Package ‘redist’

May 15, 2021

Version 3.0.7
Date 2021-05-03
Title Simulation Methods for Legislative Redistricting
Maintainer Christopher T. Kenny <christopherkenny@fas.harvard.edu>
Description Enables researchers to sample redistricting plans from a pre-specified target distribution using Sequential Monte Carlo and Markov Chain Monte Carlo algorithms. The package allows for the implementation of various constraints in the redistricting process such as geographic compactness and population parity requirements. Tools for analysis such as computation of various summary statistics and plotting functionality are also included. The package implements methods described in Fifield, Higgins, Imai and Tarr (2020) <doi:10.1080/10618600.2020.1739532>, Fifield, Imai, Kawahara, and Kenny (2020) <doi:10.1080/2330443X.2020.1791773>, and McCartan and Imai (2020) <arXiv:2008.06131>.
Depends R (>= 3.5.0)
Imports Rcpp (>= 0.11.0), rlang, vctrs, tidyselect, stringr, dplyr (>= 1.0.0), sf, doParallel, foreach, servr, sys, coda, ggplo02, patchwork, readr
Suggests igraph, s2, lwgeom, withr, loo, Rmpi, knitr, rmarkdown, rmapshaper, scales, units, spdep, RSpectra, testthat (>= 3.0.0)
LinkingTo Rcpp, RcppArmadillo
License GPL (>= 2)
SystemRequirements OpenMPI, gmp, libxml2, python
NeedsCompilation yes
BugReports https://github.com/alarm-redist/redist/issues
URL https://alarm-redist.github.io/redist/
RoxygenNote 7.1.1
VignetteBuilder knitr
Encoding UTF-8
Config/testthat/edition 3
Author  Christopher T. Kenny [aut, cre],
        Cory McCartan [aut],
        Ben Fifield [aut],
        Kosuke Imai [aut],
        Jun Kawahara [ctb],
        Alexander Tarr [ctb],
        Michael Higgins [ctb]

Repository  CRAN

Date/Publication  2021-05-15 00:40:06 UTC

R topics documented:

redist-package ................................................. 4
add_reference ................................................. 5
as.matrix.redist ................................................. 5
classify_plans ................................................. 6
compare_plans .................................................. 6
competitiveness ................................................. 8
county_splits ................................................... 9
distr_compactness .............................................. 10
fl25 ............................................................. 13
fl250 ............................................................ 14
fl25_adj .......................................................... 15
fl25_enum ......................................................... 16
fl70 ............................................................... 17
flip_constraints_helper ........................................ 18
freeze ............................................................ 19
get_adj ............................................................ 20
get_existing ....................................................... 21
get_mh_acceptance_rate ....................................... 21
get_plans_matrix ............................................... 22
get_plans_weights ............................................. 22
get_pop_tol ....................................................... 23
get_sampling_info ............................................. 23
get_target ......................................................... 24
group_frac ........................................................ 24
imp_confint ....................................................... 25
iowa ............................................................... 26
is_contiguous .................................................... 27
is_county_split ................................................. 27
last_plan ........................................................ 28
make_cores ......................................................... 28
match_numbers ................................................... 30
merge_by ........................................................ 30
number_by ........................................................ 31
partisan_metrics ............................................... 32
plan_distances ................................................... 34
R topics documented:

plot.redist_classified ........................................... 36
plot.redist_map .................................................... 36
plot.redist_plans .................................................. 37
prec_assignment .................................................... 38
prec_cooccurrence .................................................. 38
print.redist ......................................................... 39
print.redist_classified ........................................... 39
print.redist_map ..................................................... 40
print.redist_plans .................................................. 40
pullback ............................................................. 41
redist.adjacency .................................................... 41
redist.calc.frontier.size .......................................... 42
redist.choropleth ................................................... 42
redist.coarsen.adjacency ........................................... 43
redist.combine ....................................................... 44
redist.combine.anneal .............................................. 46
redist.combine.mpi .................................................. 47
redist.constraint.helper ............................................ 49
redist.county.id ..................................................... 50
redist.county.relabel ............................................... 51
redist.crsdg ........................................................ 51
redist.diagplot ....................................................... 53
redist.enumerate .................................................... 55
redist.enumerpart .................................................... 56
redist.find.target ................................................... 58
redist.findparams ................................................... 59
redist.init.enumpart ............................................... 61
redist.ipw .......................................................... 62
redist.map .......................................................... 64
redist.mcmc .......................................................... 65
redist.mcmc.anneal ................................................. 72
redist.mcmc.mpi ..................................................... 76
redist.parity ........................................................ 80
redist.plot.adj ....................................................... 81
redist.plot.cores ..................................................... 82
redist.plot.distr_qtys .............................................. 83
redist.plot.hist ..................................................... 84
redist.plot.majmin ................................................... 85
redist.plot.map ...................................................... 85
redist.plot.penalty .................................................. 86
redist.plot.plans .................................................... 87
redist.plot.scatter .................................................. 88
redist.plot.varinfo .................................................. 89
redist.prep.enumpart .............................................. 89
redist.prep.polsbypopper ......................................... 90
redist.random.subgraph .......................................... 91
redist.read.enumpart .............................................. 91
redist.reduce.adjacency ............................................ 92
redist-package

Simulation Methods for Legislative Redistricting

Description


References


### add_reference

**Add a reference plan to a set of plans**

**Description**

This function facilitates comparing an existing (i.e., non-simulated) redistricting plan to a set of simulated plans.

**Usage**

```r
add_reference(plans, ref_plan, name = NULL)
```

**Arguments**

- `plans`: a `redist_plans` object
- `ref_plan`: an integer vector containing the reference plan. It will be renumbered to `1..ndists`.
- `name`: a human-readable name for the reference plan. Defaults to the name of `ref_plan`.

**Value**

a modified `redist_plans` object containing the reference plan

### as.matrix.redist

**Extract the redistricting matrix from a redist object**

**Description**

Extract the redistricting matrix from a `redist` object

**Usage**

```r
## S3 method for class 'redist'
as.matrix(x, ...)
```

**Arguments**

- `x`: redist object
- `...`: additional arguments

**Value**

matrix of district assignments
classify_plans  
Hierarchically classify a set of redistricting plans

Description

Applies hierarchical clustering to a distance matrix computed from a set of plans and takes the first k splits.

Usage

classify_plans(dist_mat, k = 8, method = "complete")

Arguments

dist_mat  
a distance matrix, the output of plan_distances()

k  
the number of groupings to create

method  
the clustering method to use. See hclust() for options.

Value

An object of class redist_classified, which is a list with two elements:

groups  
A character vector of group labels of the form "I.A.1.a.i", one for each plan.

splits  
A list of splits in the hierarchical clustering. Each list element is a list of two mutually exclusive vectors of plan indices, labeled by their group classification, indicating the plans on each side of the split.

Use plot.redist_classified() for a visual summary.

compare_plans  
Make a comparison between two sets of plans

Description

This function provides one way to identify the structural differences between two sets of redistricting plans. It operates by computing the precinct co-occurrence matrix (a symmetric matrix where the i,j-th entry is the fraction of plans where precinct i and j are in the same district) for each set, and then computing the first eigenvalue of the difference in these two matrices (in each direction). These eigenvalues identify the important parts of the map.
Usage

```r
compare_plans(
    plans,
    set1,
    set2,
    shp = NULL,
    plot = "fill",
    thresh = 0.1,
    labs = c("Set 1", "Set 2")
)
```

Arguments

- `plans`: a redist_plans object
- `set1`: indexing vectors for the plan draws to compare. Alternatively, a second redist_plans object to compare to.
- `set2`: indexing vectors for the plan draws to compare. Must be mutually exclusive with `set1`.
- `shp`: a shapefile for plotting.
- `plot`: If `plot="line"`, display a plot for each set showing the set of boundaries which most distinguish it from the other set (the squared differences in the eigenvector values across the boundary). If `plot="fill"`, plot the eigenvector for each set as a choropleth. See below for more information. Set to `FALSE` to disable plotting (or leave out `shp`).
- `thresh`: the value to threshold the eigenvector at in determining the relevant set of precincts for comparison.
- `labs`: the names of the panels in the plot.

Details

The co-occurrence matrices are regularized with a $Beta(1/ndists, 1 - 1/ndists)$ prior, which is useful for when either `set1` or `set2` is small.

Value

If possible, makes a comparison plot according to `plot`. Otherwise returns the following list:

- `eigen1`: A numeric vector containing the first eigenvector of $p_1 - p_2$, where $p_1$ and $p_2$ are the co-occurrence matrices for `set1` and `set2`, respectively.
- `eigen2`: A numeric vector containing the first eigenvector of $p_2 - p_1$, where $p_1$ and $p_2$ are the co-occurrence matrices for `set1` and `set2`, respectively.
- `group_1a`, `group_1b`: Lists of precincts. Compared to `set2`, in the `set1` plans these precincts were much more likely to be in separate districts. Computed by thresholding `eigen1` at `thresh`. 

group_2a, group_2b
Lists of precincts. Compared to set1, in the set2 plans these precincts were much more likely to be in separate districts. Computed by thresholding eigen2 at thresh.

coopcur_sep_1 The difference in the average co-occurrence of precincts in group_1a and group_1b between set2 and set1. Higher indicates better separation.

coopcur_sep_2 The difference in the average co-occurrence of precincts in group_2a and group_2b between set1 and set2. Higher indicates better separation.

Examples

data(iowa)
iowa_map = redist_map(iowa, ndists=4, pop_tol=0.05)
plans1 = redist_smc(iowa_map, 100, silent=TRUE)
plans2 = redist_mergesplit(iowa_map, 100, silent=TRUE)
compare_plans(plans1, plans2, shp=iowa_map)
compare_plans(plans2, as.integer(draw) <= 20,
  as.integer(draw) > 20, shp=iowa_map, plot="line")

competitiveness

Description
Currently only implements the competitiveness function in equation (5) of Cho & Liu 2016.

Usage

competitiveness(map, rvote, dvote, .data = cur_plans())

redist.competitiveness(plans, rvote, dvote)

Arguments

map a redist_map object
rvote A numeric vector with the Republican vote for each precinct.
dvote A numeric vector with the Democratic vote for each precinct.
.data a redist_plans object
plans A numeric vector (if only one map) or matrix with one row for each precinct and one column for each map. Required.

Value
Numeric vector with competitiveness scores
Examples

data(fl25)
data(fl25_enum)

plans_05 <- fl25_enum$plans[, fl25_enum$pop_dev <= 0.05]
comp <- redist.competitiveness(plans_05, fl25$mccain, fl25$obama)

county_splits

Description

Count County Splits

Usage

county_splits(map, counties, .data = cur_plans())

redist.splits(plans, counties, district_membership)

Arguments

map a redist_map object

counties A vector of county names or county ids.

.data a redist_plans object

plans A numeric vector (if only one map) or matrix with one row for each precinct and one column for each map. Required.

district_membership

Deprecated, use plans. A numeric vector (if only one map) or matrix with one row for each precinct and one column for each map. Required.

Value

integer vector with one number for each map
Calculate compactness measures for a set of plans

Description

redist.compactness is used to compute different compactness statistics for a shapefile. It currently computes the Polsby-Popper, Schwartzberg score, Length-Width Ratio, Convex Hull score, Reock score, Boyce Clark Index, Fryer Holden score, Edges Removed number, and the log of the Spanning Trees.

Usage

distr_compactness(map, measure = "FracKept", .data = cur_plans(), ...)

Arguments

map a redist_map object
.data a redist_plans object
... passed on to redist.compactness
shp A SpatialPolygonsDataFrame or sf object. Required unless "EdgesRemoved" and "logSpanningTree" with adjacency provided.
plans A numeric vector (if only one map) or matrix with one row for each precinct and one column for each map. Required.
This function computes specified compactness scores for a map. If there is more than one shape specified for a single district, it combines them, if necessary, and computes one score for each district.

Polsby-Popper is computed as

\[
\frac{4 \times \pi \times A(d)}{P(d)^2}
\]

where A is the area function, the district is d, and P is the perimeter function. All values are between 0 and 1, where larger values are more compact.
Schwartzberg is computed as

\[
P(d) \frac{2 \pi \sqrt{A(d) \pi}}{2 \pi \sqrt{A(d) \pi}}
\]

where \( A \) is the area function, the district is \( d \), and \( P \) is the perimeter function. All values are between 0 and 1, where larger values are more compact.

The Length Width ratio is computed as

\[
\frac{\text{length}}{\text{width}}
\]

where length is the shorter of the maximum x distance and the maximum y distance. Width is the longer of the two values. All values are between 0 and 1, where larger values are more compact.

The Convex Hull score is computed as

\[
\frac{A(d)}{A(CVH)}
\]

where \( A \) is the area function, \( d \) is the district, and \( CVH \) is the convex hull of the district. All values are between 0 and 1, where larger values are more compact.

The Reock score is computed as

\[
\frac{A(d)}{A(MBC)}
\]

where \( A \) is the area function, \( d \) is the district, and \( MBC \) is the minimum bounding circle of the district. All values are between 0 and 1, where larger values are more compact.

The Boyce Clark Index is computed as

\[
1 - \sum_{i=1}^{16} \left\{ \frac{r_i}{\sum_i r_i} \times 100 - 6.25 \right\} \frac{200}{200}
\]

where \( r_i \) are the distances of the 16 radii computed from the geometric centroid of the shape to the most outward point of the shape that intersects the radii, if the centroid is contained within the shape. If the centroid lies outside of the shape, a point on the surface is used, which will naturally incur a penalty to the score. All values are between 0 and 1, where larger values are more compact.

The Fryer Holden score for each district is computed with

\[
\text{Pop} \odot D(\text{precinct})^2
\]

where \( \text{Pop} \) is the population product matrix. Each element is the product of the ith and jth precinct's populations. \( D \) represents the distance, where the matrix is the distance between each precinct. To fully compute this index, for any map, the sum of these values should be used as the numerator. The denominator can be calculated from the full enumeration of districts as the smallest calculated numerator. This produces very large numbers, where smaller values are more compact.

The log spanning tree measure is the logarithm of the product of the number of spanning trees which can be drawn on each district.

The edges removed measure is number of edges removed from the underlying adjacency graph. A smaller number of edges removed is more compact.

The fraction kept measure is the fraction of edges that were not removed from the underlying adjacency graph. This takes values 0 - 1, where 1 is more compact.
A tibble with a column that specifies the district, a column for each specified measure, and a column that specifies the map number.

References


Examples

```r
data(f125)
data(f125_enum)

plans_05 <- f125_enum$plans[, f125_enum$pop_dev <= 0.05]

redist.compactness(shp = f125, plans = plans_05[,1:3],
measure = c('PolsbyPopper', 'EdgesRemoved'))
```

Description

This data set contains the 25-precinct shapefile and related data for each precinct. All possible partitions of the 25 precincts into three contiguous congressional districts are stored in `f125_enum`, and the corresponding adjacency graph is stored in `f125_adj`. This is generally useful for demonstrating basic algorithms locally.

Usage

```r
data("f125")
```
Format

sf data.frame containing columns for useful data related to the redistricting process, subsettled from real data in Florida, and sf geometry column.

- **geoId** Contains unique identifier for each precinct which can be matched to the full Florida dataset.
- **pop** Contains the population of each precinct.
- **vap** Contains the voting age population of each precinct.
- **obama** Contains the 2012 presidential vote for Obama.
- **mccain** Contains the 2012 presidential vote for McCain.
- **TotPop** Contains the population of each precinct. Identical to pop.
- **BlackPop** Contains the black population of each precinct.
- **HispPop** Contains the Hispanic population of each precinct.
- **VAP** Contains the voting age population of each precinct. Identical to vap.
- **BlackVAP** Contains the voting age population of black constituents of each precinct.
- **HispVAP** Contains the voting age population of hispanic constituents of each precinct.
- **geometry** Contains sf geometry of each precinct.

References


Examples

data(fl25)

---

**f1250**  
*Florida 250 Precinct Shape File*

Description

This data set contains the 250 Precinct shapefile and related data for each precinct.

Usage

data("f1250")
Format

sf data.frame containing columns for useful data related to the redistricting process, subsetted from real data in Florida, and sf geometry column.

- geoid: Contains unique identifier for each precinct which can be matched to the full Florida dataset.
- pop: Contains the population of each precinct.
- vap: Contains the voting age population of each precinct.
- obama: Contains the 2012 presidential vote for Obama.
- mccain: Contains the 2012 presidential vote for McCain.
- TotPop: Contains the population of each precinct. Identical to pop.
- BlackPop: Contains the black population of each precinct.
- HispPop: Contains the Hispanic population of each precinct.
- VAP: Contains the voting age population of each precinct. Identical to vap.
- BlackVAP: Contains the voting age population of black constituents of each precinct.
- HispVAP: Contains the voting age population of hispanic constituents of each precinct.
- geometry: Contains sf geometry of each precinct.

Details

It is a random 70 precinct connected subset from Florida’s precincts. This was introduced by [doi:10.1080/2330443X.2020.1791773]

References


Examples

data(f1250)

---

f125_adj  Florida 25 Precinct File

Description

This data set contains the 25-precinct shapefile and related data for each precinct. All possible partitions of the 25 precincts into three contiguous congressional districts are stored in f125_enum, and the corresponding adjacency graph is stored in f125_adj.

Format

A list storing the adjacency graph for the 25-precinct subset of Florida.
References


Examples

data(fl25_adj)

do.call(capture.output, c('source(system.file("extdata", "fl25.R", package = "redist"))'))

f25_enum

All Partitions of 25 Precincts into 3 Congressional Districts (No Population Constraint)

Description

This data set contains demographic and geographic information about 25 contiguous precincts in the state of Florida. The data lists all possible partitions of the 25 precincts into three contiguous congressional districts. The 25-precinct shapefile may be found in fl25.

Usage

data("f25_enum")

Format

A list with two entries:

- plans A matrix containing every partition of the 25 precincts into three contiguous congressional districts, with no population constraint.
- pop_dev A vector containing the maximum population deviation across the three districts for each plan.

References


Examples

data(f25_enum)
**Description**

This data set contains the 70 Precinct shapefile and related data for each precinct.

**Usage**

```r
data("fl70")
```

**Format**

- `sf` data.frame containing columns for useful data related to the redistricting process, subsetted from real data in Florida, and sf geometry column.
- `geoid` Contains unique identifier for each precinct which can be matched to the full Florida dataset.
- `pop` Contains the population of each precinct.
- `vap` Contains the voting age population of each precinct.
- `obama` Contains the 2012 presidential vote for Obama.
- `mccain` Contains the 2012 presidential vote for McCain.
- `TotPop` Contains the population of each precinct. Identical to `pop`.
- `BlackPop` Contains the black population of each precinct.
- `HispPop` Contains the Hispanic population of each precinct.
- `VAP` Contains the voting age population of each precinct. Identical to `vap`.
- `BlackVAP` Contains the voting age population of black constituents of each precinct.
- `HispVAP` Contains the voting age population of hispanic constituents of each precinct.
- `geometry` Contains sf geometry of each precinct.

**Details**

It is a random 70 precinct connected subset from Florida’s precincts. This was introduced by [doi:10.1080/2330443X.2020.1791773](https://doi.org/10.1080/2330443X.2020.1791773)

**References**


**Examples**

```r
data(fl70)
```
flip_constraints_helper

Flip Constraints Helper

Description

Creates a constraints list for use with redist_flip.

Usage

flip_constraints_helper(
  map,
  constraint = "compact",
  constraintweight = 0.6,
  init_plan = NULL,
  compactness_metric = "edges-removed",
  areas,
  borderlength_mat,
  ssdmat,
  ssd_denom,
  counties = NULL,
  partisan_metric = "efficiency-gap",
  rvote,
  dvote,
  group_pop = NULL,
  target_min = 0.55,
  target_other = 0.25,
  minorityprop
)

Arguments

map A redist_map object.
constraint character vector of constraints to use. Current accepted are "compact", "population", "countysplit", "hinge", "vra", "minority", "similarity", "partisan", and "segregation." Defaults to compact.
constraintweight corresponding weights to use with constraint. Weights must be nonzero if provided. Defaults to a weak compactness constraint
init_plan initial plan to use for the similarity constraint
compactness_metric character with "edges-removed", "polsby-popper", or "fryer-holden". Default is edges-removed.
areas areas to use with compact:polsby-popper. Computed from map if not provided and needed.
freeze

Freeze Parts of a Map

Description

Freeze Parts of a Map

Usage

freeze(freeze_row, plan, .data = cur_map())

redist.freeze(adj, freeze_row, plan = rep(1, length(adj)))

Examples

data(iowa)
iowa_map <- redist_map(iowa, existing_plan = cd_2010, total_pop = pop, pop_tol = 0.01)
cons <- flip_constraints_helper(
  map = iowa_map, constraint = c('compact', 'vra'),
  constraintweight = c(0.05, 10), target_min = 0.05,
  target_other = 0.01, group_pop = black
)

redist.flip(iowa_map, 10, constraints = cons)
get_adj

Arguments

freeze_row  Required, logical vector where TRUE freezes and FALSE lets a precinct stay free or a vector of indices to freeze
plan        A vector of district assignments, which if provided will create separate groups by district. Recommended. In freeze defaults to the existing plan, if one exists.
.data       a redist_map object
adj         Required, zero indexed adjacency list.

