# Package 'i3pack'

July 22, 2025

**Title** Incentives for Inter- And Intra-Party Electoral Competition

**Version** 0.1.0

**Description** Suite of functions that help simulate elections under different electoral systems, which are then used to compute incentives generated by these systems in terms of the inter- and intra-party dimensions of electoral competition.

**License** GPL (>= 2)

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.3.2

**Imports** dplyr (>= 1.1.4), gbm (>= 2.2.2), magrittr (>= 2.0.3), readr (>= 2.1.5), rlang (>= 1.1.6), STV (>= 1.0.2), tidyr (>= 1.3.1)

**Depends** R (>= 3.5.0)

**NeedsCompilation** no

**Author** Santiago Olivella [aut, cre],
Patrick Cunha [aut],
Ayelen Vanegas [aut],
Matias Tarillo [aut],
Guillermo Rosas [ctb],
Brian Crisp [ctb]

**Maintainer** Santiago Olivella <olivella@unc.edu>

**Repository** CRAN

**Date/Publication** 2025-06-04 15:30:05 UTC

## Contents

---

ElecFuns *Electoral Functions*

---

## Description

This file contains details and examples of the electoral functions (electoral formulas) implemented in the i3pack package.

## Usage

```
a_v(v, ...)

bc(v, m, mod = TRUE, n_cand_bc, ...)

dhondt(v, m, threshold = 0, ...)

droop(v, m, threshold = 0, ...)

fortified_pr(
  v,
  m,
  threshold = 0,
  fpr_cutoff,
  pr_formula,
  include_first_party,
  ...
)

hagenbachbischoff(v, m, threshold = 0, ...)

hare(v, m, threshold = 0, ...)

imperiali(v, m, threshold = 0, ...)

lim_nom(v, m, ...)

modsaintelague(v, m, threshold = 0, ...)

plurality(v, m, ...)

saintelague(v, m, threshold = 0, ...)
```

```
stv(v, m, ...)
```

## Arguments

| | |
|---|---|
| v | Matrix with candidates/parties (the interpretation depends on the function) in the columns. Each cell has the number of votes cast for each candidate/party. For AV and STV, the matrix should have ranked votes, with each rank in a separate row. |
| ... | Additional optional arguments (currently ignored). |
| m | Number of seats to be awarded. |
| mod | Included in the BC function. If TRUE, makes the sequence run from 1 to the number of votes and inverses it; if FALSE, the sequence run from the number of votes to 1). Should be TRUE if the intention is to use the BC system described in the details. |
| n_cand_bc | Number of candidates in the Borda Count system. This is only used for the Borda Count system. |
| threshold | Proportion of votes that a party (or candidate) needs to surpass in order to be eligible to receive seats. |
| fpr_cutoff | Included in the fortified_pr function. It is a percentage of votes that a party needs to surpass in order to be eligible to receive the "bonus" seats assigned to the winner of the election. |
| pr_formula | A character vector that specifies the quota implemented. In general, is equal to "hare". The Hare quota is the number of votes cast in a district divided by M. |
| include_first_party | A logical value that indicates whether the top-voted list party participate in the distribution of the remaining seats or not. If TRUE, it does. |

## Value

For Alternative Vote, the name of the candidate that obtains majority support.

For Borda count and all PR formulas, a vector of seats awarded to each candidate.

For plurality, a matrix with all candidates participating (1 if a seat was awarded, 0 if not). If m = 1, candidates can be interpreted as parties.

## Details

The a_v function is used in single member districts and lets voters rank the candidates competing in the district from most to least preferred. If a candidate is ranked first by a majority of voters, he or she wins the single seat available. If no candidate obtains a majority of votes, the candidate with the fewest first-place votes is eliminated and her votes are distributed to the candidates that her supporters ranked second. If this redistribution gives another candidate a majority of the votes, that candidate is elected; if not, the second weakest candidate is eliminated and his votes are redistributed among the surviving candidates. This process of redistribution and recounting continues until a candidate obtains majority support.

