Package ‘autoFC’

March 13, 2025

Type Package

Title Automatic Construction of Forced-Choice Tests

Version 0.2.0.1002

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

Maintainer Mengtong Li <ml70@illinois.edu>

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, termed item pairing, is thus critical to the quality of an FC test. This pairing process is essentially an optimization process which is currently carried out manually. However, given that we often need to simultaneously meet multiple objectives, manual pairing becomes impractical or even not feasible once the number of latent traits and/or number of items per trait are relatively large. To address these problems, autoFC is developed as a practical 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 and reducing the computational costs and biases induced by simple ranking methods. 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., 2024 <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

URL https://github.com/tspsyched/autoFC

BugReports https://github.com/tspsyched/autoFC/issues

Imports dplyr, irrCAC, lavaan, MASS, SimDesign, thurstonianIRT, MplusAutomation, glue, tidyr

Suggests knitr, rmarkdown

VignetteBuilder knitr

Date/Publication 2025-03-13 16:00:01 UTC

Encoding UTF-8

NeedsCompilation no

Repository CRAN

RoxygenNote 7.2.3

Depends R (>= 2.10)

LazyData true

Contents

build_scale_with_blueprint 3
build_TIRT_var_names 6
cal_block_energy 7
cal_block_energy_with_iia 8
construct_blueprint 10
convert_to_TIRT_response 11
empirical_reliability 14
facfun 15
fit_TIRT_model 15
get_CFA_estimates 17
get_iia 19
get_simulation_matrices 20
get_TIRT_long_data 22
HEXACO_example_data 23
make_random_block 24
plot_scores 25
predict_scores 26
RMSE_range 27
sa_pairing_generalized 28
triplet_block_info 30
triplet_example_data 31

Index **33**

build_scale_with_blueprint

Construct Forced-Choice Blocks Aligned with the Specifications in a Blueprint

Description

This function takes in the information of all available items as well as a blueprint data frame specifying the design of blocks, and returns a data frame of item blocks consistent with the blueprint (if possible).

Usage

```
build_scale_with_blueprint(
  item_df,
  blueprint,
  bp_block_name,
  bp_item_nums_name,
  bp_trait_name,
  bp_sign_name,
  bp_matching_criterion_name,
  df_item_nums_name,
  df_trait_name,
  df_sign_name,
  df_matching_criterion_name,
  df_matching_function,
  df_matching_adjust_factor,
  max_attempts_in_comb = 100,
  max_attempts_in_adjust
)
```

Arguments

`item_df` Data frame containing information related to all the available items

`blueprint` Pre-specified blueprint for your blocks. Preferably constructed from `construct_blueprint()`

`bp_block_name`, `bp_item_nums_name`, `bp_trait_name`, `bp_sign_name`
: Column names in the blueprint that specifies block number, item number in the block, desired trait of the item, and desired keying of the item, respectively

`bp_matching_criterion_name`
: Column name in the blueprint that indicates the additional matching criterion (cutoff value) you wish to test

`df_item_nums_name`, `df_trait_name`, `df_sign_name`
: Column names in `item_df` that specifies item_number, trait of the item, and keying of the item, respectively

df_matching_criterion_name
: Optional. Column name in item_df that is used to evaluate the matching criterion specified in bp_matching_criterion_name

df_matching_function
: Optional. A character string containing function name for evaluating the matching criterion

df_matching_adjust_factor
: Optional. A numeric value. If after max_attempts_in_comb attempts the additional criteria in df_matching_criterion_name cannot be met (> the cutoff value specified in blueprint[, bp_matching_criterion_name]), multiply that cutoff value by this adjusting factor.

max_attempts_in_comb
: Optional. An integer value. How many attempts will be made for finding a block that satisfies the blueprint, before we adjust the cutoff value?

max_attempts_in_adjust
: Optional. An integer value. How many attempts will be made for adjusting cutoff value? Will throw a warning and return the currently partially constructed scale (and specify which block might have problems) if number of attempts exceeds this value.

Details

Although automatically finding the block combinations that can satisfy multiple certain criteria for matching can be helpful (as the primary functionality of the previous version of autoFC is about), users may also wish to have exact specifications for some blocks in many cases. For example, typically in FC construction, we may want to explicitly specify the trait and keying combinations for each block. This function allows you to explicitly do that. Users are free to extend this function if further exact specifications are needed.

For now, this function also allows users to specify one additional matching criterion for the blocks. Users can designate the function for calculating this criterion (df_matching_function) and specify a multiplicative adjusting factor (df_matching_adjust_factor), if the criterion fails to be met after a specified number of attempts (max_attempts_in_comb). One good example of matching criterion is matching in social desirability rating, where you want ratings of the items in the same block to be less than a certain cutoff.

If after a certain number of times (max_attempts_in_adjust) the given block is still unable to be constructed (i.e., criterion matching still fails even if we relax the cutoff multiple times), a warning message will be shown and a partially built scale will be returned. Warnings along with a partially built scale may also be returned when it is impossible for the remaining items in item_df to satisfy the specification in the blueprint (e.g. we have no items for trait1 left, but the blueprint requires a block with an item measuring trait1).

Value

A data frame containing the selected items for each specified block. If matching criteria is specified, the data frame will also contain the number of times we adjusted the cutoffs for each block, and the final matching criteria cutoff resulting from adjustments.

