Package 'fastei'

March 11, 2025

Title Methods for a Fast Alternative for the R x C Ecological Inference Problem.

Version 0.0.0.1

Author Charles Thraves, Pablo Ubilla, Daniel Hermosilla

Maintainer Daniel Hermosilla <daniel.hermosilla.r@ug.uchile.cl>

Description Estimates the probability matrix for the R×C Ecological Inference problem using the Expectation-Maximization Algorithm with four approximation methods for the E-Step, and an exact method as well. Additionally, it provides a bootstrap function to estimate the standard deviation on the estimated probabilities.

License MIT + file LICENSE

Encoding UTF-8

Imports Rcpp,
jsonlite

LinkingTo Rcpp

NeedsCompilation yes

BugReports
https://github.com/DanielHermosilla/ecological-inference-elections/issues

Suggests knitr,
rmarkdown

Roxygen list(markdown = TRUE, load = ``source")

VignetteBuilder knitr

Depends R (>= 3.5)

LazyData true

RoxygenNote 7.3.2

Contents

`fastei-package` *fastei: Methods for "A Fast Ecological Inference Algorithm for the R×C case"*

Description

Package that implements the methods of Thraves, C. and Ubilla, P. (2024): *"A Fast Ecological Inference Algorithm for the R×C Case"*.

Details

Includes a method (run_em) to solve the R×C Ecological Inference problem for the non-parametric case by using the EM algorithm with different approximation methods for the E-Step. The standard deviation of the estimated probabilities can be computed using bootstrapping (bootstrap).

It also provides a function that generates synthetic election data (simulate_election) and a function that imports real election data (chilean_election_2021) from the Chilean first-round presidential election of 2021.

The setting in which the documentation presents the Ecological Inference problem is an election context where for a set of ballot-boxes we observe (i) the votes obtained by each candidate and (ii) the number of voters of each demographic group (for example, these can be defined by age ranges or sex). See Thraves, C. and Ubilla, P. (2024): *"A Fast Ecological Inference Algorithm for the R×C Case"*.

The methods to compute the conditional probabilities of the E-Step included in this package are the following:

- **Hit and Run** (`hnr`): Performs MCMC to sample vote outcomes for each ballot-box consistent with the observed data. This sample is used to estimate the conditional probability of the *E*-Step.
- **Multivariate Normal PDF** (`mvn_pdf`): Uses the PDF of a Multivariate Normal to approximate the conditional probability.
- **Multivariate Normal CDF** (`mvn_cdf`): Uses the CDF of a Multivariate Normal to approximate the conditional probability.
- **Multinomial** (`mult`): A single Multinomial is used to approximate the sum of Multinomial distributions.
- **Exact** (`exact`): Solves the E-Step exactly using the Total Probability Law, which requires enumerating an exponential number of terms.

On average, the **Multinomial** method is the most efficient and precise. Its precision matches the **Exact** method.

The documentation uses the following notation:

- b: number of ballot-boxes.
- g: number of demographic groups.
- c: number of candidates.
- a: number of aggregated macro-groups.

To learn more about `fastei`, please consult the available vignettes:

```
browseVignettes("fastei")
```

References

Thraves, C. and Ubilla, P. (2024): *"A Fast Ecological Inference Algorithm for the R×C Case"*.

See Also

Useful links:

- Report bugs at https://github.com/DanielHermosilla/ecological-inference-elections/issues

---

`bootstrap` *Runs a Bootstrap to Estimate the* **Standard Deviation** *of Predicted Probabilities*

Description

This function computes the Expected-Maximization (EM) algorithm "`nboot`" times. It then computes the standard deviation from the `nboot` estimated probability matrices on each component.

Usage

```
bootstrap(
 object = NULL,
 X = NULL,
 W = NULL,
 json_path = NULL,
 nboot = 50,
 seed = NULL
 ...
)
```

Arguments

`object`	An object of class `eim`, which can be created using the eim function. This parameter should not be used if either (i) `X` and `W` matrices or (ii) `json_path` is supplied. See **Note**.
`X`	A `(b x c)` matrix representing candidate votes per ballot box.
`W`	A `(b x g)` matrix representing group votes per ballot box.
`json_path`	A path to a JSON file containing `X` and `W` fields, stored as nested arrays.
`nboot`	Integer specifying how many times to run the EM algorithm.
`seed`	If provided, overrides the current global seed. Defaults to `NULL`.
`...`	Additional arguments passed to the run_em function that will execute the EM algorithm.

