Package 'semFromKeys'

July 24, 2026

Type Package

Title Run 'lavaan' Models from Keys Lists

Version 0.2.2

Description Specifying 'lavaan' models manually can be time consuming when multiple similar models are required. The 'semFromKeys' package streamlines the process of running 'lavaan' models by generating model code from simple keys lists and running entire collections of models at once.
The package was inspired by the process used in the code for
Bainbridge, T. F., Ludeke, S. G., & Smillie, L. D. (2022)
<doi:10.1037/pspp0000395>.
The package also optionally checks that identical models have not been run on the same data, which saves time when code needs to be run again.

License GPL (>= 3)

Encoding UTF-8

LazyData true

Config/roxygen2/version 8.0.0

Depends R (>= 3.6.0)

Imports stringr, lavaan, openssl, withr, tools

Suggests testthat (>= 3.0.0), here, rstudioapi

Config/testthat/edition 3

URL https://github.com/timbainbridge/semFromKeys

BugReports https://github.com/timbainbridge/semFromKeys/issues

NeedsCompilation no

Author Timothy F. Bainbridge [aut, cre, cph]

Maintainer Timothy F. Bainbridge <tfbainbridge@gmail.com>

Repository CRAN

Date/Publication 2026-07-24 09:40:08 UTC

Contents

`BFIGritHope` *BFI, Grit, and Hope data from Bainbridge, Ludeke, & Smillie (2022), Study 2b.*

Description

The subset of the data reported in the paper include only the BFI-2, Grit, and hope.

Usage

```
BFIGritHope
```

Format

`BFIGritHopeImp`**:**

A data frame with 388 rows and 95 columns:

bfi_e1_1 BFI-2 Extraversion: Sociability, Item 1

bfi_e1_2 BFI-2 Extraversion: Sociability, Item 2

bfi_e1_3 BFI-2 Extraversion: Sociability, Item 3

bfi_e1_4 BFI-2 Extraversion: Sociability, Item 4

bfi_e2_1 BFI-2 Extraversion: Assertiveness, Item 1

bfi_e2_2 BFI-2 Extraversion: Assertiveness, Item 2

bfi_e2_3 BFI-2 Extraversion: Assertiveness, Item 3

bfi_e2_4 BFI-2 Extraversion: Assertiveness, Item 4

bfi_e3_1 BFI-2 Extraversion: Energy Level, Item 1

bfi_e3_2 BFI-2 Extraversion: Energy Level, Item 2

bfi_e3_3 BFI-2 Extraversion: Energy Level, Item 3

bfi_e3_4 BFI-2 Extraversion: Energy Level, Item 4

bfi_a1_1 BFI-2 Agreeableness: Compassion, Item 1

bfi_a1_2 BFI-2 Agreeableness: Compassion, Item 2

bfi_a1_3 BFI-2 Agreeableness: Compassion, Item 3

bfi_a1_4 BFI-2 Agreeableness: Compassion, Item 4

bfi_a2_1 BFI-2 Agreeableness: Respectfulness, Item 1
bfi_a2_2 BFI-2 Agreeableness: Respectfulness, Item 2
bfi_a2_3 BFI-2 Agreeableness: Respectfulness, Item 3
bfi_a2_4 BFI-2 Agreeableness: Respectfulness, Item 4
bfi_a3_1 BFI-2 Agreeableness: Trust, Item 1
bfi_a3_2 BFI-2 Agreeableness: Trust, Item 2
bfi_a3_3 BFI-2 Agreeableness: Trust, Item 3
bfi_a3_4 BFI-2 Agreeableness: Trust, Item 4
bfi_c1_1 BFI-2 Conscientiousness: Organization, Item 1
bfi_c1_2 BFI-2 Conscientiousness: Organization, Item 2
bfi_c1_3 BFI-2 Conscientiousness: Organization, Item 3
bfi_c1_4 BFI-2 Conscientiousness: Organization, Item 4
bfi_c2_1 BFI-2 Conscientiousness: Productiveness, Item 1
bfi_c2_2 BFI-2 Conscientiousness: Productiveness, Item 2
bfi_c2_3 BFI-2 Conscientiousness: Productiveness, Item 3
bfi_c2_4 BFI-2 Conscientiousness: Productiveness, Item 4
bfi_c3_1 BFI-2 Conscientiousness: Responsibility, Item 1
bfi_c3_2 BFI-2 Conscientiousness: Responsibility, Item 2
bfi_c3_3 BFI-2 Conscientiousness: Responsibility, Item 3
bfi_c3_4 BFI-2 Conscientiousness: Responsibility, Item 4
bfi_n1_1 BFI-2 Negative Emotionality: Anxiety, Item 1
bfi_n1_2 BFI-2 Negative Emotionality: Anxiety, Item 2
bfi_n1_3 BFI-2 Negative Emotionality: Anxiety, Item 3
bfi_n1_4 BFI-2 Negative Emotionality: Anxiety, Item 4
bfi_n2_1 BFI-2 Negative Emotionality: Depression, Item 1
bfi_n2_2 BFI-2 Negative Emotionality: Depression, Item 2
bfi_n2_3 BFI-2 Negative Emotionality: Depression, Item 3
bfi_n2_4 BFI-2 Negative Emotionality: Depression, Item 4
bfi_n3_1 BFI-2 Negative Emotionality: Emotional Volatility, Item 1
bfi_n3_2 BFI-2 Negative Emotionality: Emotional Volatility, Item 2
bfi_n3_3 BFI-2 Negative Emotionality: Emotional Volatility, Item 3
bfi_n3_4 BFI-2 Negative Emotionality: Emotional Volatility, Item 4
bfi_o1_1 BFI-2 Open-Mindedness: Intellectual Curiosity, Item 1
bfi_o1_2 BFI-2 Open-Mindedness: Intellectual Curiosity, Item 2
bfi_o1_3 BFI-2 Open-Mindedness: Intellectual Curiosity, Item 3
bfi_o1_4 BFI-2 Open-Mindedness: Intellectual Curiosity, Item 4
bfi_o2_1 BFI-2 Open-Mindedness: Aesthetic Sensitivity, Item 1
bfi_o2_2 BFI-2 Open-Mindedness: Aesthetic Sensitivity, Item 2
bfi_o2_3 BFI-2 Open-Mindedness: Aesthetic Sensitivity, Item 3
bfi_o2_4 BFI-2 Open-Mindedness: Aesthetic Sensitivity, Item 4
bfi_o3_1 BFI-2 Open-Mindedness: Creative Imagination, Item 1
bfi_o3_2 BFI-2 Open-Mindedness: Creative Imagination, Item 2