Value

integer vector to group by

Examples

library(redist)
library(dplyr)
data(fl25)
data(fl25_enum)
data(fl25_adj)
plan = fl25_enum$plans[, 5118]
freeze_id <- redist.freeze(adj = fl25_adj, freeze_row = (plan == 2), plan = plan)

data(iowa)
map <- redist_map(iowa, existing_plan = cd_2010, pop_tol = 0.02)
map <- map %>% merge_by(freeze(cd_2010 == 1, .data = .))

get_adj  Get and set the adjacency graph from a redist_map object

Description

Get and set the adjacency graph from a redist_map object

Usage

get_adj(x)

set_adj(x, adj)

Arguments

x         the redist_map object
adj       a new adjacency list.
**get_existing**

**Value**

- a zero-indexed adjacency list (get_adj)
- the modified redist_map object (set_adj)

---

**get_mh_acceptance_rate**

**Extract the Metropolis Hastings Acceptance Rate**

**Description**

Extract the Metropolis Hastings Acceptance Rate

**Usage**

get_mh_acceptance_rate(plans)

**Arguments**

- plans: the redist_plans object

**Value**

- a numeric acceptance rate
get_plans_matrix

Extract the matrix of district assignments from a redistricting simulation.

Description

Extract the matrix of district assignments from a redistricting simulation.

Usage

get_plans_matrix(x)

## S3 method for class 'redist_plans'
as.matrix(x, ...)

Arguments

x
the redist_plans object

... ignored

Value

matrix

get_plans_weights

Extract the sampling weights from a redistricting simulation.

Description

May be NULL if no weights exist (MCMC or optimization methods).

Usage

get_plans_weights(plans)

## S3 method for class 'redist_plans'
weights(object, ...)

Arguments

plans, object
the redist_plans object

... Ignored.
get_pop_tol

Value
A numeric vector of weights, with an additional attribute resampled indicating whether the plans have been resampled according to these weights.
numeric vector

Description
Get and set the population tolerance from a redist_map object

Usage
get_pop_tol(map)
set_pop_tol(map, pop_tol)

Arguments
map the redist_map object
pop_tol the population tolerance

Value
For get_pop_tol, a single numeric value, the population tolerance
For set_pop_tol, an updated redist_map object

get_sampling_info
Extract the sampling information from a redistricting simulation

Description
Extract the sampling information from a redistricting simulation

Usage
get_sampling_info(plans)

Arguments
plans the redist_plans object

Value
a list of parameters and information about the sampling problem.
get_target

Extract the target district population from a redist_map object

Description

Extract the target district population from a redist_map object

Usage

get_target(x)

Arguments

x the redist_map object

Value

a single numeric value, the target population

group_frac

Calculate Group Percent by District

Description

redist.group.percent computes the percentage that a group makes up in each district across a matrix of maps.

Usage

group_frac(
map,
  group_pop,
  total_pop = map[attr(map, "pop_col")],
  .data = cur_plans()
)

redist.group.percent(
  plans,
  group_pop,
  total_pop,
  ncores = 1,
  district_membership,
  grouppop,
  fullpop
)
imp_confint

Arguments

map a redist_map object

 group_pop A numeric vector with the population of the group for every precinct.

 total_pop A numeric vector with the population for every precinct.

.data a redist_plans object

 plans A matrix with one row for each precinct and one column for each map. Required.

 ncores Number of cores to use for parallel computing. Default is 1.

 district_membership Deprecated, use plans. A matrix with one row for each precinct and one column for each map. Required.

 group_pop Deprecated, use group_pop. A numeric vector with the population of the group for every precinct.

 fullpop Deprecated, use total_pop. A numeric vector with the population for every precinct.

Value

matrix with percent for each district

Examples

data(fl25)
data(fl25_enum)

 cd <- fl25_enum$plans[, fl25_enum$pop_dev <= 0.05]

 redist.group.percent(plans = cd,
 group_pop = fl25$BlackPop,
 total_pop = fl25$TotPop)

Description

Builds a confidence interval for the mean of a vector of interest, given importance sampling weights.

Usage

imp_confint(x, conf = 0.95, .data = cur_plans())
Arguments

- **x**: <data-masking> the vector to build importance sampling confidence intervals for.
- **conf**: The confidence level for the intervals.
- **.data**: a `redist_plans` object

Value

A tibble with three columns: `X`, `X_lower`, and `X_upper`, where `X` is the name of the vector of interest, containing the mean and confidence interval. When used inside `summarize()` this will create three columns in the output data.

---

**iowa**  
*Iowa County File*

Description

This data contains geographic and demographic information on the 99 counties of the state of Iowa.

Usage

```r
data("iowa")
```

Format

- sf tibble containing columns for useful data related to the redistricting process
- **fips**: The FIPS code for the county.
- **cd_2010**: The 2010 congressional district assignments.
- **pop**: The total population of the precinct, according to the 2010 Census.
- **white**: The non-Hispanic white population of the precinct.
- **black**: The non-Hispanic Black population of the precinct.
- **hisp**: The Hispanic population (of any race) of the precinct.
- **vap**: The voting-age population of the precinct.
- **wvap**: The white voting-age population of the precinct.
- **bvap**: The Black voting-age population of the precinct.
- **hvap**: The Hispanic voting-age population of the precinct.
- **tot_08**: Number of total votes for president in the county in 2008.
- **dem_08**: Number of votes for Barack Obama in 2008.
- **rep_08**: Number of votes for John McCain in 2008.
- **region**: The 28E agency regions for counties.
- **geometry**: The sf geometry column containing the geographic information.
**is_contiguous**

**Examples**

```r
data(iowa)
print(iowa)
```

---

**is_contiguous**  
*Check that a redist_map object is contiguous*

**Description**

Check that a redist_map object is contiguous

**Usage**

```r
is_contiguous(x)
```

**Arguments**

- `x` the object

**Value**

TRUE if contiguous.

---

**is_county_split**

*Identify which counties are split by a plan*

**Description**

Identify which counties are split by a plan

**Usage**

```r
is_county_split(plan, counties)
```

**Arguments**

- `plan` A vector of precinct/unit assignments
- `counties` A vector of county names or county ids.

**Value**

A logical vector which is TRUE for precincts belonging to counties which are split
last_plan

Extract the last plan from a set of plans

Description
Extract the last plan from a set of plans

Usage
```r
last_plan(plans)
```

Arguments
- `plans` A `redist_plans` object

Value
An integer vector containing the final plan assignment.

make_cores

Identify Cores of a District (Heuristic)

Description
Creates a grouping ID to unite geographies and perform analysis on a smaller set of precincts. It identifies all precincts more than boundary edges of a district district boundary. Each contiguous group of precincts more than boundary steps away from another district gets its own group. Some districts may have multiple, disconnected components that make up the core, but each of these is assigned a separate grouping id so that a call to `sf::st_union()` would produce only connected pieces.

Usage
```r
make_cores(.data = cur_map(), boundary = 1, focus = NULL)
```

```
redist.identify.cores(
  adj,
  plan,
  boundary = 1,
  focus = NULL,
  simplify = TRUE,
  adjacency,
  district_membership
)
```
make_cores

Arguments

- `.data` a `redist_map` object
- `boundary` Number of steps to check for. Defaults to 1.
- `focus` Optional. Integer. A single district to focus on.
- `adj` zero indexed adjacency list.
- `plan` An integer vector or matrix column of district assignments.
- `simplify` Optional. Logical. Whether to return extra information or just grouping ID.
- `adjacency` Deprecated, use `adj`. A zero indexed adjacency list.
- `district_membership` Deprecated, use `plan`. An integer vector or matrix column of district assignments.

Details

This is a loose interpretation of the NCSL's summary of redistricting criteria to preserve the cores of prior districts. Using the adjacency graph for a given plan, it will locate the precincts on the boundary of the district, within `boundary` steps of the edge. Each of these is given their own group. Each remaining entry that is not near the boundary of the district is given an id that can be used to group the remainder of the district by connected component. This portion is deemed the core of the district.

Value

integer vector (if `simplify` is false). Otherwise it returns a tibble with the grouping variable as `group_id` and additional information on connected components.

See Also

[redist.plot.cores()] for a plotting function

Examples

data(f1250)
adj <- redist.adjacency(f1250)
plan <- redist.smc(adj = adj, total_pop = f1250$pop,
  nsims = 1, ndists = 4)$plans
core <- redist.identify.cores(adj = adj, plan = plan)
redist.plot.cores(shp = f1250, plan = plan, core = core)
match_numbers

*Renumber districts to match an existing plan*

**Description**

District numbers in simulated plans are by and large random. This function attempts to renumber the districts across all simulated plans to match the numbers in a provided plan.

**Usage**

```r
match_numbers(data, plan, col = "pop_overlap", force = FALSE)
```

**Arguments**

- `data`: a `redist_plans` object
- `plan`: a character vector giving the name of the plan to match to (e.g., for a reference plan), or an integer vector containing the plan itself.
- `col`: the name of a new column to store the vector of population overlap with the reference plan: the fraction of the total population who are in the same district under each plan and the reference plan. Set to `NULL` if no column should be created.
- `force`: if `TRUE`, force computation when there are more than 1,000 renumbering options in any plan.

**Value**

a modified `redist_plans` object. New district numbers will be stored as an ordered factor variable in the `district` column. The district numbers in the plan matrix will match the levels of this factor.

merge_by

*Merge map units*

**Description**

In performing a county-level or cores-based analysis it is often necessary to merge several units together into a larger unit. This function performs this operation, modifying the adjacency graph as needed and attempting to properly aggregate other data columns.

**Usage**

```r
merge_by(.data, ..., by_existing = TRUE, drop_geom = TRUE, collapse_chr = TRUE)
```
Arguments

.data 

... 

by_existing

drop_geom

collapse_chr

Value

A merged redist_map object

Description

District numbers in simulated plans are by and large random. This function will renumber the districts across all simulated plans in order of a provided quantity of interest.

Usage

number_by(data, x, desc = F)

Arguments

data 

x

desc

Value

A merged redist_map object

A modified redist_plans object. New district numbers will be stored as an ordered factor variable in the district column. The district numbers in the plan matrix will match the levels of this factor.
partisan_metrics  Calculate gerrymandering metrics for a set of plans

Description

redist.metrics is used to compute different gerrymandering metrics for a set of maps.

Usage

partisan_metrics(map, measure, rvote, dvote, ..., .data = cur_plans())

redist.metrics(
  plans,
  measure = "DSeats",
  rvote,
  dvote,
  tau = 1,
  biasV = 0.5,
  respV = 0.5,
  bandwidth = 0.01,
  draw = 1,
  ncores = 1,
  district_membership,
  nloop
)

Arguments

map  a redist_map object
measure  A vector with a string for each measure desired from list "DSeats", "DVS", "EffGap", "EffGapEqPop", "TauGap", "MeanMedian", "Bias", "BiasV", "Declination", "Responsiveness", "LopsidedWins", "RankedMarginal", and "SmoothedSeat". Use "all" to get all metrics. "DSeats" and "DVS" are always computed, so it is recommended to always return those values.
rvote  A numeric vector with the Republican vote for each precinct.
dvote  A numeric vector with the Democratic vote for each precinct.
...  passed on to redist.metrics
.data  a redist_plans object
plans  A numeric vector (if only one map) or matrix with one row for each precinct and one column for each map. Required.
tau  A non-negative number for calculating Tau Gap. Only used with option "TauGap". Defaults to 1.
biasV  A value between 0 and 1 to compute bias at. Only used with option "BiasV". Defaults to 0.5.
**respV**  A value between 0 and 1 to compute responsiveness at. Only used with option "Responsiveness". Defaults to 0.5.

**bandwidth**  A value between 0 and 1 for computing responsiveness. Only used with option "Responsiveness." Defaults to 0.01.

**draw**  A numeric to specify draw number. Defaults to 1 if only one map provided and the column number if multiple maps given. Can also take a factor input, which will become the draw column in the output if its length matches the number of entries in plans. If the ‘plans’ input is a ‘redist_plans’ object, it extracts the ‘draw’ identifier.

**ncores**  Number of cores to use for parallel computing. Default is 1.

**district_membership**  Deprecated, use plans. A numeric vector (if only one map) or matrix with one row for each precinct and one column for each map. Required.

**nloop**  Deprecated, use draw. A numeric to specify loop number. Defaults to 1 if only one map provided and the column number if multiple maps given.

**Details**

This function computes specified compactness scores for a map. If there is more than one precinct specified for a map, it aggregates to the district level and computes one score.

DSeats is computed as the expected number of Democratic seats with no change in votes. DVS is the Democratic Vote Share, which is the two party vote share with Democratic votes as the numerator. EffGap is the Efficiency Gap, calculated with votes directly. EffGapEqPop is the Efficiency Gap under an Equal Population assumption, calculated with the DVS. TauGap is the Tau Gap, computed with the Equal Population assumption. MeanMedian is the Mean Median difference. Bias is the Partisan Bias computed at 0.5. BiasV is the Partisan Bias computed at value V. Declination is the value of declination at 0.5. Responsiveness is the responsiveness at the user-supplied value with the user-supplied bandwidth. LopsidedWins computed the Lopsided Outcomes value, but does not produce a test statistic. RankedMarginal computes the Ranked Marginal Deviation (0-1, smaller is better). This is also known as the "Gerrymandering Index" and is sometimes presented as this value divided by 10000. SmoothedSeat computes the Smoothed Seat Count Deviation (0-1, smaller is R Bias, bigger is D Bias).

**Value**

A tibble with a column for each specified measure and a column that specifies the map number.

**References**


Examples

data(fl25)
data(fl25_enum)
plans_05 <- fl25_enum$plans[, fl25_enum$pop_dev <= 0.05]
redist.metrics(plans_05, measure = 'all', rvote = fl25$mccain, dvote = fl25$obama)

plan_distances

Description

Compute Distance between Partitions

Usage

plan_distances(plans, measure = "variation of information", ncores = 1)

redist.distances(
  plans,
  measure = "Hamming",
  ncores = 1,
  total_pop = NULL,
  district_membership,
  pop
)

Arguments

plans A matrix with one row for each precinct and one column for each map. Required.

measure String vector indicating which distances to compute. Implemented currently are "Hamming", "Manhattan", "Euclidean", and "variation of information". Use "all" to return all implemented measures. Not case sensitive, and any unique substring is enough, e.g. "ham" for Hamming, or "info" for variation of information.

ncores Number of cores to use for parallel computing. Default is 1.

total_pop The vector of precinct populations. Used only if computing variation of information. If not provided, equal population of precincts will be assumed, i.e. the VI will be computed with respect to the precincts themselves, and not the population.
**Details**

Hamming distance measures the number of different precinct assignments between plans. Manhattan and Euclidean distances are the 1- and 2-norms for the assignment vectors. All three of the Hamming, Manhattan, and Euclidean distances implemented here are not invariant to permutations of the district labels; permuting will cause large changes in measured distance, and maps which are identical up to a permutation may be computed to be maximally distant.

Variation of Information is a metric on population partitions (i.e., districtings) which is invariant to permutations of the district labels, and arises out of information theory. It is calculated as

$$VI(\xi, \xi') = -\sum_{i=1}^{n} \sum_{j=1}^{n} \frac{\text{pop}(\xi_i \cap \xi'_j)}{P} \left( \log(\text{pop}(\xi_i \cap \xi'_j)) - \log(\text{pop}(\xi_i)) - \log(\text{pop}(\xi'_j)) \right)$$

where $\xi, \xi'$ are the partitions, $\xi_i, \xi_j$ the individual districts, $\text{pop}(\cdot)$ is the population, and $P$ the total population of the state. \(VI\) is also expressible as the difference between the joint entropy and the mutual information (see references).

**Value**

`distance_matrix` returns a numeric distance matrix for the chosen metric.

a named list of distance matrices, one for each distance measure selected.

**References**


**Examples**

```r
data(fl25)
data(fl25_enum)

plans_05 <- fl25_enum$plans[, fl25_enum$pop_dev <= 0.05]
distances <- redist.distances(plans_05)
distances$Hamming[1:5, 1:5]
```
### plot.redist_classified

*Plot a plan classification*

#### Description

Plot a plan classification

#### Usage

```r
## S3 method for class 'redist_classified'
plot(x, plans, shp, type = "fill", which = NULL, ...)
```

#### Arguments

- **x**
  - a `redist_classified` object, the output of `classify_plans()`.

- **plans**
  - a `redist_plans` object.

- **shp**
  - a shapefile or `redist_map` object.

- **type**
  - either "line" or "fill". Passed on to `compare_plans()` as plot.

- **which**
  - indices of the splits to plot. Defaults to all

- **...**
  - passed on to `compare_plans()`

#### Value

ggplot comparison plot

### plot.redist_map

*Plot a redist_map*

#### Description

Plot a redist_map

#### Usage

```r
## S3 method for class 'redist_map'
plot(x, fill = NULL, by_distr = FALSE, adj = FALSE, ...)
```
Arguments

- **x**: the `redist_map` object.
- **fill**: `<data-masking>` If present, will be used to color the map units. If using data masking, may need to explicitly name argument `fill=...` in non-interactive contexts to avoid S3 generic issues.
- **by_distr**: if `TRUE` and `fill` is not missing and, color by district and indicate the `fill` variable by shading.
- **adj**: if `TRUE`, force plotting the adjacency graph. Overrides `by_distr`.
- **...**: passed on to `redist.plot.map` (or `redist.plot.adj` if `adj=TRUE`). Useful parameters may include `zoom_to`, `boundaries`, and `title`.

Value

- ggplot2 object

Examples

```r
data(fl25)
d = redist_map(fl25, ndists=3, pop_tol=0.05)
plot(d)
plot(d, BlackPop/pop)

data(fl25_enum)
fl25$dist <- fl25_enum$plans[, 5118]
d <- redist_map(fl25, existing_plan = dist)
plot(d)
```

---

**plot.redist_plans**

Summary plots for `redist_plans`

Description

If no arguments are passed, defaults to plotting the sampling weights for the `redist_plans` object. If no weights exist, plots district populations.

Usage

```r
## S3 method for class 'redist_plans'
plot(x, ..., type = "distr_qtys")
```

Arguments

- **x**: the `redist_plans` object.
- **...**: passed on to the underlying function
- **type**: the name of the plotting function to use. Will have `redist.plot.`, prepended to it; e.g., use `type="plans"` to call `redist.plot.plans`. 
prec_assignment

Extract the district assignments for a precinct across all simulated plans

Description

Extract the district assignments for a precinct across all simulated plans

Usage

prec_assignment(prec, .data = cur_plans())

Arguments

prec
the precinct number or ID
.data
a redist_plans object

Value

integer vector, a row from a plans matrix

prec_cooccurrence

Compute a matrix of precinct co-occurrences

Description

For a map with n precincts Returns an n-by-n matrix, where each entry measures the fraction of the plans in which the row and column precincts were in the same district.

Usage

prec_cooccurrence(plans, which = NULL, sampled_only = TRUE)

Arguments

plans
a redist_plans object.
which <data-masking> which plans to compute the co-occurrence over. Defaults to all.
sampled_only
if TRUE, do not include reference plans.

Value

a symmetric matrix the size of the number of precincts.
print.redist

Print legacy redist objects

Description

Print legacy redist objects

Usage

```r
## S3 method for class 'redist'
print(x, ...)
```

Arguments

- `x`: redist object
- `...`: additional arguments

Value

prints to console

print.redist_classified

Print redist_classified objects

Description

Print redist_classified objects

Usage

```r
## S3 method for class 'redist_classified'
print(x, ...)
```

Arguments

- `x`: redist_classified object
- `...`: additional arguments

Value

prints to console
print.redist_map  Generic to print redist_map

Description
Generic to print redist_map

Usage
## S3 method for class 'redist_map'
print(x, ...)

Arguments
  x   redist_map
  ... additional arguments

Value
Prints to console and returns input redist_map

print.redist_plans  Print method for redist_plans

Description
Print method for redist_plans

Usage
## S3 method for class 'redist_plans'
print(x, ...)