Value

Returns an `eim` object with the `sd` field containing the estimated standard deviations of the probabilities. If an `eim` object is provided, its attributes (see run_em) are retained in the returned object.

Note

This function can be executed using one of three mutually exclusive approaches:

1. By providing an existing `eim` object.
2. By supplying both input matrices (`X` and `W`) directly.
3. By specifying a JSON file (`json_path`) containing the matrices.

These input methods are **mutually exclusive**, meaning that you must provide **exactly one** of these options. Attempting to provide more than one or none of these inputs will result in an error.

When called with an `eim` object, the function updates the object with the computed results. If an `eim` object is not provided, the function will create one internally using either the supplied matrices or the data from the JSON file before executing the algorithm.

See Also

The eim object and run_em implementation.

Examples

```
# Example 1: Using an 'eim' object directly
simulations <- simulate_election(
    num_ballots = 200,
    num_candidates = 5,
    num_groups = 3,
)

model <- eim(X = simulations$X, W = simulations$W)

model <- bootstrap(
    object = model,
    nboot = 30,
    method = "mvn_cdf",
    maxiter = 500,
    verbose = TRUE,
    mc_samples = 5000
)

# Access standard deviation throughout 'model'
print(model$sd)


# Example 2: Providing 'X' and 'W' matrices directly
model <- bootstrap(
    X = simulations$X,
    W = simulations$W,
    nboot = 70,
    method = "mvn_pdf",
    maxiter = 100,
    maxtime = 15,
    stop_threshold = 0.01
)

print(model$sd)

# Example 3: Using a JSON file as input
```

```
## Not run:
model <- bootstrap(
    json_path = "path/to/election_data.json",
    nboot = 70,
    method = "mult",
)

print(model$sd)

## End(Not run)
```

`chile_election_2021` *Chilean 2021 First Round Presidential Election*

Description

This dataset contains the results of the first round of the 2021 Chilean presidential elections. It provides 9 possible voting options (7 candidates, blank, and null). Each ballot-box is identified by its id (`BALLOT.BOX`) and an electoral circumscription (`ELECTORAL.DISTRICT`). Additionally, it provides demographic information on voters' age range for each ballot box.

Usage

```
data("chile_election_2021")
```

Format

A data frame with 46,639 rows and 14 variables:

`ELECTORAL.DISTRICT` The electoral circumscription of the ballot box.

`BALLOT.BOX` An identifier for the ballot box within the `ELECTORAL.DISTRICT`.

`C1` The number of votes cast for the candidate *Gabriel Boric*.

`C2` The number of votes cast for the candidate *José Antonio Kast*.

`C3` The number of votes cast for the candidate *Yasna Provoste*.

`C4` The number of votes cast for the candidate *Sebastián Sichel*.

`C5` The number of votes cast for the candidate *Eduardo Artés*.

`C6` The number of votes cast for the candidate *Marco Enríquez-Ominami*.

`C7` The number of votes cast for the candidate *Franco Parisi*.

`BLANK.VOTES` The number of blank votes.

`NULL.VOTES` The number of null votes.

`X18.19` Number of voters aged 18–19.

`X20.29` Number of voters aged 20–29.

`X30.39` Number of voters aged 30–39.

`X40.49` Number of voters aged 40–49.

`X50.59` Number of voters aged 50–59.

`X60.69` Number of voters aged 60–69.

`X70.79` Number of voters aged 70–79.

`X80.` Number of voters aged 80 and older.

`MISMATCH` Boolean that takes value `TRUE` if the ballot-box has a mismatch between the total number of votes and the total number of voters. If this is not the case, its value is `FALSE`.

Source

Chilean Electoral Service (SERVEL)

Examples

```
data("chile_election_2021")
head(chile_election_2021)
```

`eim` *S3 Object for the Expectation-Maximization Algorithm*

Description

This constructor creates an `eim` S3 object, either by using matrices `X` and `W` directly or by reading them from a JSON file. Each `eim` object encapsulates the data (votes for candidates and demographic groups) required by the underlying Expectation-Maximization algorithm.