bfi_o3_3 BFI-2 Open-Mindedness: Creative Imagination, Item 3

bfi_o3_4 BFI-2 Open-Mindedness: Creative Imagination, Item 4

grit_c_1 Grit Consistency of Interests, Item 1

grit_c_2 Grit Consistency of Interests, Item 2

grit_c_3 Grit Consistency of Interests, Item 3

grit_c_4 Grit Consistency of Interests, Item 4

grit_c_5 Grit Consistency of Interests, Item 5

grit_c_6 Grit Consistency of Interests, Item 6

grit_p_1 Grit Persistence, Item 1

grit_p_2 Grit Persistence, Item 2

grit_p_3 Grit Persistence, Item 3

grit_p_4 Grit Persistence, Item 4

grit_p_5 Grit Persistence, Item 5

grit_p_6 Grit Persistence, Item 6

hope_a_1 Hope Agency, Item 1

hope_a_2 Hope Agency, Item 2

hope_a_3 Hope Agency, Item 3

hope_a_4 Hope Agency, Item 4

hope_p_1 Hope Pathways, Item 1

hope_p_2 Hope Pathways, Item 2

hope_p_3 Hope Pathways, Item 3

hope_p_4 Hope Pathways, Item 4

Source

https://osf.io/f9hmg/files/eu8p9

`bifactor.from.keys` *Runs bifactor models for multiple scales based on keys lists.*

Description

`bifactor.from.keys` runs a series of bifactor model from three keys lists— one for items on general factor; one for items on group factors; and one for group factors on general factors. The keys list must be named appropriately (i.e., general factor names, group factor names, and general factor names for the three lists respectively). The function is designed to streamline running measurement models for all scales in a sample and to input model outputs into downstream functions.

Usage

```
bifactor.from.keys(
  keys_g,
  keys_b,
  keys,
  data,
  fit_save = TRUE,
  fit_measures = "all",
  std.lv = TRUE,
  miss = "ML",
  est = "default",
  name = "bifactor",
  check = FALSE,
  save_out = FALSE
)
```

Arguments

keys_g	A named list of items in general factors. Names must be the names of the general factors. Each list element must be a vector of items that load on the general factors. Must be the same length as keys_b.
keys_b	A named list of group factors in general factors. Names must be the names of the general factors. Each list element must be a vector of group factors that load on the general factors. Must be the same length as keys_g.
keys	A named list of items in group factors. Names must be the names of the group factors. Each list element must be a vector of items that load on the group factors. Need not be the same length as keys_g and keys_b.
data	A dataframe or object coercible to a dataframe. Data must include all observed variables in any of the keys.
fit_save	Logical. TRUE indicates model fit measures should be included in the output; FALSE indicates model fit measures should not be included in the output. FALSE may be desirable when fit measures are of little interest and when they may take a long time to estimate. Fit can still be examined for individual models with, lavaan::fitMeasures(fitfit[model name]).
fit_measures	A vector of fit measures to save or 'all' to select all fit measures, as per the fit.measures parameter from lavaan's lavaan::fitMeasures() function. Defaults to 'all'. Irrelevant if fit_save = FALSE.
std.lv	Logical. Sets the std.lv parameter, as per lavaan (see lavaan::lavOptions()). TRUE indicates that factor variances should be fixed to 1. FALSE indicates that loadings of the first items of factors should be fixed to 1. Defaults to TRUE.
miss	A string. Sets the missing parameter, as per lavaan (see lavaan::lavOptions()). Defaults to 'ML'.
est	A string. Sets the estimator parameter, as per lavaan (see lavaan::lavOptions()). The default ('default') uses the lavaan default for the model being run.

name	A string indicating a subdirectory where model outputs will be saved when save_out = TRUE and checked against when check = TRUE. Defaults to 'bifactor'. Irrelevant if both save_out = FALSE and check = FALSE. The name should be unique for each set of models or outputs from calls with the same name will be overwritten.
check	Logical. TRUE indicates that current inputs should be compared to previous inputs if they exist and that the model should not be rerun if nothing has changed; FALSE indicates that these checks should not be made and the model should be run regardless of the existence of previous inputs.
save_out	Logical. TRUE indicates that model code, a hash of the data, important input parameter values, and output will be saved; FALSE indicates that nothing will be saved. Selecting save_out = TRUE enables the function to not rerun models next time if check = TRUE the next time the code is run and nothing has changed in the meantime.

Details

The model relies on sem.check() for the back-end of running the models. This enables saving inputs and outputs from model runs (with save_out = TRUE) and checking to see if anything has changed from prior runs before running again (with check = TRUE). The functionality was included for a number of very slow models or a lot of faster models, such that time spent rerunning them would be onerous. For further details on how this works, see the sem.check() function documentation.