Arguments
  x   redist_plans object
  ... additional arguments

Value
prints to console
**pullback**

Pull back plans to unmerged units

---

**Description**

Merging map units through `merge_by` or `summarize` changes the indexing of each unit. Use this function to take a set of redistricting plans from a redist algorithm and re-index them to be compatible with the original set of units.

**Usage**

```r
pullback(plans)
```

**Arguments**

- `plans` a `redist_plans` object

**Value**

a new, re-indexed, `redist_plans` object

---

**redist.adjacency**

Adjacency List functionality for redist

---

**Description**

Creates an adjacency list that is zero indexed with no skips

**Usage**

```r
redist.adjacency(shp, plan, district_membership)
```

**Arguments**

- `shp` A `SpatialPolygonsDataFrame` or `sf` object. Required.
- `plan` A numeric vector (if only one map) or matrix with one row
- `district_membership` Deprecated – Use plan. A numeric vector (if only one map) or matrix with one row for each precinct and one column for each map. Optional. Checks for contiguity within districts if provided.

**Value**

Adjacency list
redist.calc.frontier.size

*Calculate Frontier Size*

**Description**

Calculate Frontier Size

**Usage**

redist.calc.frontier.size(ordered_path)

**Arguments**

- **ordered_path**  
  path to ordered path created by redist.prep.enumpart

**Value**

List, four objects

- maxnumeric, maximum frontier size
- averagenumeric, average frontier size
- average_sqnumeric, average((frontier size)^2)
- sequencenumeric vector, lists out all sizes for every frontier

**Examples**

```r
## Not run:
data(fl25)
adj <- redist.adjacency(fl25)
redist.prep.enumpart(adj, 'unordered', 'ordered')
redist.calc.frontier.size('ordered')

## End(Not run)
```

redist.choropleth

*Creates a Choropleth*

**Description**

Creates a Choropleth
redist.coarsen.adjacency

Usage

redist.choropleth(
    shp,
    fill = NULL,
    fill_label = "",
    title = "",
    grad = 1,
    lwd = 0
)

Arguments

shp A SpatialPolygonsDataFrame or sf object. Required.
fill A numeric/integer vector with values to color the plot with. Optional.
fill_label A string title of plot. Defaults to empty string. Optional.
title A string title of plot. Defaults to empty string. Optional.
grad Number of colors to make a gradient with. Accepts values of 1 or 2.
lwd Line width. Defaults to 0

Details

Creates a basic choropleth for a provided shp with value. Recommended to normalize data to avoid absolute values, in most use cases.

Value

ggplot map

redist.coarsen.adjacency

Coarsen Adjacency List

Description

Coarsen Adjacency List

Usage

redist.coarsen.adjacency(adj, groups, adjacency)

Arguments

adj A zero-indexed adjacency list. Required.
groups integer vector of elements of adjacency to group
adjacency Deprecated – use adj. A zero-indexed adjacency list
**redist.combine**

*Combine successive runs of redist.flip*

**Description**

`redist.combine` is used to combine successive runs of `redist.flip` into a single data object.

**Usage**

```
redist.combine(savename, nloop, nthin, temper)
```

**Arguments**

- `savename`: The name (without the loop or .rds suffix) of the saved simulations.
- `nloop`: The number of loops being combined. Savename must be non-null.
- `nthin`: How much to thin the simulations being combined.
- `temper`: Whether simulated tempering was used (1) or not (0) in the simulations. Default is 0.

**Details**

This function allows users to combine multiple successive runs of `redist.flip` into a single `redist` object for analysis.

**Value**

`redist.combine` returns an object of class "redist". The object `redist` is a list that contains the following components (the inclusion of some components is dependent on whether tempering techniques are used):

- `plans`: Matrix of congressional district assignments generated by the algorithm. Each row corresponds to a geographic unit, and each column corresponds to a simulation.
- `distance_parity`: Vector containing the maximum distance from parity for a particular simulated redistricting plan.
- `mhdecisions`: A vector specifying whether a proposed redistricting plan was accepted (1) or rejected (0) in a given iteration.
- `mhprob`: A vector containing the Metropolis-Hastings acceptance probability for each iteration of the algorithm.
- `pparam`: A vector containing the draw of the `p` parameter for each simulation, which dictates the number of swaps attempted.
constraint_pop  A vector containing the value of the population constraint for each accepted redistricting plan.

constraint_compact  A vector containing the value of the compactness constraint for each accepted redistricting plan.

constraint_segregation  A vector containing the value of the segregation constraint for each accepted redistricting plan.

constraint_vra  A vector containing the value of the vra constraint for each accepted redistricting plan.

constraint_similar  A vector containing the value of the similarity constraint for each accepted redistricting plan.

constraint_partisan  A vector containing the value of the partisan constraint for each accepted redistricting plan.

constraint_minority  A vector containing the value of the minority constraint for each accepted redistricting plan.

constraint_hinge  A vector containing the value of the hinge constraint for each accepted redistricting plan.

beta_sequence  A vector containing the value of beta for each iteration of the algorithm. Returned when tempering is being used.

mhdecisions_beta  A vector specifying whether a proposed beta value was accepted (1) or rejected (0) in a given iteration of the algorithm. Returned when tempering is being used.

mhprob_beta  A vector containing the Metropolis-Hastings acceptance probability for each iteration of the algorithm. Returned when tempering is being used.

a redist object with entries combined

References

Examples

data(fl25)
data(fl25_enum)
data(fl25_adj)

## Code to run the simulations in Figure 4 in Fifield, Higgins, Imai and Tarr (2015)

## Get an initial partition
init_plan <- fl25_enum$plans[, 5118]

# Run the algorithm
set.seed(1)
temp <- tempdir()
alg_253 <- redist.flip(adj = fl25_adj, total_pop = fl25$pop,
init_plan = init_plan, nsims = 10000,
nloop = 2, savename = paste0(temp, "/test"))
out <- redist.combine(savename = paste0(temp, "/test"), nloop = 2, nthin = 10)

redist.combine.anneal  redist.combine.anneal

Description

Combine files generated by redist.flip.anneal()

Usage

redist.combine.anneal(file_name)

Arguments

file_name  The file name to search for in current working directory.

Value

redist.combine.anneal returns an object of class "redist". The object redist is a list that contains the following components (the inclusion of some components is dependent on whether tempering techniques are used):

plans  Matrix of congressional district assignments generated by the algorithm. Each row corresponds to a geographic unit, and each column corresponds to a simulation.

distance_parity  Vector containing the maximum distance from parity for a particular simulated redistricting plan.

mhdecisions  A vector specifying whether a proposed redistricting plan was accepted (1) or rejected (0) in a given iteration.

mhprob  A vector containing the Metropolis-Hastings acceptance probability for each iteration of the algorithm.

pparam  A vector containing the draw of the p parameter for each simulation, which dictates the number of swaps attempted.

constraint_pop  A vector containing the value of the population constraint for each accepted redistricting plan.
constraint_compact
A vector containing the value of the compactness constraint for each accepted redistricting plan.

constraint_segregation
A vector containing the value of the segregation constraint for each accepted redistricting plan.

constraint_vra
A vector containing the value of the vra constraint for each accepted redistricting plan.

constraint_similar
A vector containing the value of the similarity constraint for each accepted redistricting plan.

constraint_partisan
A vector containing the value of the partisan constraint for each accepted redistricting plan.

constraint_minority
A vector containing the value of the minority constraint for each accepted redistricting plan.

constraint_hinge
A vector containing the value of the hinge constraint for each accepted redistricting plan.

beta_sequence
A vector containing the value of beta for each iteration of the algorithm. Returned when tempering is being used.

mhdecisions_beta
A vector specifying whether a proposed beta value was accepted (1) or rejected (0) in a given iteration of the algorithm. Returned when tempering is being used.

mhprob_beta
A vector containing the Metropolis-Hastings acceptance probability for each iteration of the algorithm. Returned when tempering is being used.

**Description**

redist.combine.mpi is used to combine successive runs of redist.mcmc.mpi into a single data object.

**Usage**

redist.combine.mpi(savename, nloop, nthin, tempadj)

**Arguments**

savename  The name (without the loop or .RData suffix) of the saved simulations.
nloop  The number of loops being combined.
nthin  How much to thin the simulations being combined.
tempadj  The temperature adjacency object saved by redist.mcmc.mpi.
Details

This function allows users to combine multiple successive runs of redist.mcmc.mpi into a single redist object for analysis.

Value

redist.combine.mpi returns an object of class "redist". The object redist is a list that contains the following components (the inclusion of some components is dependent on whether tempering techniques are used):

- **plans**: Matrix of congressional district assignments generated by the algorithm. Each row corresponds to a geographic unit, and each column corresponds to a simulation.
- **distance_parity**: Vector containing the maximum distance from parity for a particular simulated redistricting plan.
- **mhdecisions**: A vector specifying whether a proposed redistricting plan was accepted (1) or rejected (0) in a given iteration.
- **mhprob**: A vector containing the Metropolis-Hastings acceptance probability for each iteration of the algorithm.
- **pparam**: A vector containing the draw of the p parameter for each simulation, which dictates the number of swaps attempted.
- **constraint_pop**: A vector containing the value of the population constraint for each accepted redistricting plan.
- **constraint_compact**: A vector containing the value of the compactness constraint for each accepted redistricting plan.
- **constraint_vra**: A vector containing the value of the vra constraint for each accepted redistricting plan.
- **constraint_similar**: A vector containing the value of the similarity constraint for each accepted redistricting plan.
- **beta_sequence**: A vector containing the value of beta for each iteration of the algorithm. Returned when tempering is being used.
- **mhdecisions_beta**: A vector specifying whether a proposed beta value was accepted (1) or rejected (0) in a given iteration of the algorithm. Returned when tempering is being used.
- **mhprob_beta**: A vector containing the Metropolis-Hastings acceptance probability for each iteration of the algorithm. Returned when tempering is being used.

References

redist.constraint.helper

Create Constraints for SMC

Description
Create Constraints for SMC

Usage
redist.constraint.helper(
  constraints = "vra",
  tgt_min = 0.55,
  group_pop,
  total_pop,
  ndists,
  mmmd,
  strength_vra = 2500,
  pow_vra = 1.5,
  grouppop,
  fullpop
)

Arguments

  constraints Vector of constraints to include. Currently only 'vra' implemented.
tgt_min Defaults to 0.55. If `vra` included, the minority percent to encourage in each district.
group_pop A vector of populations for some subgroup of interest.
total_pop A vector containing the populations of each geographic unit.
ndists The total number of districts.
nmmd The number of majority minority districts to target for `vra` constraint
strength_vra The strength of the `vra` constraint. Defaults to 2500.
pow_vra The exponent for the `vra` constraint. Defaults to 1.5.
grouppop Deprecated, use group_pop. A vector of populations for some subgroup of interest.
fullpop Deprecated, use total_pop. A vector containing the populations of each geographic unit.

Value

list of lists for each constraint selected

---

redist.county.id Create County IDs

Description
Create County IDs

Usage
redist.county.id(counties)

Arguments

- counties vector of counties, required.

Value
A vector with an ID that corresponds from 1:n counties

Examples

```r
set.seed(2)
counties <- sample(c(rep('a', 20), rep('b', 5)))
redist.county.id(counties)
```
redist.county.relabel  Relabel Discontinuous Counties

Description

Relabel Discontinuous Counties

Usage

redist.county.relabel(adj, counties, simplify = TRUE, adjacency)

Arguments

adj  adjacency list
counties  character vector of county names
simplify  boolean - TRUE returns a numeric vector of ids, while FALSE appends a number when there are multiple connected components.
adjacency  Deprecated, use adj. adjacency list

Value

character vector of county names

Examples

set.seed(2)
data(fl25)
data(fl25_adj)
counties <- sample(c(rep('a', 20), rep('b', 5)))redist.county.relabel(fl25_adj, counties)

redist.crsg  Redistricting via Compact Random Seed and Grow Algorithm

Description

redist.crsg generates redistricting plans using a random seed a grow algorithm. This is the compact districting algorithm described in Chen and Rodden (2013).
Usage

redist.crsg(
  adj,
  total_pop,
  shp,
  ndists,
  pop_tol,
  verbose = TRUE,
  maxiter = 5000,
  adj.list,
  population,
  area,
  x_center,
  y_center,
  thresh
)

Arguments

adj List of length N, where N is the number of precincts. Each list element is an integer vector indicating which precincts that precinct is adjacent to. It is assumed that precinct numbers start at 0.

total_pop numeric vector of length N, where N is the number of precincts. Each element lists the population total of the corresponding precinct, and is used to enforce pop_tol constraints.

shp An sf dataframe to compute area and centroids with.

ndists integer, the number of districts we want to partition the precincts into.

pop_tol numeric, indicating how close district population targets have to be to the target population before algorithm converges. pop_tol=0.05 for example means that all districts must be between 0.95 and 1.05 times the size of target.pop in population size.

verbose boolean, indicating whether the time to run the algorithm is printed.

maxiter integer, indicating maximum number of iterations to attempt before convergence to population constraint fails. If it fails once, it will use a different set of start values and try again. If it fails again, redist.rsg() returns an object of all NAs, indicating that use of more iterations may be advised. Default is 5000.

adj.list Deprecated, use adj. List of length N, where N is the number of precincts. Each list element is an integer vector indicating which precincts that precinct is adjacent to. It is assumed that precinct numbers start at 0.

population Deprecated, use total_pop. numeric vector of length N, where N is the number of precincts. Each element lists the population total of the corresponding precinct, and is used to enforce population constraints.

area Deprecated, use shp. numeric vector of length N, where N is the number of precincts. Each element is the area of the corresponding precinct.
x_center  Deprecated, use shp. numeric vector of length N, where N is the number of precincts. Each element is the x coordinate of the geographic centroid of the corresponding precinct.

y_center  Deprecated, use shp. numeric vector of length N, where N is the number of precincts. Each element is the y coordinate of the geographic centroid of the corresponding precinct.

thresh  Deprecated, use pop_tol. numeric, indicating how close district population targets have to be to the target population before algorithm converges. thresh=0.05 for example means that all districts must be between 0.95 and 1.05 times the size of target.pop in population size.

Value

list, containing three objects containing the completed redistricting plan.

- plan  A vector of length N, indicating the district membership of each precinct.
- district_list  A list of length Ndistrict. Each list contains a vector of the precincts in the respective district.
- district_pop  A vector of length Ndistrict, containing the population totals of the respective districts.

References


Examples

```r
data("fl25")
adj <- redist.adjacency(fl25)
redist.crsg(adj = adj, total_pop = fl25$pop, shp = fl25, ndists = 2, pop_tol = .1)
```

redist.diagplot  Diagnostic plotting functionality for MCMC redistricting.

Description

redist.diagplot generates several common MCMC diagnostic plots.

Usage

```r
redist.diagplot(sumstat, plot = c("trace", "autocorr", "densplot", "mean", "gelmanrubin"), logit = FALSE, savename = NULL)
```
Arguments

sumstat  A vector, list, mcmc or mcmc.list object containing a summary statistic of choice.

plot    The type of diagnostic plot to generate: one of "trace", "autocorr", "densplot", "mean", "gelmanrubin". If plot = "gelmanrubin", the input sumstat must be of class mcmc.list or list.

logit   Flag for whether to apply the logistic transformation for the summary statistic. The default is FALSE.

savename Filename to save the plot. Default is NULL.

Details

This function allows users to generate several standard diagnostic plots from the MCMC literature, as implemented by Plummer et. al (2006). Diagnostic plots implemented include trace plots, autocorrelation plots, density plots, running means, and Gelman-Rubin convergence diagnostics (Gelman & Rubin 1992).

Value

Returns a plot of file type .pdf.

References


Gelman, Andrew and Donald Rubin. (1992) "Inference from iterative simulations using multiple sequences (with discussion)." Statistical Science.


Examples

data(fl25)
data(fl25_enum)
data(fl25_adj)

## Get an initial partition
init_plan <- fl25_enum$plans[, 5118]

## 25 precinct, three districts - no pop constraint ##
alg_253 <- redist.flip(adj = fl25_adj, total_pop = fl25$pop, init_plan = init_plan, nsims = 10000)

## Get Republican Dissimilarity Index from simulations
rep_dmi_253 <- redist.segcalc(alg_253, fl25$mccain, fl25$pop)

## Generate diagnostic plots
redist.enumerate

redist.diagplot(rep_dmi_253, plot = "trace")
redist.diagplot(rep_dmi_253, plot = "autocorr")
redist.diagplot(rep_dmi_253, plot = "densplot")
redist.diagplot(rep_dmi_253, plot = "mean")

## Gelman Rubin needs two chains, so we run a second
alg_253_2 <- redist.flip(adj = fl25_adj,
total_pop = fl25$pop,
init_plan = init_plan, nsims = 10000)

rep_dmi_253_2 <- redist.segcalc(alg_253_2, fl25$mccain, fl25$pop)

## Make a list out of the objects:
rep_dmi_253_list <- list(rep_dmi_253, rep_dmi_253_2)

## Generate Gelman Rubin diagnostic plot
redist.diagplot(sumstat = rep_dmi_253_list, plot = 'gelmanrubin')

---

redist.enumerate  

**Deprecated: Exact Redistricting Plan Enumerator**

**Description**

redist.enumerate uses a spanning-tree method to fully enumerate all valid redistricting plans with $n$ districts given a set of geographic units. redist.enumerate also allows users to implement minimum and maximum numbers of geographic units per district, as well as population parity requirements.

**Usage**

```r
redist.enumerate(
  adj,  
  ndists = 2,  
  total_pop = NULL,  
  nconstraintlow = NULL,  
  nconstrainthigh = NULL,  
  pop_tol = NULL,  
  adjobj,  
  popvec,  
  popcons,  
  contiguitymap = "rooks"
)
```

**Arguments**

- **adj**  
  An adjacency list, matrix, or object of class SpatialPolygonsDataFrame.

- **ndists**  
  The desired number of congressional districts. The default is 2.
total_pop  A vector of geographic unit populations. The default is NULL.
nconstraintlow Lower bound for number of geographic units to include in a district. The default is NULL.
nconstrainthigh Lower bound for number of geographic units to include in a district. The default is NULL.
pop_tol The strength of the hard population constraint. pop_tol = 0.05 means that any proposed swap that brings a district more than 5% away from population parity will be rejected. The default is NULL.
adjob Deprecated, use adj. An adjacency list, matrix, or object of class SpatialPolygonsDataFrame.
popvec Deprecated, use total_pop. A vector of geographic unit populations.
popcons Deprecated, use pop_tol. The strength of the hard population constraint. popcons = 0.05 means that any proposed swap that brings a district more than 5% away from population parity will be rejected. The default is NULL.
contiguitymap Use queens or rooks distance criteria for generating an adjacency list from a "SpatialPolygonsDataFrame" data type. Default is "rooks".

Details

This function allows users to input a set of geographic units to generate all valid partitions of $n$ congressional districts. The function uses a set of spanning-tree methods to generate all valid, contiguous partitions, which makes it more efficient than brute-force methods. However, even with these methods, full redistricting problems quickly become intractable, necessitating the use of the MCMC-based methods implemented in redist.mcmc.

Value

redist.enumerate returns an object of class "list". Each entry in the list is a vector of congressional district assignments, where the first entry in the vector corresponds to the congressional district assignment of the first geographic unit.

References


redist.enumpart  

Enumerate All Partitions

Description

Single function for standard enumeration analysis.
Usage

```r
redist.enumpart(
  adj,
  unordered_path,
  ordered_path,
  out_path,
  ndists = 2,
  all = TRUE,
  n = NULL,
  weight_path = NULL,
  lower = NULL,
  upper = NULL,
  init = FALSE,
  read = TRUE,
  total_pop = NULL,
  adjlist,
  population,
  ndist
)
```

Arguments

- `adj`: zero indexed adjacency list.
- `unordered_path`: valid path to output the unordered adjacency map to.
- `ordered_path`: valid path to output the ordered adjacency map to.
- `out_path`: Valid path to output the enumerated districts.
- `ndists`: number of districts to enumerate.
- `all`: boolean. TRUE outputs all districts. FALSE samples n districts.
- `n`: integer. Number of districts to output if all is FALSE. Returns districts selected from uniform random distribution.
- `weight_path`: A path (not including ".dat") to a space-delimited file containing a vector of vertex weights, to be used along with lower and upper.
- `lower`: A lower bound on each partition’s total weight, implemented by rejection sampling.
- `upper`: An upper bound on each partition’s total weight.
- `init`: Runs redist.init.enumpart. Defaults to false. Should be run on first use.
- `read`: boolean. Defaults to TRUE. reads.
- `total_pop`: Integer Vector. Defaults to NULL. If supplied, computes the parity.
- `adjlist`: Deprecated, use adj. zero indexed adjacency list.
- `population`: Deprecated, use total_pop. Integer Vector. Defaults to NULL. If supplied, computes the parity.
- `ndist`: Deprecated, use ndists. number of districts to enumerate.
**redist.find.target**  
*Find Majority Minority Remainder*

**Description**

Given a percent goal for majority minority districts, this computes the average value of minority in non-majority minority districts. This value is "tgt_other" in `redist.mcmc` and `redist.smc`.

