Package 'semFromKeys'

November 18, 2026

Type Package

Title Run 'lavaan' Models from Keys Lists

Version 0.6.0

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 8.1.1

Depends R (>= 3.6.0)

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

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 2026-11-18 06:50:02 UTC

Contents

BFIGritHope . . . 2
bifactor.from.keys . . . 4
cache.clean . . . 8
cache.setup . . . 10
cfa.from.keys . . . 12
efa.from.keys . . . 14
esem.from.keys . . . 17
r2.change . . . 22
sem.check . . . 27
sem.cor . . . 31
sem.path . . . 36

Index **41**

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

Description

The subset of the Sample 2b data reported in Bainbridge, Ludeke, & Smillie (2022) including only the BFI-2, Grit, and Hope. The data were collected in May 2019 from Mechanical Turk.

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,
  extra = NULL,
  fit_save = TRUE,
  fit_measures = "all",
  progress = TRUE,
  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. keys_g 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. keys_b 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. keys will only be the same length as keys_g and keys_b when each model has only one group factor.
data	A data frame or object coercible to a data frame. Data must include all observed variables used in any of the models.
extra	A vector of strings of residuals to correlate in lavaan code (e.g., "item1 ~~ item2"). Strings of code can be in any order. Whenever a model includes all items in a string, the string will be appended to the model code.
fit_save	Logical. If TRUE, model fit measures are included in the output. If FALSE, model fit measures are 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.
progress	If TRUE, messages showing progress on individual models are printed. If FALSE, general messages on progress and warning and error messages for individual models are printed. Defaults to TRUE. By default, lots of models will result in

extensive messages being printed. To include only bare-bones progress (e.g., "Fitting models", "Generating parameter estimates"), errors, and warnings, set `progress = FALSE`.

`std.lv` Logical. Sets the `std.lv` parameter, as per lavaan (see lavaan::lavOptions). If `TRUE`, factor variances are fixed to 1. If `FALSE`, the loading of each factor's first item is 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. If `TRUE`, inputs are compared to previous inputs and, if they exist, the model will not be rerun if nothing has changed. If `FALSE`, the model will be run regardless of the existence of previous inputs.

`save_out` Logical. If `TRUE`, model code, a hash of the data, input parameter values, and output are saved. If `FALSE`, nothing is saved. Selecting `save_out = TRUE` enables the function to not rerun models next time if `check = TRUE` 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.

In simulation studies, bi-factor models 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 bi-factor 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 (standardised 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
keys <- list(
  grit_p = paste0("grit_p_", 1:6),
  grit_c = paste0("grit_c_", 1:6),
  hope_a = paste0("hope_a_", 1:4),
  hope_p = paste0("hope_p_", 1:4)
)
keys_g <- list(grit = unlist(keys[1:2]), hope = unlist(keys[3:4]))
keys_b1 <- sapply(
  names(keys_g),
  function(x) names(keys)[grep(x, names(keys))],
  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,
  extra = c("grit_c_1 ~~ grit_p_1", "hope_a_1 ~~ hope_p_1")
)
# 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 should be a named list of scales, where each element contains a vector of items 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,
  extra = NULL,
  fit_save = TRUE,
  fit_measures = "all",
  progress = TRUE,
  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 vector of items included in each scale.
`data`	A data frame or object coercible to a data frame. Data must include all observed variables in any of the keys.
`extra`	A vector of strings of residuals to correlate in lavaan code (e.g., `"item1 ~~ item2"`). Strings of code can be in any order. Whenever a model includes all items in a string, the string will be appended to the model code.
`fit_save`	Logical. If `TRUE`, model fit measures are included in the output. If `FALSE`, model fit measures are 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`.

progress — If TRUE, messages showing progress on individual models are printed. If FALSE, general messages on progress and warning and error messages for individual models are printed. Defaults to TRUE. By default, lots of models will result in extensive messages being printed. To include only bare-bones progress (e.g., "Fitting models", "Generating parameter estimates"), errors, and warnings, set progress = 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. If TRUE, inputs are compared to previous inputs and, if they exist, the model will not be rerun if nothing has changed. If FALSE, the model will be run regardless of the existence of previous inputs.

save_out — Logical. If TRUE, model code, a hash of the data, input parameter values, and output are saved. If FALSE, nothing is saved. Selecting save_out = TRUE enables the function to not rerun models next time if check = TRUE and nothing has changed in the meantime.