Please be careful with bifactor models. In simulation studies, they can fit better than the models that generated the data in the presence of unspecified complexity (which is usually the case; Murray & Johnson, 2013). They can also produce negative residual variances. lavaan will warn you of this and you should deal with it before proceeding. Items can also load in atheoretical ways on the general or group factors. This could be some items loading negatively instead of positively or vice versa or one or more items having insignificant loadings on a factor that they should theoretically load on.

These problems can often be solved by omitting a factor. When items from one group factor dominate the general factor, the best solution might be to remove the general factor and treat the group factors as separate constructs. Perhaps more frequently, these problems might be solved by omitting one (or more) group factors (i.e., an S-1 model; Eid et al., 2017). This can either be done a priori (perhaps based on which one should theoretically be most 'central' to the general factor) or after examining the fully specified bifactor model and dropping the worst performing factor (see the example). When more than one group factor is poor, I am not aware of any specific advice on which to remove, so use your judgment if previous research has not got any advice for your case.

Value

Returns a list of length 2 (if fit_save = FALSE) or 3 (if fit_save = TRUE). The elements of the list are: a list of lavaan model output objects; a list of parameter estimates from the models (standardized if std = TRUE); and, if fit_save = TRUE, a matrix of fit measures for each model.

References

Eid, M., Geiser, C., Koch, T., & Heene, M. (2017). Anomalous results in G-factor models: Explanations and alternatives. Psychological Methods, 22(3), 541-562. https://doi.org/10.1037/met0000083.

Murray, A. L. & Johnson, W. (2013). The limitations of model fit in comparing the bi-factor versus higher-order models of human cognitive ability structure. Intelligence, 41(5), 407-422. https://doi.org/10.1016/j.intell.2013.06.004.

See Also

`sem.check()`, which `bifactor.from.keys()` uses for all the back-end, and `lavaan::sem()`, which is used to estimate the models.

Examples

```
# Create keys
keys0 <- c("grit_c", "grit_p", "hope_a", "hope_p")
keys <- sapply(
  keys0, function(x) names(BFIGritHope)[grep(x, names(BFIGritHope))]
)
keys_g0 <- c("grit", "hope")
keys_g <- sapply(
  keys_g0, function(x) names(BFIGritHope)[grep(x, names(BFIGritHope))]
)
keys_b1 <- sapply(
  keys_g0, function(x) keys0[grep(x, keys0)], simplify = FALSE
)
# Including all factors produces a negative residual variance for hope.
bif_fit0 <- bifactor.from.keys(
  keys_g, keys_b1, keys, BFIGritHope, check = FALSE, fit_save = TRUE
)
summary(bif_fit0$fit$hope)
# Fix negative residual variance by removing the first group factor.
keys_b <- keys_b1
keys_b$hope <- keys_b1$hope[-1]
bif_fit <- bifactor.from.keys(
  keys_g, keys_b, keys, BFIGritHope, check = FALSE, fit_save = TRUE
)
# Examine some results
summary(bif_fit$fit$grit)                  # Standard lavaan summary
bif_fit$fit_measures[, c("cfi", "rmsea")]  # Fit measures
```

`cache.clean` *Cleans selected files from the current cache directory*

Description

Functions in the package include options to write files to a cache directory, set up with the `cache.setup()` function. Obsoletely files might be created if the `name` parameter in a function call is changed or if a function call is no longer being used. The `cache.clean()` function provides a way to find files that have not be modified in the last `older_than` days and lists the files for the user to agree to delete or not.

Usage

```
cache.clean(older_than = NULL, interactive = TRUE)
```

Arguments

older_than	A positive number indicating number of days (fractions allowed). Files with older modification times than this will be deleted after confirmation if interactive = TRUE.
interactive	Logical. TRUE indicates that confirmation will be required before files are deleted. FALSE indicates that files will be deleted without requiring confirmation. It is recommended to use TRUE except for carefully checked automated processes. Defaults to TRUE.

Details

The function is interactive if run in an interactive session with interctive = TRUE. In addition to deleting old files, the function will also optionally delete empty directories and, if the top level cache directory is also empty, then it will also optionally delete the cache directory and unset it. The latter will only be done in interactive sessions with interactive = TRUE to avoid breaking non-interactive code.

One way to clean only unused files is to run all current code and then run cache.clean() with older_than set to something greater than the number of days it takes for the code to run and less than when the code was previously run. For example, if your code takes 5 minutes to run and you previously ran it 2 days ago, you could run all your code and, when it has finished, use cache.clean(older_than = 1).

Value

NULL (invisibly). Primarily called to clean up the cache directory.

See Also

cache.setup()

Examples

```
# Setup a cache directory
cache.setup()
# Now code with check = TRUE or save_out = TRUE will work, e.g.,
# Create CFA keys
keys0 <- c("grit_c", "grit_p", "hope_a", "hope_p")
keys <- sapply(
  keys0, function(x) names(BFIGritHope)[grep(x, names(BFIGritHope))]
)
# Run models
cfa_fit <- cfa.from.keys(
  keys, BFIGritHope, fit_save = TRUE, check = TRUE, save_out = TRUE
)
# Check that they are not estimated again.
cfa_fit <- cfa.from.keys(
```

```
  keys, BFIGritHope, fit_save = TRUE, check = TRUE, save_out = TRUE
)
cache.clean(60/86400)  # Delete files not modified in the last minute.
```

---

cache.setup *Sets up a cache directory for storing model inputs and outputs*

Description

Using save_out = TRUE or check = TRUE from cfa.from.keys(), bifactor.from.keys(), efa.from.keys(), or esem.from.mods() requires a cache directory to save model objects to or to check against. By default, cache.setup uses the system's default cache location, otherwise a location within the current working directory or project can be selected. The function needs to be run once after the environment is cleared whenever a cache directory is required.