Author(s)

Mengtong Li

See Also

construct_blueprint()

Examples

```
#### For the case you do not need additional matching criterion
item_info <- triplet_block_info
test_bp <- construct_blueprint(N_blocks = 2, block_size = 3,
                           traits = c("honestyhumility", "emotionality", "extraversion",
                                      "agreeableness", "conscientiousness", "openness"),
                               signs = c(-1, 1, 1,
                                         -1, -1, -1))
### Some arguments can be omitted if you don't have extra matching criteria.
picked_scale <- build_scale_with_blueprint(item_df = item_info,
                                           blueprint = test_bp,
                                        ### These parameters are column names in test_bp
                                           bp_block_name = "block",
                                           bp_item_nums_name = "item_num",
                                           bp_trait_name = "traits",
                                           bp_sign_name = "signs",
                                      ### These parameters are column names in item_info
                                           df_item_nums_name = "ID",
                                           df_trait_name = "Factor",
                                           df_sign_name = "Keying")

#### Or you may want to match social desirability ratings, for example
test_bp2 <- test_bp
test_bp2$SD_matching <- rep(0.5, 6)

#### Suppose that the items also have their own ratings
item_info2 <- item_info
item_info2$SD_rating <- rnorm(15, 3.5, 1)
range_m <- function(x) {
  return(max(x) - min(x))
}

picked_scale2 <- build_scale_with_blueprint(item_df = item_info2,
                                            blueprint = test_bp2,
                                       ### These parameters are column names in test_bp2
                                            bp_block_name = "block",
                                            bp_item_nums_name = "item_num",
                                            bp_trait_name = "traits",
                                            bp_sign_name = "signs",
                                      ### These parameters are column names in item_info2
                                            df_item_nums_name = "ID",
                                            df_trait_name = "Factor",
                                            df_sign_name = "Keying",
                                            ### These parameters will be used
```

```
                                    ### when you have extra matching criteria
                                    df_matching_criterion_name = "SD_rating",
                                    bp_matching_criterion_name = "SD_matching",
                                ## Which function is used to calculate matching?
                                    df_matching_function = "range_m",
                                    df_matching_adjust_factor = 1.25,
                                    max_attempts_in_comb = 100,
                                    max_attempts_in_adjust = 20)
```

`build_TIRT_var_names` *Build Variable Names for the Pairwise/Rank Responses in the TIRT Model*

Description

This function builds the variable names that corresponds to the pairwise comparisons or ranks among items within each block.

Usage

```
build_TIRT_var_names(
  item_name = "i",
  block_size,
  N_blocks,
  format = "pairwise"
)
```

Arguments

`item_name`	The prefix you want to have for your response variables.
`block_size, N_blocks`	The block size and total number of the forced-choice scale.
`format`	What format should the converted responses be in? Can be `"pairwise"` or `"ranks"`.

Details

Choose the correct `item_name` so that they are consistent with the item names in the data frame storing information of the items.

Value

A vector of variable names

Author(s)

Mengtong Li

See Also

`get_TIRT_long_data()`

Examples

```
build_TIRT_var_names("i", block_size = 3, N_blocks = 20, format = "pairwise")
build_TIRT_var_names("i", block_size = 5, N_blocks = 12, format = "ranks")
```

---

cal_block_energy *Calculation of Item Block "Energy"*

Description

Calculates the total "energy" of one or multiple paired item blocks, which is a linear combination of different functions applied to different item characteristics of interest.

Usage

```
cal_block_energy(block, item_chars, weights, FUN)
```

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.
`item_chars`	An m by r data frame, where m is the total number of items to sample from, whether it is included in the block or not, whereas r is the number of item characteristics.
`weights`	A vector of length r with weights for each item characteristics in `item_chars`. Should provide a weight of 0 for specific characteristics not of interest, such as item ID.
`FUN`	A vector of customized function names for optimizing each item characteristic within each block, with length r.

Details

This energy calculation function serves as the core for determining the acceptance or rejection of a newly built block over the previous one.

Higher energy is considered more preferable in this case.

Items in the same block can be paired based on characteristics such as:

Mean score, Item Factor, Factor loading, Item IRT Parameters, Reverse Coding, etc.

Pairings of different characteristics can be optimized in different way, by determining the customized function vector `FUN` and the corresponding `weights`.

Value

A numeric value indicating the total energy for the given item block(s).

Note

Use `cal_block_energy_with_iia` if inter-item agreement (IIA) metrics are needed.

Author(s)

Mengtong Li

Examples

```
## Simulate 60 items loading on different Big Five dimensions,
## with different mean and item difficulty

item_dims <- sample(c("Openness","Conscientiousness","Neuroticism",
                     "Extraversion","Agreeableness"), 60, replace = TRUE)
item_mean <- rnorm(60, 5, 2)
item_difficulty <- runif(60, -1, 1)

## Construct data frame for item characteristics and produce
## 20 random triplet blocks with these 60 items

item_df <- data.frame(Dimensions = item_dims, Mean = item_mean,
                     Difficulty = item_difficulty)
solution <- make_random_block(60, 60, 3)

## See ?facfun for its use.
cal_block_energy(solution, item_chars = item_df,
              weights = c(1,1,1), FUN = c("facfun", "var", "var"))
```

`cal_block_energy_with_iia`
Calculation of Item Block "Energy" with IIAs Included

Description

Calculates the total "energy" of one or multiple paired item blocks, which is a linear combination of different functions applied to different item characteristics of interest.

