Package: distanamo (via r-universe)

March 5, 2025

Title Create Distance Cartograms

Version 0.2.0

Description Creation of distance cartograms by deforming map layers to show the local deformations (computed using Waldo Tobler's bidimensional regression) between source points and image points.

License GPL (>= 3)

URL https://riatelab.github.io/distanamo/, https://github.com/riatelab/distanamo

BugReports https://github.com/riatelab/distanamo/issues/

Depends R (>= 3.6.0)

Imports sf, terra, units, graphics, grDevices

Suggests knitr, rmarkdown

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

SystemRequirements Cargo (Rust's package manager), rustc

VignetteBuilder knitr

Language en-US

Config/pak/sysreqs libgdal-dev gdal-bin libgeos-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev

Repository https://riatelab.r-universe.dev

RemoteUrl https://github.com/riatelab/distanamo

RemoteRef HEAD

RemoteSha 1a06dbd3be94e90b1050a7739687f98cd369152d

Contents

`dc_adjust` *Adjusts image points to source points*

Description

Computes the positions of the adjusted points by fitting image points to source points. Note that points generated by the `dc_generate_positions_from_durations` function and by the `dc_move_from_reference_point` do not need to be adjusted.

Usage

```
dc_adjust(source_points, image_points, adjustment_type = "euclidean")
```

Arguments

`source_points`	The source point layer, sf POINT object
`image_points`	The layer of point to be adjusted to fit `source_points`, sf POINT object
`adjustment_type`	The adjustment type to use, either "euclidean" or "affine"

Value

A list object with the transformation matrix, various metrics and the adjusted points

Examples

```
library(sf)

# Read source points
source_pts <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "prefecture", quiet = TRUE
)

# Read non adjusted image points
image_pts_not_adj <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "image-points-not-adjusted", quiet = TRUE
)

# Read the background layer to deform
background_layer <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "departement", quiet = TRUE
)

# Adjust image points to source points
adj_result <- dc_adjust(
  source_points = source_pts,
  image_points = image_pts_not_adj,
  "euclidean"
)

# Use adjusted points to create the interpolation grid
igrid <- dc_create(
  source_points = source_pts,
  image_points = adj_result$image_points,
  precision = 2,
  bbox = st_bbox(background_layer)
)

# Deform the target layer
background_deformed <- dc_interpolate(
  interpolation_grid = igrid,
  layer_to_deform = background_layer
)
```

---

dc_combine_bbox *Compute the bounding box that covers all the layers*

Description

Takes a list of sf objects and compute the bounding box that covers them all.

Usage

```
dc_combine_bbox(list_layers)
```

Arguments

list_layers	A list of sf objects

Value

An sf bounding box is returned.

Examples

```
library(sf)

# Read source points
source_pts <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "prefecture", quiet = TRUE
)

# Read the background layer to deform
background_layer <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "departement", quiet = TRUE
)

bbox <- dc_combine_bbox(list(source_pts, background_layer))
bbox
```

---

dc_concentric_circles *Create concentric circles around the reference point*

Description

Create concentric circles around the reference point, using the results of the dc_move_from_reference_point.

Usage

```
dc_concentric_circles(
  positioning_result,
  steps = list(30, 60, 90, 120, 150, 180, 210, 240, 270, 300, 330, 360)
)
```

Arguments

positioning_result	The object returned by the dc_move_from_reference_point function.
steps	The steps (in the unit of the durations used ine the dc_move_from_reference_point function) at which creating the circles.

Value

A sf LINESTRING object

Examples

```
library(sf)

# Read source points
source_pts <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "prefecture", quiet = TRUE
)

# Read the background layer to deform
background_layer <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "departement", quiet = TRUE
)

durations_mat <- read.csv(system.file("csv/mat.csv", package = "distanamo"), row.names = 1)
dur <- durations_mat["CAEN", ]

source_pts$durations <- as.double(dur)

ref_point <- subset(source_pts, source_pts$NOM_COM == "CAEN")
other_points <- subset(source_pts, !source_pts$NOM_COM == "CAEN")

# Generate position from durations between the reference point
# and the other points
positioning_result <- dc_move_from_reference_point(
  reference_point = ref_point,
  other_points = other_points,
  duration_col_name = "durations",
  factor = 1
)

# Create the interpolation grid
igrid <- dc_create(
  source_points = positioning_result$source_points,
  image_points = positioning_result$image_points,
  precision = 2.0,
  bbox = st_bbox(background_layer)
)

# Use the positioning result to create concentric circles
# every 60-minutes
circles <- dc_concentric_circles(
  positioning_result,
  c(60, 120, 180, 240, 300, 360, 420, 480, 540, 600, 660, 720)
)

# Deform the background layer
```

```
background_deformed <- dc_interpolate(
  interpolation_grid = igrid,
  layer_to_deform = background_layer
)
```

`dc_create` *Create an interpolation grid*

Description

Create a new interpolation grid which covers the source points and with a cell size deduced from the precision.

