Package 'proporz'

July 23, 2025

Type Package

Title Proportional Apportionment

Version 1.5.1

Description Calculate seat apportionment for legislative bodies with various methods. The algorithms include divisor or highest averages methods (e.g. Jefferson, Webster or Adams), largest remainder methods and biproportional apportionment.
Gaffke, N. & Pukelsheim, F. (2008) <doi:10.1016/j.mathsocsci.2008.01.004>
Oelbermann, K. F. (2016) <doi:10.1016/j.mathsocsci.2016.02.003>.

License GPL (>= 3)

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

Depends R (>= 3.6.0)

Suggests shiny, shinyMatrix, testthat, knitr, rmarkdown

URL https://polettif.github.io/proporz/,
https://github.com/polettif/proporz

BugReports https://github.com/polettif/proporz/issues

VignetteBuilder knitr

NeedsCompilation no

Author Flavio Poletti [aut, cre, cph]

Maintainer Flavio Poletti <flavio.poletti@hotmail.ch>

Repository CRAN

Date/Publication 2025-03-10 12:10:02 UTC

Contents

`apply_quorum` *Apply quorum to votes vector or matrix*

Description

This quorum calculation implementation is called within `proporz()`, `biproporz()` and related functions. Generally, there's no need to call `apply_quorum` directly.

Usage

```
apply_quorum(votes, quorum)
```

Arguments

`votes`	votes vector or votes matrix
`quorum`	Depending on `votes`: • For a vector: Vote threshold a party must reach. Used as fraction of total votes if less than 1 otherwise as number of votes. • For a matrix: List of quorum functions (created with `quorum_functions`) or a logical vector with the same length as the number of `votes` rows.

Value

Vector or matrix with same dimension as `votes`. Parties that failed to reach the specified quorum have their votes set to zero.

See Also

quorum_functions for more matrix examples.

Examples

```
# vector
(votes = c(81, 9, 10))

apply_quorum(votes, 10)

apply_quorum(votes, .11)

# matrix
(votes_matrix = matrix(c(91, 9, 199, 1), nrow = 2))

apply_quorum(votes_matrix, quorum_all(total = 0.1))

apply_quorum(votes_matrix, c(FALSE, TRUE))
```

---

biproporz *Biproportional apportionment*

Description

Method to proportionally allocate seats among parties (or lists) and districts (or entities, regions), thus bi-proportional.

Usage

```
biproporz(
  votes_matrix,
  district_seats,
  quorum,
  use_list_votes = TRUE,
  method = "round"
)
```

Arguments

votes_matrix	Vote count matrix with votes by party in rows and votes by district in columns.
district_seats	Vector defining the number of seats per district. Must be the same length as ncol(votes_matrix). Values are name-matched to votes_matrix columns if both are named. If the number of seats per district should be calculated according to the number of votes (not the general use case), a single number for the total number of seats can be used.

`quorum` Optional list of functions which take the votes_matrix and return a logical vector that denotes for each party/row whether they reached the quorum (i.e. are eligible for seats). The easiest way to do this is via `quorum_any()` or `quorum_all()`, see examples. Alternatively you can pass a precalculated logical vector. No quorum is applied if parameter is missing or `NULL`.

`use_list_votes` By default (`TRUE`) it's assumed that each voter in a district has as many votes as there are seats in a district. Thus, votes are weighted according to the number of available district seats with `weight_list_votes()`. Set to `FALSE` if `votes_matrix` shows the number of voters (i.e. they can only cast one vote for one party).

`method` Defines which method is used to assign seats. The following methods are recommended:

- `round`: Uses the Sainte-Laguë/Webster method (rounding half up) for the upper and lower apportionment which is the standard for biproportional apportionment and the only method guaranteed to terminate.
- `wto`: "winner take one" works like "round" with a condition that the party that got the most votes in a district must get *at least* one seat ('Majorzbedingung') in said district. This only applies if they got enough seats in the upper apportionment (which uses the Sainte-Laguë/Webster method). See `lower_apportionment()` for more details.

It is also possible to use any divisor method name listed in `proporz()`. If you want to use a different method for the upper and lower apportionment, provide a list with two entries.

