Package 'cSEM'

July 22, 2025

Title Composite-Based Structural Equation Modeling

Version 0.6.1

Date 2025-05-15

Maintainer Florian Schuberth <f.schuberth@utwente.nl>

Depends R (>= 3.5.0)

Description Estimate, assess, test, and study linear, nonlinear, hierarchical and multigroup structural equation models using composite-based approaches and procedures, including estimation techniques such as partial least squares path modeling (PLS-PM) and its derivatives (PLSc, ordPLSc, robustPLSc), generalized structured component analysis (GSCA), generalized structured component analysis with uniqueness terms (GSCAm), generalized canonical correlation analysis (GCCA), principal component analysis (PCA), factor score regression (FSR) using sum score, regression or Bartlett scores (including bias correction using Croon's approach), as well as several tests and typical postestimation procedures (e.g., verify admissibility of the estimates, assess the model fit, test the model fit etc.).

BugReports https://github.com/FloSchuberth/cSEM/issues/

URL https://github.com/FloSchuberth/cSEM/, https://floschuberth.github.io/cSEM/

License GPL-3

Encoding UTF-8

NeedsCompilation yes

LazyData true

Imports alabama, cli, crayon, expm (>= 0.999-5), future.apply, future, lifecycle, lavaan, magrittr, MASS, Matrix, matrixcalc, matrixStats, polycor, progressr, psych, purrr, Rdpack, rlang, stats, symmoments, TruncatedNormal, utils

RdMacros Rdpack, lifecycle

RoxygenNote 7.3.2

Suggests DiagrammeR, DiagrammeRsvg, dplyr, tidyr, knitr, nnls, prettydoc, plotly, rsvg, rmarkdown, rootSolve, listviewer, testthat (>= 3.0.0), ggplot2, openxlsx, graphics, spelling

Config/testthat/edition 3

VignetteBuilder knitr

Language en-US

Author Manuel E. Rademaker [aut] (ORCID: <https://orcid.org/0000-0002-8902-3561>), Florian Schuberth [aut, cre] (ORCID: <https://orcid.org/0000-0002-2110-9086>), Tamara Schamberger [ctb] (ORCID: <https://orcid.org/0000-0002-7845-784X>), Michael Klesel [ctb] (ORCID: <https://orcid.org/0000-0002-2884-1819>), Huu Phuc Nguyen [ctb] (ORCID: <https://orcid.org/0009-0000-9344-7132>), Theo K. Dijkstra [ctb], Jörg Henseler [ctb] (ORCID: <https://orcid.org/0000-0002-9736-3048>)

Repository CRAN

Date/Publication 2025-05-16 09:40:14 UTC

Contents

`Anime` *Data: Anime*

Description

A data frame with 183 observations and 13 variables.

Usage

```
Anime
```

Format

An object of class `data.frame` with 183 rows and 13 columns.

Details

The data set for the example on github.com/ISS-Analytics/pls-predict/ with irrelevant variables removed.

Source

Original source: github.com/ISS-Analytics/pls-predict/

`args_default` *Show argument defaults or candidates*

Description

Show all arguments used by package functions including default or candidate values. For argument descriptions see: csem_arguments.

Usage

```
args_default(.choices = FALSE)
```

Arguments

`.choices`	Logical. Should candidate values for the arguments be returned? Defaults to `FALSE`.

Details

By default `args_default()`returns a list of default values by argument name. If the list of accepted candidate values is required instead, use `.choices = TRUE`.

Value

A named list of argument names and defaults or accepted candidates.

See Also

`handleArgs()`, csem_arguments, `csem()`, `foreman()`

assess *Assess model*

Description

[Maturing]

Usage

```
assess(
  .object              = NULL,
  .quality_criterion   = c("all", "aic", "aicc", "aicu", "bic", "fpe", "gm", "hq",
                           "hqc", "mallows_cp", "ave",
                           "rho_C", "rho_C_mm", "rho_C_weighted",
                           "rho_C_weighted_mm", "dg", "dl", "dml", "df",
                  "effects", "f2", "fl_criterion", "chi_square", "chi_square_df",
                           "cfi", "cn", "gfi", "ifi", "nfi", "nnfi",
                           "reliability",
                           "rmsea", "rms_theta", "srmr",
                           "gof", "htmt", "htmt2", "r2", "r2_adj",
                           "rho_T", "rho_T_weighted", "vif",
                           "vifmodeB"),
  .only_common_factors = TRUE,
  ...
)
```

Arguments

`.object`	An R object of class cSEMResults resulting from a call to `csem()`.
`.quality_criterion`	Character string. A single character string or a vector of character strings naming the quality criterion to compute. See the Details section for a list of possible candidates. Defaults to "*all*" in which case all possible quality criteria are computed.
`.only_common_factors`	Logical. Should only concepts modeled as common factors be included when calculating one of the following quality criteria: AVE, the Fornell-Larcker criterion, HTMT, and all reliability estimates. Defaults to `TRUE`.
`...`	Further arguments passed to functions called by `assess()`. See args_assess_dotdotdot for a complete list of available arguments.

Details

Assess a model using common quality criteria. See the Postestimation: Assessing a model article on the cSEM website for details.

The function is essentially a wrapper around a number of internal functions that perform an "assessment task" (called a **quality criterion** in **cSEM** parlance) like computing reliability estimates, the effect size (Cohen's f^2), the heterotrait-monotrait ratio of correlations (HTMT) etc.

By default every possible quality criterion is calculated (`.quality_criterion = "all"`). If only a subset of quality criteria are needed a single character string or a vector of character strings naming the criteria to be computed may be supplied to `assess()` via the `.quality_criterion` argument. Currently, the following quality criteria are implemented (in alphabetical order):

Average variance extracted (AVE); "ave" An estimate of the amount of variation in the indicators that is due to the underlying latent variable. Practically, it is calculated as the ratio of the (indicator) true score variances (i.e., the sum of the squared loadings) relative to the sum of the total indicator variances. The AVE is inherently tied to the common factor model. It is therefore unclear how to meaningfully interpret AVE results for constructs modeled as composites. It is possible to report the AVE for constructs modeled as composites by setting `.only_common_factors = FALSE`, however, result should be interpreted with caution as they may not have a conceptual meaning. Calculation is done by `calculateAVE()`.

Congeneric reliability; "rho_C", "rho_C_mm", "rho_C_weighted", "rho_C_weighted_mm" An estimate of the reliability assuming a congeneric measurement model (i.e., loadings are allowed to differ) and a test score (proxy) based on unit weights. There are four different versions implemented. See the Methods and Formulae section of the Postestimation: Assessing a model article on the cSEM website for details. Alternative but synonymous names for `"rho_C"` are: composite reliability, construct reliability, reliability coefficient, Jöreskog's rho, coefficient omega, or Dillon-Goldstein's rho. For `"rho_C_weighted"`: (Dijkstra-Henselers) rhoA. `rho_C_mm` and `rho_C_weighted_mm` have no corresponding names. The former uses unit weights scaled by (w'Sw)^(-1/2) and the latter weights scaled by (w'Sigma_hat w)^(-1/2) where Sigma_hat is the model-implied indicator correlation matrix. The Congeneric reliability is inherently tied to the common factor model. It is therefore unclear how to meaningfully interpret congeneric reliability estimates for constructs modeled as composites. It is possible to report the congeneric reliability for constructs modeled as composites by setting `.only_common_factors = FALSE`, however, result should be interpreted with caution as they may not have a conceptual meaning. Calculation is done by `calculateRhoC()`.

Distance measures; "dg", "dl", "dml" Measures of the distance between the model-implied and the empirical indicator correlation matrix. Currently, the geodesic distance (`"dg"`), the squared Euclidean distance (`"dl"`) and the the maximum likelihood-based distance function are implemented (`"dml"`). Calculation is done by `calculateDL()`, `calculateDG()`, and `calculateDML()`.

Degrees of freedom, "df" Returns the degrees of freedom. Calculation is done by `calculateDf()`.

Effects; "effects" Total and indirect effect estimates. Additionally, the variance accounted for (VAF) is computed. The VAF is defined as the ratio of a variables indirect effect to its total effect. Calculation is done by `calculateEffects()`.

Effect size; "f2" An index of the effect size of an independent variable in a structural regression equation. This measure is commonly known as Cohen's f^2. The effect size of the k'th independent variable in this case is defined as the ratio (R2_included - R2_excluded)/(1 - R2_included), where R2_included and R2_excluded are the R squares of the original structural model regression equation (R2_included) and the alternative specification with the k'th variable dropped (R2_excluded). Calculation is done by `calculatef2()`.

Fit indices; "chi_square", "chi_square_df", "cfi", "cn", "gfi", "ifi", "nfi", "nnfi", "rmsea", "rms_theta", "srmr" Several absolute and incremental fit indices. Note that their suitability for models containing

constructs modeled as composites is still an open research question. Also note that fit indices are not tests in a hypothesis testing sense and decisions based on common one-size-fits-all cutoffs proposed in the literature suffer from serious statistical drawbacks. Calculation is done by `calculateChiSquare()`, `calculateChiSquareDf()`, `calculateCFI()`, `calculateGFI()`, `calculateIFI()`, `calculateNFI()`, `calculateNNFI()`, `calculateRMSEA()`, `calculateRMSTheta()` and `calculateSRMR()`.

Fornell-Larcker criterion; "fl_criterion" A rule suggested by Fornell and Larcker (1981) to assess discriminant validity. The Fornell-Larcker criterion is a decision rule based on a comparison between the squared construct correlations and the average variance extracted. FL returns a matrix with the squared construct correlations on the off-diagonal and the AVEs on the main diagonal. Calculation is done by `calculateFLCriterion()`.

Goodness of Fit (GoF); "gof" The GoF is defined as the square root of the mean of the R squares of the structural model times the mean of the variances in the indicators that are explained by their related constructs (i.e., the average over all lambda^2_k). For the latter, only constructs modeled as common factors are considered as they explain their indicator variance in contrast to a composite where indicators actually build the construct. Note that, contrary to what the name suggests, the GoF is **not** a measure of model fit in a Chi-square fit test sense. Calculation is done by `calculateGoF()`.

Heterotrait-monotrait ratio of correlations (HTMT); "htmt" An estimate of the correlation between latent variables assuming tau equivalent measurement models. The HTMT is used to assess convergent and/or discriminant validity of a construct. The HTMT is inherently tied to the common factor model. If the model contains less than two constructs modeled as common factors and `.only_common_factors = TRUE`, `NA` is returned. It is possible to report the HTMT for constructs modeled as composites by setting `.only_common_factors = FALSE`, however, result should be interpreted with caution as they may not have a conceptual meaning. Calculation is done by `calculateHTMT()`.

HTMT2; "htmt2" An estimate of the correlation between latent variables assuming congeneric measurement models. The HTMT2 is used to assess convergent and/or discriminant validity of a construct. The HTMT is inherently tied to the common factor model. If the model contains less than two constructs modeled as common factors and `.only_common_factors = TRUE`, `NA` is returned. It is possible to report the HTMT for constructs modeled as composites by setting `.only_common_factors = FALSE`, however, result should be interpreted with caution as they may not have a conceptual meaning. Calculation is done by `calculateHTMT()`.

Model selection criteria: "aic", "aicc", "aicu", "bic", "fpe", "gm", "hq", "hqc", "mallows_cp" Several model selection criteria as suggested by Sharma et al. (2019) in the context of PLS. See: `calculateModelSelectionCriteria()` for details.

Reliability: "reliability" As described in the Methods and Formulae section of the Postestimation: Assessing a model article on the cSEM website there are many different estimators for the (internal consistency) reliability. Choosing `.quality_criterion = "reliability"` computes the three most common measures, namely: "Cronbach's alpha" (identical to "rho_T"), "Jöreskog's rho" (identical to "rho_C_mm"), and "Dijkstra-Henseler's rho A" (identical to "rho_C_weighted_mm"). Reliability is inherently tied to the common factor model. It is therefore unclear how to meaningfully interpret reliability estimates for constructs modeled as composites. It is possible to report the three common reliability estimates for constructs modeled as composites by setting `.only_common_factors = FALSE`, however, result should be interpreted with caution as they may not have a conceptual meaning.

R square and R square adjusted; "r2", "r2_adj" The R square and the adjusted R square for each structural regression equation. Calculated when running `csem()`.

Tau-equivalent reliability; "rho_T" An estimate of the reliability assuming a tau-equivalent measurement model (i.e. a measurement model with equal loadings) and a test score (proxy) based on unit weights. Tau-equivalent reliability is the preferred name for reliability estimates that assume a tau-equivalent measurement model such as Cronbach's alpha. The tau-equivalent reliability (Cronbach's alpha) is inherently tied to the common factor model. It is therefore unclear how to meaningfully interpret tau-equivalent reliability estimates for constructs modeled as composites. It is possible to report tau-equivalent reliability estimates for constructs modeled as composites by setting `.only_common_factors = FALSE`, however, result should be interpreted with caution as they may not have a conceptual meaning. Calculation is done by `calculateRhoT()`.

Variance inflation factors (VIF); "vif" An index for the amount of (multi-)collinearity between independent variables of a regression equation. Computed for each structural equation. Practically, VIF_k is defined as the ratio of 1 over (1 - R2_k) where R2_k is the R squared from a regression of the k'th independent variable on all remaining independent variables. Calculated when running `csem()`.

Variance inflation factors for PLS-PM mode B (VIF-ModeB); "vifmodeB" An index for the amount of (multi-)collinearity between independent variables (indicators) in mode B regression equations. Computed only if `.object` was obtained using `.weight_approach = "PLS-PM"` and at least one mode was mode B. Practically, VIF-ModeB_k is defined as the ratio of 1 over (1 - R2_k) where R2_k is the R squared from a regression of the k'th indicator of block j on all remaining indicators of the same block. Calculation is done by `calculateVIFModeB()`.

For details on the most important quality criteria see the Methods and Formulae section of the Postestimation: Assessing a model article on the on the cSEM website.

Some of the quality criteria are inherently tied to the classical common factor model and therefore only meaningfully interpreted within a common factor model (see the Postestimation: Assessing a model article for details). It is possible to force computation of all quality criteria for constructs modeled as composites by setting `.only_common_factors = FALSE`, however, we explicitly warn to interpret quality criteria in analogy to the common factor model in this case, as the interpretation often does not carry over to composite models.

Resampling: To resample a given quality criterion supply the name of the function that calculates the desired quality criterion to `csem()`'s `.user_funs` argument. See `resamplecSEMResults()` for details.

Value

A named list of quality criteria. Note that if only a single quality criteria is computed the return value is still a list!

See Also

`csem()`, `resamplecSEMResults()`, `exportToExcel()`

Examples

```
# ============================================================================
# Using the three common factors dataset
# ============================================================================
model <- "
# Structural model
eta2 ~ eta1
eta3 ~ eta1 + eta2

# Each concept is measured by 3 indicators, i.e., modeled as latent variable
eta1 =~ y11 + y12 + y13
eta2 =~ y21 + y22 + y23
eta3 =~ y31 + y32 + y33
"

res <- csem(threecommonfactors, model)
a   <- assess(res) # computes all quality criteria (.quality_criterion = "all")
a

## The return value is a named list. Type for example:
a$HTMT

# You may also just compute a subset of the quality criteria
assess(res, .quality_criterion = c("ave", "rho_C", "htmt"))

## Resampling ------------------------------------------------------------------
# To resample a given quality criterion use csem()'s .user_funs argument
# Note: The output of the quality criterion needs to be a vector or a matrix.
#       Matrices will be vectorized columnwise.
res <- csem(threecommonfactors, model,
            .resample_method = "bootstrap",
            .R               = 40,
            .user_funs       = cSEM:::calculateSRMR
)

## Look at the resamples
res$Estimates$Estimates_resample$Estimates1$User_fun$Resampled[1:4, ]

## Use infer() to compute e.g., the 95% percentile confidence interval
res_infer <- infer(res, .quantity = "CI_percentile")

## The results are saved under the name "User_fun"
res_infer$User_fun

## Several quality criteria can be resampled simultaneously
res <- csem(threecommonfactors, model,
            .resample_method = "bootstrap",
            .R               = 40,
            .user_funs       = list(
              "SRMR" = cSEM:::calculateSRMR,
              "RMS_theta" = cSEM:::calculateRMSTheta
            ),
```

```
            .tolerance = 1e-04
)
res$Estimates$Estimates_resample$Estimates1$SRMR$Resampled[1:4, ]
res$Estimates$Estimates_resample$Estimates1$RMS_theta$Resampled[1:4]
```

`Benitezetal2020` *Data: Benitezetal2020*

Description

A data frame containing 22 variables with 300 observations.

Usage

```
Benitezetal2020
```

Format

An object of class `data.frame` with 300 rows and 22 columns.

Details

The simulated data contains variables about the social executive and employee behavior. Moreover, it contains variables about the social media capability and business performance. The dataset was used as an illustrative example in Benitez et al. (2020).

Source

The dataset is provided as supplementary material by Benitez et al. (2020).

References

Benitez J, Henseler J, Castillo A, Schuberth F (2020). "How to perform and report an impactful analysis using partial least squares: Guidelines for confirmatory and explanatory IS research." *Information & Management*, **2**(57), 103168. doi:10.1016/j.im.2019.05.003.

Examples

```
#============================================================================
# Example is taken from Benitez et al. (2020)
#============================================================================
model_Benitez <-"
# Reflective measurement models# Reflective measurement models
SEXB =~ SEXB1 + SEXB2 + SEXB3 +SEXB4
SEMB =~ SEMB1 + SEMB2 + SEMB3 + SEMB4

# Composite models
SMC <~ SMC1 + SMC2 + SMC3 + SMC4
```

```
BPP <~ BPP1 + BPP2 + BPP3 + BPP4 + BPP5

# Control variables
FS<~ FirmSize
Ind <~ Industry1 + Industry2 + Industry3

# Structural model
SMC ~ SEXB + SEMB
BPP ~ SMC + Ind + FS
"

out <- csem(.data = Benitezetal2020, .model = model_Benitez,
            .PLS_weight_scheme_inner = 'factorial',
            .tolerance = 1e-06)
```

`BergamiBagozzi2000` *Data: BergamiBagozzi2000*

Description

A data frame containing 22 variables with 305 observations.

Usage

```
BergamiBagozzi2000
```

Format

An object of class `data.frame` with 305 rows and 22 columns.

Details

The dataset contains 22 variables and originates from a larger survey among South Korean employees conducted and reported by Bergami and Bagozzi (2000). It is also used in Hwang and Takane (2004) and Henseler (2021) for demonstration purposes, see the corresponding tutorial.

Source

Survey among South Korean employees conducted and reported by Bergami and Bagozzi (2000).

References

Bergami M, Bagozzi RP (2000). "Self-categorization, affective commitment and group self-esteem as distinct aspects of social identity in the organization." *British Journal of Social Psychology*, **39**(4), 555–577. doi:10.1348/014466600164633.

Henseler J (2021). *Composite-Based Structural Equation Modeling: Analyzing Latent and Emergent Variables*. Guilford Press, New York.

Hwang H, Takane Y (2004). "Generalized Structured Component Analysis." *Psychometrika*, **69**(1), 81–99.

Examples

```
#===========================================================================
# Example is taken from Henseler (2021)
#===========================================================================
model_Bergami_Bagozzi_Henseler="
# Measurement models
OrgPres =~ cei1 + cei2 + cei3 + cei4 + cei5 + cei6 + cei7 + cei8
OrgIden =~ ma1 + ma2 + ma3 + ma4 + ma5 + ma6
AffLove =~ orgcmt1 + orgcmt2 + orgcmt3 + orgcmt7
AffJoy  =~ orgcmt5 + orgcmt8
Gender  <~ gender

# Structural model
OrgIden ~ OrgPres
AffLove ~ OrgPres + OrgIden + Gender
AffJoy  ~ OrgPres + OrgIden + Gender
"

out <- csem(.data = BergamiBagozzi2000,
            .model = model_Bergami_Bagozzi_Henseler,
            .PLS_weight_scheme_inner = 'factorial',
            .tolerance = 1e-06
)

#===========================================================================
# Example is taken from Hwang et al. (2004)
#===========================================================================

model_Bergami_Bagozzi_Hwang="
# Measurement models
OrgPres =~ cei1 + cei2 + cei3 + cei4 + cei5 + cei6 + cei7 + cei8
OrgIden =~ ma1 + ma2 + ma3 + ma4 + ma5 + ma6
AffJoy =~ orgcmt1 + orgcmt2 + orgcmt3 + orgcmt7
AffLove  =~ orgcmt5 + orgcmt6 + orgcmt8

# Structural model
OrgIden ~ OrgPres
AffLove ~ OrgIden
AffJoy  ~ OrgIden"

out_Hwang <- csem(.data = BergamiBagozzi2000,
                  .model = model_Bergami_Bagozzi_Hwang,
                  .approach_weights = "GSCA",
                  .disattenuate = FALSE,
                  .id = "gender",
                  .tolerance = 1e-06)
```

`calculateAVE` *Average variance extracted (AVE)*

Description

Calculate the average variance extracted (AVE) as proposed by Fornell and Larcker (1981). For details see the cSEM website

Usage

```
calculateAVE(
 .object              = NULL,
 .only_common_factors = TRUE
)
```

Arguments

`.object`	An R object of class cSEMResults resulting from a call to `csem()`.
`.only_common_factors`	Logical. Should only concepts modeled as common factors be included when calculating one of the following quality criteria: AVE, the Fornell-Larcker criterion, HTMT, and all reliability estimates. Defaults to `TRUE`.

Details

The AVE is inherently tied to the common factor model. It is therefore unclear how to meaningfully interpret the AVE in the context of a composite model. It is possible, however, to force computation of the AVE for constructs modeled as composites by setting `.only_common_factors = FALSE`.

Value

A named vector of numeric values (the AVEs). If `.object` is a list of `cSEMResults` objects, a list of AVEs is returned.

References

Fornell C, Larcker DF (1981). "Evaluating structural equation models with unobservable variables and measurement error." *Journal of Marketing Research*, **XVIII**, 39–50.

See Also

`assess()`, cSEMResults

`calculateDf` *Degrees of freedom*

Description

Calculate the degrees of freedom for a given model from a cSEMResults object.

Usage

```
calculateDf(
  .object     = NULL,
  .null_model = FALSE,
  ...
  )
```

Arguments

`.object`	An R object of class cSEMResults resulting from a call to `csem()`.
`.null_model`	Logical. Should the degrees of freedom for the null model be computed? Defaults to `FALSE`.
`...`	Ignored.

Details

Although, composite-based estimators always retrieve parameters of the postulated models via the estimation of a composite model, the computation of the degrees of freedom depends on the postulated model.

See: cSEM website for details on how the degrees of freedom are calculated.

To compute the degrees of freedom of the null model use `.null_model = TRUE`. The degrees of freedom of the null model are identical to the number of non-redundant off-diagonal elements of the empirical indicator correlation matrix. This implicitly assumes a null model with model-implied indicator correlation matrix equal to the identity matrix.

Value

A single numeric value.

See Also

`assess()`, cSEMResults

calculatef2 *Calculate Cohen's f^2*

Description

Calculate the effect size for regression analysis (Cohen 1992) known as Cohen's f^2.

Usage

```
calculatef2(.object = NULL)
```

Arguments

.object	An R object of class cSEMResults resulting from a call to csem().

Value

A matrix with as many rows as there are structural equations. The number of columns is equal to the total number of right-hand side variables of these equations.

References

Cohen J (1992). "A power primer." *Psychological Bulletin*, **112**(1), 155–159.

See Also

assess(), csem, cSEMResults

calculateFLCriterion *Fornell-Larcker criterion*

Description

Computes the Fornell-Larcker matrix.

Usage

```
calculateFLCriterion(
  .object              = NULL,
  .only_common_factors = TRUE,
  ...
  )
```

Arguments

`.object`	An R object of class cSEMResults resulting from a call to `csem()`.
`.only_common_factors`	Logical. Should only concepts modeled as common factors be included when calculating one of the following quality criteria: AVE, the Fornell-Larcker criterion, HTMT, and all reliability estimates. Defaults to `TRUE`.
`...`	Ignored.

Details

The Fornell-Larcker criterion (FL criterion) is a rule suggested by Fornell and Larcker (1981) to assess discriminant validity. The Fornell-Larcker criterion is a decision rule based on a comparison between the squared construct correlations and the average variance extracted (AVE).

The FL criterion is inherently tied to the common factor model. It is therefore unclear how to meaningfully interpret the FL criterion in the context of a model that contains constructs modeled as composites.

Value

A matrix with the squared construct correlations on the off-diagonal and the AVEs on the main diagonal.

References

Fornell C, Larcker DF (1981). "Evaluating structural equation models with unobservable variables and measurement error." *Journal of Marketing Research*, **XVIII**, 39–50.

See Also

`assess()`, cSEMResults

`calculateGoF` *Goodness of Fit (GoF)*

Description

Calculate the Goodness of Fit (GoF) proposed by Tenenhaus et al. (2004). Note that, contrary to what the name suggests, the GoF is **not** a measure of model fit in the sense of SEM. See e.g. Henseler and Sarstedt (2012) for a discussion.

Usage

```
calculateGoF(
 .object              = NULL
)
```

Arguments

`.object`	An R object of class cSEMResults resulting from a call to `csem()`.

Details

The GoF is inherently tied to the common factor model. It is therefore unclear how to meaningfully interpret the GoF in the context of a model that contains constructs modeled as composites.

Value

A single numeric value.

References

Henseler J, Sarstedt M (2012). "Goodness-of-fit Indices for Partial Least Squares Path Modeling." *Computational Statistics*, **28**(2), 565–580. doi:10.1007/s0018001203171.

Tenenhaus M, Amanto S, Vinzi VE (2004). "A Global Goodness-of-Fit Index for PLS Structural Equation Modelling." In *Proceedings of the XLII SIS Scientific Meeting*, 739–742.

