Package 'semFromKeys'

September 14, 2026

Type Package

Title Run 'lavaan' Models from Keys Lists

Version 0.5.5

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 8.1.0

Depends R (>= 3.6.0)

Imports stringr, lavaan, openssl, withr, tools, matrixcalc, Matrix, stats, stringr

Suggests testthat (>= 3.0.0), here

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 2026-09-14 02:50:02 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.

Usage

```
bifactor.from.keys(
  keys_g,
  keys_b,
  keys,
  data,
```

```
  fit_save = TRUE,
  fit_measures = "all",
  std.lv = TRUE,
  miss = "default",
  est = "default",
  ordered = NULL,
  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. 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 'default', which uses 'ML' for ordered = NULL and 'pairwise' for any other value of ordered.
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.
ordered	A character vector or NULL, as per the ordered lavaan argument of the same name (see, e.g., lavaan::sem). NULL indicates that all variables should be treated as continuous. Otherwise, variables matching an element of the vector will be treated as continuous. Defaults to NULL.
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 function is designed to streamline running measurement models for all scales in a sample and to input model outputs into downstream functions. Keys list must be named appropriately; that is, `keys_g` and `keys_b` names must be the general factor names, and `keys` must be the group factor names.

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, lavaan::sem

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, name = 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 or without confirmation if interactive = FALSE. At least one of older_than and name must be set.
name	A character indicating a model name. Files associated with a model of that name will be deleted after confirmation if interactive = TRUE or without confirmation if interactive = FALSE. At least one of older_than and name must be set.
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 interactive = TRUE, so it will ask for confirmation before deleting anything. An error will occur if attempting to run in a non-interactive session without setting interactive = FALSE. In addition to deleting obsolete 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.

If both older_than and name are specified then both have to be matched for files to be selected for deletion.

name finds files using grep. This means that model names with '.' will match anything where the '.' is in the string. Thus, a.1 will match a_1, a11, ab1, etc. This can also be used to easily match all files in the cache directory (with creation dates older than older_than if specified) by specifying name = '.*', which indicates anything ('.') matched any number of times ('*'). However, the function also includes code to only match outputs created by semFromKeys, so if files have been added to the cache directory that do not match semFromKeys outputs, these will not be selected for deletion, even if name = '.*'.

semFromKeys has no way to know what directories you have specified as the cache in the past and cannot clean up unknown former cache directories. Therefore, it is recommended that you use either the default location or the same location for all models within the same project. Either of these will enable easy detection, so that unneeded cached files can be found and deleted.

Value

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

See Also

cache.setup

Examples

```
## Not run:
  # 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
  )

  # The cache.clean lines are commented out so you do not inadvertently
  # delete any models created with your own code in the same cache directory.

  # cache.clean(30)  # Delete files not modified in the last 30 days.
  # cache.clean(60/86400)  # Delete files not modified in the last minute.
  # Delete all files matching those created by the package in the current
  # cache directory and subdirectories without confirmation.
  # cache.clean(0, interactive = FALSE)

## End(Not run)
```

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

Description

Using `save_out = TRUE` or `check = TRUE` from various `semFromKeys` functions 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", projectname = NULL, 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.
`projectname`	The project name to use or `NULL` for auto-detection. Only relevant if `location = "user"`.
`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"`. 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', 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]`. By default, `projectname` will be auto-detected using the last directory from here::here, if available, or getwd, if not. Alternatively, `projectname` can be set manually by specifying with the `projectname` argument.

To use a location within the current working directory structure, the desired location should be specified with the `location` argument. This will create a directory matching `location` within the current directory structure with here::here, if available, or getwd, if not. It is a relative path, so should not include directories above the current working directory. If `interactive = TRUE` in an interactive session, you will be asked to confirm the cache location in this case.

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 any recently used cache directories.

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.

`semFromKeys` has no way to know what directories you have specified as the cache in the past and cannot clean up unknown former cache directories. Therefore, it is recommended that you use either the default location or the same location for all models within the same project. Either of these will enable easy detection, so that unneeded cached files can be found and deleted.

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

Value

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

See Also

cache.clean, cfa.from.keys, bifactor.from.keys, efa.from.keys, esem.from.mods, sem.check, tools::R_user_dir, here::here, getwd

Examples

```
## Not run:
  # 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
  )

