Package 'autoFC'

June 10, 2026

Type Package

Title Automatic Toolkit for Construction, Optimization, Scoring and Simulation of Forced-Choice Tests

Version 1.0.0.1001

Description Forced-choice (FC) response has gained increasing popularity and interest for its resistance to faking when well-designed (Cao & Drasgow, 2019 <doi:10.1037/apl0000414>). To established well-designed FC scales, typically each item within a block should measure different trait and have similar level of social desirability (Zhang et al., 2020 <doi:10.1177/1094428119836486>). Recent study also suggests the importance of high inter-item agreement of social desirability between items within a block (Pavlov et al., 2021 <doi:10.31234/osf.io/hmnrc>). In addition to this, FC developers may also need to maximize factor loading differences (Brown & Maydeu-Olivares, 2011 <doi:10.1177/0013164410375112>) or minimize item location differences (Cao & Drasgow, 2019 <doi:10.1037/apl0000414>) depending on scoring models. Decision of which items should be assigned to the same block, also called as item pairing, is thus critical to the quality of an FC test. Because such pairing process often requires researchers to meet multiple objectives, manual pairing becomes impractical or even not feasible once the number of latent traits and/or number of items per elevates. To address these problems, autoFC is developed as a automatic and efficient tool for facilitating the automatic construction of FC tests (Li et al., 2022 <doi:10.1177/01466216211051726>), essentially exempting users from the burden of manual item pairing.
Given characteristics of each item (and item responses), FC measures can be constructed either automatically based on user-defined pairing criteria and weights, or based on exact specifications of each block (i.e., blueprint; see Li et al., 2025 <doi:10.1177/10944281241229784>). Users can also generate simulated responses based on the Thurstonian Item Response Theory model (Brown & Maydeu-Olivares, 2011 <doi:10.1177/0013164410375112>) and predict trait scores of simulated/actual respondents based on an estimated model.

License GPL (>= 3)

Depends R (>= 3.5)

Imports lavaan, MASS, MplusAutomation, pbapply, rstan, stats

Suggests knitr, rmarkdown, cmdstanr

Additional_repositories https://stan-dev.r-universe.dev

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

VignetteBuilder knitr

NeedsCompilation no

Author Mengtong Li [cre, aut] (ORCID: <https://orcid.org/0000-0002-1766-4976>),
Tianjun Sun [aut] (ORCID: <https://orcid.org/0000-0002-3655-0042>),
Bo Zhang [aut] (ORCID: <https://orcid.org/0000-0002-6730-7336>)

Maintainer Mengtong Li <mt_li@fudan.edu.cn>

Repository CRAN

Date/Publication 2026-06-10 06:00:02 UTC

Contents

`build_blueprint_blocks` *Build Forced-Choice Blocks from a Custom Blueprint*

Description

Constructs an initial set of forced-choice blocks that strictly adhere to a user-specified blueprint (e.g., exact trait and keying combinations for every block). It utilizes a fast random-search heuristic to simultaneously optimize a specified matching criterion (e.g., minimizing within-block difficulty variance).

This function acts as an excellent "Smart Initializer" before passing the resulting blocks into `assemble_blocks()` for further global optimization.

Usage

```
build_blueprint_blocks(
  item_chars,
  blueprint,
  bp_block_col = "block",
  bp_trait_col = "trait",
  bp_sign_col = "sign",
  bp_crit_col = NULL,
  df_id_col = "item_num",
  df_trait_col = "trait",
  df_sign_col = "sign",
  df_crit_col = NULL,
  optim_func = NULL,
  adjust_factor = 1.1,
  max_comb_attempts = 100,
  max_adjust_attempts = 5
)
```

Arguments

`item_chars`	A data frame containing the item pool characteristics. Each row represents a single item.
`blueprint`	A data frame detailing the exact structural requirements for every block. Must contain columns mapping to the block ID, required trait, and required keying direction.

bp_block_col — Character string. The column name in blueprint identifying the block ID. Defaults to "block".

bp_trait_col — Character string. The column name in blueprint identifying the required trait. Defaults to "trait".

bp_sign_col — Character string. The column name in blueprint identifying the required keying sign. Defaults to "sign".

bp_crit_col — Character string. Optional. The column name in blueprint defining the initial numerical target for the matching criterion (e.g., maximum allowed difference). Defaults to NULL.

df_id_col — Character string. The column name in item_chars containing item number.

df_trait_col — Character string. The column name in item_chars containing the items' traits. Defaults to "trait".

df_sign_col — Character string. The column name in item_chars containing the items' keying signs. Defaults to "sign".

df_crit_col — Character string. Optional. The column name in item_chars containing the numerical values to be optimized (e.g., item difficulty). Defaults to NULL.

optim_func — Character string or function object. Optional. The function used to evaluate the items within a block (e.g., "var" or a custom function). The result is compared against the target in bp_crit_col. Defaults to NULL.

adjust_factor — Numeric value > 1. The multiplier used to relax the matching criterion target if a valid combination cannot be found. For example, 1.1 increases the allowed variance by 10 percents per failure. Defaults to 1.1.

max_comb_attempts — Integer. The maximum number of random combinations to generate and test against the matching criterion before relaxing the target. Defaults to 100.

max_adjust_attempts — Integer. The maximum number of times to relax the target criterion using adjust_factor before giving up on the block. Defaults to 5.

