Package ‘latentFactoR’

April 18, 2024

Title Data Simulation Based on Latent Factors
Version 0.0.6
Date 2024-04-17
Maintainer Alexander Christensen <alexpaulchristensen@gmail.com>
Description Generates data based on latent factor models. Data can be continuous, polytomous, dichotomous, or mixed. Skews, cross-loadings, wording effects, population errors, and local dependencies can be added. All parameters can be manipulated. Data categorization is based on Garrido, Abad, and Ponsoda (2011) <doi:10.1177/0013164410389489>.
Depends R (>= 3.6.0)
License GPL (>= 3.0)
Imports BBmisc, EGAnet, fspe, googledrive, ineq, lavaan, Matrix, methods, mlr, mvtnorm, psych, rstudioapi, xgboost
Suggests ggplot2
Encoding UTF-8
RoxygenNote 7.3.1
NeedsCompilation no
Author Alexander Christensen [aut, cre]
(Luis Eduardo Garrido [aut] (<https://orcid.org/0000-0001-8932-6063>),
Maria Dolores Nieto Canaveras [aut],
Hudson Golino [aut] (<https://orcid.org/0000-0002-1601-1447>),
Marcos Jimenez [aut],
Francisco Abad [ctb],
Eduardo Garcia-Garzon [ctb],
Vithor Franco [aut]
Repository CRAN
Date/Publication 2024-04-18 21:23:04 UTC

R topics documented:

latentFactoR-package ............................................. 2
Description

Generates data based on latent factor models. Data can be continuous, polytomous, dichotomous, or mixed. Skew, cross-loadings, and population error can be added. All parameters can be manipulated. Data categorization is based on Garrido, Abad, and Ponsoda (2011).

Author(s)

Alexander P. Christensen <alexpaulchristensen@gmail.com>, Maria Dolores Nieto Canaveras <mnietoca@nebrija.es>, Hudson Golino <hfg9s@virginia.edu>, Luis Eduardo Garrido <luisgarrido@pucmm.edu>

References


Description

Intended to add substantial cross-loadings to simulated data from `simulate_factors`. See examples to get started.

Usage

```r
add_cross_loadings(
  lf_object,
  proportion_cross_loadings,
  proportion_cross_loadings_range = NULL,
  magnitude_cross_loadings,
  magnitude_cross_loadings_range = NULL,
  leave_cross_loadings = FALSE
)
```

Arguments

- `lf_object`: Data object from `simulate_factors`
- `proportion_cross_loadings`: Numeric (length = 1 or factors). Proportion of variables that should be cross-loaded randomly onto one other factor. Accepts number of variables to cross-load onto one other factor as well.
- `proportion_cross_loadings_range`: Numeric (length = 2). Range of proportion of variables that should be cross-loaded randomly onto one other factor. Accepts number of variables to cross-load onto one other factor as well.
- `magnitude_cross_loadings`: Numeric (length = 1, factors, or total number of variables to cross-load across all factors). The magnitude or size of the cross-loadings. Must range between -1 and 1.
- `magnitude_cross_loadings_range`: Numeric (length = 2). The range of the magnitude or size of the cross-loadings. Defaults to NULL.
- `leave_cross_loadings`: Boolean. Should cross-loadings be kept? Defaults to FALSE. Convergence problems can arise if cross-loadings are kept, so setting them to zero is the default. Only set to TRUE with careful consideration of the structure. Make sure to perform additional checks that the data are adequate.

Value

Returns a list containing the same parameters as the original `lf_object` but with updated data, `population_correlation`, and parameters (specifically, `loadings` matrix). Also returns original `lf_object` in `original_results`
Author(s)
Alexander P. Christensen <alexpaulchristensen@gmail.com>, Hudson Golino <hfg9s@virginia.edu>,
Luis Eduardo Garrido <luisgarrido@pucmm.edu>

References
psychometrics method to detect local dependence. PsyArXiv

Examples

```r
# Generate factor data
two_factor <- simulate_factors(
  factors = 2, # factors = 2
  variables = 6, # variables per factor = 6
  loadings = 0.55, # loadings between = 0.45 to 0.65
  cross_loadings = 0.05, # cross-loadings N(0, 0.05)
  correlations = 0.30, # correlation between factors = 0.30
  sample_size = 1000 # number of cases = 1000
)

# Add substantial cross-loadings
two_factor_CL <- add_cross_loadings(
  lf_object = two_factor,
  proportion_cross_loadings = 0.25,
  magnitude_cross_loadings = 0.35
)

# Randomly vary proportions
two_factor_CL <- add_cross_loadings(
  lf_object = two_factor,
  proportion_cross_loadings_range = c(0, 0.25),
  magnitude_cross_loadings = 0.35
)

# Randomly vary magnitudes
two_factor_CL <- add_cross_loadings(
  lf_object = two_factor,
  proportion_cross_loadings = 0.25,
  magnitude_cross_loadings_range = c(0.35, 0.45)
)

# Set number of cross-loadings per factor (rather than proportion)
two_factor_CL <- add_cross_loadings(
  lf_object = two_factor,
  proportion_cross_loadings = 2,
  magnitude_cross_loadings = 0.35
)
```
add_local_dependence

Description

Adds local dependence to simulated data from `simulate_factors`. See examples to get started.