## End(Not run)
```

`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 = "default",
  est = "default",
  ordered = NULL,
  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(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 'default', which uses 'ML' for ordered = NULL and 'pairwise' for any other value of ordered.
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.
ordered	A character vector or NULL, as per the ordered lavaan argument of the same name (see, e.g., lavaan::sem). NULL indicates that all variables should be treated as continuous. Otherwise, variables matching an element of the vector will be treated as continuous. Defaults to NULL.
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, lavaan::sem

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 = "default",
  est = "default",
  ordered = NULL,
  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(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	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 'default', which uses 'ML' for ordered = NULL and 'pairwise' for any other value of ordered.
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.
ordered	A character vector or NULL, as per the ordered lavaan argument of the same name (see, e.g., lavaan::sem). NULL indicates that all variables should be treated as continuous. Otherwise, variables matching an element of the vector will be treated as continuous. Defaults to NULL.
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, lavaan::sem

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 with selected fit measures.
efa_fit <- efa.from.keys(
  keys_e, BFIGritHope, check = FALSE,
  fit_save = TRUE, fit_measures = c("chisq", "df", "pvalue")
)
# Examine results
summary(efa_fit$fit)    # Standard lavaan summary
efa_fit$fit_measures    # Selected fit measures
```

`esem.from.keys` *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 in separate models for each CFA model. The function takes a keys list to describe the EFA factors and keys lists to describe the CFA models.

Usage

```
esem.from.keys(
  data,
  keys_e,
  keys,
  fit_save = TRUE,
  fit_measures = "all",
  miss = "default",
  est = "default",
  name = "esam",
  check = FALSE,
  save_out = FALSE
)
```

Arguments

`data`	A dataframe or object coercible to a dataframe. Data must include all observed variables in keys.
`keys_e`	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.

keys — A named list of items in uni-dimensional factors. Names must be the names of the factors. List element must be a vector of items that load on the factors. For bi-factor models, these should be group factor names and items.

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 'default', which uses 'ML' for ordered = NULL and 'pairwise' for any other value of ordered.

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 "esam". 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 Rosseel and Loh's (2022) SAM method to prevent interpretational confounding (see below). The function is designed to run analyses analogous to that of Bainbridge, Ludeke, and Smillie (2022) only using the SAM method instead of Burt's 2-stage method.

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.

In SEM, standard methods do not distinguish between measurement and structural parameters. As a result, measurement model parameters can change with the addition of theoretically unrelated constructs in a structural model, and can change differently for different sets of unrelated constructs. This means that the unrelated constructs are changing the interpretation of the latent variable, which Burt (1976) referred to as "interpretational confounding".

There are various ways to deal with interpretational confounding. The standard solution (other than ignoring it) is to create good fitting measurement models first, then freely estimate the structural model with checks to ensure adequate fit of the model and that measurement parameters do not substantially change with different combinations of factors. This is sometimes a good solution, but, in other cases, it is not. For example, if the measurement model was for a well-established scale and it requires changing, then it loses easy comparison with past research. This issue is most clearly relevant when changes to a measurement model require entirely different factors, or items to be removed but it is still an issue for less dramatic changes. When a single scale is being assessed, these issues can be resolved by suggesting a thorough evaluation of the scale and, perhaps, the suggestion of a new measurement model or a new scale for a particular population; however, when many scales are being assessed this solution is impractical, and may not solve the interpretational confounding issue regardless.

An alternative solution, proposed by Burt (1976) is to fix measurement model parameters in a model estimating structural parameters. This method means that misspecification of one measurement model cannot affect other measurement models and that the interpretation of measured constructs cannot change based on unrelated factors. However, it is not a perfect solution because, by fixing measurement parameters, uncertainty in their estimation is neglected (e.g., Nagy et al., 2017), which results in biased standard errors and fit statistics.

A third option was proposed by Nagy and colleagues (2017), who introduced an extension procedure such that item residuals are allowed to correlate with external variables (or factors). To make the model identifiable, these relationships are constrained using one of a number of methods. If the sums of squares of correlations between all combinations of factors' items and external factors are minimised, measurement parameters in isolated measurement models are preserved in the structural model without having to constrain them directly. As a result, unbiased standard errors are preserved while simultaneously eliminating interpretational confounding since the measurement parameters from the measurement models are preserved regardless of external factors. Unfortunately, estimating these models becomes increasingly slow with more items and factors, such that it quickly becomes untenable. Moreover, the method only works with correlations, not regressions, so some method to run regressions using the correlations needs to be implemented that does not itself result in biased estimates due to ignored uncertainty in the correlation estimates.