Value

A data frame containing the successfully matched items. The data frame will include all original columns from item_chars, plus tracking columns N_adjusts, final_criteria, and block_id. If the algorithm fails to construct a block due to depleted inventory or overly strict criteria, it returns the partially built scale up to the point of failure.

---

build_target_dist *Create Target Pair Distribution for Scale-Level Fit*

Description

Create Target Pair Distribution for Scale-Level Fit

Usage

```
build_target_dist(
  traits,
  total_pairs,
  equal_mixed_ratio = c(1, 1),
  allow_same_trait = FALSE
)
```

Arguments

traits	A character vector of unique trait names (e.g., c("O", "C", "E", "A", "N")).
total_pairs	Integer. The total number of pairs your test will generate.
equal_mixed_ratio	Numeric vector of length 2. The desired ratio of "equal" keyed pairs to "mixed" keyed pairs (e.g., c(1, 1) for 50/50 split).
allow_same_trait	Logical. Do we allow items measuring the same traits to be paired together?

Value

A flexible data frame containing target counts for the SA algorithm.

---

cal_block_energy *Fast Calculation of Item Block "Energy"*

Description

Calculates the total "energy" of one or multiple paired item blocks. This function has been heavily optimized for speed, as it is called thousands of times during the Simulated Annealing optimization loop.

Usage

```
cal_block_energy(block, char_list, weights, fun_list)
```

Arguments

block	An *n* by *k* integer matrix of item indices.
char_list	A list of vectors, where each vector contains the full item pool data for one specific characteristic.
weights	A numeric vector of weights.
fun_list	A list of pre-matched R function objects (e.g., `list(facfun, var)`).

Value

A numeric value indicating the total energy for the given item blocks.

`cal_block_energy_with_iia` *Fast Calculation of Item Block "Energy" with Inter-Item Agreement*

Description

Calculates local block energy including IIA metrics. Heavily optimized for Simulated Annealing.

Usage

```
cal_block_energy_with_iia(
  block,
  char_list,
  weights,
  fun_list,
  rater_chars,
  iia_weights = c(BPlin = 1, BPquad = 1, AClin = 1, ACquad = 1),
  verbose = FALSE
)
```

Arguments

`block`	An *n* by *k* integer matrix of item indices.
`char_list`	A list of vectors for item characteristics.
`weights`	A numeric vector of weights for item characteristics.
`fun_list`	A list of pre-matched R function objects.
`rater_chars`	A matrix of participant responses.
`iia_weights`	A numeric vector of length 4 for IIA metrics.
`verbose`	Logical. If TRUE, prints metrics. Defaults to FALSE.

Value

A numeric value indicating the total energy.

`convert_mole_to_pairwise` *Convert Most/Least (MOLE) Survey Data into Pairwise Binary Outcomes*

Description

Convert Most/Least (MOLE) Survey Data into Pairwise Binary Outcomes

Usage

```
convert_mole_to_pairwise(data, n_blocks, block_size)
```

Arguments

data	A data frame with exactly (2 * n_blocks) columns. Odd columns = "Most" choice, Even columns = "Least" choice. Values should be numeric item IDs (either global or local block IDs).
n_blocks	Integer. Number of blocks in the questionnaire.
block_size	Integer. Number of items per block.

Value

A data frame of pairwise binary outcomes (1s, 0s, and NAs).

---

convert_ranks_to_pairwise

Convert Ranked Blocks into Pairwise Binary Outcomes

Description

Converts a dataset of ranked forced-choice blocks into pairwise binary comparisons suitable for Thurstonian IRT modeling.

Usage

```
convert_ranks_to_pairwise(
  data,
  n_blocks,
  block_size,
  lower_rank_is_better = TRUE
)
```

Arguments

data	A data frame or matrix where rows are respondents and columns are items. The columns must be ordered by block (e.g., Block 1 Item 1, Block 1 Item 2, Block 1 Item 3, Block 2 Item 1, etc.). Each column represents the assigned rank of that corresponding item in the block.
n_blocks	Integer. Number of blocks in the questionnaire.
block_size	Integer. Number of items per block.
lower_rank_is_better	Logical. In your data, does a smaller number mean a higher preference? (e.g., 1 = Most Preferred, 2 = Second Most). Defaults to TRUE.

Value

A data frame of binary outcomes (1s, 0s, and NAs). The column names will match the "i1i2", "i1i3" format expected by the syntax generators.

`create_blueprint_template` *Generate a Blueprint Template for Forced-Choice Scales*

Description

Generates a structured data frame representing a forced-choice test blueprint. Optionally exports it directly to a CSV file so test developers can manually design their block structures (assigning specific traits and keying) in spreadsheet software.

Usage

```
create_blueprint_template(n_blocks, block_size, file_path = NULL)
```

Arguments

`n_blocks`	Integer. Number of total FC blocks.
`block_size`	Integer. Desired block size for the FC scale.
`file_path`	Character. Optional. If provided, the template will be saved to this path as a CSV file (e.g., `"my_blueprint.csv"`).

Value

A data frame containing the `block` and `item_num` columns, plus empty columns for `trait` and `sign` ready to be filled.

`empirical_reliability` *Empirical Reliability Estimates*

Description

Calculates empirical reliability for IRT-based trait scores using the formula provided by Brown & Maydeu-Olivares (2018).