Details

If keys is specified as a vector of items, the function will assume those items are meant to comprise a single scale, and will convert the input to a length 1 list with the element named 'factor'. A warning will be sent when this occurs.

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
keys <- list(
  grit_p = paste0("grit_p_", 1:6),
  grit_c = paste0("grit_c_", 1:6),
  hope_a = paste0("hope_a_", 1:4),
  hope_p = paste0("hope_p_", 1:4)
)
# Run models
cfa_fit <- cfa.from.keys(
  keys, BFIGritHope,
  extra = c("grit_c_1 ~~ grit_c_2", "grit_p_1 ~~ grit_p_2")
)
# 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,
  extra = NULL,
  orthogonal = FALSE,
  fit_save = TRUE,
  fit_measures = "all",
  progress = TRUE,
  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 data frame or object coercible to a data frame. Data must include all observed variables used in any of the models.
extra	A vector of strings of residuals to correlate in lavaan code (e.g., "item1 ~~ item2"). Strings of code can be in any order. Whenever a model includes all items in a string, the string will be appended to the model code.
orthogonal	Logical. Sets the orthogonal parameter, as per lavaan (see lavaan::lavOptions). If TRUE, unspecified latent variable correlations should be fixed at 0. If FALSE, unspecified latent variable correlations should be freely estimated. Defaults to FALSE.
fit_save	Logical. If TRUE, model fit measures are included in the output. If FALSE, model fit measures are 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.
progress	If TRUE, messages showing progress on individual models are printed. If FALSE, general messages on progress and warning and error messages for individual models are printed. Defaults to TRUE. By default, lots of models will result in extensive messages being printed. To include only bare-bones progress (e.g., "Fitting models", "Generating parameter estimates"), errors, and warnings, set progress = FALSE.
std.lv	Logical. Sets the std.lv parameter, as per lavaan (see lavaan::lavOptions). If TRUE, factor variances are fixed to 1. If FALSE, the loading of each factor's first item is 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 '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. If TRUE, inputs are compared to previous inputs and, if they exist, the model will not be rerun if nothing has changed. If FALSE, the model will be run regardless of the existence of previous inputs.
save_out	Logical. If TRUE, model code, a hash of the data, input parameter values, and output are saved. If FALSE, nothing is saved. Selecting save_out = TRUE enables the function to not rerun models next time if check = TRUE and nothing has changed in the meantime.

Details

The function was designed to streamline running exploratory structural equation models (ESEM) using esem.from.mods. 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 bi-factor 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, esem.from.mods