Although this latter issue may be solvable for Nagy and colleagues' (2017) method, a more practical solution to these issues was proposed by Rosseel and Loh (2022) with their SAM approach. This method essentially follows Burt's (1976) method but adjust the procedure to overcome its issues. They distinguish two SAM varieties–"local SAM" and "global SAM". Local SAM uses the observed summary statistics of the parameters of the measurement models to generate mean and covariance matrices to use in the structural model, which preserves the structure of the measurement models while also preserving the uncertainty. Global SAM treats the measurement parameters as given, but corrects the standard errors of the structural model. Although local SAM is preferable in most circumstances, it currently (as at version 0.7-2) sets ESEM factor covariances as equal, which is likely a bug.

Given its practicality and the absence of the aforementioned bug, `esem.from.keys` uses the global SAM method.

Note that bifactor models are not currently supported in `esem.from.keys`. There is currently (as at version 0.7-2) no way to force lavaan to freely estimate covariances with factors that are not involved in a structural path. Instead the lavaan::sam function treats them as 0, which alters the model from what it ought to be. This may also be a bug in the lavaan::sam function.

In the meantime, I suggest using esem.from.mods for bifactor models. Standard errors and fit statistics are not quite right with that method but they are likely close enough for most contexts and there are not any better options that are easily implemented.

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 (standardized 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.

Nagy, G., Brunner, M., Lüdtke, O., and Greiff, S. (2017). Extension Procedures for Confirmatory Factor Analysis. Journal of Experimental Education, 85(4). https://doi.org/10.1080/00220973.2016.1260524.

Rosseel, Y. & Loh, W. W. (2022). A structural after measurement approach to structural equation modeling. Psychological Methods, 29(3), 561-588. https://doi.org/10.1037/met0000503.

See Also

sem.check, lavaan::sam

Examples

```
# Create CFA keys
keys0 <- c("hope_a", "hope_p")
keys <- sapply(
  keys0,
  function(x) names(BFIGritHope)[grep(x, names(BFIGritHope))],
  simplify = FALSE
)
# 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, "[1-2]_[1-2]"), names(BFIGritHope))]
```

```
  },
  simplify = FALSE
)
# Run models
esem_fit <- esem.from.keys(
  BFIGritHope, keys_e, keys, fit_save = FALSE
)
# Examine results
summary(esem_fit$fit$hope_a)  # Standard lavaan summary
esem_fit$r2                   # R-squareds
esem_fit$b                    # Betas
```

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

Description

`esem.from.mods` 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. *NOTE:* It is recommended to use esem.from.keys for CFA factors (see details).

Usage

```
esem.from.mods(
  data,
  efa_fit,
  cfa_fit = NULL,
  bif_fit = NULL,
  fit_save = FALSE,
  fit_measures = "all",
  miss = "default",
  est = "default",
  name = "esem",
  check = FALSE,
  save_out = FALSE
)
```

Arguments

`data`	A dataframe or object coercible to a dataframe. Data must include all observed variables used in any of the models.
`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`.

`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`.

`miss` A string. Sets the `missing` parameter, as per lavaan (see lavaan::lavOptions). Defaults to 'default', which uses 'ML' for `ordered = NULL` and 'pairwise' for any other value of `ordered`.

`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 (see below). 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. The function currently does not support ordinal variables, so inputs should be created with `ordered = NULL`.

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.

In SEM, standard methods do not distinguish between measurement and structural parameters. As a result, measurement model parameters can change with the addition of theoretically unrelated constructs in a structural model, and can change differently for different sets of unrelated constructs. This means that the unrelated constructs are changing the interpretation of the latent variable, which Burt (1976) referred to as "interpretational confounding".