Usage

```
empirical_reliability(dataset, score_names, se_names = NULL)
```

Arguments

`dataset`	A data frame containing the trait estimates and standard errors.
`score_names`	Character vector. The names of the columns specifying trait scores.
`se_names`	Optional character vector. The names of the columns specifying trait standard errors. If `NULL`, the function automatically searches for columns named `"[score_name]_SE"`.

Details

For trait scores estimated using item response theory (IRT) models, a single test-level reliability coefficient (like Cronbach's Alpha) is often inappropriate because the standard error of measurement varies across the latent continuum.

Empirical reliability provides a summary estimate of how reliable the trait scores are "as a whole" by comparing the variance of the estimated scores to the average error variance.

Value

A named numeric vector containing the empirical reliability estimates for each trait.

Author(s)

Mengtong Li

References

Brown, A., & Maydeu-Olivares, A. (2018). Ordinal factor analysis of graded-preference questionnaire data. *Structural Equation Modeling, 25*(4), 516-529. doi:10.1080/10705511.2017.1392247

Examples

```
# Create fake scores and standard errors
fake_scores <- data.frame(
  Trait1 = rnorm(100), Trait1_SE = runif(100, 0.1, 0.3),
  Trait2 = rnorm(100), Trait2_SE = runif(100, 0.2, 0.4)
)

# Auto-detects the "_SE" columns
empirical_reliability(fake_scores, score_names = c("Trait1", "Trait2"))
```

`extract_adjacent_pairs_stan` *Extract Only Adjacent Rank Pairs for Stan (Heister et al., 2025)*

Description

Extract Only Adjacent Rank Pairs for Stan (Heister et al., 2025)

Usage

```
extract_adjacent_pairs_stan(raw_ranks, n_blocks, block_size)
```

Arguments

`raw_ranks`	Dataframe of raw rankings (1 = Best).
`n_blocks`	Integer.
`block_size`	Integer.

Value

A long-format list ready to inject directly into the Stan data list.

`extract_tirt_stan_scores` *Extract Trait Scores and SEs from a Fitted Stan TIRT Model*

Description

Extract Trait Scores and SEs from a Fitted Stan TIRT Model

Usage

```
extract_tirt_stan_scores(fit, stan_data)
```

Arguments

`fit`	A fitted cmdstanr object.
`stan_data`	The data list passed to Stan (must contain the 'trait_names' attribute).

Value

A data frame with respondents as rows, and Traits and Trait SEs as columns.

facfun *Function for Checking If All Items in a Vector Are Unique*

Description

Returns *1* if each element in the vector is unique, and *0* otherwise.

Usage

```
facfun(vec)
```

Arguments

vec	Input vector.

Value

1 if each element in the vector is unique, and *0* otherwise.

Author(s)

Mengtong Li

Examples

```
facfun(c("Openness", "Neuroticism", "Agreeableness"))
facfun(c("Openness", "Openness", "Agreeableness"))
```

FC_blocks *Actual FC Blocks construted from HEXACO-60 Items from Four Respondent Groups*

Description

The study conducted by Li and colleagues (2025) included four experimental groups each completing a differently designed FC scale. This dataset shows how the four scales are assembled in each group

Usage

```
FC_blocks
```

Format

A data frame with 60 rows and 4 columns

FC1_Blocks

FC2_Blocks

FC3_Blocks

FC4_Blocks

Block

References

Ashton, M. C., & Lee, K. (2009). The HEXACO-60: A short measure of the major dimensions of personality. *Journal of Personality Assessment, 91*(4), 340-345. doi:10.1080/00223890902935878

Li, M., Zhang, B., Li, L., Sun, T., & Brown, A. (2025). Mixed-keying or desirability-matching in the construction of forced-choice measures? An empirical investigation and practical recommendations. *Organizational Research Methods, 28*(2), 296-329. doi:10.1177/10944281241229784

`FC_item_info` *Keying and factor information for HEXACO-60 items*

Description

Includes the keying and measured factor of the 60 HEXACO-60 items.

Usage

```
FC_item_info
```

Format

A data frame with 60 rows and 3 columns

Item ID Item number

keying Keying of each item

factor Measured factor of each item

References

Ashton, M. C., & Lee, K. (2009). The HEXACO-60: A short measure of the major dimensions of personality. *Journal of Personality Assessment, 91*(4), 340-345. doi:10.1080/00223890902935878

Li, M., Zhang, B., Li, L., Sun, T., & Brown, A. (2025). Mixed-keying or desirability-matching in the construction of forced-choice measures? An empirical investigation and practical recommendations. *Organizational Research Methods, 28*(2), 296-329. doi:10.1177/10944281241229784

`generate_tirt_lavaan_syntax` *Generate lavaan Syntax for Thurstonian Models*

Description

Generates the syntax required to fit either a Second-Order Thurstonian Factor Model or a First-Order Thurstonian IRT Model in lavaan.