This function extends `cal_block_energy` function with consideration of inter item agreement (IIA) metrics.

Usage

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

Arguments

`block`, `item_chars`, `weights`, `FUN`
: See `?cal_block_energy` for details.

`rater_chars`
: A p by m numeric matrix with scores of each of the p participants for the m items.

`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.

`verbose`
: Logical. Should IIAs be printed when this function is called?

Details

This energy calculation function serves as the core for determining the acceptance or rejection of a newly built block over the previous one. Higher energy is considered more preferable in this case.

Items in the same block can be paired based on characteristics such as: Mean score, Item Factor, Factor loading, Item IRT Parameters, Reverse Coding, etc.

In addition, IIAs can be adopted to further estimate rater agreements between different items, if such information is available for the researchers.

Pairings of different characteristics can be optimized in different way, by determining the customized function vector `FUN` and the corresponding `weights`. Currently only linear weighted combination for IIAs can be used in optimization.

Value

A numeric value indicating the total energy for the given item block(s).

Note

Use `cal_block_energy_with_iia` if inter-item agreement (IIA) metrics are needed.

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.

See Also

cal_block_energy

Examples

```
## Simulate 60 items loading on different Big Five dimensions,
## with different mean and item difficulty

item_dims <- sample(c("Openness","Conscientiousness","Neuroticism",
                      "Extraversion","Agreeableness"), 60, replace = TRUE)
item_mean <- rnorm(60, 5, 2)
item_difficulty <- runif(60, -1, 1)

## Construct data frame for item characteristics and produce
## 20 random triplet blocks with these 60 items

item_df <- data.frame(Dimensions = item_dims, Mean = item_mean,
                      Difficulty = item_difficulty)
solution <- make_random_block(60, 60, 3)


## Simple simulation of responses from 600 participants on the 60 items.
## In practice, should use real world data or simluation based on IRT parameters.

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


cal_block_energy_with_iia(solution, item_chars = item_df, weights = c(1,1,1),
                           FUN = c("facfun", "var", "var"),
                           rater_chars = item_responses, iia_weights = c(1,1,1,1))
```

---

construct_blueprint *Build a Blueprint Data Frame for the Focal FC Scale*

Description

This function takes in specifications of block size, number of blocks, as well as trait and keying of each item in these blocks, and returns a data frame incorporating these information and ready to be further used for constructing FC blocks by other functions like `build_scale_with_blueprint()`.

Usage

```
construct_blueprint(N_blocks, block_size, traits, signs)
```

Arguments

`N_blocks`	Number of total FC blocks
`block_size`	Desired block size for the FC scale
`traits, signs`	Optional vectors. If given, specifies which traits and signs each item in the FC scale should have. `traits` is a string vector, while `signs` is a numeric vector (1 for positive items and -1 for negative items)

Details

A "blueprint" of the forced-choice scale is essentially a data frame where each row represents one item in the forced-choice scale, and columns specify which block the item belongs to, the trait that the item measures, and the keying of that item.

Note that these are only the basic item information typically needed when matching items into FC blocks; Users can further add other columns to the blueprint if they want to match based on more criteria.

Value

A data frame, containing the block membership, trait and keying information of all the items.

Author(s)

Mengtong Li

Examples

```
example_blueprint <- construct_blueprint(N_blocks = 5, block_size = 3,
                                         traits = sample(c("Openness",
                                                           "Conscientiousness",
                                                           "Extraversion",
                                                           "Agreeableness",
                                                    "Neuroticism"), 15, replace = TRUE),
                                        signs = sample(c(-1, 1), 15, replace = TRUE))
```

`convert_to_TIRT_response` *Convert the Latent Utility Values into Thurstonian IRT Pairwise/Rank Responses with Pre-Specified Block Design*

Description

This function simulates the responses to forced-choice blocks (both MOLE and RANK format), with the raw responses converted into pairwise or rank data to be understood by the Thurstonian IRT model.

Usage

```
convert_to_TIRT_response(
  Utility,
  block_design,
  format = "pairwise",
  partial = FALSE,
  block_size,
  N_blocks,
  N_response
)
```

Arguments

- `Utility` The utility matrix of all items. Note that if this matrix is produced from `get_simulation_matrices()`, the item order will be consistent with the order they appear in the CFA model. Users may need to re-order the columns back into 1, 2, 3...order before using this matrix as the input.
- `block_design` A numeric matrix specifying which items will be in the same forced-choice block (row).
- `format` What format should the converted responses be in? Can be `"pairwise"` or `"ranks"`.
- `partial` Only used when `format == "ranks"`. Should partial ranking responses be produced?
- `block_size`, `N_blocks` The block size and total number of the forced-choice scale. Preferably left blank and obtained through `block_design`.
- `N_response` Number of simulated responses you wish to generate. Default to `nrow(Utility)`.

Details

According to the Thurstonian IRT model, when a respondent needs to make a choice between two items, they elicit a latent utility value for the two items and choose the item that has a higher utility value. Choosing/Ranking among >2 items follows a similar procedure where the respondent generate latent utility for each item and produces a ranking or preference.

For forced-choice blocks, the above choice procedure is conducted among the `block_size` items in the same block, and the respondent can either indicate the most/least preferred item (MOLE format) or rank all the items in terms of preference (RANK format).