See Also

`assess()`, cSEMResults

`calculateHTMT` *HTMT*

Description

Computes either the heterotrait-monotrait ratio of correlations (HTMT) based on Henseler et al. (2015) or the HTMT2 proposed by Roemer et al. (2021). While the HTMT is a consistent estimator for the construct correlation in case of tau-equivalent measurement models, the HTMT2 is a consistent estimator for congeneric measurement models. In general, they are used to assess discriminant validity.

Usage

```
calculateHTMT(
 .object              = NULL,
 .type_htmt           = c('htmt','htmt2'),
 .absolute            = TRUE,
 .alpha               = 0.05,
 .ci                  = c("CI_percentile", "CI_standard_z", "CI_standard_t",
                          "CI_basic", "CI_bc", "CI_bca", "CI_t_interval"),
 .inference           = FALSE,
 .only_common_factors = TRUE,
 .R                   = 499,
```

```
  .seed                 = NULL,
  ...
)
```

Arguments

`.object`	An R object of class cSEMResults resulting from a call to `csem()`.
`.type_htmt`	Character string indicating the type of HTMT that should be calculated, i.e., the original HTMT ("*htmt*") or the HTMT2 ("*htmt2*"). Defaults to "*htmt*"
`.absolute`	Logical. Should the absolute HTMT values be returned? Defaults to `TRUE` .
`.alpha`	A numeric value giving the significance level. Defaults to `0.05`.
`.ci`	A character strings naming the type of confidence interval to use to compute the 1-alpha% quantile of the bootstrap HTMT values. For possible choices see `infer()`. Ignored if `.inference = FALSE`. Defaults to "*CI_percentile*".
`.inference`	Logical. Should critical values be computed? Defaults to `FALSE`.
`.only_common_factors`	Logical. Should only concepts modeled as common factors be included when calculating one of the following quality criteria: AVE, the Fornell-Larcker criterion, HTMT, and all reliability estimates. Defaults to `TRUE`.
`.R`	Integer. The number of bootstrap replications. Defaults to `499`.
`.seed`	Integer or `NULL`. The random seed to use. Defaults to `NULL` in which case an arbitrary seed is chosen. Note that the scope of the seed is limited to the body of the function it is used in. Hence, the global seed will not be altered!
`...`	Ignored.

Details

Computation of the HTMT/HTMT2 assumes that all intra-block and inter-block correlations between indicators are either all-positive or all-negative. A warning is given if this is not the case.

To obtain bootstrap confidence intervals for the HTMT/HTMT2 values, set `.inference = TRUE`. To choose the type of confidence interval, use `.ci`. To control the bootstrap process, arguments `.R` and `.seed` are available. Note, that `.alpha` is multiplied by two because typically researchers are interested in one-sided bootstrap confidence intervals for the HTMT/HTMT2.

Since the HTMT and the HTMT2 both assume a reflective measurement model only concepts modeled as common factors are considered by default. For concepts modeled as composites the HTMT may be computed by setting `.only_common_factors = FALSE`, however, it is unclear how to interpret values in this case.

Value

A named list containing:

- the values of the HTMT/HTMT2, i.e., a matrix with the HTMT/HTMT2 values at its lower triangular and if `.inference = TRUE` the upper triangular contains the upper limit of the 1-2*.alpha% bootstrap confidence interval if the HTMT/HTMT2 is positive and the lower limit if the HTMT/HTMT2 is negative.

- the lower and upper limits of the 1-2*.alpha% bootstrap confidence interval if `.inference = TRUE`; otherwise it is `NULL`.
- the number of admissible bootstrap runs, i.e., the number of HTMT/HTMT2 values calculated during bootstrap if `.inference = TRUE`; otherwise it is `NULL`. Note, the HTMT2 is based on the geometric and thus cannot always be calculated.

References

Henseler J, Ringle CM, Sarstedt M (2015). "A New Criterion for Assessing Discriminant Validity in Variance-based Structural Equation Modeling." *Journal of the Academy of Marketing Science*, **43**(1), 115–135. doi:10.1007/s1174701404038.

Roemer E, Schuberth F, Henseler J (2021). "HTMT2 – an improved criterion for assessing discriminant validity in structural equation modeling." *Industrial Management & Data Systems*, **121**(12), 2637–2650.

See Also

`assess()`, csem, cSEMResults

`calculateModelSelectionCriteria`
Model selection criteria

Description

Calculate several information or model selection criteria (MSC) such as the Akaike information criterion (AIC), the Bayesian information criterion (BIC) or the Hannan-Quinn criterion (HQ).

Usage

```
calculateModelSelectionCriteria(
  .object          = NULL,
  .ms_criterion    = c("all", "aic", "aicc", "aicu", "bic", "fpe", "gm", "hq",
                       "hqc", "mallows_cp"),
  .by_equation     = TRUE,
  .only_structural = TRUE
  )
```

Arguments

`.object`	An R object of class cSEMResults resulting from a call to `csem()`.
`.ms_criterion`	Character string. Either a single character string or a vector of character strings naming the model selection criterion to compute. Defaults to `"all"`.
`.by_equation`	Should the criteria be computed for each structural model equation separately? Defaults to `TRUE`.
`.only_structural`	Should the the log-likelihood be based on the structural model? Ignored if `.by_equation == TRUE`. Defaults to `TRUE`.

Details

By default, all criteria are calculated (`.ms_criterion == "all"`). To compute only a subset of the criteria a vector of criteria may be given.

If `.by_equation == TRUE` (the default), the criteria are computed for each structural equation of the model separately, as suggested by Sharma et al. (2019) in the context of PLS. The relevant formula can be found in Table B1 of the appendix of Sharma et al. (2019).

If `.by_equation == FALSE` the AIC, the BIC and the HQ for whole model are calculated. All other criteria are currently ignored in this case! The relevant formula are (see, e.g., (Akaike 1974), Schwarz (1978), Hannan and Quinn (1979)):

$$AIC = -2 * log(L) + 2 * k$$

$$BIC = -2 * log(L) + k * ln(n)$$

$$HQ = -2 * log(L) + 2 * k * ln(ln(n))$$

where log(L) is the log likelihood function of the multivariate normal distribution of the observable variables, k the (total) number of estimated parameters, and n the sample size.

If `.only_structural == TRUE`, log(L) is based on the structural model only. The argument is ignored if `.by_equation == TRUE`.

Value

If `.by_equation == TRUE` a named list of model selection criteria.

References

Akaike H (1974). "A New Look at the Statistical Model Identification." *IEEE Transactions on Automatic Control*, **19**(6), 716–723.

Hannan EJ, Quinn BG (1979). "The Determination of the order of an autoregression." *Journal of the Royal Statistical Society: Series B (Methodological)*, **41**(2), 190–195.

Schwarz G (1978). "Estimating the Dimension of a Model." *The Annals of Statistics*, **6**(2), 461–464. doi:10.1214/aos/1176344136.

Sharma P, Sarstedt M, Shmueli G, Kim KH, Thiele KO (2019). "PLS-Based Model Selection: The Role of Alternative Explanations in Information Systems Research." *Journal of the Association for Information Systems*, **20**(4).

See Also

`assess()`, cSEMResults

`calculateRelativeGoF` *Relative Goodness of Fit (relative GoF)*

Description

Calculate the Relative Goodness of Fit (GoF) proposed by Vinzi et al. (2010). Note that, contrary to what the name suggests, the Relative GoF is **not** a measure of model fit in the sense of SEM. See e.g. Henseler and Sarstedt (2012) for a discussion.

Usage

```
calculateRelativeGoF(
 .object              = NULL
)
```

Arguments

`.object`	An R object of class cSEMResults resulting from a call to `csem()`.

Value

A single numeric value.

References

Henseler J, Sarstedt M (2012). "Goodness-of-fit Indices for Partial Least Squares Path Modeling." *Computational Statistics*, **28**(2), 565–580. doi:10.1007/s0018001203171.

Vinzi VE, Trinchera L, Amato S (2010). "PLS path modeling: From foundations to recent developments and open issues for model assessment and improvement." In Vinzi VE, Wang H (eds.), *Handbook of Partial Least Squares*, 47–82. Springer.

See Also

`assess()`, cSEMResults

`calculateVIFModeB` *Calculate variance inflation factors (VIF) for weights obtained by PLS Mode B*

Description

Calculate the variance inflation factor (VIF) for weights obtained by PLS-PM's Mode B.

Usage

```
calculateVIFModeB(.object = NULL)
```

Arguments

`.object`	An R object of class cSEMResults resulting from a call to `csem()`.

Details

Weight estimates obtained by Mode B can suffer from multicollinearity. VIF values are commonly used to assess the severity of multicollinearity.

The function is only applicable to objects of class `cSEMResults_default`. For other object classes use `assess()`.

Value

A named list of vectors containing the VIF values. Each list name is the name of a construct whose weights were obtained by Mode B. The vectors contain the VIF values obtained from a regression of each explanatory variable of a given construct on the remaining explanatory variables of that construct.

If the weighting approach is not `"PLS-PM"` or for none of the constructs Mode B is used, the function silently returns `NA`.

References

There are no references for Rd macro `\insertAllCites` on this help page.

See Also

`assess()`, cSEMResults

`calculateWeightsGSCA` *Calculate composite weights using GSCA*

Description

Calculate composite weights using generalized structure component analysis (GSCA). The first version of this approach was presented in Hwang and Takane (2004). Since then, several advancements have been proposed. The latest version of GSCA can been found in Hwang and Takane (2014). This is the version **cSEM**s implementation is based on.

Usage

```
calculateWeightsGSCA(
  .X                        = args_default()$.X,
  .S                        = args_default()$.S,
  .csem_model               = args_default()$.csem_model,
  .conv_criterion           = args_default()$.conv_criterion,
  .iter_max                 = args_default()$.iter_max,
  .starting_values          = args_default()$.starting_values,
  .tolerance                = args_default()$.tolerance
  )
```

Arguments

`.X`	A matrix of processed data (scaled, cleaned and ordered).
`.S`	The (K x K) empirical indicator correlation matrix.
`.csem_model`	A (possibly incomplete) cSEMModel-list.
`.conv_criterion`	Character string. The criterion to use for the convergence check. One of: "*diff_absolute*", "*diff_squared*", or "*diff_relative*". Defaults to "*diff_absolute*".
`.iter_max`	Integer. The maximum number of iterations allowed. If `iter_max = 1` and `.approach_weights = "PLS-PM"` one-step weights are returned. If the algorithm exceeds the specified number, weights of iteration step `.iter_max - 1` will be returned with a warning. Defaults to `100`.
`.starting_values`	A named list of vectors where the list names are the construct names whose indicator weights the user wishes to set. The vectors must be named vectors of `"indicator_name" = value` pairs, where `value` is the (scaled or unscaled) starting weight. Defaults to `NULL`.
`.tolerance`	Double. The tolerance criterion for convergence. Defaults to `1e-05`.

Value

A named list. J stands for the number of constructs and K for the number of indicators.

`$W` A (J x K) matrix of estimated weights.

`$E NULL`

`$Modes` A named vector of Modes used for the outer estimation, for GSCA the mode is automatically set to "gsca".

`$Conv_status` The convergence status. `TRUE` if the algorithm has converged and `FALSE` otherwise.

`$Iterations` The number of iterations required.

References

Hwang H, Takane Y (2004). "Generalized Structured Component Analysis." *Psychometrika*, **69**(1), 81–99.

Hwang H, Takane Y (2014). *Generalized Structured Component Analysis: A Component-Based Approach to Structural Equation Modeling*, Chapman & Hall/CRC Statistics in the Social and Behavioral Sciences. Chapman and Hall/CRC.

`calculateWeightsGSCAm` *Calculate weights using GSCAm*

Description

Calculate composite weights using generalized structured component analysis with uniqueness terms (GSCAm) proposed by Hwang et al. (2017).

Usage

```
calculateWeightsGSCAm(
  .X                          = args_default()$.X,
  .csem_model                 = args_default()$.csem_model,
  .conv_criterion             = args_default()$.conv_criterion,
  .iter_max                   = args_default()$.iter_max,
  .starting_values            = args_default()$.starting_values,
  .tolerance                  = args_default()$.tolerance
  )
```

Arguments

`.X`	A matrix of processed data (scaled, cleaned and ordered).
`.csem_model`	A (possibly incomplete) cSEMModel-list.
`.conv_criterion`	Character string. The criterion to use for the convergence check. One of: "*diff_absolute*", "*diff_squared*", or "*diff_relative*". Defaults to "*diff_absolute*".
`.iter_max`	Integer. The maximum number of iterations allowed. If `iter_max = 1` and `.approach_weights = "PLS-PM"` one-step weights are returned. If the algorithm exceeds the specified number, weights of iteration step `.iter_max - 1` will be returned with a warning. Defaults to `100`.
`.starting_values`	A named list of vectors where the list names are the construct names whose indicator weights the user wishes to set. The vectors must be named vectors of `"indicator_name" = value` pairs, where `value` is the (scaled or unscaled) starting weight. Defaults to `NULL`.
`.tolerance`	Double. The tolerance criterion for convergence. Defaults to `1e-05`.

Details

If there are only constructs modeled as common factors calling `csem()` with `.appraoch_weights = "GSCA"` will automatically call `calculateWeightsGSCAm()` unless `.disattenuate = FALSE`. GSCAm currently only works for pure common factor models. The reason is that the implementation in **cSEM** is based on (the appendix) of Hwang et al. (2017). Following the appendix, GSCAm fails if there is at least one construct modeled as a composite because calculating weight estimates with GSCAm leads to a product involving the measurement matrix. This matrix does not have full rank if a construct modeled as a composite is present. The reason is that the measurement matrix has

a zero row for every construct which is a pure composite (i.e. all related loadings are zero) and, therefore, leads to a non-invertible matrix when multiplying it with its transposed.

Value

A list with the elements

`$W` A (J x K) matrix of estimated weights.

`$C` The (J x K) matrix of estimated loadings.

`$B` The (J x J) matrix of estimated path coefficients.

`$E` `NULL`

`$Modes` A named vector of Modes used for the outer estimation, for GSCA the mode is automatically set to 'gsca'.

`$Conv_status` The convergence status. `TRUE` if the algorithm has converged and `FALSE` otherwise.

`$Iterations` The number of iterations required.

References

Hwang H, Takane Y, Jung K (2017). "Generalized structured component analysis with uniqueness terms for accommodating measurement error." *Frontiers in Psychology*, **8**(2137), 1–12.

`calculateWeightsKettenring`
Calculate composite weights using GCCA

Description

Calculates composite weights according to one of the the five criteria "*SUMCORR*", "*MAXVAR*", "*SSQCORR*", "*MINVAR*", and "*GENVAR*" suggested by Kettenring (1971).

Usage

```
calculateWeightsKettenring(
  .S             = args_default()$.S,
  .csem_model    = args_default()$.csem_model,
  .approach_gcca = args_default()$.approach_gcca
  )
```

Arguments

`.S`	The (K x K) empirical indicator correlation matrix.
`.csem_model`	A (possibly incomplete) cSEMModel-list.
`.approach_gcca`	Character string. The Kettenring approach to use for GCCA. One of "*SUMCORR*", "*MAXVAR*", "*SSQCORR*", "*MINVAR*" or "*GENVAR*". Defaults to "*SUMCORR*".

Value

A named list. J stands for the number of constructs and K for the number of indicators.

`$W` A (J x K) matrix of estimated weights.

`$E` NULL

`$Modes` The GCCA mode used for the estimation.

`$Conv_status` The convergence status. `TRUE` if the algorithm has converged and `FALSE` otherwise. For `.approach_gcca = "MINVAR"` or `.approach_gcca = "MAXVAR"` the convergence status is `NULL` since both are closed-form estimators.

`$Iterations` The number of iterations required. 0 for `.approach_gcca = "MINVAR"` or `.approach_gcca = "MAXVAR"`

References

Kettenring JR (1971). "Canonical Analysis of Several Sets of Variables." *Biometrika*, **58**(3), 433–451.

`calculateWeightsPCA` *Calculate composite weights using principal component analysis (PCA)*

Description

Calculate weights for each block by extracting the first principal component of the indicator correlation matrix S_jj for each blocks, i.e., weights are the simply the first eigenvector of S_jj.

Usage

```
calculateWeightsPCA(
 .S                = args_default()$.S,
 .csem_model       = args_default()$.csem_model
 )
```

Arguments

`.S`	The (K x K) empirical indicator correlation matrix.
`.csem_model`	A (possibly incomplete) cSEMModel-list.

Value

A named list. J stands for the number of constructs and K for the number of indicators.

`$W` A (J x K) matrix of estimated weights.

`$E` NULL

`$Modes` The mode used. Always "PCA".

`$Conv_status` `NULL` as there are no iterations

`$Iterations` 0 as there are no iterations

`calculateWeightsPLS` *Calculate composite weights using PLS-PM*

Description

Calculate composite weights using the partial least squares path modeling (PLS-PM) algorithm (Wold 1975).

Usage

```
calculateWeightsPLS(
  .data                        = args_default()$.data,
  .S                           = args_default()$.S,
  .csem_model                  = args_default()$.csem_model,
  .conv_criterion              = args_default()$.conv_criterion,
  .iter_max                    = args_default()$.iter_max,
  .PLS_ignore_structural_model = args_default()$.PLS_ignore_structural_model,
  .PLS_modes                   = args_default()$.PLS_modes,
  .PLS_weight_scheme_inner     = args_default()$.PLS_weight_scheme_inner,
  .starting_values             = args_default()$.starting_values,
  .tolerance                   = args_default()$.tolerance
  )
```

Arguments

`.data`	A `data.frame` or a `matrix` of standardized or unstandardized data (indicators/items/manifest variables). Possible column types or classes of the data provided are: `"logical"`, `"numeric"` (`"double"` or `"integer"`), `"factor"` (`"ordered"` and/or `"unordered"`), `"character"` (converted to factor), or a mix of several types.
`.S`	The (K x K) empirical indicator correlation matrix.
`.csem_model`	A (possibly incomplete) cSEMModel-list.
`.conv_criterion`	Character string. The criterion to use for the convergence check. One of: "*diff_absolute*", "*diff_squared*", or "*diff_relative*". Defaults to "*diff_absolute*".
`.iter_max`	Integer. The maximum number of iterations allowed. If `iter_max = 1` and `.approach_weights = "PLS-PM"` one-step weights are returned. If the algorithm exceeds the specified number, weights of iteration step `.iter_max - 1` will be returned with a warning. Defaults to `100`.
`.PLS_ignore_structural_model`	Logical. Should the structural model be ignored when calculating the inner weights of the PLS-PM algorithm? Defaults to `FALSE`. Ignored if `.approach_weights` is not PLS-PM.
`.PLS_modes`	Either a named list specifying the mode that should be used for each construct in the form `"construct_name" = mode`, a single character string giving the mode

that should be used for all constructs, or NULL. Possible choices for mode are: "*modeA*", "*modeB*", "*modeBNNLS*", "*unit*", "*PCA*", a single integer or a vector of fixed weights of the same length as there are indicators for the construct given by "construct_name". If only a single number is provided this is identical to using unit weights, as weights are rescaled such that the related composite has unit variance. Defaults to NULL. If NULL the appropriate mode according to the type of construct used is chosen. Ignored if .approach_weight is not PLS-PM.

.PLS_weight_scheme_inner
: Character string. The inner weighting scheme used by PLS-PM. One of: "*centroid*", "*factorial*", or "*path*". Defaults to "*path*". Ignored if .approach_weight is not PLS-PM.

.starting_values
: A named list of vectors where the list names are the construct names whose indicator weights the user wishes to set. The vectors must be named vectors of "indicator_name" = value pairs, where value is the (scaled or unscaled) starting weight. Defaults to NULL.

.tolerance
: Double. The tolerance criterion for convergence. Defaults to 1e-05.

Value

A named list. J stands for the number of constructs and K for the number of indicators.

$W A (J x K) matrix of estimated weights.

$E A (J x J) matrix of inner weights.

$Modes A named vector of modes used for the outer estimation.

$Conv_status The convergence status. TRUE if the algorithm has converged and FALSE otherwise. If one-step weights are used via .iter_max = 1 or a non-iterative procedure was used, the convergence status is set to NULL.

$Iterations The number of iterations required.

References

Wold H (1975). "Path models with latent variables: The NIPALS approach." In Blalock HM, Aganbegian A, Borodkin FM, Boudon R, Capecchi V (eds.), *Quantitative Sociology*, International Perspectives on Mathematical and Statistical Modeling, 307–357. Academic Press, New York.

---

calculateWeightsUnit *Calculate composite weights using unit weights*

Description

Calculate unit weights for all blocks, i.e., each indicator of a block is equally weighted.

Usage

```
calculateWeightsUnit(
 .S                = args_default()$.S,
 .csem_model       = args_default()$.csem_model,
 .starting_values  = args_default()$.starting_values
 )
```

Arguments

`.S`	The (K x K) empirical indicator correlation matrix.
`.csem_model`	A (possibly incomplete) cSEMModel-list.
`.starting_values`	A named list of vectors where the list names are the construct names whose indicator weights the user wishes to set. The vectors must be named vectors of `"indicator_name" = value` pairs, where `value` is the (scaled or unscaled) starting weight. Defaults to `NULL`.

Value

A named list. J stands for the number of constructs and K for the number of indicators.

`$W` A (J x K) matrix of estimated weights.

`$E NULL`

`$Modes` The mode used. Always "unit".

`$Conv_status NULL` as there are no iterations

`$Iterations` 0 as there are no iterations

---

csem *Composite-based SEM*

Description

[Stable]

Usage

```
csem(
.data                 = NULL,
.model                = NULL,
.approach_2ndorder    = c("2stage", "mixed"),
.approach_cor_robust  = c("none", "mcd", "spearman"),
.approach_nl          = c("sequential", "replace"),
.approach_paths       = c("OLS", "2SLS"),
.approach_weights     = c("PLS-PM", "SUMCORR", "MAXVAR", "SSQCORR",
                          "MINVAR", "GENVAR","GSCA", "PCA",
```

```
                            "unit", "bartlett", "regression"),
  .conv_criterion        = c("diff_absolute", "diff_squared", "diff_relative"),
  .disattenuate          = TRUE,
  .dominant_indicators   = NULL,
  .estimate_structural   = TRUE,
  .id                    = NULL,
  .instruments           = NULL,
  .iter_max              = 100,
  .normality             = FALSE,
  .PLS_approach_cf       = c("dist_squared_euclid", "dist_euclid_weighted",
                            "fisher_transformed", "mean_arithmetic",
                            "mean_geometric", "mean_harmonic",
                            "geo_of_harmonic"),
  .PLS_ignore_structural_model = FALSE,
  .PLS_modes                   = NULL,
  .PLS_weight_scheme_inner     = c("path", "centroid", "factorial"),
  .reliabilities         = NULL,
  .starting_values       = NULL,
  .resample_method       = c("none", "bootstrap", "jackknife"),
  .resample_method2      = c("none", "bootstrap", "jackknife"),
  .R                     = 499,
  .R2                    = 199,
  .handle_inadmissibles  = c("drop", "ignore", "replace"),
  .user_funs             = NULL,
  .eval_plan             = c("sequential", "multicore", "multisession"),
  .seed                  = NULL,
  .sign_change_option    = c("none", "individual", "individual_reestimate",
                            "construct_reestimate"),
  .tolerance             = 1e-05
)
```

Arguments

`.data` A `data.frame` or a `matrix` of standardized or unstandardized data (indicators/items/manifest variables). Additionally, a `list` of data sets (data frames or matrices) is accepted in which case estimation is repeated for each data set. Possible column types or classes of the data provided are: `"logical"`, `"numeric"` (`"double"` or `"integer"`), `"factor"` (`"ordered"` and/or `"unordered"`), `"character"` (will be converted to factor), or a mix of several types.

`.model` A model in lavaan model syntax or a cSEMModel list.

`.approach_2ndorder`
Character string. Approach used for models containing second-order constructs. One of: "*2stage*", or "*mixed*". Defaults to "*2stage*".

`.approach_cor_robust`
Character string. Approach used to obtain a robust indicator correlation matrix. One of: "*none*" in which case the standard Bravais-Pearson correlation is used, "*spearman*" for the Spearman rank correlation, or "*mcd*" via `MASS::cov.rob()` for a robust correlation matrix. Defaults to "*none*". Note that many postestima-

tion procedures (such as `testOMF()` or `fit()` implicitly assume a continuous indicator correlation matrix (e.g. Bravais-Pearson correlation matrix). Only use if you know what you are doing.

`.approach_nl` Character string. Approach used to estimate nonlinear structural relationships. One of: "*sequential*" or "*replace*". Defaults to "*sequential*".

`.approach_paths`
Character string. Approach used to estimate the structural coefficients. One of: "*OLS*" or "*2SLS*". If "*2SLS*", instruments need to be supplied to `.instruments`. Defaults to "*OLS*".

`.approach_weights`
Character string. Approach used to obtain composite weights. One of: "*PLS-PM*", "*SUMCORR*", "*MAXVAR*", "*SSQCORR*", "*MINVAR*", "*GENVAR*", "*GSCA*", "*PCA*", "*unit*", "*bartlett*", or "*regression*". Defaults to "*PLS-PM*".

`.conv_criterion`
Character string. The criterion to use for the convergence check. One of: "*diff_absolute*", "*diff_squared*", or "*diff_relative*". Defaults to "*diff_absolute*".

`.disattenuate` Logical. Should composite/proxy correlations be disattenuated to yield consistent loadings and path estimates if at least one of the construct is modeled as a common factor? Defaults to `TRUE`.

`.dominant_indicators`
A character vector of `"construct_name" = "indicator_name"` pairs, where `"indicator_name"` is a character string giving the name of the dominant indicator and `"construct_name"` a character string of the corresponding construct name. Dominant indicators may be specified for a subset of the constructs. Default to `NULL`.

`.estimate_structural`
Logical. Should the structural coefficients be estimated? Defaults to `TRUE`.

`.id` Character string or integer. A character string giving the name or an integer of the position of the column of `.data` whose levels are used to split `.data` into groups. Defaults to `NULL`.

`.instruments` A named list of vectors of instruments. The names of the list elements are the names of the dependent (LHS) constructs of the structural equation whose explanatory variables are endogenous. The vectors contain the names of the instruments corresponding to each equation. Note that exogenous variables of a given equation **must** be supplied as instruments for themselves. Defaults to `NULL`.