The grid is then interpolated to match the image points. This then allows one to interpolate any point on the grid (enabling the deformation of geometries such as background layers) and to retrieve useful metrics about the deformation.

If the bbox provided does not cover all the source points, the grid will be extended to cover all the source points.

The precision controls the size of the grid cells (higher is more precise, for example 0.5 generally gives a coarse result, 2 a satisfactory result and 4 a particularly fine result). A precision of 2 is usually a good default value.

The number of iterations controls the number of iterations for the interpolation. It is generally 4 times the square root of the number of points (and this is the default value it the niter parameter is not provided.

Note that the number of source points must be equal to the number of image points, and either they must be supplied in the same order (as they are homologous points), or the name of a field containing an identifier must be supplied to enable them to be sorted in the same order.

Usage

```
dc_create(source_points, image_points, precision, bbox, niter, sort_by)
```

Arguments

`source_points`	The source point layer, sf POINT object
`image_points`	The image point layer, sf POINT object
`precision`	The precision of the grid to be created (a higher value means a higher precision - 0.5 gives usually a coarse result, 2 is good default, 4 is very detailed)
`bbox`	The bounding box of the grid to be created
`niter`	The number of iterations (default is `floor(4 * sqrt(length(source_points)))`)
`sort_by`	The field to sort the source and image points by

Value

An interpolation grid to be used to transform the layers

Examples

```
library(sf)

# Read source points
source_pts <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "prefecture", quiet = TRUE
)

# Read image points
image_pts <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "image-points", quiet = TRUE
)

# Read the background layer to deform
background_layer <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "departement", quiet = TRUE
)

# Create the interpolation grid
igrid <- dc_create(
  source_points = source_pts,
  image_points = image_pts,
  precision = 2,
  bbox = st_bbox(background_layer)
)

# Plot various depictions of the interpolation grid
plot(igrid)

# Useful information about the interpolation grid
summary(igrid)

# Deform the target layer
background_deformed <- dc_interpolate(
  interpolation_grid = igrid,
  layer_to_deform = background_layer
)

plot(st_geometry(background_deformed))
```

`dc_generate_positions_from_durations`

Generate positions from durations matrix

Description

Generate positions from durations matrix, using PCoA to find the relative positions between points, then find the best fit between the source points and the image points using an affine or a Euclidean transformation.

Usage

```
dc_generate_positions_from_durations(
  durations,
  source_points,
  adjustment_type = "euclidean"
)
```

Arguments

durations	The durations matrix
source_points	The source points, an sf POINT object
adjustment_type	The adjustment type to use, either "euclidean" or "affine"

Value

A list object with the source points and the image points, ready to be used with the dc_create function

Examples

```
library(sf)

# Read source points
source_pts <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "prefecture", quiet = TRUE
)

# Read the background layer to deform
background_layer <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "departement", quiet = TRUE
)

# Read durations matrix
durations_mat <- read.csv(system.file("csv/mat.csv", package = "distanamo"), row.names = 1)

# Generate the positions from the whole duration matrix and
# adjust the result to the source points
pos_result <- dc_generate_positions_from_durations(
  durations = durations_mat,
  source_points = source_pts,
  adjustment_type = "euclidean"
```

```
)

# Display and plot useful information about the positioning step
plot(pos_result)
summary(pos_result)

# Use the result of the positioning to create the interpolation grid
igrid <- dc_create(
  source_points = pos_result$source_points,
  image_points = pos_result$image_points,
  precision = 2.0,
  bbox = st_bbox(background_layer)
)

# Deform the background layer
background_deformed <- dc_interpolate(
  interpolation_grid = igrid,
  layer_to_deform = background_layer
)
```

`dc_interpolate` *Interpolate a sf layer using the interpolation grid*

Description

Interpolate a sf layer using the interpolation grid.