The bc function is used in single-member or multi-member districts, though it is typically discussed in settings that return a single choice. Voters assign candidates a rank, but seats are then awarded by plurality rather than majority. Each rank is assigned a weight, and a candidate's vote total is the sum of the full and fractional votes he or she receives. For example, a first place rank might be weighted by one, a second place rank by 1/2, a third place rank by 1/3. A candidate ranked first by one hundred voters, second by twenty voters, and third by ten voters would be awarded a total of 100/1 + 20/2 + 12/3 =114 votes. This, in fact, is the modified bc system used to elect the Parliament in the country of Nauru (not included in our dataset), an island in Micronesia. To our knowledge, the bc is not used in national elections elsewhere.

The dhondt function divides parties' vote totals successively by 1, 2, 3, 4, 5, and so on (until m). Seats are then awarded sequentially starting with the party that enjoys the largest quotient until no more seats are available.

The droop function assigns seats by calculating the Droop quota, which is Q = V/(M+1) + 1 (rounded to the nearest integer). The seats are assigned in two stages. First, the number of votes obtained by each party is divided by Q and rounded down. That integer is the number of seats that the party will obtain in the first stage. Second, Q is multiplied by the number of seats obtained by each party (Si) and that number if substracted from the total number of votes obtained by that party. That would be a residual: Ri = Vi - Q*Si. The remaining seats are assigned to the parties with the largest residuals.

The fortified_pr function is used for proportional representation with a majority bonus. The seat allocation formula is different from other list PR systems. Under this set of rules, the list which receives the largest vote share receives a bonus in seats. Sometimes, that list needs to surpass a certain percentage of votes (the cutoff) in order to be eligible for that. In this case, the function assigns half the seats to the party with most votes and assigns the other half of the seats proportionally.

The hagenbachbischoff function works with the same procedure as the droop function, but in this case Q = V/(M+1).

The hare function works with the same procedure as the droop function, but in this case Q = V/M.

The imperiali function works with the same procedure as the droop function, but in this case Q = V/(M+2).

The lim_nom function is used to calculate the seats obtained with closed-list plurality with limited nomination system. Voters only get one vote, which is cast for a closed party list. District magnitude needs to be 3 (i.e., M=3) and the top vote-getting party is awarded two seats while the third seat goes to the second-place finisher — even if its level of support is abysmally low.

The modsaintelague function works with the same procedure as the dhondt function, but in this case the sequence of numbers used for the division is only comprised by odd numbers except for the first one, which is 1.4 instead of 1. It ends up being: 1.4, 3, 5, 7 and so on. It uses an amount of numbers equal to m.

The plurality function returns the number of seats according to the seat allocation formula—plurality. In a single-member district decided by plurality system, voters get a single vote, cast at the party level, to fill the only contested seat, and that seat goes to the top vote-earner regardless of level of support. In a multiple non-transferable vote system, the votes are cast at the candidate level and m is greater than 1. The number of candidates should always be greater or equal to m.

The saintelague function works with the same procedure as the dhondt function, but in this case the sequence of numbers used for the division is only comprised by odd numbers (1, 3, 5, 7 and so on). It uses an amount of odd numbers equal to m.

## Examples

```
a_v(v=ranked)


bc(v=ranked, 2, mod=TRUE, n_cand_bc=3)

## D'hondt without threshold:

dhondt(v=example, m=3)

## D'hondt with 30% threshold:

dhondt(v=example, m=3, threshold=0.3)

## Droop without threshold:

droop(v=example, m=3)

## Droop with 20% threshold:

droop(v=example, m=3, threshold=0.2)

## Fortified PR without cutoff:

fortified_pr(v=example, m=4, fpr_cutoff=0, include_first_party=TRUE, pr_formula="hare")

## Fortified PR with a 50% cutoff (including first party):

fortified_pr(v=example, m=4, fpr_cutoff=0.5, include_first_party=TRUE, pr_formula="hare")

## Fortified PR with a 50% cutoff (without including first party):

fortified_pr(v=example, m=4, fpr_cutoff=0.5, include_first_party=FALSE, pr_formula="hare")

## Hagenbach-Bischoff without threshold:

hagenbachbischoff(v=example, m=3)

## Hagenbach-Bischoff with 20% threshold:

hagenbachbischoff(v=example, m=3, threshold=0.2)

## Hare without threshold

hare (v=example, m=3)

## Hare with 20% threshold

hare (v=example, m=3, threshold=0.2)

## Imperiali without threshold:
```

```
imperiali(v=example, m=3)

## Imperiali with 20% threshold:

imperiali(v=example, m=3, threshold=0.2)


## Lim_nom (only works with m=3)

lim_nom(v=example, m=3)

## Modified Sainte-Lague without threshold:

modsaintelague(v=example, m=3)

## Modified Sainte-Lague with 20% threshold:

modsaintelague(v=example, m=3, threshold=0.2)

plurality (v=example, m=3)
## Sainte-Lague without threshold:

saintelague(v=example, m=3)

## Sainte-Lague with 20% threshold:

saintelague(v=example, m=3, threshold=0.2)


stv (v=ranked, m=2)
```

