Package: cartogram (via r-universe)

January 13, 2025

Title Create Cartograms with R

Version 0.4.0

Description Construct continuous and non-contiguous area cartograms.

License GPL-3

URL https://github.com/sjewo/cartogram,
https://sjewo.github.io/cartogram/

BugReports https://github.com/sjewo/cartogram/issues

Imports methods, packcircles, rlang, sf

Suggests future, future.apply, parallelly, progressr, testthat (>=
3.0.0)

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.2

Repository https://ar-puuk.r-universe.dev

RemoteUrl https://github.com/sjewo/cartogram

RemoteRef HEAD

RemoteSha 8f16a9b7a5b87dc706758042607ea1ea8a70c177

Contents

cartogram_cont *Calculate Contiguous Cartogram Boundaries*

Description

Construct a continuous area cartogram by a rubber sheet distortion algorithm (Dougenik et al. 1985)

Usage

```
cartogram_cont(
  x,
  weight,
  itermax = 15,
  maxSizeError = 1.0001,
  prepare = "adjust",
  threshold = "auto",
  verbose = FALSE,
  n_cpu = "respect_future_plan",
  show_progress = TRUE
)

## S3 method for class 'SpatialPolygonsDataFrame'
cartogram_cont(
  x,
  weight,
  itermax = 15,
  maxSizeError = 1.0001,
  prepare = "adjust",
  threshold = "auto",
  verbose = FALSE,
  n_cpu = "respect_future_plan",
  show_progress = TRUE
)

## S3 method for class 'sf'
cartogram_cont(
  x,
  weight,
  itermax = 15,
  maxSizeError = 1.0001,
  prepare = "adjust",
  threshold = "auto",
  verbose = FALSE,
  n_cpu = "respect_future_plan",
  show_progress = TRUE
)
```

Arguments

`x` a polygon or multiplogyon sf object

`weight` Name of the weighting variable in x

`itermax` Maximum iterations for the cartogram transformation, if maxSizeError ist not reached

`maxSizeError` Stop if meanSizeError is smaller than maxSizeError

`prepare` Weighting values are adjusted to reach convergence much earlier. Possible methods are "adjust", adjust values to restrict the mass vector to the quantiles defined by threshold and 1-threshold (default), "remove", remove features with values lower than quantile at threshold, "none", don't adjust weighting values

`threshold` "auto" or a threshold value between 0 and 1. With "auto", the value is 0.05 or, if the proportion of zeros in the weight is greater than 0.05, the value is adjusted accordingly.

`verbose` print meanSizeError on each iteration

`n_cpu` Number of cores to use. Defaults to "respect_future_plan". Available options are: * "respect_future_plan" - By default, the function will run on a single core, unless the user specifies the number of cores using `plan` (e.g. ‘future::plan(future::multisession, workers = 4)‘) before running the ‘cartogram_cont‘ function. * "auto" - Use all except available cores (identified with `availableCores`) except 1, to keep the system responsive. * a ‘numeric‘ value - Use the specified number of cores. In this case ‘cartogram_cont‘ will use set the specified number of cores internally with ‘future::plan(future::multisession, workers = n_cpu)‘ and revert that back by switching the plan back to whichever plan might have been set before by the user. If only 1 core is set, the function will not require ‘future‘ and ‘future.apply‘ and will run on a single core.

`show_progress` A ‘logical‘ value. If TRUE, show progress bar. Defaults to TRUE.

Value

An object of the same class as x

References

Dougenik, J. A., Chrisman, N. R., & Niemeyer, D. R. (1985). An Algorithm To Construct Continuous Area Cartograms. In The Professional Geographer, 37(1), 75-81.