Usage

```
dc_interpolate(interpolation_grid, layer_to_deform)
```

Arguments

`interpolation_grid`
: The interpolation grid

`layer_to_deform`
: The sf layer to interpolate

Value

The sf layer deformed by the interpolation grid

Examples

```
library(sf)

# Read source points
source_pts <- st_read(
```

```
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "prefecture", quiet = TRUE
)

# Read image points
image_pts <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "image-points", quiet = TRUE
)

# Read the background layer to deform
background_layer <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "departement", quiet = TRUE
)

# Create the interpolation grid
igrid <- dc_create(
  source_points = source_pts,
  image_points = image_pts,
  precision = 2,
  bbox = st_bbox(background_layer)
)

# Plot various depictions of the interpolation grid
plot(igrid)

# Useful information about the interpolation grid
summary(igrid)

# Deform the target layer
background_deformed <- dc_interpolate(
  interpolation_grid = igrid,
  layer_to_deform = background_layer
)

plot(st_geometry(background_deformed))
```

`dc_interpolate_parallel` *Deform a list of sf layers using the interpolation grid*

Description

Interpolate a list of sf layers using the interpolation grid.

Usage

```
dc_interpolate_parallel(interpolation_grid, layers_to_deform)
```

Arguments

`interpolation_grid`
: The interpolation grid

`layers_to_deform`
: A list of sf layers to interpolate

Value

The sf layers deformed by the interpolation grid

Examples

```
library(sf)

# Read source points
source_pts <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "prefecture", quiet = TRUE
)

# Read non adjusted image points
image_pts_not_adj <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "image-points-not-adjusted", quiet = TRUE
)

# Read the background layer to deform
background_layer <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "departement", quiet = TRUE
)

# Adjust image points to source points
adj_result <- dc_adjust(
  source_points = source_pts,
  image_points = image_pts_not_adj,
  "euclidean"
)

# Use adjusted points to create the interpolation grid
igrid <- dc_create(
  source_points = source_pts,
  image_points = adj_result$image_points,
  precision = 2,
  bbox = st_bbox(background_layer)
)

# Deform the target layer
background_deformed <- dc_interpolate_parallel(
  interpolation_grid = igrid,
  layers_to_deform = list(background_layer = background_layer, source_pts = source_pts)
)
```

dc_move_from_reference_point *Move points from a reference point*

Description

Move points from a reference point using durations between the reference point and all the other points.

Usage

```
dc_move_from_reference_point(
  reference_point,
  other_points,
  duration_col_name,
  factor
)
```

Arguments

reference_point	The point from which the other points will be moved, an sf POINT object
other_points	The other points to move, an sf POINT object
duration_col_name	The name of the column containing the durations in the other_points sf object
factor	The factor of displacement (default: 1)

Value

A list object with the source points and the image points, ready to be used with the dc_create function

Examples

```
library(sf)

# Read source points
source_pts <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "prefecture", quiet = TRUE
)

# Read the background layer to deform
background_layer <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "departement", quiet = TRUE
```

```
)

durations_mat <- read.csv(system.file("csv/mat.csv", package = "distanamo"), row.names = 1)
dur <- durations_mat["CAEN", ]

source_pts$durations <- as.double(dur)

ref_point <- subset(source_pts, source_pts$NOM_COM == "CAEN")
other_points <- subset(source_pts, !source_pts$NOM_COM == "CAEN")

# Generate position from durations between the reference point
# and the other points
positioning_result <- dc_move_from_reference_point(
  reference_point = ref_point,
  other_points = other_points,
  duration_col_name = "durations",
  factor = 1
)

# Display and plot useful information about the positioning step
plot(positioning_result)
summary(positioning_result)

# Create the interpolation grid
igrid <- dc_create(
  source_points = positioning_result$source_points,
  image_points = positioning_result$image_points,
  precision = 2.0,
  bbox = st_bbox(background_layer)
)

# Deform the background layer
background_deformed <- dc_interpolate(
  interpolation_grid = igrid,
  layer_to_deform = background_layer
)
```

---

distanamo *Create Distance Cartograms*

Description

This package enables the creation of what is often defined as a distance cartogram. Distance cartograms are a type of cartogram that deforms the layers of a map according to the distances between a set of source points and a set of image points. This is done by extending (by interpolation) to the layer(s) of the study area (territorial divisions, network...) the local displacement between the source coordinates and the image coordinates, derived from the distances between each pair of homologous points (source / image points).