Examples

```
# Create EFA keys
# Using only 3 factors and 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 <- list(
  bfi_e = c(
    paste0("bfi_e", c("1_1", "1_2", "2_1", "2_2", "3_1", "3_2"))
  ),
  bfi_a = c(
    paste0("bfi_a", c("1_1", "1_2", "2_1", "2_2", "3_1", "3_2"))
  ),
```

```
  bfi_c = c(
    paste0("bfi_c", c("1_1", "1_2", "2_1", "2_2", "3_1", "3_2"))
  )
)
# 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` and `efa.from.mods` run exploratory structural equation models (ESEM) in lavaan where exploratory factor analysis (EFA) factors predict confirmatory factor analysis (CFA) factors.

Usage

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

esem.from.mods(
  data,
  efa_fit,
  cfa_fit = NULL,
  bif_fit = NULL,
  fit_save = FALSE,
  fit_measures = "all",
  progress = TRUE,
  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 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.
extra	A vector of strings of residuals to correlate in lavaan code (e.g., "item1 ~~ item2"). Strings of code can be in any order. Whenever a model includes all items in a string, the string will be appended to the model code. Currently the argument only supports allowing correlations between item residuals.
fit_save	Logical. If TRUE, model fit measures are included in the output. If FALSE, model fit measures are 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.
progress	If TRUE, messages showing progress on individual models are printed. If FALSE, general messages on progress and warning and error messages for individual models are printed. Defaults to TRUE. By default, lots of models will result in extensive messages being printed. To include only bare-bones progress (e.g., "Fitting models", "Generating parameter estimates"), errors, and warnings, set progress = 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" for esem.from.keys and "esem" for esem.from.mods. 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. If TRUE, inputs are compared to previous inputs and, if they exist, the model will not be rerun if nothing has changed. If FALSE, the model will be run regardless of the existence of previous inputs.

save_out	Logical. If TRUE, model code, a hash of the data, input parameter values, and output are saved. If FALSE, nothing is saved. Selecting save_out = TRUE enables the function to not rerun models next time if check = TRUE and nothing has changed in the meantime.
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.

Details

esem.from.keys and esem.from.mods streamline running exploratory structural equation models (ESEM) where EFA factors predict a series of latent variables in separate models, similar to the primary analyses of Bainbridge, Ludeke, and Smillie (2022). The functions are designed to determine how well a broad instrument, such as the BFI-2 (Soto & John, 2017) or the HEXACO-PI (Lee & Ashton, 2004) can describe scales' variance.

esem.from.keys takes keys lists as inputs and esem.from.mods takes fitted measurement models as inputs. To prevent interpretational confounding (Burt, 1976), esem.from.keys uses Rosseel and Loh's (2022) "Structural-After-Measurement" (SAM) method and esem.from.mods uses Burt's (1976) 2-stage procedure. In general, esem.from.keys should be used for CFA measurement models and esem.from.mods should be used for bi-factor measurement models, which are not supported in esem.from.keys (see the Bi-factor Measurement Models section).

The SAM method includes "local" and "global" versions (see the Interpretational Confounding section). Although local SAM is preferable in most circumstances, as implemented in lavaan it currently (as at version 0.7-2) incorrectly sets ESEM factor covariances as equal, so it is not appropriate for esem.from.keys. Therefore, esem.from.keys uses global SAM.

The functions rely 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.

When using esem.from.keys with fit_save = TRUE, lavaan will produce a warning for every fit calculation (at version 0.7-2). The warning notes that, "the behaviour of fit measures in this context has not been studied, and they should be interpreted with caution." As fit statistics are used to determine if a model approximates a complete description of the data, this should not be an issue for esem.from.keys as the goal of the function is to assess the overlap between ESEM factors and independent scales with the better measurement of latent variable models, not to create a parsimonious model of the items. Moreover, given interpretational confounding has been dealt with and all structural paths are specified, fit is entirely determined by the measurement models and should be evaluated in separate measurement models, not in the full structural model, created by the esem.from.keys function.

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 data frame of R-squared values from each model.

Interpretational Confounding

In Structural Equation Models (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) labelled "interpretational confounding".

There are various ways to deal with interpretational confounding. The standard solution (other than ignoring it) is to create measurement models with good fit first, then freely estimate the structural model with checks to ensure adequate fit of the model and that measurement parameters do not change substantially with different combinations of factors. This is sometimes a good solution, but if, for example, the measurement model was for a well-established scale and it requires changing, then it loses easy comparison with past research.

An alternative solution, proposed by Burt (1976), is to fix measurement model parameters in the 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—by fixing measurement parameters, uncertainty in their estimation is neglected (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 each factor's items' residuals and each external factor are minimised, measurement parameters 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. Unfortunately, estimating these models becomes increasingly slow with more items and factors, and the method only works with correlations, not regressions, so some method to run regressions using the correlations needs to be implemented whenever more complicated structural models are required. These methods will typically bias the estimates due to ignored uncertainty in the correlation estimates, thereby undermining the primary benefit of the method.

A final solution to interpretation confounding was proposed by Rosseel and Loh (2022) with their "Structural-After-Measurement" (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, thereby preserving 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 to account for the uncertainty in their measurement.

Bi-factor Measurement Models

Bi-factor models are not currently supported in `esem.from.keys`. lavaan::sam—the lavaan function that implements the SAM method— currently treats 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 bi-factor model on EFA factors requires the group factors to correlate with the EFA factors, lavaan::sam is currently inappropriate for bi-factor models predicted by ESEM factors (as at lavaan version 0.7-2).

As a result, `esem.from.mods` should be used whenever bi-factor models are included. Note, however, that the solution to interpretational confounding means that standard errors and fit statistics will be biased, so inferring precise p-values, determining an optimal model, or checking that a model surpasses some cut-off of "good fit" should only be used with care. 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.

Bi-factor models do come with a further complication—they 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 `esem.from.mods` already uses the 2-stage procedure, this method is employed when bi-factor models are used.

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.

Lee, K. & Ashton, M. C. (2004). Psychometric Properties of the HEXACO Personality Inventory. Multivariate Behavioral Research, 39(2), 329-358. https://doi.org/10.1207/s15327906mbr3902_8.

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.

Soto, C. J. & John, O. P. (2017). The next Big Five Inventory (BFI-2): Developing and assessing a hierarchical model with 15 facets to enhance bandwidth, fidelity, and predictive power. Journal of Personality and Social Psychology, 113(1), 117-143. https://doi.org/10.1037/pspp0000096.

See Also

sem.check, lavaan::sam

Examples