Details

Each party nominates a candidate list for every district. The voters vote for the parties of their district. The seat allocation is calculated in two steps:

1. In the so called `upper apportionment` the number of seats for each party (over all districts) is determined. Normally, the number of seats for each region are defined before the election and are independent of the vote counts.
2. In the so called `lower apportionment` the seats are distributed to the regional party list respecting the results from the upper apportionment.

Parties failing to reach quorums cannot get seats. This function does not handle seat assignment to candidates.

Value

Matrix with the same dimension as `votes_matrix` containing the number of seats with the row and column divisors stored in attributes (hidden from print, see `get_divisors()`).

Note

The iterative process in the lower apportionment is only guaranteed to terminate with the default Sainte-Laguë/Webster method.

References

Gaffke, Norbert; Pukelsheim, Friedrich (2008): Divisor methods for proportional representation systems: An optimization approach to vector and matrix apportionment problems. Mathematical Social Sciences, 56 (2), 166-184.

See Also

`pukelsheim()` for biproportional apportionment with `data.frames` as inputs.

Examples

```
votes_matrix = uri2020$votes_matrix
district_seats = uri2020$seats_vector

biproporz(votes_matrix, district_seats)

# apply quorum (high values for illustrative purposes)
biproporz(votes_matrix, district_seats,
          quorum_all(any_district = 0.1, total = 0.25))
```

---

ceil_at *Rounding with predefined thresholds*

Description

Round `x` up to `ceiling(x)` if `x-floor(x) >= threshold`, otherwise round down to `floor(x)`.

Usage

```
ceil_at(x, threshold)
```

Arguments

`x`	numeric vector or matrix >= 0 (`NaN` is not supported)
`threshold`	threshold in [0,1] or "harmonic"/"geometric" to use harmonic or geometric mean thresholds

Value

the rounded vector or matrix

Examples

```
ceil_at(c(0.5, 1.5, 2.49, 2.5, 2.51), 0.5)
# compare to
round(c(0.5, 1.5, 2.49, 2.5, 2.51))

ceil_at(c(1.45, 2.45, 3.45), 0) # like floor()
ceil_at(c(1.45, 2.45, 3.45, 0.2), "geometric")
```

district_winner_matrix *Find which party has the most votes in a district*

Description

Create a logical matrix that shows whether a party got the most votes in a district or not.

Usage

```
district_winner_matrix(votes_matrix, district_seats = 1L)
```

Arguments

votes_matrix	Vote count matrix with votes by party in rows and votes by district in columns.
district_seats	Vector defining the number of seats per district. Must be the same length as ncol(votes_matrix). Values are name-matched to votes_matrix columns if both are named. If a single value is supplied (like 1 as default), it is used as the number of seats for every district.

Details

If two or more parties are tied and there are not enough seats for each tied party, the matrix value is NA.

Value

logical matrix with the same dimensions and names as votes_matrix

Examples

```
(vm = matrix(c(60,30,0,20,10,30), nrow = 3, dimnames = list(1:3, c("A", "B"))))

district_winner_matrix(vm)

# NA values if parties are tied (here in district B)
vm[1,2] <- 30
district_winner_matrix(vm)

# No NA values for tied parties if enough seats are available
district_winner_matrix(vm, c(1, 2))
```

divisor_methods *Divisor methods*

Description

Functions to directly apply divisor apportionment methods instead of calling `proporz()` with a method parameter. All divisor functions call `highest_averages_method()` with a different sequence of divisors.

Usage

```
divisor_round(votes, n_seats, quorum = 0)

divisor_floor(votes, n_seats, quorum = 0)

divisor_harmonic(votes, n_seats, quorum = 0)

divisor_geometric(votes, n_seats, quorum = 0)

divisor_ceiling(votes, n_seats, quorum = 0)
```

Arguments

`votes`	numeric vector with number of votes for each party
`n_seats`	total number of seats
`quorum`	Vote threshold a party must reach. Used as fraction of total votes within if less than 1 otherwise as number of votes.