Usage

```
cache.setup(location = "user", interactive = TRUE)
```

Arguments

location	The cache location to save model objects to to enable checking. If location = "user" (default, recommended if unsure), then the system's default cache location is used. If any other string is used a folder will be created within either the getwd() directory or, if available, within the directory identified by here::here().
interactive	Logical. TRUE indicates that confirmation will be required before a cache directory is set *if* the default location is not used. FALSE indicates that a cache directory will be set without confirmation. Irrelevant if location = "user". It is recommended to use TRUE. Defaults to TRUE.

Details

Saving outputs to save time running identical models more than once requires files to be saved on your computer. To avoid writing to your computer without your permission, you are required to run this function first to ensure that you know *that* files are being saved and *where* files are being saved. The function temporarily sets a hidden environment variable '.cache_env' (with, if empty, assign(".cache_env", new.env(parent = emptyenv()), envir = parent.frame(1)) and with assign("cache_dir", cache_dir, envir = get(".cache_env", envir = parent.frame(1))), if not), which will be removed whenever the environment is cleared.

By default, the function creates the cache directory in the standard place for the operating system being used, appended by semFromKeys/[projectname], if a project is being used and the rstudioapi is available. If a project is not being used or the rstudioapi package is not available, then the relative path with be semFromKeys.

In the later case, if two function calls include the same name assignment, then outputs from one will overwrite the other. Therefore, it is recommended to either set the cache directory to something

other than the default or use the default within a project with the `rstudioapi` package installed. Obviously this latter option will likely not work outside of RStudio, so, in that case, it is recommended to use a custom location.

To ensure that R knows what the cache directory is, a hidden environment variable containing the information—`.cache_env`—is saved within the working environment. Other functions from the package will look for and use `.cache_env` to identify the cache directory. As a result, whenever the working environment is cleared (e.g., upon closing RStudio) or whenever the `.cache_env` is otherwise removed, the `cache.setup` function will need to be re-run before the cache functionality will work, regardless of the status of the any recently used cache directories.

Note also that the function relies on code that was unavailable in versions of R prior to version 4.0, so anyone using an earlier version of R will either have to update R or forgo the caching functionality.

Functions that directly or indirectly might require a cache directory are: `cfa.from.keys()`, `bifactor.from.keys()`, `efa.from.keys()`, `esem.from.mods()`, and `sem.check()`.

Value

The cache directory path (invisibly). Primarily called to set up the cache configuration.

See Also

`cfa.from.keys()`, `bifactor.from.keys()`, `efa.from.keys()`, `esem.from.mods()`, and `sem.check()` for dependent functions; `tools::R_user_dir()` for default cache directories; `here::here()` for project-based cache directory setting; and `cache.clean()` for a function to clean cache.

Examples

```
# Setup a cache directory
cache.setup()
# Now code with check = TRUE or save_out = TRUE will work, e.g.,
# Create CFA keys
keys0 <- c("grit_c", "grit_p", "hope_a", "hope_p")
keys <- sapply(
  keys0, function(x) names(BFIGritHope)[grep(x, names(BFIGritHope))]
)
# Run models
cfa_fit <- cfa.from.keys(
  keys, BFIGritHope, fit_save = TRUE, check = TRUE, save_out = TRUE
)
# Check that models are not run again.
cfa_fit <- cfa.from.keys(
  keys, BFIGritHope, fit_save = TRUE, check = TRUE, save_out = TRUE
)
```

`cfa.from.keys` *Runs CFA models for multiple scales based on items in a keys list.*

Description

`cfa.from.keys` runs a confirmatory factor analysis (CFA) model for each element of a keys list. The keys list must be a named list of scales, where each element is an item from the corresponding scale. The function is designed to streamline running CFA models for all scales in a sample and to input model outputs into downstream functions.

Usage

```
cfa.from.keys(
  keys,
  data,
  fit_save = TRUE,
  fit_measures = "all",
  std.lv = TRUE,
  miss = "ML",
  est = "default",
  name = "cfa",
  check = FALSE,
  save_out = FALSE
)
```

Arguments

`keys`	A named list of keys. Names should be scale names, elements should a list of items included in each scale.
`data`	A dataframe or object coercible to a dataframe. Data must include all observed variables in any of the keys.
`fit_save`	Logical. `TRUE` indicates model fit measures should be included in the output; `FALSE` indicates model fit measures should not be included in the output. `FALSE` may be desirable when fit measures are of little interest and when they may take a long time to estimate. Fit can still be examined for individual models with, `lavaan::fitMeasures(fit$fit$[model name])`.
`fit_measures`	A vector of fit measures to save or 'all' to select all fit measures, as per the `fit.measures` parameter from lavaan's `lavaan::fitMeasures()` function. Defaults to 'all'. Irrelevant if `fit_save = FALSE`.
`std.lv`	Logical. Sets the `std.lv` parameter, as per lavaan (see `lavaan::lavOptions()`). `TRUE` indicates that factor variances should be fixed to 1. `FALSE` indicates that loadings of the first items of factors should be fixed to 1. Defaults to `TRUE`.
`miss`	A string. Sets the `missing` parameter, as per lavaan (see `lavaan::lavOptions()`). Defaults to 'ML'.

est	A string. Sets the estimator parameter, as per lavaan (see lavaan::lavOptions()). The default ('default') uses the lavaan default for the model being run.
name	A string indicating a subdirectory where model outputs will be saved when save_out = TRUE and checked against when check = TRUE. Defaults to 'cfa'. Irrelevant if both save_out = FALSE and check = FALSE. The name should be unique for each set of models or outputs from calls with the same name will be overwritten.
check	Logical. TRUE indicates that current inputs should be compared to previous inputs if they exist and that the model should not be rerun if nothing has changed; FALSE indicates that these checks should not be made and the model should be run regardless of the existence of previous inputs.
save_out	Logical. TRUE indicates that model code, a hash of the data, important input parameter values, and output will be saved; FALSE indicates that nothing will be saved. Selecting save_out = TRUE enables the function to not rerun models next time if check = TRUE the next time the code is run and nothing has changed in the meantime.