```
# Create CFA keys
keys <- list(
  hope_a = paste0("hope_a_", 1:4),
  hope_p = paste0("hope_p_", 1:4)
)
# 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_e <- list(
  bfi_e = c(
    paste0("bfi_e", c("1_1", "1_2", "2_1", "2_2", "3_1", "3_2"))
  ),
  bfi_a = c(
    paste0("bfi_a", c("1_1", "1_2", "2_1", "2_2", "3_1", "3_2"))
  ),
  bfi_c = c(
    paste0("bfi_c", c("1_1", "1_2", "2_1", "2_2", "3_1", "3_2"))
  )
)

# Run 'esem.from.keys' models
esam_fit <- esem.from.keys(
  BFIGritHope, keys_e, keys,
  extra = c("hope_a_1 ~~ hope_a_2", "hope_p_1 ~~ hope_p_2")
)
# Examine results
summary(esam_fit$fit$grit_c)  # Standard lavaan summary
esam_fit$r2                   # R-squareds
esam_fit$b                    # Betas

# Run 'esem.from.mods' models
# First, 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
)
# Examine results
summary(esem_fit$fit$grit_c)  # Standard lavaan summary
esem_fit$r2                   # R-squareds
esem_fit$b                    # Betas
```

`r2.change` *Run incremental validity analysis using latent variables*

Description

Incremental validity using standard linear regressions is invalid with less than perfect (or close to perfect) reliability (Westfall & Yarkoni, 2016). Latent variable models must be used instead. However, this is difficult, especially if confidence intervals are required, which can only be created with bootstrap sampling.

`r2.change` streamlines the process of running incremental validity analyses with latent variables, including generating bootstrap confidence intervals.

The function was inspired by work from Hayes (2021) and uses `boot:::norm.inter` copied from `boot`.

Usage