Regardless of the format, the raw responses to the forced-choice blocks need to be converted into either all pairwise comparisons (`format = "pairwise"`), or a full ranking (`format = "ranks"`), among the the `block_size` items in the same block.

We note that the when `block_size` is larger than 3 and when the MOLE format is used, some pairwise comparisons among the items in the block will be missing by design. As for now, the current technique is not yet able to handle missing pairwise responses when `format = "pairwise"`. Thus, if users wish to simulate responses to MOLE format blocks with `block_size` larger than 3, we recommend using `format = "ranks"` and also set `partial = TRUE`.

Value

A data frame containing pairwise (if `format == "pairwise"`) or rank (if `format == "ranks"`) responses to each block for the `N_response` participants.

Note

Importantly, the `Utility` matrix produced by `get_simulation_matrices()` may not be directly used in this function because that utility matrix will have the item columns placed in the order they appear in the CFA model, not in the original Item 1, Item 2...order. Users need to re-order the columns of the `Utility` matrix produced by `get_simulation_matrices()` accordingly before feeding the utility matrix to this function.

Author(s)

Mengtong Li

Examples

```
library(lavaan)
rating_data <- HEXACO_example_data
cfa_model <- paste0("H =~ ", paste0("SS", seq(6,60,6), collapse = " + "), "\n",
                    "E =~ ", paste0("SS", seq(5,60,6), collapse = " + "), "\n",
                    "X =~ ", paste0("SS", seq(4,60,6), collapse = " + "), "\n",
                    "A =~ ", paste0("SS", seq(3,60,6), collapse = " + "), "\n",
                    "C =~ ", paste0("SS", seq(2,60,6), collapse = " + "), "\n",
                    "O =~ ", paste0("SS", seq(1,60,6), collapse = " + "), "\n")
cfa_estimates <- get_CFA_estimates(response_data = rating_data,
                                   fit_model = cfa_model,
                                   item_names = paste0("SS",c(1:60)))
cfa_matrices <- get_simulation_matrices(loadings = cfa_estimates$loadings,
                                        intercepts = cfa_estimates$intercepts,
                                        residuals = cfa_estimates$residuals,
                                        covariances = cfa_estimates$covariances,
                                        N = 100, N_items = 60, N_dims = 6,
                                        dim_names = c("H", "E", "X", "A", "C", "O"),
                                        empirical = TRUE)

### Re-order the Utility columns!
cfa_matrices$Utility <- cfa_matrices$Utility[,c(t(matrix(1:60, ncol = 6)[,6:1]))]
### N_response need to be consistent with those specified in get_simulated_matrices()
FC_resp <- convert_to_TIRT_response(Utility = cfa_matrices$Utility,
                                    block_design = make_random_block(60, 60, 3),
                                    N_response = 100, format = "pairwise",
                                    block_size = 3, N_blocks = 20)
FC_rank_resp <- convert_to_TIRT_response(Utility = cfa_matrices$Utility,
                                         block_design = make_random_block(60, 60, 5),
                                         N_response = 100, format = "ranks",
                                         block_size = 5, N_blocks = 12)
FC_rank_partial_resp <- convert_to_TIRT_response(Utility = cfa_matrices$Utility,
                                            block_design = make_random_block(60, 60, 5),
                                      N_response = 100, format = "ranks", partial = TRUE,
```

```
                                     block_size = 5, N_blocks = 12)
```

FC_resp
FC_rank_resp
FC_rank_partial_resp

`empirical_reliability` *Calculate the Empirical Reliability of the Latent Trait Scores, Following the Formula in Brown & Maydeu-Olivares (2018).*

Description

Calculates the empirical reliability using the formula in Brown & Maydeu-Olivares (2018).

Usage

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

Arguments

dataset	Data frame with trait estimates and standard errors
score_names	Vector of characters. Which columns specify trait scores?
se_names	Vector of characters. Which columns specify trait standard errors?

Details

For trait scores estimated using item response theory models, a suitable reliability estimate is empirical reliability, which provides a summary estimate on how reliable the trait scores are "as a whole".

Value

A numeric vector containing empirical reliability estimates, ordered the same as in `score_names`.

Author(s)

Mengtong Li

References

Brown, A., & Maydeu-Olivares, A. (2018). Ordinal factor analysis of graded-preference questionnaire data. *Structural Equation Modeling: A Multidisciplinary Journal, 25*(4), 516-529. https://doi.org/10.1080/10705511.20

Examples

```
## Not run: empirical_reliability(dataset, c("Trait1", "Trait2", "Trait3"), c("se1", "se2", "se3"))
```

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"))
```

fit_TIRT_model *Fit the Thurstonian IRT Model with Long Format Response Data*

Description

Fits the Thurstonian IRT response model using either lavaan, Mplus, or stan methods. A long format response data set needs to be provided.

Usage

```
fit_TIRT_model(
  data_TIRT,
  method = "lavaan",
  lavaan_estimator = "WLSMV",
  stan_cores = 4,
  chains = 4,
  iter = 2000,
  verbose = TRUE,
```