There are various ways to deal with interpretational confounding. The standard solution (other than ignoring it) is to create good fitting measurement models first, then freely estimate the structural model with checks to ensure adequate fit of the model and that measurement parameters do not substantially change with different combinations of factors. This is sometimes a good solution, but, in other cases, it is not. For example, if the measurement model was for a well-established scale and it requires changing, then it loses easy comparison with past research. This issue is most clearly relevant when changes to a measurement model require entirely different factors, or items to be removed but it is still an issue for less dramatic changes. When a single scale is being assessed, these issues can be resolved by suggesting a thorough evaluation of the scale and, perhaps, the suggestion of a new measurement model or a new scale for a particular population; however, when many scales are being assessed this solution is impractical, and may not solve the interpretational confounding issue regardless.

An alternative solution, proposed by Burt (1976) is to fix measurement model parameters in a model estimating structural parameters. This method means that misspecification of one measurement model cannot affect other measurement models and that the interpretation of measured constructs cannot change based on unrelated factors. However, it is not a perfect solution because, by fixing measurement parameters, uncertainty in their estimation is neglected (e.g., Nagy et al., 2017), which results in biased standard errors and fit statistics.

A third option was proposed by Nagy and colleagues (2017), who introduced an extension procedure such that item residuals are allowed to correlate with external variables (or factors). To make the model identifiable, these relationships are constrained using one of a number of methods. If the sums of squares of correlations between all combinations of factors' items and external factors are minimised, measurement parameters in isolated measurement models are preserved in the structural model without having to constrain them directly. As a result, unbiased standard errors are preserved while simultaneously eliminating interpretational confounding since the measurement parameters from the measurement models are preserved regardless of external factors. Unfortunately, estimating these models becomes increasingly slow with more items and factors, such that it quickly becomes untenable. Moreover, the method only works with correlations, not regressions, so some method to run regressions using the correlations needs to be implemented that does not itself result in biased estimates due to ignored uncertainty in the correlation estimates.

A more practical solution than Nagy and colleague's method was proposed by Rosseel and Loh (2022) with their SAM approach. This method essentially follows Burt's (1976) method but adjust the procedure to overcome its issues. They distinguish two SAM varieties–"local SAM" and "global SAM". Local SAM uses the observed summary statistics of the parameters of the measurement models to generate mean and covariance matrices to use in the structural model, which preserves the structure of the measurement models while also preserving the uncertainty. Global SAM treats the measurement parameters as given, but corrects the standard errors of the structural model.

Despite the benefits of the SAM methods, they are not used in `esem.from.mods`. Largely this is a legacy issues; however, the function is currently maintained due to lavaan::sam currently treating all latent variables that are not in a regression path in the structural model as unrelated to the other factors. Given regressing the general factor of a bifactor model on EFA factors requires the group factors to correlate with the EFA factors, lavaan::sam is currently inappropriate for bifactor models

(as at lavaan version 0.7-2).

As a result of these considerations, the 2-stage procedure of fixing measurement parameters in the structural models remains appropriate for bifactor models. Note, however, that the solution to interpretational confounding means that standard errors and fit statistics will be biased, so should not be used to infer precise p-values, to determine an optimal model, nor that a model surpasses some cut-off of "good fit". Instead, the function is intended to allow variables to be regressed on EFA factors with much better measurement than would be achieved with aggregate scores.

A further complication occurs for bifactor models. Recall that bifactor models require orthogonal relationships between the general and group factors. When a factor is an outcome of a regression in a structural model, it is not possible to include such a constraint because the instructions normally used to do so will constrain residual variance instead. However, given the 2-stage procedure is employed, there is little room for measurement models to change to allow factor correlations to change. As a result, the parameter can be relaxed, and implied correlations between the group and general factors should remain close to zero. Given that the function already uses the 2-stage procedure, this method is employed when bifactor models are used in `esem.from.mods`.

Finally, `esem.from.mods` will not complain if measurement models have poor fit or other undesirable characteristics (beyond warnings and errors produced by lavaan). In cases where keeping the same measurement model as prior research is important, it may make sense to include a poor fitting measurement model in the ESEM. In cases where the measurement model might reasonably be adjusted, however, it is important to check measurement model fit before running `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 (standardized 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.

Nagy, G., Brunner, M., Lüdtke, O., and Greiff, S. (2017). Extension Procedures for Confirmatory Factor Analysis. Journal of Experimental Education, 85(4). https://doi.org/10.1080/00220973.2016.1260524.

See Also

sem.check, cfa.from.keys, efa.from.keys, bifactor.from.keys, lavaan::sem.