Usage

```r
add_local_dependence(
  lf_object,
  method = c("correlate_residuals", "minor_factors", "threshold_shifts"),
  proportion_LD,
  proportion_LD_range = NULL,
  add_residuals = NULL,
  add_residuals_range = NULL,
  allow_multiple = FALSE
)
```

Arguments

- `lf_object` Data object from `simulate_factors`
- `method` Character (length = 1). Method to generate local dependence between variables. Only "correlate_residuals" at the moment. Future developments will include minor factor and threshold-shift methods. Description of methods:
  - "correlate_residuals" — Adds residuals directly to the population correlation matrix prior to data generation (uses population correlation matrix from `simulate_factors`)
  - "minor_factors" — Coming soon...
  - "threshold_shifts" — Coming soon...
- `proportion_LD` Numeric (length = 1 or factors). Proportion of variables that should be locally dependent across all or each factor. Accepts number of locally dependent values as well
- `proportion_LD_range` Numeric (length = 2). Range of proportion of variables that are randomly selected from a random uniform distribution. Accepts number of locally dependent values as well. Defaults to NULL
- `add_residuals` Numeric (length = 1, factors, or total number of locally dependent variables). Amount of residual to add to the population correlation matrix between two variables. Only used when `method = "correlate_residuals"`. Magnitudes are drawn from a random uniform distribution using +/- 0.05 of value input. Can also be specified directly (same length as total number of locally dependent variables). General effect sizes range from small (0.20), moderate (0.30), to large (0.40)
add_residuals_range

Numeric (length = 2). Range of the residuals to add to the correlation matrix are randomly selected from a random uniform distribution. Defaults to NULL

allow_multiple

Boolean. Whether a variable should be allowed to be locally dependent with more than one other variable. Defaults to FALSE. Set to TRUE for more complex locally dependence patterns

Value

Returns a list containing:

data

Simulated data from the specified factor model

population_correlation

Population correlation matrix with local dependence added

original_correlation

Original population correlation matrix before local dependence was added

correlated_residuals

A data frame with the first two columns specifying the variables that are locally dependent and the third column specifying the magnitude of the added residual for each locally dependent pair

original_results

Original lf_object input into function

Author(s)

Alexander P. Christensen <alexpaulchristensen@gmail.com>, Hudson Golino <hfg9s@virginia.edu>, Luis Eduardo Garrido <luisgarrido@pucmm.edu>

References


Examples

# Generate factor data
two_factor <- simulate_factors(
  factors = 2, # factors = 2
  variables = 6, # variables per factor = 6
  loadings = 0.55, # loadings between = 0.45 to 0.65
  cross_loadings = 0.05, # cross-loadings N(0, 0.05)
  correlations = 0.30, # correlation between factors = 0.30
  sample_size = 1000 # number of cases = 1000
)

# Add local dependence
two_factor_LD <- add_local_dependence(
  lf_object = two_factor,
  proportion_LD = 0.25,
  add_residuals = 0.20,
add_method_factors

   allows_multiple = FALSE
 )

# Randomly vary proportions
two_factor_LD <- add_local_dependence(
    lf_object = two_factor,
    proportion_LD_range = c(0.10, 0.50),
    add_residuals = 0.20,
    allows_multiple = FALSE
)

# Randomly vary residuals
two_factor_LD <- add_local_dependence(
    lf_object = two_factor,
    proportion_LD = 0.25,
    add_residuals_range = c(0.20, 0.40),
    allows_multiple = FALSE
)

# Randomly vary proportions, residuals, and allow multiple
two_factor_LD <- add_local_dependence(
    lf_object = two_factor,
    proportion_LD_range = c(0.10, 0.50),
    add_residuals_range = c(0.20, 0.40),
    allows_multiple = TRUE
)

add_method_factors

    Adds Methods Factors to simulate_factors Data

Description

    Adds methods factors to simulated data from simulate_factors. See examples to get started

Usage

    add_method_factors(
        lf_object,
        proportion_negative = 0.5,
        proportion_negative_range = NULL,
        methods_factors,
        methods_loadings,
        methods_loadings_range = 0,
        methods_correlations,
        methods_correlations_range = NULL
    )
Arguments

**lf_object**
Data object from `simulate_factors`. Data **must** be categorical. If data are not categorical, then there function will throw an error.

**proportion_negative**
Numeric (length = 1 or factors). Proportion of variables that should have negative (or flipped) dominant loadings across all or each factor. Accepts number of variables as well. The first variables on each factor, up to the corresponding proportion, will be flipped. Set to 0 to not have any loadings flipped. Defaults to 0.50.

**proportion_negative_range**
Numeric (length = 2). Range of proportion of variables that are randomly selected from a uniform distribution. Accepts number of number of variables as well. Defaults to NULL.

**methods_factors**
Numeric

**methods_loadings**
Numeric

**methods_loadings_range**
Numeric

**methods_correlations**
Numeric

**methods_correlations_range**
Numeric

Value

Returns a list containing:

**data**
Biased data simulated data from the specified factor model

**unbiased_data**
The corresponding unbiased data prior to replacing values to generate the (biased) data

**parameters**
Bias-adjusted parameters of the lf_object input into function

**original_results**
Original lf_object input into function

Author(s)

Alexander P. Christensen <alexpaulchristensen@gmail.com>, Luis Eduardo Garrido <luisgarrido@pucmm.edu>

References

**Examples**

```r
# Generate factor data
two_factor <- simulate_factors(
  factors = 2, # factors = 2
  variables = 6, # variables per factor = 6
  loadings = 0.55, # loadings between = 0.45 to 0.65
  cross_loadings = 0.05, # cross-loadings N(0, 0.05)
  correlations = 0.30, # correlation between factors = 0.30
  sample_size = 1000, # number of cases = 1000
  variable_categories = 5 # 5-point Likert scale
)

# Add methods factors
two_factor_methods_effect <- add_method_factors(
  lf_object = two_factor,
  proportion_negative = 0.50,
  methods_loadings = 0.20,
  methods_loadings_range = 0.10
)
```

---

**add_population_error**  
*Adds Population Error to simulate_factors Data*

**Description**

Adds population error to simulated data from `simulate_factors`. See examples to get started.