**Usage**

```r
redist.find.target(
    tgt_min,
    group_pop,
    grouppop,  # Deprecated, use group_pop.
    total_pop,
    fullpop,  # Deprecated, use total_pop.
    ndists,
    nmmd  # The number of majority minority districts.
)
```

**Arguments**

- `tgt_min` target group population for majority minority district
- `group_pop` A vector of populations for some subgroup of interest.
- `grouppop` A vector of populations for some subgroup of interest. (Deprecated, use `group_pop`.)
- `total_pop` A vector containing the populations of each geographic unit.
- `fullpop` A vector containing the populations of each geographic unit. (Deprecated, use `total_pop`.)
- `ndists` The number of congressional districts.
- `nmmd` The number of majority minority districts.

**Value**

numeric value to target
Description

`redist.findparams` is used to find optimal parameter values of `redist.flip` for a given map.

Usage

```r
redist.findparams(
  adj,  
  total_pop,  
  nsims,  
  ndists = NULL,  
  init_plan = NULL,  
  adapt_lambda = FALSE,  
  adapt_eprob = FALSE,  
  params,  
  ssdmat = NULL,  
  group_pop = NULL,  
  counties = NULL,  
  nstartval_store = 1,  
  maxdist_startval = 100,  
  maxiterrsg = 5000,  
  report_all = TRUE,  
  parallel = FALSE,  
  ncores = NULL,  
  log = FALSE,  
  verbose = TRUE,
  adjobj,  
  popvec,  
  initcds,  
  grouppopvec,  
  countymembership,  
  nthreads
)
```

Arguments

- **adj**: An adjacency matrix, list, or object of class "SpatialPolygonsDataFrame."
- **total_pop**: A vector containing the populations of each geographic unit.
- **nsims**: The number of simulations run before a save point.
- **ndists**: The number of congressional districts. The default is NULL.
- **init_plan**: A vector containing the congressional district labels of each geographic unit. The default is NULL. If not provided, random and contiguous congressional district assignments will be generated using `redist.rsg`. 
adapt_lambda  Whether to adaptively tune the lambda parameter so that the Metropolis-Hastings acceptance probability falls between 20% and 40%. Default is FALSE.
adapt_eprob  Whether to adaptively tune the edgecut probability parameter so that the Metropolis-Hastings acceptance probability falls between 20% and 40%. Default is FALSE.
params  A matrix of parameter values to test, such as the output of expand.grid. Parameters accepted for params include eprob, lambda, pop_tol, beta, and constraint.
ssdmatrix  A matrix of squared distances between geographic units. The default is NULL.
group_pop  A vector of populations for some sub-group of interest. The default is NULL.
counties  A vector of county membership assignments. The default is NULL.
nstartval_store  The number of maps to sample from the preprocessing chain for use as starting values in future simulations. Default is 1.
maxdist_startval  The maximum distance from the starting map that sampled maps should be. Default is 100 (no restriction).
maxiterrsg  Maximum number of iterations for random seed-and-grow algorithm to generate starting values. Default is 5000.
report_all  Whether to report all summary statistics for each set of parameter values. Default is TRUE.
parallel  Whether to run separate parameter settings in parallel. Default is FALSE.
ncores  Number of parallel tasks to run, declared outside of the function. Default is NULL.
log  Whether to open a log to track progress for each parameter combination being tested. Default is FALSE.
verbose  Whether to print additional information about the tests. Default is TRUE.
adjob  Deprecated, use adj. An adjacency matrix, list, or object of class "SpatialPolygonsDataFrame."
popvec  Deprecated, use total_pop. A vector containing the populations of each geographic unit.
initcds  Deprecated, use init_plan. A vector containing the congressional district labels of each geographic unit. The default is NULL. If not provided, random and contiguous congressional district assignments will be generated using redist.rsg.
group_popvec  A vector of populations for some sub-group of interest. The default is NULL.
countymembership  Deprecated, use counties. A vector of county membership assignments. The default is NULL.
nthreads  Deprecated, use ncores. Number of parallel tasks to run, declared outside of the function. Default is NULL.

Details

This function allows users to test multiple parameter settings of redist.flip in preparation for a longer run for analysis.
Values

redist.findparams returns a print-out of summary statistics about each parameter setting.

References


Examples

data(fl25)
data(fl25_enum)
data(fl25_adj)

## Get an initial partition
init_plan <- fl25_enum$plans[, 5118]

params <- expand.grid(eprob = c(.01, .05, .1))

## Run the algorithm
redist.findparams(adj = fl25_adj, total_pop = fl25$pop,
                   init_plan = init_plan, nsims = 10000, params = params)

redist.init.enumpart  Initialize enumpart

Description

This ensures that the enumerate partitions programs is prepared to run. This must be run once per install of the redist package.

Usage

redist.init.enumpart()

Value

0 on success

References

redist.ipw

Inverse probability reweighting for MCMC Redistricting

Description

redist.ipw properly weights and resamples simulated redistricting plans so that the set of simulated plans resemble a random sample from the underlying distribution. redist.ipw is used to correct the sample when population parity, geographic compactness, or other constraints are implemented.

Usage

redist.ipw(
algout,
resampleconstraint = c("pop", "compact", "segregation", "similar"),
targetbeta,
targetpop = NULL,
temper = 0
)

Arguments

algout An object of class "redist".
resampleconstraint The constraint implemented in the simulations: one of "pop", "compact", "segregation", or "similar".
targetbeta The target value of the constraint.
targetpop The desired level of population parity. targetpop = 0.01 means that the desired distance from population parity is 1%. The default is NULL.
temper A flag for whether simulated tempering was used to improve the mixing of the Markov Chain. The default is 1.

Details

This function allows users to resample redistricting plans using inverse probability weighting techniques described in Rubin (1987). This techniques reweights and resamples redistricting plans so that the resulting sample is representative of a random sample from the uniform distribution.
**Value**

`redist.ipw` returns an object of class "redist". The object `redist` is a list that contains the following components (the inclusion of some components is dependent on whether tempering techniques are used):

- **plans**: Matrix of congressional district assignments generated by the algorithm. Each row corresponds to a geographic unit, and each column corresponds to a simulation.
- **distance_parity**: Vector containing the maximum distance from parity for a particular simulated redistricting plan.
- **mhdecisions**: A vector specifying whether a proposed redistricting plan was accepted (1) or rejected (0) in a given iteration.
- **mhprob**: A vector containing the Metropolis-Hastings acceptance probability for each iteration of the algorithm.
- **pparam**: A vector containing the draw of the p parameter for each simulation, which dictates the number of swaps attempted.
- **constraint_pop**: A vector containing the value of the population constraint for each accepted redistricting plan.
- **constraint_compact**: A vector containing the value of the compactness constraint for each accepted redistricting plan.
- **constraint_segregation**: A vector containing the value of the segregation constraint for each accepted redistricting plan.
- **constraint_similar**: A vector containing the value of the similarity constraint for each accepted redistricting plan.
- **constraint_vra**: A vector containing the value of the vra constraint for each accepted redistricting plan.
- **constraint_partisan**: A vector containing the value of the partisan constraint for each accepted redistricting plan.
- **constraint_minority**: A vector containing the value of the minority constraint for each accepted redistricting plan.
- **constraint_hinge**: A vector containing the value of the hinge constraint for each accepted redistricting plan.
- **beta_sequence**: A vector containing the value of beta for each iteration of the algorithm. Returned when tempering is being used.
- **mhdecisions_beta**: A vector specifying whether a proposed beta value was accepted (1) or rejected (0) in a given iteration of the algorithm. Returned when tempering is being used.
- **mhprob_beta**: A vector containing the Metropolis-Hastings acceptance probability for each iteration of the algorithm. Returned when tempering is being used.
References


Examples

data(iowa)
adj <- redist.adjacency(iowa)
init_plan <- iowa$cd_2010
alg <- redist.flip(adj = adj, total_pop = iowa$pop,
    init_plan = init_plan, nsims = 1000,
    constraint = 'population', constraintweights = 5.4)

alg_ipw <- redist.ipw(algout = alg,
    resampleconstraint = 'pop',
    targetbeta = 1,
    targetpop = 0.05)

redist.map

Creates a map with optional graph overlay

Description

Creates a map with optional graph overlay

Usage

redist.map(
    shp = NULL,
    adj = NULL,
    plan = NULL,
    centroids = TRUE,
    edges = TRUE,
    boundaries = TRUE,
    drop = FALSE,
    title = "",
    adjacency,
    district_membership
)
redist.mcmc

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>shp</td>
<td>A SpatialPolygonsDataFrame or sf object. Required.</td>
</tr>
<tr>
<td>adj</td>
<td>A zero-indexed adjacency list. Created with redist.adjacency if not supplied. Default is NULL. Optional.</td>
</tr>
<tr>
<td>plan</td>
<td>A numeric vector with one entry for each precinct in shp. Used to color the districts. Default is NULL. Optional.</td>
</tr>
<tr>
<td>centroids</td>
<td>A logical indicating if centroids should be plotted. Default is TRUE.</td>
</tr>
<tr>
<td>edges</td>
<td>A logical indicating if edges should connect adjacent centroids. Default is TRUE.</td>
</tr>
<tr>
<td>boundaries</td>
<td>A logical indicating if precinct boundaries should be plotted.</td>
</tr>
<tr>
<td>drop</td>
<td>A logical indicating if edges that cross districts should be dropped. Default is FALSE.</td>
</tr>
<tr>
<td>title</td>
<td>A string title of plot. Defaults to empty string. Optional.</td>
</tr>
<tr>
<td>adjacency</td>
<td>Deprecated, use adj. A zero-indexed adjacency list. Created with redist.adjacency if not supplied. Default is NULL.</td>
</tr>
<tr>
<td>district_membership</td>
<td>Deprecated, use plan. A numeric vector with one row for each precinct in shp. Used to color the districts. Default is NULL. Optional.</td>
</tr>
</tbody>
</table>

Value

ggplot map

redist.mcmc MCMC Redistricting Simulator

Description

redist.mcmc is used to simulate Congressional redistricting plans using Markov Chain Monte Carlo methods.

redist.mcmc is used to simulate Congressional redistricting plans using Markov Chain Monte Carlo methods.

Usage

redist.mcmc(
    adj,
    total_pop,
    nsims,
    ndists = NULL,
    init_plan = NULL,
    loopscompleted = 0,
    nloop = 1,
    warmup = 0,
    nthin = 1,
eprob = 0.05,
lambda = 0,
pop_tol = NULL,
group_pop = NULL,
areasvec = NULL,
counties = NULL,
borderlength_mat = NULL,
ssdmat = NULL,
temper = FALSE,
constraint = NULL,
constraintweights = NULL,
compactness_metric = "fryer-holden",
partisan_metric = "efficiency-gap",
ssd_denom = 1,
betaseq = "powerlaw",
betaseqlength = 10,
betaweights = NULL,
adjswaps = TRUE,
rngseed = NULL,
maxiterrsg = 5000,
adapt_lambda = FALSE,
adapt_eprob = FALSE,
extact_mh = FALSE,
savename = NULL,
verbose = TRUE,
tgt_min = 0.55,
tgt_other = 0.25,
rvote = NULL,
dvote = NULL,
minorityprop = NULL,
adjobj,
popvec,
initcds,
popcons,
grouppopvec,
countymembership,
contiguitymap
)

redist.flip(
adj,
total_pop,
nsims,
ndists = NULL,
init_plan = NULL,
loopscompleted = 0,
nloop = 1,
warmup = 0,
nthin = 1,
eprob = 0.05,
lambda = 0,
pop_tol = NULL,
group_pop = NULL,
areasvec = NULL,
counties = NULL,
borderlength_mat = NULL,
ssdmat = NULL,
temper = FALSE,
constraint = NULL,
constraintweights = NULL,
compactness_metric = "fryer-holden",
partisan_metric = "efficiency-gap",
ssd_denom = 1,
betaseq = "powerlaw",
betaseqlength = 10,
betaweights = NULL,
adjswaps = TRUE,
rngseed = NULL,
maxiterrsg = 5000,
adapt_lambda = FALSE,
adapt_eprob = FALSE,
exact_mh = FALSE,
savename = NULL,
verbose = TRUE,
tgt_min = 0.55,
tgt_other = 0.25,
rvote = NULL,
dvote = NULL,
minorityprop = NULL,
adjobj,
popvec,
initcds,
popcons,
grouppopvec,
countymembership,
contiguitymap
)

Arguments

adj adjacency matrix, list, or object of class "SpatialPolygonsDataFrame."
total_pop A vector containing the populations of each geographic unit
nsims The number of simulations run before a save point.
ndists The number of congressional districts. The default is NULL.
init_plan A vector containing the congressional district labels of each geographic unit.
If not provided, random and contiguous congressional district assignments will
be generated using redist.smc. To use the old behavior of generating with redist.rsg, provide init_plan = 'rsg'.

loopscompleted
Number of save points reached by the algorithm. The default is 0.
nloop
The total number of save points for the algorithm. The default is 1. Note that the total number of simulations run will be nsims * nloop. savename must be non-null.

warmup
The number of warmup samples to discard. The default is 0.
nthin
The amount by which to thin the Markov Chain. The default is 1.
eprob
The probability of keeping an edge connected. The default is 0.05.
lambda
The parameter determining the number of swaps to attempt each iteration of the algorithm. The number of swaps each iteration is equal to Pois(lambda) + 1. The default is 0.

pop_tol
The strength of the hard population constraint. pop_tol = 0.05 means that any proposed swap that brings a district more than 5% away from population parity will be rejected. The default is NULL.
group_pop
A vector of populations for some sub-group of interest. The default is NULL.
areasvec
A vector of precinct areas for discrete Polsby-Popper. The default is NULL.
counties
A vector of county membership assignments. The default is NULL.
borderlength_mat
A matrix of border length distances, where the first two columns are the indices of precincts sharing a border and the third column is its distance. Default is NULL.

ssdmat
A matrix of squared distances between geographic units. The default is NULL.
temper
Whether to use simulated tempering algorithm. Default is FALSE.
constraint
Which constraint to apply. Accepts any combination of compact, segregation, vra, population, similarity, partisan, minority, hinge, countysplit, or none (no constraint applied). The default is NULL.
constraintweights
The weights to apply to each constraint. Should be a vector the same length as constraint. Default is NULL.

compactness_metric
The compactness metric to use when constraining on compactness. Default is fryer-holden, the other implemented options are polsby/popper and edges-removed.

partisan_metric
The partisan metric to use when constraining on partisan metrics. Only implemented is "efficiency-gap", the default.

ssd_denom
The normalizing constant for the sum-of-squared distance Fryer-Holden metric. Default is 1.0 (unnormalized).

betaseq
Sequence of beta values for tempering. The default is powerlaw (see Fifield et. al (2015) for details).

betaseqlength
Length of beta sequence desired for tempering. The default is 10.

betaweights
Sequence of weights for different values of beta. Allows the user to upweight certain values of beta over others. The default is NULL (equal weighting).
adjswaps: Flag to restrict swaps of beta so that only values adjacent to current constraint are proposed. The default is TRUE.

rngseed: Allows the user to set the seed for the simulations. Default is NULL.

maxiterrsg: Maximum number of iterations for random seed-and-grow algorithm to generate starting values. Default is 5000.

adapt_lambda: Whether to adaptively tune the lambda parameter so that the Metropolis-Hastings acceptance probability falls between 20% and 40%. Default is FALSE.

adapt_eprob: Whether to adaptively tune the edgecut probability parameter so that the Metropolis-Hastings acceptance probability falls between 20% and 40%. Default is FALSE.

exact_mh: Whether to use the approximate (0) or exact (1) Metropolis-Hastings ratio calculation for accept-reject rule. Default is FALSE.

savename: Filename to save simulations. Default is NULL.

verbose: Whether to print initialization statement. Default is TRUE.

tgt_min: The majority minority target percent as a decimal. Default is 0.55.

tgt_other: The remaining target percent as a decimal. Default is 0.25.

rvote: integer vector of votes for Republicans by precinct

dvote: integer vector of votes for Democrats by precinct

minorityprop: numeric vector of targeted minority proportions for the top districts with that proportion

adjob: Deprecated, use adj. An adjacency matrix, list, or object of class "SpatialPolygonsDataFrame."

popvec: Deprecated, use total_pop. A vector containing the populations of each geographic unit

initcds: Deprecated, use init_plan. A vector containing the congressional district labels of each geographic unit. The default is NULL. If not provided, random and contiguous congressional district assignments will be generated using redist.rsg.

popcons: Deprecated, use pop_tol. The strength of the hard population constraint. popcons = 0.05 means that any proposed swap that brings a district more than 5% away from population parity will be rejected. The default is NULL.

groupopvec: Deprecated, use group_pop. A vector of populations for some sub-group of interest. The default is NULL.

countymembership: Deprecated, use counties. A vector of county membership assignments. The default is NULL.

contiguitymap: Deprecated. Use queens or rooks distance criteria for generating an adjacency list from a "SpatialPolygonsDataFrame" data type. Default is "rooks".

Details

This function allows users to simulate redistricting plans using Markov Chain Monte Carlo methods. Several constraints corresponding to substantive requirements in the redistricting process are implemented, including population parity and geographic compactness. In addition, the function
includes multiple-swap and simulated tempering functionality to improve the mixing of the Markov Chain.

This function allows users to simulate redistricting plans using Markov Chain Monte Carlo methods. Several constraints corresponding to substantive requirements in the redistricting process are implemented, including population parity and geographic compactness. In addition, the function includes multiple-swap and simulated tempering functionality to improve the mixing of the Markov Chain.

Value

redist.mcmc returns an object of class "redist". The object redist is a list that contains the following components (the inclusion of some components is dependent on whether tempering techniques are used):

- **plans** Matrix of congressional district assignments generated by the algorithm. Each row corresponds to a geographic unit, and each column corresponds to a simulation.
- **distance_parity** Vector containing the maximum distance from parity for a particular simulated redistricting plan.
- **mhdecisions** A vector specifying whether a proposed redistricting plan was accepted (1) or rejected (0) in a given iteration.
- **mhprob** A vector containing the Metropolis-Hastings acceptance probability for each iteration of the algorithm.
- **pparam** A vector containing the draw of the $p$ parameter for each simulation, which dictates the number of swaps attempted.
- **constraint_pop** A vector containing the value of the population constraint for each accepted redistricting plan.
- **constraint_compact** A vector containing the value of the compactness constraint for each accepted redistricting plan.
- **constraint_segregation** A vector containing the value of the segregation constraint for each accepted redistricting plan.
- **constraint_vra** A vector containing the value of the vra constraint for each accepted redistricting plan.
- **constraint_similar** A vector containing the value of the similarity constraint for each accepted redistricting plan.
- **constraint_partisan** A vector containing the value of the partisan constraint for each accepted redistricting plan.
- **constraint_minority** A vector containing the value of the minority constraint for each accepted redistricting plan.
redist.mcmc

constraint_hinge
A vector containing the value of the hinge constraint for each accepted redistricting plan.

beta_sequence
A vector containing the value of beta for each iteration of the algorithm. Returned when tempering is being used.

mhdecisions_beta
A vector specifying whether a proposed beta value was accepted (1) or rejected (0) in a given iteration of the algorithm. Returned when tempering is being used.

mhprob_beta
A vector containing the Metropolis-Hastings acceptance probability for each iteration of the algorithm. Returned when tempering is being used.

redist.mcmc returns an object of class "redist". The object redist is a list that contains the following components (the inclusion of some components is dependent on whether tempering techniques are used):

plans
Matrix of congressional district assignments generated by the algorithm. Each row corresponds to a geographic unit, and each column corresponds to a simulation.

distance_parity
Vector containing the maximum distance from parity for a particular simulated redistricting plan.

mhdecisions
A vector specifying whether a proposed redistricting plan was accepted (1) or rejected (0) in a given iteration.

mhprob
A vector containing the Metropolis-Hastings acceptance probability for each iteration of the algorithm.

pparam
A vector containing the draw of the p parameter for each simulation, which dictates the number of swaps attempted.

constraint_pop
A vector containing the value of the population constraint for each accepted redistricting plan.

constraint_compact
A vector containing the value of the compactness constraint for each accepted redistricting plan.

constraint_segregation
A vector containing the value of the segregation constraint for each accepted redistricting plan.

constraint_vra
A vector containing the value of the vra constraint for each accepted redistricting plan.

constraint_similar
A vector containing the value of the similarity constraint for each accepted redistricting plan.

constraint_partisan
A vector containing the value of the partisan constraint for each accepted redistricting plan.

constraint_minority
A vector containing the value of the minority constraint for each accepted redistricting plan.
constraint_hinge
A vector containing the value of the hinge constraint for each accepted redistricting plan.

beta_sequence
A vector containing the value of beta for each iteration of the algorithm. Returned when tempering is being used.

mhdecisions_beta
A vector specifying whether a proposed beta value was accepted (1) or rejected (0) in a given iteration of the algorithm. Returned when tempering is being used.

mhprob_beta
A vector containing the Metropolis-Hastings acceptance probability for each iteration of the algorithm. Returned when tempering is being used.