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 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)
# Run models
esem_fit <- esem.from.mods(
  efa_fit$fit, cfa_fit$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 takes a model list, keys lists, and data, and produces outputs from a series of lavaan models. The function is primarily used as a function to be called by other function within the package but it can also be run independently.

For each model, the code optionally 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 = "default",
  est = "default",
  std.lv = FALSE,
  std = TRUE,
  ordered = NULL,
  orthogonal = FALSE,
  target = NULL,
  name = "sem",
  check = FALSE,
  save_out = FALSE,
  use_sam = 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 'default', which uses 'ML' for ordered = NULL and 'pairwise' for any other value of ordered.
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.
ordered	A character vector or NULL, as per the ordered lavaan argument of the same name (see, e.g., lavaan::sem). NULL indicates that all variables should be treated as continuous. Otherwise, variables matching an element of the vector will be treated as continuous. Defaults to NULL.

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.

use_sam — Logical. TRUE indicates that the lavaan::sam function should be used for model estimation. FALSE indicates that the lavaan::sem function should be used for model estimation.

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.keys, among others. 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 (with the exception of the extra argument in sem.path; 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 or simply use lavaan without the assistance of semFromKeys.

The function includes functionality designed to save time re-running code when lots of slow models are included. To do this, when save_out = TRUE and a cache directory has been set, the model will save various inputs and outputs from the function call, and, 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, then the function will re-run the models where something has changed (but it will not for those where nothing 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 orthogonalparameters;
- 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), but may be beneficial in cases with lots of models or very slow models. 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, esem.from.mods, esem.from.keys, sem.cor, sem.path, cache.setup, lavaan::sem, lavaan::sam, lavaan::efa, lavaan::parameterEstimates, lavaan::standardizedSolution, lavaan::fitMeasures, openssl::md5, lavaan::lavOptions

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)
```

`sem.cor` *Creates a latent variable correlation matrix from fitted lavaan measurement models*

Description

`sem.cor` takes CFA outputs and produces a correlation matrix between latent variables.

Usage