```
r2.change(
  data,
  X,
  Y,
  Z,
  keys,
  keys_g = NULL,
  keys_b = NULL,
  extra = NULL,
  boot = NULL,
  boot_ci = 0.95,
  seed = NULL,
  fit_save = TRUE,
  fit_measures = "all",
  progress = TRUE,
  std.lv = TRUE,
  miss = "default",
  est = "default",
  ordered = NULL,
  name = "dr2",
  check = FALSE,
  save_out = FALSE
)
```

Arguments

`data`	A data frame or object coercible to a data frame. Data must include all observed variables used in any of the models.
`X`	A length 1 character vector of the name of the the variable being tested.
`Y`	A string of the name of the outcome variable.
`Z`	A character vector of the set of variables that X is being pitted against or `NULL` to use all variables in `cfa_fit` and `bif_fit`. Defaults to `NULL`.
`keys`	A named list of items in uni-dimensional factors. These can be names of single-factor latent variables or group factor latent variables in a bifactor model. Each list element must be a vector of items that load on the latent variable.
`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. `keys_g` must be the same length as `keys_b`. Leave `keys_g = NULL` when no bifactor models are required.
`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. `keys_b` must be the same length as `keys_g`. Leave `keys_b = NULL` when no bifactor models are required.

extra	A string or vector of strings of lavaan code to be appended to the model. The argument can be used to set constraints, fix parameters to specific values, or allow correlations between item residuals.
boot	Numeric or NULL. If NULL, then bootstrap confidence intervals are not calculated. If numeric, then bootstrap confidence intervals are calculated with a number of bootstrap samples equal to the value of boot.
boot_ci	Numeric. The confidence interval proportion for bootstrap CIs. boot_ci must be greater than 0 and less than 1. Irrelevant if boot = NULL.
seed	A number used in set.seed() for bootstrap data selection. Defaults to NA indicating a random
fit_save	Logical. If TRUE, model fit measures are included in the output. If FALSE, model fit measures are 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.
progress	If TRUE, messages showing progress on individual models are printed. If FALSE, general messages on progress and warning and error messages for individual models are printed. Defaults to TRUE. By default, lots of models will result in extensive messages being printed. To include only bare-bones progress (e.g., "Fitting models", "Generating parameter estimates"), errors, and warnings, set progress = FALSE.
std.lv	Logical. Sets the std.lv parameter, as per lavaan (see lavaan::lavOptions). If TRUE, factor variances are fixed to 1. If FALSE, the loading of each factor's first item is 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 "sem". The name should be unique for each set of models, or outputs from calls with the same name will be overwritten.
check	Logical. If TRUE, inputs are compared to previous inputs and, if they exist, the model will not be rerun if nothing has changed. If FALSE, the model will be run regardless of the existence of previous inputs.
save_out	Logical. If TRUE, model code, a hash of the data, input parameter values, and output are saved. If FALSE, nothing is saved. Selecting save_out = TRUE enables the function to not rerun models next time if check = TRUE and nothing has changed in the meantime.

Details

Incremental validity analyses and 2-stage regressions are invalid when using mean or sum scores (Westfall & Yarkoni, 2016). To illustrate, take two separate scales that measure the same construct. Neither scale should have incremental validity over the other. However, unique item variance means that they will not correlate at 1 and any uncertainty in the measurement due to unique items variance (and indicated by reliability) is obscured from the model. Thus, the standard errors of the overlap in the scales' variance is incorrectly too narrow and the likelihood of false positives increases. The chance of a false positive increases with sample size and lower reliabilities (down to .5), and rises towards 1 as N increases (Westfall & Yarkoni, 2016).

To solve this issue, latent variables must be used (Westfall & Yarkoni, 2016). Latent variable models have two advantages over aggregate scores in this context. First, they measure relationships between the shared variance of their respective items, preserving the possibility of correlations of 1 and drastically reducing the impact of idiosyncratic item variance. Second, poorer reliability increases the uncertainty in the measurement, such that, as the correlation between two different scales of the same constructs becomes smaller, uncertainty increases. Together these factors mean that separate scales measuring the same constructs should fail to demonstrate incremental validity at an appropriate rate.

`r2.change` makes the process of running incremental validity analyses with latent variables much easier. It takes data, keys, and names for the focal variable (X), the outcome variable (Y), and variables to test against (Z) as inputs, and produces an r-squared change figure, optionally with bootstrap confidence intervals. Various other outputs of the process are also returned (see `Value`).

Traditionally incremental validity analyses rely on two models. A reduced model regresses Y on Z and a full model adds X as a predictor. R-squared change between the models is used to assess the incremental validity of X predicting Y. Such a method cannot be used naively with latent variables because excluding X from the reduced model can change the structural model such that the r-squared values cannot be compared. Hayes (2021) discussed the different approaches to the issue. One appropriate method leaves the X variable in the reduced model but allows it to correlate with Y's residual. Another method calculates r-squared for the reduced model using correlations from the full model. `r2.change` uses the first of these approaches to produce a regression table for the reduced model (so it can be included in a table, for example) and it uses the second approach to calculate r-squared changes both in the base models and bootstrap samples. The latter method means that minor changes in the models' convergence do not affect the r-squared estimates, even though the models are merely different versions of the same model (i.e., same fit statistics, degrees of freedom etc. just with a different structural path).

The problem of models being incomparable when variables are added or removed is known as "interpretational confounding" (Burt, 1976) and occurs because the model fitting method does not distinguish between measurement and structural parameters. Thus, measurement parameters will shift depending on what other variables are in the model, thereby changing the interpretation of the latent variables. `r2.change` uses Burt's 2-stage approach to account for interpretational confounding. The method produces accurate point estimates but somewhat biases standard errors and fit statistics (Nagy et al., 2017). This means standard errors for regression coefficients and other model parameters will be slightly biased, but the r-squared change or bootstrap CIs will no be affected because these are entirely calculated with point estimates. For further details, see the "Interpretational Confounding" section.

If using bootstrap confidence intervals, it is highly recommended that you use `save_out = TRUE` and `check = TRUE`, especially for more complex models. The method requires setting up a cache

directory with `cache.setup()` before running but it then means that the bootstrap models will not have to be run again if they have already been run and nothing has changed (including number of bootstraps, argument inputs, or relevant data). The functionality can be helpful if something has been changed in earlier code and you are not sure if it affects the bootstrap models. If it did not, then running the code will be quick and you will know that it is still accurate.

Value

A list of length X. Elements include the fitted lavaan objects for the full and reduced models; estimates of R-squared and R-squared change; and data frames of covariate effect sizes from both the full and reduced models.

Interpretational Confounding

In Structural Equation Models (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) labelled "interpretational confounding".

There are various ways to deal with interpretational confounding. The standard solution (other than ignoring it) is to create measurement models with good fit first, then freely estimate the structural model with checks to ensure adequate fit of the model and that measurement parameters do not change substantially with different combinations of factors. This is sometimes a good solution, but if, for example, the measurement model was for a well-established scale and it requires changing, then it loses easy comparison with past research.

An alternative solution, proposed by Burt (1976), is to fix measurement model parameters in the 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—by fixing measurement parameters, uncertainty in their estimation is neglected (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 each factor's items' residuals and each external factor are minimised, measurement parameters 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. Unfortunately, estimating these models becomes increasingly slow with more items and factors, and the method only works with correlations, not regressions, so some method to run regressions using the correlations needs to be implemented whenever more complicated structural models are required. These methods will typically bias the estimates due to ignored uncertainty in the correlation estimates, thereby undermining the primary benefit of the method.

A final solution to interpretation confounding was proposed by Rosseel and Loh (2022) with their "Structural-After-Measurement" (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,

thereby preserving 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 to account for the uncertainty in their measurement.

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. Behavior Research Methods, 53(5), 2127-2157. https://doi.org/10.3758/s13428-020-01532-y.

Westfall, J., and Yarkoni, T. (2016). Statistically controlling for confounding constructs is harder than you think. PLoS ONE, 11(3), 1-23. https://doi.org/10.1371/journal.pone.0152719.

See Also

`cache.setup()` `sem.check()` `cfa.from.keys()` `bifactor.from.keys()`

Examples