Details

The model relies on sem.check() for the back-end of running the models. This enables saving inputs and outputs from model runs (with save_out = TRUE) and checking to see if anything has changed from prior runs before running again (with check = TRUE). The functionality was included for a number of very slow models or a lot of faster models, such that time spent rerunning them would be onerous. For further details on how this works, see the sem.check() function documentation.

The function does not provide any warnings for poor fit beyond those provided by lavaan. If any CFA models have poor fit, there is currently no capability to update them beyond removing items by omitting them from the keys. Any other changes to CFA models have to be made manually currently. (Note that with save_out = TRUE, model code is saved in file.path(out_dir, name, paste0(name, _mod.rds), which could help with manually updating models.)

Value

Returns a list of length 2 (if fit_save = FALSE) or 3 (if fit_save = TRUE). The elements of the list are: a list of lavaan model output objects; a list of parameter estimates from the models (standardized if std = TRUE); and, if fit_save = TRUE, a matrix of fit measures for each model.

See Also

sem.check(), which cfa.from.keys() uses for all the back-end, and lavaan::sem(), which is used to estimate the models.

Examples

```
# Create CFA keys
keys0 <- c("grit_c", "grit_p", "hope_a", "hope_p")
keys <- sapply(
  keys0, function(x) names(BFIGritHope)[grep(x, names(BFIGritHope))]
)
```

```
# Run models
cfa_fit <- cfa.from.keys(keys, BFIGritHope, check = FALSE, fit_save = TRUE)
# Examine some results
summary(cfa_fit$fit$grit_c)                # Standard lavaan summary
cfa_fit$fit_measures[, c("cfi", "rmsea")]  # Fit measures
```

`efa.from.keys` *Runs an EFA model based on items in a keys list.*

Description

`efa.from.keys` runs a exploratory factor analysis (EFA) in lavaan with a rotation targeted based on a keys list.

Usage

```
efa.from.keys(
  keys,
  data,
  orthogonal = FALSE,
  fit_save = TRUE,
  fit_measures = "all",
  std.lv = TRUE,
  miss = "ML",
  est = "default",
  name = "efa",
  check = FALSE,
  save_out = FALSE
)
```

Arguments

`keys`	A named list of keys. Names must be factor names, elements must be vectors of items that should be targeted to load on the factor.
`data`	A dataframe or object coercible to a dataframe. Data must include all observed variables in any of the keys.
`orthogonal`	Logical. Sets the `orthogonal` parameter, as per lavaan (see `lavaan::lavOptions()`). `TRUE` indicates that unspecified latent variable correlations should be fixed at 0; `FALSE` indicates that unspecified latent variable correlations should be freely estimated. Defaults to `FALSE`.
`fit_save`	Logical. `TRUE` indicates model fit measures should be included in the output; `FALSE` indicates model fit measures should not be included in the output. `FALSE` may be desirable when fit measures are of little interest and when they may take a long time to estimate. Fit can still be examined for individual models with, `lavaan::fitMeasures(fit$fit$[model name])`.

`fit_measures` A vector of fit measures to save or 'all' to select all fit measures, as per the `fit.measures` parameter from lavaan's `lavaan::fitMeasures()` function. Defaults to 'all'. Irrelevant if `fit_save = FALSE`.

`std.lv` Sets the `std.lv` param, as per lavaan (see `lavaan::lavOptions()`). Defaults to `TRUE`.

`miss` A string. Sets the `missing` parameter, as per lavaan (see `lavaan::lavOptions()`). Defaults to 'ML'.

`est` A string. Sets the `estimator` parameter, as per lavaan (see `lavaan::lavOptions()`). The default ('default') uses the lavaan default for the model being run.

`name` A string indicating a subdirectory where model outputs will be saved when `save_out = TRUE` and checked against when `check = TRUE`. Defaults to 'efa'. Irrelevant if both `save_out = FALSE` and `check = FALSE`. The name should be unique for each set of models or outputs from calls with the same name will be overwritten.

`check` Logical. `TRUE` indicates that current inputs should be compared to previous inputs if they exist and that the model should not be rerun if nothing has changed; `FALSE` indicates that these checks should not be made and the model should be run regardless of the existence of previous inputs.

`save_out` Logical. `TRUE` indicates that model code, a hash of the data, important input parameter values, and output will be saved; `FALSE` indicates that nothing will be saved. Selecting `save_out = TRUE` enables the function to not rerun models next time if `check = TRUE` the next time the code is run and nothing has changed in the meantime.

Details

The function was designed to streamline running exploratory structural equation models (ESEM) using Burt's (1976) 2-stage procedure to prevent interpretational confounding in the context of ESEM. However, it can also be used to easily run a targeted EFA with only a keys list to avoid having to manually specify the target and model.

The function was designed for use with established multidimensional scales, such that a target is always reasonable. The function does not currently support untargeted rotations.