```
sem.cor(
  data,
  fit_y,
  fit_x = NULL,
  items = NULL,
  item_loadings = NULL,
  nagy = TRUE,
  fit_save = FALSE,
  fit_measures = "all",
  miss = "default",
  est = "default",
  name = "cors",
  check = FALSE,
  save_out = FALSE
)
```

Arguments

`data`	A dataframe or object coercible to a dataframe. Data must include all observed variables used in any of the models.
`fit_y`	A named list of CFA or bi-factor fitted objects.
`fit_x`	'NULL' or a named list of CFA or bi-factor fitted objects to be correlated with fit_y variables. Defaults to 'NULL'.
`items`	A vector of single-item variables to correlate with `fit_y` latent variables. Must not include any items contributing to the measurement of a `fit_y` latent variable.
`item_loadings`	When single items are specified, items are included in models with single item latent variables. `item_loadings` sets the loading of the item on the factor. It can be a single number to set all loadings equal or a vector of length equal to the length of items. Defaults allows the value to be free (which assumes perfect reliability). Irrelevant if `items = NULL`.
`nagy`	Logical. Indicates whether to use Nagy and colleagues' (2017) extension procedure instead of Burt's (1976) 2-stage procedure.
`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`.

`miss` A string. Sets the `missing` parameter, as per lavaan (see lavaan::lavOptions). Defaults to 'default', which uses 'ML' for `ordered = NULL` and 'pairwise' for any other value of `ordered`.

`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 "cors". 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 computes correlations between latent variables from fitted CFA or bi-factor models. If both `fit_x = NULL` and `items = NULL`, then correlations between all `fit_y` latent variables are computed. If either `fit_x` or `items` are specified, then correlations will be computed between `fit_y` latent variables and any specified `fit_x` latent variables and `items`. Items are treated as single item latent variables with loadings of `item_loadings` if specified or freely estimated otherwise. Hierarchical models are not supported and the function will exit with an error if included.

Correlation are calculated in separate models. Primarily, this approach means that correlations can be calculated with typical sample sizes in a manageable time frame compared to including everything in one model.

To control for interpretational confounding (Burt, 1976), the function uses either Burt's (1976) 2-stage procedure (`nagy = FALSE`) or Nagy and colleagues' (2017) extension procedure (`nagy = TRUE`). Interpretational confounding occurs when the interpretation of a latent variable is confounded by the inclusion of a conceptually distinct variable in the model. That is, the loadings of items on a latent variable can change, in some cases markedly, due to the inclusion of a conceptually distinct factor. This means that the factor does not represent a consistent concept for different models, even in the same sample, as the loadings can change model to model.

Burt (1976) proposed simply fixing measurement model parameters in the model estimating structural parameters. This method means that latent variables' interpretations cannot change with the

addition or removal of different variables and less than ideal fit at the measurement level cannot affect latent variable correlations. Burt's (1976) method therefore solve interpretational confounding. However, Burt's method also underestimates uncertainty in the measurement part of the full model (e.g., Nagy et al., 2017), which results in biased standard errors and fit statistics, although effects are usually small.

Alternatively, Nagy's method involves allowing item residuals to correlate with external variables and constrains those relationships such that the model is identifiable. Specifically, one of the methods to constrain these relationships minimises the sum of squares of the correlations between each latent variable's items' residuals and each of the structural variables in the model (excluding the factor(s) they form part of the measurement of). By using this method, measurement parameters in the structural model match those of isolated measurement models without having to constrain them directly. As a result, unbiased standard errors are preserved while simultaneously eliminating interpretational confounding.

Which method should be chosen? Burt's method is faster but artificially constrains parameters, thereby biasing standard errors and model fit indices. They should, however, give very similar point estimates for correlations. Therefore, Nagy's method should be preferred whenever confidence intervals or model fit matter, except, perhaps, for large sets of variables.

If Nagy and colleagues' (2017) method is selected (with `nagy = TRUE`, the default), correlations between factors and item residuals will be included in the output and may provide useful insight into idiosyncratic item variance (Nagy et al., 2017). `nagy = TRUE` is not currently supported for measurement models with more than one latent variable. Any such models (except for hierarchical models, which are not supported at all) will be switched to use Burt's method with a warning. If all included measurement models included more than one latent variable, then outputs will be switched to match `nagy = FALSE` with a warning.

It is possible for latent variable correlations to produce a non-positive definite correlation matrix between variables included in `fit_y` (when `fit_x` and `items` are not specified), especially when closely related factors are included. If the matrix of latent variables is not positive definite, then the matrix will be adjusted to the nearest positive definite matrix using the Matrix::nearPD function, which employs the method developed by Higham (2002), and a message will state that the matrix was adjusted, and what the maximum adjustment to any cell was. Confidence intervals will be adjusted by the same amount as the corresponding cell of the correlation matrix. P-values will not be adjusted, however, so will be very slightly incorrect. In general, inaccuracies in p-values will be most likely for larger correlations (i.e., ones more likely to be highly significant) but if the maximum adjustment, printed in the warning, is large, then use p-values with care.

The model relies on sem.check for the back-end of running the models, which 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. In the case of `sem.cor`, the number of correlations can add up quickly. When `nagy = FALSE`, this is unlikely to be an issue as each model typically runs in a fraction of a second, but with `nagy = TRUE` and 20 scales, 19 + 18 + 17 + ... + 1 = 190 correlations would need to be calculated and, if they took an average of 5 seconds each to run, the total run time would be ~16 minutes. In these cases, the functionality may be useful. For further details on how this works, see the sem.check function documentation.

Value

Returns a list of length 4-7 depending on option selections. All versions include fitted lavaan models (`fit`); a correlation matrix (`cor_mat`); p-values of the correlations (`pvalues`); and a list of upper and lower 95% confidence intervals of the correlations (`ci`). If `fit_save = TRUE`, a list of fit measures is also returned (`fit_measures`), and, if `nagy = TRUE`, matrices of residual correlations, their p-values, and lists of their 95% confidence intervals are also returned (`residual_cors`). If both `fit_y` and `fit_x` are specified, then there is one set of residual correlation results for the items from each set of factors (i.e., `residual_cors_x` and `residual_cors_y` for the items of 'x' and 'y' factors, respectively).

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.

Higham, N. J. (2002). Computing the nearest correlation matrix—a problem from finance. IMA Journal of Numerical Analysis, 22(3), 329-343. https://doi.org/10.1093/imanum/22.3.329.

Nagy, G., Brunner, M., Lüdtke, O., and Greiff, S. (2017). Extension Procedures for Confirmatory Factor Analysis. Journal of Experimental Education, 85(4), 574-596. https://doi.org/10.1080/00220973.2016.1260524.

See Also

sem.check, lavaan::sem, matrixcalc::is.positive.definite, Matrix::nearPD

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 CFA models
cfa_fit <- cfa.from.keys(keys, BFIGritHope, check = FALSE, fit_save = FALSE)
# Find correlations between all cfa_fit constructs.
cors <- sem.cor(BFIGritHope, cfa_fit$fit, nagy = FALSE)
# View the correlation matrix
cors$cor_mat

# Correlations of grit facets with hope facets and the first item from each
# Big Five factor.
items <- names(BFIGritHope)[grep("bfi_.*1_1", names(BFIGritHope))]
cors2 <- sem.cor(
  BFIGritHope, cfa_fit$fit[1:2], cfa_fit$fit[3:4], items, nagy = FALSE
)
# View correlations
cors2$cor_mat
```