```
# Test incremental validity of grit_c over hope predicting grit_p
# Create keys
keys <- list(
  grit_p = paste0("grit_p_", 1:6),
  grit_c = paste0("grit_c_", 1:6),
  hope_p = paste0("hope_p_", 1:4)
)
keys_g <- list(hope = c(keys$hope_p, paste0("hope_a_", 1:4)))
keys_b <- list(hope = "hope_p")
dr2 <- r2.change(
  BFIGritHope, X = "grit_c", Y = "grit_p", Z = "hope",
  keys = keys, keys_g = keys_g, keys_b = keys_b, fit_save = FALSE
)
dr2$dr2
```

---

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.

`check.cache` looks at previously save model input and outputs and returns tests for whether anything has changed. `check.cache` is primarily used by `sem.check` to test whether to re-run models (see details).

Usage

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

check.cache(
  name,
  mods = NULL,
  miss = NULL,
  est = NULL,
  std = NULL,
  std.lv = NULL,
  orthogonal = NULL,
  ordered = NULL,
  data = NULL,
  keys_s = NULL,
  keys_e = NULL,
  seed = NULL
)
```

Arguments

mods	A named list of lavaan models to run.
data	A data frame or object coercible to a data frame. 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. If TRUE, model fit measures are included in the output. If FALSE, model fit measures are 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.

`std.lv` Logical. Sets the `std.lv` parameter, as per lavaan (see lavaan::lavOptions). If `TRUE`, factor variances are fixed to 1. If `FALSE`, the loading of each factor's first item is fixed to 1. Defaults to `TRUE`.

`std` Logical. If `TRUE`, standardised parameter estimates are saved. If `FALSE`, unstandardised parameters estimates are 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). If `TRUE`, unspecified latent variable correlations should be fixed at 0. If `FALSE`, 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.

`progress` If `TRUE`, messages showing progress on individual models are printed. If `FALSE`, general messages on progress and warning and error messages for individual models are printed. Defaults to `TRUE`. By default, lots of models will result in extensive messages being printed. To include only bare-bones progress (e.g., "Fitting models", "Generating parameter estimates"), errors, and warnings, set `progress = FALSE`.

`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. If `TRUE`, inputs are compared to previous inputs and, if they exist, the model will not be rerun if nothing has changed. If `FALSE`, the model will be run regardless of the existence of previous inputs.

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

use_sam — Logical. If TRUE, the lavaan::sam function is used for model estimation. If FALSE, the lavaan::sem function is not used for model estimation.

seed — Numeric. Sets a seed for bootstrap sampling (used in r2.change). Checked it has not changed since last model run in check.cache.

Details

Both sem.check and check.cache are largely intended to be used as helper functions to upstream functions. sem.check is the backbone of the package and takes models and keys inputs and runs the models; check.cache generates tests for whether models need to be run when check = TRUE.