Details

Divisor methods are known under different names:

- d'hondt, jefferson, hagenbach-bischoff: `divisor_floor()`
- sainte-lague, webster: `divisor_round()`
- adams: `divisor_ceiling()`
- dean: `divisor_harmonic()`
- huntington-hill, hill-huntington: `divisor_geometric()`

Value

The number of seats per party as a vector

See Also

`proporz()`, `highest_averages_method()`

Examples

```
votes = c("Party A" = 690, "Party B" = 400,
          "Party C" = 250, "Party D" = 120)

divisor_round(votes, 10)

divisor_floor(votes, 10)

divisor_ceiling(votes, 10)

divisor_ceiling(votes, 5)

divisor_geometric(votes, 10, quorum = 0.05)

divisor_harmonic(votes, 10)
```

---

finland2019 *Finnish Parliamentary Elections Data (2019)*

Description

Example data from the 2019 Finnish parliamentary elections. The data has been cleaned up and only contains information relevant for this package.

Usage

```
finland2019
```

Format

List containing two data.frames:

- `votes_df` containing the number of votes for each party and district. 229 rows, 3 columns (`party_name`, `district_name`, `votes`)
- `district_seats_df` with the number of seats per district. 12 rows, 2 columns (`district_name`, `seats`)

Source

https://tulospalvelu.vaalit.fi/EKV-2019/en/ladattavat_tiedostot.html

Examples

```
finland2019$district_seats_df

head(finland2019$votes_df)
```

get_divisors *Get district and party divisors from biproporz result*

Description

Show the district and party divisors used to assign seats. This method provides easier access to divisors stored in `attributes(...)$divisors`.

Usage

```
get_divisors(biproporz_result)
```

Arguments

`biproporz_result`	a matrix created by `biproporz()` or a data.frame created by `pukelsheim()`

Value

The district and party divisors (named "districts" and "parties") in a list, each as a vector

Examples

```
seats_matrix = biproporz(uri2020$votes_matrix, uri2020$seats_vector)
get_divisors(seats_matrix)

seats_df = pukelsheim(pivot_to_df(uri2020$votes_matrix),
                      data.frame(names(uri2020$seats_vector), uri2020$seats_vector))
get_divisors(seats_df)

# summary() also prints the divisors for a biproporz matrix
summary(seats_matrix)
```

highest_averages_method *Highest averages method*

Description

Allocate seats proportionally for `divisor methods`.

Usage

```
highest_averages_method(votes, n_seats, divisors)
```

Arguments

votes	numeric vector with number of votes for each party
n_seats	total number of seats
divisors	sequence of divisors (length equal to the number of seats). If it is a single number (e.g. 0.5), a sequence is generated starting with it, increasing by 1.

Details

The highest averages method requires the number of votes for each party to be divided successively by a series of divisors. This produces a table of quotients, or averages, with a row for each divisor and a column for each party. The nth seat is allocated to the party whose column contains the nth largest entry in this table, up to the total number of seats available. (Wikipedia)

Value

The number of seats per party as a vector

Examples

```
highest_averages_method(c(5200, 1700, 3100), 15, 0.5)

highest_averages_method(votes = c(50, 0, 30), n_seats = 3,
                        divisors = c(0, 1.3333, 2.4))
```

---

largest_remainder_method

Largest remainder method

Description

Allocate seats based on the largest fractional remainder. The largest remainder method is also known as: Hamilton, Hare-Niemeyer or Vinton method.

Usage

```
largest_remainder_method(votes, n_seats, quorum = 0)
```

Arguments

votes	numeric vector with number of votes for each party
n_seats	total number of seats
quorum	Vote threshold a party must reach. Used as fraction of total votes within if less than 1 otherwise as number of votes.

Details

The numbers of votes for each party is divided by a quota representing the number of votes required for a seat. Then, each party receives the rounded down quota value as seats. The remaining seats are given to the party with the largest remainder until all seats have been distributed.

Value

The number of seats per party as a vector

Note

Only the quota `total votes / total seats` (which is used by the aforementioned methods) is implemented.

See Also

`proporz()`

Examples

```
votes = c(47000, 16000, 15800, 12000, 6100, 3100)
largest_remainder_method(votes, 10)
```

---

lower_apportionment *Lower apportionment*

Description

In the second biproportional apportionment step, party and district divisors are calculated such that the row and column sums of the resulting seats matrix satisfy the constraints given by the upper apportionment.