References


Examples

data(fl25)
data(fl25_enum)
data(fl25_adj)

## Code to run the simulations in Figure 4 in Fifield, Higgins, Imai and Tarr (2015)

## Get an initial partition
init_plan <- fl25_enum$plans[, 5118]

## Run the algorithm
alg_253 <- redist.flip(adj = fl25_adj, total_pop = fl25$pop,
init_plan = init_plan, nsims = 10000)

## You can also let it find a plan on its own!
sims <- redist.flip(adj = fl25_adj, total_pop = fl25$pop,
ndists = 3, nsims = 10000)
Description

redist.flip.anneal simulates congressional redistricting plans using Markov chain Monte Carlo methods coupled with simulated annealing.

Usage

redist.mcmc.anneal(
  adj,
  total_pop,
  ndists = NULL,
  init_plan = NULL,
  num_hot_steps = 40000,
  num_annealing_steps = 60000,
  num_cold_steps = 20000,
  eprob = 0.05,
  lambda = 0,
  pop_tol = NULL,
  group_pop = NULL,
  areasvec = NULL,
  counties = NULL,
  borderlength_mat = NULL,
  ssdmat = NULL,
  constraint = NULL,
  constraintweights = NULL,
  compactness_metric = "fryer-holden",
  partisan_metric = "efficiency-gap",
  rngseed = NULL,
  maxiterrsg = 5000,
  adapt_lambda = FALSE,
  adapt_eprob = FALSE,
  exact_mh = FALSE,
  savename = NULL,
  verbose = TRUE,
  ncores = 1,
  tgt_min = 0.55,
  tgt_other = 0.25,
  rvote = NULL,
  dvote = NULL,
  minorityprop = NULL,
  adjobj,
  popvec,
  initcds,
  popcons,
  grouppopvec,
  countymembership,
  contiguitymap)
redist.mcmc.anneal(
    adj,
    total_pop,
    ndists = NULL,
    init_plan = NULL,
    num_hot_steps = 40000,
    num_annealing_steps = 60000,
    num_cold_steps = 20000,
    eprob = 0.05,
    lambda = 0,
    pop_tol = NULL,
    group_pop = NULL,
    areasvec = NULL,
    counties = NULL,
    borderlength_mat = NULL,
    ssdmat = NULL,
    constraint = NULL,
    constraintweights = NULL,
    compactness_metric = "fryer-holden",
    partisan_metric = "efficiency-gap",
    rngseed = NULL,
    maxiterrsg = 5000,
    adapt_lambda = FALSE,
    adapt_eprob = FALSE,
    exact_mh = FALSE,
    savename = NULL,
    verbose = TRUE,
    ncores = 1,
    tgt_min = 0.55,
    tgt_other = 0.25,
    rvote = NULL,
    dvote = NULL,
    minorityprop = NULL,
    adjobj,
    popvec,
    initcds,
    popcons,
    grouppopvec,
    countymembership,
    contiguitymap
)

Arguments

adj adjacency matrix, list, or object of class "SpatialPolygonsDataFrame."

total_pop A vector containing the populations of each geographic unit
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ndists</td>
<td>The number of congressional districts. The default is NULL.</td>
</tr>
<tr>
<td>init_plan</td>
<td>A vector containing the congressional district labels of each geographic unit. If not provided, random and contiguous congressional district assignments will be generated using redist.smc. To use the old behavior of generating with redist.rsg, provide init_plan = 'rsg'.</td>
</tr>
<tr>
<td>num_hot_steps</td>
<td>The number of steps to run the simulator at beta = 0. Default is 40000.</td>
</tr>
<tr>
<td>num_annealing_steps</td>
<td>The number of steps to run the simulator with linearly changing beta schedule. Default is 60000</td>
</tr>
<tr>
<td>num_cold_steps</td>
<td>The number of steps to run the simulator at beta = 1. Default is 20000.</td>
</tr>
<tr>
<td>eprob</td>
<td>The probability of keeping an edge connected. The default is 0.05.</td>
</tr>
<tr>
<td>lambda</td>
<td>The parameter determining the number of swaps to attempt each iteration of the algorithm. The number of swaps each iteration is equal to Pois(lambda) + 1. The default is 0.</td>
</tr>
<tr>
<td>pop_tol</td>
<td>The strength of the hard population constraint. pop_tol = 0.05 means that any proposed swap that brings a district more than 5% away from population parity will be rejected. The default is NULL.</td>
</tr>
<tr>
<td>group_pop</td>
<td>A vector of populations for some sub-group of interest. The default is NULL.</td>
</tr>
<tr>
<td>areasvec</td>
<td>A vector of precinct areas for discrete Polsby-Popper. The default is NULL.</td>
</tr>
<tr>
<td>counties</td>
<td>A vector of county membership assignments. The default is NULL.</td>
</tr>
<tr>
<td>borderlength_mat</td>
<td>A matrix of border length distances, where the first two columns are the indices of precincts sharing a border and the third column is its distance. Default is NULL.</td>
</tr>
<tr>
<td>ssdmat</td>
<td>A matrix of squared distances between geographic units. The default is NULL.</td>
</tr>
<tr>
<td>constraint</td>
<td>Which constraint to apply. Accepts any combination of compact, segregation, vra, population, similarity, partisan, minority, hinge, countysplit, or none (no constraint applied). The default is NULL.</td>
</tr>
<tr>
<td>constraintweights</td>
<td>The weights to apply to each constraint. Should be a vector the same length as constraint. Default is NULL.</td>
</tr>
<tr>
<td>compactness_metric</td>
<td>The compactness metric to use when constraining on compactness. Default is fryer-holden, the other implemented options are polsby-popper and edges-removed.</td>
</tr>
<tr>
<td>partisan_metric</td>
<td>The partisan metric to use when constraining on partisan metrics. Only implemented are &quot;efficiency-gap&quot; (default) and &quot;proportional-representation&quot;.</td>
</tr>
<tr>
<td>rngseed</td>
<td>Allows the user to set the seed for the simulations. Default is NULL.</td>
</tr>
<tr>
<td>maxiterrsg</td>
<td>Maximum number of iterations for random seed-and-grow algorithm to generate starting values. Default is 5000.</td>
</tr>
<tr>
<td>adapt_lambda</td>
<td>Whether to adaptively tune the lambda parameter so that the Metropolis-Hastings acceptance probability falls between 20% and 40%. Default is FALSE.</td>
</tr>
<tr>
<td>adapt_eprob</td>
<td>Whether to adaptively tune the edgecut probability parameter so that the Metropolis-Hastings acceptance probability falls between 20% and 40%. Default is FALSE.</td>
</tr>
</tbody>
</table>
exact_mh  Whether to use the approximate (0) or exact (1) Metropolis-Hastings ratio calculation for accept-reject rule. Default is FALSE.
savename  Filename to save simulations. Default is NULL.
verbose  Whether to print initialization statement. Default is TRUE.
ncores  The number of cores available to parallelize over. Default is 1.
tgt_min  The majority minority target percent as a decimal. Default is 0.55.
tgt_other  The remaining target percent as a decimal. Default is 0.25.
rvote  integer vector of votes for Republicans by precinct
dvote  integer vector of votes for Democrats by precinct
minorityprop  numeric vector of targeted minority proportions for the top districts with that proportion
adobj  Deprecated, use adj. An adjacency matrix, list, or object of class "SpatialPolygonsDataFrame."
popvec  A vector containing the populations of each geographic unit
initcds  Deprecated, use init_plan. A vector containing the congressional district labels of each geographic unit. The default is NULL. If not provided, random and contiguous congressional district assignments will be generated using redist.rsg.
popcons  Deprecated, use pop_tol. The strength of the hard population constraint. popcons = 0.05 means that any proposed swap that brings a district more than 5% away from population parity will be rejected. The default is NULL.
grouppopvec  Deprecated, use group_pop. A vector of populations for some sub-group of interest. The default is NULL.
countymembership  Deprecated, use counties. A vector of county membership assignments. The default is NULL.
contiguitymap  Deprecated. Use queens or rooks distance criteria for generating an adjacency list from a "SpatialPolygonsDataFrame" data type. Default is "rooks".

Value

list of class redist

---

redist.mcmc.mpi  *MCMC Redistricting Simulator using MPI*

**Description**

*redist.mcmc.mpi* is used to simulate Congressional redistricting plans using Markov Chain Monte Carlo methods.
Usage

redist.mcmc.mpi(
  adj,
  total_pop,
  nsims,
  ndists = NA,
  init_plan = NULL,
  loopscompleted = 0,
  nloop = 1,
  nthin = 1,
  eprob = 0.05,
  lambda = 0,
  pop_tol = NA,
  group_pop = NA,
  areasvec = NA,
  counties = NA,
  borderlength_mat = NA,
  ssdmat = NA,
  compactness_metric = "fryer-holden",
  rngseed = NA,
  constraint = NA,
  constraintweights = NA,
  betaseq = "powerlaw",
  betaseqlength = 10,
  adjswaps = TRUE,
  freq = 100,
  savename = NA,
  maxiterrsg = 5000,
  verbose = FALSE,
  adjobj,
  popvec,
  initcds,
  popcons,
  grouppopvec,
  countymembership,
  contiguitymap
)

Arguments

adj  An adjacency matrix, list, or object of class "SpatialPolygonsDataFrame."

total_pop  A vector containing the populations of each geographic unit.

nsims  The number of simulations run before a save point.

ndists  The number of congressional districts. The default is NULL.

init_plan  A vector containing the congressional district labels of each geographic unit. The default is NULL. If not provided, random and contiguous congressional district assignments will be generated using redist.rsg.
loopscompleted: Number of save points reached by the algorithm. The default is 0.
nloop: The total number of save points for the algorithm. The default is 1. Note that the total number of simulations run will be nsims * nloop.
nthin: The amount by which to thin the Markov Chain. The default is 1.
eprob: The probability of keeping an edge connected. The default is 0.05.
lambda: The parameter determining the number of swaps to attempt each iteration of the algorithm. The number of swaps each iteration is equal to Pois(lambda) + 1. The default is 0.
pop_tol: The strength of the hard population constraint. pop_tol = 0.05 means that any proposed swap that brings a district more than 5% away from population parity will be rejected. The default is NULL.
group_pop: A vector of populations for some sub-group of interest. The default is NULL.
areavec: A vector of precinct areas for discrete Polsby-Popper. The default is NULL.
counties: A vector of county membership assignments. The default is NULL.
borderlength_mat: A matrix of border length distances, where the first two columns are the indices of precincts sharing a border and the third column is its distance. Default is NULL.
ssdmat: A matrix of squared distances between geographic units. The default is NULL.
compactness_metric: The compactness metric to use when constraining on compactness. Default is fryer-holden, the other implemented option is polsby-popper.
rngseed: Allows the user to set the seed for the simulations. Default is NULL.
constraint: Which constraint to apply. Accepts any combination of compact, vra, population, similarity, or none (no constraint applied). The default is NULL.
constraintweights: The weights to apply to each constraint. Should be a vector the same length as constraint. Default is NULL.
betaseq: Sequence of beta values for tempering. The default is power-law (see Fifield et. al (2015) for details).
betaseqlength: Length of beta sequence desired for tempering. The default is 10.
adjswaps: Flag to restrict swaps of beta so that only values adjacent to current constraint are proposed. The default is TRUE.
freq: Frequency of between-chain swaps. Default to once every 100 iterations.
savename: Filename to save simulations. Default is NULL.
maxiterrsg: Maximum number of iterations for random seed-and-grow algorithm to generate starting values. Default is 5000.
verbose: Whether to print initialization statement. Default is TRUE.
adobj: Deprecated, use adj. An adjacency matrix, list, or object of class "SpatialPolygonsDataframe."
popvec: Deprecated, use total_pop. A vector containing the populations of each geographic unit.
**Details**

This function allows users to simulate redistricting plans using Markov Chain Monte Carlo methods. Several constraints corresponding to substantive requirements in the redistricting process are implemented, including population parity and geographic compactness. In addition, the function includes multiple-swap and parallel tempering functionality in MPI to improve the mixing of the Markov Chain.

**Value**

`redist.mcmc.mpi` returns an object of class "redist". The object `redist` is a list that contains the following components (the inclusion of some components is dependent on whether tempering techniques are used):

- **partitions**: Matrix of congressional district assignments generated by the algorithm. Each row corresponds to a geographic unit, and each column corresponds to a simulation.
- **distance_parity**: Vector containing the maximum distance from parity for a particular simulated redistricting plan.
- **mhdecisions**: A vector specifying whether a proposed redistricting plan was accepted (1) or rejected (0) in a given iteration.
- **mhprob**: A vector containing the Metropolis-Hastings acceptance probability for each iteration of the algorithm.
- **pparam**: A vector containing the draw of the p parameter for each simulation, which dictates the number of swaps attempted.
- **constraint_pop**: A vector containing the value of the population constraint for each accepted redistricting plan.
- **constraint_compact**: A vector containing the value of the compactness constraint for each accepted redistricting plan.
- **constraint_vra**: A vector containing the value of the vra constraint for each accepted redistricting plan.
constraint_similar
A vector containing the value of the similarity constraint for each accepted redistricting plan.

beta_sequence
A vector containing the value of beta for each iteration of the algorithm. Returned when tempering is being used.

mhdecisions_beta
A vector specifying whether a proposed beta value was accepted (1) or rejected (0) in a given iteration of the algorithm. Returned when tempering is being used.

mhprob_beta
A vector containing the Metropolis-Hastings acceptance probability for each iteration of the algorithm. Returned when tempering is being used.

References

Examples
## Not run:
# Cannot run on machines without Rmpi
data(f25)
data(f25_enum)
data(f25_adj)

## Code to run the simulations in Figure 4 in Fifield, Higgins, Imai and Tarr (2015)

## Get an initial partition
init_plan <- fl25_enum$plans[, 5118]

## Run the algorithm
redist.mcmc.mpi(adj = fl25_adj, total_pop = fl25$pop,
                init_plan = init_plan, nsims = 10000, savename = "test")

## End(Not run)

redist.parity
Calculates Maximum Deviation from Population Parity

Description
Computes the deviation from population parity from a plan. Higher values indicate that (at least) a single district in the map deviates from population parity. See Details.

Usage
redist.parity(plans, total_pop, ncores = 1, district_membership, population)

plan_parity(map, .data = cur_plans(), ...)
Arguments

- **plans**: A matrix with one row for each precinct and one column for each map. Required.
- **total_pop**: A numeric vector with the population for every precinct.
- **ncores**: Number of cores to use for parallel computing. Default is 1.
- **district_membership**: Deprecated, use plans. A matrix with one row for each precinct and one column for each map. Required.
- **population**: Deprecated, use total_pop. A numeric vector with the population for every precinct.
- **map**: A `redist_map` object.
- **.data**: A `redist_plans` object.
- **...**: Passed on to `redist.parity`.

Details

With a map with `pop` representing the populations of each district, the deviation from population parity is given as `max(abs(pop - parity) / parity)` where `parity = sum(pop) / length(pop)` is the population size for the average district. Therefore, the metric can be thought of as the maximum percent deviation from equal population. For example, a value of 0.03 in this metric indicates that all districts are within 3 percent of population parity.

Value

Numeric vector with the population parity for each column.

---

**redist.plot.adj**  
*Creates a Graph Overlay*

Description

Creates a Graph Overlay

Usage

```r
redist.plot.adj(
  shp = NULL,
  adj = NULL,
  plan = NULL,
  centroids = TRUE,
  drop = FALSE,
  plot_shp = TRUE,
  zoom_to = NULL,
  title = ""
)
```
Arguments

- **shp**: A SpatialPolygonsDataFrame or sf object. Required.
- **adj**: A zero-indexed adjacency list. Created with redist.adjacency if not supplied. Default is NULL.
- **plan**: A numeric vector with one entry for each precinct in shp. Used to remove edges that cross boundaries. Default is NULL. Optional.
- **centroids**: A logical indicating if centroids should be plotted. Default is TRUE.
- **drop**: A logical indicating if edges that cross districts should be dropped. Default is FALSE.
- **plot_shp**: A logical indicating if the shp should be plotted under the graph. Default is TRUE.
- **zoom_to**: An indexing vector of units to zoom the map to.
- **title**: A string title of plot. Defaults to empty string. Optional.

Value

- ggplot map

Examples

```r
data(iowa)
redist.plot.adj(shp = iowa, plan = iowa$cd_2010)
```

redist.plot.cores  Plot Cores

Description

Plot Cores

Usage

```
redist.plot.cores(shp, plan = NULL, core = NULL, lwd = 2)
```

Arguments

- **shp**: A SpatialPolygonsDataFrame or sf object. Required.
- **plan**: A numeric vector with one entry for each precinct in shp. Used to color the districts. Required.
- **core**: Required. integer vector produced by redist.identify.cores().
- **lwd**: Line width. Defaults to 2.

Value

- ggplot
redist.plot.distr_qtys

Plot quantities by district

Description

Plots a boxplot of a quantity of interest across districts, with districts optionally sorted by this quantity. Adds reference points for each reference plan, if applicable.

Usage

redist.plot.distr_qtys(
  plans,
  qty,
  sort = "asc",
  geom = "jitter",
  color_thresh = NULL,
  ...
)

Arguments

- **plans**: the `redist_plans` object.
- **qty**: <data-masking> the quantity of interest.
- **sort**: set to "asc" to sort districts in ascending order of qty (the default), "desc" for descending order, or FALSE or "none" for no sorting.
- **geom**: the geom to use in plotting the simulated districts: either "jitter" or "boxplot"
- **color_thresh**: if a number, the threshold to use in coloring the points. Plans with quantities of interest above the threshold will be colored differently than plans below the threshold.
- **...**: passed on to `geom_boxplot`

Value

A ggplot

Examples

```r
library(dplyr)
data(iowa)

iowa = redist_map(iowa, existing_plan=cd_2010, pop_tol=0.05, total_pop = pop)
plans = redist_smc(iowa, nsims=100, silent=TRUE)
plans %>%
  mutate(pct_dem = group_frac(iowa, dem_08, tot_08)) %>%
  redist.plot.distr_qtys(pct_dem)
```
redist.plot.hist  

Plot a histogram of a summary statistic

Description

Plots a histogram of a statistic of a `redist_plans` object, with a reference line for each reference plan, if applicable.

Usage

```r
redist.plot.hist(plans, qty, bins = NULL, ...) # S3 method for class 'redist_plans'
hist(x, qty, ...)
```

Arguments

- `plans`: the `redist_plans` object.
- `qty`: the statistic.
- `bins`: the number of bins to use in the histogram. Defaults to Freedman-Diaconis rule.
- `...`: passed on to `geom_histogram`
- `x`: the statistic.

Value

A ggplot

Examples

```r
library(dplyr)
data(iowa)
iowa = redist_map(iowa, existing_plan=cd_2010, pop_tol=0.05)
plans = redist_smc(iowa, nsims=100, silent=TRUE)
group_by(plans, draw) %>%
  summarize(pop_dev = max(abs(total_pop / mean(total_pop) - 1))) %>%
  redist.plot.hist(pop_dev)
```
redist.plot.majmin  Majority Minority Plots

Description
Majority Minority Plots

Usage
redist.plot.majmin(grouppercent, type = "hist", title = "")

Arguments
grouppercent  output from redist.group.percent
type  string in 'hist', 'toptwo', or 'box'
title  ggplot title

Value
ggplot

redist.plot.map  Plot a Map

Description
Create a ggplot map. It fills by plan or argument fill. If both are supplied, plan is used as the color and fill as the alpha parameter.

Usage
redist.plot.map(
  shp,
  adj,
  plan = NULL,
  fill = NULL,
  fill_label = "",
  zoom_to = NULL,
  boundaries = is.null(fill),
  title = ""
)
Arguments

- **shp**: A SpatialPolygonsDataFrame, sf object, or redist_map. Required.
- **adj**: A zero-indexed adjacency list. Created with redist.adjacency if not supplied and needed for coloring. Default is NULL.
- **plan**: `<data-masking>` A numeric vector with one entry for each precinct in shp. Used to color the districts. Default is NULL. Optional.
- **fill**: `<data-masking>` A numeric/integer vector with values to color the plot with. Optional.
- **fill_label**: A string title of plot. Defaults to the empty string
- **zoom_to**: `<data-masking>` An indexing vector of units to zoom the map to.
- **boundaries**: A logical indicating if precinct boundaries should be plotted.
- **title**: A string title of plot. Defaults to empty string. Optional.