Matching the philosophy of the package, sem.check 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.

sem.check includes functionality designed to save time re-running code. To do this, when save_out = TRUE and a cache directory has been set, various inputs and outputs from the function call are saved, and, when check = TRUE, check.cache is used (within sem.check) to 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 load the previous output 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, both functions 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, orthogonal, ordered, and seed 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). However, the functionality may be beneficial in cases with lots of models or very slow 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

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
keys <- list(
  grit_p = paste0("grit_p_", 1:6),
  grit_c = paste0("grit_c_", 1:6),
  hope_a = paste0("hope_a_", 1:4),
  hope_p = paste0("hope_p_", 1:4)
)
# 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[[2]] <- paste0(mods[[2]], "\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",
  progress = TRUE,
  miss = "default",
  est = "default",
  name = "cors",
  check = FALSE,
```

```
  save_out = FALSE
)
```

Arguments

data	A data frame or object coercible to a data frame. 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. If `TRUE`, model fit measures are included in the output. If `FALSE`, model fit measures are 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`.
progress	If `TRUE`, messages showing progress on individual models are printed. If `FALSE`, general messages on progress and warning and error messages for individual models are printed. Defaults to `TRUE`. By default, lots of models will result in extensive messages being printed. To include only bare-bones progress (e.g., "Fitting models", "Generating parameter estimates"), errors, and warnings, set `progress = 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. If `TRUE`, inputs are compared to previous inputs and, if they exist, the model will not be rerun if nothing has changed. If `FALSE`, the model will be run regardless of the existence of previous inputs.

save_out — Logical. If TRUE, model code, a hash of the data, input parameter values, and output are saved. If FALSE, nothing is saved. Selecting save_out = TRUE enables the function to not rerun models next time if check = TRUE 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 items and fit_x latent variables. 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) (see the Interpretational Confounding section below). When CFA input models include correlated residuals, these are included as fixed values when nagy = FALSE and as free parameters when nagy = TRUE. Although the effect of correlated residuals on model performance has not been tested with Nagy and colleagues' method, they have been included because it is likely the best option (i.e., better than employing Burt's method, omitting them, or ignoring interpretational confounding).

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.

Which method should be used? Burt's method is faster but artificially constrains parameters, thereby biasing standard errors and model fit indices. Nagy's method is slower, but produces accurate standard errors and fit statistics as well as correlations between item residuals and factors (noted above). Both methods should, however, produce very similar point estimates for correlations. Therefore, Nagy and colleagues' method should be preferred whenever confidence intervals, model fit, or residuals' correlations with factors matter, except, perhaps, for large sets of variables. Burt's method should be preferred whenever multi-factor models are specified, although the function will make this change automatically for such models when nagy = TRUE. Either can be used in other cases.

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).

Non-Positive Definite Correlation Matrix 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). In such cases, 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) so will rarely or never make any material difference.

save_out and check

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.

Interpretational Confounding

In Structural Equation Models (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) labelled "interpretational confounding".

There are various ways to deal with interpretational confounding. The standard solution (other than ignoring it) is to create measurement models with good fit first, then freely estimate the structural model with checks to ensure adequate fit of the model and that measurement parameters do not change substantially with different combinations of factors. This is sometimes a good solution, but if, for example, the measurement model was for a well-established scale and it requires changing, then it loses easy comparison with past research.

An alternative solution, proposed by Burt (1976), is to fix measurement model parameters in the 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—by fixing measurement parameters, uncertainty in their estimation is neglected (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 each factor's items' residuals and each external factor are minimised, measurement parameters 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. Unfortunately, estimating these models becomes increasingly slow with more items and factors, and the method only works with correlations, not regressions, so some method to run regressions using the correlations needs to be implemented whenever more complicated structural models are required. These methods will typically bias the estimates due to ignored uncertainty in the correlation estimates, thereby undermining the primary benefit of the method.