Usage

```
lower_apportionment(votes_matrix, seats_cols, seats_rows, method = "round")
```

Arguments

`votes_matrix`	matrix with votes by party in rows and votes by district in columns.
`seats_cols`	number of seats per column (districts/regions), predetermined or calculated with `upper_apportionment()`.
`seats_rows`	number of seats per row (parties/lists), calculated with `upper_apportionment()`.
`method`	Apportion method that defines how seats are assigned. The following methods are supported: • `round`: The default Sainte-Laguë/Webster method is the standard for biproportional apportionment and the only method guaranteed to terminate.

- `wto`: "winner take one" works like `round` with a condition that the party that got the most votes in a district must get *at least* one seat ('Majorzbedingung', also called 'strongest party constrained' rule (SPC)). `votes_matrix` must have row and column names to use this method. A district winner can only get a seat if they are entitled to one from the upper apportionment (`seats_rows`). The condition does not apply in a district if two or more parties have the same number of votes and there are not enough seats for these parties. A warning is issued in this case. Modify the votes matrix to explicitly break ties.
- You can provide a custom function that rounds a matrix (i.e. the the `votes_matrix` divided by party and district divisors) without further parameters.
- It is possible to use any divisor method name listed in `proporz()`.

Details

The result is obtained by an iterative process ('Alternate Scaling Algorithm', see Reference). Initially, for each district a divisor is chosen using the highest averages method for the votes allocated to each regional party list in this region. For each party a party divisor is initialized with 1.

Effectively, the objective of the iterative process is to modify the regional divisors and party divisors so that the number of seats in each regional party list equals the number of their votes divided by both the regional and the party divisors.

The following two correction steps are executed until this objective is satisfied:

- modify the party divisors such that the apportionment within each party is correct with the chosen rounding method,
- modify the regional divisors such that the apportionment within the region is correct with the chosen rounding method.

Value

A seat matrix with district (columns) and party (rows) divisors stored in attributes.

Note

If the maximum number of optimization iterations is reached, an error is thrown since no solution can be found. You can overwrite the default (1000) with `options(proporz_max_iterations = ...)` but it is very likely that the result is undefined given the structure of the input parameters.

References

Oelbermann, K. F. (2016): Alternate scaling algorithm for biproportional divisor methods. Mathematical Social Sciences, 80, 25-32.

See Also

`biproporz()`, `upper_apportionment()`, `district_winner_matrix()`

Examples

```
votes_matrix = matrix(c(123,912,312,45,714,255,815,414,215), nrow = 3)
district_seats = c(7,5,8)
party_seats = c(5,11,4)

lower_apportionment(votes_matrix, district_seats, party_seats)


# using "winner take one"
vm = matrix(c(200,100,10,11), 2,
            dimnames = list(c("Party A", "Party B"), c("I", "II")))
district_seats = setNames(c(2,1), colnames(vm))
ua = upper_apportionment(vm, district_seats)

lower_apportionment(vm, ua$district, ua$party, method = "wto")

# compare to standard method
lower_apportionment(vm, ua$district, ua$party, method = "round")
```

`pivot_to_matrix` *Pivot long data.frame to wide matrix and vice versa*

Description

Create a matrix in 'wide' format from a data.frame with 3 columns with `pivot_to_matrix` or create a data.frame in long format from a matrix with `pivot_to_df`.

Usage

```
pivot_to_matrix(df_long)

pivot_to_df(matrix_wide, value_colname = "values")
```

Arguments

`df_long`	data.frame in long format with exactly 3 columns
`matrix_wide`	matrix in wide format
`value_colname`	name for the new value column in the resulting data.frame

Details

These pivot functions are used to prepare data for `biproporz()` in `pukelsheim()`. They are not supposed to cover general use cases or provide customization. They mainly exist because reshape is hard to handle and the package should have no dependencies.

Value

A data.frame with 3 columns or a matrix. Note that the results are sorted by the first and second column (data.frame) or row/column names (matrix).