Author(s)

Maintainer: Matthieu Viry <matthieu.viry@cnrs.fr> (ORCID)

Other contributors:

- Timothée Giraud <timothee.giraud@cnrs.fr> (ORCID) [contributor]

See Also

Useful links:

- https://riatelab.github.io/distanamo/
- https://github.com/riatelab/distanamo
- Report bugs at https://github.com/riatelab/distanamo/issues/

`plot.adjustment_result` *plot adjustment_result object*

Description

Plot the result of dc_adjust

Usage

```
## S3 method for class 'adjustment_result'
plot(x, ...)
```

Arguments

x	object of class adjustment_result
...	further specifications, see plot for details

Examples

```
library(sf)

# Read source points
source_pts <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "prefecture", quiet = TRUE
)

# Read non adjusted image points
image_pts_not_adj <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "image-points-not-adjusted", quiet = TRUE
)
```

```
# Read the background layer to deform
background_layer <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "departement", quiet = TRUE
)

# Adjust image points to source points
adj_result <- dc_adjust(
  source_points = source_pts,
  image_points = image_pts_not_adj,
  "euclidean"
)

# Plot result of the adjustment step
plot(adj_result)

# Summary statistics of the adjustment step
summary(adj_result)
```

`plot.interpolation_grid` *plot interpolation_grid object*

Description

Plot the interpolation grid, resulting from dc_create

Usage

```
## S3 method for class 'interpolation_grid'
plot(
  x,
  which = 1:4,
  ask = interactive(),
 caption = list("Source grid", "Interpolated grid", "Image to interpolated points",
    "Deformation strength"),
  ...
)
```

Arguments

`x`	object of class interpolation_grid
`which`	which plot to display, a subset of 1:4 (the default)
`ask`	logical; if TRUE, the user is asked before each plot
`caption`	captions to appear above the plots; character vector or list of valid graphics annotations
`...`	further specifications, see plot for details

Examples

```
library(sf)

# Read source points
source_pts <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "prefecture", quiet = TRUE
)

# Read image points
image_pts <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "image-points", quiet = TRUE
)

# Read the background layer to deform
background_layer <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "departement", quiet = TRUE
)

# Create the interpolation grid
igrid <- dc_create(
  source_points = source_pts,
  image_points = image_pts,
  precision = 2,
  bbox = st_bbox(background_layer)
)

# Plot various depictions of the interpolation grid
plot(igrid)

# Useful information about the interpolation grid
summary(igrid)
```

---

plot.multipolar_displacement_result

plot multipolar_displacement_result object

Description

Plot the result of dc_generate_positions_from_durations

Usage

```
## S3 method for class 'multipolar_displacement_result'
plot(x, ...)
```

Arguments

x	object of class multipolar_displacement_result
...	further specifications, see plot for details

Examples

```
library(sf)

# Read source points
source_pts <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "prefecture", quiet = TRUE
)

# Read the background layer to deform
background_layer <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "departement", quiet = TRUE
)

# Read durations matrix
durations_mat <- read.csv(system.file("csv/mat.csv", package = "distanamo"), row.names = 1)

# Generate the positions from the whole duration matrix and
# adjust the result to the source points
pos_result <- dc_generate_positions_from_durations(
  durations = durations_mat,
  source_points = source_pts,
  adjustment_type = "euclidean"
)

# Plot result of the positioning step
plot(pos_result)

# Summary statistics of the positioning step
summary(pos_result)
```

---

plot.unipolar_displacement_result

plot unipolar_displacement_result object

Description

Plot the result of dc_move_from_reference_point

Usage

```
## S3 method for class 'unipolar_displacement_result'
plot(x, ...)
```

Arguments

x	object of class unipolar_displacement_result
...	further specifications, see plot for details

Examples

```
library(sf)

# Read source points
source_pts <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "prefecture", quiet = TRUE
)

# Read the background layer to deform
background_layer <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "departement", quiet = TRUE
)

durations_mat <- read.csv(system.file("csv/mat.csv", package = "distanamo"), row.names = 1)
dur <- durations_mat["CAEN", ]

source_pts$durations <- as.double(dur)

ref_point <- subset(source_pts, source_pts$NOM_COM == "CAEN")
other_points <- subset(source_pts, !source_pts$NOM_COM == "CAEN")

# Generate position from durations between the reference point
# and the other points
positioning_result <- dc_move_from_reference_point(
  reference_point = ref_point,
  other_points = other_points,
  duration_col_name = "durations",
  factor = 1
)

# Plot result of the positioning step
plot(positioning_result)

# Summary statistics of the positioning step
summary(positioning_result)
```