Usage

```
generate_tirt_lavaan_syntax(
  n_blocks,
  block_size,
  key_matrix,
  trait_col,
  key_col,
  cor_matrix = NULL,
  model_type = c("TFM", "TIRT"),
  force_positive_variances = TRUE,
  more_constraints = ""
)
```

Arguments

`n_blocks`	Integer. Number of blocks in the questionnaire.
`block_size`	Integer. Number of items per block.
`key_matrix`	A data.frame with two columns, one indicating item trait, another indicating item sign.
`trait_col`	Character string. The name of the column in `key_matrix` indicating the trait measured by the item.
`key_col`	Character string. The name of the column in `key_matrix` indicating the keying direction of the item (e.g., positive/negative).
`cor_matrix`	Optional matrix. Starting values for trait correlations.
`model_type`	Character. Either "TFM" (Second-Order Thurstonian Factor Model) or "IRT" (First-Order).
`force_positive_variances`	Logical. If TRUE, prevents Heywood cases. In "Factor" models, utility variances > 0.0001. In "IRT" models, utility variances and derived correlated errors > 0.0001. Defaults to TRUE.
`more_constraints`	Additional lavaan syntax strings for constraining certain parameters if needed.

Value

A character string containing the lavaan model syntax.

`generate_tirt_mplus_syntax` *Generate Mplus Syntax for Thurstonian IRT (TIRT) or Thurstonian Factor Model (TFM)*

Description

Generate Mplus Syntax for Thurstonian IRT (TIRT) or Thurstonian Factor Model (TFM)

Usage

```
generate_tirt_mplus_syntax(
  title = "TIRT Model",
  data_file,
  n_blocks,
  block_size,
  key_matrix,
  trait_col,
  key_col,
  cor_matrix = NULL,
  model_type = c("TIRT", "TFM"),
  tfm_free_pair_residuals = TRUE,
  estimator = c("WLSMV", "ULSMV", "ULS"),
  missing_code = "*",
  force_positive_variances = TRUE,
  quick_run = FALSE,
  out_file = "tirt_model.inp"
)
```

Arguments

`title`	Character. Title of the Mplus model.
`data_file`	Character. Name of the input data file.
`n_blocks`	Integer. Number of blocks in the questionnaire.
`block_size`	Integer. Number of items per block.
`key_matrix`	A data.frame with two columns, one indicating item trait, another indicating item sign.
`trait_col`	Character string. The name of the column in `key_matrix` indicating the trait measured by the item.
`key_col`	Character string. The name of the column in `key_matrix` indicating the keying direction of the item (e.g., positive/negative).
`cor_matrix`	Optional matrix. Starting values for trait correlations.
`model_type`	Character. Choose "TIRT" for the First-Order parameterization or "TFM" for the Second-Order parameterization.

`tfm_free_pair_residuals`
: Logical. In the TFM model, should the residual variance of the observed pairs be freely estimated? Defaults to TRUE.

`estimator`
: Character. The Mplus estimator to use ("WLSMV", "ULSMV", "ULS").

`missing_code`
: Numeric or Character. The value representing missing data.

`force_positive_variances`
: Logical. Constrain residual variances to be strictly positive (> 0.001/0) to prevent Heywood cases. Defaults to TRUE.

`quick_run`
: Logical. Specify TRUE to speed up estimation by not producing Chi-Square and standard errors. Defaults to FALSE.

`out_file`
: Character. The filename to save the generated Mplus syntax.

`generate_tirt_stan_syntax`
Generate Stan Syntax and Data for TIRT Models

Description

Automatically reduces logical dependencies for full rank data (Heister et al. 2025), handles missing/MOLE data natively, and generates Stan syntax capable of within-chain parallelization (reduce_sum).

Usage

```
generate_tirt_stan_syntax(
  pairwise_data,
  n_blocks,
  block_size,
  n_traits,
  key_matrix,
  trait_col,
  key_col,
  apply_heister = TRUE
)
```

Arguments

`pairwise_data`
: Dataframe of binary outcomes.

`n_blocks`
: Integer. Number of blocks.

`block_size`
: Integer. Number of items per block.

`n_traits`
: Integer. Number of traits.

`key_matrix`
: A data.frame with two columns, one indicating item trait, another indicating item sign.

`trait_col`
: Character string. The name of the column in `key_matrix` indicating the trait measured by the item.

key_col	Character string. The name of the column in key_matrix indicating the keying direction of the item (e.g., positive/negative).
apply_heister	Logical. Apply logical dependency reduction? Defaults to TRUE.

Value

A list containing syntax (Stan code) and data (List for rstan).

---

get_CFA_estimates *Conduct Confirmatory Factor Analysis (CFA) and Obtain Parameter Estimates*

Description

Performs CFA (or accepts an existing fitted model) and extracts the parameter estimates required for generating simulated Thurstonian IRT data.

Usage

```
get_CFA_estimates(fit_model, response_data = NULL, item_names = NULL)
```

Arguments

fit_model	Either a character string containing lavaan syntax, OR a pre-fitted lavaan object.
response_data	Optional. Data frame containing Likert-type responses. Required only if fit_model is a syntax string.
item_names	Optional character vector. Names of the items. If NULL, the function automatically detects the observed variables from the model.

Value

A list containing loadings, intercepts, residuals, covariances, and the full model_fit object.

get_iia *Helper Function for Outputting IIA Characteristics of Each Block*

Description

This function prints IIA metrics for select items, given the individual responses for the items.

Usage

```
get_iia(block, data)
```

Arguments

block	An n by k integer matrix, where n is the number of item blocks and k is the number of items per block.
data	A p by m numeric matrix with scores of each of the p participants for the m items.

Value

An n by k matrix indicating the four IIA metrics for each item block.