Value

ggplot map

Examples

```r
data(iowa)
redist.plot.map(shp = iowa, plan = iowa$cd_2010)
iowa_map = redist_map(iowa, existing_plan = cd_2010)
redist.plot.map(iowa_map, fill=dem_08/tot_08, zoom_to=(cd_2010 == 1))
```

---

redist.plot.penalty  
*Visualize VRA Penalty*

Description

Plots the shape of the VRA Gibbs penalty.

Usage

```r
redist.plot.penalty(
  tgt_min = 0.55,
  tgt_other = 0.25,
  strength_vra = 2500,
  pow_vra = 1.5,
  limits = TRUE
)```
Arguments

- **tgt_min**: double, defaults to 0.55. The minority target percent.
- **tgt_other**: double, defaults to 0.25. The other group target percent.
- **strength_vra**: double, strength of the VRA constraint.
- **pow_vra**: double, exponent of the VRA constraint.
- **limits**: Whether to limit y axis to 0,500. Default is TRUE for comparability across values.

Details

This function allows you to plot the un-exponentiated VRA Gibbs penalty implemented as ‘vra’ within MCMC and ‘vra_old’ within Merge-Split and SMC. The function takes two key inputs, ‘tgt_min’ and ‘tgt_other’ which center the minimum penalty spots. A higher y-value indicates a higher penalty and incentivizes moving towards a spot with a lower y-value. The x-axis indicates the group population proportion in a given district. The default indicates the default settings within `redist.smc()`.

Value

ggplot

Examples

`redist.plot.penalty()`

---

**redist.plot.plans**

*Plot a district assignment*

Description

Plot a district assignment

Usage

`redist.plot.plans(plans, draws, geom, qty = NULL)`

Arguments

- **plans**: a `redist_plans` object.
- **draws**: the plan(s) to plot. Will match the draw column of `x`.
- **geom**: the `redist_map` geometry to use.
- **qty**: the quantity to plot. Defaults to the district assignment.
Value

A ggplot

Examples

```r
library(dplyr)
data(iowa)

iowa = redist_map(iowa, existing_plan=cd_2010, pop_tol=0.05, total_pop = pop)
plans = redist_smc(iowa, nsims=100, silent=TRUE)
redist.plot.plans(plans, c(1, 2, 3, 4), iowa)
```

---

redist.plot.scatter  
*Scatter plot of plan summary statistics*

Description

Makes a scatterplot of two quantities of interest across districts or plans.

Usage

```r
redist.plot.scatter(plans, x, y, ..., bigger = TRUE)
```

Arguments

- `plans`  
  the redist_plans object.
- `x`  
  <data-masking> the quantity to plot on the horizontal axis.
- `y`  
  <data-masking> the quantity to plot on the vertical axis.
- `...`  
  passed on to `geom_point`.
- `bigger`  
  if TRUE, make the point corresponding to the reference plan larger.

Value

A ggplot

Examples

```r
library(dplyr)
data(iowa)

iowa = redist_map(iowa, existing_plan=cd_2010, pop_tol=0.05, total_pop = pop)
plans = redist_smc(iowa, nsims=100, silent=TRUE)
```

```r
plans %>%
  mutate(comp = distr_compactness(iowa)) %>%
  group_by(draw) %>%
  summarize(pop_dev = max(abs(total_pop / mean(total_pop) - 1)),
```
comp = comp[1]) %>%
redist.plot.scatter(pop_dev, comp)

---

### redist.plot.varinfo  
**Static Variation of Information Plot**

**Description**

Static Variation of Information Plot

**Usage**

```
redist.plot.varinfo(plans, group_pop, total_pop, shp)
```

**Arguments**

- **plans**: matrix of district assignments
- **group_pop**: Required Population of subgroup being studied in each precinct.
- **total_pop**: Required. Population of each precinct.
- **shp**: sf dataframe

**Value**

patchworked ggplot

---

### redist.prep.enumpart  
**Prepares a run of the enumpart algorithm by ordering edges**

**Description**

_prepares a run of the enumpart algorithm by ordering edges

**Usage**

```
redist.prep.enumpart(adj, unordered_path, ordered_path, adjlist)
```

**Arguments**

- **adj**: zero indexed adjacency list
- **unordered_path**: valid path to output the unordered adjacency map to
- **ordered_path**: valid path to output the ordered adjacency map to
- **adjlist**: Deprecated, use adj. zero indexed adjacency list
redist.prep.polsbypopper

Value

0 on success

References


Examples

```r
## Not run:
tmp <- tempdir()
data(fl25)
adj <- redist.adjacency(fl25)
redist.prep.enumpart(adj = adj, unordered_path = paste0(tmp, '/unordered'),
                      ordered_path = paste0(tmp, '/ordered'))
## End(Not run)
```

redist.prep.polsbypopper

Prep Polsby Popper Perimeter Dataframe

Description

Prep Polsby Popper Perimeter Dataframe

Usage

`redist.prep.polsbypopper(shp, planarize = 3857, perim_path, ncores = 1)`

Arguments

- `shp`: A SpatialPolygonsDataFrame or sf object. Required unless "EdgesRemoved" and "logSpanningTree" with adjacency provided.
- `planarize`: a number, indicating the CRS to project the shapefile to if it is latitude-longitude based. Set to FALSE to avoid planarizing.
- `perim_path`: A path to save an Rds
- `ncores`: the number of cores to parallelize over

Value

A perimeter dataframe
Examples

```r
data(fl25)
perim_df <- redist.prep.polsbypopper(shp = fl25)
```

---

**redist.random.subgraph**

*Return a random subgraph of a shape*

---

**Description**

`random.subgraph` returns a random subset of the `shp` provided.

**Usage**

```r
redist.random.subgraph(shp, n, adj = NULL, adjacency)
```

**Arguments**

- `shp`: sf object or SpatialPolygonsDataFrame
- `n`: number of edges to sample. `n` must be a positive integer.
- `adj`: Optional. zero indexed adjacency list.
- `adjacency`: Deprecated, use `adj`. Optional. zero indexed adjacency list.

**Details**

Snowball sampling with backtracking

**Value**

sf dataframe with `n` rows

---

**redist.read.enumpart**

*Read Results from enumpart*

---

**Description**

Read Results from enumpart

**Usage**

```r
redist.read.enumpart(out_path, skip = 0, n_max = -1L)
```
Arguments

- **out_path**
  - out_path specified in `redist.run.enumpart`
- **skip**
  - number of lines to skip
- **n_max**
  - max number of lines to read

Value

district_membership matrix

References


Examples

```r
## Not run:
temp <- tempdir()
cds <- redist.read.enumpart(out_path = paste0(temp, '/quotesingle.Var
enumerated/quotesingle.Var'))
## End(Not run)
```

redist.reduce.adjacency

*Reduce Adjacency List*

Description

Tool to help reduce adjacency lists for analyzing subsets of maps.

Usage

`redist.reduce.adjacency(adj, keep_rows, adjacency)`

Arguments

- **adj**
  - A zero-indexed adjacency list. Required.
- **keep_rows**
  - row numbers of precincts to keep
- **adjacency**
  - Deprecated. Use `adj`. A zero-indexed adjacency list.

Value

zero indexed adjacency list with max value length(keep_rows) - 1

Examples

```r
data(fl25_adj)
redist.reduce.adjacency(fl25_adj, c(2, 3, 4, 6, 21))
```
redist.reindex  

Reorders district numbers

Description

Ensures that for each column in the plans object, the first district listed is 1, the second is 2, up to n districts. Assumes that all columns have the same number of districts as the first.

Usage

redist.reindex(plans, district_membership)

Arguments

- **plans**: A numeric vector (if only one map) or matrix with one row for each precinct and one column for each map.
- **district_membership**: Deprecated, use plans. A numeric vector (if only one map) or matrix with one row for each precinct and one column for each map.

Value

integer matrix

Examples

```r
cds <- matrix(c(rep(c(4L,5L,2L,1L,3L),5), rep(c(5L,4L,3L,2L,1L),2), rep(c(4L,5L,2L,1L,3L),3)), nrow = 25)
redist.reindex(cds)
```

redist.rsg  

Redistricting via Random Seed and Grow Algorithm

Description

redist.rsg generates redistricting plans using a random seed a grow algorithm. This is the non-compact districting algorithm described in Chen and Rodden (2013). The algorithm can provide start values for the other redistricting routines in this package.
Usage

redist.rsg(
  adj,
  total_pop,
  ndists,
  pop_tol,
  verbose = TRUE,
  maxiter = 5000,
  adj.list,
  population,
  thresh
)

Arguments

adj List of length N, where N is the number of precincts. Each list element is an integer vector indicating which precincts that precinct is adjacent to. It is assumed that precinct numbers start at 0.

total_pop numeric vector of length N, where N is the number of precincts. Each element lists the population total of the corresponding precinct, and is used to enforce population constraints.

ndists integer, the number of districts we want to partition the precincts into.

pop_tol numeric, indicating how close district population targets have to be to the target population before algorithm converges. thresh=0.05 for example means that all districts must be between 0.95 and 1.05 times the size of target.pop in population size.

verbose boolean, indicating whether the time to run the algorithm is printed.

maxiter integer, indicating maximum number of iterations to attempt before convergence to population constraint fails. If it fails once, it will use a different set of start values and try again. If it fails again, redist.rsg() returns an object of all NAs, indicating that use of more iterations may be advised.

adj.list Deprecated, use adj. List of length N, where N is the number of precincts. Each list element is an integer vector indicating which precincts that precinct is adjacent to. It is assumed that precinct numbers start at 0.

population Deprecated, use total_pop. numeric vector of list N, where N is the number of precincts. Each element lists the population total of the corresponding precinct, and is used to enforce population constraints.

thresh Deprecated, use pop_tol. numeric, indicating how close district population targets have to be to the target population before algorithm converges. thresh=0.05 for example means that all districts must be between 0.95 and 1.05 times the size of target.pop in population size.

Value

list, containing three objects containing the completed redistricting plan.

- **plan** A vector of length N, indicating the district membership of each precinct.
• district_list A list of length Ndistrict. Each list contains a vector of the precincts in the respective district.
• district_pop A vector of length Ndistrict, containing the population totals of the respective districts.

Author(s)

Benjamin Fifield, Department of Politics, Princeton University <benfifield@gmail.com>, https://benfifield.com/

Michael Higgins, Department of Statistics, Kansas State University <mikehiggins@k-state.edu>, http://www-personal.k-state.edu/~mikehiggins/

Kosuke Imai, Department of Politics, Princeton University <imai@harvard.edu>, http://imai.fas.harvard.edu

James Lo, <jameslo@princeton.edu>

Alexander Tarr, Department of Electrical Engineering, Princeton University <atarr@princeton.edu>

References


Examples

```r
### Real data example from test set
data(f25)
data(f25_adj)

res <- redist.rsg(adj = f25_adj, total_pop = f25$pop,
                  ndists = 3, pop_tol = 0.05)
```

---

redist.run.enumpart  
Runs the enumpart algorithm

Description

Runs the enumpart algorithm

Usage

```r
redist.run.enumpart(
  ordered_path,
  out_path,
  ndists = 2,
  all = TRUE,
  n = NULL,
  weight_path = NULL,
)```
lower = NULL,
upper = NULL,
options = NULL,
ndist
)

Arguments

ordered_path  Path used in redist.prep.enumpart (not including ".dat")
out_path  Valid path to output the enumerated districts
ndists  number of districts to enumerate
all  boolean. TRUE outputs all districts. FALSE samples n districts.
n  integer. Number of districts to output if all is FALSE. Returns districts selected from uniform random distribution.
weight_path  A path (not including ".dat") to a space-delimited file containing a vector of vertex weights, to be used along with lower and upper.
lower  A lower bound on each partition's total weight, implemented by rejection sampling.
upper  An upper bound on each partition's total weight.
options  Additional enumpart arguments. Not recommended for use.
ndist  Deprecated, use ndists. number of districts to enumerate

Value

0 on success

References


Examples

## Not run:
temp <- tempdir()
redist.run.enumpart(ordered_path = paste0(temp, '/ordered'),
out_path = paste0(temp, '/enumerated'))
## End(Not run)
DEFUNCT: Sample partitions using spanning trees

Description

redist.samplepart uses a spanning tree method to randomly sample redistricting plans. DEFUNCT, use redist_smc instead.

Usage

redist.samplepart(
  adj,
  ndists,
  total_pop = NULL,
  pop_filter = FALSE,
  pop_tol = 0.5,
  nsims = 1000,
  ncores = 1,
  adjobj,
  popvec,
  pop_constraint,
  contiguitymap = "rooks",
  nsamp,
  n_cores
)

Arguments

adj An adjacency list, matrix, or object of class SpatialPolygonsDataFrame.
ndists The desired number of congressional districts
total_pop Population vector for adjacency object. Provide if filtering by population
pop_filter Boolean. Whether or not to filter on population parity. Default is FALSE.
pop_tol Strength of population filter if filtering on distance to parity.
nsims Number of samples to draw. Default is 1000.
ncores Number of cores to parallelize over for parity calculation and compactness calculation. Default is 1.
adjob Deprecated, use adj. An adjacency list, matrix, or object of class SpatialPolygonsDataFrame.
popvec Deprecated, use total_pop. Population vector for adjacency object. Provide if filtering by population
pop_constraint Deprecated, use pop_tol. Strength of population filter if filtering on distance to parity.
contiguitymap Use queens or rooks distance criteria for generating an adjacency list from a "SpatialPolygonsDataFrame" data type. Default is "rooks".
ssamp Deprecated, use nsims. Number of samples to draw. Default is 1000.
n_cores Deprecated, use ncores. Number of cores to parallelize over for parity calculation and compactness calculation. Default is 1.
redist.smc_is_ci

Confidence Intervals for Importance Sampling Estimates

Description

Builds a confidence interval for a quantity of interest, given importance sampling weights.

Usage

redist.smc_is_ci(x, wgt, conf = 0.99)
Arguments

- **x**: A numeric vector containing the quantity of interest
- **wgt**: A numeric vector containing the nonnegative importance weights. Will be normalized automatically.
- **conf**: The confidence level for the interval.

Value

A two-element vector of the form [lower, upper] containing the importance sampling confidence interval.

redist.subset  Subset a shp

Description

Subsets a shp object along with its adjacency. Useful for running smaller analyses on pairs of districts. Provide population, ndists, pop_tol, and sub_ndists to get proper population parity constraints on subsets.

Usage

```r
redist.subset(
    shp,
    adj,
    keep_rows,
    total_pop,
    ndists,
    pop_tol,
    sub_ndists,
    adjacency,
    population,
    ndist,
    popcons,
    subndist
)
```

Arguments

- **shp**: An sf object
- **adj**: A zero-indexed adjacency list. Created with `redist.adjacency` if not supplied.
- **keep_rows**: row numbers of precincts to keep. Random submap selected if not supplied.
- **total_pop**: numeric vector with one entry for the population of each precinct.
- **ndists**: integer, number of districts in whole map
- **pop_tol**: The strength of the hard population constraint.
redist.uncoarsen

**Value**

A list containing the following components:

- **shp**
  - The subsetted shp object.
- **adj**
  - The subsetted adjacency list for shp.
- **keep_rows**
  - The indices of the rows kept.
- **sub_ndists**
  - The number of districts in the subset.
- **sub_pop_tol**
  - The new parity constraint for a subset.

---

**redist.uncoarsen**

*Uncoarsen a District Matrix*

**Description**

After a cores analysis or other form of coarsening, sometimes you need to be at the original geography level to be comparable. This takes in a coarsened matrix and uncoarsens it to the original level.

**Usage**

\[
\text{redist.uncoarsen}(\text{plans}, \text{district_membership}, \text{group_index})
\]

**Arguments**

- **plans**
  - A coarsened matrix of plans.
- **district_membership**
  - Deprecated, use plans. A coarsened district membership matrix.
- **group_index**
  - The index used to coarsen the shape.

**Value**

matrix
redist_flip provides a tidy interface to the methods in redist.mcmc.

Usage

```r
redist_flip(
  map,
  nsims,
  warmup = 0,
  init_plan,
  pop_tol,
  constraints = list(),
  nthin = 1,
  eprob = 0.05,
  lambda = 0,
  temper = FALSE,
  betaseq = "powerlaw",
  betaseq_length = 10,
  betaweights = NULL,
  adapt_lambda = FALSE,
  adapt_eprob = FALSE,
  exact_mh = FALSE,
  adjswaps = TRUE,
  init_name = NULL,
  verbose = TRUE
)
```

Arguments

- **map**: A `redist_map` object.
- **nsims**: The number of samples to draw, not including warmup.
- **warmup**: The number of warmup samples to discard.
- **init_plan**: A vector containing the congressional district labels of each geographic unit. The default is NULL. If not provided, a random initial plan will be generated using `redist.smc`. You can also request to initialize using `redist.rsg` by supplying 'rsg', though this is not recommended behavior.
- **pop_tol**: The strength of the hard population constraint. `pop_tol = 0.05` means that any proposed swap that brings a district more than 5% rejected. The default is `get_pop_tol(map)`. Providing an entry here ignores the pop_tol within the object provided to map.
- **constraints**: a list of constraints to implement. Can be created with `flip_constraints_helper`
nthin  The amount by which to thin the Markov Chain. The default is 1.

eprob  The probability of keeping an edge connected. The default is 0.05.

lambda  The parameter determining the number of swaps to attempt each iteration of the algorithm. The number of swaps each iteration is equal to Pois(lambda) + 1. The default is 0.

temper  Whether to use simulated tempering algorithm. Default is FALSE.

betaseq  Sequence of beta values for tempering. The default is powerlaw (see Fifield et al (2020) for details).

betaseq_length  Length of beta sequence desired for tempering. The default is 10.

betaweights  Sequence of weights for different values of beta. Allows the user to upweight certain values of beta over others. The default is NULL (equal weighting).

adapt_lambda  Whether to adaptively tune the lambda parameter so that the Metropolis-Hastings acceptance probability falls between 20% and 40%. Default is FALSE.

adapt_eprob  Whether to adaptively tune the edgecut probability parameter so that the Metropolis-Hastings acceptance probability falls between 20% and 40%. Default is FALSE.

exact_mh  Whether to use the approximate (FALSE) or exact (TRUE) Metropolis-Hastings ratio calculation for accept-reject rule. Default is FALSE.

adjswaps  Flag to restrict swaps of beta so that only values adjacent to current constraint are proposed. The default is TRUE.

init_name  a name for the initial plan, or FALSE to not include the initial plan in the output. Defaults to the column name of the existing plan, or "<init>" if the initial plan is sampled.

verbose  Whether to print initialization statement. Default is TRUE.

Details

This function allows users to simulate redistricting plans using the Markov Chain Monte Carlo methods of Fifield et al. Several constraints corresponding to substantive requirements in the redistricting process are implemented, including population parity and geographic compactness. In addition, the function includes multiple-swap and simulated tempering functionality to improve the mixing of the Markov Chain.

redist_flip allows for Gibbs constraints to be supplied via a list object passed to constraints. This is a change from the original redist.mcmc behavior to allow for a more straightforward function call when used within a pipe. A key difference between redist_flip and redist.mcmc is that redist_flip uses a small compactness constraint by default, as this improves the realism of the maps greatly and also leads to large speed improvements. (One of the most time consuming aspects of the mcmc/flip backend is checking for district shattering, which is slowed down even further by non-compact districts. As such, it is recommended that all flip simulations use at least a minimal compactness constraint, even if you weaken it from the default settings.) The default is a compact constraint using the edges-removed metric with a weight of 0.6. For very small maps (< 100 precincts), you will likely want to weaken (lower) this constraint, while for very large maps (>
5000 precincts), you will likely want to strengthen (increase) this constraint. Otherwise, for most maps, the default constraint should be a good starting place.

`redist_flip` samples from a known target distribution which can be described using the constraints. We recommend setting up the constraints for `redist_flip` with `flip_constraints_helper` to ensure that you are supplying the exact information needed. As a quick shorthand, if you want to run a simulation with no constraints at all, you can use `flip_constraints_helper(map = NULL, constraint = NULL)` and pass this to `constraints`. The following describes the constraints available. The general advice is to set weights in a way that gets between 20% on average, though more tuning advice is available in the vignette on using mcmc methods. Having too small of an acceptance rate indicates that the weights within constraints are too large and will impact sampling efficiency. If the Metropolis Hastings acceptance rate is too large, this may impact the target distribution, but may be fine for general exploration of possible maps.