Usage

```
eim(X = NULL, W = NULL, json_path = NULL)
```

Arguments

`X`	A `(b x c)` matrix representing candidate votes per ballot box.
`W`	A `(b x g)` matrix representing group votes per ballot box.
`json_path`	A path to a JSON file containing `X` and `W` fields, stored as nested arrays.

Details

If `X` and `W` are directly supplied, they must match the dimensions of ballot boxes `(b)`. Alternatively, if `json_path` is provided, the function expects the JSON file to contain elements named `"X"` and `"W"` under the top-level object. This two approaches are **mutually exclusable**, yielding an error otherwise.

Internally, this function also initializes the corresponding instance within the low-level (C-based) API, ensuring the data is correctly registered for further processing by the EM algorithm.

Value

A list of class `eim` containing:

`X` The candidate votes matrix `(b x c)`.

`W` The group votes matrix `(b x g)`.

Methods

In addition to this constructor, the "eim" class provides several S3 methods for common operations. Some of these methods are fully documented, while others are ommited due to its straightfoward implementantion. The available methods are:

- run_em – Runs the EM algorithm.
- bootstrap – Estimates the standard deviation of the probabilities.
- save – Save the results to a specified file.
- get_agg_proxy – Estimates an ideal group aggregation given their standard deviations.
- `print` – Print useful information about the object.
- `summary` – Shows, in form of a list, the most important attributes.
- `as.matrix` – Returns the probability matrix.
- `write.csv` – Writes the probability matrix in a `.csv` file.
- `dput` – Writes the object in a `.rda` file.
- `logLik` – Returns the log-likelihood from the last iteration.

Note

A way to generate synthetic data for `X` and `W` is by using the simulate_election function. See Example 2 below.

Examples

```
# Example 1: Create an eim object from a JSON file
## Not run:
model1 <- eim(json_path = "path/to/file.json")

## End(Not run)

# Example 2: Use simulate_election with optional parameters, then create an eim object
# from matrices

# Simulate data for 500 ballot boxes, 4 candidates and 5 groups
sim_result <- simulate_election(
    num_ballots = 500,
    num_candidates = 3,
    num_groups = 5,
    group_proportions = c(0.2, 0.2, 0.4, 0.1, 0.1)
)

model2 <- eim(X = sim_result$X, W = sim_result$W)

# Example 3: Create an object from a user defined matrix with 8 ballot boxes,
# 2 candidates and 7 groups.

x_mat <- matrix(c(
    57, 90,
    60, 84,
    43, 102,
    72, 71,
    63, 94,
    52, 80,
```

```
    60, 72,
    54, 77,
), nrow = 8, ncol = 2, byrow = TRUE)

w_mat <- matrix(c(
    10, 15, 25, 21, 10, 40, 26,
    11, 21, 37, 32, 8, 23, 12,
    17, 12, 43, 27, 12, 19, 15,
    20, 18, 25, 15, 22, 17, 26,
    21, 19, 27, 16, 23, 22, 29,
    18, 16, 20, 14, 19, 22, 23,
    10, 15, 21, 18, 20, 16, 32,
    12, 17, 19, 22, 15, 18, 28
), nrow = 8, ncol = 7, byrow = TRUE)

model3 <- eim(X = x_mat, W = w_mat)
```

`get_agg_proxy` *Runs the EM algorithm aggregating adjacent groups, maximizing the variability of macro-group allocation in ballot boxes.*

Description

This function estimates the voting probabilities (computed using run_em) aggregating adjacent groups so that the estimated probabilities' standard deviation (computed using bootstrap) is below a given threshold. See **Details** for more information.

Usage

```
get_agg_proxy(
  object = NULL,
  X = NULL,
  W = NULL,
  json_path = NULL,
  sd_statistic = "maximum",
  sd_threshold = 0.05,
  nboot = 50,
  seed = NULL,
  ...
)
```

Arguments

`object`	An object of class `eim`, which can be created using the eim function. This parameter should not be used if either (i) `X` and `W` matrices or (ii) `json_path` is supplied. See **Note** in run_em.
`X`	A `(b x c)` matrix representing candidate votes per ballot box.
`W`	A `(b x g)` matrix representing group votes per ballot box.
`json_path`	A path to a JSON file containing `X` and `W` fields, stored as nested arrays.