A final solution to interpretation confounding was proposed by Rosseel and Loh (2022) with their "Structural-After-Measurement" (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, thereby preserving 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 to account for the uncertainty in their measurement.

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
keys <- list(
  grit_p = paste0("grit_p_", 1:6),
  grit_c = paste0("grit_c_", 1:6),
  hope_a = paste0("hope_a_", 1:4),
  hope_p = paste0("hope_p_", 1:4)
)
# 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 <- c("bfi_e1_1", "bfi_a1_1", "bfi_c1_1", "bfi_n1_1", "bfi_o1_1")
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",
  progress = TRUE,
  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 data frame or object coercible to a data frame. 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`	A string of lavaan code to be appended to the model. The argument can be used to set constraints, fix parameters to specific values, or allow correlations between item residuals.

`fit_save` Logical. If `TRUE`, model fit measures are included in the output. If `FALSE`, model fit measures are 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`.

`progress` If `TRUE`, messages showing progress on individual models are printed. If `FALSE`, general messages on progress and warning and error messages for individual models are printed. Defaults to `TRUE`. By default, lots of models will result in extensive messages being printed. To include only bare-bones progress (e.g., "Fitting models", "Generating parameter estimates"), errors, and warnings, set `progress = 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. If `TRUE`, inputs are compared to previous inputs and, if they exist, the model will not be rerun if nothing has changed. If `FALSE`, the model will be run regardless of the existence of previous inputs.

`save_out` Logical. If `TRUE`, model code, a hash of the data, input parameter values, and output are saved. If `FALSE`, nothing is saved. Selecting `save_out = TRUE` enables the function to not rerun models next time if `check = TRUE` 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. Specifically, `sem.path` uses the local SAM method not the global SAM method because Rosseel and Loh (2022) "recommend local SAM over global SAM whenever possible." (p. 21 of the pre-print version).

Like lavaan, `sem.path` models assume 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.

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 and, by extension, sem.path. Therefore, sem.path should *NOT* be used with bi-factor 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).

Interpretational Confounding

In Structural Equation Models (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) labelled "interpretational confounding".

There are various ways to deal with interpretational confounding. The standard solution (other than ignoring it) is to create measurement models with good fit first, then freely estimate the structural model with checks to ensure adequate fit of the model and that measurement parameters do not change substantially with different combinations of factors. This is sometimes a good solution, but if, for example, the measurement model was for a well-established scale and it requires changing, then it loses easy comparison with past research.

An alternative solution, proposed by Burt (1976), is to fix measurement model parameters in the 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—by fixing measurement parameters, uncertainty in their estimation is neglected (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 each factor's items' residuals and each external factor are minimised, measurement parameters 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. Unfortunately, estimating these models becomes increasingly slow with more items and factors, and the method only works with correlations, not regressions, so some method to run regressions using the correlations needs to be implemented whenever more complicated structural models are required. These methods will typically bias the estimates due to ignored uncertainty in the correlation estimates, thereby undermining the primary benefit of the method.

A final solution to interpretation confounding was proposed by Rosseel and Loh (2022) with their "Structural-After-Measurement" (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, thereby preserving 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 to account for the uncertainty in their measurement.

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
keys <- list(
  grit_p = paste0("grit_p_", 1:6),
  grit_c = paste0("grit_c_", 1:6),
  hope_a = paste0("hope_a_", 1:4),
  hope_p = paste0("hope_p_", 1:4)
)
# 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

∗ **datasets**
    BFIGritHope, 2

BFIGritHope, 2
bifactor.from.keys, 4, *11*, *31*
bifactor.from.keys(), *27*

cache.clean, 8, *11*
cache.setup, *8, 9*, 10, *30, 31*
cache.setup(), *26, 27*
cfa.from.keys, *11*, 12, *31*
cfa.from.keys(), *27*
check.cache *(*sem.check*)*, 27

efa.from.keys, *11*, 14, *31*
esem.from.keys, *11*, 17, *31*
esem.from.mods, *11*, *16*, *31*
esem.from.mods *(*esem.from.keys*)*, 17

getwd, *10, 11*
grep, *9*

here::here, *10, 11*

lavaan::efa, *29*, *31*
lavaan::fitMeasures, *5*, *12*, *15*, *18*, *24*, *29*, *31, 32*, *37*
lavaan::lavOptions, *6*, *13*, *15*, *18*, *24*, *29*, *31, 32*, *37*
lavaan::parameterEstimates, *31*
lavaan::sam, *20, 21*, *30, 31*, *39*
lavaan::sem, *6, 7*, *13–16*, *24*, *29–31*, *35*
lavaan::standardizedSolution, *31*

Matrix::nearPD, *34, 35*
matrixcalc::is.positive.definite, *35*

openssl::md5, *30, 31*

r2.change, 22, *30*

sem.check, *6, 7*, *11*, *13, 14*, *16*, *19*, *21*, 27, *34, 35*, *37–39*
sem.check(), *27*
sem.cor, *11*, *31*, 31
sem.path, *11*, *31*, 36
set.seed(), *24*

tools::R_user_dir, *11*