**Usage**

```r
add_population_error(
  lf_object,  
  cfa_method = c("minres", "ml"),  
  fit = c("cfi", "rmsea", "rmsr", "raw"),  
  misfit = c("close", "acceptable"),  
  error_method = c("cudeck", "yuan"),  
  tolerance = 0.01,  
  convergence_iterations = 10,  
  leave_cross_loadings = FALSE
)
```

**Arguments**

- `lf_object`: Data object from `simulate_factors`
- `cfa_method`: Character (length = 1). Method to generate population error. Defaults to "minres". Available options:
  - "minres" — Minimum residual
- "ml" — Maximum likelihood

**fit** Character (length = 1). Fit index to control population error. Defaults to "rmsr". Available options:
- "cfi" — Comparative fit index
- "rmsea" — Root mean square error of approximation
- "rmsr" — Root mean square residuals
- "raw" — Direct application of error

**misfit** Character or numeric (length = 1). Magnitude of error to add. Defaults to "close". Available options:
- "close" — Slight deviations from original population correlation matrix
- "acceptable" — Moderate deviations from original population correlation matrix

While numbers can be used, they are **not** recommended. They can be used to specify misfit but the level of misfit will vary depending on the factor structure

**error_method** Character (length = 1). Method to control population error. Defaults to "cudeck". Description of methods:
- "cudeck" — Description coming soon... see Cudeck & Browne, 1992 for more details
- "yuan" — Description coming soon...

**tolerance** Numeric (length = 1). Tolerance of SRMR difference between population error correlation matrix and the original population correlation matrix. Ensures that appropriate population error was added. Similarly, verifies that the MAE of the loadings are not greater than the specified amount, ensuring proper convergence. Defaults to 0.01

**convergence_iterations** Numeric (length = 1). Number of iterations to reach parameter convergence within the specified ‘tolerance’. Defaults to 10

**leave_cross_loadings** Boolean. Should cross-loadings be kept? Defaults to FALSE. Convergence problems can arise if cross-loadings are kept, so setting them to zero is the default. Only set to TRUE with careful consideration of the structure. Make sure to perform additional checks that the data are adequate

**Value**

Returns a list containing:

- **data** Simulated data from the specified factor model
- **population_correlation** Population correlation matrix with local dependence added
- **population_error** A list containing the parameters used to generate population error:
  - **error_correlation** — Correlation matrix with population error added (same as population_correlation)
• fit — Fit measure used to control population error
• delta — Minimum of the objective function corresponding to the misfit value
• misfit — Specified misfit value
• loadings — Estimated CFA loadings after error has been added

original_results
Original lf_object input into function

Author(s)

bifactor authors
Marcos Jimenez, Francisco J. Abad, Eduardo Garcia-Garzon, Vithor R. Franco, Luis Eduardo Garrido <luisgarrido@pucmm.edu>

latentFactor authors
Alexander P. Christensen <alexpaulchristensen@gmail.com>, Hudson Golino <hfg9s@virginia.edu>, Luis Eduardo Garrido <luisgarrido@pucmm.edu>, Marcos Jimenez, Francisco J. Abad, Eduardo Garcia-Garzon, Vithor R. Franco

References


Examples

# Generate factor data
two_factor <- simulate_factors(
  factors = 2, # factors = 2
  variables = 6, # variables per factor = 6
  loadings = 0.55, # loadings between = 0.45 to 0.65
  cross_loadings = 0.05, # cross-loadings N(0, 0.05)
  correlations = 0.30, # correlation between factors = 0.30
  sample_size = 1000 # number of cases = 1000
)

# Add small population error using Cudeck method
two_factor_Cudeck <- add_population_error(
  lf_object = two_factor,
  cfa_method = "minres",
  fit = "rmsr", misfit = "close",
  error_method = "cudeck"
)

# Add small population error using Yuan method
two_factor_Yuan <- add_population_error(
  lf_object = two_factor,
  cfa_method = "minres",
  fit = "rmsr", misfit = "close",
  error_method = "yuan"
)
add_wording_effects  

Adds Wording Effects to simulate_factors Data

Description

Adds wording effects to simulated data from simulate_factors. See examples to get started

Usage

add_wording_effects(
  lf_object, 
  method = c("acquiescence", "difficulty", "random_careless", "straight_line", "mixed"), 
  proportion_negative = 0.5, 
  proportion_negative_range = NULL, 
  proportion_biased_cases = 0.1, 
  proportion_biased_variables = 1, 
  proportion_biased_variables_range = NULL, 
  proportion_biased_person = 1, 
  proportion_biased_person_range = NULL
)

Arguments

lf_object Data object from simulate_factors. Data must be categorical. If data are not categorical, then there function with throw an error

method Character (length = 1). Method to generate wording effect to add to the data. Description of methods:

- "acquiescence" — Generates new data with flipped dominant loadings (based on proportion_negative) and ensures a bias such that variables have a restricted range of responding (e.g., only 4s and 5s on a 5-point Likert scale)
- "difficulty" — Generates new data with flipped dominant loadings (based on proportion_negative) and uses this data as the data without wording effects. Then, the signs of the dominant loadings are obtained and the dominant loadings are made to be absolute. Finally, the skews are multiplied by the signs of the original dominant loadings when generating the data with the wording effects
- "random_careless" — Number of cases up to proportion_biased_cases are sampled and replaced by values from a random uniform distribution ranging between the lowest and highest response category for each variable. These values then replace the values in the original data
- "straight_line" — Coming soon...

proportion_negative Numeric (length = 1 or factors). Proportion of variables that should have negative (or flipped) dominant loadings across all or each factor. Accepts number of variables as well. The first variables on each factor, up to the corresponding
add_wording_effects

proportion, will be flipped. Set to 0 to not have any loadings flipped. Defaults to 0.5.