`sd_statistic`	String indicates the statistic for the standard deviation (`g x c`) matrix for the stopping condition, i.e., the algorithm stops when the statistic is below the threshold. It can take the value `maximum`, in which case computes the maximum over the standard deviation matrix, or `average`, in which case computes the average.
`sd_threshold`	Numeric with the value to use as a threshold for the statistic (`sc_statistic`) of the standard deviation of the estimated probabilities. Defaults to 0.05.
`nboot`	Integer specifying how many times to run the EM algorithm.
`seed`	If provided, overrides the current global seed. Defaults to `NULL`.
`...`	Additional arguments passed to the run_em function that will execute the EM algorithm.

Details

Groups need to have an order relation so that adjacent groups can be merged. For example, consider the following seven groups defined by voters' age ranges: 20-29, 30-39, 40-49, 50-59, 60-69, 70-79, and 80+. A possible group aggregation can be a macro-group composed of the three following age ranges: 20-39, 40-59, and 60+. Since there are multiple group aggregations, even for a fixed number of macro-groups, a Dynamic Program (DP) mechanism is used to find the group aggregation that maximizes the sum of the standard deviation of the macro-groups proportions among ballot boxes for a specific number of macro-groups.

In order to find the best group aggregation, the function runs the DP iteratively, starting with all groups (this case is trivial since, in this case, the group aggregation is such that all macro-groups match exactly the original groups). If the standard deviation statistic is below a threshold, it stops. Otherwise, it runs the DP such that the number of macro-groups is one unit less than the original number of macro-groups. If the standard deviation statistic (`sd_statistic`) is below the threshold (`sd_threshold`), it stops. And so on until either the algorithm stops, or until the group aggregation composed of a single macro-group does not meet the stopping condition, in which case the output is the EM algorithm run over a single macro-group.

Value

It returns an eim object with the same attributes as the output of run_em, plus the attributes:

- sd: A (`g x a`) matrix with the standard deviation of the estimated probabilities computed with bootstrapping. Note that `a` denotes the number of macro-groups of the resulting group aggregation, it should be between 2 and `g`.
- sd_statistic: The statistic used as input.
- sd_threshold: The threshold used as input.
- group_agg: Vector with the resulting group aggregation. See **Examples** for more details.

Aditionally, it will update the `W` attribute with the aggregated groups.

See Also

The eim object and run_em implementation.

Examples

```
# Example 1: Using a simulated instance
simulations <- simulate_election(
    num_ballots = 400,
    num_candidates = 3,
    num_groups = 6,
    group_proportions = c(0.4, 0.1, 0.1, 0.1, 0.2, 0.1),
    lambda = 0.7
)

result <- get_agg_proxy(
    X = simulations$X,
    W = simulations$W,
    sd_threshold = 0.01
)

result$group_agg # c(2, 6)
# This would mean that the ideal group aggregation would
# be {[1, 2], [3, 6]}

# Example 2: Using the chilean election results
data(chile_election_2021)

niebla_df <- chile_election_2021[chile_election_2021$ELECTORAL.DISTRICT == "NIEBLA", ]

# Create the X matrix with selected columns
X <- as.matrix(niebla_df[, c("C1", "C2", "C3", "C4", "C5", "C6", "C7")])

# Create the W matrix with selected columns
W <- as.matrix(niebla_df[, c(
    "X18.19", "X20.29",
    "X30.39", "X40.49",
    "X50.59", "X60.69",
    "X70.79", "X80."
)])

solution <- get_agg_proxy(
    X = X, W = W,
    allow_mismatch = TRUE, sd_threshold = 0.03,
    sd_statistic = "average", nboot = 100
)

solution$group_agg # c(3, 4, 5, 6, 7)
# This would mean that the ideal group aggregation would
# be {[1, 3], [4], [5], [6], [7, 8]}
```

`run_em` *Compute the Expected-Maximization Algorithm*

Description

Executes the Expectation-Maximization (EM) algorithm indicating the approximation method to use in the E-step. Certain methods may require additional arguments, which can be passed through ... (see fastei-package for more details).