Author(s)

Mengtong Li

Examples

```
item_responses <- matrix(sample(seq(1:5), 600*60, replace = TRUE), ncol = 60, byrow = TRUE)
 get_iia(matrix(seq(1:60), ncol = 3, byrow = TRUE), item_responses)
```

get_simulation_matrices *Generate Simulated Person and Item Parameter Matrices for the TIRT Model*

Description

Takes the factor analysis results extracted by `get_CFA_estimates()` and generates simulated person traits, item parameters, and latent utility values for a Thurstonian IRT model.

Usage

```
get_simulation_matrices(cfa_estimates, N, empirical = FALSE)
```

Arguments

cfa_estimates	A list object returned by get_CFA_estimates().
N	Integer. Number of simulated responses (participants) to generate.
empirical	Logical. As in MASS::mvrnorm(): Should mu and sigma specify the empirical, rather than population, mean and covariance? Defaults to FALSE.

Value

A list containing matrices for Lambda (Loadings), Mu (Intercepts), Epsilon (Residuals), Theta (Latent Traits), and Utility.

Examples

```
cfa_model <- paste0("H =~ ", paste0("SS", seq(6, 18, by = 6), collapse = " + "), "\n",
                    "E =~ ", paste0("SS", seq(5, 18, by = 6), collapse = " + "), "\n",
                    "X =~ ", paste0("SS", seq(4, 18, by = 6), collapse = " + "), "\n")
cfa_fit <- lavaan::cfa(cfa_model, data = HEXACO_example_data[1:500,], ordered = TRUE,
                       estimator = "WLSMV", verbose = TRUE, test = "none", se = "none")
cfa_estimates <- get_CFA_estimates(cfa_fit)
simu_estimates <- get_simulation_matrices(cfa_estimates, N = 500)

### Because utilities orders are exactly the order each item appears in the cfa model,
### you may need to re-order the columns back in your own case.

num_order <- order(as.numeric(gsub("[^0-9]", "", colnames(simu_estimates$Utility))))
new_Utility <- simu_estimates$Utility[, num_order]
```

HEXACO_example_data *Example HEXACO Response Data*

Description

Actual Responses to the HEXACO-60 (Ashton & Lee, 2009) scale. These response data are taken from the study conducted by Li and colleagues (2025).

Usage

```
HEXACO_example_data
```

Format

A data frame with 2177 rows and 61 columns:

Group

SS1

SS2

SS3

SS4
SS5
SS6
SS7
SS8
SS9
SS10
SS11
SS12
SS13
SS14
SS15
SS16
SS17
SS18
SS19
SS20
SS21
SS22
SS23
SS24
SS25
SS26
SS27
SS28
SS29
SS30
SS31
SS32
SS33
SS34
SS35
SS36
SS37
SS38
SS39
SS40

SS41

SS42

SS43

SS44

SS45

SS46

SS47

SS48

SS49

SS50

SS51

SS52

SS53

SS54

SS55

SS56

SS57

SS58

SS59

SS60

Source

https://osf.io/yvpz3/?view_only=08601755f471440b80973194571b60bd

References

Ashton, M. C., & Lee, K. (2009). The HEXACO-60: A short measure of the major dimensions of personality. *Journal of Personality Assessment, 91*(4), 340-345. doi:10.1080/00223890902935878

Li, M., Zhang, B., Li, L., Sun, T., & Brown, A. (2025). Mixed-keying or desirability-matching in the construction of forced-choice measures? An empirical investigation and practical recommendations. *Organizational Research Methods, 28*(2), 296-329. doi:10.1177/10944281241229784

make_random_block *Construction of Random Item Blocks*

Description

Generates a matrix of randomly paired blocks, where each row represents a block of items. This serves as a fast, unstructured initial solution for forced-choice test assembly algorithms.

Usage

```
make_random_block(total_items, target_items = total_items, block_size)
```

Arguments

total_items	Integer. The total number of items available in the item pool.
target_items	Integer. The number of items to sample from total_items to build the blocks. Defaults to total_items. Must be $\leq$ total_items.
block_size	Integer. The number of items in each block (e.g., 2 for pairs, 3 for triplets). Must be ≥ 2.

Details

Given the total number of items in the pool, this function randomly samples target_items and formats them into a block matrix.

If target_items is not a multiple of block_size, the function will randomly draw additional items from the sampled pool to fill the incomplete final block, ensuring the matrix structure is strictly maintained.

Value

A matrix of integers indicating the item IDs. The number of rows equals ceiling(target_items / block_size), and the number of columns equals block_size.

Examples

```
# Use all 60 items to build 20 blocks of 3
make_random_block(total_items = 60, block_size = 3)

# Sample 45 items from a pool of 60 to build 15 blocks of 3
make_random_block(total_items = 60, target_items = 45, block_size = 3)

# Handle cases where target_items is not a multiple of block_size
# (Will randomly reuse 1 item to fill the last triplet)
make_random_block(total_items = 60, target_items = 50, block_size = 3)
```

MOLE_data *Actual Response to FC scales in MOLE format*

Description

In Li et al's (2025) study, respondents completed one of the four different FC scales. The current dataset contains their actual responses to the MOLE blocks, as well as their group information.