There are currently 9 implemented constraint types, though `compact` and `partisan` have sub-types which are specified via a character `metric` within their respective list objects. The constraints are as follows:

- **compact** - biases the algorithm towards drawing more compact districts.
  - **weight** - the coefficient to put on the Gibbs constraint
  - **metric** - which metric to use. Must be one of `edges-removed` (the default), `polsby-popper`, `fryer-holden`, or `log-st`. Using Polsby Popper is generally not recommended, as `edges-removed` is faster and highly correlated. `log-st` can be used to match the target distribution of `redist_smc` or `redist_mergesplit`.
  - **areas** - Only used with `polsby-popper` - A vector of precinct areas.
  - **borderlength_mat** - Only used with `polsby-popper` - A matrix of precinct border lengths.
  - **ssdmat** - Only used with `fryer-holden` - A matrix of squared distances between precinct centroids.
  - **ssd_denom** - Only used with `fryer-holden` - a positive integer to use as the normalizing constant for the Relative Proximity Index.

- **population** - A Gibbs constraint to complement the hard population constraint set by `pop_tol`. This penalizes moves which move away from smaller population parity deviations. It is very useful when an `init_plan` sits outside of the desired `pop_tol` but there are substantive reasons to use that plan. This constraint uses the input to `total_pop`.
  - **weight** - the coefficient to put on the Gibbs constraint

- **countysplit** This is a Gibbs constraint to minimize county splits. Unlike SMC’s county constraint, this allows for more than `ndists - 1` splits and does not require that counties are contiguous.
  - **weight** - the coefficient to put on the Gibbs constraint

- **hinge** This uses the proportion of a group in a district and matches to the nearest target proportion, and then creates a penalty of $\sqrt{\max(0, \text{nearest.target} - \text{group.pct})}$. This is equivalent to the `vra-old` constraint in `redist_smc`.
  - **weight** - the coefficient to put on the Gibbs constraint
  - **minorityprop** - A numeric vector of minority proportions (between 0 and 1) which districts should aim to have

- **vra** This takes two target proportions of the presence of a minority group within a district. $(|\text{target.min} - \text{group.pct}||\text{target.other} - \text{group.pct}|)^{1.5}$ This is equivalent to the `vra-old` constraint in `redist_smc`. 
- weight - the coefficient to put on the Gibbs constraint
- target_min - the target minority percentage. Often, this is set to 0.55 to encourage minority majority districts.
- target_other - the target minority percentage for non majority minority districts.

- minority This constraint sorts the districts by the proportion of a group in a district and compares the highest districts to the entries of minorityprop. This takes the form \[ \sum_{i=1}^{n} \sqrt{|\text{group.pct}(i) - \text{minorityprop}(i)|} \] where \( n \) is the length of minorityprop input.
- weight - the coefficient to put on the Gibbs constraint
- minorityprop - A numeric vector of minority proportions (between 0 and 1) which districts should aim to have

- similarity This is a status-quo constraint which penalizes plans which are very different from the starting place. It is useful for local exploration.
- weight - the coefficient to put on the Gibbs constraint

- partisan This is a constraint which minimizes partisan bias, either as measured as the difference from proportional representation or as the magnitude of the efficiency gap.
- weight - the coefficient to put on the Gibbs constraint
- rvote - An integer vector of votes for Republicans or other party
- dvote - An integer vector of votes for Democrats or other party
- metric - which metric to use. Must be one of proportional-representation or efficiency-gap.

- segregation This constraint attempts to minimize the degree of dissimilarity between districts by group population.
- weight - the coefficient to put on the Gibbs constraint

Value

A `redist_plans` object containing the simulated plans.

References


Examples

data(iowa)
iowa_map <- redist_map(iowa, ndists = 4, existing_plan = cd_2010, total_pop = pop, pop_tol = 0.01)
sims <- redist_flip(map = iowa_map, nsims = 100)
Create a redist_map object.

Description

Sets up a redistricting problem.

Usage

redist_map(
    ..., 
    existing_plan = NULL, 
    pop_tol = NULL, 
    total_pop = c("pop", "population", "total_pop"), 
    ndists = NULL, 
    pop_bounds = NULL, 
    adj = NULL, 
    adj_col = "adj", 
    planarize = 3857 
)

as_redist_map(x)

Arguments

... column elements to be bound into a redist_map object or a single list or 
data.frame. These will be passed on to the tibble constructor.

existing_plan <tidy-select> the existing district assignment.

pop_tol <data-masking> the population tolerance. The percentage deviation from the 
average population will be constrained to be no more than this number. If 
existing_plan is provided, defaults to the parity of that plan; otherwise, de-
defaults to 0.01.

total_pop <tidy-select> the vector of precinct populations. Defaults to the pop, population, 
or total_pop columns, if one exists.

ndists <data-masking> the integer number of districts to partition the map into. Must 
be specified if existing_plan is not supplied.

pop_bounds <data-masking> more specific population bounds, in the form of c(lower, target, upper).

adj the adjacency graph for the object. Defaults to being computed from the data if 
it is coercible to a shapefile.

adj_col the name of the adjacency graph column

planarize a number, indicating the CRS to project the shapefile to if it is latitude-longitude 
based. Set to NULL or FALSE to avoid planarizing.

x an object to be coerced
Details

A redist_map object is a tibble which contains an adjacency list and additional information about the number of districts and population bounds. It supports all of the dplyr generics, and will adjust the adjacency list and attributes according to these functions; i.e., if we filter to a subset of units, the graph will change to subset to these units, and the population bounds will adjust accordingly. If an existing map is also attached to the object, the number of districts will also adjust. Subsetting with `\` and `\[` does not recompute graphs or attributes.

Other useful methods for redist_map objects:

- `merge_by`
- `get_adj`
- `plot.redist_map`

Value

A redist_map object

Examples

data(fl25)
d = redist_map(fl25, ndists=3, pop_tol=0.05, total_pop = pop)
dplyr::filter(d, pop >= 10e3)

redist_mergesplit

Merge-Split/Recombination MCMC Redistricting Sampler

Description

redist_mergesplit uses a Markov Chain Monte Carlo algorithm to generate congressional or legislative redistricting plans according to contiguity, population, compactness, and administrative boundary constraints. The MCMC proposal is the same as is used in the SMC sampler; it is similar but not identical to those used in the references. 1-level hierarchical Merge-split is supported through the counties parameter; unlike in the SMC algorithm, this does not guarantee a maximum number of county splits.

Usage

redist_mergesplit(
  map,
  nsims,
  warmup = floor(nsims/2),
  init_plan = NULL,
  counties = NULL,
  compactness = 1,
  constraints = list(),
  constraint_fn = function(m) rep(0, ncol(m)),
)
redist.mergesplit(
  adj,
  total_pop,
  nsims,
  ndists,
  pop_tol = 0.01,
  init_plan,
  counties,
  compactness = 1,
  constraints = list(),
  constraint_fn = function(m) rep(0, ncol(m)),
  adapt_k_thresh = 0.975,
  k = NULL,
  verbose = TRUE,
  silent = FALSE
)

Arguments

map
A redist_map object.

nsims
The number of samples to draw, including warmup.

warmup
The number of warmup samples to discard.

init_plan
The initial state of the map. If not provided, will default to the reference map of the map object, or if none exists, will sample a random initial state using redist_smc. You can also request a random initial state by setting init_plan="sample".

counties
A vector containing county (or other administrative or geographic unit) labels for each unit, which may be integers ranging from 1 to the number of counties, or a factor or character vector. If provided, the algorithm will generate maps tend to follow county lines. You may combine this with a Gibbs constraint on the number of county splits using the constraints parameter; see below. If no county-split considerations are desired, this parameter should be left blank.

compactness
Controls the compactness of the generated districts, with higher values preferring more compact districts. Must be nonnegative. See the 'Details' section for more information, and computational considerations.

constraints
A list containing information on constraints to implement. See the 'Details' section for more information.

constraint_fn
A function which takes in a matrix where each column is a redistricting plan and outputs a vector of log-weights, which will be added to the final weights.
adapt_k_thresh  The threshold value used in the heuristic to select a value \( k_i \) for each splitting iteration. Set to 0.9999 or 1 if the algorithm does not appear to be sampling from the target distribution. Must be between 0 and 1.

\( k \)  The number of edges to consider cutting after drawing a spanning tree. Should be selected automatically in nearly all cases.

init_name  a name for the initial plan, or FALSE to not include the initial plan in the output. Defaults to the column name of the existing plan, or "<init>" if the initial plan is sampled.

verbose  Whether to print out intermediate information while sampling. Recommended.

silent  Whether to suppress all diagnostic information.

adj  adjacency matrix, list, or object of class "SpatialPolygonsDataFrame."

total_pop  A vector containing the populations of each geographic unit

ndists  The number of congressional districts.

pop_tol  The desired population constraint. All sampled districts will have a deviation from the target district size no more than this value in percentage terms, i.e., pop_tol=0.01 will ensure districts have populations within 1% of the target population.

Details

This function draws samples from a specific target measure, controlled by the compactness, constraints, and constraint_fn parameters.

Higher values of compactness sample more compact districts; setting this parameter to 1 is computationally efficient and generates nicely compact districts.

The constraints parameter allows the user to apply several common redistricting constraints without implementing them by hand. This parameter is a list, which may contain any of the following named entries:

• status_quo: a list with two entries:
  – strength, a number controlling the tendency of the generated districts to respect the status quo, with higher values preferring more similar districts.
  – current, a vector containing district assignments for the current map.

• vra: a list with three entries:
  – strength, a number controlling the strength of the Voting Rights Act (VRA) constraint, with higher values prioritizing majority-minority districts over other considerations.
  – tgts_min, the target percentage(s) of minority voters in minority opportunity districts. Defaults to c(0.55).
  – min_pop, A vector containing the minority population of each geographic unit.

• incumbency: a list with two entries:
  – strength, a number controlling the tendency of the generated districts to avoid pairing up incumbents.
  – incumbents, a vector of precinct indices, one for each incumbent’s home address.

• splits: a list with one entry:
- **strength**, a number controlling the tendency of the generated districts to avoid splitting counties.

- **vra_old**: a list with five entries, which may be set up using `redist.constraint.helper`:
  - **strength**, a number controlling the strength of the Voting Rights Act (VRA) constraint, with higher values prioritizing majority-minority districts over other considerations.
  - **tgt_vra_min**, the target percentage of minority voters in minority opportunity districts. Defaults to 0.55.
  - **tgt_vra_other** The target percentage of minority voters in other districts. Defaults to 0.25, but should be set to reflect the total minority population in the state.
  - **pow_vra**, which controls the allowed deviation from the target minority percentage; higher values are more tolerant. Defaults to 1.5
  - **min_pop**, A vector containing the minority population of each geographic unit.

All constraints are fed into a Gibbs measure, with coefficients on each constraint set by the corresponding strength parameters. The status quo constraint adds a term measuring the variation of information distance between the plan and the reference, rescaled to [0, 1]. The **vra** constraint takes a list of target minority percentages. It matches each district to its nearest target percentage, and then applies a penalty of the form $\sqrt{\max(0, tgt - minpct)}$, summing across districts. This penalizes districts which are below their target population. The incumbency constraint adds a term counting the number of districts containing paired-up incumbents. The splits constraint adds a term counting the number of counties which contain precincts belonging to more than one district. The **vra_old** constraint adds a term of the form $(|tgt_vra_min - minpct||tgt_vra_other - minpct|^pow_vra)$, which encourages districts to have minority percentages near either $tgt_vra_min$ or $tgt_vra_other$. This can be visualized with `redist.plot.penalty`.

### Value

`redist.mergesplit` returns an object of class `redist_plans` containing the simulated plans.

`redist.mergesplit` returns an object of class list containing the simulated plans.

### References


### Examples

```r
data(fl25)
fl_map = redist_map(fl25, ndists=3, pop_tol=0.1)
sampled_basic = redist_mergesplit(fl_map, 10000)
sampled_constr = redist_mergesplit(fl_map, 10000, constraints=list(
  incumbency = list(strength=1000, incumbents=c(3, 6, 25))
)
data(fl25)
adj <- redist.adjacency(fl25)
out <- redist.mergesplit(adj = adj, total_pop = fl25$pop,
                       nsims = 5, ndists = 3, pop_tol = 0.1)

redist_plans

A set of redistricting plans

Description
A redist_plans object is essentially a data frame of summary information on each district and each plan, along with the matrix of district assignments and information about the simulation process used to generate the plans.

Usage
redist_plans(plans, map, algorithm, wgt = NULL, ...)

Arguments
plans a matrix with n_precinct columns and n_sims rows, or a single vector of precinct assignments.
map a redist_map object
algorithm the algorithm used to generate the plans (usually "smc" or "mcmc")
wgt the weights to use, if any.
... Other named attributes to set

Details
The first two columns of the data frame will be draw, a factor indexing the simulation draw, and district, an integer indexing the districts within a plan. The data frame will therefore have n_sims*ndists rows. As a data frame, the usual dplyr methods will work.

Other useful methods for redist_plans objects:
- add_reference
- subset_sampled
- subset_ref
- pullback
- number_by
- match_numbers
- is_county_split
• prec_assignment
• plan_distances
• get_plans_matrix
• get_plans_weights
• get_sampling_info
• as.matrix.redist_plans
• plot.redist_plans

Value

a new redist_plans object.

Examples

data(iowa)

iowa = redist_map(iowa, existing_plan=cd_2010, pop_tol=0.05, total_pop = pop)
rs_plan = redist.rsg(iowa$adj, iowa$pop, ndists=4, pop_tol=0.05)$plan
redist_plans(rs_plan, iowa, "rsg")

redist_quantile_trunc

Helper function to truncate importance weights

Description

Defined as \( \text{pmin}(x, \text{quantile}(x, 1 - \text{length}(x)^{-0.5})) \)

Usage

redist_quantile_trunc(x)

Arguments

x the weights

Value

numeric vector

Examples

redist_quantile_trunc(c(1,2,3,4))
redist_shortburst  Redistricting Optimization through Short Bursts

Description

This function uses redist_mergesplit() to optimize a redistrict plan according to a user-provided criteria. It does so by running the Markov chain for "short bursts" of usually 10 iterations, and then starting the chain anew from the best plan in the burst, according to the criteria. This implements the ideas in the below-referenced paper, "Voting Rights, Markov Chains, and Optimization by Short Bursts."

Usage

```r
redist_shortburst(
  map,
  score_fn = NULL,
  stop_at = NULL,
  burst_size = ifelse(backend == "mergesplit", 10L, 50L),
  max_bursts = 500L,
  maximize = TRUE,
  init_plan = NULL,
  counties = NULL,
  compactness = 1,
  adapt_k_thresh = 0.975,
  return_all = TRUE,
  backend = "mergesplit",
  flip_lambda = 0,
  flip_eprob = 0.05,
  flip_constraints = list(),
  verbose = TRUE
)
```

Arguments

- **map**
  A redist_map object.

- **score_fn**
  A function which takes a matrix of plans and returns a score for each plan. Can also be a purrr-style anonymous function. See ?scorers for some function factories for common scoring rules.

- **stop_at**
  A threshold to stop optimization at.

- **burst_size**
  The size of each burst. 10 is recommended for mergesplit and 50 for flip.

- **max_bursts**
  The maximum number of bursts to run before returning.

- **maximize**
  If TRUE, try to maximize the score; otherwise, try to minimize it.

- **init_plan**
  The initial state of the map. If not provided, will default to the reference map of the map object, or if none exists, will sample a random initial state using redist_smc. You can also request a random initial state by setting init_plan="sample".
A vector containing county (or other administrative or geographic unit) labels for each unit, which may be integers ranging from 1 to the number of counties, or a factor or character vector. If provided, the algorithm will only generate maps which split up to \( \text{ndists-1} \) counties. If no county-split constraint is desired, this parameter should be left blank.

Controls the compactness of the generated districts, with higher values preferring more compact districts. Must be non-negative. See `redist_mergesplit` for more information.

The threshold value used in the heuristic to select a value \( k_i \) for each splitting iteration. Set to 0.9999 or 1 if the algorithm does not appear to be sampling from the target distribution. Must be between 0 and 1.

Whether to return all the Recommended for monitoring purposes.

the MCMC algorithm to use within each burst, either "mergesplit" or "flip".

The parameter determining the number of swaps to attempt each iteration of flip mcmc. The number of swaps each iteration is equal to \( \text{Pois}(\lambda) + 1 \). The default is 0.

The probability of keeping an edge connected in flip mcmc. The default is 0.05.

A list of constraints to use for flip mcmc. Can be created with `flip_constraints_helper`. Defaults to an edges-removed compactness constraint with weight 0.6.

Whether to print out intermediate information while sampling. Recommended for monitoring purposes.

a `redist_plans` object containing the final best plan (or the best plans after each burst, if `return_all=TRUE`.


```r
data(iowa)

iowa_map = redist_map(iowa, existing_plan=cd_2010, pop_tol=0.01)
redist_shortburst(iowa_map, scorer_frac_kept(iowa_map), max_bursts=50)
redist_shortburst(iowa_map, ~ 1 - scorer_frac_kept(iowa_map)(.), max_bursts=50)
```
**redist_smc**  
*SMC Redistricting Sampler*

**Description**

*redist_smc* uses a Sequential Monte Carlo algorithm to generate nearly independent congressional or legislative redistricting plans according to contiguity, population, compactness, and administrative boundary constraints.

**Usage**

```r
redist_smc(  
  map,  
  nsims,  
  counties = NULL,  
  compactness = 1,  
  constraints = list(),  
  resample = TRUE,  
  constraint_fn = function(m) rep(0, ncol(m)),  
  adapt_k_thresh = 0.975,  
  seq_alpha = 0.2 + 0.3 * compactness,  
  truncate = (compactness != 1),  
  trunc_fn = redist_quantile_trunc,  
  pop_temper = 0,  
  ref_name = NULL,  
  verbose = TRUE,  
  silent = FALSE  
)
```