`.iter_max` Integer. The maximum number of iterations allowed. If `iter_max = 1` and `.approach_weights = "PLS-PM"` one-step weights are returned. If the algorithm exceeds the specified number, weights of iteration step `.iter_max - 1` will be returned with a warning. Defaults to `100`.

`.normality` Logical. Should joint normality of $[\eta_{1:p}; \zeta; \epsilon]$ be assumed in the nonlinear model? See (Dijkstra and Schermelleh-Engel 2014) for details. Defaults to `FALSE`. Ignored if the model is not nonlinear.

`.PLS_approach_cf`
Character string. Approach used to obtain the correction factors for PLSc. One of: "*dist_squared_euclid*", "*dist_euclid_weighted*", "*fisher_transformed*",

"*mean_arithmetic*", "*mean_geometric*", "*mean_harmonic*", "*geo_of_harmonic*". Defaults to "*dist_squared_euclid*". Ignored if `.disattenuate = FALSE` or if `.approach_weights` is not PLS-PM.

`.PLS_ignore_structural_model`
: Logical. Should the structural model be ignored when calculating the inner weights of the PLS-PM algorithm? Defaults to `FALSE`. Ignored if `.approach_weights` is not PLS-PM.

`.PLS_modes`
: Either a named list specifying the mode that should be used for each construct in the form `"construct_name" = mode`, a single character string giving the mode that should be used for all constructs, or `NULL`. Possible choices for `mode` are: "*modeA*", "*modeB*", "*modeBNNLS*", "*unit*", "*PCA*", a single integer or a vector of fixed weights of the same length as there are indicators for the construct given by `"construct_name"`. If only a single number is provided this is identical to using unit weights, as weights are rescaled such that the related composite has unit variance. Defaults to `NULL`. If `NULL` the appropriate mode according to the type of construct used is chosen. Ignored if `.approach_weight` is not PLS-PM.

`.PLS_weight_scheme_inner`
: Character string. The inner weighting scheme used by PLS-PM. One of: "*centroid*", "*factorial*", or "*path*". Defaults to "*path*". Ignored if `.approach_weight` is not PLS-PM.

`.reliabilities`
: A character vector of `"name" = value` pairs, where `value` is a number between 0 and 1 and `"name"` a character string of the corresponding construct name, or `NULL`. Reliabilities may be given for a subset of the constructs. Defaults to `NULL` in which case reliabilities are estimated by `csem()`. Currently, only supported for `.approach_weights = "PLS-PM"`.

`.starting_values`
: A named list of vectors where the list names are the construct names whose indicator weights the user wishes to set. The vectors must be named vectors of `"indicator_name" = value` pairs, where `value` is the (scaled or unscaled) starting weight. Defaults to `NULL`.

`.resample_method`
: Character string. The resampling method to use. One of: "*none*", "*bootstrap*" or "*jackknife*". Defaults to "*none*".

`.resample_method2`
: Character string. The resampling method to use when resampling from a resample. One of: "*none*", "*bootstrap*" or "*jackknife*". For "*bootstrap*" the number of draws is provided via `.R2`. Currently, resampling from each resample is only required for the studentized confidence interval ("*CI_t_interval*") computed by the `infer()` function. Defaults to "*none*".

`.R`
: Integer. The number of bootstrap replications. Defaults to `499`.

`.R2`
: Integer. The number of bootstrap replications to use when resampling from a resample. Defaults to `199`.

`.handle_inadmissibles`
: Character string. How should inadmissible results be treated? One of "*drop*", "*ignore*", or "*replace*". If "*drop*", all replications/resamples yielding an inadmissible result will be dropped (i.e. the number of results returned will potentially

be less than `.R`). For "*ignore*" all results are returned even if all or some of the replications yielded inadmissible results (i.e. number of results returned is equal to `.R`). For "*replace*" resampling continues until there are exactly `.R` admissible solutions. Depending on the frequency of inadmissible solutions this may significantly increase computing time. Defaults to "*drop*".

`.user_funs` A function or a (named) list of functions to apply to every resample. The functions must take `.object` as its first argument (e.g., `myFun <- function(.object, ...) {body-of-the-`
Function output should preferably be a (named) vector but matrices are also accepted. However, the output will be vectorized (columnwise) in this case. See the examples section for details.

`.eval_plan` Character string. The evaluation plan to use. One of "*sequential*", "*multicore*", or "*multisession*". In the two latter cases all available cores will be used. Defaults to "*sequential*".

`.seed` Integer or `NULL`. The random seed to use. Defaults to `NULL` in which case an arbitrary seed is chosen. Note that the scope of the seed is limited to the body of the function it is used in. Hence, the global seed will not be altered!

`.sign_change_option`
Character string. Which sign change option should be used to handle flipping signs when resampling? One of "*none*","*individual*", "*individual_reestimate*", "*construct_reestimate*". Defaults to "*none*".

`.tolerance` Double. The tolerance criterion for convergence. Defaults to `1e-05`.

Details

Estimate linear, nonlinear, hierarchical and multigroup structural equation models using a composite-based approach. In **cSEM** any method or approach that involves linear compounds (scores/proxies/composites) of observables (indicators/items/manifest variables) is defined as composite-based. See the Get started section of the cSEM website for a general introduction to composite-based SEM and **cSEM**.

`csem()` estimates linear, nonlinear, hierarchical or multigroup structural equation models using a composite-based approach.

Data and model:: The `.data` and `.model` arguments are required. `.data` must be given a `matrix` or a `data.frame` with column names matching the indicator names used in the model description. Alternatively, a `list` of data sets (matrices or data frames) may be provided in which case estimation is repeated for each data set. Possible column types/classes of the data provided are: `"logical"`, `"numeric"` (`"double"` or `"integer"`), `"factor"` (`"ordered"` and/or `"unordered"`), `"character"`, or a mix of several types. Character columns will be treated as (unordered) factors.

Depending on the type/class of the indicator data provided cSEM computes the indicator correlation matrix in different ways. See `calculateIndicatorCor()` for details.

In the current version `.data` must not contain missing values. Future versions are likely to handle missing values as well.

To provide a model use the lavaan model syntax. Note, however, that **cSEM** currently only supports the "standard" lavaan model syntax (Types 1, 2, 3, and 7 as described on the help page). Therefore, specifying e.g., a threshold or scaling factors is ignored. Alternatively, a standardized (possibly incomplete) cSEMModel-list may be supplied. See `parseModel()` for details.

Weights and path coefficients:: By default weights are estimated using the partial least squares path modeling algorithm (`"PLS-PM"`). A range of alternative weighting algorithms may be supplied to `.approach_weights`. Currently, the following approaches are implemented

1. (Default) Partial least squares path modeling (`"PLS-PM"`). The algorithm can be customized. See `calculateWeightsPLS()` for details.
2. Generalized structured component analysis (`"GSCA"`) and generalized structured component analysis with uniqueness terms (GSCAm). The algorithms can be customized. See `calculateWeightsGSCA()` and `calculateWeightsGSCAm()` for details. Note that GSCAm is called indirectly when the model contains constructs modeled as common factors only and `.disattenuate = TRUE`. See below.
3. Generalized canonical correlation analysis (*GCCA*), including `"SUMCORR"`, `"MAXVAR"`, `"SSQCORR"`, `"MINVAR"`, `"GENVAR"`.
4. Principal component analysis (`"PCA"`)
5. Factor score regression using sum scores (`"unit"`), regression (`"regression"`) or Bartlett scores (`"bartlett"`)

It is possible to supply starting values for the weighting algorithm via `.starting_values`. The argument accepts a named list of vectors where the list names are the construct names whose indicator weights the user wishes to set. The vectors must be named vectors of `"indicator_name" = value` pairs, where `value` is the starting weight. See the examples section below for details.

Composite-indicator and composite-composite correlations are properly disattenuated by default to yield consistent loadings, construct correlations, and path coefficients if any of the concepts are modeled as a common factor.

For *PLS-PM* disattenuation is done using *PLSc* (Dijkstra and Henseler 2015). For *GSCA* disattenuation is done implicitly by using *GSCAm* (Hwang et al. 2017). Weights obtained by *GCCA*, *unit*, *regression*, *bartlett* or *PCA* are disattenuated using Croon's approach (Croon 2002). Disattenuation my be suppressed by setting `.disattenuate = FALSE`. Note, however, that quantities in this case are inconsistent estimates for their construct level counterparts if any of the constructs in the structural model are modeled as a common factor!

By default path coefficients are estimated using ordinary least squares (`.approach_path = "OLS"`). For linear models, two-stage least squares (`"2SLS"`) is available, however, *only if instruments are internal*, i.e., part of the structural model. Future versions will add support for external instruments if possible. Instruments must be supplied to `.instruments` as a named list where the names of the list elements are the names of the dependent constructs of the structural equations whose explanatory variables are believed to be endogenous. The list consists of vectors of names of instruments corresponding to each equation. Note that exogenous variables of a given equation **must** be supplied as instruments for themselves.

If reliabilities are known they can be supplied as `"name" = value` pairs to `.reliabilities`, where `value` is a numeric value between 0 and 1. Currently, only supported for "PLS-PM".

Nonlinear models:: If the model contains nonlinear terms `csem()` estimates a polynomial structural equation model using a non-iterative method of moments approach described in Dijkstra and Schermelleh-Engel (2014). Nonlinear terms include interactions and exponential terms. The latter is described in model syntax as an "interaction with itself", e.g., `xi^3 = xi.xi.xi`. Currently only exponential terms up to a power of three (e.g., three-way interactions or cubic terms) are allowed:

1. - Single, e.g., `eta1`
2. - Quadratic, e.g., `eta1.eta1`

3. - Cubic, e.g., `eta1.eta1.eta1`
4. - Two-way interaction, e.g., `eta1.eta2`
5. - Three-way interaction, e.g., `eta1.eta2.eta3`
6. - Quadratic and two-way interaction, e.g., `eta1.eta1.eta3`

The current version of the package allows two kinds of estimation: estimation of the reduced form equation (`.approach_nl = "replace"`) and sequential estimation (`.approach_nl = "sequential"`, the default). The latter does not allow for multivariate normality of all exogenous variables, i.e., the latent variables and the error terms.

Distributional assumptions are kept to a minimum (an i.i.d. sample from a population with finite moments for the relevant order); for higher order models, that go beyond interaction, we work in this version with the assumption that as far as the relevant moments are concerned certain combinations of measurement errors behave as if they were Gaussian. For details see: Dijkstra and Schermelleh-Engel (2014).

Models containing second-order constructs: Second-order constructs are specified using the operators `=~` and `<~`. These operators are usually used with indicators on their right-hand side. For second-order constructs the right-hand side variables are constructs instead. If c1, and c2 are constructs forming or measuring a higher-order construct, a model would look like this:

```
my_model <- "
# Structural model
SAT  ~ QUAL
VAL  ~ SAT

# Measurement/composite model
QUAL =~ qual1 + qual2
SAT  =~ sat1 + sat2

c1 =~ x11 + x12
c2 =~ x21 + x22

# Second-order construct (in this case a second-order composite build by common
# factors)
VAL <~ c1 + c2
"
```

Currently, two approaches are explicitly implemented:

- (Default) `"2stage"`. The (disjoint) two-stage approach as proposed by Agarwal and Karahanna (2000). Note that by default a correction for attenuation is applied if common factors are involved in modeling second-order constructs. For instance, the three-stage approach proposed by Van Riel et al. (2017) is applied in case of a second-order construct specified as a composite of common factors. On the other hand, if no common factors are involved the two-stage approach is applied as proposed by Schuberth et al. (2020).
- `"mixed"`. The mixed repeated indicators/two-stage approach as proposed by Ringle et al. (2012).

The repeated indicators approach as proposed by Joereskog and Wold (1982) and the extension proposed by Becker et al. (2012) are not directly implemented as they simply require a respecification of the model. In the above example the repeated indicators approach would require to

change the model and to append the repeated indicators to the data supplied to `.data`. Note that the indicators need to be renamed in this case as `csem()` does not allow for one indicator to be attached to multiple constructs.

```
my_model <- "
# Structural model
SAT  ~ QUAL
VAL  ~ SAT

VAL ~ c1 + c2

# Measurement/composite model
QUAL =~ qual1 + qual2
SAT  =~ sat1 + sat2
VAL  =~ x11_temp + x12_temp + x21_temp + x22_temp

c1 =~ x11 + x12
c2 =~ x21 + x22
"
```

According to the extended approach indirect effects of `QUAL` on `VAL` via `c1` and `c2` would have to be specified as well.

Multigroup analysis: To perform a multigroup analysis provide either a list of data sets or one data set containing a group-identifier-column whose column name must be provided to `.id`. Values of this column are taken as levels of a factor and are interpreted as group identifiers. `csem()` will split the data by levels of that column and run the estimation for each level separately. Note, the more levels the group-identifier-column has, the more estimation runs are required. This can considerably slow down estimation, especially if resampling is requested. For the latter it will generally be faster to use `.eval_plan = "multisession"` or `.eval_plan = "multicore"`.

Inference:: Inference is done via resampling. See `resamplecSEMResults()` and `infer()` for details.

Value

An object of class `cSEMResults` with methods for all postestimation generics. Technically, a call to `csem()` results in an object with at least two class attributes. The first class attribute is always `cSEMResults`. The second is one of `cSEMResults_default`, `cSEMResults_multi`, or `cSEMResults_2ndorder` and depends on the estimated model and/or the type of data provided to the `.model` and `.data` arguments. The third class attribute `cSEMResults_resampled` is only added if resampling was conducted. For a details see the cSEMResults helpfile .

Postestimation

`assess()` Assess results using common quality criteria, e.g., reliability, fit measures, HTMT, R2 etc.

`infer()` Calculate common inferential quantities, e.g., standard errors, confidence intervals.

`plot()` Creates a plot of the model. For the help file see `plot.cSEMResults_default()`.

`predict()` Predict endogenous indicator scores and compute common prediction metrics.

`summarize()` Summarize the results. Mainly called for its side-effect the print method.

`verify()` Verify/Check admissibility of the estimates.

Tests are performed using the test-family of functions. Currently the following tests are implemented:

`testCVPAT()` Cross-validated predictive ability test proposed by Liengaard et al. (2021)

`testOMF()` Bootstrap-based test for overall model fit based on Beran and Srivastava (1985).

`testMICOM()` Permutation-based test for measurement invariance of composites proposed by Henseler et al. (2016).

`testMGD()` Several (mainly) permutation-based tests for multi-group comparisons.

`testHausman()` Regression-based Hausman test to test for endogeneity.

Other miscellaneous postestimation functions belong do the do-family of functions. Currently three do functions are implemented:

`doIPMA()` Performs an importance-performance matrix analysis (IPMA).

`doNonlinearEffectsAnalysis()` Perform a nonlinear effects analysis as described in e.g., Spiller et al. (2013)

`doRedundancyAnalysis()` Perform a redundancy analysis (RA) as proposed by Hair et al. (2016) with reference to Chin (1998)

References

Agarwal R, Karahanna E (2000). "Time Flies When You're Having Fun: Cognitive Absorption and Beliefs about Information Technology Usage." *MIS Quarterly*, **24**(4), 665.

Becker J, Klein K, Wetzels M (2012). "Hierarchical Latent Variable Models in PLS-SEM: Guidelines for Using Reflective-Formative Type Models." *Long Range Planning*, **45**(5-6), 359–394. doi:10.1016/j.lrp.2012.10.001.

Beran R, Srivastava MS (1985). "Bootstrap Tests and Confidence Regions for Functions of a Covariance Matrix." *The Annals of Statistics*, **13**(1), 95–115. doi:10.1214/aos/1176346579.

Chin WW (1998). "Modern Methods for Business Research." In Marcoulides GA (ed.), chapter The Partial Least Squares Approach to Structural Equation Modeling, 295–358. Mahwah, NJ: Lawrence Erlbaum.

Croon MA (2002). "Using predicted latent scores in general latent structure models." In Marcoulides GA, Moustaki I (eds.), *Latent Variable and Latent Structure Models*, chapter 10, 195–224. Lawrence Erlbaum. ISBN 080584046X, Pagination: 288.

Dijkstra TK, Henseler J (2015). "Consistent and Asymptotically Normal PLS Estimators for Linear Structural Equations." *Computational Statistics & Data Analysis*, **81**, 10–23.

Dijkstra TK, Schermelleh-Engel K (2014). "Consistent Partial Least Squares For Nonlinear Structural Equation Models." *Psychometrika*, **79**(4), 585–604.

Hair JF, Hult GTM, Ringle C, Sarstedt M (2016). *A Primer on Partial Least Squares Structural Equation Modeling (PLS-SEM)*. Sage publications.

Henseler J, Ringle CM, Sarstedt M (2016). "Testing Measurement Invariance of Composites Using Partial Least Squares." *International Marketing Review*, **33**(3), 405–431. doi:10.1108/imr092014-0304.

Hwang H, Takane Y, Jung K (2017). "Generalized structured component analysis with uniqueness terms for accommodating measurement error." *Frontiers in Psychology*, **8**(2137), 1–12.

Joereskog KG, Wold HO (1982). *Systems under Indirect Observation: Causality, Structure, Prediction - Part II*, volume 139. North Holland.

Liengaard BD, Sharma PN, Hult GTM, Jensen MB, Sarstedt M, Hair JF, Ringle CM (2021). "Prediction: Coveted, Yet Forsaken? Introducing a Cross-Validated Predictive Ability Test in Partial Least Squares Path Modeling." *Decision Sciences*, **52**(2), 362–392.

Ringle CM, Sarstedt M, Straub D (2012). "A Critical Look at the Use of PLS-SEM in MIS Quarterly." *MIS Quarterly*, **36**(1), iii–xiv.

Schuberth F, Rademaker ME, Henseler J (2020). "Estimating and assessing second-order constructs using PLS-PM: the case of composites of composites." *Industrial Management & Data Systems*, **120**(12), 2211-2241. doi:10.1108/imds1220190642.

Spiller SA, Fitzsimons GJ, Lynch JG, Mcclelland GH (2013). "Spotlights, Floodlights, and the Magic Number Zero: Simple Effects Tests in Moderated Regression." *Journal of Marketing Research*, **50**(2), 277–288. doi:10.1509/jmr.12.0420.

Van Riel ACR, Henseler J, Kemeny I, Sasovova Z (2017). "Estimating hierarchical constructs using Partial Least Squares: The case of second order composites of factors." *Industrial Management & Data Systems*, **117**(3), 459–477. doi:10.1108/IMDS0720160286.

See Also

`args_default()`, cSEMArguments, cSEMResults, `foreman()`, `resamplecSEMResults()`, `assess()`, `infer()`, `plot.cSEMResults_default()`, `predict()`, `summarize()`, `verify()`, `testCVPAT()`, `testOMF()`, `testMGD()`, `testMICOM()`, `testHausman()`

Examples

```
# ==============================================================================
# Basic usage
# ==============================================================================
### Linear model ---------------------------------------------------------------
# Most basic usage requires a dataset and a model. We use the
#  `threecommonfactors` dataset.

## Take a look at the dataset
#?threecommonfactors
```

```
## Specify the (correct) model
model <- "
# Structural model
eta2 ~ eta1
eta3 ~ eta1 + eta2

# (Reflective) measurement model
eta1 =~ y11 + y12 + y13
eta2 =~ y21 + y22 + y23
eta3 =~ y31 + y32 + y33
"

## Estimate
res <- csem(threecommonfactors, model)

## Postestimation
verify(res)
summarize(res)
assess(res)

# Notes:
#   1. By default no inferential quantities (e.g. Std. errors, p-values, or
#      confidence intervals) are calculated. Use resampling to obtain
#      inferential quantities. See "Resampling" in the "Extended usage"
#      section below.
#   2. `summarize()` prints the full output by default. For a more condensed
#       output use:
print(summarize(res), .full_output = FALSE)

## Dealing with endogeneity ------------------------------------------------

# See: ?testHausman()

### Models containing second constructs-------------------------------------
## Take a look at the dataset
#?dgp_2ndorder_cf_of_c

model <- "
# Path model / Regressions
c4   ~ eta1
eta2 ~ eta1 + c4

# Reflective measurement model
c1   <~ y11 + y12
c2   <~ y21 + y22 + y23 + y24
c3   <~ y31 + y32 + y33 + y34 + y35 + y36 + y37 + y38
eta1 =~ y41 + y42 + y43
eta2 =~ y51 + y52 + y53

# Composite model (second order)
c4   =~ c1 + c2 + c3
"
```

```
res_2stage <- csem(dgp_2ndorder_cf_of_c, model, .approach_2ndorder = "2stage")
res_mixed  <- csem(dgp_2ndorder_cf_of_c, model, .approach_2ndorder = "mixed")

# The standard repeated indicators approach is done by 1.) respecifying the model
# and 2.) adding the repeated indicators to the data set

# 1.) Respecify the model
model_RI <- "
# Path model / Regressions
c4   ~ eta1
eta2 ~ eta1 + c4
c4   ~ c1 + c2 + c3

# Reflective measurement model
c1   <~ y11 + y12
c2   <~ y21 + y22 + y23 + y24
c3   <~ y31 + y32 + y33 + y34 + y35 + y36 + y37 + y38
eta1 =~ y41 + y42 + y43
eta2 =~ y51 + y52 + y53

# c4 is a common factor measured by composites
c4 =~ y11_temp + y12_temp + y21_temp + y22_temp + y23_temp + y24_temp +
      y31_temp + y32_temp + y33_temp + y34_temp + y35_temp + y36_temp +
      y37_temp + y38_temp
"

# 2.) Update data set
data_RI <- dgp_2ndorder_cf_of_c
coln <- c(colnames(data_RI), paste0(colnames(data_RI), "_temp"))
data_RI <- data_RI[, c(1:ncol(data_RI), 1:ncol(data_RI))]
colnames(data_RI) <- coln

# Estimate
res_RI <- csem(data_RI, model_RI)
summarize(res_RI)

### Multigroup analysis ------------------------------------------------

# See: ?testMGD()

# ============================================================================
# Extended usage
# ============================================================================
# `csem()` provides defaults for all arguments except `.data` and `.model`.
#   Below some common options/tasks that users are likely to be interested in.
#   We use the threecommonfactors data set again:

model <- "
# Structural model
eta2 ~ eta1
eta3 ~ eta1 + eta2

# (Reflective) measurement model
```

```
eta1 =~ y11 + y12 + y13
eta2 =~ y21 + y22 + y23
eta3 =~ y31 + y32 + y33
"

### PLS vs PLSc and disattenuation
# In the model all concepts are modeled as common factors. If
#   .approach_weights = "PLS-PM", csem() uses PLSc to disattenuate composite-indicator
#   and composite-composite correlations.
res_plsc <- csem(threecommonfactors, model, .approach_weights = "PLS-PM")
res$Information$Model$construct_type # all common factors

# To obtain "original" (inconsistent) PLS estimates use `.disattenuate = FALSE`
res_pls <- csem(threecommonfactors, model,
                .approach_weights = "PLS-PM",
                .disattenuate = FALSE
                )

s_plsc <- summarize(res_plsc)
s_pls  <- summarize(res_pls)

# Compare
data.frame(
  "Path"      = s_plsc$Estimates$Path_estimates$Name,
  "Pop_value" = c(0.6, 0.4, 0.35), # see ?threecommonfactors
  "PLSc"      = s_plsc$Estimates$Path_estimates$Estimate,
  "PLS"       = s_pls$Estimates$Path_estimates$Estimate
  )

### Resampling ------------------------------------------------------------
## Not run:
## Basic resampling
res_boot <- csem(threecommonfactors, model, .resample_method = "bootstrap")
res_jack <- csem(threecommonfactors, model, .resample_method = "jackknife")

# See ?resamplecSEMResults for more examples

### Choosing a different weightning scheme --------------------------------

res_gscam  <- csem(threecommonfactors, model, .approach_weights = "GSCA")
res_gsca   <- csem(threecommonfactors, model,
                   .approach_weights = "GSCA",
                   .disattenuate = FALSE
)

s_gscam <- summarize(res_gscam)
s_gsca  <- summarize(res_gsca)

# Compare
data.frame(
  "Path"      = s_gscam$Estimates$Path_estimates$Name,
  "Pop_value" = c(0.6, 0.4, 0.35), # see ?threecommonfactors
  "GSCAm"      = s_gscam$Estimates$Path_estimates$Estimate,
```

```
  "GSCA"       = s_gsca$Estimates$Path_estimates$Estimate
)
## End(Not run)
### Fine-tuning a weighting scheme ----------------------------------------
## Setting starting values

sv <- list("eta1" = c("y12" = 10, "y13" = 4, "y11" = 1))
res <- csem(threecommonfactors, model, .starting_values = sv)

## Choosing a different inner weighting scheme
#?args_csem_dotdotdot

res <- csem(threecommonfactors, model, .PLS_weight_scheme_inner = "factorial",
            .PLS_ignore_structural_model = TRUE)


## Choosing different modes for PLS
# By default, concepts modeled as common factors uses PLS Mode A weights.
modes <- list("eta1" = "unit", "eta2" = "modeB", "eta3" = "unit")
res   <- csem(threecommonfactors, model, .PLS_modes = modes)
summarize(res)
```

---

dgp_2ndorder_cf_of_c *Data: Second order common factor of composites*

Description

A dataset containing 500 standardized observations on 19 indicator generated from a population model with 6 concepts, three of which (c1-c3) are composites forming a second order common factor (c4). The remaining two (eta1, eta2) are concepts modeled as common factors .