Usage

```
MOLE_data
```

Format

A data frame with 2173 rows (respondents, 4 groups) and 40 columns #'

Q1_0_GROUP_T1
Q1_1_GROUP_T1
Q2_0_GROUP_T1
Q2_1_GROUP_T1
Q3_0_GROUP_T1
Q3_1_GROUP_T1
Q4_0_GROUP_T1
Q4_1_GROUP_T1
Q5_0_GROUP_T1
Q5_1_GROUP_T1
Q6_0_GROUP_T1
Q6_1_GROUP_T1
Q7_0_GROUP_T1
Q7_1_GROUP_T1
Q8_0_GROUP_T1
Q8_1_GROUP_T1
Q9_0_GROUP_T1
Q9_1_GROUP_T1
Q10_0_GROUP_T1
Q10_1_GROUP_T1
Q11_0_GROUP_T1
Q11_1_GROUP_T1
Q12_0_GROUP_T1
Q12_1_GROUP_T1

Q13_0_GROUP_T1

Q13_1_GROUP_T1

Q14_0_GROUP_T1

Q14_1_GROUP_T1

Q15_0_GROUP_T1

Q15_1_GROUP_T1

Q16_0_GROUP_T1

Q16_1_GROUP_T1

Q17_0_GROUP_T1

Q17_1_GROUP_T1

Q18_0_GROUP_T1

Q18_1_GROUP_T1

Q19_0_GROUP_T1

Q19_1_GROUP_T1

Q20_0_GROUP_T1

Q20_1_GROUP_T1

Group

References

Li, M., Zhang, B., Li, L., Sun, T., & Brown, A. (2025). Mixed-keying or desirability-matching in the construction of forced-choice measures? An empirical investigation and practical recommendations. *Organizational Research Methods, 28*(2), 296-329. doi:10.1177/10944281241229784

`optimize_blocks` *Automatic Item Pairing Method in Forced-Choice Test Construction*

Description

Automatic construction of forced-choice tests based on the Simulated Annealing algorithm. Allows items to be:

1. Matched in either pairs, triplets, quadruplets or blocks of any size;
2. Matched based on any number of item-level characteristics (e.g. Social desirability, factor) based on any customized criteria;
3. Matched based on person-level inter-item agreement (IIA) metrics;
4. Optimally distributed at the global scale-level to ensure specific trait-pair combinations (e.g., evenly/unevenly distributing equal and/or mixed keyed pairs)

Usage

```
optimize_blocks(
  blocks = NULL,
  total_items = NULL,
  temp_initial = NULL,
  temp_eta = 0.01,
  temp_cooling = 0.999,
  temp_stop_ratio = 1e-06,
  item_chars,
  char_weights = NULL,
  optim_funcs = NULL,
  n_exchange = 2,
  prob_new_item = 0.25,
  use_iia = FALSE,
  response_matrix = NULL,
  iia_weights = c(BPlin = 1, BPquad = 1, AClin = 1, ACquad = 1),
  trait_col = NULL,
  key_col = NULL,
  target_dist = NULL,
  scale_fit_weight = 1,
  prevent_overlap = FALSE
)
```

Arguments

`blocks`	An n by k integer matrix, where n is the number of item blocks and k is the number of items per block. Serves as the initial starting blocks for the automatic pairing method.
`total_items`	Integer value. How many items do we sample from in order to build these `blocks`? Should be more than number of unique values in `block`.
`temp_initial`	Initial temperature value. Can be left as `NULL` and be computed based on the absolute value of initial energy of `blocks` (Recommended), and scaled by `temp_eta`. In general, higher temperature represents a higher probability of accepting an inferior solution.
`temp_eta`	A positive numeric value. The ratio of initial temperature to initial energy of `blocks`, if `temp_initial` is not designated. Default is 0.01.
`temp_cooling`	A positive numeric value less than 1. Determines the reduction rate of the temperature after each iteration. Default is 0.999.
`temp_stop_ratio`	A positive numeric value less than 1. Iteration stops when the temperature drops below `temp_stop_ratio` * `temp_initial`. Default is 10^-6.
`item_chars`	An m by r data frame, where m is the total number of items to sample from, whereas r is the number of item characteristics.
`char_weights`	A vector of length r with weights for each item characteristic in `item_chars`. Should provide a weight of 0 for specific characteristics not of interest, such as item IDs.

`optim_funcs` A vector of customized function names for optimizing each item characteristic within each block, with length *r*.

`n_exchange` Integer value. Determines how many blocks are exchanged in order to produce a new solution for each iteration. Should be a value larger than 1 and less than `nrow(blocks)`. Default is 2.

`prob_new_item` A value between *0* and *1*. Probability of choosing the strategy of picking a new item, when not all candidate items are used to build the FC scale. Default is 0.25.

`use_iia` Logical. Are IIA metrics used when performing automatic pairing? Default is FALSE.

`response_matrix`
A *p* by *m* numeric matrix with scores of each of the *p* participants for the *m* items. Ignored when `use_iia == FALSE`.

`iia_weights` A vector of length 4 indicating weights given to each IIA metric: Linearly weighted AC (Gwet, 2008; 2014); Quadratic weighted AC; Linearly weighted Brennan-Prediger (BP) Index (Brennan & Prediger, 1981; Gwet, 2014); Quadratic weighted BP.

`trait_col` Character string. The name of the column in `item_chars` indicating the trait measured by the item. Required if `target_dist` is used for global scale-level fit.