sem.path *Runs ESEM based on CFA and EFA model outputs.*

Description

sem.path runs latent variable structural equation models in lavaan. The function takes lists of fitted CFA lavaan model objects for the measurement models and lavaan code for the structural model and runs uses these to create the model code and run the model.

Usage

```
sem.path(
  path,
  data,
  cfa_fit,
  extra = NULL,
  fit_save = TRUE,
  fit_measures = "all",
  miss = "default",
  est = "default",
  name = "sam",
  check = FALSE,
  save_out = FALSE
)
```

Arguments

path	Regression or paths between latent variables or single-item indicators in lavaan code.
data	A dataframe or object coercible to a dataframe. Data must include all observed variables used in any of the models.
cfa_fit	A named list of fitted lavaan objects of CFA models.
extra	Extra lavaan code to be added that is not created from cfa_fit, or included in path. For example, the argument can be used to set constraints, fix parameters to specific values, or allow correlations between item residuals or latent variables.
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 'default', which uses 'ML' for ordered = NULL and 'pairwise' for any other value of ordered.

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 "sem". Should never be "" or NULL but otherwise 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 runs an SEM with fitted CFA models, a specified path, and, optionally, 'extra' lavaan code. The function uses Rosseel and Loh's (2022) "Structure-After-Measurement" (SAM) procedure to control for interpretational confounding.

Like lavaan, the model assumes independent variables in the structural model should be allowed to correlate by default but extends the treatment to single-item variables. As a corollary, if you do not want any combination of independent variables to correlate freely, you will have to specify that in 'extra' (see examples).

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. Given sem.path runs a single model at a time, it is unlikely to be necessary except for extremely complex models. For further details on how this works, see the sem.check function documentation.

In SEM, standard methods do not distinguish between measurement and structural parameters. As a result, measurement model parameters can change with the addition of theoretically unrelated constructs in a structural model, and can change differently for different sets of unrelated constructs. This means that the unrelated constructs are changing the interpretation of the latent variable, which Burt (1976) referred to as "interpretational confounding".

There are various ways to deal with interpretational confounding. The standard solution (other than ignoring it) is to create good fitting measurement models first, then freely estimate the structural model with checks to ensure adequate fit of the model and that measurement parameters do not substantially change with different combinations of factors. This is sometimes a good solution, but, in other cases, it is not. For example, if the measurement model was for a well-established scale and it requires changing, then it loses easy comparison with past research. This issue is most clearly relevant when changes to a measurement model require entirely different factors, or items to be removed but it is still an issue for less dramatic changes. When a single scale is being assessed, these issues can be resolved by suggesting a thorough evaluation of the scale and, perhaps, the

suggestion of a new measurement model or a new scale for a particular population; however, when many scales are being assessed this solution is impractical, and may not solve the interpretational confounding issue regardless.

An alternative solution, proposed by Burt (1976) is to fix measurement model parameters in a model estimating structural parameters. This method means that misspecification of one measurement model cannot affect other measurement models and that the interpretation of measured constructs cannot change based on unrelated factors. However, it is not a perfect solution because, by fixing measurement parameters, uncertainty in their estimation is neglected (e.g., Nagy et al., 2017), which results in biased standard errors and fit statistics.