Usage

```
run_em(
 object = NULL,
 json_path = NULL,
 method = "mult",
 initial_prob = "group_proportional",
 allow_mismatch = FALSE,
 maxiter = 1000,
 maxtime = 3600,
 stop_threshold = 0.001,
 log_threshold = as.double(-Inf),
 verbose = FALSE,
 ...
)
```

Arguments

`object`	An object of class `eim`, which can be created using the eim function. This parameter should not be used if either (i) `X` and `W` matrices or (ii) `json_path` is supplied. See **Note**.
`X`	A `(b x c)` matrix representing candidate votes per ballot box.
`W`	A `(b x g)` matrix representing group votes per ballot box.
`json_path`	A path to a JSON file containing `X` and `W` fields, stored as nested arrays.
`method`	An optional string specifying the method used for estimating the E-step. Valid options are: • `mult`: The default method, using a single sum of Multinomial distributions. • `mvn_cdf`: Uses a Multivariate Normal CDF distribution to approximate the conditional probability. • `mvn_pdf`: Uses a Multivariate Normal PDF distribution to approximate the conditional probability. • `hnr`: Uses MCMC to sample vote outcomes. This is used to estimate the conditional probability of the E-step. • `exact`: Solves the E-step using the Total Probability Law. For a detailed description of each method, see fastei-package and **References**.
`initial_prob`	An optional string specifying the method used to obtain the initial probability. Accepted values are: • `uniform`: Assigns equal probability to every candidate within each group. • `proportional`: Assigns probabilities to each group based on the proportion of candidates votes. • `group_proportional`: Computes the probability matrix by taking into account both group and candidate proportions. This is the default method.
`allow_mismatch`	Boolean, if `TRUE`, allows a mismatch between the voters and votes for each ballot-box, only works if method is `mvn_cdf`, `mvn_pdf`, and `mult`. If `FALSE`, throws an error if there is a mismatch. By default it is `FALSE`.
`maxiter`	An optional integer indicating the maximum number of EM iterations. The default value is `1000`.
`maxtime`	An optional numeric specifying the maximum running time (in seconds) for the algorithm. This is checked at every iteration of the EM algorithm. The default value is `3600`, which corresponds to an hour.

`stop_threshold` An optional numeric value indicating the minimum difference between consecutive probability values required to stop iterating. The default value is `0.001`. Note that the algorithm will stop if either `log_threshold` **or** `stop_threshold` is accomplished.

`log_threshold` An optional numeric value indicating the minimum difference between consecutive log-likelihood values to stop iterating. The default value is `inf`, essentially deactivating the threshold. Note that the algorithm will stop if either `log_threshold` **or** `stop_threshold` is accomplished.

`verbose` An optional boolean indicating whether to print informational messages during the EM iterations. The default value is `FALSE`.

`step_size` An optional integer specifying the step size for the `hnr` algorithm. This parameter is only applicable when `method = hnr` and will be ignored otherwise. The default value is `3000`.

`samples` An optional integer indicating the number of samples to generate for the **Hit and Run** method. This parameter is only relevant when `method = hnr`. The default value is `1000`.

`mc_method` An optional string specifying the method used to estimate the `mvn_cdf` method via a Monte Carlo simulation. Accepted values are `genz` and `genz2`, with `genz2` set as the default. This parameter is only applicable when `method = mvn_cdf`. See **References** for more details.

`mc_error` An optional numeric value defining the error threshold for the Monte Carlo simulation when estimating the `mvn_cdf` method. The default value is `1e-6`. This parameter is only relevant when `method = mvn_cdf`.

`mc_samples` An optional integer specifying the number of Monte Carlo samples for the `mvn_cdf` method. The default value is `5000`. This argument is only applicable when `method = mvn_cdf`.

Value

The function returns an 'eim' object with the following attributes:

prob The estimated probability matrix `(g x c)`.

cond_prob A `(b x g x c)` 3d-array with the probability that a at each ballot-box a voter of each group voted for each candidate, given the observed outcome at the particular ballot-box.

logLik The log-likelihood value from the last iteration.

iterations The total number of iterations performed by the EM algorithm.

time The total execution time of the algorithm in seconds.

status The final status ID of the algorithm upon completion:

- `0`: Convergence achieved.
- `1`: Maximum time reached.
- `2`: Maximum iterations reached.

message The finishing status displayed as a message.

method The method for estimating the conditional probability in the E-step.