Usage

```
dgp_2ndorder_cf_of_c
```

Format

A matrix with 500 rows and 19 variables:

y11-y12 Indicators attached to c1. Population weights are: 0.8; 0.4. Population loadings are: 0.925; 0.65

y21-y24 Indicators attached to c2. Population weights are: 0.5; 0.3; 0.2; 0.4. Population loadings are: 0.804; 0.68; 0.554; 0.708

y31-y38 Indicators attached to c3. Population weights are: 0.3; 0.3; 0.1; 0.1; 0.2; 0.3; 0.4; 0.2. Population loadings are: 0.496; 0.61; 0.535; 0.391; 0.391; 0.6; 0.5285; 0.53

y41-y43 Indicators attached to eta1. Population loadings are: 0.8; 0.7; 0.7

y51-y53 Indicators attached to eta1. Population loadings are: 0.8; 0.8; 0.7

The model is:

$$`c4` = gamma1 * `eta1` + zeta1$$

$$`eta2` = gamma2 * `eta1` + beta * `c4` + zeta2$$

with population values `gamma1` = 0.6, `gamma2` = 0.4 and `beta` = 0.35. The second order common factor is

$$`c4` = lambdac1 * `c1` + lambdac2 * `c2` + lambdac3 * `c3` + epsilon$$

`distance_measures` *Calculate difference between S and Sigma_hat*

Description

Calculate the difference between the empirical (S) and the model-implied indicator variance-covariance matrix (Sigma_hat) using different distance measures.

Usage

```
calculateDG(
  .object = NULL,
  .matrix1 = NULL,
  .matrix2 = NULL,
  .saturated = FALSE,
  ...
)

calculateDL(
  .object = NULL,
  .matrix1 = NULL,
  .matrix2 = NULL,
  .saturated = FALSE,
  ...
)

calculateDML(
  .object = NULL,
  .matrix1 = NULL,
  .matrix2 = NULL,
  .saturated = FALSE,
  ...
)
```

Arguments

`.object`	An R object of class cSEMResults resulting from a call to `csem()`.
`.matrix1`	A `matrix` to compare.
`.matrix2`	A `matrix` to compare.
`.saturated`	Logical. Should a saturated structural model be used? Defaults to `FALSE`.
`...`	Ignored.

Details

The distances may also be computed for any two matrices A and B by supplying A and B directly via the `.matrix1` and `.matrix2` arguments. If A and B are supplied `.object` is ignored.

Value

A single numeric value giving the distance between two matrices.

Functions

- `calculateDG()`: The geodesic distance (dG).
- `calculateDL()`: The squared Euclidean distance
- `calculateDML()`: The distance measure (fit function) used by ML

`doIPMA` *Do an importance-performance matrix analysis*

Description

[Maturing]

Usage

```
doIPMA(.object)
```

Arguments

`.object`	A `cSEMResults` object.‘

Details

Performs an importance-performance matrix analysis (IPMA).

To calculate the performance and importance, the weights of the indicators are unstandardized using the standard deviation of the original indicators but normed to have a length of 1. Normed construct scores are calculated based on the original indicators and the unstandardized weights.

The importance is calculated as the mean of the original indicators or the unstandardized construct scores, respectively. The performance is calculated as the unstandardized total effect if `.level == "construct"` and as the normed weight times the unstandardized total effect if `.level == "indicator"`. The literature recommends to use an estimation as input for 'doIPMA() that is based on normed indicators, e.g., by scaling all indicators to 0 to 100, see e.g., Henseler (2021); Ringle and Sarstedt (2016).

Note, indicators are not normed internally, as theoretical maximum/minimum can differ from the empirical maximum/minimum which would lead to an incorrect normalization.

Value

A list of class `cSEMIPA` with a corresponding method for `plot()`. See: `plot.cSEMIPMA()`.

See Also

`csem()`, cSEMResults, `plot.cSEMIPMA()`

---

doNonlinearEffectsAnalysis

Do a nonlinear effects analysis

Description

[Maturing]

Usage

```
doNonlinearEffectsAnalysis(
 .object            = NULL,
 .dependent         = NULL,
 .independent       = NULL,
 .moderator         = NULL,
 .n_steps           = 100,
 .values_moderator  = c(-2, -1, 0, 1, 2),
 .value_independent = 0,
 .alpha             = 0.05
 )
```

Arguments

`.object`	An R object of class cSEMResults resulting from a call to `csem()`.
`.dependent`	Character string. The name of the dependent variable.
`.independent`	Character string. The name of the independent variable.
`.moderator`	Character string. The name of the moderator variable.
`.n_steps`	Integer. A value giving the number of steps (the spotlights, i.e., values of .moderator in surface analysis or floodlight analysis) between the minimum and maximum value of the moderator. Defaults to `100`.
`.values_moderator`	A numeric vector. The values of the moderator in a the simple effects analysis. Typically these are difference from the mean (=0) measured in standard deviations. Defaults to `c(-2, -1, 0, 1, 2)`.
`.value_independent`	Integer. Only required for floodlight analysis; The value of the independent variable in case that it appears as a higher-order term.
`.alpha`	An integer or a numeric vector of significance levels. Defaults to `0.05`.

Details

Calculate the expected value of the dependent variable conditional on the values of an independent variables and a moderator variable. All other variables in the model are assumed to be zero, i.e., they are fixed at their mean levels. Moreover, it produces the input for the floodlight analysis.

Value

A list of class `cSEMNonlinearEffects` with a corresponding method for `plot()`. See: `plot.cSEMNonlinearEffects()`.

See Also

`csem()`, cSEMResults, `plot.cSEMNonlinearEffects()`

Examples

```
## Not run:
model_Int <- "
# Measurement models
INV =~ INV1 + INV2 + INV3 +INV4
SAT =~ SAT1 + SAT2 + SAT3
INT =~ INT1 + INT2

# Structrual model containing an interaction term.
INT ~ INV + SAT + INV.SAT
"

# Estimate model
out <- csem(.data = Switching, .model = model_Int,
            # ADANCO settings
            .PLS_weight_scheme_inner = 'factorial',
```

```
            .tolerance = 1e-06,
            .resample_method = 'bootstrap'
)

# Do nonlinear effects analysis
neffects <- doNonlinearEffectsAnalysis(out,
                                       .dependent = 'INT',
                                       .moderator = 'INV',
                                       .independent = 'SAT')

# Get an overview
neffects

# Simple effects plot
plot(neffects, .plot_type = 'simpleeffects')

# Surface plot using plotly
plot(neffects, .plot_type = 'surface', .plot_package = 'plotly')

# Surface plot using persp
plot(neffects, .plot_type = 'surface', .plot_package = 'persp')

# Floodlight analysis
plot(neffects, .plot_type = 'floodlight')

## End(Not run)
```

`doRedundancyAnalysis` *Do a redundancy analysis*

Description

[Stable]

Usage

```
doRedundancyAnalysis(.object = NULL)
```

Arguments

`.object`	An R object of class cSEMResults resulting from a call to `csem()`.

Details

Perform a redundancy analysis (RA) as proposed by Hair et al. (2016) with reference to Chin (1998).

RA is confined to PLS-PM, specifically PLS-PM with at least one construct whose weights are obtained by mode B. In cSEM this is the case if the construct is modeled as a composite or if

argument .PLS_modes was explicitly set to mode B for at least one construct. Hence RA is only conducted if .approach_weights = "PLS-PM" and if at least one construct's mode is mode B.

The principal idea of RA is to take two different measures of the same construct and regress the scores obtained for each measure on each other. If they are similar they are likely to measure the same "thing" which is then taken as evidence that both measurement models actually measure what they are supposed to measure (validity).

There are several issues with the terminology and the reasoning behind this logic. RA is therefore only implemented since reviewers are likely to demand its computation, however, its actual application for validity assessment is discouraged.

Currently, the function is not applicable to models containing second-order constructs.

Value

A named numeric vector of correlations. If the weighting approach used to obtain .object is not "PLS-PM" or non of the PLS outer modes was mode B, the function silently returns NA.

References

Chin WW (1998). "Modern Methods for Business Research." In Marcoulides GA (ed.), chapter The Partial Least Squares Approach to Structural Equation Modeling, 295–358. Mahwah, NJ: Lawrence Erlbaum.

Hair JF, Hult GTM, Ringle C, Sarstedt M (2016). *A Primer on Partial Least Squares Structural Equation Modeling (PLS-SEM)*. Sage publications.

See Also

cSEMResults

---

exportToExcel *Export to Excel (.xlsx)*

Description

[Experimental]

Usage

```
exportToExcel(
  .postestimation_object = NULL,
  .filename              = "results.xlsx",
  .path                  = NULL
  )
```

Arguments

.postestimation_object	An object resulting from a call to one of cSEM's postestimation functions (e.g. summarize()).
.filename	Character string. The file name.
.path	Character string. Path of the directory to save the file to. Defaults to NULL.

Details

Export results from postestimation functions assess(), predict(), summarize() and testOMF() to an .xlsx (Excel) file. The function uses the openxlsx package which does not depend on Java!

The function is deliberately kept simple: it takes all the relevant elements in .postestimation_object and writes them (worksheet by worksheet) into an .xlsx file named .filename in the directory given by .path (the current working directory by default).

If .postestimation_object has class attribute _2ndorder two .xlsx files named ".filename_first_stage.xlsx" and ".filename_second_stage.xlsx" are created. If .postestimation_object is a list of appropriate objects, one file for each list elements is created.

Note: rerunning exportToExcel() without changing .filename and .path overwrites the file!

See Also

assess(), summarize(), predict(), testOMF()

---

fit *Model-implied indicator or construct variance-covariance matrix*

Description

Calculate the model-implied indicator or construct variance-covariance (VCV) matrix. Currently only the model-implied VCV for recursive linear models is implemented (including models containing second order constructs).

Usage

```
fit(
  .object    = NULL,
  .saturated = args_default()$.saturated,
  .type_vcv  = args_default()$.type_vcv
  )
```

Arguments

.object	An R object of class cSEMResults resulting from a call to csem().
.saturated	Logical. Should a saturated structural model be used? Defaults to FALSE.
.type_vcv	Character string. Which model-implied correlation matrix should be calculated? One of "*indicator*" or "*construct*". Defaults to "*indicator*".

Details

Notation is taken from Bollen (1989). If `.saturated = TRUE` the model-implied variance-covariance matrix is calculated for a saturated structural model (i.e., the VCV of the constructs is replaced by their correlation matrix). Hence: V(eta) = WSW' (possibly disattenuated).

Value

Either a (K x K) matrix or a (J x J) matrix depending on the `type_vcv`.

References

Bollen KA (1989). *Structural Equations with Latent Variables*. Wiley-Interscience. ISBN 978-0471011712.

See Also

`csem()`, `foreman()`, cSEMResults

`fit_measures` *Model fit measures*

Description

Calculate fit measures.

Usage

```
calculateChiSquare(.object, .saturated = FALSE)

calculateChiSquareDf(.object)

calculateCFI(.object)

calculateGFI(.object, .type_gfi = c("ML", "GLS", "ULS"), ...)

calculateCN(.object, .alpha = 0.05, ...)

calculateIFI(.object)

calculateNFI(.object)

calculateNNFI(.object)

calculateRMSEA(.object)

calculateRMSTheta(.object)
```

```
calculateSRMR(
  .object = NULL,
  .matrix1 = NULL,
  .matrix2 = NULL,
  .saturated = FALSE,
  ...
)
```

Arguments

.object	An R object of class cSEMResults resulting from a call to csem().
.saturated	Logical. Should a saturated structural model be used? Defaults to FALSE.
.type_gfi	Character string. Which fitting function should the GFI be based on? One of *"ML"* for the maximum likelihood fitting function, *"GLS"* for the generalized least squares fitting function or *"ULS"* for the unweighted least squares fitting function (same as the squared Euclidean distance). Defaults to *"ML"*.
...	Ignored.
.alpha	An integer or a numeric vector of significance levels. Defaults to 0.05.
.matrix1	A matrix to compare.
.matrix2	A matrix to compare.

Details

See the Fit indices section of the cSEM website for details on the implementation.

Value

A single numeric value.

Functions

- calculateChiSquare(): The chi square statistic.
- calculateChiSquareDf(): The Chi square statistic divided by its degrees of freedom.
- calculateCFI(): The comparative fit index (CFI).
- calculateGFI(): The goodness of fit index (GFI).
- calculateCN(): The Hoelter index alias Hoelter's (critical) N (CN).
- calculateIFI(): The incremental fit index (IFI).
- calculateNFI(): The normed fit index (NFI).
- calculateNNFI(): The non-normed fit index (NNFI). Also called the Tucker-Lewis index (TLI).
- calculateRMSEA(): The root mean square error of approximation (RMSEA).
- calculateRMSTheta(): The root mean squared residual covariance matrix of the outer model residuals (RMS theta).
- calculateSRMR(): The standardized root mean square residual (SRMR).

getConstructScores *Get construct scores*

Description

[Stable]

Usage

```
getConstructScores(
 .object        = NULL,
 .standardized  = TRUE
 )
```

Arguments

.object	An R object of class cSEMResults resulting from a call to csem().
.standardized	Logical. Should standardized scores be returned? Defaults to TRUE.

Details

Get the standardized or unstandardized construct scores.

Value

A list of three with elements Construct_scores, W_used, Indicators_used.

See Also

csem(), cSEMResults

infer *Inference*

Description

[Stable]

Usage

```
infer(
 .object           = NULL,
 .quantity         = c("all", "mean", "sd", "bias", "CI_standard_z",
                       "CI_standard_t", "CI_percentile", "CI_basic",
                       "CI_bc", "CI_bca", "CI_t_interval"),
 .alpha            = 0.05,
 .bias_corrected   = TRUE
)
```

Arguments

`.object`	An R object of class cSEMResults resulting from a call to `csem()`.
`.quantity`	Character string. Which statistic should be returned? One of "*all*", "*mean*", "*sd*", "*bias*", "*CI_standard_z*", "*CI_standard_t*", "*CI_percentile*", "*CI_basic*", "*CI_bc*", "*CI_bca*", "*CI_t_interval*" Defaults to "*all*" in which case all quantities that do not require additional resampling are returned, i.e., all quantities but "*CI_bca*", "*CI_t_interval*".
`.alpha`	An integer or a numeric vector of significance levels. Defaults to `0.05`.
`.bias_corrected`	Logical. Should the standard and the tStat confidence interval be bias-corrected using the bootstrapped bias estimate? If `TRUE` the confidence interval for some estimated parameter `theta` is centered at `2*theta - theta*_hat`, where `theta*_hat` is the average over all `.R` bootstrap estimates of `theta`. Defaults to `TRUE`

Details

Calculate common inferential quantities. For users interested in the estimated standard errors, t-values, p-values and/or confidences intervals of the path, weight or loading estimates, calling `summarize()` directly will usually be more convenient as it has a much more user-friendly print method. `infer()` is useful for comparing different confidence interval estimates.

`infer()` is a convenience wrapper around a number of internal functions that compute a particular inferential quantity, i.e., a value or set of values to be used in statistical inference.

cSEM relies on resampling (bootstrap and jackknife) as the basis for the computation of e.g., standard errors or confidence intervals. Consequently, `infer()` requires resamples to work. Technically, the cSEMResults object used in the call to `infer()` must therefore also have class attribute `cSEMResults_resampled`. If the object provided by the user does not contain resamples yet, `infer()` will obtain bootstrap resamples first. Naturally, computation will take longer in this case.

`infer()` does as much as possible in the background. Hence, every time `infer()` is called on a cSEMResults object the quantities chosen by the user are automatically computed for every estimated parameter contained in the object. By default all possible quantities are computed (`.quantity = all`). The following table list the available inferential quantities alongside a brief description. Implementation and terminology of the confidence intervals is based on Hesterberg (2015) and Davison and Hinkley (1997).

`"mean"`, `"sd"` The mean or the standard deviation over all `M` resample estimates of a generic statistic or parameter.

`"bias"` The difference between the resample mean and the original estimate of a generic statistic or parameter.

`"CI_standard_z"` **and** `"CI_standard_t"` The standard confidence interval for a generic statistic or parameter with standard errors estimated by the resample standard deviation. While `"CI_standard_z"` assumes a standard normally distributed statistic, `"CI_standard_t"` assumes a t-statistic with N - 1 degrees of freedom.

`"CI_percentile"` The percentile confidence interval. The lower and upper bounds of the confidence interval are estimated as the alpha and 1-alpha quantiles of the distribution of the resample estimates.

`"CI_basic"` The basic confidence interval also called the reverse bootstrap percentile confidence interval. See Hesterberg (2015) for details.

`"CI_bc"` The bias corrected (Bc) confidence interval. See Davison and Hinkley (1997) for details.

`"CI_bca"` The bias-corrected and accelerated (Bca) confidence interval. Requires additional jackknife resampling to compute the influence values. See Davison and Hinkley (1997) for details.

`"CI_t_interval"` The "studentized" t-confidence interval. If based on bootstrap resamples the interval is also called the bootstrap t-interval confidence interval. See Hesterberg (2015) on page 381. Requires resamples of resamples. See `resamplecSEMResults()`.

By default, all but the studendized t-interval confidence interval and the bias-corrected and accelerated confidence interval are calculated. The reason for excluding these quantities by default are that both require an additional resampling step. The former requires jackknife estimates to compute influence values and the latter requires double bootstrap. Both can potentially be time consuming. Hence, computation is triggered only if explicitly chosen.

Value

A list of class `cSEMInfer`.

References

Davison AC, Hinkley DV (1997). *Bootstrap Methods and their Application*. Cambridge University Press. doi:10.1017/cbo9780511802843.

Hesterberg TC (2015). "What Teachers Should Know About the Bootstrap: Resampling in the Undergraduate Statistics Curriculum." *The American Statistician*, **69**(4), 371–386. doi:10.1080/00031305.2015.1089789.

See Also

`csem()`, `resamplecSEMResults()`, `summarize()` cSEMResults

Examples

```
model <- "
# Structural model
QUAL ~ EXPE
EXPE ~ IMAG
SAT  ~ IMAG + EXPE + QUAL + VAL
LOY  ~ IMAG + SAT
VAL  ~ EXPE + QUAL

# Measurement model
EXPE =~ expe1 + expe2 + expe3 + expe4 + expe5
IMAG =~ imag1 + imag2 + imag3 + imag4 + imag5
LOY  =~ loy1  + loy2  + loy3  + loy4
QUAL =~ qual1 + qual2 + qual3 + qual4 + qual5
SAT  =~ sat1  + sat2  + sat3  + sat4
VAL  =~ val1  + val2  + val3  + val4
"
```

```
## Estimate the model with bootstrap resampling
a <- csem(satisfaction, model, .resample_method = "bootstrap", .R = 20,
          .handle_inadmissibles = "replace")

## Compute inferential quantities
inf <- infer(a)

inf$Path_estimates$CI_basic
inf$Indirect_effect$sd

### Compute the bias-corrected and accelerated and/or the studentized t-inverval.
## For the studentied t-interval confidence interval a double bootstrap is required.
## This is pretty time consuming.
## Not run:
  inf <- infer(a, .quantity = c("all", "CI_bca")) # requires jackknife estimates

## Estimate the model with double bootstrap resampling:
# Notes:
#   1. The .resample_method2 arguments triggers a bootstrap of each bootstrap sample
#   2. The double bootstrap is is very time consuming, consider setting
#      `.eval_plan = "multisession`.
a1 <- csem(satisfaction, model, .resample_method = "bootstrap", .R = 499,
          .resample_method2 = "bootstrap", .R2 = 199, .handle_inadmissibles = "replace")
infer(a1, .quantity = "CI_t_interval")
## End(Not run)
```

`ITFlex` *Data: ITFlex*

Description

A data frame containing 16 variables with 100 observations.

Usage

```
ITFlex
```

Format

A data frame containing the following variables:

`ITCOMP1` Software applications can be easily transported and used across multiple platforms.

`ITCOMP2` Our firm provides multiple interfaces or entry points (e.g., web access) for external end users.

`ITCOMP3` Our firm establishes corporate rules and standards for hardware and operating systems to ensure platform compatibility.

ITCOMP4 Data captured in one part of our organization are immediately available to everyone in the firm.

ITCONN1 Our organization has electronic links and connections throughout the entire firm.

ITCONN2 Our firm is linked to business partners through electronic channels (e.g., websites, e-mail, wireless devices, electronic data interchange).

ITCONN3 All remote, branch, and mobile offices are connected to the central office.

ITCONN4 There are very few identifiable communications bottlenecks within our firm.

MOD1 Our firm possesses a great speed in developing new business applications or modifying existing applications.

MOD2 Our corporate database is able to communicate in several different protocols.

MOD3 Reusable software modules are widely used in new systems development.

MOD4 IT personnel use object-oriented and prepackaged modular tools to create software applications.

ITPSF1 Our IT personnel have the ability to work effectively in cross-functional teams.

ITPSF2 Our IT personnel are able to interpret business problems and develop appropriate technical solutions.

ITPSF3 Our IT personnel are self-directed and proactive.

ITPSF4 Our IT personnel are knowledgeable about the key success factors in our firm.

Details

The dataset was studied by Benitez et al. (2018) and is used in Henseler (2021) for demonstration purposes, see the corresponding tutorial. All questionnaire items are measured on a 5-point scale.

Source

The data was collected through a survey by Benitez et al. (2018).

References

Benitez J, Ray G, Henseler J (2018). "Impact of Information Technology Infrastructure Flexibility on Mergers and Acquisitions." *MIS Quarterly*, **42**(1), 25–43.

Henseler J (2021). *Composite-Based Structural Equation Modeling: Analyzing Latent and Emergent Variables*. Guilford Press, New York.

Examples

```
#================================================================================
# Example is taken from Henseler (2020)
#================================================================================
model_IT_Fex="
# Composite models
ITComp  <~ ITCOMP1 + ITCOMP2 + ITCOMP3 + ITCOMP4
Modul   <~ MOD1 + MOD2 + MOD3 + MOD4
ITConn  <~ ITCONN1 + ITCONN2 + ITCONN3 + ITCONN4
```

```
ITPers  <~ ITPSF1 + ITPSF2 + ITPSF3 + ITPSF4

# Saturated structural model
ITPers ~ ITComp + Modul + ITConn
Modul  ~ ITComp + ITConn
ITConn ~ ITComp
"

out <- csem(.data = ITFlex, .model = model_IT_Fex,
            .PLS_weight_scheme_inner = 'factorial',
            .tolerance = 1e-06,
            .PLS_ignore_structural_model = TRUE)
```

`LancelotMiltgenetal2016`

Data: LancelotMiltgenetal2016

Description

A data frame containing 10 variables with 1090 observations.

Usage

```
LancelotMiltgenetal2016
```

Format

An object of class `data.frame` with 1090 rows and 11 columns.

Details

The data was analysed by Lancelot-Miltgen et al. (2016) to study young consumers' adoption intentions of a location tracker technology in the light of privacy concerns. It is also used in Henseler (2021) for demonstration purposes, see the corresponding tutorial.

Source

This data has been collected through a cooperation with the European Commission Joint Research Center Institute for Prospective Technological Studies, contract "Young People and Emerging Digital Services: An Exploratory Survey on Motivations, Perceptions, and Acceptance of Risk" (EC JRC Contract IPTS No: 150876-2007 F1ED-FR).

References

Henseler J (2021). *Composite-Based Structural Equation Modeling: Analyzing Latent and Emergent Variables*. Guilford Press, New York.

Lancelot-Miltgen C, Henseler J, Gelhard C, Popovic A (2016). "Introducing new products that affect consumer privacy: A mediation model." *Journal of Business Research*, **69**(10), 4659–4666. doi:10.1016/j.jbusres.2016.04.015.

Examples

```
#===============================================================================
# Example is taken from Henseler (2020)
#===============================================================================
model_Med <- "
# Reflective measurement model
Trust =~ trust1 + trust2
PrCon =~ privcon1 + privcon2 + privcon3 + privcon4
Risk  =~ risk1 + risk2 + risk3
Int   =~ intent1 + intent2

# Structural model
Int   ~ Trust + PrCon + Risk
Risk  ~ Trust + PrCon
Trust ~ PrCon
"

out <- csem(.data = LancelotMiltgenetal2016, .model = model_Med,
            .PLS_weight_scheme_inner = 'factorial',
            .tolerance = 1e-06
)
```

`parseModel` *Parse lavaan model*

Description

Turns a model written in lavaan model syntax into a cSEMModel list.

Usage

```
parseModel(
  .model        = NULL,
  .instruments  = NULL,
  .check_errors = TRUE
  )
```

Arguments

`.model`	A model in lavaan model syntax or a cSEMModel list.
`.instruments`	A named list of vectors of instruments. The names of the list elements are the names of the dependent (LHS) constructs of the structural equation whose explanatory variables are endogenous. The vectors contain the names of the instruments corresponding to each equation. Note that exogenous variables of a given equation **must** be supplied as instruments for themselves. Defaults to `NULL`.
`.check_errors`	Logical. Should the model to parse be checked for correctness in a sense that all necessary components to estimate the model are given? Defaults to `TRUE`.

Details

Instruments must be supplied separately as a named list of vectors of instruments. The names of the list elements are the names of the dependent constructs of the structural equation whose explanatory variables are endogenous. The vectors contain the names of the instruments corresponding to each equation. Note that exogenous variables of a given equation **must** be supplied as instruments for themselves.

By default `parseModel()` attempts to check if the model provided is correct in a sense that all necessary components required to estimate the model are specified (e.g., a construct of the structural model has at least 1 item). To prevent checking for errors use `.check_errors = FALSE`.

Value

An object of class cSEMModel is a standardized list containing the following components. J stands for the number of constructs and K for the number of indicators.

`$structural` A matrix mimicking the structural relationship between constructs. If constructs are only linearly related, `structural` is of dimension (J x J) with row- and column names equal to the construct names. If the structural model contains nonlinear relationships `structural` is (J x (J + J*)) where J* is the number of nonlinear terms. Rows are ordered such that exogenous constructs are always first, followed by constructs that only depend on exogenous constructs and/or previously ordered constructs.

`$measurement` A (J x K) matrix mimicking the measurement/composite relationship between constructs and their related indicators. Rows are in the same order as the matrix `$structural` with row names equal to the construct names. The order of the columns is such that `$measurement` forms a block diagonal matrix.

`$error_cor` A (K x K) matrix mimicking the measurement error correlation relationship. The row and column order is identical to the column order of `$measurement`.

`$cor_specified` A matrix indicating the correlation relationships between any variables of the model as specified by the user. Mainly for internal purposes. Note that `$cor_specified` may also contain inadmissible correlations such as a correlation between measurement errors indicators and constructs.