proportion_negative_range
Numeric (length = 2). Range of proportion of variables that are randomly sele-
ected from a uniform distribution. Accepts number of number of variables as well. Defaults to NULL.

proportion_biased_cases
Numeric (length = 1). Proportion of cases that should be biased with wording ef-
effects. Also accepts number of cases to be biased. The first n number of cases, up
to the corresponding proportion, will be biased. Defaults to 0.10 or 10 percent
of cases.

proportion_biased_variables
Numeric (length = 1 or factors). Proportion of variables that should be biased
with wording effects. For method = "difficulty", proportion of biased vari-
ables will only count for the negative variables. For method = "acquiescence",
proportion of biased variables will only count for variables below the mid-point
of the variable_categories. Defaults to 1 or all possible variables.

proportion_biased_variables_range
Numeric (length = 2). Range of proportion of variables that should be biased
with wording effects. Values are drawn randomly from a uniform distribution.
Defaults to NULL.

proportion_biased_person
Numeric (length = 1 or proportion_biased_cases x sample_size). Person-
specific parameter of how many much bias the proportion_biased_cases
will have over the possible biased variables. This parameter interacts with
proportion_biased_variables. Parameter specifies the proportion of vari-
ables that should have bias per person. If one value is provided, then all biased
cases will have the same proportion of variables biased. Individual values are
possible by providing values for each biased case (round(nrow(lf_object$data)
* proportion_biased_cases)). Setting individual values for each biased case
is not recommended (use proportion_biased_person_range instead). De-
defaults to 1 or all possible biased variables for all biased cases.

proportion_biased_person_range
Numeric (length = 2). Range to randomly draw bias from a uniform distribution.
Allows for random person-specific bias to be obtained. Defaults to NULL.

Value
Returns a list containing:

- data: Biased data simulated data from the specified factor model
- unbiased_data: The corresponding unbiased data prior to replacing values to generate the (bi-
ased) data
- biased_sample_size: The number of cases that have biased data
- adjusted_results: Bias-adjusted lf_object input into function
- original_results: Original lf_object input into function
Author(s)
Alexander P. Christensen <alexpaulchristensen@gmail.com>, Luis Eduardo Garrido <luisgarrido@pucmm.edu>

References


Examples

```r
# Generate factor data
two_factor <- simulate_factors(
  factors = 2, # factors = 2
  variables = 6, # variables per factor = 6
  loadings = 0.55, # loadings between = 0.45 to 0.65
  cross_loadings = 0.05, # cross-loadings N(0, 0.05)
  correlations = 0.30, # correlation between factors = 0.30
  sample_size = 1000, # number of cases = 1000
  variable_categories = 5 # 5-point Likert scale
)

# Add wording effects using acquiescence method
two_factor_acquiescence <- add_wording_effects(
  lf_object = two_factor,
  proportion_negative = 0.50,
  proportion_biased_cases = 0.10,
  method = "acquiescence"
)

# Add wording effects using difficulty method
two_factor_difficulty <- add_wording_effects(
  lf_object = two_factor,
  proportion_negative = 0.50,
  proportion_biased_cases = 0.10,
  method = "difficulty"
)

# Add wording effects using random careless method
two_factor_random_careless <- add_wording_effects(
  lf_object = two_factor,
  proportion_negative = 0.50,
  proportion_biased_cases = 0.10,
  method = "random_careless"
)

# Add wording effects using straight line method
two_factor_random_careless <- add_wording_effects(
```

Categorize Continuous Data

Description

Categorizes continuous data based on Garrido, Abad and Ponsoda (2011; see references). Categorical data with 2 to 6 categories can include skew between -2 to 2 in increments of 0.05.

Usage

categorize(data, categories, skew_value = 0)

Arguments

data Numeric (length = n). A vector of continuous data with n values. For matrices, use apply

categories Numeric (length = 1). Number of categories to create. Between 2 and 6 categories can be used with skew

skew_value Numeric (length = 1). Value of skew. Ranges between -2 to 2 in increments of 0.05. Skews not in this sequence will be converted to the nearest value in this sequence. Defaults to 0 or no skew

Value

Returns a numeric vector of the categorize data
Author(s)

Maria Dolores Nieto Canaveras <mnietoca@nebrija.es>, Luis Eduardo Garrido <luisgarrido@pucmm.edu>, Hudson Golino <hfg9s@virginia.edu>, Alexander P. Christensen <alexpaulchristensen@gmail.com>

References


Examples

```r
# Dichotomous data (no skew)
dichotomous <- categorize(
  data = rnorm(1000),
  categories = 2
)

# Dichotomous data (with positive skew)
dichotomous_skew <- categorize(
  data = rnorm(1000),
  categories = 2,
  skew_value = 1.25
)

# 5-point Likert scale (no skew)
five_likert <- categorize(
  data = rnorm(1000),
  categories = 5
)

# 5-point Likert scale (negative skew)
five_likert <- categorize(
  data = rnorm(1000),
  categories = 5,
  skew_value = -0.45
)
```

---

**data_to_zipfs** Transforms *simulate_factors Data to Zipf’s Distribution*
Description

Zipf’s distribution is commonly found for text data. Closely related to the Pareto and power-law distributions, the Zipf’s distribution produces highly skewed data. This transformation is intended to mirror the data generating process of Zipf’s law seen in semantic network and topic modeling data.