`key_col` Character string. The name of the column in `item_chars` indicating the keying direction of the item (e.g., positive/negative). Required if `target_dist` is used.

`target_dist` A data frame detailing the target distribution of trait-pair combinations for global scale-level fit. Must contain columns `trait1`, `trait2`, `match_type` (e.g., "equal" or "mixed"), and `target` (numeric count). See `build_target_dist()` for a helper function to generate this.

`scale_fit_weight`
Numeric value. The weight applied to the global scale-level fit penalty relative to the local block-level energy. Default is 1.0. Increase this value to prioritize global trait distribution over local block matching.

`prevent_overlap`
Logical. If target_dist is specified, should strict prevention of trait overlap (i.e., block containing items measuring the same trait) be enforced? Default is FALSE.

Value

A list containing:

- `block_initial`: Initial starting block
- `energy_initial`: Initial energy for `block_initial`
- `block_final`: Final paired block after optimization by SA
- `energy_final`: Final energy for `block_final`

Note

The essence of SA is the probablistic acceptance of solutions inferior to the current state, which avoids getting stuck in local maxima/minima. It is also recommended to try out different values of `char_weights, iia_weights, temp_eta` to find out the best combination of initial temperature and energy value in order to provide optimally paired blocks.

Author(s)

Mengtong Li

References

Brennan, R. L., & Prediger, D. J. (1981). Coefficient kappa: Some uses, misuses, and alternatives. *Educational and Psychological Measurement, 41*(3), 687-699. https://doi.org/10.1177/001316448104100307

Gwet, K. L. (2008). Computing inter rater reliability and its variance in the presence of high agreement. *British Journal of Mathematical and Statistical Psychology, 61*(1), 29-48. https://doi.org/10.1348/000711006X126600

Gwet, K. L. (2014). *Handbook of inter-rater reliability (4th ed.): The definitive guide to measuring the extent of agreement among raters*. Gaithersburg, MD: Advanced Analytics Press.

`plot_scores` *Scatter Plot for True vs Estimated Scores, True Score vs Absolute Error, etc.*

Description

This function provides a simple, enhanced diagnostic plot examining the performance of the forced-choice scale based on simulated or known true scores.

Usage

```
plot_scores(x_scores, y_scores, type = c("simple", "abs.diff"), ...)
```

Arguments

`x_scores`	Numeric vector. Scores to be plotted on the x-axis (typically True Scores).
`y_scores`	Numeric vector. Scores to be plotted on the y-axis (typically Estimated Scores).
`type`	Character. Which type of plot to display? Options are `"simple"` for an x-y scatter plot, or `"abs.diff"` for plotting the absolute difference `abs(y - x)` against x. Defaults to `"simple"`.
`...`	Additional graphical parameters passed to `plot()` (e.g., `main`, `pch`).

Details

This function extends base R `plot()` by automatically adding standard diagnostic reference lines (a 1:1 identity line for "simple", and a 0-error baseline for "abs.diff") to aid visual inspection of bias.

Value

A base R scatter plot.

Examples

```
true <- rnorm(100)
est <- true + rnorm(100, 0, 0.3)
plot_scores(true, est, type = "simple", main = "Recovery")
plot_scores(true, est, type = "abs.diff", main = "Error Magnitude")
```

---

predict_tirt_stan *Predict Trait Scores for New Data using a Fitted Stan Model*

Description

Extracts posterior means of item parameters from a fitted Stan TIRT model and uses analytical BFGS optimization to instantly score new respondents.

Usage

```
predict_tirt_stan(fit, stan_data, new_pairwise_data)
```

Arguments

fit	A fitted model object from either cmdstanr or rstan.
stan_data	The original data list passed to Stan (used to map the test design).
new_pairwise_data	A dataframe of pairwise binary responses for the NEW respondents.

Value

A data frame containing MAP trait scores and Standard Errors.

---

prepare_tirt_stan_data *Prepare pairwise list data for Stan*

Description

Prepare pairwise list data for Stan

Usage

```
prepare_tirt_stan_data(
  pairwise_data,
  n_blocks,
  block_size,
  key_matrix,
  trait_col,
  key_col,
  apply_heister = FALSE
)
```

Arguments

pairwise_data	A data frame of binary pairwise outcomes (output of convert_ranks_to_pairwise or convert_mole_to_pairwise).
n_blocks	Integer. Number of blocks in the questionnaire.
block_size	Integer. Number of items per block.
key_matrix	A data.frame with two columns, one indicating item trait, another indicating item sign.
trait_col	Character string. The name of the column in key_matrix indicating the trait measured by the item.
key_col	Character string. The name of the column in key_matrix indicating the keying direction of the item (e.g., positive/negative).
apply_heister	Experimental. Logical. Should the optimization techinques proposed by Heister et al., (2025) be applied? Default is FALSE.

RMSE_range *Calculate Overall or Binned RMSE of Trait Scores*

Description

A diagnostic function to examine measurement accuracy. It calculates the Root Mean Square Error (RMSE) either overall, or within specific intervals on the latent trait continuum.

Usage

```
RMSE_range(true_scores, estimated_scores, range_breaks = NULL)
```

Arguments

true_scores	Numeric vector. Actual/known trait scores.
estimated_scores	Numeric vector. Estimated trait scores from the model.
range_breaks	Numeric vector. Optional. Specifies the cut points for the trait score bins (e.g., c(-3, -1, 1, 3)). If NULL, calculates overall RMSE.