```
  remove_mplus_file = FALSE,
  export_estimates = TRUE,
  file_name
)
```

Arguments

`data_TIRT` Long format TIRT response data as generated from `get_TIRT_long_data()` or `thurstonianIRT::make_TIRT_data()`.

`method` Estimation method for the TIRT model. Can be `"lavaan"`, `"mplus"` or `"stan"`.

`lavaan_estimator` Which estimator to use when lavaan is chosen as the method of estimating the TIRT model. Defaults to "WLSMV".

`stan_cores, chains, verbose, iter` Parameters used in `thurstonianIRT::fit_TIRT_stan`

`remove_mplus_file` Whether the input/output files will be removed after model estimation, when Mplus is chosen as the method of estimating the TIRT model.

`export_estimates` Logical. Should trait estimates be written to external files?

`file_name` If `export_estimates == TRUE`, specify the file name for the output file.

Details

This function incorporates the fit TIRT models functions in the `thurstonianIRT` package (Bürkner, 2019) and by (a) providing a wrapper interface for users to choose from estimating from lavaan, MPLUS, or stan, (b) placing the fit object, resulting trait estimates, and the original long format response data into one list as the return object. Users need to provide a long format TIRT response data set as generated from `get_TIRT_long_data()` or from `thurstonianIRT::make_TIRT_data()`, and they can choose from three estimation methods: lavaan, MPLUS or stan. For lavaan and stan, additional arguments can be specified.

We note that currently the lavaan method does not provide standard error estimates. The stan method is the most stable but can take a very long time for estimation. The mplus method can be a good choice but may occassionally produce errors due to model convergence issues. In these rare cases, users may also consider using the Excel macro developed by Brown & Maydeu-Olivares (2012) to generate Mplus syntax and directly run the syntax in Mplus.

Value

A list containing:

`final_estimates` Final trait score and standard error estimates

`fit_object` TIRT model fit object

`responses_TIRT` The long format TIRT response

`long_estimates` Final trait score and standard error estimates, in long format

Author(s)

Mengtong Li

References

Bürkner, P. C. (2019). thurstonianIRT: Thurstonian IRT models in R. *Journal of Open Source Software, 4*(42), 1662. https://doi.org/10.21105/joss.01662

Brown, A., & Maydeu-Olivares, A. (2012). Fitting a Thurstonian IRT model to forced-choice data using Mplus. *Behavior Research Methods, 44*, 1135-1147. https://doi.org/10.3758/s13428-012-0217-x

See Also

`thurstonianIRT::fit_TIRT_lavaan`,

`thurstonianIRT::fit_TIRT_mplus`,

`thurstonianIRT::fit_TIRT_stan`

Examples

```
set.seed(2024)
test_data <- triplet_example_data[1:20,]
block_info <- triplet_block_info
test_data_long <- get_TIRT_long_data(block_info = triplet_block_info, response_data = test_data,
                                    response_varname = build_TIRT_var_names(N_blocks = 5,
                                      block_size = 3, item_name = "i"),
                                      block_name = "Block", item_name = "ID",
                                      trait_name = "Factor", sign_name = "Keying")
## Not run:
    test_fit <- fit_TIRT_model(test_data_long, method = "lavaan")
    test_fit$fit_object
    test_fit$final_estimates

## End(Not run)
```

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

Description

Reads in responses to Likert-type scales and a specified factor model, performs CFA, and produces parameter estimates required for producing subsequent simulation data (i.e., use as inputs for `get_simulation_matrices()`).

This function returns factor loadings, intercepts, residual variances, and covariances among latent variables

Usage

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

Arguments

response_data Likert-type response data. Requires the header including variable names.

fit_model A pre-specified CFA model written in lavaan syntax.

item_names Names of the items you wish to obtain loadings, intercepts, and residuals. These variable names should appear in response_data.

Details

This function is essentially a wrapper for lavaan::parameterEstimates() to obtain specific set of parameter estimates.

Notice that we assume your CFA model does not have hierarchical factor structure, nor does it have cross loadings or correlated residuals.

Value

A list containing:

- loadings Item loadings for item_names
- intercepts Item intercepts for item_names
- residuals Item residual variances for item_names
- covariances Covariances between latent variables defined in fit_model
- model_fit Model fit for fit_model on response_data

Author(s)

Mengtong Li

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. https://doi.org/10.1080/00223890902935878

See Also

get_simulation_matrices()

Examples

```
## We have a small response sample data (N = 100) on
## the HEXACO-60 (Ashton & Lee, 2009) scale in the package
## Names of the items are SS1-SS60 (Single-statement)
rating_data <- HEXACO_example_data
cfa_model <- paste0("H =~ ", paste0("SS", seq(6,60,6), collapse = " + "), "\n",
                    "E =~ ", paste0("SS", seq(5,60,6), collapse = " + "), "\n",
                    "X =~ ", paste0("SS", seq(4,60,6), collapse = " + "), "\n",
                    "A =~ ", paste0("SS", seq(3,60,6), collapse = " + "), "\n",
                    "C =~ ", paste0("SS", seq(2,60,6), collapse = " + "), "\n",
                    "O =~ ", paste0("SS", seq(1,60,6), collapse = " + "), "\n")
cfa_estimates <- get_CFA_estimates(response_data = rating_data,
```

```
                                    fit_model = cfa_model,
                                    item_names = paste0("SS",c(1:60)))
```

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 Thurstonian IRT Model Based on Confirmatory Factor Analysis Results*

Description

This function takes in factor analysis results from `lavaan::cfa()` or `get_CFA_estimates()`, and generates simulated person and item parameter matrices for the Thurstonian IRT model. The latent "utility" value of each item for each simulated person is also produced.

Usage