---

`example` *Example of total vote count vector*

---

## Description

This is an example of a total vote count vector for 5 parties/candidates that can be used to illustrate different seat allocation formulas.

## Usage

```
example
```

## Format

A vector of integers representing the total votes received by each party/candidate.

**A** Votes received by party A

**B** Votes received by party B ...

## Source

This is a synthetic example created for demonstration purposes.

---

`find_best_candidates` *Auxiliary Internal Functions*

---

## Description

Auxiliary Internal Functions

## Usage

```
find_best_candidates(x, n, rank = TRUE)
```

## Arguments

| | |
|---|---|
| `x` | Matrix with two columns: candidate ID (`candidate`) and utility (`dist`) |
| `n` | Number of candidates to be nominated |
| `rank` | Boolean: should nominated candidates be ranked? Defaults to `TRUE`. |

---

`lsq` *Auxiliary Internal Functions*

---

## Description

Auxiliary Internal Functions

Auxiliary Internal Functions

Auxiliary Internal Functions

## Usage

```
lsq(v, s)

max_ninf(x)

stat_mode(x)
```

## Arguments

| | |
|---|---|
| `v` | vector of vote totals by party |
| `s` | vector with the seats received by each party. |
| `x` | Numeric vector |

## Value

The "least squares" index of disproportionality

The maximum of a vector, after removing non-finite elements

Statistical mode

---

`max_n` *Auxiliary Internal Functions*

---

## Description

Auxiliary Internal Functions

## Usage

```
max_n(x, n)
```

## Arguments

| | |
|---|---|
| `x` | Numeric vector |
| `n` | Number of elements to return |

## Value

The largest `n` elements of a vector

---

`nominating` *Auxiliary Internal Functions*

---

## Description

Auxiliary Internal Functions

## Usage

```
nominating(
  parties,
  lists_per_party,
  rank_cand,
  n_cand,
  party_1 = TRUE,
  party = NULL
)
```

### Arguments

| | |
|---|---|
| parties | See simulate_election(). |
| lists_per_party | See simulate_election(). |
| rank_cand | See simulate_election(). |
| n_cand | Numeric maximum number of candidates running in a party list; defaults to 0, which is internally interpreted as the district magnitude. |
| party_1 | Boolean: Are we generating the last (only) party list? Defaults to TRUE. |
| party | Optional numeric party ID. |

### Value

data.frame with the following variable

**rank** List rank/position

**candidate** Candidate ID

**pos** Candidate's ideological position

**list** List ID

**party** Party ID

---

## predict_iii *Predict II Score for a given set of electoral rule configurations*

---

### Description

Predict II Score for a given set of electoral rule configurations

### Usage

```
predict_iii(
  data,
  score = c("TDE", "AP"),
  district_level = TRUE,
  return_avg = TRUE
)
```

### Arguments

| | |
|---|---|
| data | A data.frame containing the following variables: ballot_type (factor), pool_level (factor), votes_per_voter (factor), M (numeric), threshold (numeric), and formula (factor). See simulate_election() for more details. |
| score | Character string indicating type of score to predict; one of TDE (default) or AP. |

district_level Boolean: Should district level, or country level models be used? If TRUE (default), the function uses district level models, which are more accurate for district-level electoral systems. If FALSE, the function uses country-level models, which are more accurate for country-level electoral systems.

return_avg Boolean: Should the average score across imputed models be returned? The original models were trained on millions of simulated elections, with intermediate values for some parameters interpolated using 5 multiple imputations. If TRUE (default), the function returns the average score across all imputations. If FALSE, the function returns a list of scores.

