# Package 'matrixpls'

March 2, 2020

**Encoding** UTF-8

**Type** Package

**Title** Matrix-Based Partial Least Squares Estimation

**Version** 1.0.9

**Date** 2020-03-02

**Maintainer** Mikko Rönkkö <mikko.ronkko@jyu.fi>

**Description** Partial Least Squares Path Modeling
algorithm and related algorithms. The algorithm implementations aim for computational efficiency using matrix algebra and covariance data. The package is designed toward Monte Carlo simulations and includes functions to perform simple Monte Carlo simulations.

**URL** https://github.com/mronkko/matrixpls

**BugReports** https://github.com/mronkko/matrixpls/issues

**ByteCompile** TRUE

**Imports** assertive, matrixcalc, lavaan, MASS, methods, psych

**Suggests** plspm, simsem, RUnit, parallel, semPLS, boot, Matrix, ASGSCA, knitr, R.rsp

**License** GPL-3

**LazyLoad** yes

**RoxygenNote** 7.0.2

**Collate** 'convCheck.R' 'estimator.R' 'innerEstim.R' 'matrixpls.R' 'matrixpls.boot.R' 'matrixpls.crossvalidate.R' 'matrixpls.optim.R' 'matrixpls.plspm.R' 'matrixpls.postestimation.R' 'matrixpls.predict.R' 'matrixpls.sempls.R' 'matrixpls.sim.R' 'matrixpls.util.R' 'outerEstim.R' 'paramEstimator.R' 'reliabilityEstim.R' 'signChange.R' 'weightFun.R' 'weightSign.R'

**VignetteBuilder** R.rsp

**NeedsCompilation** no

**Author** Mikko Rönkkö [aut, cre]

**Repository** CRAN

**Date/Publication** 2020-03-02 19:50:09 UTC

# R topics documented:

`matrixpls-package` *Matrix-based Partial Least Squares estimation*

## Description

**matrixpls** calculates composite variable models using partial least squares (PLS) algorithm and related methods. In contrast to most other PLS software which implement the raw data version of the algorithm, **matrixpls** works with data covariance matrices. The algorithms are designed to be computationally efficient, modular in programming, and well documented. **matrixpls** integrates with **simsem** to enable Monte Carlo simulations with as little custom programming as possible.

## Details

**matrixpls** calculates models where sets of indicator variables are combined as weighted composites. These composites are then used to estimate a statistical model describing the relationships between the composites and composites and indicators. While a number of such methods exists, the partial least squares (PLS) technique is perhaps the most widely used.

The **matrixpls** package implements a collection of PLS techniques as well as the more recent GSCA and PLSc techniques and older methods based on analysis with composite variables, such as regression with unit weighted composites or factor scores. The package provides a unified framework that enables the comparison and analysis of these algorithms. In contrast to previous R packages for PLS, such as **plspm** and **semPLS** and all currently available commercial PLS software, which work with raw data, **matrixpls** calculates the indicator weights and model estimates from data covariance matrices. Working with covariance data allows for reanalyzing covariance matrices that are sometimes published as appendices of articles, is computationally more efficient, and lends itself more easily for formal analysis than implementations based on raw data.

**matrixpls** has modular design that is easily expanded and contains more calculation options than the two other PLS packages for R. To allow validation of the algorithms by end users and to help porting existing analysis files from the two other R packages to **matrixpls**, the package contains compatibility functions for both **plspm** and **semPLS**.

The desing principles and functionality of the package is best explained by first explaining the main function `matrixpls`. The function performs two tasks. It first calculates a set of indicator weights to form composites based on data covariance matrix and then estimates a statistical model with the indicators and composites using the weights. The main function takes the following arguments:

```
matrixpls(S, model, W.model = NULL,
          weightFun = weightFun.pls,
          parameterEstim = parameterEstim.separate,
          weightSign = NULL, ...,
          validateInput = TRUE, standardize = TRUE)
```

The first five arguments of `matrixpls` are most relevant for understanding how the package works. `S`, is the data covariance or correlation matrix. `model` defines the model which is estimated in the second stage and `W.model` defines how the indicators are to be aggregated as composites. If `W.model` is left undefined, it will be constructed based on `model` following rules that are explained elsewhere in the documentation. `weightFun` and `parameterEstim` are functions that implement the

first and second task of the function respectively. All other arguments are passed down to these two functions, which in turn can pass arguments to other functions that they call.