```
get_simulation_matrices(
  loadings,
  intercepts,
  residuals,
  covariances,
  N,
  N_items,
  N_dims,
  dim_names,
  empirical
)
```

Arguments

`loadings`, `intercepts`, `residuals`, `covariances`
: Data frame of factor loadings, intercepts, residuals and latent variable covariances, preferably obtained from `get_CFA_estimates()`, or extracted from `lavaan::parameterEstimat`

`N`
: Number of simulated responses you wish to generate.

`N_items`
: Optional. Total number of response items. Default to the number of rows in `loadings`.

`N_dims`
: Optional. Total number of response items. Default to the length of `dim_names`.

`dim_names`
: Name of the latent variables (dimensions); Order should be consistent with how they appear in your CFA model as you have specified in `get_CFA_estimates()`.

`empirical`
: As in `MASS::mvrnorm()`; Should mu and sigma specify the empirical, rather than population mean and covariance?

Details

Based on the Thurstonian IRT model (Brown & Maydeu-Olivares, 2011), this function generates the latent utility value of `N_item` Likert items for each of the `N` participants.

Readers can refer to Brown & Maydeu-Olivares (2011) and the online tutorial in Li et al., (in press) for detailed description of simulation procedures.

Value

A list containing:

- `Lambda` Item loading matrix specifying which items load onto which dimension,
- `Mu` Item intercept matrix,
- `Epsilon` Item residual matrix,
- `Theta` Simulated latent scores for each of the `N_dims` dimensions for all `N` simulated respondents,
- `Utility` latent utility value of `N_item` Likert items for each of the `N` participants.

Author(s)

Mengtong Li

References

Brown, A., & Maydeu-Olivares, A. (2011). Item response modeling of forced-choice questionnaires. *Educational and Psychological Measurement, 71*(3), 460-502. https://doi.org/10.1177/0013164410375112
Li, M., Zhang, B., Li, L., Sun, T., & Brown, A., (2024). Mixed-Keying or Desirability-Matching in the Construction of Forced-Choice Measures? An Empirical Investigation and Practical Recommendations. *Organizational Research Methods*. https://doi.org/10.1177/10944281241229784

See Also

`get_CFA_estimates()`

Examples

```
rating_data <- HEXACO_example_data
cfa_model <- paste0("H =~ ", paste0("SS", seq(6,60,6), collapse = " + "), "\n",
                    "E =~ ", paste0("SS", seq(5,60,6), collapse = " + "), "\n",
                    "X =~ ", paste0("SS", seq(4,60,6), collapse = " + "), "\n",
                    "A =~ ", paste0("SS", seq(3,60,6), collapse = " + "), "\n",
                    "C =~ ", paste0("SS", seq(2,60,6), collapse = " + "), "\n",
                    "O =~ ", paste0("SS", seq(1,60,6), collapse = " + "), "\n")
cfa_estimates <- get_CFA_estimates(response_data = rating_data,
                                   fit_model = cfa_model,
                                   item_names = paste0("SS",c(1:60)))
cfa_matrices <- get_simulation_matrices(loadings = cfa_estimates$loadings,
                                        intercepts = cfa_estimates$intercepts,
                                        residuals = cfa_estimates$residuals,
                                        covariances = cfa_estimates$covariances,
                                        N = 100, N_items = 60, N_dims = 6,
                                        dim_names = c("H", "E", "X", "A", "C", "O"),
                                        empirical = TRUE)
```

get_TIRT_long_data *Convert the TIRT Pairwise/Rank Response Data into Long Format Compatible with the thurstonianIRT Package*

Description

To estimate the TIRT model using the thurstonianIRT Package, the pairwise/rank data needs to be converted into long format. Current function serves as that purpose.

Usage

```
get_TIRT_long_data(
  block_info,
  response_data,
  response_varname,
  format = "pairwise",
  partial = FALSE,
  direction = "larger",
  family = "bernoulli",
  range = c(0, 1),
  block_name = "Block",
  item_name = "ID",
  trait_name = "Factor",
  sign_name = "Reversed"
)
```

Arguments

`block_info` Information data frame related to keying, dimension, and ID of each item in each block. The order of the rows in `block_info` need to be consistent with the order of the FC items.

`response_data` TIRT pairwise/rank response data.

`response_varname` Column names of TIRT pairwise/ranked responses. Can be generated from `build_TIRT_var_names()`.

`format`, `direction`, `family`, `range`, `partial` These parameters works the same as `thurstonianIRT::make_TIRT_data()`.

`block_name`, `item_name`, `trait_name`, `sign_name` These parameters indicate the column names in `block_info` that specify the following information of each item:

`block_name`: Which block does this item belong to? `item_name`: What is the name of this item? `trait_name`: Which trait does this item belong to? `sign_name`: What is the keying of this item?

Details

This function is essentially a wrapper of `thurstonianIRT::make_TIRT_data()` to allow more functionalities to be incorporated in a single function.

Value

A long format data frame that is compatible with subsequent analyses using the thurstonianIRT package

Author(s)

Mengtong Li

See Also

`thurstonianIRT::set_blocks_from_df()`, `thurstonianIRT::make_TIRT_data()`

Examples