Usage

data_to_zipfs(lf_object, beta = 2.7, alpha = 1, dichotomous = FALSE)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lf_object</td>
<td>Data object from <code>simulate_factors</code></td>
</tr>
<tr>
<td>beta</td>
<td>Numeric (length = 1). Sets the shift in rank. Defaults to 2.7</td>
</tr>
<tr>
<td>alpha</td>
<td>Numeric (length = 1). Sets the power of the rank. Defaults to 1</td>
</tr>
<tr>
<td>dichotomous</td>
<td>Boolean (length = 1). Whether data should be dichotomized rather than frequencies (e.g., semantic network analysis). Defaults to FALSE</td>
</tr>
</tbody>
</table>

Details

The formula used to transform data is (Piantadosi, 2014):

\[ f(r) \propto \frac{1}{(r + \beta)^\alpha} \]

where \(f(r)\) is the \(r\)th most frequency, \(r\) is the rank-order of the data, \(\beta\) is a shift in the rank (following Mandelbrot, 1953, 1962), and \(\alpha\) is the power of the rank with greater values suggesting greater differences between the largest frequency to the next, and so forth.

The function will transform continuous data output from `simulate_factors`. See examples to get started.

Value

Returns a list containing:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Simulated data that has been transform to follow Zipf’s distribution</td>
</tr>
<tr>
<td>RMSE</td>
<td>A vector of root mean square errors for transformed data and data assumed to follow theoretical Zipf’s distribution and Spearman’s correlation matrix of the transformed data compared to the original population correlation matrix</td>
</tr>
<tr>
<td>spearman_correlation</td>
<td>Spearman’s correlation matrix of the transformed data</td>
</tr>
<tr>
<td>original_correlation</td>
<td>Original population correlation matrix (before) the data were transformed</td>
</tr>
<tr>
<td>original_results</td>
<td>Original (lf_object) input into function</td>
</tr>
</tbody>
</table>

Author(s)

Alexander P. Christensen <alexpaulchristensen@gmail.com>, Hudson Golino <hfg9s@virginia.edu>, Luis Eduardo Garrido <luisgarrido@pucmm.edu>
References


Examples

```r
# Generate factor data
two_factor <- simulate_factors(
  factors = 2, # factors = 2
  variables = 6, # variables per factor = 6
  loadings = 0.55, # loadings between = 0.45 to 0.65
  cross_loadings = 0.05, # cross-loadings N(0, 0.05)
  correlations = 0.30, # correlation between factors = 0.30
  sample_size = 1000 # number of cases = 1000
)

# Transform data to Mandelbrot's Zipf's
two_factor_zipfs <- data_to_zipfs(
  lf_object = two_factor,
  beta = 2.7,
  alpha = 1
)

# Transform data to Mandelbrot's Zipf's (dichotomous)
two_factor_zipfs_binary <- data_to_zipfs(
  lf_object = two_factor,
  beta = 2.7,
  alpha = 1,
  dichotomous = TRUE
)
```

---

**EKC**

**Estimate Number of Dimensions using Empirical Kaiser Criterion**

**Description**

Estimates the number of dimensions in data using Empirical Kaiser Criterion (Braeken & Van Assen, 2017). See examples to get started.
**Usage**

EKC(data, sample_size)

**Arguments**

- **data**: Matrix or data frame. Either a dataset with all numeric values (rows = cases, columns = variables) or a symmetric correlation matrix.
- **sample_size**: Numeric (length = 1). If input into data is a correlation matrix, then specifying the sample size is required.

**Value**

Returns a list containing:

- **dimensions**: Number of dimensions identified.
- **eigenvalues**: Eigenvalues.
- **reference**: Reference values compared against eigenvalues.

**Author(s)**

Alexander P. Christensen <alexpaulchristensen@gmail.com>, Hudson Golino <hfg9s@virginia.edu>, Luis Eduardo Garrido <luisgarrido@pucmm.edu>

**References**


**Examples**

```r
# Generate factor data
two_factor <- simulate_factors(
  factors = 2, # factors = 2
  variables = 6, # variables per factor = 6
  loadings = 0.55, # loadings between = 0.45 to 0.65
  cross_loadings = 0.05, # cross-loadings N(0, 0.05)
  correlations = 0.30, # correlation between factors = 0.30
  sample_size = 1000 # number of cases = 1000
)

# Perform Empirical Kaiser Criterion
EKC(two_factor$data)
```
ESEM

Estimates Exploratory Structural Equation Model

Description

A general function to estimate an Exploratory Structural Equation Model (ESEM) using the lavaan package. With latentFactoR objects, the function requires fewer inputs.

Usage

ESEM(
  data,
  factors,
  variables,
  estimator = c("MLR", "WLSMV"),
  fit_measures = NULL,
  variable_polarity = NULL,
  wording_factor = c("none", "CTCM1", "CTCM1_each", "RI", "RI_each"),
  CTCM1_polarity = c("negative", "positive"),
  ...
)

Arguments

data        Numeric matrix, data frame, or latentFactoR object
factors     Numeric (length = 1). Number of ESEM factors to estimate
variables   Numeric (length = 1 or factors). Number of variables per factor. A vector
            the length of the number of factors can be specified to allow varying number of
            variables on each factor (necessary for some wording_factor arguments)
estimator   Character. Estimator to be used in cfa. Default options are "MLR" for continuous
            data and "WLSMV" for categorical data
fit_measures Character. Fit measures to be computed using fitMeasures. Defaults to: "chisq",
            "df", "pvalue", "cfi", "tli", "rmsea", "rmsea.ci.lower", "rmsea.ci.upper",
            "rmsea.pvalue", and "srmr". Other measures can be added but these measures
            will always be produced.
            If scaled values are available (not NA), then scaled fit measures will be used.
            variable_polarity     Numeric/character (length = 1 or total variables). Whether all (length = 1) or
                                    each variable (length = total variables) are positive (1, "p", "pos", "positive")
                                    or negative (-1, "n", "neg", "negative") polarity on the factor
wording_factor Character (length = 1). Whether wording factor(s) should be estimated. Defaults
to "none". Options include:
  • "CTCM1" — Description coming soon...
  • "CTCM1_each" — Description coming soon...
• "RI" — Description coming soon...
• "RI_each" — Description coming soon...