A third option was proposed by Nagy and colleagues (2017), who introduced an extension procedure such that item residuals are allowed to correlate with external variables (or factors). To make the model identifiable, these relationships are constrained using one of a number of methods. If the sums of squares of correlations between all combinations of factors' items and external factors are minimised, measurement parameters in isolated measurement models are preserved in the structural model without having to constrain them directly. As a result, unbiased standard errors are preserved while simultaneously eliminating interpretational confounding since the measurement parameters from the measurement models are preserved regardless of external factors. Unfortunately, estimating these models becomes increasingly slow with more items and factors, such that it quickly becomes untenable. Moreover, the method only works with correlations, not regressions, so some method to run regressions using the correlations needs to be implemented that does not itself result in biased estimates due to ignored uncertainty in the correlation estimates.

Although this latter issue may be solvable for Nagy and colleagues' (2017) method, a more practical solution to these issues was proposed by Rosseel and Loh (2022) with their SAM approach. This method essentially follows Burt's (1976) method but adjust the procedure to overcome its issues. They distinguish two SAM varieties–"local SAM" and "global SAM". Local SAM uses the observed summary statistics of the parameters of the measurement models to generate mean and covariance matrices to use in the structural model, which preserves the structure of the measurement models while also preserving the uncertainty. Global SAM treats the measurement parameters as given, but corrects the standard errors of the structural model.

Given its practicality, `sem.path` uses the local SAM method. Local SAM was selected over the global SAM because Rosseel and Loh (2022) "recommend local SAM over global SAM whenever possible." (p. 21 of the pre-print version).

Note that latent variables that are included in a measurement model that are not part of a regression path are assumed (by lavaan) to be unrelated, even if explicitly freed in the code. In most cases, you would not want such variables; however, if you include a bi-factor model, group variables that are not part of the structural path model will be assumed to be unrelated to other structural variables. Similarly, if you are attempting to follow Hayes (2021) recommendations for incremental validity, then there is no way to force the focal variable to correlate with the outcome using SAM. `sem.path` should, therefore, *NOT* be used with bifactor models or to test for incremental validity using Hayes (2021) method.

Finally, the function also does not currently work correctly with ESEM. The latest lavaan version (0.7-2 at time of writing) and earlier incorrectly treats factor covariances as equal in SAM with `sam_method = "local"`. In the future, the function may change to allow users to select `sem_method = "global"` or to select "global" dynamically when an ESEM is included but neither of these is currently implemented, so you should also *NOT* include ESEM in `sem.path` models.

Value

Returns a list of length 5 (if `fit_save = FALSE`) or 6 (if `fit_save = TRUE`). The elements of the list are: a lavaan model output object (fit); a data frame of standardised parameter estimates (par_std); a vector of fit measures for the model if `fit_save = TRUE` (fit_measures); a data frame of structural regression parameters (b); a data frame of R-squared values (r2); and a data frame of structural correlation parameters (cors).

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.

Hayes, T. (2021). R-squared change in structural equation models with latent variables and missing data, 53(5), 2127-2157. https://doi.org/10.3758/s13428-020-01532-y.

Nagy, G., Brunner, M., Lüdtke, O., and Greiff, S. (2017). Extension Procedures for Confirmatory Factor Analysis. Journal of Experimental Education, 85(4). https://doi.org/10.1080/00220973.2016.1260524.

Rosseel, Y. & Loh, W. W. (2022). A structural after measurement approach to structural equation modeling. Psychological Methods, 29(3), 561-588. https://doi.org/10.1037/met0000503.

See Also

lavaan::sam, sem.check

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 CFA models
cfa_fit <- cfa.from.keys(keys, BFIGritHope, check = FALSE, fit_save = FALSE)
# Run a path model with grit_c allowed to correlate with hope_p's residual
# and the correlation between hope_a and grit_c constrained to 0.
# Note: "\n" indicates a new line and is interpreted identically to an new
# line by lavaan.
sem_fit <- sem.path(
  path = "grit_p ~ grit_c + hope_p + bfi_c1_1\nhope_p ~ hope_a",
  data = BFIGritHope,
  cfa_fit = cfa_fit$fit,
  extra = "grit_c ~~ hope_p\nhope_a ~~ 0*grit_c"
)
# Examine results
summary(sem_fit)      # Standard lavaan summary
sem_fit$b             # Standardised regression path coefficients
sem_fit$r2            # R^2 values
sem_fit$cors          # Correlations
sem_fit$fit_measures  # Fit measures
```

Index