```
## See example in convert_to_TIRT_response() for FC_resp
## This example is just for demonstrative purposes showing how the long format data would look like.
## Not run: get_TIRT_long_data(block_info = triplet_block_info,
                              response_data = FC_resp[,c(1:15)],
                    response_varname = build_TIRT_var_names("i", 3, 5, format = "pairwise"),
                              trait_name = "Factor", sign_name = "Keying")
## End(Not run)
```

`HEXACO_example_data` *Example HEXACO Response Data*

Description

Responses to the HEXACO-60 (Ashton & Lee, 2009) scale from 100 actual respondents

Usage

```
HEXACO_example_data
```

Format

A data frame with 100 rows and 60 columns.

SS1, SS2, SS3, SS4, SS5, SS6, SS7, SS8, SS9, SS10, SS11, SS12, SS13, SS14, SS15, SS16, SS17, SS18, SS19, SS20, SS21
: Represents the 60 HEXACO items.

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. https://doi.org/10.1080/00223890902935878

make_random_block *Construction of Random Item Blocks*

Description

Returns a matrix of randomly paired blocks where each row represents a block.

Usage

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

Arguments

total_items	Integer value. Determines the total number of items we sample from.
target_items	Integer value. Determines the number of items to use from total_items to build item blocks. Default to be equal to total_items. Should be no more than total_items.
item_per_block	Integer value. Determines the number of items in each item block. Should be no less than 2.

Details

Given the total number of items to pair from, number of items to build paired blocks and number of items in each block, make_random_block produces a matrix randomly paired blocks where each row represents a block.

It can also accommodate cases when target_items is not a multiple of item_per_block.

Can be used as initial solution for other functions in this package.

Value

A matrix of integers indicating the item numbers, where the number of rows equals target_items divided by item_per_block, rounded up, and number of columns equals item_per_block.

Note

If target_items is not a multiple of item_per_block, the item set produced by target_items will be looped until number of sampled items becomes a multiple of item_per_block.

Author(s)

Mengtong Li

Examples

```
# Try out cases where you make target_items the default.
make_random_block(60, item_per_block = 3)

# You can also set your own values of target_items.
make_random_block(60, 45, item_per_block = 3)

# Also see what happens if target_items is not a multiple of item_per_block.
make_random_block(60, 50, item_per_block = 3)
```

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

Description

This function is a simple plot for diagnostic purposes examining the performance of the FC scale based on simulated data.

Usage

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

Arguments

`x_scores`	Scores to be plotted on the x axis
`y_scores`	Scores to be plotted on the y axis
`type`	Which type of plots is plotted? Can be `"simple"` for simple x-y plot, or `"abs.diff"` for plotting absolute difference of (y-x) vs x.
`...`	Other parameters used in `plot()`

Details

This is only a very crude plot function extending `plot()` for demonstrative purposes. Users are free to develop their own versions of plotting.

Value

A scatter plot

Author(s)

Mengtong Li

Examples

```
plot_scores(rnorm(100), rnorm(100))
```

predict_scores *Predict trait scores based on estimated model*

Description

An easy wrapper for the `thurstonianIRT::predict()` function

Usage

```
predict_scores(estimated_model, newdata, output_file = NULL)
```

Arguments

estimated_model	Estimated model
newdata	Response data from new response samples, in TIRT data format. Preferably be generated from `thurstonianIRT::make_TIRT_data()` or `get_TIRT_long_data()`.
output_file	Character string. If specified, output the trait scores into a specified csv file.

Value

A data frame containing estimated trait scores of the new response sample

Author(s)

Mengtong Li

Examples

```
test_data <- triplet_example_data[1:20,]
block_info <- triplet_block_info
test_data_long <- get_TIRT_long_data(block_info = triplet_block_info, response_data = test_data,
                                   response_varname = build_TIRT_var_names(N_blocks = 5,
                                     block_size = 3, item_name = "i"),
                                     block_name = "Block", item_name = "ID",
                                     trait_name = "Factor", sign_name = "Keying")
## Not run:
    test_fit <- fit_TIRT_model(test_data_long, method = "mplus")
    predict_scores(test_fit$fit_object, newdata = test_data[21:40,])

## End(Not run)
```

RMSE_range *Calculate the Overall RMSE of the Trait Scores, or the RMSE in a Certain Trait Score Range*

Description

This function is also for diagnostic purposes, examining which interval on the latent trait continuum does the FC scale demonstrate the best measurement accuracy.

Usage

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

Arguments

true_scores	Actual trait scores
estimated_scores	Estimated trait scores from a specified model
range_breaks	A numeric vector. Specifies which trait scores ranges will the RMSE be calculated

Details

TO BE DONE

Value

If range_breaks is not specified, an overall RMSE numeric value will be returned; else, a named list showing the RMSE in each score range will be returned.

Author(s)

Mengtong Li

Examples

```
RMSE_range(rnorm(100), rnorm(100))
RMSE_range(rnorm(100), rnorm(100), range_breaks = c(-3, -2, -1, 0, 1, 2, 3))
```

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

Description

Automatic construction of forced-choice tests based on 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.

Usage