The model relies on `sem.check()` for the back-end of running the models. This enables saving inputs and outputs from model runs (with `save_out = TRUE`) and checking to see if anything has changed from prior runs before running again (with `check = TRUE`). The functionality was included for a number of very slow models or a lot of faster models, such that time spent rerunning them would be onerous. For further details on how this works, see the `sem.check()` function documentation.

Value

Returns a list of lists. The elements are a list of lavaan bifactor model output objects; a list of parameter estimates from the models (standardized if `std = TRUE`); and, if `fit_save = TRUE`, a matrix of fit measures for each model.

References

Burt, R. S. (1976). Interpretational confounding of unobserved variables in Structural Equation Models. Sociological Methods & Research, 5(1), 3-52. https://doi.org/10.1177/004912417600500101.

See Also

`sem.check()`, which `efa.from.keys()` uses for all the back-end, and `lavaan::sem()`, which is used to estimate the models.

Examples

```
# Create EFA keys
# Using only 3 factors to save time
keys_e0 <- paste0("bfi_", c("e", "a", "c"))
# Using less than all items to save time
# (This results in a less than ideal solution but it shouldn't matter for an
# example)
keys_e <- sapply(
  keys_e0,
  function(x) {
    names(BFIGritHope)[grep(paste0(x, "\\d_[1-2]"), names(BFIGritHope))]
  },
  simplify = FALSE
)
# Run model
efa_fit <- efa.from.keys(keys_e, BFIGritHope, check = FALSE, fit_save = TRUE)
# Examine results
summary(efa_fit$fit$efa)                   # Standard lavaan summary
efa_fit$fit_measures[, c("cfi", "rmsea")]  # Fit measures
```

`esem.from.mods` *Runs ESEM based on CFA and EFA model outputs.*

Description

`esem.from.keys` runs exploratory structural equation models (ESEM) in lavaan where the exploratory factor analysis (EFA) factors predict confirmatory factor analysis (CFA) factors and/or bifactor factors in separate models for each CFA or bifactor model. The function takes a fitted lavaan object from an EFA and lists of fitted CFA and/or bifactor lavaan model objects as inputs so will typically use outputs from `efa.from.keys()`, and `cfa.from.keys()` or `bifactor.from.keys()`.

Usage

```
esem.from.mods(
  efa_fit,
  cfa_fit = NULL,
  bif_fit = NULL,
  data,
```

```
  fit_save = FALSE,
  fit_measures = "all",
  miss = "ML",
  est = "default",
  name = "esem",
  check = FALSE,
  save_out = FALSE
)
```

Arguments

efa_fit	A fitted lavaan object of an EFA model.
cfa_fit	A named list of fitted lavaan objects of CFA models. Can be NULL if bif_fit is not NULL.
bif_fit	A named list of fitted lavaan objects of bifactor models. Can be NULL if cfa_fit is not NULL.
data	A dataframe or object coercible to a dataframe. Data must include all observed variables used in any of the models.
fit_save	Logical. TRUE indicates model fit measures should be included in the output; FALSE indicates model fit measures should not be included in the output. FALSE may be desirable when fit measures are of little interest and when they may take a long time to estimate. Fit can still be examined for individual models with, lavaan::fitMeasures(fitfit[model name]).
fit_measures	A vector of fit measures to save or 'all' to select all fit measures, as per the fit.measures parameter from lavaan's lavaan::fitMeasures() function. Defaults to 'all'. Irrelevant if fit_save = FALSE.
miss	A string. Sets the missing parameter, as per lavaan (see lavaan::lavOptions()). Defaults to 'ML'.
est	A string. Sets the estimator parameter, as per lavaan (see lavaan::lavOptions()). The default ('default') uses the lavaan default for the model being run.
name	A string indicating a subdirectory where model outputs will be saved when save_out = TRUE and checked against when check = TRUE. Defaults to "esem". Irrelevant if both save_out = FALSE and check = FALSE. The name should be unique for each set of models or outputs from calls with the same name will be overwritten.
check	Logical. TRUE indicates that current inputs should be compared to previous inputs if they exist and that the model should not be rerun if nothing has changed; FALSE indicates that these checks should not be made and the model should be run regardless of the existence of previous inputs.
save_out	Logical. TRUE indicates that model code, a hash of the data, important input parameter values, and output will be saved; FALSE indicates that nothing will be saved. Selecting save_out = TRUE enables the function to not rerun models next time if check = TRUE the next time the code is run and nothing has changed in the meantime.

Details

The function was designed to streamline running exploratory structural equation models (ESEM) where EFA factors predict a series of latent variables in separate models using Burt's (1976) 2-stage procedure to prevent interpretational confounding. The function is designed to run analyses equivalent to that of Bainbridge, Ludeke, and Smillie (2022).

The function requires fitted lavaan objects as inputs in order to properly employ the 2-stage procedure. Using `efa.from.keys()`, `cfa.from.keys()`, and/or `bifactor.from.keys()` should make this relatively straight-forward.

Matching the philosophy of the package, the function is designed to run for multiple models with a similar design. If you are using the function for a single model, transform inputs into lists as appropriate.

The model relies on `sem.check()` for the back-end of running the models. This enables saving inputs and outputs from model runs (with `save_out = TRUE`) and checking to see if anything has changed from prior runs before running again (with `check = TRUE`). The functionality was included for a number of very slow models or a lot of faster models, such that time spent rerunning them would be onerous. For further details on how this works, see the `sem.check()` function documentation.

Although the function will take any lavaan models that have successfully produced standard lavaan outputs, it will not complain if these have poor fit or other undesirable characteristics (beyond warnings and errors produced by lavaan). If this matters to you, it would be worth checking the outputs of upstream functions prior to running `esem.from.mods()`.