CTCM1_polarity Character. Polarity of the CTCM1 wording factor(s). Defaults to "negative" for negative polarity variables

... Additional arguments to be passed on to cfa

Value

Returns a list containing:

model Estimated ESEM model
fit Fit measures of estimated ESEM model

Author(s)

Alexander P. Christensen <alexpaulchristensen@gmail.com>, Luis Eduardo Garrido <luisgarrido@pucmm.edu>

Examples

# Generate factor data
two_factor <- simulate_factors(
  factors = 2, # factors = 2
  variables = 6, # variables per factor = 6
  loadings = 0.55, # loadings between = 0.45 to 0.65
  cross_loadings = 0.05, # cross-loadings N(0, 0.05)
  correlations = 0.30, # correlation between factors = 0.30
  sample_size = 1000, # number of cases = 1000
  variable_categories = 5 # 5-point Likert scale
)

## Not run:
# Estimate ESEM model with no wording effects
esem_no_wording_effects <- ESEM(
  data = two_factor,
  estimator = "WLSMV"
)

# Add wording effects using acquiescence method
two_factor_acquiescence <- add_wording_effects(
  lf_object = two_factor,
  proportion_negative = 0.50,
  proportion_biased_cases = 0.10,
  method = "acquiescence"
)

# Estimate ESEM model with wording effects
esem_wording_effects <- ESEM(
  data = two_factor_acquiescence,
  estimator = "WLSMV",
  wording_factor = "RI"
)
estimate_dimensions

Estimates Dimensions using Several State-of-the-art Methods

Description

Estimates dimensions using Exploratory Graph Analysis (EGA), Empirical Kaiser Criterion (EKC), Factor Forest (factor_forest), Exploratory Factor Analysis with out-of-sample prediction (fspe), Next Eigenvalue Sufficiency Test (NEST), and parallel analysis (fa.parallel)

Usage

```
estimate_dimensions(
  data,              # Matrix or data frame. Either a dataset with all numeric values (rows = cases, columns = variables) or a symmetric correlation matrix
  sample_size,      # Numeric (length = 1). If input into data is a correlation matrix, then specifying the sample size is required
  EGA_args = list(corr = "auto", uni.method = "louvain", model = "glasso", consensus.method = "most_common", plot.EGA = FALSE),
  FF_args = list(maximum_factors = 8, PA_correlation = "cor"),
  FSPE_args = list(maxK = 8, rep = 1, method = "PE", pbar = FALSE),
  NEST_args = list(iterations = 1000, maximum_iterations = 500, alpha = 0.05, convergence = 0.00001),
  PA_args = list(fm = "minres", fa = "both", cor = "cor", n.iter = 20, sim = FALSE, plot = FALSE)
)
```

Arguments

- **data**: Matrix or data frame. Either a dataset with all numeric values (rows = cases, columns = variables) or a symmetric correlation matrix
- **sample_size**: Numeric (length = 1). If input into data is a correlation matrix, then specifying the sample size is required
- **EGA_args**: List. List of arguments to be passed along to EGA. Defaults are listed
- **FF_args**: List. List of arguments to be passed along to factor_forest. Defaults are listed
- **FSPE_args**: List. List of arguments to be passed along to fspe. Defaults are listed
- **NEST_args**: List. List of arguments to be passed along to NEST. Defaults are listed
- **PA_args**: List. List of arguments to be passed along to fa.parallel. Defaults are listed

Value

Returns a list containing:

- **dimensions**: Dimensions estimated from each method

A list of each methods output (see their respective functions for their outputs)
Examples

# Generate factor data
two_factor <- simulate_factors(
  factors = 2, # factors = 2
  variables = 6, # variables per factor = 6
  loadings = 0.55, # loadings between = 0.45 to 0.65
  cross_loadings = 0.05, # cross-loadings N(0, 0.05)
  correlations = 0.30, # correlation between factors = 0.30
  sample_size = 1000 # number of cases = 1000
)

## Not run:
# Estimate dimensions
estimate_dimensions(two_factor$data)
## End(Not run)

---

**factor_forest**

Estimate Number of Dimensions using Factor Forest

### Description

Estimates the number of dimensions in data using the pre-trained Random Forest model from Goretzko and Buhner (2020, 2022). See examples to get started.

### Usage

```
factor_forest(
  data, 
  sample_size, 
  maximum_factors = 8, 
  PA_correlation = c("cor", "poly", "tet")
)
```

### Arguments

- **data**
  - Matrix or data frame. Either a dataset with all numeric values (rows = cases, columns = variables) or a symmetric correlation matrix.

- **sample_size**
  - Numeric (length = 1). If input into `data` is a correlation matrix, then specifying the sample size is required.

- **maximum_factors**
  - Numeric (length = 1). Maximum number of factors to search over. Defaults to 8.
PA_correlation Character (length = 1). Type of correlation used in `fa.parallel`. Must be set:

- "cor" — Pearson’s correlation
- "poly" — Polychoric correlation
- "tet" — Tetrachoric correlation

Value

Returns a list containing:

- dimensions Number of dimensions identified
- probabilities Probability that the number of dimensions is most likely

Author(s)

# Authors of Factor Forest
David Goretzko and Markus Buhner

# Authors of `latentFactoR`
Alexander P. Christensen <alexpaulchristensen@gmail.com>, Hudson Golino <hfg9s@virginia.edu>,
Luis Eduardo Garrido <luisgarrido@pucmm.edu>