Examples

```
# From data.frame to matrix
df = data.frame(party = c("A", "A", "A", "B", "B", "B"),
                region = c("III", "II", "I", "I", "II", "III"),
                seats = c(5L, 3L, 1L, 2L, 4L, 6L))
pivot_to_matrix(df)

# from matrix to data.frame
mtrx = matrix(1:6, nrow = 2)
pivot_to_df(mtrx)

# from matrix to data.frame using dimnames
dimnames(mtrx) <- list(party = c("A", "B"), region = c("I", "II", "III"))
pivot_to_df(mtrx, "seats")

# Note that pivot results are sorted
pivot_to_df(pivot_to_matrix(df)) == df[order(df[[1]], df[[2]]),]
```

`proporz` *Proportional apportionment*

Description

Calculate seat apportionment for legislative bodies.

Usage

```
proporz(votes, n_seats, method, quorum = 0)
```

Arguments

`votes`	numeric vector with number of votes for each party
`n_seats`	total number of seats
`method`	Apportionment method to use, as character. Not case sensitive. See details.
`quorum`	Vote threshold a party must reach. Used as fraction of total votes within if less than 1 otherwise as number of votes.

Details

The following methods are available:

- d'hondt, jefferson, hagenbach-bischoff, floor: `divisor_floor()`
- sainte-lague, webster, round: `divisor_round()`
- adams, ceiling: `divisor_ceiling()`
- dean, harmonic: `divisor_harmonic()`
- huntington-hill, hill-huntington, geometric: `divisor_geometric()`
- hare-niemeyer, hamilton, vinton, largest_remainder_method: `largest_remainder_method()`

Value

The number of seats per party as a vector

Note

Seats can also be apportioned among regions instead of parties. The parameter `votes` is then normally used with census data (e.g. population counts).

Examples

```
votes = c("Party A" = 651, "Party B" = 349, "Party C" = 50)

proporz(votes, 10, "sainte-lague")

proporz(votes, 10, "hill-huntington")

proporz(votes, 10, "hill-huntington", quorum = 0.05)

proporz(votes, 10, "jefferson", quorum = 70)
```

---

pukelsheim *Biproportional apportionment with data frames*

Description

Method to proportionally allocate seats among parties/lists and districts/regions/entities ('Doppelter Pukelsheim').

Usage

```
pukelsheim(
  votes_df,
  district_seats_df,
  quorum,
  new_seats_col = "seats",
  use_list_votes = TRUE,
  winner_take_one = FALSE
)
```

Arguments

votes_df	data.frame (long format) with 3 columns (actual colnames can differ): • party id/name • district id/name • votes
district_seats_df	data.frame with 2 columns (actual colnames can differ): • district id/name • number of seats for a district
quorum	Optional list of functions which take the votes_matrix and return a logical vector that denotes for each party/row whether they reached the quorum (i.e. are eligible for seats). The easiest way to do this is via quorum_any() or quorum_all(), see examples. Alternatively you can pass a precalculated logical vector. No quorum is applied if parameter is missing or NULL.
new_seats_col	name of the new column
use_list_votes	By default (TRUE) it's assumed that each voter in a district has as many votes as there are seats in a district. Set to FALSE if votes_df shows the number of voters (e.g. they can only vote for one party).
winner_take_one	Set to TRUE if the party that got the most votes in a district must get *at least* one seat ('Majorzbedingung') in this district. This only applies if they are entitled to a seat in the upper apportionment. Default is FALSE.

Details

Each party nominates a candidate list for every district. The voters vote for the parties of their district. The seat allocation is calculated in two steps:

1. In the so called upper apportionment the number of seats for each party (over all districts) is determined.
2. In the so called lower apportionment the seats are distributed to the regional party list respecting the results from the upper apportionment.

Parties failing to reach quorums cannot get seats. This function does not handle seat assignment to candidates.

If you want to use other apportion methods than Sainte-Laguë use biproporz().

Value

A data.frame like `votes_df` with a new column denoting the number seats per party and district. Party and district divisors stored in attributes in attributes (hidden from print, see `get_divisors()`).

See Also

This function calls `biproporz()` after preparing the input data.