The 2-stage procedure employed mitigates some of the problems of poor fitting measurement models but, depending on your purpose and research question(s), it could nevertheless be important to update models to ensure good fit prior to running this function. There is currently no capability within `semFromKeys` to add modifications to CFA or bifactor models (beyond removing items by removing them from the keys).

For bifactor models predicted by ESEM factors, a complication is that fixing orthogonal relationships between general and group factors is not possible if any of the factors are regressed on any others. This is because, in SEM, fixing correlations with a factor that is an outcome fixes correlations with the residual, which is, of course, not the requirement of a bifactor model.

There are three solutions to this problem. First, avoid using bifactor models. This may be possible sometimes, but it frequently is not. Second, avoid using regressions in models with bifactor models and compute regressions based on latent variable correlations. (This can be done by, e.g., creating a correlation matrix from the factor correlations and using the `psych::setCor()` function). This method produces accurate point estimates in the regressions, but standard errors are biased as the method cannot account for uncertainty in the correlations. This is an adequate solution if p-values and confidence intervals are not required. Finally, if the 2-stage procedure is employed, then there is little room for measurement models to change to allow factor correlations to change. Therefore, the parameter can be relaxed and implied correlations between the group and general factors ought to remain close to zero. This is the method employed by `esem.from.mods()`.

Value

Returns a list of length 4 (if `fit_save = FALSE`) or 5 (if `fit_save = TRUE`). The elements of the list are: a list of lavaan model output objects; a list of parameter estimates from the models (standard-

ized if `std = TRUE`); if `fit_save = TRUE`, a matrix of fit measures for each model; a list of regression beta parameters from each model; and a dataframe of R-squared values from each model.

References

Bainbridge, T. F., Ludeke, S. G., & Smillie, L. D. (2022). Evaluating the Big Five as an organizing framework for commonly used psychological trait scales. Journal of Personality and Social Psychology, 122(4), 749-777. https://doi.org/10.1037/pspp0000395.

Burt, R. S. (1976). Interpretational confounding of unobserved variables in Structural Equation Models. Sociological Methods & Research, 5(1), 3-52. https://doi.org/10.1177/004912417600500101.

Eid, M., Geiser, C., Koch, T., & Heene, M. (2017). Anomalous results in G-factor models: Explanations and alternatives. Psychological Methods, 22(3), 541-562. https://doi.org/10.1037/met0000083.

See Also

`sem.check()`, which this function uses for all the back-end; `cfa.from.keys()`, `efa.from.keys()`, and `bifactor.from.keys()`, which are useful functions for creating inputs into `esem.from.mods()`; and `lavaan::sem()`, which is used to estimate the models.

Examples

```
# Create CFA keys
keys0 <- c("grit_c", "grit_p", "hope_a", "hope_p")
keys <- sapply(
  keys0, function(x) names(BFIGritHope)[grep(x, names(BFIGritHope))]
)
# Create bifactor keys
keys_g0 <- c("grit", "hope")
keys_g <- sapply(
  keys_g0, function(x) names(BFIGritHope)[grep(x, names(BFIGritHope))]
)
keys_b1 <- sapply(
  keys_g0, function(x) keys0[grep(x, keys0)], simplify = FALSE
)
# Avoid fit problems with an S-1 model (Eid et al., 2017)
keys_b <- keys_b1
keys_b$hope <- keys_b1$hope[-1]
# Create EFA keys
# Using only 3 factors and fewer items to save time for a simple example
# (This results in a less than ideal solution but it doesn't matter for an
# example)
keys_e0 <- paste0("bfi_", c("e", "a", "c"))
keys_e <- sapply(
  keys_e0,
  function(x) {
    names(BFIGritHope)[grep(paste0(x, "\\d_[1-2]"), names(BFIGritHope))]
  },
  simplify = FALSE
)
# Create fitted objects to use as inputs
cfa_fit <- cfa.from.keys(keys, BFIGritHope, fit_save = FALSE)
```

```
efa_fit <- efa.from.keys(keys_e, BFIGritHope, fit_save = FALSE)
bif_fit <- bifactor.from.keys(
  keys_g, keys_b, keys, BFIGritHope, fit_save = FALSE
)
# Run models
esem_fit <- esem.from.mods(
  efa_fit$fit$efa, cfa_fit$fit, bif_fit$fit, data = BFIGritHope,
  fit_save = FALSE, check = FALSE
)
# Examine results
summary(esem_fit$fit$grit_c)  # Standard lavaan summary
esem_fit$r2                   # R-squareds
esem_fit$b                    # Betas
```

---

sem.check *Runs lavaan models after checking for code or data changes.*

Description

sem.check produces outputs from a series of lavaan models. For each model, the code checks for previously saved model code, a hash of data, and important parameter inputs. If they exist and match values for the current code and data, the model is not run, the previous output is loaded instead. If either the code has changed, the hash of the data has changed, or important parameter inputs have change, or any of these do not exist, then the models are run as normal.