Aditionally, it will create `samples` and `step_size` parameters if the specified method is `hnr`, or `mc_method`, `mc_error` and `mc_samples` if the method is `mvn_cdf`.

Note

This function can be executed using one of three mutually exclusive approaches:

1. By providing an existing `eim` object.
2. By supplying both input matrices (`X` and `W`) directly.
3. By specifying a JSON file (`json_path`) containing the matrices.

These input methods are **mutually exclusive**, meaning that you must provide **exactly one** of these options. Attempting to provide more than one or none of these inputs will result in an error.

When called with an `eim` object, the function updates the object with the computed results. If an `eim` object is not provided, the function will create one internally using either the supplied matrices or the data from the JSON file before executing the algorithm.

References

Thraves, C. and Ubilla, P.: *"Fast Ecological Inference Algorithm for the R×C Case"*. Aditionally, the MVN CDF is computed by the methods introduced in Genz, A. (2000). Numerical computation of multivariate normal probabilities. *Journal of Computational and Graphical Statistics*

See Also

The eim object implementation.

Examples

```
# Example 1: Compute the Expected-Maximization with default settings
simulations <- simulate_elections(
    num_ballots = 300,
    num_candidates = 5,
    num_groups = 3,
)
model <- eim(simulations$X, simulations$W)
model <- run_em(model) # Returns the object with updated attributes

# Example 2: Compute the Expected-Maximization using the Hit and Run method
model <- run_em(
    object = model,
    method = "hnr",
    step_size = 1500,
    samples = 800
)

# Example 3: Run the MVN CDF method with default settings
model <- run_em(object = model, method = "mvn_cdf")

# Example 4: Perform an Exact estimation using user-defined parameters
## Not run:
run_em(
    json_path = "a/json/file.json",
    method = "exact",
    initial_prob = "uniform",
    maxiter = 10,
    maxtime = 600,
    stop_threshold = 1e-3,
    log_threshold = 1e-5,
```

```
    verbose = TRUE
)

## End(Not run)
```

`save_eim` *Save an* `eim` *object to a file*

Description

This function saves an `eim` object to a specified file format. Supported formats are **RDS**, **JSON**, and **CSV**. The function dynamically extracts and saves all available attributes when exporting to JSON. If the `prob` field exists, it is saved when using CSV; otherwise, it yields an error.

Usage

```
save_eim(object, filename, ...)
```

Arguments

`object`	An `eim` object.
`filename`	A character string specifying the file path, including the desired file extension (`.rds`, `.json`, or `.csv`).
`...`	Additional arguments (currently unused but included for compatibility).

Details

- If the file extension is **RDS**, the entire object is saved using `saveRDS()`.
- If the file extension is **JSON**, all available attributes of the object are stored in JSON format.
- If the file extension is **CSV**:
  - If the object contains a `prob` field, only that field is saved as a CSV.
  - Otherwise, returns an error.

Value

The function does not return anything explicitly but saves the object to the specified file.

See Also

The eim object implementation.

Examples

```
model <- eim(X = matrix(1:9, 3, 3), W = matrix(1:9, 3, 3))

run_em(model)

# Save as RDS
save_eim(model, "model_results.rds")
```

```
# Save as JSON
save_eim(model, "model_results.json")

# Save as CSV
save_eim(model, "model_results.csv")
```

`simulate_election` *Simulate an Election*

Description

This function simulates an election by creating matrices representing candidate votes (`X`) and voters' demographic group (`W`) across a specified number of ballot-boxes. It either (i) receives as input or (ii) generates a probability matrix (`prob`), indicating how likely each demographic group is to vote for each candidate.

By default, the number of voters per ballot box '(ballot_voters)' is set to a vector of 100 with length 'num_ballots'. You can optionally override this by providing a custom vector.

Optional parameters are available to control the distribution of votes:

- `group_proportions`: A vector of length `num_groups` specifying the overall proportion of each demographic group. Entries must sum to one and be non-negative.
- `prob`: A user-supplied probability matrix of dimension (`num_groups` $\times$ `num_candidates`). If provided, this matrix is used directly. Otherwise, voting probabilities for each group are drawn from a Dirichlet distribution.