Examples

```
# Zug 2018
votes_df = unique(zug2018[c("list_id", "entity_id", "list_votes")])
district_seats_df = unique(zug2018[c("entity_id", "election_mandates")])

seats_df = pukelsheim(votes_df,
                      district_seats_df,
                      quorum_any(any_district = 0.05, total = 0.03),
                      winner_take_one = TRUE)

head(seats_df)

# Finland 2019
finland19_result = pukelsheim(finland2019$votes_df,
                              finland2019$district_seats_df,
                              new_seats_col = "mandates",
                              use_list_votes = FALSE)
tail(finland19_result[order(finland19_result$mandates),])
```

`quorum_functions` *Create quorum functions for biproportional apportionment*

Description

`quorum_any()` and `quorum_all()` are used for the `quorum` parameter in `biproporz()`/`pukelsheim()` and help describe how quorums should be applied prior to seat distributions.

Usage

```
quorum_all(any_district, total)

quorum_any(any_district, total)
```

Arguments

`any_district`	Vote threshold a party must reach in *at least* one district. Used as share of total votes within a district if less than 1 otherwise as number of votes. Must be greater than 0. Uses `reached_quorum_any_district()`.

`total` Vote threshold a party must reach for all votes cast. Used as share of total votes if less than 1. Otherwise as number of votes. Note that votes are not weighted with `weight_list_votes()` across districts. Must be greater than 0. Uses `reached_quorum_total()`.

Details

There's a difference in how the functions work. With `quorum_any`, *at least one* quorum must be reached. With `quorum_all` *all* (i.e. both) quorums must be reached. If you only use one parameter, `quorum_any()` and `quorum_all()` are identical.

Value

a function which, when called with `function(votes_matrix)`, returns a boolean vector with length equal to the number of lists/parties (`votes_matrix` rows). The vector shows whether a party has reached any/all quorums.

See Also

`apply_quorum()` for standalone quorum calculations

Examples

```
votes_matrix = matrix(c(502, 55, 80, 10, 104, 55, 0, 1), ncol = 2)
dimnames(votes_matrix) <- list(c("A", "B", "C", "D"), c("Z1", "Z2"))
seats = c(Z1 = 50, Z2 = 20)

# use as parameter in biproporz or pukelsheim (general use case)
biproporz(votes_matrix, seats,
          quorum = quorum_any(any_district = 0.1, total = 100))

biproporz(votes_matrix, seats,
          quorum = quorum_all(any_district = 0.1, total = 100))

biproporz(votes_matrix, seats, quorum = quorum_any(any_district = 0.1))

biproporz(votes_matrix, seats, quorum = quorum_any(total = 100))

biproporz(votes_matrix, seats, quorum = quorum_any(total = 0.5))

# the quorum parameter also accepts vectors (e.g. calculated elsewhere)
biproporz(votes_matrix, seats, quorum = c(FALSE, TRUE, TRUE, TRUE))
```

`reached_quorum_any_district` *Check if parties reached a quorum in at least one district*

Description

Base implementation, used by quorum_functions.

Usage

```
reached_quorum_any_district(votes_matrix, quorum_districts)
```

Arguments

votes_matrix	votes matrix
quorum_districts	Vote threshold a party must reach in *at least* one district. Used as fraction of total votes within a district if less than 1, otherwise as number of votes. Must be greater than 0.

Value

Logical vector with length equal to the number of lists/parties (votes_matrix rows) showing whether they reached the quorum or not.

See Also

reached_quorum_total()

Examples

```
(vm = matrix(c(239, 10, 308, 398, 20, 925), nrow = 3))
reached_quorum_any_district(vm, 25)
```

---

reached_quorum_total *Check if parties reached the quorum for all votes*

Description

Base implementation, used by quorum_functions.

Usage

```
reached_quorum_total(votes_matrix, quorum_total)
```

Arguments

votes_matrix	votes matrix
quorum_total	Vote threshold a party must reach for all votes cast. Used as fraction of total votes if less than 1, otherwise as number of votes. Must be greater than 0.

Value

Logical vector with length equal to the number of lists/parties (`votes_matrix` rows) showing whether they reached the quorum or not.