```
sa_pairing_generalized(block, total_items, Temperature,
                                eta_Temperature = 0.01, r = 0.999,
                                end_criteria = 10^(-6),
                                item_chars, weights, FUN,
                                n_exchange = 2, prob_newitem = 0.25,
                                use_IIA = FALSE, rater_chars,
                                iia_weights = c(BPlin = 1, BPquad = 1,
                                AClin = 1, ACquad = 1))
```

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.
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 this `block`? Should be more than number of unique values in `block`.

`Temperature` Initial temperature value. Can be left blank and be computed based on the absolute value of initial energy of `block` (Recommended), and scaled by `eta_Temperature`.
In general, higher temperature represents a higher probability of accepting an inferior solution.

`eta_Temperature` A positive numeric value. The ratio of initial temperature to initial energy of `block`, if `Temperature` is not designated.

`r` A positive numeric value less than 1. Determines the reduction rate of `Temperature` after each iteration.

`end_criteria` A positive numeric value less than 1. Iteration stops when temperature drops to below `end_criteria * Temperature`. Default to be 10^-6.

`item_chars` An m by r data frame, where m is the total number of items to sample from, whether it is included in the block or not, whereas r is the number of item characteristics.

weights	A vector of length *r* with weights for each item characteristics in `item_chars`. Should provide a weight of 0 for specific characteristics not of interest, such as item ID.
FUN	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(block)`.
prob_newitem	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.
use_IIA	Logical. Are IIA metrics used when performing automatic pairing?
rater_chars	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.

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 `weights, iia_weights, eta_Temperature` to find out the best combination of initial temperature and energy value in order to provide optimally paired blocks.

Use `cal_block_energy_with_iia` if inter-item agreement (IIA) metrics are needed.

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.

Examples

```
## Simulate 60 items loading on different Big Five dimensions,
## with different mean and item difficulty

item_dims <- sample(c("Openness","Conscientiousness","Neuroticism",
                     "Extraversion","Agreeableness"), 60, replace = TRUE)
item_mean <- rnorm(60, 5, 2)
item_difficulty <- runif(60, -1, 1)


item_df <- data.frame(Dimensions = item_dims,
                     Mean = item_mean, Difficulty = item_difficulty)
solution <- make_random_block(60, 60, 3)


item_responses <- matrix(sample(seq(1:5), 600*60, replace = TRUE), nrow = 60, byrow = TRUE)

## Automatic pairing, without use of IIAs
## See ?facfun for information about what it does

sa_pairing_generalized(solution, 60, eta_Temperature = 0.01,
                                   r = 0.999, end_criteria = 0.001,
                                   weights = c(1,1,1),
                                   item_chars = item_df,
                                   FUN = c("facfun", "var", "var"))


## Automatic pairing, with IIAs

sa_pairing_generalized(solution, 60, eta_Temperature = 0.01,
                                   r = 0.999, end_criteria = 0.001,
                                   weights = c(1,1,1),
                                   item_chars = item_df,
                                   FUN = c("facfun", "var", "var"),
                                   use_IIA = TRUE,
                                   rater_chars = item_responses,
                                   iia_weights = c(BPlin = 1, BPquad = 1,
                                   AClin = 1, ACquad = 1))
```

`triplet_block_info` *Block Information for the Example Triplet Response Data*

Description

Block design information of the 5-block triplet for `triplet_example_data`, containing ID, factor, keying, and block membership for the 15 items.

Usage

```
triplet_block_info
```

Format

A data frame with 15 rows and 4 columns.

Keying Indicates whether the item is positively keyed (1) or negatively keyed (-1).

Factor Indicates the factor of the item. It can be either one of the six HEXACO traits.

Block Indicates which block this item belongs to.

ID Indicates the ID of the item.

Source

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

---

`triplet_example_data` *Example Triplet Response Data*

Description

Responses to a 5-block triplet forced-choice measure, converted into pairwise comparisons. This dataset originates from Li et al.'s (in press) study on forced-choice measurement.

Usage

```
triplet_example_data
```

Format

A data frame with 541 rows and 15 columns.

i1i2, i1i3, i2i3, i4i5, i4i6, i5i6, i7i8, i7i9, i8i9, i10i11, i10i12, i11i12, i13i14, i13i15, i14i15 All possible pairwise comparisons among items in the same block. With a triplet format, each block contains three items, producing three possible pairwise comparisons among these items. Therefore, i1, i2, and i3 belong to block 1, and so on.

i1i2 represents the pairwise comparison indicating whether i1 is preferable over i2. If i1 is preferable over i2, then i1i2 = 1, else i1i2 = 0.

Source

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

References

Li, M., Zhang, B., Li, L., Sun, T., & Brown, A., (in press). Mixed-Keying or Desirability-Matching in the Construction of Forced-Choice Measures? An Empirical Investigation and Practical Recommendations. *Organizational Research Methods*.

Index

∗ **datasets**
  HEXACO_example_data, 23
  triplet_block_info, 30
  triplet_example_data, 31

build_scale_with_blueprint, 3
build_TIRT_var_names, 6

cal_block_energy, 7
cal_block_energy_with_iia, 8
construct_blueprint, 10
convert_to_TIRT_response, 11

empirical_reliability, 14

facfun, 15
fit_TIRT_model, 15

get_CFA_estimates, 17
get_iia, 19
get_simulation_matrices, 20
get_TIRT_long_data, 22

HEXACO_example_data, 23

make_random_block, 24

plot_scores, 25
predict_scores, 26

RMSE_range, 27

sa_pairing_generalized, 28

triplet_block_info, 30
triplet_example_data, 31