```r
redist.smc(  
  adj,  
  total_pop,  
  nsims,  
  ndists,  
  counties = NULL,  
  pop_tol = 0.01,  
  pop_bounds = NULL,  
  compactness = 1,  
  constraints = list(),  
  resample = TRUE,  
  constraint_fn = function(m) rep(0, ncol(m)),  
  adapt_k_thresh = 0.975,  
  seq_alpha = 0.2 + 0.2 * compactness,  
  truncate = (compactness != 1),  
  trunc_fn = function(x) pmin(x, 0.01 * nsims^0.4),  
  pop_temper = 0,  
  verbose = TRUE,  
  silent = FALSE  
)
```
Arguments

map A redist_map object.
nsims The number of samples to draw.
counties A vector containing county (or other administrative or geographic unit) labels for each unit, which may be integers ranging from 1 to the number of counties, or a factor or character vector. If provided, the algorithm will only generate maps which split up to ndists-1 counties. If no county-split constraint is desired, this parameter should be left blank.
compactness Controls the compactness of the generated districts, with higher values preferring more compact districts. Must be nonnegative. See the ’Details’ section for more information, and computational considerations.
constraints A list containing information on constraints to implement. See the ’Details’ section for more information.
resample Whether to perform a final resampling step so that the generated plans can be used immediately. Set this to FALSE to perform direct importance sampling estimates, or to adjust the weights manually.
constraint_fn A function which takes in a matrix where each column is a redistricting plan and outputs a vector of log-weights, which will be added to the final weights.
adapt_k_thresh The threshold value used in the heuristic to select a value $k_i$ for each splitting iteration. Set to 0.9999 or 1 if the algorithm does not appear to be sampling from the target distribution. Must be between 0 and 1.
seq_alpha The amount to adjust the weights by at each resampling step; higher values prefer exploitation, while lower values prefer exploration. Must be between 0 and 1.
truncate Whether to truncate the importance sampling weights at the final step by trunc_fn. Recommended if compactness is not 1. Truncation only applied if resample=TRUE.
trunc_fn A function which takes in a vector of weights and returns a truncated vector. If loo package is installed (strongly recommended), will default to Pareto-smoothed Importance Sampling (PSIS) rather than naive truncation.
pop_temper The strength of the automatic population tempering.
ref_name a name for the existing plan, which will be added as a reference plan, or FALSE to not include the initial plan in the output. Defaults to the column name of the existing plan.
verbose Whether to print out intermediate information while sampling. Recommended.
silent Whether to suppress all diagnostic information.
adj An adjacency matrix, list, or object of class "SpatialPolygonsDataFrame."
total_pop A vector containing the populations of each geographic unit.
The number of districts in each redistricting plan.

The desired population constraint. All sampled districts will have a deviation from the target district size no more than this value in percentage terms, i.e., pop_tol=0.01 will ensure districts have populations within 1% of the target population.

A numeric vector with three elements c(lower, target, upper) providing more precise population bounds for the algorithm. Districts will have population between lower and upper, with a goal of target. If set, overrides popcons.

Deprecated, use adj. An adjacency matrix, list, or object of class "SpatialPolygonsDataFrame."

Deprecated, use total_pop. A vector containing the populations of each geographic unit.

The desired population constraint. All sampled districts will have a deviation from the target district size no more than this value in percentage terms, i.e., popcons=0.01 will ensure districts have populations within 1% of the target population.

This function draws nearly-independent samples from a specific target measure, controlled by the popcons, compactness, constraints, and constraint_fn parameters.

Key to ensuring good performance is monitoring the efficiency of the resampling process at each SMC stage. Unless silent=F, this function will print out the effective sample size of each resampling step to allow the user to monitor the efficiency. If verbose=T the function will also print out information on the ki values automatically chosen and the acceptance rate (based on the population constraint) at each step.

Higher values of compactness sample more compact districts; setting this parameter to 1 is computationally efficient and generates nicely compact districts. Values of other than 1 may lead to highly variable importance sampling weights. By default these weights are truncated using redist_quantile_trunc to stabilize the resulting estimates, but if truncation is used, a specific truncation function should probably be chosen by the user.

The constraints parameter allows the user to apply several common redistricting constraints without implementing them by hand. This parameter is a list, which may contain any of the following named entries:

- **status_quo**: a list with two entries:
  - strength, a number controlling the tendency of the generated districts to respect the status quo, with higher values preferring more similar districts.
  - current, a vector containing district assignments for the current map.

- **vra**: a list with three entries:
  - strength, a number controlling the strength of the Voting Rights Act (VRA) constraint, with higher values prioritizing majority-minority districts over other considerations.
  - tgts_min, the target percentage(s) of minority voters in minority opportunity districts. Defaults to c(0.55).
  - min_pop, A vector containing the minority population of each geographic unit.
incumbency: a list with two entries:
  – strength, a number controlling the tendency of the generated districts to avoid pairing up incumbents.
  – incumbents, a vector of precinct indices, one for each incumbent’s home address.

• vra_old: a list with five entries, which may be set up using redist.constraint.helper:
  – strength, a number controlling the strength of the Voting Rights Act (VRA) constraint, with higher values prioritizing majority-minority districts over other considerations.
  – tgt_vra_min, the target percentage of minority voters in minority opportunity districts. Defaults to 0.55.
  – tgt_vra_other The target percentage of minority voters in other districts. Defaults to 0.25, but should be set to reflect the total minority population in the state.
  – pow_vra, which controls the allowed deviation from the target minority percentage; higher values are more tolerant. Defaults to 1.5
  – min_pop, A vector containing the minority population of each geographic unit.

All constraints are fed into a Gibbs measure, with coefficients on each constraint set by the corresponding strength parameters. The status_quo constraint adds a term measuring the variation of information distance between the plan and the reference, rescaled to [0, 1]. The vra constraint takes a list of target minority percentages. It matches each district to its nearest target percentage, and then applies a penalty of the form \( \sqrt{\max(0, tgt - minpct)} \), summing across districts. This penalizes districts which are below their target population. The incumbency constraint adds a term counting the number of districts containing paired-up incumbents. The vra_old constraint adds a term of the form \( \left| tgtvramin - minpct \right| \left| tgtvraother - minpct \right|^{powvra} \), which encourages districts to have minority percentages near either tgt_vra_min or tgt_vra_other. This can be visualized with redist.plot.penalty.

Value

redist_smc returns an object of class redist_plans containing the simulated plans.
redist.smc returns an object of class redist, which is a list containing the following components:

- aList: The adjacency list used to sample
- cdvec: The matrix of sampled plans. Each row is a geographical unit, and each column is a sample.
- wgt: The importance sampling weights, normalized to sum to 1.
- orig_wgt: The importance sampling weights before resampling or truncation, normalized to have mean 1.
- nsims: The number of plans sampled.
- pct_dist_parity: The population constraint.
- compactness: The compactness constraint.
- counties: The computed constraint options list (see above).
- maxdev: The maximum population deviation of each sample.
- total_pop: The provided vector of unit populations.
- counties: The provided county vector.
adapt_k_thresh  The provided control parameter.
seq_alpha      The provided control vector.
algorithm      The algorithm used, here "smc".

References


Examples

```r
set.seed(1)
data(fl25)
fl_map = redist_map(fl25, ndists=3, pop_tol=0.1)
sampled_basic = redist_smc(fl_map, 10000)
sampled_constr = redist_smc(fl_map, 10000, constraints=list(
               incumbency = list(strength=100, incumbents=c(3, 6, 25))
))

data(fl25)
data(fl25_adj)
data(fl25_enum)
sampled_basic = redist.smc(fl25_adj, fl25$pop,
               nsims=10000, ndists=3, pop_tol=0.1)
sampled_constr = redist.smc(fl25_adj, fl25$pop,
               nsims=10000, ndists=3, pop_tol=0.1,
               constraints=list(
               status quo = list(strength=10, current=fl25_enum$plans[,5118]),
               incumbency = list(strength=100, incumbents=c(3, 6, 25))
))
```

scorer-arith  Scoring function arithmetic
Description

`redist_scorer` functions may be multiplied by constants and/or added together to form linear combinations.

Usage

```r
## S3 method for class 'redist_scorer'
x * fn2
## S3 method for class 'redist_scorer'
fn1 + fn2
## S3 method for class 'redist_scorer'
fn1 - fn2
```

Arguments

- `x` a numeric
- `fn2` a `redist_scorer` function, from `['scorers']`
- `fn1` a `redist_scorer` function, from `['scorers']`

Value

function of class `redist_scorer`

---

### scorer_group_pct

**Scoring functions for `redist_shortburst`**

Description

The output of these functions may be passed into `redist_shortburst()` as `score_fn`. Scoring functions have type `redist_scorer` and may be combined together using basic arithmetic operations.

Usage

```r
scorer_group_pct(map, group_pop, total_pop, k = 1)
scorer_pop_dev(map)
scorer_splits(map, counties)
scorer_frac_kept(map)
scorer_polsby_popper(map, perim_df = NULL, areas = NULL, m = 1)
scorer_status_quo(map, existing_plan = get_existing(map))
```
scorer_group_pct

Arguments

- **map**: A `redist_map` object.
- **group_pop**: A numeric vector with the population of the group for every precinct.
- **total_pop**: A numeric vector with the population for every precinct.
- **k**: the k-th from the top group fraction to return as the score.
- **counties**: A numeric vector with an integer from 1:n_counties
- **perim_df**: perimeter distance dataframe from `redist.prep.polsbypopper`
- **areas**: area of each precinct (i.e., `st_area(map)`)
- **m**: the m-th from the bottom Polsby Popper to return as the score. Defaults to 1, the minimum Polsby Popper score
- **existing_plan**: A vector containing the current plan.

Details

Function details:

- **scorer_group_pct** returns the k-th top group percentage across districts. For example, if the group is Democratic voters and k=3, then the function returns the 3rd-highest fraction of Democratic voters across all districts. Can be used to target k VRA districts or partisan gerrymanders.
- **scorer_pop_dev** returns the maximum population deviation within a plan. Smaller values are closer to population parity, so use `maximize=FALSE` with this scorer.
- **scorer_splits** returns the fraction of counties that are split within a plan. Higher values have more county splits, so use `maximize=FALSE` with this scorer.
- **scorer_frac_kept** returns the fraction of edges kept in each district. Higher values mean more compactness.
- **scorer_polsby_popper** returns the m-th Polsby Popper score within a plan. Higher scores correspond to more compact districts. Use `m=ndists/2` to target the median compactness, `m=1` to target the minimum compactness.
- **scorer_status_quo** returns 1 - the rescaled variation of information distance between the plan and the existing_plan. Larger values indicate the plan is closer to the existing plan.

Value

A scoring function of class `redist_scorer`. Single numeric value, where larger values are better for `frac_kept, group_pct, and polsby_popper` and smaller values are better for `splits` and `pop_dev`.

Examples

```r
data(iowa)
iowa_map = redist_map(iowa, existing_plan=cd_2010, pop_tol=0.05, total_pop = pop)
scorer_frac_kept(iowa_map)
```
segregation_index

scorer_status_quo(iowa_map)
scorer_group_pct(iowa_map, dem_08, tot_08, k=2)
1.5*scorer_frac_kept(iowa_map) + 0.4*scorer_status_quo(iowa_map)

segregation_index  Segregation index calculation for MCMC redistricting.

Description

redist.segcalc calculates the dissimilarity index of segregation (see Massey & Denton 1987 for more details) for a specified subgroup under any redistricting plan.

Usage

segregation_index(
  map,
  group_pop,
  total_pop = map[[attr(map, "pop_col")]],
  .data = cur_plans()
)

redist.segcalc(plans, group_pop, total_pop, algout, grouppop, fullpop)

Arguments

map  a redist_map object

group_pop  A vector of populations for some subgroup of interest.

total_pop  A vector containing the populations of each geographic unit.

.data  a redist_plans object

plans  A matrix of congressional district assignments or a redist object.

algout  Deprecated. Use plans. A matrix of congressional district assignments or a redist object.

grouppop  Deprecated. Use group_pop. A vector of populations for some subgroup of interest.

fullpop  Deprecated. Use total_pop. A vector containing the populations of each geographic unit.

Value

redist.segcalc returns a vector where each entry is the dissimilarity index of segregation (Massey & Denton 1987) for each redistricting plan in algout.
References


Examples

data(fl25)
data(fl25_enum)
data(fl25_adj)

## Get an initial partition
init_plan <- fl25_enum$plans[, 5118]

## 25 precinct, three districts - no pop constraint ##
alg_253 <- redist.flip(adj = fl25_adj, total_pop = fl25$pop, init_plan = init_plan, nsims = 10000)

## Get Republican Dissimilarity Index from simulations
rep_dmi_253 <- redist.segcalc(alg_253, fl25$mccain, fl25$pop)

subset_sampled

Subset to sampled or reference draws

Description

Subset to sampled or reference draws

Usage

subset_sampled(plans)
subset_ref(plans)

Arguments

plans the redist_plans object

Value

a redist_plans object, with only rows corresponding to simulated (or reference) draws remaining.
Index

* analyze
  add_reference, 5
  classify_plans, 6
  compare_plans, 6
  competitiveness, 8
  county_splits, 9
  distr_compactness, 10
  get_plans_matrix, 22
  get_plans_weights, 22
  group_frac, 24
  imp_confint, 25
  is_county_split, 27
  last_plan, 28
  match_numbers, 30
  number_by, 31
  partisan_metrics, 32
  plan_distances, 34
  plot.redist_classified, 36
  prec_assignment, 38
  prec_cooccurrence, 38
  pullback, 41
  redist.parity, 80
  redist.plans, 110
  segregation_index, 121
  subset_sampled, 122
* data
  f125, 13
  f1250, 14
  f125_adj, 15
  f125_enum, 16
  f170, 17
  iowa, 26
* enumerate
  redist.calc.frontier.size, 42
  redist.enumerate, 55
  redist.enumpart, 56
  redist.init.enumpart, 61
  redist.prep.enumpart, 89
  redist.read.enumpart, 91
  redist.run.enumpart, 95
* package
  redist-package, 4
* plot
  plot.redist_map, 36
  plot.redist_plans, 37
  redist.diagplot, 53
  redist.plot.adj, 81
  redist.plot.cores, 82
  redist.plot.distr_qtys, 83
  redist.plot.hist, 84
  redist.plot.majmin, 85
  redist.plot.map, 85
  redist.plot.plans, 87
  redist.plot.scatter, 88
  redist.plot.varinfo, 89
* post
  redist.combine, 44
  redist.combine.anneal, 46
  redist.combine.mpi, 47
  redist.ipw, 62
  redist.smc_is_ci, 98
  redist.uncoarsen, 100
* prepare
  freeze, 19
  get_adj, 20
  get_existing, 21
  get_pop_tol, 23
  get_target, 24
  is_contiguous, 27
  make_cores, 28
  merge_by, 30
  plot.redist_map, 36
  redist.coarsen.adjacency, 43
  redist.constraint.helper, 49
  redist.county.id, 50
  redist.county.relabel, 51
  redist.find.target, 58
  redist.findparams, 59
redist.plot.penalty, 86
redist.reduce.adjacency, 92
redist.sink.plan, 98
redist.subset, 99
redist_map, 105
scorer-arith, 118
scorer_group_pct, 119

* simulate*
get_mh_acceptance_rate, 21
get_sampling_info, 23
redist.crgs, 51
redist.mcmc, 65
redist.mcmc.anneal, 72
redist.mcmc.mpi, 76
redist.rsg, 93
redist.flip, 101
redist.mergesplit, 106
redist.shortburst, 112
redist.smc, 114
.redist.scorer (scorer-arith), 118
+.redist.scorer (scorer-arith), 118
-.redist.scorer (scorer-arith), 118
?scorers, 112

add_reference, 5, 110
as.matrix.redist, 5
as.matrix.redist.plans, 111
as.matrix.redist.plans
  (get_plans_matrix), 22
as_redist_map (redist_map), 105

classify_plans, 6
classify_plans(), 36
compare_plans, 6
compare_plans(), 36
competitiveness, 8
county_splits, 9
distr_compactness, 10
dplyr, 110
f125, 13, 16
f1250, 14
f125_adj, 13, 15, 15
f125_enum, 13, 15, 16
f170, 17
flip_constraints_helper, 18, 103
freeze, 19
geom_boxplot, 83
gem_histogram, 84
gem_point, 88
get_adj, 20, 106
get_existing, 21
get_mh_acceptance_rate, 21
get_plans_matrix, 22, 111
get_plans_weights, 22, 111
get_pop_tol, 23
get_sampling_info, 23, 111
get_target, 24
group_frac, 24
hclust(), 6
hist.redist.plans (redist.plot.hist), 84
imp_confint, 25
iowa, 26
is.contiguous, 27
is_county_split, 27, 110
last_plan, 28
loo, 115
make.cores, 28
match_numbers, 30, 110
merge_by, 30, 41, 106
number_by, 31, 110

partisan_metrics, 32
plan_distances, 34, 111
plan_distances(), 6
plan_parity (redist.parity), 80
plot.redist_classified, 36
plot.redist_classified(), 6
plot.redist_map, 36, 106
plot.redist.plans, 37, 111
prec_assignment, 38, 111
prec_cooccurrence, 38
print.redist, 39
print.redist_classified, 39
print.redist_map, 40
print.redist.plans, 40
pullback, 41, 110

redist (redist-package), 4
redist-package, 4
redist.adjacency, 41
redist.calc.frontier.size, 42
redist.choropleth, 42
<table>
<thead>
<tr>
<th>Function</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>redist.coarsen.adjacency</td>
<td>43</td>
</tr>
<tr>
<td>redist.combine</td>
<td>44</td>
</tr>
<tr>
<td>redist.combine.anneal</td>
<td>46</td>
</tr>
<tr>
<td>redist.combine.mpi</td>
<td>47</td>
</tr>
<tr>
<td>redist.compactness</td>
<td>10</td>
</tr>
<tr>
<td>redist.competitiveness</td>
<td>8</td>
</tr>
<tr>
<td>redist.constraint.helper</td>
<td>49, 109, 117</td>
</tr>
<tr>
<td>redist.county.id</td>
<td>50</td>
</tr>
<tr>
<td>redist.county.relabel</td>
<td>51</td>
</tr>
<tr>
<td>redist.crsg</td>
<td>51</td>
</tr>
<tr>
<td>redist.diagplot</td>
<td>53</td>
</tr>
<tr>
<td>redist.distances</td>
<td>34</td>
</tr>
<tr>
<td>redist.enumerate</td>
<td>55</td>
</tr>
<tr>
<td>redist.enumpart</td>
<td>56</td>
</tr>
<tr>
<td>redist.find.target</td>
<td>58</td>
</tr>
<tr>
<td>redist.findparams</td>
<td>59</td>
</tr>
<tr>
<td>redist.flip</td>
<td>65</td>
</tr>
<tr>
<td>redist.flip.anneal</td>
<td>72</td>
</tr>
<tr>
<td>redist.freeze</td>
<td>19</td>
</tr>
<tr>
<td>redist.group.percent</td>
<td>24</td>
</tr>
<tr>
<td>redist.identify.cores</td>
<td>28</td>
</tr>
<tr>
<td>redist.init.enumpart</td>
<td>61</td>
</tr>
<tr>
<td>redist.ipw</td>
<td>62</td>
</tr>
<tr>
<td>redist.map</td>
<td>64</td>
</tr>
<tr>
<td>redist.mcmc</td>
<td>65, 101</td>
</tr>
<tr>
<td>redist.mcmc.anneal</td>
<td>72</td>
</tr>
<tr>
<td>redist.mcmc.mpi</td>
<td>76</td>
</tr>
<tr>
<td>redist.mergesplit</td>
<td>106, 113</td>
</tr>
<tr>
<td>redist.prep.enumpart</td>
<td>89</td>
</tr>
<tr>
<td>redist.plans</td>
<td>7–10, 18, 20, 23, 25, 29, 31, 32, 36, 81, 101, 105, 107, 110, 112, 115, 120, 121</td>
</tr>
<tr>
<td>redist.reindex</td>
<td>93</td>
</tr>
<tr>
<td>redist.rsg</td>
<td>93</td>
</tr>
<tr>
<td>redist.run.enumpart</td>
<td>95</td>
</tr>
<tr>
<td>redist.samplepart</td>
<td>97</td>
</tr>
<tr>
<td>redist.segcalc</td>
<td>121</td>
</tr>
<tr>
<td>redist.sink.plan</td>
<td>98</td>
</tr>
<tr>
<td>redist.smc</td>
<td>114</td>
</tr>
<tr>
<td>redist.smc_is_ci</td>
<td>98</td>
</tr>
<tr>
<td>redist.splits</td>
<td>9</td>
</tr>
<tr>
<td>redist.subset</td>
<td>99</td>
</tr>
<tr>
<td>redist.uncoarsen</td>
<td>100</td>
</tr>
<tr>
<td>redist_flip</td>
<td>101</td>
</tr>
<tr>
<td>redist_map</td>
<td>8–10, 18, 20, 23, 25, 29, 31, 32, 36, 81, 101, 105, 107, 110, 112, 115, 120, 121</td>
</tr>
<tr>
<td>redist_merge_split</td>
<td>106, 113</td>
</tr>
<tr>
<td>redist_merge_split()</td>
<td>112</td>
</tr>
<tr>
<td>redist_plans</td>
<td>7–10, 25, 26, 28, 32, 36–38, 81, 84, 104, 109, 110, 117, 121</td>
</tr>
<tr>
<td>redist_quantile_trunc</td>
<td>111, 116</td>
</tr>
<tr>
<td>redist_shortburst</td>
<td>112</td>
</tr>
<tr>
<td>redist_smc</td>
<td>97, 107, 112, 114</td>
</tr>
<tr>
<td>scorer_arith</td>
<td>118</td>
</tr>
<tr>
<td>scorer_frac_kept</td>
<td>119</td>
</tr>
<tr>
<td>scorer_group_pct</td>
<td>119</td>
</tr>
<tr>
<td>scorer_polsby_popper</td>
<td>(scorer_group_pct), 119</td>
</tr>
<tr>
<td>scorer_pop_dev</td>
<td>(scorer_group_pct), 119</td>
</tr>
<tr>
<td>scorer_splits</td>
<td>(scorer_group_pct), 119</td>
</tr>
<tr>
<td>scorer_status_quo</td>
<td>(scorer_group_pct), 119</td>
</tr>
<tr>
<td>scorers</td>
<td>(scorer_group_pct), 119</td>
</tr>
<tr>
<td>segregation_index</td>
<td>121</td>
</tr>
<tr>
<td>set_adj</td>
<td>20</td>
</tr>
<tr>
<td>set_pop_tol</td>
<td>23</td>
</tr>
<tr>
<td>subset_ref</td>
<td>110</td>
</tr>
<tr>
<td>subset_ref (subset_sampled)</td>
<td>122</td>
</tr>
<tr>
<td>subset_sampled</td>
<td>110, 122</td>
</tr>
<tr>
<td>summarize</td>
<td>41</td>
</tr>
<tr>
<td>tibble</td>
<td>105, 106</td>
</tr>
<tr>
<td>weights_redist_plans</td>
<td>22</td>
</tr>
<tr>
<td>(get_plans_weights)</td>
<td>22</td>
</tr>
</tbody>
</table>