

```
# Load required packages
require(mapview)
require(sp)

# Create sample geo dataset
sample_coord <-
   matrix(
        c(
            sample(327131680:419648450, 3) / 10000000,
            sample(-1147301410:-1241938690, 3) / 10000000
        ),
        ncol = 2
    )

# Generate sed center and radius
gtc <- geo_trivial_circle(sample_coord)

# Create 80 sided polygon based on gtc's center and radius
gtc_poly <- geo_surround_poly(gtc$center, gtc$radius, 80)

# Join all the points into a single matrix
bound_poly <- rbind(sample_coord, gtc$center, gtc_poly)

# Create SpacialPoints object and pass to mapview for visualization
```

```
    mapview(
        SpatialPoints(
            bound_poly[,c(2, 1)],
            proj4string = CRS("+proj=longlat +datum=WGS84")
        )
    )
```

# Index


∗ **circle**
geo_midpoint, 2
geo_move_point, 3
geo_point_dist, 5
geo_points_bearing, 4
geo_sed, 7
geo_surround_poly, 8
geo_trivial_circle, 9
∗ **disk**
geo_midpoint, 2
geo_move_point, 3
geo_point_dist, 5
geo_points_bearing, 4
geo_sed, 7
geo_surround_poly, 8
geo_trivial_circle, 9
∗ **enclosing**
geo_midpoint, 2
geo_move_point, 3
geo_point_dist, 5
geo_points_bearing, 4
geo_sed, 7
geo_surround_poly, 8
geo_trivial_circle, 9
∗ **geo**
geo_midpoint, 2
geo_move_point, 3
geo_point_dist, 5
geo_points_bearing, 4
geo_sed, 7
geo_surround_poly, 8
geo_trivial_circle, 9
∗ **latitude**
geo_midpoint, 2
geo_move_point, 3
geo_point_dist, 5
geo_points_bearing, 4
geo_sed, 7
geo_surround_poly, 8
geo_trivial_circle, 9
∗ **longitude**
geo_midpoint, 2
geo_move_point, 3
geo_point_dist, 5
geo_points_bearing, 4
geo_sed, 7
geo_surround_poly, 8
geo_trivial_circle, 9
∗ **sed**
geo_midpoint, 2
geo_move_point, 3
geo_point_dist, 5
geo_points_bearing, 4
geo_sed, 7
geo_surround_poly, 8
geo_trivial_circle, 9
∗ **smallest**
geo_midpoint, 2
geo_move_point, 3
geo_point_dist, 5
geo_points_bearing, 4
geo_sed, 7
geo_surround_poly, 8
geo_trivial_circle, 9

geo_midpoint, 2, *10*
geo_move_point, 3
geo_point_dist, *2, 3*, *5*, 5, *6, 7*, *9, 10*
geo_points_bearing, 4
geo_sed, *2, 3*, *5, 6*, 7, *9, 10*
geo_surround_poly, 8
geo_trivial_circle, *7*, 9