Many of the commonly used arguments of `matrixpls` function are functions themselves. For example, executing a PLS analysis with Mode B outer estimation for all indicator blocks and centroid inner estimation could be specified as follows:

```
matrixpls(S, model,
          outerEstim = outerEstim.modeB,
          innerEstim = innerEstim.centroid)
```

The arguments `outerEstim` and `innerEstim` are not defined by the `matrixpls` function, but are passed down to `weightFun.pls` which is used as the default `weightFun`. `outerEstim.modeB` and `innerEstim.centroid` are themselves functions provided by the **matrixpls** package, which perform the actual inner and outer estimation stages of the PLS algorithm. Essentially, all parts of the estimation algorithm can be provided as arguments for the main function. This allows for adjusting the inner workings of the algorithm in a way that is currently not possible with any other PLS software.

It is also possible to define custom functions. For example, we could define a new Mode B outer estimator that only produces positive weights by creating a custom function:

```
myModeB <- function(...){
  abs(outerEstim.ModeB(...))
}

matrixpls(S, model,
          outerEstim = myModeB,
          innerEstim = innerEstim.centroid)
```

Model can be specified in the lavaan format or the native matrixpls format. The native model format is a list of three binary matrices, `inner`, `reflective`, and `formative` specifying the free parameters of a model: `inner` (`l` x `l`) specifies the regressions between composites, `reflective` (`k` x `l`) specifies the regressions of observed data on composites, and `formative` (`l` x `k`) specifies the regressions of composites on the observed data. Here `k` is the number of observed variables and `l` is the number of composites.

If the model is specified in lavaan format, the native format model is derived from this model by assigning all regressions between latent variables to `inner`, all factor loadings to `reflective`, and all regressions of latent variables on observed variables to `formative`. Regressions between observed variables and all free covariances are ignored. All parameters that are specified in the model will be treated as free parameters.

The original papers about Partial Least Squares, as well as many of the current PLS implementations, impose restrictions on the matrices `inner`, `reflective`, and `formative`: `inner` must be a lower triangular matrix, `reflective` must have exactly one non-zero value on each row and must have at least one non-zero value on each column, and `formative` must only contain zeros. Some PLS implementations allow `formative` to contain non-zero values, but impose a restriction that the sum of `reflective` and `t(formative)` must satisfy the original restrictions of `reflective`. The only restrictions that matrixpls imposes on `inner`, `reflective`, and `formative` is that these must be binary matrices and that the diagonal of `inner` must be zeros.

## Author(s)

**Maintainer**: Mikko Rönkkö <mikko.ronkko@jyu.fi>

## References

Rosseel, Y. (2012). lavaan: An R Package for Structural Equation Modeling. *Journal of Statistical Software*, 48(2), 1–36. Retrieved from http://www.jstatsoft.org/v48/i02

Lohmöller J.-B. (1989) *Latent variable path modeling with partial least squares.* Heidelberg: Physica-Verlag.

Rönkkö, M., McIntosh, C. N., & Antonakis, J. (2015). On the adoption of partial least squares in psychological research: Caveat emptor. *Personality and Individual Differences*, (87), 76–84. DOI:10.1016/j.paid.2015.07.019

Wold, H. (1982). Soft modeling - The Basic Design And Some Extensions. In K. G. Jöreskog & S. Wold (Eds.),*Systems under indirect observation: causality, structure, prediction* (pp. 1–54). Amsterdam: North-Holland.

## See Also

Useful links:

- https://github.com/mronkko/matrixpls
- Report bugs at https://github.com/mronkko/matrixpls/issues

---

# ave *Average Variance Extracted indices for matrixpls results*

---

## Description

The `matrixpls` method for the generic function `ave` computes Average Variance Extracted indices for the model using the formula presented by Fornell and Larcker (1981).

## Usage

```
ave(object, ...)
```

## Arguments

| | |
|---|---|
| `object` | matrixpls estimation result object produced by the `matrixpls` function. |
| `...` | All other arguments are ignored. |

## Value

A list containing the Average Variance Extracted indices in the first position and the differences between aves and largest squared correlations with other composites in the second position.

## References

Fornell, C., & Larcker, D. F. (1981). Evaluating structural equation models with unobservable variables and measurement error. *Journal of marketing research*, 18(1), 39–50.