Value

If `range_breaks` is `NULL`, returns a single numeric RMSE value. If `range_breaks` is specified, returns a named numeric vector showing the RMSE within each score bin.

Examples

```
true <- rnorm(100)
est <- true + rnorm(100, 0, 0.3)

# Overall RMSE
RMSE_range(true, est)

# Binned RMSE
RMSE_range(true, est, range_breaks = c(-3, -1, 1, 3))
```

---

score_tirt_ipsative *Calculate Classical Ipsative (Sum) Scores for Forced-Choice Measures*

Description

Calculates traditional ipsative sum scores from pairwise binary outcomes. Natively handles both full rankings and partial rankings (MOLE format) via pro-rated scaling, and automatically reverses scores for negatively-keyed items.

Usage

```
score_tirt_ipsative(
  pairwise_data,
  n_blocks,
  block_size,
  key_matrix,
  trait_col,
  key_col
)
```

Arguments

`pairwise_data`	A data frame of binary pairwise outcomes (output of `convert_ranks_to_pairwise` or `convert_mole_to_pairwise`).
`n_blocks`	Integer. Number of blocks in the questionnaire.
`block_size`	Integer. Number of items per block.
`key_matrix`	A data.frame with two columns, one indicating item trait, another indicating item sign.
`trait_col`	Character string. The name of the column in `key_matrix` indicating the trait measured by the item.

key_col	Character string. The name of the column in key_matrix indicating the keying direction of the item (e.g., positive/negative).

Value

A data frame of classical ipsative scores for each unique trait, with rows representing respondents.

---

score_tirt_lavaan *Fast, Analytical Trait Scoring for Second-Order TIRT using optim()*

Description

Calculates MAP trait scores and Standard Errors for a fitted lavaan model. Automatically detects whether the model uses the First-Order (TIRT) or Second-Order (TFM) parameterization.

Usage

```
score_tirt_lavaan(fit, data, trait_names = NULL, floor_value = 1e-04)
```

Arguments

fit	A fitted lavaan object.
data	The pairwise binary dataset used to fit the model.
trait_names	Character vector of the traits to score (e.g., c("Trait1", "Trait2")). Can be left NULL and detected by the function.
floor_value	Numeric. If negative residual variances for observed variables are provided, replace those negative residual variances with this floor value.

Value

A data frame containing Trait scores and their Standard Errors (SE).

---

score_tirt_mplus *Fast Analytical Trait Scoring for Mplus TIRT/TFM Models*

Description

Extracts the estimated parameters from an Mplus .out file and uses a fast analytical gradient in R to instantly calculate MAP trait scores and Standard Errors. Works seamlessly for both First-Order (TIRT) and Second-Order (TFM) models.

Usage

```
score_tirt_mplus(inp_file, data, trait_names = NULL, run_model = TRUE)
```

Arguments

inp_file	Character. Path to the Mplus .inp (or .out) file.
data	The pairwise binary dataset used to fit the model.
trait_names	Optional character vector. The original names of the traits to use as column names in the final output. If NULL, uses the Mplus generated names.
run_model	Logical. If TRUE, runs Mplus before scoring. If FALSE, assumes the .out file already exists. Defaults to TRUE.

Value

A data frame containing Trait scores and their Standard Errors (SE).

---

score_tirt_stan *Run Stan TIRT Model and Extract Formatted Scores*

Description

Run Stan TIRT Model and Extract Formatted Scores

Usage

```
score_tirt_stan(
  stan_data,
  chains = 4,
  parallel_chains = 4,
  threads_per_chain = 2,
  iter_warmup = 1000,
  iter_sampling = 1000,
  init = 0
)
```

Arguments

stan_data	The data list generated by prepare_tirt_stan_data().
chains	Integer. Number of MCMC chains. Defaults to 4.
parallel_chains	Integer. How many chains to run simultaneously. Defaults to 4.
threads_per_chain	Integer. CPU threads allocated INSIDE each chain. Defaults to 2.
iter_warmup	Integer. Warmup iterations. Defaults to 1000.
iter_sampling	Integer. Sampling iterations. Defaults to 1000.
init	Same as the init parameter in rstan.

Value

A list containing scores (Data frame of traits) and fit (The cmdstanr fit object).

summarize_trait_pairs *Summarize Trait and Keying Pairs in a Constructed Forced-Choice Test*

Description

Analyzes a constructed forced-choice block design and tallies the number of equally-keyed and mixed-keyed pairs for every trait combination.

Usage

```
summarize_trait_pairs(
  blocks,
  item_chars,
  trait_col,
  key_col,
  block_size = NULL,
  output_format = c("wide", "long")
)
```

Arguments

blocks	Either an n-by-k matrix of item IDs (where rows are blocks), or a flat vector of ordered item IDs.
item_chars	Data frame containing the item characteristics.
trait_col	Character string. Name of the column in item_chars containing traits.
key_col	Character string. Name of the column in item_chars containing keying directions.
block_size	Integer. Required only if blocks is a flat vector. Number of items per block.
output_format	Character. Either "wide" (default, cross-tabulated summary) or "long" (matches the exact data frame format of build_target_dist()).

Value

A data frame of the tallied pairs.

Index