References


Examples

```r
# Generate factor data
two_factor <- simulate_factors(
  factors = 2, # factors = 2
  variables = 6, # variables per factor = 6
  loadings = 0.55, # loadings between = 0.45 to 0.65
  cross_loadings = 0.05, # cross-loadings N(0, 0.05)
  correlations = 0.30, # correlation between factors = 0.30
  sample_size = 1000 # number of cases = 1000
)

## Not run:
# Perform Factor Forest
factor_forest(two_factor$data)
## End(Not run)
```
Estimate Number of Dimensions using Next Eigenvalue Sufficiency Test

Description

Estimates the number of dimensions in data using NEST (Achim, 2017). See examples to get started

Usage

```r
NEST(
  data,
  sample_size,
  iterations = 1000,
  maximum_iterations = 500,
  alpha = 0.05,
  convergence = 0.00001
)
```

Arguments

data Matrix or data frame. Either a dataset with all numeric values (rows = cases, columns = variables) or a symmetric correlation matrix

sample_size Numeric (length = 1). If input into data is a correlation matrix, then specifying the sample size is required

iterations Numeric (length = 1). Number of iterations to estimate rank. Defaults to 1000

maximum_iterations Numeric (length = 1). Maximum number of iterations to obtain convergence of eigenvalues. Defaults to 500

alpha Numeric (length = 1). Significance level for determine sufficient eigenvalues. Defaults to 0.05

convergence Numeric (length = 1). Value necessary to be less than or equal to when establishing convergence of eigenvalues

Value

Returns a list containing:

dimensions Number of dimensions identified

loadings Loading matrix

converged Whether estimation converged. If FALSE, then results are reported from last convergence point. Interpret results with caution.

Author(s)

Alexander P. Christensen <alexpaulchristensen@gmail.com>, Hudson Golino <hfg9s@virginia.edu>, Luis Eduardo Garrido <luisgarrido@pucmm.edu>
References


Examples

```r
# Generate factor data
two_factor <- simulate_factors(
  factors = 2, # factors = 2
  variables = 6, # variables per factor = 6
  loadings = 0.55, # loadings between = 0.45 to 0.65
  cross_loadings = 0.05, # cross-loadings N(0, 0.05)
  correlations = 0.30, # correlation between factors = 0.30
  sample_size = 1000 # number of cases = 1000
)

## Not run:
# Perform NEST
NEST(two_factor$data)
## End(Not run)
```

obtain_zipfs_parameters

*Obtain Zipf’s Distribution Parameters from Data*

Description

Zipf’s distribution is commonly found for text data. Closely related to the Pareto and power-law distributions, the Zipf’s distribution produces highly skewed data. This function obtains the best fitting parameters to Zipf’s distribution

Usage

```r
obtain_zipfs_parameters(data)
```

Arguments

- `data` Numeric vector, matrix, or data frame. Numeric data to determine Zipf’s distribution parameters
Details

The best parameters are optimized by minimizing the absolute difference between the original frequencies and the frequencies obtained by the beta and alpha parameters in the following formula (Piantadosi, 2014):

\[ f(r) \propto \frac{1}{(r + \beta)^\alpha} \]

where \( f(r) \) is the \( r \)th most frequency, \( r \) is the rank-order of the data, \( \beta \) is a shift in the rank (following Mandelbrot, 1953, 1962), and \( \alpha \) is the power of the rank with greater values suggesting greater differences between the largest frequency to the next, and so forth.

Value

Returns a vector containing the estimated beta and alpha parameters. Also contains zipfs_sse which corresponds to the sum of square error between frequencies based on the parameter values estimated and the original data frequencies.

Author(s)

Alexander P. Christensen <alexpaulchristensen@gmail.com>, Hudson Golino <hfg9s@virginia.edu>, Luis Eduardo Garrido <luisgarrido@pucmm.edu>

References


Examples

```r
# Generate factor data
two_factor <- simulate_factors(
  factors = 2, # factors = 2
  variables = 6, # variables per factor = 6
  loadings = 0.55, # loadings between = 0.45 to 0.65
  cross_loadings = 0.05, # cross-loadings N(0, 0.05)
  correlations = 0.30, # correlation between factors = 0.30
  sample_size = 1000 # number of cases = 1000
)

# Transform data to Mandelbrot's Zipf's
two_factor_zipfs <- data_to_zipfs(
  lf_object = two_factor,
  beta = 2.7,
  alpha = 1
)

# Obtain Zipf's distribution parameters
```
**simulate_factors**

Simulates Latent Factor Data

**Description**

Simulates data from a latent factor model based on many manipulable parameters. Parameters do not have default values and must each be set. See examples to get started.

**Usage**

```r
simulate_factors(
  factors, 
  variables, 
  variables_range = NULL, 
  loadings, 
  loadings_range = NULL, 
  cross_loadings, 
  cross_loadings_range = NULL, 
  correlations, 
  correlations_range = NULL, 
  sample_size, 
  variable_categories = Inf, 
  categorical_limit = 7, 
  skew = 0, 
  skew_range = NULL
)
```