## Value

Predicted TDE or AP score for given electoral system

## Examples

```
## Create example data for PR system with closed party lists,
## magnitude 5, and Droop quota
new_system <- data.frame(ballot_type = as.factor("closed"),
                         pool_level = as.factor("party"),
                         votes_per_voter = as.factor("One"),
                         M = 5.0,
                         threshold = 0.05,
                         formula = as.factor("droop"))
predict_iii(data = new_system, score = "AP", district_level = FALSE)
```

---

# ranked *Example of ranked vote matrix*

---

## Description

This is an example of a ranked vote matrix for 4 voters and 3 candidates. It can be used to illustrate different seat allocation formulas that require ranked votes.

## Usage

```
ranked
```

## Format

A matrix with 3 rows and 4 columns, where each row represents a ranking, each column is a voter, and each cell is the candidate ID (numeric or character) #' that the voter ranked in that position.

**Voter1** Candidates ranked first, second, and third by the first voter

**Voter2** Candidates ranked first, second, and third by the second voter ...

## Source

Created by the package authors for demonstration purposes.

`simulate_election` *Function to simulate a full election in a single district*

## Description

The function runs a complete election in a single district, using the simulation framework described in detail in Chapter 4 of Crisp et al. 2024.

## Usage

```
simulate_election(
  voters = NULL,
  parties = NULL,
  cands = NULL,
  nominated = NULL,
  nvoters = 3000,
  nparties = 5,
  nvotes = 1,
  M = 5,
  rank_cand = TRUE,
  strategic = TRUE,
  strategic_error = 0.05,
  who_ranks = c("parties", "voters", "none"),
  gamma_val = NULL,
  gamma_rank = 1,
  elec_fun_name = "dhondt",
  ballot_type = "open",
  primary = FALSE,
  two_round = FALSE,
  pool_level = c("party_list", "party", "candidate"),
  ranked_vote = FALSE,
  free_vote = FALSE,
  max_cand = 0,
  threshold = 0,
  lists_per_party = 1,
  seed = 123,
  elec_results_only = FALSE,
  multiplier = 1,
  system_name,
  ...
)
```

## Arguments

| | |
|---|---|
| `voters` | Optional vector of voter positions in 1d ideological space. |
| `parties` | Optional vector of party positions in 1d ideological space. Maximum of 10 parties allowed. |

| | |
|---|---|
| cands | Optional matrix with three columns: candidate 1d ideological position, unique numerical candidate ID, and positive numerical candidate valence |
| nominated | Optional data.frame with five variables: rank (candidate ranking in the party list); candidate (numeric candidate ID); pos (1d ideological position of candidate); list (numeric list ID; equal to 1, unless parties are allowed to have multiple lists); party (numeric party ID). |
| nvoters | Number of voters; defaults to 3,000. |
| nparties | Number of parties; defaults to 5; maximum allowable: 10. |
| nvotes | Number of votes per voter; defaults to 1. Can also take on special values 0 (which then is internally replaced by the district magnitude) and -1 (which is then internally replaced by 1 fewer vote than the district magnitude). |
| M | District magnitude; defaults to 5. |
| rank_cand | Boolean: should candidates be ranked on the party list? Defaults to TRUE. |
| strategic | Boolean: do parties and voters behave strategically? Defaults to TRUE. |
| strategic_error | Numeric probability with which strategic actors fail to choose the optimal alternative. |
| who_ranks | Character actor who arranges party lists, one of parties, voters,none; defaults to parties. |
| gamma_val | Numeric weight assigned to the valence component of voters' utility function. |
| gamma_rank | Numeric weight assigned to the candidate ranking on the party list when computing the voter's utility. |
| elec_fun_name | Name of function implementing electoral system formula. |
| ballot_type | Character string indicating type of ballot, one of open,closed, or flexible; defaults to open. |
| primary | Boolean: should a primary election be conducted? Defaults to FALSE. |
| two_round | Boolean: should a second election round be conducted? Defaults to FALSE. |
| pool_level | Character level at which votes are pooled, one of party_list (or sub-party list),party, or candidate. Defaults to party_list |
| ranked_vote | Boolean: Do voters cast a ranked vote? Defaults to FALSE. |
| free_vote | Boolean: If voters can cast multiple votes, can the be for candidates in different parties? Defaults to FALSE. |
| max_cand | Numeric maximum number of candidates running in a party list; defaults to 0, which is internally interpreted as the district magnitude. |
| threshold | Numerical legal electoral threshold; defaults to 0 (i.e., no threshold). |
| lists_per_party | Integer allowed number of lists per party; defaults to 1. |
| seed | Random number generator seed; defaults to 123. |
| elec_results_only | Boolean: Should function return ancillary information on election, or just election results? Defaults to FALSE. |