Examples

```
# ========= Basic example =========
library(sf)
library(cartogram)

nc = st_read(system.file("shape/nc.shp", package="sf"), quiet = TRUE)

# transform to NAD83 / UTM zone 16N
nc_utm <- st_transform(nc, 26916)
```

```
# Create cartogram
nc_utm_carto <- cartogram_cont(nc_utm, weight = "BIR74", itermax = 5)

# Plot
par(mfrow=c(2,1))
plot(nc[,"BIR74"], main="original", key.pos = NULL, reset = FALSE)
plot(nc_utm_carto[,"BIR74"], main="distorted", key.pos = NULL, reset = FALSE)


# ========= Advanced example 1 =========
# Faster cartogram using multiple CPU cores
# using n_cpu parameter
library(sf)
library(cartogram)

nc = st_read(system.file("shape/nc.shp", package="sf"), quiet = TRUE)

# transform to NAD83 / UTM zone 16N
nc_utm <- st_transform(nc, 26916)

# Create cartogram using 2 CPU cores on local machine
nc_utm_carto <- cartogram_cont(nc_utm, weight = "BIR74", itermax = 5,
n_cpu = 2)

# Plot
par(mfrow=c(2,1))
plot(nc[,"BIR74"], main="original", key.pos = NULL, reset = FALSE)
plot(nc_utm_carto[,"BIR74"], main="distorted", key.pos = NULL, reset = FALSE)


# ========= Advanced example 2 =========
# Faster cartogram using multiple CPU cores
# using future package plan

library(sf)
library(cartogram)
library(future)

nc = st_read(system.file("shape/nc.shp", package="sf"), quiet = TRUE)

# transform to NAD83 / UTM zone 16N
nc_utm <- st_transform(nc, 26916)

# Set the future plan with 2 CPU local cores
# You can of course use any other plans, not just multisession
future::plan(future::multisession, workers = 2)

# Create cartogram with multiple CPU cores
# The cartogram_cont() will respect the plan set above
nc_utm_carto <- cartogram_cont(nc_utm, weight = "BIR74", itermax = 5)

# Shutdown the R processes that were created by the future plan
future::plan(future::sequential)
```

```
# Plot
par(mfrow=c(2,1))
plot(nc[,"BIR74"], main="original", key.pos = NULL, reset = FALSE)
plot(nc_utm_carto[,"BIR74"], main="distorted", key.pos = NULL, reset = FALSE)
```

`cartogram_dorling` *Calculate Non-Overlapping Circles Cartogram*

Description

Construct a cartogram which represents each geographic region as non-overlapping circles (Dorling 1996).

Usage

```
cartogram_dorling(x, weight, k = 5, m_weight = 1, itermax = 1000)

## S3 method for class 'sf'
cartogram_dorling(x, weight, k = 5, m_weight = 1, itermax = 1000)

## S3 method for class 'SpatialPolygonsDataFrame'
cartogram_dorling(x, weight, k = 5, m_weight = 1, itermax = 1000)
```

Arguments

`x`	a polygon or multiplogyon sf object
`weight`	Name of the weighting variable in x
`k`	Share of the bounding box of x filled by the larger circle
`m_weight`	Circles' movements weights. An optional vector of numeric weights (0 to 1 inclusive) to apply to the distance each circle moves during pair-repulsion. A weight of 0 prevents any movement. A weight of 1 gives the default movement distance. A single value can be supplied for uniform weights. A vector with length less than the number of circles will be silently extended by repeating the final value. Any values outside the range [0, 1] will be clamped to 0 or 1.
`itermax`	Maximum iterations for the cartogram transformation.

Value

Non overlaping proportional circles of the same class as x.

References

Dorling, D. (1996). Area Cartograms: Their Use and Creation. In Concepts and Techniques in Modern Geography (CATMOG), 59.

Examples

```
library(sf)
library(cartogram)

nc = st_read(system.file("shape/nc.shp", package="sf"), quiet = TRUE)

# transform to NAD83 / UTM zone 16N
nc_utm <- st_transform(nc, 26916)

# Create cartogram
nc_utm_carto <- cartogram_dorling(nc_utm, weight = "BIR74")

# Plot
par(mfrow=c(2,1))
plot(nc[,"BIR74"], main="original", key.pos = NULL, reset = FALSE)
plot(nc_utm_carto[,"BIR74"], main="distorted", key.pos = NULL, reset = FALSE)
```

cartogram_ncont *Calculate Non-Contiguous Cartogram Boundaries*

Description

Construct a non-contiguous area cartogram (Olson 1976).