**Arguments**

- **factors** Numeric (length = 1). Number of factors.
- **variables** Numeric (length = 1 or factors). Number of variables per factor. Can be a single value or as many values as there are factors. Minimum three variables per factor.
- **variables_range** Numeric (length = 2). Range of variables to randomly select from a random uniform distribution. Minimum three variables per factor.
- **loadings** Numeric or matrix (length = 1, factors, total number of variables (factors x variables), or factors x total number of variables. Loadings drawn from a random uniform distribution using +/- 0.10 of value input. Can be a single value or as many values as there are factors (corresponding to the factors). Can also be a loading matrix. Columns must match factors and rows must match total variables (factors x variables) General effect sizes range from small (0.40), moderate (0.55), to large (0.70).
simulate_factors Numeric (length = 2). Range of loadings to randomly select from a random uniform distribution. General effect sizes range from small (0.40), moderate (0.55), to large (0.70)
cross_loadings Numeric or matrix(length = 1, factors, or factors x total number of variables). Cross-loadings drawn from a random normal distribution with a mean of 0 and standard deviation of value input. Can be a single value or as many values as there are factors (corresponding to the factors). Can also be a loading matrix. Columns must match factors and rows must match total variables (factors x variables)
cross_loadings_range Numeric (length = 2). Range of cross-loadings to randomly select from a random uniform distribution
correlations Numeric (length = 1 or factors x factors). Can be a single value that will be used for all correlations between factors. Can also be a square matrix (factors x factors). General effect sizes range from orthogonal (0.00), small (0.30), moderate (0.50), to large (0.70)
correlations_range Numeric (length = 2). Range of correlations to randomly select from a random uniform distribution. General effect sizes range from orthogonal (0.00), small (0.30), moderate (0.50), to large (0.70)
sample_size Numeric (length = 1). Number of cases to generate from a random multivariate normal distribution using rmvnorm
variable_categories Numeric (length = 1 or total variables (factors x variables)). Number of categories for each variable. Inf is used for continuous variables; otherwise, values reflect number of categories
categorical_limit Numeric (length = 1). Values greater than input value are considered continuous. Defaults to 7 meaning that 8 or more categories are considered continuous (i.e., variables are not categorized from continuous to categorical)
skew Numeric (length = 1 or categorical variables). Skew to be included in categorical variables. It is randomly sampled from provided values. Can be a single value or as many values as there are (total) variables. Current skew implementation is between -2 and 2 in increments of 0.05. Skews that are not in this sequence will be converted to their nearest value in the sequence. Not recommended to use with variables_range. Future versions will incorporate finer skews
skew_range Numeric (length = 2). Randomly selects skews within in the range. Somewhat redundant with skew but more flexible. Compatible with variables_range

Value
Returns a list containing:
data Simulated data from the specified factor model
population_correlation Population correlation matrix
parameters

A list containing the parameters used to generate the data:

• factors — Number of factors
• variables — Variables on each factor
• loadings — Loading matrix
• factor_correlations — Correlations between factors
• categories — Categories for each variable
• skew — Skew for each variable

Author(s)

Maria Dolores Nieto Canaveras <mnietoca@nebrija.es>, Alexander P. Christensen <alexpaulchristensen@gmail.com>, Hudson Golino <hfg9s@virginia.edu>, Luis Eduardo Garrido <luisgarrido@pucmm.edu>

References


Examples

```r
# Generate factor data
two_factor <- simulate_factors(
  factors = 2, # factors = 2
  variables = 6, # variables per factor = 6
  loadings = 0.55, # loadings between = 0.45 to 0.65
  cross_loadings = 0.05, # cross-loadings N(0, 0.05)
  correlations = 0.30, # correlation between factors = 0.30
  sample_size = 1000 # number of cases = 1000
)

# Randomly vary loadings
two_factor_loadings <- simulate_factors(
  factors = 2, # factors = 2
  variables = 6, # variables per factor = 6
  loadings_range = c(0.30, 0.80), # loadings between = 0.30 to 0.80
  cross_loadings = 0.05, # cross-loadings N(0, 0.05)
  correlations = 0.30, # correlation between factors = 0.30
  sample_size = 1000 # number of cases = 1000
)

# Generate dichotomous data
two_factor_dichotomous <- simulate_factors(
  factors = 2, # factors = 2
  variables = 6, # variables per factor = 6
```
loadings = 0.55, # loadings between = 0.45 to 0.65
cross_loadings = 0.05, # cross-loadings N(0, 0.05)
correlations = 0.30, # correlation between factors = 0.30
sample_size = 1000, # number of cases = 1000
variable_categories = 2 # dichotomous data
)

# Generate dichotomous data with skew
two_factor_dichotomous_skew <- simulate_factors(
  factors = 2, # factors = 2
  variables = 6, # variables per factor = 6
  loadings = 0.55, # loadings between = 0.45 to 0.65
  cross_loadings = 0.05, # cross-loadings N(0, 0.05)
correlations = 0.30, # correlation between factors = 0.30
sample_size = 1000, # number of cases = 1000
variable_categories = 2, # dichotomous data
skew = 1 # all variables with have a skew of 1
)

# Generate dichotomous data with variable skew
two_factor_dichotomous_skew <- simulate_factors(
  factors = 2, # factors = 2
  variables = 6, # variables per factor = 6
  loadings = 0.55, # loadings between = 0.45 to 0.65
  cross_loadings = 0.05, # cross-loadings N(0, 0.05)
correlations = 0.30, # correlation between factors = 0.30
sample_size = 1000, # number of cases = 1000
variable_categories = 2, # dichotomous data
skew_range = c(-2, 2) # skew = -2 to 2 (increments of 0.05)
)

---

skew_tables

**Skew Tables**

---

**Description**

Tables for skew based on the number of categories (2, 3, 4, 5, or 6) in the data

**Usage**

```r
data(skew_tables)
```

**Format**

A list (length = 5)

**Examples**

```r
data("skew_tables")
```
Index

* datasets
  skew_tables, 31

add_cross_loadings, 3
add_local_dependence, 5
add_method_factors, 7
add_population_error, 9
add_wording_effects, 12

bifactor, 11

categorize, 15
cfa, 20, 21

data_to_zipfs, 16

EGA, 22
EKC, 18, 22
ESEM, 20
estimate_dimensions, 22

fa.parallel, 22, 24
factor_forest, 22, 23
fitMeasures, 20
fspe, 22

latentFactoR, 11, 20
latentFactoR (latentFactoR-package), 2
latentFactoR-package, 2
lavaan, 20

NEST, 22, 25

obtain_zipfs_parameters, 26

rmvnorm, 29

simulate_factors, 3, 5, 7–9, 12, 16, 17, 28
skew_tables, 31