`$construct_type` A named vector containing the names of each construct and their respective type ("Common factor" or "Composite").

`$construct_order` A named vector containing the names of each construct and their respective order ("First order" or "Second order").

`$model_type` The type of model ("Linear" or "Nonlinear").

`$instruments` Only if instruments are supplied: a list of structural equations relating endogenous RHS variables to instruments.

`$indicators` The names of the indicators (i.e., observed variables and/or first-order constructs)

`$cons_exo` The names of the exogenous constructs of the structural model (i.e., variables that do not appear on the LHS of any structural equation)

`$cons_endo` The names of the endogenous constructs of the structural model (i.e., variables that appear on the LHS of at least one structural equation)

`$vars_2nd` The names of the constructs modeled as second orders.

`$vars_attached_to_2nd` The names of the constructs forming or building a second order construct.

`$vars_not_attached_to_2nd` The names of the constructs not forming or building a second order construct.

It is possible to supply an incomplete list to `parseModel()`, resulting in an incomplete cSEMModel list which can be passed to all functions that require `.csem_model` as a mandatory argument. Currently, only the structural and the measurement matrix are required. However, specifying an incomplete cSEMModel list may lead to unexpected behavior and errors. Use with care.

Examples

```
# ===========================================================================
# Providing a model in lavaan syntax
# ===========================================================================
model <- "
# Structural model
y1 ~ y2 + y3

# Measurement model
y1 =~ x1 + x2 + x3
y2 =~ x4 + x5
y3 =~ x6 + x7

# Error correlation
x1 ~~ x2
"

m <- parseModel(model)
m

# ===========================================================================
# Providing a complete model in cSEM format (class cSEMModel)
# ===========================================================================
# If the model is already a cSEMModel object, the model is returned as is:

identical(m, parseModel(m)) # TRUE

# ===========================================================================
# Providing a list
```

```
# ============================================================================
# It is possible to provide a list that contains at least the
# elements "structural" and "measurement". This is generally discouraged
# as this may cause unexpected errors.

m_incomplete <- m[c("structural", "measurement", "construct_type")]
parseModel(m_incomplete)

# Providing a list containing list names that are not part of a `cSEMModel`
# causes an error:

## Not run:
m_incomplete[c("name_a", "name_b")] <- c("hello world", "hello universe")
parseModel(m_incomplete)

## End(Not run)

# Failing to provide "structural" or "measurement" also causes an error:

## Not run:
m_incomplete <- m[c("structural", "construct_type")]
parseModel(m_incomplete)

## End(Not run)
```

`plot.cSEMIPMA` `cSEMIPMA` *method for* `plot()`

Description

Plot the importance-performance matrix.

Usage

```
## S3 method for class 'cSEMIPMA'
plot(
  x = NULL,
  .dependent = NULL,
  .attributes = NULL,
  .level = c("construct", "indicator"),
  ...
)
```

Arguments

`x`	An R object of class `cSEMIPMA`.
`.dependent`	Character string. Name of the target construct for which the importance-performance matrix should be created.

`.attributes`	Character string. A vector containing indicator/construct names that should be plotted in the importance-performance matrix. It must be at least of length 2.
`.level`	Character string. Indicates the level for which the importance-performance matrix should be plotted. One of `"construct"` or `"indicator"`. Defaults to `"construct"`.
`...`	Currently ignored.

See Also

`doIPMA()`

---

plot.cSEMNonlinearEffects

`cSEMNonlinearEffects` *method for* `plot()`

Description

This plot method can be used to create plots to analyze non-linear models in more depth. In doing so the following plot types can be selected:

`.plot_type = "simpleeffects"`: The plot of a simple effects analysis displays the predicted value of the dependent variable for different values of the independent variable and the moderator. As levels for the moderator the levels provided to the `doNonlinearEffectsAnalysis()` function are used. Since the constructs are standardized the values of the moderator equals the deviation from its mean measured in standard deviations.

`.plot_type = "surface"`: The plot of a surface analysis displays the predicted values of an independent variable (z). The values are predicted based on the values of the moderator and the independent variable including all their higher-order terms. For the values of the moderator and the independent variable steps between their minimum and maximum values are used.

`.plot_type = "floodlight"`: The plot of a floodlight analysis displays the direct effect of an continuous independent variable (z) on a dependent variable (y) conditional on the values of a continuous moderator variable (x), including the confidence interval and the Johnson-Neyman points. It is noted that in the floodlight plot only moderation is taken into account and higher order terms are ignored. For more details, see Spiller et al. (2013).

Plot the predicted values of an independent variable (z) The values are predicted based on a certain moderator and a certain independent variable including all their higher-order terms.

Usage

```
## S3 method for class 'cSEMNonlinearEffects'
plot(x, .plot_type = "simpleeffects", .plot_package = "plotly", ...)
```

Arguments

x	An R object of class cSEMNonlinearEffects.
.plot_type	A character string indicating the type of plot that should be produced. Options are "*simpleeffects*", "*surface*", and "*floodlight*". Defaults to "*simpleeffects*".
.plot_package	A character vector indicating the plot package used. Options are "*plotly*", and "*persp*". Defaults to "*plotly*".
...	Additional parameters that can be passed to graphics::persp, e.g., to rotate the plot.

See Also

doNonlinearEffectsAnalysis()

---

plot.cSEMResults_2ndorder

cSEMResults *method for* plot() *for second-order models.*

Description

[Experimental]

Usage

```
## S3 method for class 'cSEMResults_2ndorder'
plot(
  x,
  .title = args_default()$.title,
  .plot_significances = args_default()$.plot_significances,
  .plot_correlations = args_default()$.plot_correlations,
  .plot_structural_model_only = args_default()$.plot_structural_model_only,
  .plot_labels = args_default()$.plot_labels,
  .graph_attrs = args_default()$.graph_attrs,
  ...
)
```

Arguments

x	An R object of class cSEMResults_2ndorder object.
.title	Character string. Title of an object. Defaults to *""*.
.plot_significances	Logical. Should p-values in the form of stars be plotted? Defaults to TRUE.
.plot_correlations	Character string. Specify which correlations should be plotted, i.e., between the exogenous constructs (exo), between the exogenous constructs and the indicators (all), or not at all (none). Defaults to exo.

`.plot_structural_model_only`
: Logical. Should only the structural model, i.e., the constructs and their relationships be plotted? Defaults to `FALSE`.

`.plot_labels`
: Logical. Whether to display edge labels and node R^2 values. Defaults to TRUE.

`.graph_attrs`
: Character string. Additional attributes that should be passed to the DiagrammeR syntax, e.g., c("rankdir=LR", "ranksep=1.0"). Defaults to *c("rankdir=LR")*.

`...`
: Currently ignored.

Details

Creates a plot of a `cSEMResults_2ndorder` object using the grViz function. For more details on customizing plot, see https://rpubs.com/nguyen_mot/1275413.

See Also

`csem()`, cSEMResults, grViz

---

plot.cSEMResults_default

`cSEMResults` *method for* `plot()`

Description

[Experimental]

Usage

```
## S3 method for class 'cSEMResults_default'
plot(
  x = NULL,
  .title = args_default()$.title,
  .plot_significances = args_default()$.plot_significances,
  .plot_correlations = args_default()$.plot_correlations,
  .plot_structural_model_only = args_default()$.plot_structural_model_only,
  .plot_labels = args_default()$.plot_labels,
  .graph_attrs = args_default()$.graph_attrs,
  ...
)
```

Arguments

`x`
: An R object of class `cSEMResults_default` object.

`.title`
: Character string. Title of an object. Defaults to "".

`.plot_significances`
: Logical. Should p-values in the form of stars be plotted? Defaults to `TRUE`.

`.plot_correlations`
: Character string. Specify which correlations should be plotted, i.e., between the exogenous constructs (`exo`), between the exogenous constructs and the indicators (`all`), or not at all (`none`). Defaults to `exo`.

`.plot_structural_model_only`
: Logical. Should only the structural model, i.e., the constructs and their relationships be plotted? Defaults to `FALSE`.

`.plot_labels`
: Logical. Whether to display edge labels and R^2 values in the nodes. Defaults to TRUE (i.e. original plot).

`.graph_attrs`
: Character string. Additional attributes that should be passed to the DiagrammeR syntax, e.g., c("rankdir=LR", "ranksep=1.0"). Defaults to *c("rankdir=LR")*.

`...`
: Currently ignored.

Details

Creates a plot of a `cSEMResults` object using the grViz function. For more details on customizing plot, see https://rpubs.com/nguyen_mot/1275413.

See Also

`savePlot()` `csem()`, cSEMResults, grViz

Examples

```
## Not run:
model_Bergami_int="
  # Common factor and composite models
  OrgPres <~ cei1 + cei2 + cei3 + cei4 + cei5 + cei6 + cei7 + cei8
  OrgIden =~ ma1 + ma2 + ma3 + ma4 + ma5 + ma6
  AffJoy =~ orgcmt1 + orgcmt2 + orgcmt3 + orgcmt7
  AffLove  =~ orgcmt5 + orgcmt6 + orgcmt8

  # Structural model
  OrgIden ~ OrgPres
  AffLove ~ OrgPres+OrgIden+OrgPres.OrgIden
  AffJoy  ~ OrgPres+OrgIden
  "

  outBergamiInt <- csem(.data = BergamiBagozzi2000,.model = model_Bergami_int,
                        .disattenuate = T,
                        .PLS_weight_scheme_inner = 'factorial',
                        .tolerance = 1e-6,
                        .resample_method = 'none')

  outPlot <- plot(outBergamiInt)
  outPlot
  savePlot(outPlot,.file='plot.pdf')
  savePlot(outPlot,.file='plot.png')
  savePlot(outPlot,.file='plot.svg')
  savePlot(outPlot,.file='plot.dot')
```

```
## End(Not run)
```

`plot.cSEMResults_multi` `cSEMResults` *method for* `plot()` *for multiple groups.*

Description

[Experimental]

Usage

```
## S3 method for class 'cSEMResults_multi'
plot(
  x = NULL,
  .title = args_default()$.title,
  .plot_significances = args_default()$.plot_significances,
  .plot_correlations = args_default()$.plot_correlations,
  .plot_structural_model_only = args_default()$.plot_structural_model_only,
  .plot_labels = args_default()$.plot_labels,
  .graph_attrs = args_default()$.graph_attrs,
  ...
)
```

Arguments

`x`	An R object of class `cSEMResults_multi` object.
`.title`	Character string. Title of an object. Defaults to *""*.
`.plot_significances`	Logical. Should p-values in the form of stars be plotted? Defaults to `TRUE`.
`.plot_correlations`	Character string. Specify which correlations should be plotted, i.e., between the exogenous constructs (`exo`), between the exogenous constructs and the indicators (`all`), or not at all (`none`). Defaults to `exo`.
`.plot_structural_model_only`	Logical. Should only the structural model, i.e., the constructs and their relationships be plotted? Defaults to `FALSE`.
`.plot_labels`	Logical. Whether to display edge labels and node R^2 values. Defaults to TRUE.
`.graph_attrs`	Character string. Additional attributes that should be passed to the DiagrammeR syntax, e.g., c("rankdir=LR", "ranksep=1.0"). Defaults to *c("rankdir=LR")*.
`...`	Currently ignored.

Details

Creates a plot of a `cSEMResults` object using the grViz function. For more details on customizing plot, see https://rpubs.com/nguyen_mot/1275413.

See Also

csem(), cSEMResults, grViz

`PoliticalDemocracy` *Data: political democracy*

Description

The Industrialization and Political Democracy dataset. This dataset is used throughout Bollen's 1989 book (see pages 12, 17, 36 in chapter 2, pages 228 and following in chapter 7, pages 321 and following in chapter 8; Bollen (1989)). The dataset contains various measures of political democracy and industrialization in developing countries.

Usage

```
PoliticalDemocracy
```

Format

A data frame of 75 observations of 11 variables.

- `y1` Expert ratings of the freedom of the press in 1960
- `y2` The freedom of political opposition in 1960
- `y3` The fairness of elections in 1960
- `y4` The effectiveness of the elected legislature in 1960
- `y5` Expert ratings of the freedom of the press in 1965
- `y6` The freedom of political opposition in 1965
- `y7` The fairness of elections in 1965
- `y8` The effectiveness of the elected legislature in 1965
- `x1` The gross national product (GNP) per capita in 1960
- `x2` The inanimate energy consumption per capita in 1960
- `x3` The percentage of the labor force in industry in 1960

Source

The lavaan package (version 0.6-3).

References

Bollen KA (1989). *Structural Equations with Latent Variables*. Wiley-Interscience. ISBN 978-0471011712.

Examples

```
#==============================================================================
# Example is taken from the lavaan website
#==============================================================================
# Note: example is modified. Across-block correlations are removed
model <- "
# Measurement model
  ind60 =~ x1 + x2 + x3
  dem60 =~ y1 + y2 + y3 + y4
  dem65 =~ y5 + y6 + y7 + y8

# Regressions / Path model
  dem60 ~ ind60
  dem65 ~ ind60 + dem60

# residual correlations
  y2 ~~ y4
  y6 ~~ y8
"

aa <- csem(PoliticalDemocracy, model)
```

---

predict *Predict indicator scores*

Description

[Maturing]

Usage

```
predict(
 .object                   = NULL,
 .benchmark           = c("lm", "unit", "PLS-PM", "GSCA", "PCA", "MAXVAR", "NA"),
 .approach_predict         = c("earliest", "direct"),
 .cv_folds                 = 10,
 .handle_inadmissibles     = c("stop", "ignore", "set_NA"),
 .r                        = 1,
 .test_data                = NULL,
 .approach_score_target    = c("mean", "median", "mode"),
 .sim_points               = 100,
 .disattenuate             = TRUE,
 .treat_as_continuous      = TRUE,
 .approach_score_benchmark = c("mean", "median", "mode", "round"),
 .seed                     = NULL
 )
```

Arguments

`.object`	An R object of class cSEMResults resulting from a call to `csem()`.
`.benchmark`	Character string. The procedure to obtain benchmark predictions. One of "*lm*", "*unit*", "*PLS-PM*", "*GSCA*", "*PCA*", "*MAXVAR*", or "*NA*". Default to "*lm*".
`.approach_predict`	Character string. Which approach should be used to perform predictions? One of "*earliest*" and "*direct*". If "*earliest*" predictions for indicators associated to endogenous constructs are performed using only indicators associated to exogenous constructs. If "*direct*", predictions for indicators associated to endogenous constructs are based on indicators associated to their direct antecedents. Defaults to "*earliest*".
`.cv_folds`	Integer. The number of cross-validation folds to use. Setting `.cv_folds` to `N` (the number of observations) produces leave-one-out cross-validation samples. Defaults to `10`.
`.handle_inadmissibles`	Character string. How should inadmissible results be treated? One of "*stop*", "*ignore*", or "*set_NA*". If "*stop*", `predict()` will stop immediately if estimation yields an inadmissible result. For "*ignore*" all results are returned even if all or some of the estimates yielded inadmissible results. For "*set_NA*" predictions based on inadmissible parameter estimates are set to `NA`. Defaults to "*stop*"
`.r`	Integer. The number of repetitions to use. Defaults to `1`.
`.test_data`	A matrix of test data with the same column names as the training data.
`.approach_score_target`	Character string. How should the aggregation of the estimates of the truncated normal distribution for the predictions using OrdPLS/OrdPLSc be done? One of "*mean*", "*median*" or "*mode*". If "*mean*", the mean of the estimated endogenous indicators is calculated. If "*median*", the mean of the estimated endogenous indicators is calculated. If "*mode*", the maximum empirical density on the intervals defined by the thresholds is used. Defaults to "*mean*".
`.sim_points`	Integer. How many samples from the truncated normal distribution should be simulated to estimate the exogenous construct scores? Defaults to "*100*".
`.disattenuate`	Logical. Should the benchmark predictions be based on disattenuated parameter estimates? Defaults to `TRUE`.
`.treat_as_continuous`	Logical. Should the indicators for the benchmark predictions be treated as continuous? If `TRUE` all indicators are treated as continuous and PLS-PM/PLSc is applied. If `FALSE` OrdPLS/OrdPLSc is applied. Defaults to `TRUE`.
`.approach_score_benchmark`	Character string. How should the aggregation of the estimates of the truncated normal distribution be done for the benchmark predictions? Ignored if not OrdPLS or OrdPLSc is used to obtain benchmark predictions. One of "*mean*", "*median*", "*mode*" or "*round*". If "*round*", the benchmark predictions are obtained using the traditional prediction algorithm for PLS-PM which are rounded for categorical indicators. If "*mean*", the mean of the estimated endogenous indicators is calculated. If "*median*", the mean of the estimated endogenous indicators is calculated. If "*mode*", the maximum empirical density on the intervals defined

by the thresholds is used. If .treat_as_continuous = TRUE or if all indicators are on a continuous scale, .approach_score_benchmark is ignored. Defaults to "*round*".

.seed Integer or NULL. The random seed to use. Defaults to NULL in which case an arbitrary seed is chosen. Note that the scope of the seed is limited to the body of the function it is used in. Hence, the global seed will not be altered!

Details

The predict function implements the procedure introduced by Shmueli et al. (2016) in the PLS context known as "PLSPredict" (Shmueli et al. 2019) including its variants PLScPredcit, OrdPLSpredict and OrdPLScpredict. It is used to predict the indicator scores of endogenous constructs and to evaluate the out-of-sample predictive power of a model. For that purpose, the predict function uses k-fold cross-validation to randomly split the data into training and test datasets, and subsequently predicts the values of the test data based on the model parameter estimates obtained from the training data. The number of cross-validation folds is 10 by default but may be changed using the .cv_folds argument. By default, the procedure is not repeated (.r = 1). You may choose to repeat cross-validation by setting a higher .r to be sure not to have a particular (unfortunate) split. See Shmueli et al. (2019) for details. Typically .r = 1 should be sufficient though.

Alternatively, users may supply a test dataset as matrix or a data frame of .test_data with the same column names as those in the data used to obtain .object (the training data). In this case, arguments .cv_folds and .r are ignored and predict uses the estimated coefficients from .object to predict the values in the columns of .test_data.

In Shmueli et al. (2016) PLS-based predictions for indicator i are compared to the predictions based on a multiple regression of indicator i on all available exogenous indicators (.benchmark = "lm") and a simple mean-based prediction summarized in the Q2_predict metric. predict() is more general in that is allows users to compare the predictions based on a so-called target model/specification to predictions based on an alternative benchmark. Available benchmarks include predictions based on a linear model, PLS-PM weights, unit weights (i.e. sum scores), GSCA weights, PCA weights, and MAXVAR weights.

Each estimation run is checked for admissibility using verify(). If the estimation yields inadmissible results, predict() stops with an error ("stop"). Users may choose to "ignore" inadmissible results or to simply set predictions to NA ("set_NA") for the particular run that failed.

Value

An object of class cSEMPredict with print and plot methods. Technically, cSEMPredict is a named list containing the following list elements:

$Actual A matrix of the actual values/indicator scores of the endogenous constructs.

$Prediction_target A list containing matrices of the predicted indicator scores of the endogenous constructs based on the target model for each repetition .r. Target refers to procedure used to estimate the parameters in .object.

$Residuals_target A list of matrices of the residual indicator scores of the endogenous constructs based on the target model in each repetition .r.

$Residuals_benchmark A list of matrices of the residual indicator scores of the endogenous constructs based on a model estimated by the procedure given to .benchmark for each repetition .r.

`$Prediction_metrics` A data frame containing the predictions metrics MAE, RMSE, Q2_predict, the misclassification error rate (MER), the MAPE, the MSE2, Theil's forecast accuracy (U1), Theil's forecast quality (U2), Bias proportion of MSE (UM), Regression proportion of MSE (UR), and disturbance proportion of MSE (UD) (Hora and Campos 2015; Watson and Teelucksingh 2002).

`$Information` A list with elements `Target`, `Benchmark`, `Number_of_observations_training`, `Number_of_observations_test`, `Number_of_folds`, `Number_of_repetitions`, and `Handle_inadmissibles`.

References

Hora J, Campos P (2015). "A review of performance criteria to validate simulation models." *Expert Systems*, **32**(5), 578–595. doi:10.1111/exsy.12111.

Shmueli G, Ray S, Estrada JMV, Chatla SB (2016). "The Elephant in the Room: Predictive Performance of PLS Models." *Journal of Business Research*, **69**(10), 4552–4564. doi:10.1016/j.jbusres.2016.03.049.

Shmueli G, Sarstedt M, Hair JF, Cheah J, Ting H, Vaithilingam S, Ringle CM (2019). "Predictive Model Assessment in PLS-SEM: Guidelines for Using PLSpredict." *European Journal of Marketing*, **53**(11), 2322–2347. doi:10.1108/ejm0220190189.

Watson PK, Teelucksingh SS (2002). *A practical introduction to econometric methods: Classical and modern*. University of West Indies Press, Mona, Jamaica.

See Also

csem, cSEMResults, `exportToExcel()`

Examples

```
### Anime example taken from https://github.com/ISS-Analytics/pls-predict/

# Load data
data(Anime) # data is similar to the Anime.csv found on
            # https://github.com/ISS-Analytics/pls-predict/ but with irrelevant
            # columns removed

# Split into training and data the same way as it is done on
# https://github.com/ISS-Analytics/pls-predict/
set.seed(123)

index     <- sample.int(dim(Anime)[1], 83, replace = FALSE)
dat_train <- Anime[-index, ]
dat_test  <- Anime[index, ]

# Specify model
model <- "
# Structural model

ApproachAvoidance ~ PerceivedVisualComplexity + Arousal
```

```
# Measurement/composite model

ApproachAvoidance         =~ AA0 + AA1 + AA2 + AA3
PerceivedVisualComplexity <~ VX0 + VX1 + VX2 + VX3 + VX4
Arousal                   <~ Aro1 + Aro2 + Aro3 + Aro4
"

# Estimate (replicating the results of the `simplePLS()` function)
res <- csem(dat_train,
            model,
            .disattenuate = FALSE, # original PLS
            .iter_max = 300,
            .tolerance = 1e-07,
            .PLS_weight_scheme_inner = "factorial"
)

# Predict using a user-supplied training data set
pp <- predict(res, .test_data = dat_test)
pp

### Compute prediction metrics  ------------------------------------------------
res2 <- csem(Anime, # whole data set
            model,
            .disattenuate = FALSE, # original PLS
            .iter_max = 300,
            .tolerance = 1e-07,
            .PLS_weight_scheme_inner = "factorial"
)

# Predict using 10-fold cross-validation
## Not run:
pp2 <- predict(res, .benchmark = "lm")
pp2
## There is a plot method available
plot(pp2)
## End(Not run)

### Example using OrdPLScPredict -----------------------------------------------
# Transform the numerical indicators into factors
## Not run:
data("BergamiBagozzi2000")
data_new <- data.frame(cei1    = as.ordered(BergamiBagozzi2000$cei1),
                       cei2    = as.ordered(BergamiBagozzi2000$cei2),
                       cei3    = as.ordered(BergamiBagozzi2000$cei3),
                       cei4    = as.ordered(BergamiBagozzi2000$cei4),
                       cei5    = as.ordered(BergamiBagozzi2000$cei5),
                       cei6    = as.ordered(BergamiBagozzi2000$cei6),
                       cei7    = as.ordered(BergamiBagozzi2000$cei7),
                       cei8    = as.ordered(BergamiBagozzi2000$cei8),
                       ma1     = as.ordered(BergamiBagozzi2000$ma1),
                       ma2     = as.ordered(BergamiBagozzi2000$ma2),
                       ma3     = as.ordered(BergamiBagozzi2000$ma3),
                       ma4     = as.ordered(BergamiBagozzi2000$ma4),
```

```
                        ma5     = as.ordered(BergamiBagozzi2000$ma5),
                        ma6     = as.ordered(BergamiBagozzi2000$ma6),
                        orgcmt1 = as.ordered(BergamiBagozzi2000$orgcmt1),
                        orgcmt2 = as.ordered(BergamiBagozzi2000$orgcmt2),
                        orgcmt3 = as.ordered(BergamiBagozzi2000$orgcmt3),
                        orgcmt5 = as.ordered(BergamiBagozzi2000$orgcmt5),
                        orgcmt6 = as.ordered(BergamiBagozzi2000$orgcmt6),
                        orgcmt7 = as.ordered(BergamiBagozzi2000$orgcmt7),
                        orgcmt8 = as.ordered(BergamiBagozzi2000$orgcmt8))

model <- "
# Measurement models
OrgPres =~ cei1 + cei2 + cei3 + cei4 + cei5 + cei6 + cei7 + cei8
OrgIden =~ ma1 + ma2 + ma3 + ma4 + ma5 + ma6
AffJoy  =~ orgcmt1 + orgcmt2 + orgcmt3 + orgcmt7
AffLove =~ orgcmt5 + orgcmt 6 + orgcmt8

# Structural model
OrgIden ~ OrgPres
AffLove ~ OrgIden
AffJoy  ~ OrgIden
"
# Estimate using cSEM; note: the fact that indicators are factors triggers OrdPLSc
res <- csem(.model = model, .data = data_new[1:250,])
summarize(res)

# Predict using OrdPLSPredict
set.seed(123)
pred <- predict(
  .object = res,
  .benchmark = "PLS-PM",
  .test_data = data_new[(251):305,],
   .treat_as_continuous = TRUE, .approach_score_target = "median"
  )

pred
round(pred$Prediction_metrics[, -1], 4)
## End(Not run)
```

`reliability` *Reliability*

Description

Compute several reliability estimates. See the Reliability section of the cSEM website for details.