Usage

```
cartogram_ncont(
  x,
  weight,
  k = 1,
  inplace = TRUE,
  n_cpu = "respect_future_plan",
  show_progress = TRUE
)

## S3 method for class 'SpatialPolygonsDataFrame'
cartogram_ncont(
  x,
  weight,
  k = 1,
  inplace = TRUE,
  n_cpu = "respect_future_plan",
  show_progress = TRUE
)

## S3 method for class 'sf'
cartogram_ncont(
  x,
  weight,
  k = 1,
  inplace = TRUE,
  n_cpu = "respect_future_plan",
  show_progress = TRUE
)
```

Arguments

x — a polygon or multiplogyon sf object

weight — Name of the weighting variable in x

k — Factor expansion for the unit with the greater value

inplace — If TRUE, each polygon is modified in its original place, if FALSE multi-polygons are centered on their initial centroid

n_cpu — Number of cores to use. Defaults to "respect_future_plan". Available options are: * "respect_future_plan" - By default, the function will run on a single core, unless the user specifies the number of cores using plan (e.g. ‘future::plan(future::multisession, workers = 4)‘) before running the ‘cartogram_ncont‘ function. * "auto" - Use all except available cores (identified with availableCores) except 1, to keep the system responsive. * a ‘numeric‘ value - Use the specified number of cores. In this case ‘cartogram_ncont‘ will use set the specified number of cores internally with ‘future::plan(future::multisession, workers = n_cpu)‘ and revert that back by switching the plan back to whichever plan might have been set before by the user. If only 1 core is set, the function will not require ‘future‘ and ‘future.apply‘ and will run on a single core.

show_progress — A ‘logical‘ value. If TRUE, show progress bar. Defaults to TRUE.

Value

An object of the same class as x with resized polygon boundaries

References

Olson, J. M. (1976). Noncontiguous Area Cartograms. In The Professional Geographer, 28(4), 371-380.

Examples

```
# ========= Basic example =========
library(sf)
library(cartogram)

nc = st_read(system.file("shape/nc.shp", package="sf"), quiet = TRUE)

# transform to NAD83 / UTM zone 16N
nc_utm <- st_transform(nc, 26916)

# Create cartogram
```

```
nc_utm_carto <- cartogram_ncont(nc_utm, weight = "BIR74")

# Plot
par(mfrow=c(2,1))
plot(nc[,"BIR74"], main="original", key.pos = NULL, reset = FALSE)
plot(st_geometry(nc_utm), main="distorted", reset = FALSE)
plot(nc_utm_carto[,"BIR74"], add =TRUE)


# ========= Advanced example 1 =========
# Faster cartogram using multiple CPU cores
# using n_cpu parameter
library(sf)
library(cartogram)

nc = st_read(system.file("shape/nc.shp", package="sf"), quiet = TRUE)

# transform to NAD83 / UTM zone 16N
nc_utm <- st_transform(nc, 26916)

# Create cartogram using 2 CPU cores on local machine
nc_utm_carto <- cartogram_ncont(nc_utm, weight = "BIR74", n_cpu = 2)

# Plot
par(mfrow=c(2,1))
plot(nc[,"BIR74"], main="original", key.pos = NULL, reset = FALSE)
plot(st_geometry(nc_utm), main="distorted", reset = FALSE)
plot(nc_utm_carto[,"BIR74"], add =TRUE)


# ========= Advanced example 2 =========
# Faster cartogram using multiple CPU cores
# using future package plan
library(sf)
library(cartogram)
library(future)

nc = st_read(system.file("shape/nc.shp", package="sf"), quiet = TRUE)

# transform to NAD83 / UTM zone 16N
nc_utm <- st_transform(nc, 26916)
# Set the future plan with 2 CPU local cores
# You can of course use any other plans, not just multisession
future::plan(future::multisession, workers = 2)

# Create cartogram with multiple CPU cores
# The cartogram_cont() will respect the plan set above
nc_utm_carto <- cartogram_ncont(nc_utm, weight = "BIR74")

# Shutdown the R processes that were created by the future plan
future::plan(future::sequential)

# Plot
```

```
par(mfrow=c(2,1))
plot(nc[,"BIR74"], main="original", key.pos = NULL, reset = FALSE)
plot(st_geometry(nc_utm), main="distorted", reset = FALSE)
plot(nc_utm_carto[,"BIR74"], add =TRUE)
```

Index