Usage

```
simulate_election(
  num_ballots,
  num_candidates,
  num_groups,
  ballot_voters = rep(100, num_ballots),
  lambda = 0.5,
  seed = NULL,
  group_proportions = rep(1/num_groups, num_groups),
  prob = NULL
)
```

Arguments

`num_ballots`	Number of ballot boxes (b).
`num_candidates`	Number of candidates (c).
`num_groups`	Number of demographic groups (g).
`ballot_voters`	A vector of length `num_ballots` representing the number of voters per ballot box. Defaults to `rep(100, num_ballots)`.
`lambda`	A numeric value between 0 and 1 that represents the fraction of voters that are randomly assigned to ballot-boxes. The remaining voters are assigned sequentially according to their demographic group.

- `lambda = 0`: The assignment of voters to ballot-boxes is fully sequential in terms of their demographic group. This leads to a **high heterogeneity** of the voters' groups across ballot-boxes.
- `lambda = 1`: The assignment of voters to ballot-boxes is fully random. This leads to a **low heterogeneity** of the voters' group across ballot-boxes.

Default value is set to `0.5`. See **Shuffling Mechanish** for more details.

`seed` If provided, overrides the current global seed. Defaults to `NULL`.

`group_proportions`
Optional. A vector of length `num_groups` that indicates the fraction of voters that belong to each group. Default is that all groups are of the same size.

`prob` Optional. A user-supplied probability matrix of dimension `(g x c)`. If provided, this matrix is used as the underlying voting probability distribution. If not supplied, each row is sampled from a Dirichlet distribution with each parameter set to one.

Value

A list with three components:

`X` A `(b x c)` matrix with candidates' votes for each ballot box.

`W` A `(b x g)` matrix with voters' groups for each ballot-box.

`prob` A `(g x c)` matrix with the probability that a voter from each group votes for each candidate. If prob is provided, it would equal such probability.

Shuffling Mechanism

Without loss of generality, consider an order relation of groups and ballot-boxes. The shuffling step is controlled by the `lambda` parameter and operates as follows:

1. **Initial Assignment**: Voters are assigned to each ballot-box sequentially according to their demographic group. More specifically, the first ballot-boxes receive voters from the first group. Then, the next ballot-boxes receive voters from the second group. This continues until all voters have been assigned. Note that most ballot-boxes will contain voters from a single group (as long as the number of ballot-boxes exceeds the number of groups).
2. **Shuffling**: A fraction `lambda` of voters who have already been assigned is selected at random. Then, the ballot-box assignment of this sample is shuffled. Hence, different `lambda` values are interpreted as follows:
   - `lambda = 0` means no one is shuffled (the initial assignment remains).
   - `lambda = 1` means all individuals are shuffled.
   - Intermediate values like `lambda = 0.5` shuffle half the voters.

Using a high level of `lambda` (greater than 0.6) is not recommended, as this could make identification of the voting probabilities difficult. This is because higher values of lambda induce similar ballot-boxes in terms of voters' group.

References

The algorithm is fully explained in 'Thraves, C. and Ubilla, P.: *"A Fast Ecological Inference Algorithm for the R×C Case"*.

Examples

```
# Example 1: Default usage with 200 ballot boxes, each having 100 voters
result1 <- simulate_election(
    num_ballots = 200,
    num_candidates = 3,
    num_groups = 5
)

# Example 2: Using a custom ballot_voters vector
result2 <- simulate_election(
    num_ballots = 340,
    num_candidates = 3,
    num_groups = 7,
    ballot_voters = rep(200, 340)
)

# Example 3: Supplying group_proportions
result3 <- simulate_election(
    num_ballots = 93,
    num_candidates = 3,
    num_groups = 4,
    group_proportions = c(0.3, 0.5, 0.1, 0.1)
)

# Example 4: Providing a user-defined prob matrix
custom_prob <- matrix(c(
    0.9,  0.1,
    0.4,  0.6,
    0.25, 0.75,
    0.32, 0.68,
    0.2,  0.8
), nrow = 5, byrow = TRUE)

result4 <- simulate_election(
    num_ballots = 200,
    num_candidates = 2,
    num_groups = 5,
    alpha = 0.3,
    prob = custom_prob
)

result4$prob == custom_prob # TRUE
```

Index