Usage

```
calculateRhoC(
  .object = NULL,
```

```
  .model_implied = TRUE,
  .only_common_factors = TRUE,
  .weighted = FALSE
)

calculateRhoT(
  .object = NULL,
  .alpha = 0.05,
  .closed_form_ci = FALSE,
  .only_common_factors = TRUE,
  .output_type = c("vector", "data.frame"),
  .weighted = FALSE,
  ...
)
```

Arguments

`.object`	An R object of class cSEMResults resulting from a call to `csem()`.
`.model_implied`	Logical. Should weights be scaled using the model-implied indicator correlation matrix? Defaults to `TRUE`.
`.only_common_factors`	Logical. Should only concepts modeled as common factors be included when calculating one of the following quality criteria: AVE, the Fornell-Larcker criterion, HTMT, and all reliability estimates. Defaults to `TRUE`.
`.weighted`	Logical. Should estimation be based on a score that uses the weights of the weight approach used to obtain `.object`?. Defaults to `FALSE`.
`.alpha`	An integer or a numeric vector of significance levels. Defaults to `0.05`.
`.closed_form_ci`	Logical. Should a closed-form confidence interval be computed? Defaults to `FALSE`.
`.output_type`	Character string. The type of output. One of "vector" or "data.frame". Defaults to "vector".
`...`	Ignored.

Details

Since reliability is defined with respect to a classical true score measurement model only concepts modeled as common factors are considered by default. For concepts modeled as composites reliability may be estimated by setting `.only_common_factors = FALSE`, however, it is unclear how to interpret reliability in this case.

Reliability is traditionally based on a test score (proxy) based on unit weights. To compute congeneric and tau-equivalent reliability based on a score that uses the weights of the weight approach used to obtain `.object` use `.weighted = TRUE` instead.

For the tau-equivalent reliability (`"rho_T"` or `"cronbachs_alpha"`) a closed-form confidence interval may be computed (Trinchera et al. 2018) by setting `.closed_form_ci = TRUE` (default is `FALSE`). If `.alpha` is a vector several CIs are returned.

Value

For `calculateRhoC()` and `calculateRhoT()` (if `.output_type = "vector"`) a named numeric vector containing the reliability estimates. If `.output_type = "data.frame"` `calculateRhoT()` returns a `data.frame` with as many rows as there are constructs modeled as common factors in the model (unless `.only_common_factors = FALSE` in which case the number of rows equals the total number of constructs in the model). The first column contains the name of the construct. The second column the reliability estimate. If `.closed_form_ci = TRUE` the remaining columns contain lower and upper bounds for the (1 - `.alpha`) confidence interval(s).

Functions

- `calculateRhoC()`: Calculate the congeneric reliability
- `calculateRhoT()`: Calculate the tau-equivalent reliability

References

Trinchera L, Marie N, Marcoulides GA (2018). "A Distribution Free Interval Estimate for Coefficient Alpha." *Structural Equation Modeling: A Multidisciplinary Journal*, **25**(6), 876–887. doi:10.1080/10705511.2018.1431544.

See Also

`assess()`, cSEMResults

---

resamplecSEMResults *Resample cSEMResults*

Description

Resample a cSEMResults object using bootstrap or jackknife resampling. The function is called by `csem()` if the user sets `csem(..., .resample_method = "bootstrap")` or `csem(..., .resample_method = "jackknife")` but may also be called directly.

Usage

```
resamplecSEMResults(
 .object                = NULL,
 .resample_method       = c("bootstrap", "jackknife"),
 .resample_method2      = c("none", "bootstrap", "jackknife"),
 .R                     = 499,
 .R2                    = 199,
 .handle_inadmissibles  = c("drop", "ignore", "replace"),
 .user_funs             = NULL,
 .eval_plan             = c("sequential", "multicore", "multisession"),
 .force                 = FALSE,
 .seed                  = NULL,
 .sign_change_option    = c("none","individual","individual_reestimate",
```

```
                                "construct_reestimate"),
  ...
)
```

Arguments

`.object`	An R object of class cSEMResults resulting from a call to `csem()`.
`.resample_method`	Character string. The resampling method to use. One of: "*bootstrap*" or "*jackknife*". Defaults to "*bootstrap*".
`.resample_method2`	Character string. The resampling method to use when resampling from a resample. One of: "*none*", "*bootstrap*" or "*jackknife*". For "*bootstrap*" the number of draws is provided via `.R2`. Currently, resampling from each resample is only required for the studentized confidence interval ("*CI_t_interval*") computed by the `infer()` function. Defaults to "*none*".
`.R`	Integer. The number of bootstrap replications. Defaults to `499`.
`.R2`	Integer. The number of bootstrap replications to use when resampling from a resample. Defaults to `199`.
`.handle_inadmissibles`	Character string. How should inadmissible results be treated? One of "*drop*", "*ignore*", or "*replace*". If "*drop*", all replications/resamples yielding an inadmissible result will be dropped (i.e. the number of results returned will potentially be less than `.R`). For "*ignore*" all results are returned even if all or some of the replications yielded inadmissible results (i.e. number of results returned is equal to `.R`). For "*replace*" resampling continues until there are exactly `.R` admissible solutions. Depending on the frequency of inadmissible solutions this may significantly increase computing time. Defaults to "*drop*".
`.user_funs`	A function or a (named) list of functions to apply to every resample. The functions must take `.object` as its first argument (e.g., `myFun <- function(.object, ...) {body-of-the-` Function output should preferably be a (named) vector but matrices are also accepted. However, the output will be vectorized (columnwise) in this case. See the examples section for details.
`.eval_plan`	Character string. The evaluation plan to use. One of "*sequential*", "*multicore*", or "*multisession*". In the two latter cases all available cores will be used. Defaults to "*sequential*".
`.force`	Logical. Should .object be resampled even if it contains resamples already?. Defaults to `FALSE`.
`.seed`	Integer or `NULL`. The random seed to use. Defaults to `NULL` in which case an arbitrary seed is chosen. Note that the scope of the seed is limited to the body of the function it is used in. Hence, the global seed will not be altered!
`.sign_change_option`	Character string. Which sign change option should be used to handle flipping signs when resampling? One of "*none*","*individual*", "*individual_reestimate*", "*construct_reestimate*". Defaults to "*none*".
`...`	Further arguments passed to functions supplied to `.user_funs`.

Details

Given M resamples (for bootstrap M = .R and for jackknife M = N, where N is the number of observations) based on the data used to compute the cSEMResults object provided via .object, resamplecSEMResults() essentially calls csem() on each resample using the arguments of the original call (ignoring any arguments related to resampling) and returns estimates for each of a subset of practically useful resampled parameters/statistics computed by csem(). Currently, the following estimates are computed and returned by default based on each resample: Path estimates, Loading estimates, Weight estimates.

In practical application users may need to resample a specific statistic (e.g, the heterotrait-monotrait ratio of correlations (HTMT) or differences between path coefficients such as beta_1 - beta_2). Such statistics may be provided by a function fun(.object, ...) or a list of such functions via the .user_funs argument. The first argument of these functions must always be .object. Internally, the function will be applied on each resample to produce the desired statistic. Hence, arbitrary complicated statistics may be resampled as long as the body of the function draws on elements contained in the cSEMResults object only. Output of fun(.object, ...) should preferably be a (named) vector but matrices are also accepted. However, the output will be vectorized (columnwise) in this case. See the examples section for details.

Both resampling the original cSEMResults object (call it "first resample") and resampling based on a resampled cSEMResults object (call it "second resample") are supported. Choices for the former are "*bootstrap*" and "*jackknife*". Resampling based on a resample is turned off by default (.resample_method2 = "none") as this significantly increases computation time (there are now M * M2 resamples to compute, where M2 is .R2 or N). Resamples of a resample are required, e.g., for the studentized confidence interval computed by the infer() function. Typically, bootstrap resamples are used in this case (Davison and Hinkley 1997).

As csem() accepts a single data set, a list of data sets as well as data sets that contain a column name used to split the data into groups, the cSEMResults object may contain multiple data sets. In this case, resampling is done by data set or group. Note that depending on the number of data sets/groups, the computation may be considerably slower as resampling will be repeated for each data set/group. However, apart from speed considerations users don not need to worry about the type of input used to compute the cSEMResults object as resamplecSEMResults() is able to deal with each case.

The number of bootstrap runs for the first and second run are given by .R and .R2. The default is 499 for the first and 199 for the second run but should be increased in real applications. See e.g., Hesterberg (2015), p.380, Davison and Hinkley (1997), and Efron and Hastie (2016) for recommendations. For jackknife .R are .R2 are ignored.

Resampling may produce inadmissible results (as checked by verify()). By default these results are dropped however users may choose to "ignore" or "replace" inadmissible results in which resampling continuous until the necessary number of admissible results is reached.

The **cSEM** package supports (multi)processing via the future framework (Bengtsson 2018). Users may simply choose an evaluation plan via .eval_plan and the package takes care of all the complicated backend issues. Currently, users may choose between standard single-core/single-session evaluation ("sequential") and multiprocessing ("multisession" or "multicore"). The future package provides other options (e.g., "cluster" or "remote"), however, they probably will not be needed in the context of the **cSEM** package as simulations usually do not require high-performance clusters. Depending on the operating system, the future package will manage to distribute tasks to multiple R sessions (Windows) or multiple cores. Note that multiprocessing is not necessary

always faster when only a "small" number of replications is required as the overhead of initializing new sessions or distributing tasks to different cores will not immediately be compensated by the availability of multiple sessions/cores.

Random number generation (RNG) uses the L'Ecuyer-CRMR RGN stream as implemented in the future.apply package (Bengtsson 2018). It is independent of the evaluation plan. Hence, setting e.g., `.seed = 123` will generate the same random number and replicates for both `.eval_plan = "sequential"`, `.eval_plan = "multisession"`, and `.eval_plan = "multicore"`. See ?future_lapply for details.

Value

The core structure is the same structure as that of `.object` with the following elements added:

- `$Estimates_resamples`: A list containing the `.R` resamples and the original estimates for each of the resampled quantities (Path_estimates, Loading_estimates, Weight_estimates, user defined functions). Each list element is a list containing elements `$Resamples` and `$Original`. `$Resamples` is a `(.R x K)` matrix with each row representing one resample for each of the `K` parameters/statistics. `$Original` contains the original estimates (vectorized by column if the output of the user provided function is a matrix.
- `$Information_resamples`: A list containing additional information.

Use `str(<.object>, list.len = 3)` on the resulting object for an overview.

References

Bengtsson H (2018). *future: Unified Parallel and Distributed Processing in R for Everyone*. R package version 1.10.0, https://CRAN.R-project.org/package=future.

Bengtsson H (2018). *future.apply: Apply Function to Elements in Parallel using Futures*. R package version 1.0.1, https://CRAN.R-project.org/package=future.apply.

Davison AC, Hinkley DV (1997). *Bootstrap Methods and their Application*. Cambridge University Press. doi:10.1017/cbo9780511802843.

Efron B, Hastie T (2016). *Computer Age Statistical Inference*. Cambridge University Pr. ISBN 1107149894.

Hesterberg TC (2015). "What Teachers Should Know About the Bootstrap: Resampling in the Undergraduate Statistics Curriculum." *The American Statistician*, **69**(4), 371–386. doi:10.1080/00031305.2015.1089789.

See Also

csem, `summarize()`, `infer()`, cSEMResults

Examples

```
## Not run:
# Note: example not run as resampling is time consuming
# ==============================================================================
```

```
# Basic usage
# ============================================================================
model <- "
# Structural model
QUAL ~ EXPE
EXPE ~ IMAG
SAT  ~ IMAG + EXPE + QUAL + VAL
LOY  ~ IMAG + SAT
VAL  ~ EXPE + QUAL

# Measurement model
EXPE =~ expe1 + expe2 + expe3 + expe4 + expe5
IMAG =~ imag1 + imag2 + imag3 + imag4 + imag5
LOY  =~ loy1  + loy2  + loy3  + loy4
QUAL =~ qual1 + qual2 + qual3 + qual4 + qual5
SAT  =~ sat1  + sat2  + sat3  + sat4
VAL  =~ val1  + val2  + val3  + val4
"

## Estimate the model without resampling
a <- csem(satisfaction, model)

## Bootstrap and jackknife estimation
boot <- resamplecSEMResults(a)
jack <- resamplecSEMResults(a, .resample_method = "jackknife")

## Alternatively use .resample_method in csem()
boot_csem <- csem(satisfaction, model, .resample_method = "bootstrap")
jack_csem <- csem(satisfaction, model, .resample_method = "jackknife")

# ============================================================================
# Extended usage
# ============================================================================
### Double resampling  -------------------------------------------------------
# The confidence intervals (e.g. the bias-corrected and accelearated CI)
# require double resampling. Use .resample_method2 for this.

boot1 <- resamplecSEMResults(
  .object = a,
  .resample_method  = "bootstrap",
  .R                = 50,
  .resample_method2 = "bootstrap",
  .R2               = 20,
  .seed             = 1303
  )

## Again, this is identical to using csem
boot1_csem <- csem(
  .data             = satisfaction,
  .model            = model,
  .resample_method  = "bootstrap",
  .R                = 50,
  .resample_method2 = "bootstrap",
```

```
  .R2               = 20,
  .seed             = 1303
  )

identical(boot1, boot1_csem) # only true if .seed was set

### Inference ------------------------------------------------------------------
# To get inferencial quanitites such as the estimated standard error or
# the percentile confidence intervall for each resampled quantity use
# postestimation function infer()

inference <- infer(boot1)
inference$Path_estimates$sd
inference$Path_estimates$CI_percentile

# As usual summarize() can be called directly
summarize(boot1)

# In the example above .R x .R2 = 50 x 20 = 1000. Multiprocessing will be
# faster on most systems here and is therefore recommended. Note that multiprocessing
# does not affect the random number generation

boot2 <- resamplecSEMResults(
  .object           = a,
  .resample_method  = "bootstrap",
  .R                = 50,
  .resample_method2 = "bootstrap",
  .R2               = 20,
  .eval_plan        = "multisession",
  .seed             = 1303
  )

identical(boot1, boot2)
## End(Not run)
```

`resampleData` *Resample data*

Description

Resample data from a data set using common resampling methods. For bootstrap or jackknife resampling, package users usually do not need to call this function but directly use `resamplecSEMResults()` instead.

Usage

```
resampleData(
 .object          = NULL,
 .data            = NULL,
 .resample_method = c("bootstrap", "jackknife", "permutation",
```

```
                      "cross-validation"),
  .cv_folds        = 10,
  .id              = NULL,
  .R               = 499,
  .seed            = NULL
)
```

Arguments

`.object`	An R object of class cSEMResults resulting from a call to `csem()`.
`.data`	A `data.frame`, a `matrix` or a `list` of data of either type. Possible column types or classes of the data provided are: `"logical"`, `"numeric"` (`"double"` or `"integer"`), `"factor"` (ordered and unordered) or a mix of several types. The data may also include **one** character column whose column name must be given to `.id`. This column is assumed to contain group identifiers used to split the data into groups. If `.data` is provided, `.object` is ignored. Defaults to `NULL`.
`.resample_method`	Character string. The resampling method to use. One of: "*bootstrap*", "*jackknife*", "*permutation*", or "*cross-validation*". Defaults to "*bootstrap*".
`.cv_folds`	Integer. The number of cross-validation folds to use. Setting `.cv_folds` to `N` (the number of observations) produces leave-one-out cross-validation samples. Defaults to `10`.
`.id`	Character string or integer. A character string giving the name or an integer of the position of the column of `.data` whose levels are used to split `.data` into groups. Defaults to `NULL`.
`.R`	Integer. The number of bootstrap runs, permutation runs or cross-validation repetitions to use. Defaults to `499`.
`.seed`	Integer or `NULL`. The random seed to use. Defaults to `NULL` in which case an arbitrary seed is chosen. Note that the scope of the seed is limited to the body of the function it is used in. Hence, the global seed will not be altered!

Details

The function `resampleData()` is general purpose. It simply resamples data from a data set according to the resampling method provided via the `.resample_method` argument and returns a list of resamples. Currently, `bootstrap`, `jackknife`, `permutation`, and `cross-validation` (both leave-one-out (LOOCV) and k-fold cross-validation) are implemented.

The user may provide the data set to resample either explicitly via the `.data` argument or implicitly by providing a cSEMResults objects to `.object` in which case the original data used in the call that created the cSEMResults object is used for resampling. If both, a cSEMResults object and a data set via `.data` are provided the former is ignored.

As `csem()` accepts a single data set, a list of data sets as well as data sets that contain a column name used to split the data into groups, the cSEMResults object may contain multiple data sets. In this case, resampling is done by data set or group. Note that depending on the number of data sets/groups provided this computation may be slower as resampling will be repeated for each data set/group.

To split data provided via the .data argument into groups, the column name or the column index of the column containing the group levels to split the data must be given to .id. If data that contains grouping is taken from a cSEMResults object, .id is taken from the object information. Hence, providing .id is redundant in this case and therefore ignored.

The number of bootstrap or permutation runs as well as the number of cross-validation repetitions is given by .R. The default is 499 but should be increased in real applications. See e.g., Hesterberg (2015), p.380 for recommendations concerning the bootstrap. For jackknife .R is ignored as it is based on the N leave-one-out data sets.

Choosing resample_method = "permutation" for ungrouped data causes an error as permutation will simply reorder the observations which is usually not meaningful. If a list of data is provided each list element is assumed to represent the observations belonging to one group. In this case, data is pooled and group adherence permuted.

For cross-validation the number of folds (k) defaults to 10. It may be changed via the .cv_folds argument. Setting k = 2 (not 1!) splits the data into a single training and test data set. Setting k = N (where N is the number of observations) produces leave-one-out cross-validation samples. Note: 1.) At least 2 folds required (k > 1); 2.) k can not be larger than N; 3.) If N/k is not not an integer the last fold will have less observations.

Random number generation (RNG) uses the L'Ecuyer-CRMR RGN stream as implemented in the future.apply package (Bengtsson 2018). See ?future_lapply for details. By default a random seed is chosen.

Value

The structure of the output depends on the type of input and the resampling method:

Bootstrap If a matrix or data.frame without grouping variable is provided (i.e., .id = NULL), the result is a list of length .R (default 499). Each element of that list is a bootstrap (re)sample. If a grouping variable is specified or a list of data is provided (where each list element is assumed to contain data for one group), resampling is done by group. Hence, the result is a list of length equal to the number of groups with each list element containing .R bootstrap samples based on the N_g observations of group g.

Jackknife If a matrix or data.frame without grouping variable is provided (.id = NULL), the result is a list of length equal to the number of observations/rows (N) of the data set provided. Each element of that list is a jackknife (re)sample. If a grouping variable is specified or a list of data is provided (where each list element is assumed to contain data for one group), resampling is done by group. Hence, the result is a list of length equal to the number of group levels with each list element containing N jackknife samples based on the N_g observations of group g.

Permutation If a matrix or data.frame without grouping variable is provided an error is returned as permutation will simply reorder the observations. If a grouping variable is specified or a list of data is provided (where each list element is assumed to contain data of one group), group membership is permuted. Hence, the result is a list of length .R where each element of that list is a permutation (re)sample.

Cross-validation If a matrix or data.frame without grouping variable is provided a list of length .R is returned. Each list element contains a list containing the k splits/folds subsequently used as test and training data sets. If a grouping variable is specified or a list of data is provided (where each list element is assumed to contain data for one group), cross-validation is repeated

`.R` times for each group. Hence, the result is a list of length equal to the number of groups, each containing `.R` list elements (the repetitions) which in turn contain the k splits/folds.

References

Bengtsson H (2018). *future.apply: Apply Function to Elements in Parallel using Futures*. R package version 1.0.1, https://CRAN.R-project.org/package=future.apply.

Hesterberg TC (2015). "What Teachers Should Know About the Bootstrap: Resampling in the Undergraduate Statistics Curriculum." *The American Statistician*, **69**(4), 371–386. doi:10.1080/00031305.2015.1089789.

See Also

`csem()`, cSEMResults, `resamplecSEMResults()`

Examples

```
# ==============================================================================
# Using the raw data
# ==============================================================================
### Bootstrap (default) ---------------------------------------------------

res_boot1 <- resampleData(.data = satisfaction)
str(res_boot1, max.level = 3, list.len = 3)

## To replicate a bootstrap draw use .seed:
res_boot1a <- resampleData(.data = satisfaction, .seed = 2364)
res_boot1b <- resampleData(.data = satisfaction, .seed = 2364)

identical(res_boot1, res_boot1a) # TRUE

### Jackknife -------------------------------------------------------------

res_jack <- resampleData(.data = satisfaction, .resample_method = "jackknife")
str(res_jack, max.level = 3, list.len = 3)

### Cross-validation ------------------------------------------------------
## Create dataset for illustration:
dat <- data.frame(
  "x1" = rnorm(100),
  "x2" = rnorm(100),
  "group" = sample(c("male", "female"), size = 100, replace = TRUE),
  stringsAsFactors = FALSE)

## 10-fold cross-validation (repeated 100 times)
cv_10a <- resampleData(.data = dat, .resample_method = "cross-validation",
                       .R = 100)
str(cv_10a, max.level = 3, list.len = 3)

# Cross-validation can be done by group if a group identifyer is provided:
cv_10 <- resampleData(.data = dat, .resample_method = "cross-validation",
```

```
                      .id = "group", .R = 100)

## Leave-one-out-cross-validation (repeated 50 times)
cv_loocv  <- resampleData(.data = dat[, -3],
                          .resample_method = "cross-validation",
                          .cv_folds = nrow(dat),
                          .R = 50)
str(cv_loocv, max.level = 2, list.len = 3)

### Permuation ------------------------------------------------------------

res_perm <- resampleData(.data = dat, .resample_method = "permutation",
                         .id = "group")
str(res_perm, max.level = 2, list.len = 3)

# Forgetting to set .id causes an error
## Not run:
res_perm <- resampleData(.data = dat, .resample_method = "permutation")

## End(Not run)

# ============================================================================
# Using a cSEMResults object
# ============================================================================

model <- "
# Structural model
QUAL ~ EXPE
EXPE ~ IMAG
SAT  ~ IMAG + EXPE + QUAL + VAL
LOY  ~ IMAG + SAT
VAL  ~ EXPE + QUAL

# Measurement model
EXPE =~ expe1 + expe2 + expe3 + expe4 + expe5
IMAG =~ imag1 + imag2 + imag3 + imag4 + imag5
LOY  =~ loy1  + loy2  + loy3  + loy4
QUAL =~ qual1 + qual2 + qual3 + qual4 + qual5
SAT  =~ sat1  + sat2  + sat3  + sat4
VAL  =~ val1  + val2  + val3  + val4
"
a <- csem(satisfaction, model)

# Create bootstrap and jackknife samples
res_boot <- resampleData(a, .resample_method = "bootstrap", .R = 499)
res_jack <- resampleData(a, .resample_method = "jackknife")

# Since `satisfaction` is the dataset used the following approaches yield
# identical results.
res_boot_data   <- resampleData(.data = satisfaction, .seed = 2364)
res_boot_object <- resampleData(a, .seed = 2364)

identical(res_boot_data, res_boot_object) # TRUE
```

`Russett` *Data: Russett*

Description

A data frame containing 10 variables with 47 observations.

Usage

```
Russett
```

Format

A data frame containing the following variables for 47 countries:

`gini` The Gini index of concentration

`farm` The percentage of landholders who collectively occupy one-half of all the agricultural land (starting with the farmers with the smallest plots of land and working toward the largest)

`rent` The percentage of the total number of farms that rent all their land. Transformation: ln (x + 1)

`gnpr` The 1955 gross national product per capita in U.S. dollars. Transformation: ln (x)

`labo` The percentage of the labor force employed in agriculture. Transformation: ln (x)

`inst` Instability of personnel based on the term of office of the chief executive. Transformation: exp (x - 16.3)

`ecks` The total number of politically motivated violent incidents, from plots to protracted guerrilla warfare. Transformation: ln (x + 1)

`deat` The number of people killed as a result of internal group violence per 1,000,000 people. Transformation: ln (x + 1)

`stab` One if the country has a stable democracy, and zero otherwise

`dict` One if the country experiences a dictatorship, and zero otherwise

Details

The dataset was initially compiled by Russett (1964), discussed and reprinted by Gifi (1990), and partially transformed by Tenenhaus and Tenenhaus (2011). It is also used in Henseler (2021) for demonstration purposes.

Source

From: Henseler (2021)

References

Gifi A (1990). *Nonlinear multivariate analysis*. Wiley.

Henseler J (2021). *Composite-Based Structural Equation Modeling: Analyzing Latent and Emergent Variables*. Guilford Press, New York.

Russett BM (1964). "Inequality and Instability: The Relation of Land Tenure to Politics." *World Politics*, **16**(3), 442–454. doi:10.2307/2009581.

Tenenhaus A, Tenenhaus M (2011). "Regularized generalized canonical correlation analysis." *Psychometrika*, **76**(2), 257–284.

Examples

```
#============================================================================
# Example is taken from Henseler (2020)
#============================================================================
model_Russett="
# Composite model
AgrIneq <~ gini + farm + rent
IndDev  <~ gnpr + labo
PolInst <~ inst + ecks + deat + stab + dict

# Structural model
PolInst ~ AgrIneq + IndDev
"

out <- csem(.data = Russett, .model = model_Russett,
            .PLS_weight_scheme_inner = 'factorial',
            .tolerance = 1e-06
)
```

`satisfaction` *Data: satisfaction*

Description

A data frame with 250 observations and 27 variables. Variables from 1 to 27 refer to six latent concepts: `IMAG`=Image, `EXPE`=Expectations, `QUAL`=Quality, `VAL`=Value, `SAT`=Satisfaction, and `LOY`=Loyalty.

imag1-imag5 Indicators attached to concept `IMAG` which is supposed to capture aspects such as the institutions reputation, trustworthiness, seriousness, solidness, and caring about customer.

expe1-expe5 Indicators attached to concept `EXPE` which is supposed to capture aspects concerning products and services provided, customer service, providing solutions, and expectations for the overall quality.

qual1-qual5 Indicators attached to concept QUAL which is supposed to capture aspects concerning reliability of products and services, the range of products and services, personal advice, and overall perceived quality.

val1-val4 Indicators attached to concept VAL which is supposed to capture aspects related to beneficial services and products, valuable investments, quality relative to price, and price relative to quality.

sat1-sat4 Indicators attached to concept SAT which is supposed to capture aspects concerning overall rating of satisfaction, fulfillment of expectations, satisfaction relative to other banks, and performance relative to customer's ideal bank.

loy1-loy4 Indicators attached to concept LOY which is supposed to capture aspects concerning propensity to choose the same bank again, propensity to switch to other bank, intention to recommend the bank to friends, and the sense of loyalty.