Note

Votes are not weighted across districts. This is relevant if the quorum threshold is the minimal number of *voters* (either as percentage or absolute value). In this case, use `weight_list_votes()` before calculating the quorum.

See Also

`reached_quorum_any_district()`

Examples

```
(vm = matrix(c(239, 10, 308, 398, 20, 925), nrow = 3))
reached_quorum_total(vm, 35)
```

---

run_app *Use biproportional apportionment interactively in a shiny app*

Description

Use biproportional apportionment interactively in a shiny app

Usage

```
run_app(votes_matrix = NULL, district_seats = NULL)
```

Arguments

`votes_matrix`	optional votes_matrix to load upon start
`district_seats`	optional district_seats to load upon start

Value

Calling the function starts the shiny app

Examples

```
if(interactive()){
    # You need to have the packages 'shiny' and 'shinyMatrix' installed to run the app
    run_app()

    # It's possible to load a matrix with the app
    run_app(uri2020$votes_matrix, uri2020$seats_vector)
}
```

upper_apportionment *Upper apportionment*

Description

In the first step of biproportional apportionment parties are given seats according to the sum of their votes across all districts.

Usage

```
upper_apportionment(
  votes_matrix,
  district_seats,
  use_list_votes = TRUE,
  method = "round"
)
```

Arguments

votes_matrix	Vote count matrix with votes by party in rows and votes by district in columns.
district_seats	Vector defining the number of seats per district. Must be the same length as ncol(votes_matrix). Values are name-matched to votes_matrix columns if both are named. If the number of seats per district should be calculated according to the number of votes (not the general use case), a single number for the total number of seats can be used.
use_list_votes	By default (TRUE) it's assumed that each voter in a district has as many votes as there are seats in a district. Thus, votes are weighted according to the number of available district seats with weight_list_votes(). Set to FALSE if votes_matrix shows the number of voters (i.e. they can only cast one vote for one party).
method	Apportion method that defines how seats are assigned, see proporz(). Default is the Saintë-Lague/Webster method.

Value

A named list with district seats (for votes_matrix columns) and party seats (for rows).

Note

The results from the upper apportionment define the number of seats for each party and the number of seats for each district for the whole voting area. The lower apportionment will only determine where (i.e. which district) the party seats are allocated. Thus, after the upper apportionment is done, the final strength of a party/district within the parliament is definite.

See Also

biproporz(), lower_apportionment()

Examples

```
votes_matrix = matrix(c(123,912,312,45,714,255,815,414,215), nrow = 3)
district_seats = c(7,5,8)

upper_apportionment(votes_matrix, district_seats)
```

---

uri2020 *Election Data for the Cantonal Council of Uri (2020)*

Description

Example election data from the 2020 election for the cantonal council of Uri (Landrat) in Switzerland. The data has been extracted from the report "Landratswahlen 2020: Statistische Auswertung".

Usage

```
uri2020
```

Format

List containing:

- `votes_matrix` the number of votes for each party and district (4 rows, 4 columns)
- `seats_vector` with the number of seats per district (length 4)

Source

https://www.ur.ch/abstimmungen/termine/9322

---

weight_list_votes *Create weighted votes matrix*

Description

Weight list votes by dividing the votes matrix entries by the number of seats per district. This method is used in `upper_apportionment()` if `use_list_votes` is `TRUE` (default).

Usage

```
weight_list_votes(votes_matrix, district_seats)
```

Arguments

`votes_matrix`	votes matrix
`district_seats`	seats per district, vector with same length as `ncol(votes_matrix)`

Value

the weighted `votes_matrix`

Note

The weighted votes are not rounded. Matrix and vector names are ignored.

Examples

```
weight_list_votes(uri2020$votes_matrix, uri2020$seats_vector)
```

`zug2018` *Election Data for the Cantonal Council of Zug (2018)*

Description

Example election data from the 2018 election for the cantonal council of Zug (Kantonsrat) in Switzerland.

Usage

```
zug2018
```

Format

An object of class `data.frame` with 267 rows and 49 columns.

Source

Kanton Zug (01.07.2022, 10:27:58). Kantonsratswahl 2018 (CSV). https://wab.zug.ch/elections/kantonsratswahl-2018/data-csv

Index