Usage

```
sem.check(
  mods,
  data,
  keys_s = NULL,
  keys_e = NULL,
  fit_save = FALSE,
  fit_measures = "all",
  miss = "ML",
  est = "default",
  std.lv = FALSE,
  std = TRUE,
  orthogonal = FALSE,
  target = NULL,
  name = "sem",
  check = FALSE,
  save_out = FALSE
)
```

Arguments

mods	A named list of lavaan models to run.

data	A dataframe or object coercible to a dataframe. Data must include all observed variables used in any of the models.
keys_s	A named keys list matching the names and length of mod.
keys_e	A named keys list of the factors in an ESEM to be included.
fit_save	Logical. TRUE indicates model fit measures should be included in the output; FALSE indicates model fit measures should not be included in the output. FALSE may be desirable when fit measures are of little interest and when they may take a long time to estimate. Fit can still be examined for individual models with, lavaan::fitMeasures(fitfit[model name]).
fit_measures	A vector of fit measures to save or 'all' to select all fit measures, as per the fit.measures parameter from lavaan's lavaan::fitMeasures() function. Defaults to 'all'. Irrelevant if fit_save = FALSE.
miss	A string. Sets the missing parameter, as per lavaan (see lavaan::lavOptions()). Defaults to 'ML'.
est	A string. Sets the estimator parameter, as per lavaan (see lavaan::lavOptions()). The default ('default') uses the lavaan default for the model being run.
std.lv	Logical. Sets the std.lv parameter, as per lavaan (see lavaan::lavOptions()). TRUE indicates that factor variances should be fixed to 1. FALSE indicates that loadings of the first items of factors should be fixed to 1. Defaults to FALSE.
std	Logical. TRUE indicates that standardised parameter estimates should be saved; FALSE indicates that unstandardised parameters estimates should be saved.
orthogonal	Logical. Sets the orthogonal parameter, as per lavaan (see lavaan::lavOptions()). TRUE indicates that unspecified latent variable correlations should be fixed at 0; FALSE indicates that unspecified latent variable correlations should be freely estimated. Defaults to FALSE.
target	A matrix indicating a rotation target, as used in the rotation.args argument in lavaan (see lavaan::efa(). If NULL (default), target is not specified and lavaan uses default behaviour. Irrelevant when the model does not include an EFA or ESEM.
name	A string indicating a subdirectory where model outputs will be saved when save_out = TRUE and checked against when check = TRUE. Defaults to "sem". The name should be unique for each set of models or outputs from calls with the same name will be overwritten.
check	Logical. TRUE indicates that current inputs should be compared to previous inputs if they exist and that the model should not be rerun if nothing has changed; FALSE indicates that these checks should not be made and the model should be run regardless of the existence of previous inputs.
save_out	Logical. TRUE indicates that model code, a hash of the data, important input parameter values, and output will be saved; FALSE indicates that nothing will be saved. Selecting save_out = TRUE enables the function to not rerun models next time if check = TRUE the next time the code is run and nothing has changed in the meantime.

Details

The function is largely intended to be used as a helper function to upstream functions, including `cfa.from.keys()`, `bifactor.from.keys()`, `efa.from.keys()`, and `esem.from.mods()`. Although it is recommended to use the appropriate upstream function whenever possible, there are not (currently) options to do so when customised lavaan models are required; for example, when allowing two items' residuals to correlate in a CFA. `sem.check()` can be used in these cases (see example).

Matching the philosophy of the package, the function is designed to run for multiple models with a similar design. If you are using the function for a single model, transform inputs into lists as appropriate.

When `save_out = TRUE`, if a cache directory has been set, the model will save various inputs and outputs from the function call. When `check = TRUE`, the model will look for any previously saved outputs from earlier model runs in the same cache directory and only run again if nothing has changed. In cases where something has changed for a subset of models (e.g., a data cleaning mistake might affect only a subset of variables in a subset of models), then the function will only re-run the models where something has changed. Changes to arguments that influence all lavaan model runs (e.g., miss or est) will trigger all models to be re-run.

For either `save_out = TRUE` or `check = TRUE`, the function will look for a cache directory set and created by the `cache.setup()` function. If a cache directory has not been set for the current session, then the function will exit with an error suggesting that either `cache.setup()` be run or `save_out` and `check` set to FALSE.

When the cache directory is found and output from previous runs are detected, the comparisons performed are for: model code; hashes of the data (using `openssl::md5()`); values of the `miss`, `est`, `std`, `std.lv`, and `orthogonal`parameters; the class of model objects (i.e., class lavaan); and the class of parameter estimates (i.e., class lavaan.data.frame).

For most applications, the checking feature can be safely ignored by not saving outputs (i.e., `fit_save = FALSE`) and not checking for past saves (i.e., `check = FALSE`, both default). The functionality is intended for a number of very slow models or a very large number of faster models, such that time spent rerunning code would be onerous. However, the functionality can be safely used for faster runs too.

Value

Returns a list of length 2 (if `fit_save = FALSE`) or 3 (if `fit_save = TRUE`). The elements of the list are: a list of lavaan model output objects; a list of parameter estimates from the models (standardized if `std = TRUE`); and, if `fit_save = TRUE`, a matrix of fit measures for each model.

See Also

`cfa.from.keys()`, `efa.from.keys()`, `bifactor.from.keys()`, and `esem.from.mods()`—all these function depend upon `sem.check()` to work; `lavaan::sem()`, which is used to estimate the models; `lavaan::parameterEstimates()`, which is used to estimate parameter values; and `lavaan::fitMeasures()`, which is used to estimate fit statistics when `fit_save = TRUE`.

Examples

```
# Create CFA keys
```

```
keys0 <- c("grit_c", "grit_p", "hope_a", "hope_p")
keys <- sapply(
  keys0, function(x) names(BFIGritHope)[grep(x, names(BFIGritHope))]
)
# Create model code
mods <- mapply(
  x = keys, y = names(keys), SIMPLIFY = FALSE,
  FUN = function(x, y) paste(y, "=~", paste(x, collapse = " + "))
)
# Edit model code to add correlated residuals
mods[[1]] <- paste0(mods[[1]], "\ngrit_c_1 ~~ grit_c_2")
# Estimate the models with sem.check()
cfa_fit <- sem.check(mods, BFIGritHope, keys, name = "cfa", check = FALSE)
```

Index