`summary.adjustment_result`

summary adjustment_result object

Description

Compute summary statistics for adjustment_result

Usage

```
## S3 method for class 'adjustment_result'
summary(object, ...)
```

Arguments

object	object of class adjustment_result
...	further specifications, see summary for details

Examples

```
library(sf)

# Read source points
source_pts <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "prefecture", quiet = TRUE
)

# Read non adjusted image points
image_pts_not_adj <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "image-points-not-adjusted", quiet = TRUE
)

# Read the background layer to deform
background_layer <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "departement", quiet = TRUE
)

# Adjust image points to source points
adj_result <- dc_adjust(
  source_points = source_pts,
  image_points = image_pts_not_adj,
  "euclidean"
)

# Plot result of the adjustment step
plot(adj_result)

# Summary statistics of the adjustment step
summary(adj_result)
```

summary.interpolation_grid *summary interpolation_grid object*

Description

Compute summary statistics for interpolation_grid

Usage

```
## S3 method for class 'interpolation_grid'
summary(object, ...)
```

Arguments

object	object of class interpolation_grid
...	further specifications, see summary for details

Examples

```
library(sf)

# Read source points
source_pts <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "prefecture", quiet = TRUE
)

# Read image points
image_pts <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "image-points", quiet = TRUE
)

# Read the background layer to deform
background_layer <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "departement", quiet = TRUE
)

# Create the interpolation grid
igrid <- dc_create(
  source_points = source_pts,
  image_points = image_pts,
  precision = 2,
  bbox = st_bbox(background_layer)
)

# Plot various depictions of the interpolation grid
```

```
plot(igrid)

# Useful information about the interpolation grid
summary(igrid)
```

`summary.multipolar_displacement_result` *summary multipolar_displacement_result object*

Description

Compute summary statistics for multipolar_displacement_result

Usage

```
## S3 method for class 'multipolar_displacement_result'
summary(object, ...)
```

Arguments

`object`	object of class multipolar_displacement_result
`...`	further specifications, see summary for details

Examples

```
library(sf)

# Read source points
source_pts <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "prefecture", quiet = TRUE
)

# Read the background layer to deform
background_layer <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "departement", quiet = TRUE
)

# Read durations matrix
durations_mat <- read.csv(system.file("csv/mat.csv", package = "distanamo"), row.names = 1)

# Generate the positions from the whole duration matrix and
# adjust the result to the source points
pos_result <- dc_generate_positions_from_durations(
  durations = durations_mat,
  source_points = source_pts,
  adjustment_type = "euclidean"
```

```
)

# Plot result of the positioning step
plot(pos_result)

# Summary statistics of the positioning step
summary(pos_result)
```

`summary.unipolar_displacement_result` *summary unipolar_displacement_result object*

Description

Compute summary statistics for unipolar_displacement_result

Usage

```
## S3 method for class 'unipolar_displacement_result'
summary(object, ...)
```

Arguments

`object`	object of class unipolar_displacement_result
`...`	further specifications, see summary for details

Examples

```
library(sf)

# Read source points
source_pts <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "prefecture", quiet = TRUE
)

# Read the background layer to deform
background_layer <- st_read(
  dsn = system.file("gpkg/data-prefecture.gpkg", package = "distanamo"),
  layer = "departement", quiet = TRUE
)

durations_mat <- read.csv(system.file("csv/mat.csv", package = "distanamo"), row.names = 1)
dur <- durations_mat["CAEN", ]

source_pts$durations <- as.double(dur)

ref_point <- subset(source_pts, source_pts$NOM_COM == "CAEN")
```

```
other_points <- subset(source_pts, !source_pts$NOM_COM == "CAEN")

# Generate position from durations between the reference point
# and the other points
positioning_result <- dc_move_from_reference_point(
  reference_point = ref_point,
  other_points = other_points,
  duration_col_name = "durations",
  factor = 1
)

# Plot result of the positioning step
plot(positioning_result)

# Summary statistics of the positioning step
summary(positioning_result)
```

Index