Usage

```
satisfaction
```

Format

An object of class data.frame with 250 rows and 27 columns.

Details

This dataset contains the variables from a customer satisfaction study of a Spanish credit institution on 250 customers. The data is identical to the dataset provided by the plspm package but with the last column (gender) removed. If you are looking for the original dataset use the satisfaction_gender dataset.

Source

The plspm package (version 0.4.9). Original source according to **plspm**: "Laboratory of Information Analysis and Modeling (LIAM). Facultat d'Informatica de Barcelona, Universitat Politecnica de Catalunya".

---

satisfaction_gender *Data: satisfaction including gender*

Description

A data frame with 250 observations and 28 variables. Variables from 1 to 27 refer to six latent concepts: IMAG=Image, EXPE=Expectations, QUAL=Quality, VAL=Value, SAT=Satisfaction, and LOY=Loyalty.

imag1-imag5 Indicators attached to concept IMAG which is supposed to capture aspects such as the institutions reputation, trustworthiness, seriousness, solidness, and caring about customer.

expe1-expe5 Indicators attached to concept EXPE which is supposed to capture aspects concerning products and services provided, customer service, providing solutions, and expectations for the overall quality.

qual1-qual5 Indicators attached to concept QUAL which is supposed to capture aspects concerning reliability of products and services, the range of products and services, personal advice, and overall perceived quality.

val1-val4 Indicators attached to concept VAL which is supposed to capture aspects related to beneficial services and products, valuable investments, quality relative to price, and price relative to quality.

sat1-sat4 Indicators attached to concept SAT which is supposed to capture aspects concerning overall rating of satisfaction, fulfillment of expectations, satisfaction relative to other banks, and performance relative to customer's ideal bank.

loy1-loy4 Indicators attached to concept LOY which is supposed to capture aspects concerning propensity to choose the same bank again, propensity to switch to other bank, intention to recommend the bank to friends, and the sense of loyalty.

gender The sex of the respondent.

Usage

```
satisfaction_gender
```

Format

An object of class `data.frame` with 250 rows and 28 columns.

Details

This data set contains the variables from a customer satisfaction study of a Spanish credit institution on 250 customers. The data is taken from the plspm package. For convenience, there is a version of the dataset with the last column (`gender`) removed: satisfaction.

Source

The plspm package (version 0.4.9). Original source according to **plspm**: "Laboratory of Information Analysis and Modeling (LIAM). Facultat d'Informatica de Barcelona, Universitat Politecnica de Catalunya".

`savePlot` *savePlot*

Description

This function saves a given plot of a cSEMResults object to a specified file format.

Usage

```
savePlot(
 .plot_object,
 .filename,
 .path = NULL)
```

Arguments

.plot_object	Object returned by one of the following functions plot.cSEMResults_default(), plot.cSEMResults_multi(), or plot.cSEMResults_2ndorder().
.filename	Character string. The name of the file to save the plot to (supports 'pdf', 'png', 'svg', and 'dot' formats).
.path	Character string. Path of the directory to save the file to. Defaults to the current working directory.

See Also

plot.cSEMResults_default() plot.cSEMResults_multi() plot.cSEMResults_2ndorder()

Sigma_Summers_composites

Data: Summers

Description

A (18 x 18) indicator correlation matrix.

Usage

```
Sigma_Summers_composites
```

Format

An object of class matrix (inherits from array) with 18 rows and 18 columns.

Details

The indicator correlation matrix for a modified version of Summers (1965) model. All constructs are modeled as composites.

Source

Own calculation based on Dijkstra and Henseler (2015).

References

Dijkstra TK, Henseler J (2015). "Consistent and Asymptotically Normal PLS Estimators for Linear Structural Equations." *Computational Statistics & Data Analysis*, **81**, 10–23.

Summers R (1965). "A Capital Intensive Approach to the Small Sample Properties of Various Simultaneous Equation Estimators." *Econometrica*, **33**(1), 1–41.

Examples

```
require(cSEM)

model <- "
ETA1 ~ ETA2 + XI1 + XI2
ETA2 ~ ETA1 + XI3 +XI4

ETA1 ~~ ETA2

XI1  <~ x1 + x2 + x3
XI2  <~ x4 + x5 + x6
XI3  <~ x7 + x8 + x9
XI4  <~ x10 + x11 + x12
ETA1 <~ y1 + y2 + y3
ETA2 <~ y4 + y5 + y6
"

## Generate data
summers_dat <- MASS::mvrnorm(n = 300, mu = rep(0, 18),
                             Sigma = Sigma_Summers_composites, empirical = TRUE)

## Estimate
res <- csem(.data = summers_dat, .model = model) # inconsistent

##
# 2SLS
res_2SLS <- csem(.data = summers_dat, .model = model, .approach_paths = "2SLS",
                 .instruments = list(ETA1 = c('XI1', 'XI2', 'XI3', 'XI4'),
                                     ETA2 = c('XI1', 'XI2', 'XI3', 'XI4'))
)
```

SQ *Data: SQ*

Description

A data frame containing 23 variables with 411 observations. The original indicators were measured on a 6-point scale. In this version of the dataset, the indicators are scaled to be between 0 and 100.

Usage

```
SQ
```

Format

An object of class `data.frame` with 411 rows and 23 columns.

Details

The data comes from a European manufacturer of durable consumer goods and was studied by Bliemel et al. (2004) who focused on service quality. It is also used in Henseler (2021) for demonstration purposes, see the corresponding tutorial.

Source

The dataset is provided by Jörg Henseler.

References

Bliemel FW, Adolphs K, Henseler J (2004). "Reconceptualizing service quality. A formative measurement approach using PLS path modeling." In Munuera-Aleman JL (ed.), *Proceedings of the 33rd EMAC Conference*, 224.

Henseler J (2021). *Composite-Based Structural Equation Modeling: Analyzing Latent and Emergent Variables*. Guilford Press, New York.

`summarize` *Summarize model*

Description

[Stable]

Usage

```
summarize(
 .object = NULL,
 .alpha  = 0.05,
 .ci     = NULL,
 ...
 )
```

Arguments

`.object`	An R object of class cSEMResults resulting from a call to `csem()`.
`.alpha`	An integer or a numeric vector of significance levels. Defaults to `0.05`.
`.ci`	A vector of character strings naming the confidence interval to compute. For possible choices see `infer()`.
`...`	Further arguments to `summarize()`. Currently ignored.

Details

The summary is mainly focused on estimated parameters. For quality criteria such as the average variance extracted (AVE), reliability estimates, effect size estimates etc., use `assess()`.

If `.object` contains resamples, standard errors, t-values and p-values (assuming estimates are standard normally distributed) are printed as well. By default the percentile confidence interval is given as well. For other confidence intervals use the `.ci` argument. See `infer()` for possible choices and a description.

Value

An object of class `cSEMSummarize`. A `cSEMSummarize` object has the same structure as the cSEMResults object with a couple differences:

1. Elements `$Path_estimates`, `$Loadings_estimates`, `$Weight_estimates`, `$Weight_estimates`, and `$Residual_correlation` are standardized data frames instead of matrices.
2. Data frames `$Effect_estimates`, `$Indicator_correlation`, and `$Exo_construct_correlation` are added to `$Estimates`.

The data frame format is usually much more convenient if users intend to present the results in e.g., a paper or a presentation.

See Also

csem, `assess()`, cSEMResults, `exportToExcel()`

Examples

```
## Take a look at the dataset
#?threecommonfactors

## Specify the (correct) model
model <- "
# Structural model
eta2 ~ eta1
eta3 ~ eta1 + eta2

# (Reflective) measurement model
eta1 =~ y11 + y12 + y13
eta2 =~ y21 + y22 + y23
eta3 =~ y31 + y32 + y33
"

## Estimate
res <- csem(threecommonfactors, model, .resample_method = "bootstrap", .R = 40)

## Postestimation
res_summarize <- summarize(res)
res_summarize

# Extract e.g. the loadings
res_summarize$Estimates$Loading_estimates
```

```
## By default only the 95% percentile confidence interval is printed. User
## can have several confidence interval computed, however, only the first
## will be printed.

res_summarize <- summarize(res, .ci = c("CI_standard_t", "CI_percentile"),
                           .alpha = c(0.05, 0.01))
res_summarize

# Extract the loading including both confidence intervals
res_summarize$Estimates$Path_estimates
```

`Switching` *Data: Switching*

Description

A data frame containing 26 variables with 767 observations.

Usage

```
Switching
```

Format

An object of class `data.frame` with 767 rows and 26 columns.

Details

The data contains variables about the consumers' intention to switch a service provider. It is also used in Henseler (2021) for demonstration purposes, see the corresponding tutorial.

Source

The dataset is provided by Jörg Henseler.

References

Henseler J (2021). *Composite-Based Structural Equation Modeling: Analyzing Latent and Emergent Variables*. Guilford Press, New York.

Examples

```
#================================================================================
# Example is taken from Henseler (2021)
#================================================================================
model_Int <-"
# Measurement models
INV =~ INV1 + INV2 + INV3 +INV4
```

```
SAT =~ SAT1 + SAT2 + SAT3
INT =~ INT1 + INT2

# Structural model containing an interaction term.
INT ~ INV + SAT + INV.SAT
"

out <- csem(.data = Switching, .model = model_Int,
            .PLS_weight_scheme_inner = 'factorial',
            .tolerance = 1e-06)
```

`testCVPAT` *Perform a Cross-Validated Predictive Ability Test (CVPAT)*

Description

[Maturing]

Usage

```
testCVPAT(
.object1              = NULL,
.object2              = NULL,
.approach_predict     = c("earliest", "direct"),
.seed                 = NULL,
.cv_folds             = 10,
.handle_inadmissibles = c("stop", "ignore"),
.testtype             = c("twosided", "onesided"))
```

Arguments

`.object1`	An R object of class cSEMResults resulting from a call to `csem()`.
`.object2`	An R object of class cSEMResults resulting from a call to `csem()`.
`.approach_predict`	Character string. Which approach should be used to predictions? One of "*earliest*" and "*direct*". If "*earliest*" predictions for indicators associated to endogenous constructs are performed using only indicators associated to exogenous constructs. If "*direct*", predictions for indicators associated to endogenous constructs are based on indicators associated to their direct antecedents. Defaults to "*earliest*".
`.seed`	Integer or `NULL`. The random seed to use. Defaults to `NULL` in which case an arbitrary seed is chosen. Note that the scope of the seed is limited to the body of the function it is used in. Hence, the global seed will not be altered!
`.cv_folds`	Integer. The number of cross-validation folds to use. Setting `.cv_folds` to `N` (the number of observations) produces leave-one-out cross-validation samples. Defaults to `10`.

`.handle_inadmissibles`
: Character string. How should inadmissible results be treated? One of "*drop*", "*ignore*", or "*replace*". If "*drop*", all replications/resamples yielding an inadmissible result will be dropped (i.e. the number of results returned will potentially be less than `.R`). For "*ignore*" all results are returned even if all or some of the replications yielded inadmissible results (i.e. number of results returned is equal to `.R`). For "*replace*" resampling continues until there are exactly `.R` admissible solutions. Depending on the frequency of inadmissible solutions this may significantly increase computing time. Defaults to "*drop*".

`.testtype`
: Character string. One of "*twosided*" (H1: The models do not perform equally in predicting indicators belonging to endogenous constructs)" and *onesided*" (H1: Model 1 performs better in predicting indicators belonging to endogenous constructs than model2). Defaults to "*twosided*".

Details

Perform a Cross-Validated Predictive Ability Test (CVPAT) as described in (Liengaard et al. 2020). The predictive performance of two models based on the same dataset is compared. In doing so, the average difference in losses in predictions is compared for both models.

Value

An object of class `cSEMCVPAT` with print and plot methods. Technically, `cSEMCVPAT` is a named list containing the following list elements:

'$Information' Additional information.

References

Liengaard BD, Sharma PN, Hult GTM, Jensen MB, Sarstedt M, Hair JF, Ringle CM (2020). "Prediction: Coveted, Yet Forsaken? Introducing a Cross-Validated Predictive Ability Test in Partial Least Squares Path Modeling." *Decision Sciences*, **52**(2), 362–392. doi:10.1111/deci.12445.

See Also

csem, cSEMResults, `exportToExcel()`

Examples

```
### Anime example taken from https://github.com/ISS-Analytics/pls-predict/

# Load data
data(Anime) # data is similar to the Anime.csv found on
            # https://github.com/ISS-Analytics/pls-predict/ but with irrelevant
            # columns removed

# Split into training and data the same way as it is done on
# https://github.com/ISS-Analytics/pls-predict/
set.seed(123)

index     <- sample.int(dim(Anime)[1], 83, replace = FALSE)
```

```
dat_train <- Anime[-index, ]
dat_test  <- Anime[index, ]

# Specify model
model <- "
# Structural model

ApproachAvoidance ~ PerceivedVisualComplexity + Arousal

# Measurement/composite model

ApproachAvoidance         =~ AA0 + AA1 + AA2 + AA3
PerceivedVisualComplexity <~ VX0 + VX1 + VX2 + VX3 + VX4
Arousal                   <~ Aro1 + Aro2 + Aro3 + Aro4
"

# Estimate (replicating the results of the `simplePLS()` function)
res <- csem(dat_train,
            model,
            .disattenuate = FALSE, # original PLS
            .iter_max = 300,
            .tolerance = 1e-07,
            .PLS_weight_scheme_inner = "factorial"
)

# Predict using a user-supplied training data set
pp <- predict(res, .test_data = dat_test)
pp

### Compute prediction metrics  ------------------------------------------------
res2 <- csem(Anime, # whole data set
            model,
            .disattenuate = FALSE, # original PLS
            .iter_max = 300,
            .tolerance = 1e-07,
            .PLS_weight_scheme_inner = "factorial"
)

# Predict using 10-fold cross-validation
## Not run:
pp2 <- predict(res, .benchmark = "lm")
pp2
## There is a plot method available
plot(pp2)
## End(Not run)

### Example using OrdPLScPredict -----------------------------------------------
# Transform the numerical indicators into factors
## Not run:
data("BergamiBagozzi2000")
data_new <- data.frame(cei1    = as.ordered(BergamiBagozzi2000$cei1),
                       cei2    = as.ordered(BergamiBagozzi2000$cei2),
                       cei3    = as.ordered(BergamiBagozzi2000$cei3),
```

```
                        cei4    = as.ordered(BergamiBagozzi2000$cei4),
                        cei5    = as.ordered(BergamiBagozzi2000$cei5),
                        cei6    = as.ordered(BergamiBagozzi2000$cei6),
                        cei7    = as.ordered(BergamiBagozzi2000$cei7),
                        cei8    = as.ordered(BergamiBagozzi2000$cei8),
                        ma1     = as.ordered(BergamiBagozzi2000$ma1),
                        ma2     = as.ordered(BergamiBagozzi2000$ma2),
                        ma3     = as.ordered(BergamiBagozzi2000$ma3),
                        ma4     = as.ordered(BergamiBagozzi2000$ma4),
                        ma5     = as.ordered(BergamiBagozzi2000$ma5),
                        ma6     = as.ordered(BergamiBagozzi2000$ma6),
                        orgcmt1 = as.ordered(BergamiBagozzi2000$orgcmt1),
                        orgcmt2 = as.ordered(BergamiBagozzi2000$orgcmt2),
                        orgcmt3 = as.ordered(BergamiBagozzi2000$orgcmt3),
                        orgcmt5 = as.ordered(BergamiBagozzi2000$orgcmt5),
                        orgcmt6 = as.ordered(BergamiBagozzi2000$orgcmt6),
                        orgcmt7 = as.ordered(BergamiBagozzi2000$orgcmt7),
                        orgcmt8 = as.ordered(BergamiBagozzi2000$orgcmt8))

model <- "
# Measurement models
OrgPres =~ cei1 + cei2 + cei3 + cei4 + cei5 + cei6 + cei7 + cei8
OrgIden =~ ma1 + ma2 + ma3 + ma4 + ma5 + ma6
AffJoy  =~ orgcmt1 + orgcmt2 + orgcmt3 + orgcmt7
AffLove =~ orgcmt5 + orgcmt 6 + orgcmt8

# Structural model
OrgIden ~ OrgPres
AffLove ~ OrgIden
AffJoy  ~ OrgIden
"
# Estimate using cSEM; note: the fact that indicators are factors triggers OrdPLSc
res <- csem(.model = model, .data = data_new[1:250,])
summarize(res)

# Predict using OrdPLSPredict
set.seed(123)
pred <- predict(
  .object = res,
  .benchmark = "PLS-PM",
  .test_data = data_new[(251):305,],
   .treat_as_continuous = TRUE, .approach_score_target = "median"
  )

pred
round(pred$Prediction_metrics[, -1], 4)
## End(Not run)
```

---

testHausman *Regression-based Hausman test*

Description

[Experimental]

Usage

```
testHausman(
 .object               = NULL,
 .eval_plan            = c("sequential", "multicore", "multisession"),
 .handle_inadmissibles = c("drop", "ignore", "replace"),
 .R                    = 499,
 .resample_method      = c("bootstrap", "jackknife"),
 .seed                 = NULL
 )
```

Arguments

`.object`	An R object of class cSEMResults resulting from a call to `csem()`.
`.eval_plan`	Character string. The evaluation plan to use. One of "*sequential*", "*multicore*", or "*multisession*". In the two latter cases all available cores will be used. Defaults to "*sequential*".
`.handle_inadmissibles`	Character string. How should inadmissible results be treated? One of "*drop*", "*ignore*", or "*replace*". If "*drop*", all replications/resamples yielding an inadmissible result will be dropped (i.e. the number of results returned will potentially be less than `.R`). For "*ignore*" all results are returned even if all or some of the replications yielded inadmissible results (i.e. number of results returned is equal to `.R`). For "*replace*" resampling continues until there are exactly `.R` admissible solutions. Depending on the frequency of inadmissible solutions this may significantly increase computing time. Defaults to "*drop*".
`.R`	Integer. The number of bootstrap replications. Defaults to `499`.
`.resample_method`	Character string. The resampling method to use. One of: "*none*", "*bootstrap*" or "*jackknife*". Defaults to "*none*".
`.seed`	Integer or `NULL`. The random seed to use. Defaults to `NULL` in which case an arbitrary seed is chosen. Note that the scope of the seed is limited to the body of the function it is used in. Hence, the global seed will not be altered!

Details

Calculates the regression-based Hausman test to be used to compare OLS to 2SLS estimates or 2SLS to 3SLS estimates. See e.g., Wooldridge (2010) (pages 131 f.) for details.

The function is somewhat experimental. Only use if you know what you are doing.

References

Wooldridge JM (2010). *Econometric Analysis of Cross Section and Panel Data*, 2 edition. MIT Press.

See Also

csem(), cSEMResults

Examples

```
### Example from Dijkstra & Hensler (2015)
## Prepartion (values are from p. 15-16 of the paper)
Lambda <- t(kronecker(diag(6), c(0.7, 0.7, 0.7)))
Phi <- matrix(c(1.0000, 0.5000, 0.5000, 0.5000, 0.0500, 0.4000,
                0.5000, 1.0000, 0.5000, 0.5000, 0.5071, 0.6286,
                0.5000, 0.5000, 1.0000, 0.5000, 0.2929, 0.7714,
                0.5000, 0.5000, 0.5000, 1.0000, 0.2571, 0.6286,
                0.0500, 0.5071, 0.2929, 0.2571, 1.0000, sqrt(0.5),
                0.4000, 0.6286, 0.7714, 0.6286, sqrt(0.5), 1.0000),
              ncol = 6)

## Create population indicator covariance matrix
Sigma <- t(Lambda) %*% Phi %*% Lambda
diag(Sigma) <- 1
dimnames(Sigma) <- list(paste0("x", rep(1:6, each = 3), 1:3),
                        paste0("x", rep(1:6, each = 3), 1:3))

## Generate data
dat <- MASS::mvrnorm(n = 500, mu = rep(0, 18), Sigma = Sigma, empirical = TRUE)
# empirical = TRUE to show that 2SLS is in fact able to recover the true population
# parameters.

## Model to estimate
model <- "
## Structural model (nonrecurisve)
eta5 ~ eta6 + eta1 + eta2
eta6 ~ eta5 + eta3 + eta4

## Measurement model
eta1 =~ x11 + x12 + x13
eta2 =~ x21 + x22 + x23
eta3 =~ x31 + x32 + x33
eta4 =~ x41 + x42 + x43

eta5 =~ x51 + x52 + x53
eta6 =~ x61 + x62 + x63
"

library(cSEM)

## Estimate
res_ols <- csem(dat, .model = model, .approach_paths = "OLS")
sum_res_ols <- summarize(res_ols)

# Note: For the example the model-implied indicator correlation is irrelevant
#       the warnings can be ignored.
```

```
res_2sls <- csem(dat, .model = model, .approach_paths = "2SLS",
                 .instruments = list("eta5" = c('eta1','eta2','eta3','eta4'),
                                     "eta6" = c('eta1','eta2','eta3','eta4')))
sum_res_2sls <- summarize(res_2sls)
# Note that exogenous constructs are supplied as instruments for themselves!

## Test for endogeneity
test_ha <- testHausman(res_2sls, .R = 200)
test_ha
```

`testMGD` *Tests for multi-group comparisons*

Description

[Stable]

Usage

```
testMGD(
 .object                = NULL,
 .alpha                 = 0.05,
 .approach_p_adjust     = "none",
 .approach_mgd          = c("all", "Klesel", "Chin", "Sarstedt",
                            "Keil", "Nitzl", "Henseler", "CI_para","CI_overlap"),
 .output_type           = c("complete", "structured"),
 .parameters_to_compare = NULL,
 .eval_plan             = c("sequential", "multicore", "multisession"),
 .handle_inadmissibles  = c("replace", "drop", "ignore"),
 .R_permutation         = 499,
 .R_bootstrap           = 499,
 .saturated             = FALSE,
 .seed                  = NULL,
 .type_ci               = "CI_percentile",
 .type_vcv              = c("indicator", "construct"),
 .verbose               = TRUE
 )
```

Arguments

`.object`	An R object of class cSEMResults resulting from a call to `csem()`.
`.alpha`	An integer or a numeric vector of significance levels. Defaults to `0.05`.
`.approach_p_adjust`	Character string or a vector of character strings. Approach used to adjust the p-value for multiple testing. See the `methods` argument of `stats::p.adjust()` for a list of choices and their description. Defaults to "*none*".

`.approach_mgd` Character string or a vector of character strings. Approach used for the multigroup comparison. One of: "*all*", "*Klesel*", "*Chin*", "*Sarstedt*", "*Keil*, "*Nitzl*", "*Henseler*", "*CI_para*", or "*CI_overlap*". Default to "*all*" in which case all approaches are computed (if possible).

`.output_type` Character string. The type of output to return. One of "*complete*" or "*structured*". See the Value section for details. Defaults to "*complete*".

`.parameters_to_compare` A model in lavaan model syntax indicating which parameters (i.e, path (`~`), loadings (`=~`), weights (`<~`), or correlations (`~~`)) should be compared across groups. Defaults to `NULL` in which case all weights, loadings and path coefficients of the originally specified model are compared.

`.eval_plan` Character string. The evaluation plan to use. One of "*sequential*", "*multicore*", or "*multisession*". In the two latter cases all available cores will be used. Defaults to "*sequential*".

`.handle_inadmissibles` Character string. How should inadmissible results be treated? One of "*drop*", "*ignore*", or "*replace*". If "*drop*", all replications/resamples yielding an inadmissible result will be dropped (i.e. the number of results returned will potentially be less than `.R`). For "*ignore*" all results are returned even if all or some of the replications yielded inadmissible results (i.e. number of results returned is equal to `.R`). For "*replace*" resampling continues until there are exactly `.R` admissible solutions. Defaults to "*replace*" to accommodate all approaches.

`.R_permutation` Integer. The number of permutations. Defaults to `499`

`.R_bootstrap` Integer. The number of bootstrap runs. Ignored if `.object` contains resamples. Defaults to `499`

`.saturated` Logical. Should a saturated structural model be used? Defaults to `FALSE`.

`.seed` Integer or `NULL`. The random seed to use. Defaults to `NULL` in which case an arbitrary seed is chosen. Note that the scope of the seed is limited to the body of the function it is used in. Hence, the global seed will not be altered!

`.type_ci` Character string. Which confidence interval should be calculated? For possible choices, see the `.quantity` argument of the `infer()` function. Only used if `.approch_mgd` is one of "*CI_para*" or "*CI_overlap*". Ignored otherwise. Defaults to "*CI_percentile*".

`.type_vcv` Character string. Which model-implied correlation matrix should be calculated? One of "*indicator*" or "*construct*". Defaults to "*indicator*".

`.verbose` Logical. Should information (e.g., progress bar) be printed to the console? Defaults to `TRUE`.

Details

This function performs various tests proposed in the context of multigroup analysis.

The following tests are implemented:

`.approach_mgd = "Klesel"`**: Approach suggested by Klesel et al. (2019)** The model-implied variance-covariance matrix (either indicator (`.type_vcv = "indicator"`) or construct (`.type_vcv =`

"construct")) is compared across groups. If the model-implied indicator or construct correlation matrix based on a saturated structural model should be compared, set .saturated = TRUE. To measure the distance between the model-implied variance-covariance matrices, the geodesic distance (dG) and the squared Euclidean distance (dL) are used. If more than two groups are compared, the average distance over all groups is used.

.approach_mgd = "Sarstedt": **Approach suggested by Sarstedt et al. (2011)** Groups are compared in terms of parameter differences across groups. Sarstedt et al. (2011) tests if parameter k is equal across all groups. If several parameters are tested simultaneously it is recommended to adjust the significance level or the p-values (in **cSEM** correction is done by p-value). By default no multiple testing correction is done, however, several common adjustments are available via .approach_p_adjust. See stats::p.adjust() for details. Note: the test has some severe shortcomings. Use with caution.