`multiplier` Numeric factor by which to multiply the votes cast by voters with the same ideological position; defaults to 1.

`system_name` Character name of electoral system used, one of 'AV', 'BC', 'STV', 'MNTV', 'LV', 'PR', or 'SMDP'

`...` Additional arguments passed to `elec_fun_name`.

## Value

data.frame with the following variables (if `elec_results_only=FALSE`, otherwise, data.frame with candidate id's, positions, valences, votes obtained, and whether they won a seat or not):

**gamma_val** See `Usage` above

**epsilon** Maximum acceptable ideological distance used in voters' utility function

**hetero** Measure of elected candidate heterogeneity

**pers** Average valence of elected candidates

**lsq** Least Squares measure of disproportionality

**enp_v** Effective number of electoral parties

**enp_s** Effective number of legislative parties

**avg_dist** Average distance between elected candidates and voters

**var_elect** Variance of ideological positions of elected candidates

**avg_vote_util** Average utility of voters w.r.t. candidates they voted for

**avg_elect_util** Average utility of voters w.r.t. elected candidates

**sample_parties** Parties that initially could have entered the election

**ran_parties** Parties that decided to enter the election

## Examples

```
# Simulate a PR (D'Hondt) election with 3 parties, 5 candidates per party,
# 100 voters, and a district magnitude of 2, allowing for strategic voting

simulate_election(parties = c(-1, 0, 1),
                  nvoters = 100,
                  M = 2,
                  strategic = TRUE,
                  elec_fun_name = "dhondt",
                  system_name = "PR")
```

`voting` *Function to simulate the voting process*

## Description

Internal function.

## Usage

```
voting(
  voters,
  nominated,
  n_votes,
  gamma_val,
  gamma_rank,
  epsilon,
  free = TRUE,
  closed_primary = FALSE,
  strategic = FALSE,
  strategic_error = 0.05,
  party_pos = NULL
)
```

## Arguments

| | |
|---|---|
| `voters` | See `simulate_election()`. |
| `nominated` | See `simulate_election()`. |
| `n_votes` | See `simulate_election()`. |
| `gamma_val` | See `simulate_election()`. |
| `gamma_rank` | See `simulate_election()`. |
| `epsilon` | Numeric; maximum acceptable ideological distance used in voters' utility function |
| `free` | See `simulate_election()`. Defaults to `TRUE`. |
| `closed_primary` | Boolean: Are voters required to vote for a candidate in the party closest to them in the primary? Defaults to `FALSE`. |
| `strategic` | See `simulate_election()`. |
| `strategic_error` | See `simulate_election()`. Defaults to 0.05 |
| `party_pos` | Locations of parties in the election in 1d space (-2, 2). |

## Value

List with two elements:

**votes** Matrix with `n_votes` rows and `length(voters)` columns, with cells populated with candidate IDs

**max_utils** Vector of maximum utilities received by each voter from among all candidates in the election

# Index