.approach_mgd = "Chin": **Approach suggested by Chin and Dibbern (2010)** Groups are compared in terms of parameter differences across groups. Chin and Dibbern (2010) tests if parameter k is equal between two groups. If more than two groups are tested for equality, parameter k is compared between all pairs of groups. In this case, it is recommended to adjust the significance level or the p-values (in **cSEM** correction is done by p-value) since this is essentially a multiple testing setup. If several parameters are tested simultaneously, correction is by group and number of parameters. By default no multiple testing correction is done, however, several common adjustments are available via .approach_p_adjust. See stats::p.adjust() for details.

.approach_mgd = "Keil": **Approach suggested by Keil et al. (2000)** Groups are compared in terms of parameter differences across groups. Keil et al. (2000) tests if parameter k is equal between two groups. It is assumed, that the standard errors of the coefficients are equal across groups. The calculation of the standard error of the parameter difference is adjusted as proposed by Henseler et al. (2009). If more than two groups are tested for equality, parameter k is compared between all pairs of groups. In this case, it is recommended to adjust the significance level or the p-values (in **cSEM** correction is done by p-value) since this is essentially a multiple testing setup. If several parameters are tested simultaneously, correction is by group and number of parameters. By default no multiple testing correction is done, however, several common adjustments are available via .approach_p_adjust. See stats::p.adjust() for details.

.approach_mgd = "Nitzl": **Approach suggested by Nitzl (2010)** Groups are compared in terms of parameter differences across groups. Similarly to Keil et al. (2000), a single parameter k is tested for equality between two groups. In contrast to Keil et al. (2000), it is assumed, that the standard errors of the coefficients are unequal across groups (Sarstedt et al. 2011). If more than two groups are tested for equality, parameter k is compared between all pairs of groups. In this case, it is recommended to adjust the significance level or the p-values (in **cSEM** correction is done by p-value) since this is essentially a multiple testing setup. If several parameters are tested simultaneously, correction is by group and number of parameters. By default no multiple testing correction is done, however, several common adjustments are available via .approach_p_adjust. See stats::p.adjust() for details.

.approach_mgd = "Henseler": **Approach suggested by Henseler (2007)** Groups are compared in terms of parameter differences across groups. In doing so, the bootstrap estimates of one parameter are compared across groups. In the literature, this approach is also known as PLS-MGA. Originally, this test was proposed as an one-sided test. In this function we perform a left-sided and a right-sided test to investigate whether a parameter differs across two groups.

In doing so, the significance level is divided by 2 and compared to p-value of the left and right-sided test. Moreover, `.approach_p_adjust` is ignored and no overall decision is returned. For a more detailed description, see also Henseler et al. (2009).

`.approach_mgd = "CI_param"`**: Approach mentioned in Sarstedt et al. (2011)** This approach is based on the confidence intervals constructed around the parameter estimates of the two groups. If the parameter of one group falls within the confidence interval of the other group and/or vice versa, it can be concluded that there is no group difference. Since it is based on the confidence intervals `.approach_p_adjust` is ignored.

`.approach_mgd = "CI_overlap"`**: Approach mentioned in Sarstedt et al. (2011)** This approach is based on the confidence intervals (CIs) constructed around the parameter estimates of the two groups. If the two CIs overlap, it can be concluded that there is no group difference. Since it is based on the confidence intervals `.approach_p_adjust` is ignored.

Use `.approach_mgd` to choose the approach. By default all approaches are computed (`.approach_mgd = "all"`).

For convenience, two types of output are available. See the "Value" section below.

By default, approaches based on parameter differences across groups compare all parameters (`.parameters_to_compare = NULL`). To compare only a subset of parameters provide the parameters in lavaan model syntax just like the model to estimate. Take the simple model:

```
model_to_estimate <- "
Structural model
eta2 ~ eta1
eta3 ~ eta1 + eta2

# Each concept os measured by 3 indicators, i.e., modeled as latent variable
eta1 =~ y11 + y12 + y13
eta2 =~ y21 + y22 + y23
eta3 =~ y31 + y32 + y33
"
```

If only the path from eta1 to eta3 and the loadings of eta1 are to be compared across groups, write:

```
to_compare <- "
Structural parameters to compare
eta3 ~ eta1

# Loadings to compare
eta1 =~ y11 + y12 + y13
"
```

Note that the "model" provided to `.parameters_to_compare` does not need to be an estimable model!

Note also that compared to all other functions in **cSEM** using the argument, `.handle_inadmissibles` defaults to `"replace"` to accommodate the Sarstedt et al. (2011) approach.

Argument `.R_permuation` is ignored for the `"Nitzl"` and the `"Keil"` approach. `.R_bootstrap` is ignored if `.object` already contains resamples, i.e. has class `cSEMResults_resampled` and if only the `"Klesel"` or the `"Chin"` approach are used.

The argument .saturated is used by "Klesel" only. If .saturated = TRUE the original structural model is ignored and replaced by a saturated model, i.e. a model in which all constructs are allowed to correlate freely. This is useful to test differences in the measurement models between groups in isolation.

Value

If .output_type = "complete" a list of class cSEMTestMGD. Technically, cSEMTestMGD is a named list containing the following list elements:

$Information Additional information.

$Klesel A list with elements, Test_statistic, P_value, and Decision

$Chin A list with elements, Test_statistic, P_value, Decision, and Decision_overall

$Sarstedt A list with elements, Test_statistic, P_value, Decision, and Decision_overall

$Keil A list with elements, Test_statistic, P_value, Decision, and Decision_overall

$Nitzl A list with elements, Test_statistic, P_value, Decision, and Decision_overall

$Henseler A list with elements, Test_statistic, P_value, Decision, and Decision_overall

$CI_para A list with elements, Decision, and Decision_overall

$CI_overlap A list with elements, Decision, and Decision_overall

If .output_type = "structured" a tibble (data frame) with the following columns is returned.

Test The name of the test.

Comparision The parameter that was compared across groups. If "overall" the overall fit of the model was compared.

alpha% The test decision for a given "alpha" level. If TRUE the null hypotheses was rejected; if FALSE it was not rejected.

p-value_correction The p-value correction.

CI_type Only for the "CI_para" and the "CI_overlap" test. Which confidence interval was used.

Distance_metric Only for Test = "Klesel". Which distance metric was used.

References

Chin WW, Dibbern J (2010). "An Introduction to a Permutation Based Procedure for Multi-Group PLS Analysis: Results of Tests of Differences on Simulated Data and a Cross Cultural Analysis of the Sourcing of Information System Services Between Germany and the USA." In *Handbook of Partial Least Squares*, 171–193. Springer Berlin Heidelberg. doi:10.1007/9783540328278_8.

Henseler J (2007). "A new and simple approach to multi-group analysis in partial least squares path modeling." In Martens H, Næ s T (eds.), *Proceedings of PLS'07 - The 5th International Symposium on PLS and Related Methods*, 104–107. PLS, Norway: Matforsk, As.

Henseler J, Ringle CM, Sinkovics RR (2009). "The use of partial least squares path modeling in international marketing." *Advances in International Marketing*, **20**, 277–320. doi:10.1108/S1474-7979(2009)0000020014.

Keil M, Tan BC, Wei K, Saarinen T, Tuunainen V, Wassenaar A (2000). "A cross-cultural study on escalation of commitment behavior in software projects." *MIS Quarterly*, **24**(2), 299–325.

Klesel M, Schuberth F, Henseler J, Niehaves B (2019). "A Test for Multigroup Comparison Using Partial Least Squares Path Modeling." *Internet Research*, **29**(3), 464–477. doi:10.1108/intr112017-0418.

Nitzl C (2010). "Eine anwenderorientierte Einfuehrung in die Partial Least Square (PLS)-Methode." In *Arbeitspapier*, number 21. Universitaet Hamburg, Institut fuer Industrielles Management, Hamburg.

Sarstedt M, Henseler J, Ringle CM (2011). "Multigroup Analysis in Partial Least Squares (PLS) Path Modeling: Alternative Methods and Empirical Results." In *Advances in International Marketing*, 195–218. Emerald Group Publishing Limited. doi:10.1108/s14747979(2011)0000022012.

See Also

csem(), cSEMResults, testMICOM(), testOMF()

Examples

```
## Not run:
# ==============================================================================
# Basic usage
# ==============================================================================
model <- "
# Structural model
QUAL ~ EXPE
EXPE ~ IMAG
SAT  ~ IMAG + EXPE + QUAL + VAL
LOY  ~ IMAG + SAT
VAL  ~ EXPE + QUAL

# Measurement model

EXPE <~ expe1 + expe2 + expe3 + expe4 + expe5
IMAG <~ imag1 + imag2 + imag3 + imag4 + imag5
LOY  =~ loy1  + loy2  + loy3  + loy4
QUAL =~ qual1 + qual2 + qual3 + qual4 + qual5
SAT  <~ sat1  + sat2  + sat3  + sat4
VAL  <~ val1  + val2  + val3  + val4
"

## Create list of virtually identical data sets
dat <- list(satisfaction[-3,], satisfaction[-5, ], satisfaction[-10, ])
out <- csem(dat, model, .resample_method = "bootstrap", .R = 40)

## Test
testMGD(out, .R_permutation = 40,.verbose = FALSE)
```

```
# Notes:
#  1. .R_permutation (and .R in the call to csem) is small to make examples run quicker;
#     should be higher in real applications.
#  2. Test will not reject their respective H0s since the groups are virtually
#     identical.
#  3. Only exception is the approach suggested by Sarstedt et al. (2011), a
#     sign that the test is unreliable.
#  4. As opposed to other functions involving the argument,
#     '.handle_inadmissibles' the default is "replace" as this is
#     required by Sarstedt et al. (2011)'s approach.

# ============================================================================
# Extended usage
# ============================================================================
### Test only a subset ---------------------------------------------------
# By default all parameters are compared. Select a subset by providing a
# model in lavaan model syntax:

to_compare <- "
# Path coefficients
QUAL ~ EXPE

# Loadings
EXPE <~ expe1 + expe2 + expe3 + expe4 + expe5
"

## Test
testMGD(out, .parameters_to_compare = to_compare, .R_permutation = 20,
        .R_bootstrap = 20, .verbose = FALSE)

### Different p_adjustments ----------------------------------------------
# To adjust p-values to accommodate multiple testing use .approach_p_adjust.
# The number of tests to use for adjusting depends on the approach chosen. For
# the Chin approach for example it is the number of parameters to test times the
# number of possible group comparisons. To compare the results for different
# adjustments, a vector of p-adjustments may be chosen.

## Test
testMGD(out, .parameters_to_compare = to_compare,
        .approach_p_adjust = c("none", "bonferroni"),
        .R_permutation = 20, .R_bootstrap = 20, .verbose = FALSE)

## End(Not run)
```

`testMICOM` *Test measurement invariance of composites*

Description

[Stable]

Usage

```
testMICOM(
 .object               = NULL,
 .approach_p_adjust    = "none",
 .handle_inadmissibles = c("drop", "ignore", "replace"),
 .R                    = 499,
 .seed                 = NULL,
 .verbose              = TRUE
 )
```

Arguments

`.object`	An R object of class cSEMResults resulting from a call to `csem()`.
`.approach_p_adjust`	Character string or a vector of character strings. Approach used to adjust the p-value for multiple testing. See the `methods` argument of `stats::p.adjust()` for a list of choices and their description. Defaults to "*none*".
`.handle_inadmissibles`	Character string. How should inadmissible results be treated? One of "*drop*", "*ignore*", or "*replace*". If "*drop*", all replications/resamples yielding an inadmissible result will be dropped (i.e. the number of results returned will potentially be less than `.R`). For "*ignore*" all results are returned even if all or some of the replications yielded inadmissible results (i.e. number of results returned is equal to `.R`). For "*replace*" resampling continues until there are exactly `.R` admissible solutions. Depending on the frequency of inadmissible solutions this may significantly increase computing time. Defaults to "*drop*".
`.R`	Integer. The number of bootstrap replications. Defaults to `499`.
`.seed`	Integer or `NULL`. The random seed to use. Defaults to `NULL` in which case an arbitrary seed is chosen. Note that the scope of the seed is limited to the body of the function it is used in. Hence, the global seed will not be altered!
`.verbose`	Logical. Should information (e.g., progress bar) be printed to the console? Defaults to `TRUE`.

Details

The functions performs the permutation-based test for measurement invariance of composites across groups proposed by Henseler et al. (2016). According to the authors assessing measurement invariance in composite models can be assessed by a three-step procedure. The first two steps involve an assessment of configural and compositional invariance. The third steps involves mean and variance comparisons across groups. Assessment of configural invariance is qualitative in nature and hence not assessed by the `testMICOM()` function.

As `testMICOM()` requires at least two groups, `.object` must be of class `cSEMResults_multi`. As of version 0.2.0 of the package, `testMICOM()` does not support models containing second-order constructs.

It is possible to compare more than two groups, however, multiple-testing issues arise in this case. To adjust p-values in this case several p-value adjustments are available via the `approach_p_adjust` argument.

The remaining arguments set the number of permutation runs to conduct (`.R`), the random number seed (`.seed`), instructions how inadmissible results are to be handled (`handle_inadmissibles`), and whether the function should be verbose in a sense that progress is printed to the console.

The number of permutation runs defaults to `args_default()$.R` for performance reasons. According to Henseler et al. (2016) the number of permutations should be at least 5000 for assessment to be sufficiently reliable.

Value

A named list of class `cSEMTestMICOM` containing the following list element:

`$Step2` A list containing the results of the test for compositional invariance (Step 2).

`$Step3` A list containing the results of the test for mean and variance equality (Step 3).

`$Information` A list of additional information on the test.

References

Henseler J, Ringle CM, Sarstedt M (2016). "Testing Measurement Invariance of Composites Using Partial Least Squares." *International Marketing Review*, **33**(3), 405–431. doi:10.1108/imr092014-0304.

See Also

`csem()`, cSEMResults, `testOMF()`, `testMGD()`

Examples

```
## Not run:
# NOTE: to run the example. Download and load the newst version of cSEM.DGP
# from GitHub using devtools::install_github("M-E-Rademaker/cSEM.DGP").

# Create two data generating processes (DGPs) that only differ in how the composite
# X is build. Hence, the two groups are not compositionally invariant.
dgp1 <- "
# Structural model
Y ~ 0.6*X

# Measurement model
Y =~ 1*y1
X <~ 0.4*x1 + 0.8*x2

x1 ~~ 0.3125*x2
"

dgp2 <- "
# Structural model
Y ~ 0.6*X

# Measurement model
Y =~ 1*y1
X <~ 0.8*x1 + 0.4*x2
```

```
x1 ~~ 0.3125*x2
"

g1 <- generateData(dgp1, .N = 399, .empirical = TRUE) # requires cSEM.DGP
g2 <- generateData(dgp2, .N = 200, .empirical = TRUE) # requires cSEM.DGP

# Model is the same for both DGPs
model <- "
# Structural model
Y ~ X

# Measurement model
Y =~ y1
X <~ x1 + x2
"

# Estimate
csem_results <- csem(.data = list("group1" = g1, "group2" = g2), model)

# Test
testMICOM(csem_results, .R = 50, .alpha = c(0.01, 0.05), .seed = 1987)

## End(Not run)
```

`testOMF` *Test for overall model fit*

Description

[Maturing]

Usage

```
testOMF(
 .object               = NULL,
 .alpha                = 0.05,
 .fit_measures         = FALSE,
 .handle_inadmissibles = c("drop", "ignore", "replace"),
 .R                    = 499,
 .saturated            = FALSE,
 .seed                 = NULL,
 ...
)
```

Arguments

`.object`	An R object of class cSEMResults resulting from a call to `csem()`.
`.alpha`	An integer or a numeric vector of significance levels. Defaults to `0.05`.

`.fit_measures` Logical. (EXPERIMENTAL) Should additional fit measures be included? Defaults to `FALSE`.

`.handle_inadmissibles`
Character string. How should inadmissible results be treated? One of "*drop*", "*ignore*", or "*replace*". If "*drop*", all replications/resamples yielding an inadmissible result will be dropped (i.e. the number of results returned will potentially be less than `.R`). For "*ignore*" all results are returned even if all or some of the replications yielded inadmissible results (i.e. number of results returned is equal to `.R`). For "*replace*" resampling continues until there are exactly `.R` admissible solutions. Depending on the frequency of inadmissible solutions this may significantly increase computing time. Defaults to "*drop*".

`.R` Integer. The number of bootstrap replications. Defaults to `499`.

`.saturated` Logical. Should a saturated structural model be used? Defaults to `FALSE`.

`.seed` Integer or `NULL`. The random seed to use. Defaults to `NULL` in which case an arbitrary seed is chosen. Note that the scope of the seed is limited to the body of the function it is used in. Hence, the global seed will not be altered!

`...` Can be used to determine the fitting function used in the calculateGFI function.

Details

Bootstrap-based test for overall model fit originally proposed by Beran and Srivastava (1985). See also Dijkstra and Henseler (2015) who first suggested the test in the context of PLS-PM.

By default, `testOMF()` tests the null hypothesis that the population indicator correlation matrix equals the population model-implied indicator correlation matrix. Several discrepancy measures may be used. By default, `testOMF()` uses four distance measures to assess the distance between the sample indicator correlation matrix and the estimated model-implied indicator correlation matrix, namely the geodesic distance, the squared Euclidean distance, the standardized root mean square residual (SRMR), and the distance based on the maximum likelihood fit function. The reference distribution for each test statistic is obtained by the bootstrap as proposed by Beran and Srivastava (1985).

It is possible to perform the bootstrap-based test using fit measures such as the CFI, RMSEA or the GFI if `.fit_measures = TRUE`. This is experimental. To the best of our knowledge the applicability and usefulness of the fit measures for model fit assessment have not been formally (statistically) assessed yet. Theoretically, the logic of the test applies to these fit indices as well. Hence, their applicability is theoretically justified. Only use if you know what you are doing.

If `.saturated = TRUE` the original structural model is ignored and replaced by a saturated model, i.e., a model in which all constructs are allowed to correlate freely. This is useful to test misspecification of the measurement model in isolation.

Value

A list of class `cSEMTestOMF` containing the following list elements:

`$Test_statistic` The value of the test statistics.

`$Critical_value` The corresponding critical values obtained by the bootstrap.

`$Decision` The test decision. One of: `FALSE` (**Reject**) or `TRUE` (**Do not reject**).

`$Information` The `.R` bootstrap values; The number of admissible results; The seed used and the number of total runs.

References

Beran R, Srivastava MS (1985). "Bootstrap Tests and Confidence Regions for Functions of a Covariance Matrix." *The Annals of Statistics*, **13**(1), 95–115. doi:10.1214/aos/1176346579.

Dijkstra TK, Henseler J (2015). "Consistent and Asymptotically Normal PLS Estimators for Linear Structural Equations." *Computational Statistics & Data Analysis*, **81**, 10–23.

See Also

csem(), calculateSRMR(), calculateDG(), calculateDL(), cSEMResults, testMICOM(), testMGD(), exportToExcel()

Examples

```
# ===========================================================================
# Basic usage
# ===========================================================================
model <- "
# Structural model
eta2 ~ eta1
eta3 ~ eta1 + eta2

# (Reflective) measurement model
eta1 =~ y11 + y12 + y13
eta2 =~ y21 + y22 + y23
eta3 =~ y31 + y32 + y33
"

## Estimate
out <- csem(threecommonfactors, model, .approach_weights = "PLS-PM")

## Test
testOMF(out, .R = 50, .seed = 320)
```

---

threecommonfactors *Data: threecommonfactors*

Description

A dataset containing 500 standardized observations on 9 indicator generated from a population model with three concepts modeled as common factors.

Usage

```
threecommonfactors
```

Format

A matrix with 500 rows and 9 variables:

y11-y13 Indicators attached to the first common factor (`eta1`). Population loadings are: 0.7; 0.7; 0.7

y21-y23 Indicators attached to the second common factor (`eta2`). Population loadings are: 0.5; 0.7; 0.8

y31-y33 Indicators attached to the third common factor (`eta3`). Population loadings are: 0.8; 0.75; 0.7

The model is:

$$`eta2` = gamma1 * `eta1` + zeta1$$

$$`eta3` = gamma2 * `eta1` + beta * `eta2` + zeta2$$

with population values `gamma1` = 0.6, `gamma2` = 0.4 and `beta` = 0.35.

Examples

```
#============================================================================
# Correct model (the model used to generate the data)
#============================================================================
model_correct <- "
# Structural model
eta2 ~ eta1
eta3 ~ eta1 + eta2

# Measurement model
eta1 =~ y11 + y12 + y13
eta2 =~ y21 + y22 + y23
eta3 =~ y31 + y32 + y33
"

a <- csem(threecommonfactors, model_correct)

## The overall model fit is evidently almost perfect:
testOMF(a, .R = 30) # .R = 30 to speed up the example
```

`verify` *Verify admissibility*

Description

[Stable]

Usage

```
verify(.object)
```

Arguments

`.object`	An R object of class cSEMResults resulting from a call to `csem()`.

Details

Verify admissibility of the results obtained using `csem()`.

Results exhibiting one of the following defects are deemed inadmissible: non-convergence of the algorithm used to obtain weights, loadings and/or (congeneric) reliabilities larger than 1, a construct variance-covariance (VCV) and/or model-implied VCV matrix that is not positive semi-definite.

If `.object` is of class `cSEMResults_2ndorder` (i.e., estimates are based on a model containing second-order constructs) both the first and the second stage are checked separately.

Currently, a model-implied indicator VCV matrix for nonlinear model is not available. `verify()` therefore skips the check for positive definiteness of the model-implied indicator VCV matrix for nonlinear models and returns "ok".

Value

A logical vector indicating which (if any) problem occurred. A `FALSE` indicates that the specific problem did not occurred. For models containing second-order constructs estimated by the two/three-stage approach, a list of two such vectors (one for the first and one for the second stage) is returned. Status codes are:

- 1: The algorithm has converged.
- 2: All absolute standardized loading estimates are smaller than or equal to 1. A violation implies either a negative variance of the measurement error or a correlation larger than 1.
- 3: The construct VCV is positive semi-definite.
- 4: All reliability estimates are smaller than or equal to 1.
- 5: The model-implied indicator VCV is positive semi-definite. This is only checked for linear models (including models containing second-order constructs).

See Also

`csem()`, `summarize()`, cSEMResults

Examples

```
### Without higher order constructs ------------------------------------------
model <- "
# Structural model
eta2 ~ eta1
eta3 ~ eta1 + eta2

# (Reflective) measurement model
eta1 =~ y11 + y12 + y13
eta2 =~ y21 + y22 + y23
eta3 =~ y31 + y32 + y33
"
```

```
# Estimate
out <- csem(threecommonfactors, model)

# Check admissibility
verify(out) # ok!

## Examine the structure of a cSEMVerify object
str(verify(out))

### With higher order constructs ------------------------------------------------
# If the model containes higher order constructs both the first and the second-
# stage estimates estimates are checked for admissibility

## Not run:
require(cSEM.DGP) # download from https://m-e-rademaker.github.io/cSEM.DGP/

# Create DGP with 2nd order construct. Loading for indicator y51 is set to 1.1
# to produce a failing first stage model

dgp_2ndorder <- "
## Path model / Regressions
eta2 ~ 0.5*eta1
eta3 ~ 0.35*eta1 + 0.4*eta2

## Composite model
eta1 =~ 0.8*y41 + 0.6*y42 + 0.6*y43
eta2 =~ 1.1*y51 + 0.7*y52 + 0.7*y53
c1   =~ 0.8*y11 + 0.4*y12
c2   =~ 0.5*y21 + 0.3*y22

## Higher order composite
eta3 =~ 0.4*c1 + 0.4*c2
"

dat <- generateData(dgp_2ndorder) # requires the cSEM.DGP package
out <- csem(dat, .model = dgp_2ndorder)

verify(out) # not ok

## End(Not run)
```

`Yooetal2000` *Data: Yooetal2000*

Description

A data frame containing 34 variables with 569 observations.

Usage

```
Yooetal2000
```

Format

An object of class `data.frame` with 569 rows and 34 columns.

Details

The data is simulated and has the identical correlation matrix as the data that was analysed by Yoo et al. (2000) to examine how five elements of the marketing mix, namely price, store image, distribution intensity, advertising spending, and price deals, are related to the so-called dimensions of brand equity, i.e., perceived brand quality, brand loyalty, and brand awareness/associations. It is also used in Henseler (2017) and Henseler (2021) for demonstration purposes, see the corresponding tutorial.

Source

Simulated data with the same correlation matrix as the data studied by Yoo et al. (2000).

References

Henseler J (2017). "Bridging Design and Behavioral Research With Variance-Based Structural Equation Modeling." *Journal of Advertising*, **46**(1), 178–192. doi:10.1080/00913367.2017.1281780.

Henseler J (2021). *Composite-Based Structural Equation Modeling: Analyzing Latent and Emergent Variables*. Guilford Press, New York.

Yoo B, Donthu N, Lee S (2000). "An Examination of Selected Marketing Mix Elements and Brand Equity." *Journal of the Academy of Marketing Science*, **28**(2), 195–211. doi:10.1177/0092070300282002.

Examples

```
#============================================================================
# Example is taken from Henseler (2021)
#============================================================================
model_HOC="
# Measurement models FOC
PR =~ PR1 + PR2 + PR3
IM =~ IM1 + IM2 + IM3
DI =~ DI1 + DI2 + DI3
AD =~ AD1 + AD2 + AD3
DL =~ DL1 + DL2 + DL3
AA =~ AA1 + AA2 + AA3 + AA4 + AA5 + AA6
LO =~ LO1 + LO3
QL =~ QL1 + QL2 + QL3 + QL4 + QL5 + QL6

# Composite model for SOC
BR <~ QL + LO + AA

# Structural model
BR~ PR + IM + DI + AD + DL
"
```

```
out <- csem(.data = Yooetal2000, .model = model_HOC,
            .PLS_weight_scheme_inner = 'factorial',
            .tolerance = 1e-06)
```

Index