## See Also

Other post-estimation functions: `cr()`, `effects.matrixpls()`, `fitSummary()`, `fitted.matrixpls()`, `gof()`, `htmt()`, `loadings()`, `predict.matrixpls()`, `r2()`, `residuals.matrixpls()`

---

`convCheck` *Convergence checks*

---

## Description

The convergence check functions compare two weight matrices and returns the value of the convergence criterion describing the difference between the two weight matrices.

## Usage

```
convCheck.relative(Wnew, Wold)

convCheck.square(Wnew, Wold)

convCheck.absolute(Wnew, Wold)
```

## Arguments

| | |
|---|---|
| `Wnew` | Current weight matrix |
| `Wold` | Weight matrix from the previous iteration |

## Value

Scalar value of the convergence criterion

## Functions

- `convCheck.relative`: maximum of relative differences between weights from two iterations
- `convCheck.square`: maximum of squared absolute differences between weights from two iterations.
- `convCheck.absolute`: maximum of absolute differences between weights from two iterations

## cr *Composite Reliability indices for matrixpls results*

### Description

The `matrixpls` method for the generic function `cr` computes Composite Reliability indices for the model using the formula presented by Fornell and Larcker (1981).

### Usage

```
cr(object, ...)
```

### Arguments

| | |
|---|---|
| `object` | matrixpls estimation result object produced by the `matrixpls` function. |
| `...` | All other arguments are ignored. |

### Value

A named numeric vector containing the Composite Reliability indices.

### References

Fornell, C., & Larcker, D. F. (1981). Evaluating structural equation models with unobservable variables and measurement error. *Journal of marketing research*, 18(1), 39–50.

Aguirre-Urreta, M. I., Marakas, G. M., & Ellis, M. E. (2013). Measurement of composite reliability in research using partial least squares: Some issues and an alternative approach. *The DATA BASE for Advances in Information Systems*, 44(4), 11–43. DOI:10.1145/2544415.2544417

### See Also

Other post-estimation functions: `ave()`, `effects.matrixpls()`, `fitSummary()`, `fitted.matrixpls()`, `gof()`, `htmt()`, `loadings()`, `predict.matrixpls()`, `r2()`, `residuals.matrixpls()`

## effects.matrixpls *Total, Direct, and Indirect Effects for matrixpls results*

### Description

The `matrixpls` method for the standard generic function `effects` computes total, direct, and indirect effects for a matrixpls results according to the method described in Fox (1980).

Adapted from the `effects` function of the `sem` package

## Usage

```
## S3 method for class 'matrixpls'
effects(object, ...)
```

## Arguments

| | |
|---|---|
| `object` | matrixpls estimation result object produced by the `matrixpls` function. |
| `...` | All other arguments are ignored. |

## Value

A list with `Total`, `Direct`, and `Indirect` elements.

## References

Fox, J. (1980) Effect analysis in structural equation models: Extensions and simplified methods of computation. *Sociological Methods and Research* 9, 3–28.

## See Also

Other post-estimation functions: `ave()`, `cr()`, `fitSummary()`, `fitted.matrixpls()`, `gof()`, `htmt()`, `loadings()`, `predict.matrixpls()`, `r2()`, `residuals.matrixpls()`

---

# estimator *Parameter estimation of a model matrix*

---

## Description

Estimates the parameters of a model matrix (`inner`, `reflective`, or `formative`).

## Usage

```
estimator.ols(S, modelMatrix, W, ..., C = NULL, IC = NULL, n = NULL)

estimator.tsls(S, modelMatrix, W, ..., C)

estimator.plscLoadings(S, modelMatrix, W, ...)

estimator.efaLoadings(S, modelMatrix, W, ..., fm = "minres")

estimator.cfaLoadings(S, modelMatrix, W, ...)

estimator.plsreg(S, modelMatrix, W, ..., C)
```

## Arguments

| | |
|---|---|
| S | Covariance matrix of the data. |
| modelMatrix | A model matrix with dependent variables on rows, independent variables on colums, and non-zero elements indicating relationships. Can be either inner, reflective, or formative matrix. |
| W | Weight matrix, where the indicators are on colums and composites are on the rows. |
| ... | All other arguments are either ignored or passed through to third party estimation functions. |
| C | Correlation matrix of the composites. |
| IC | Correlation matrix of the composites and indicators. |
| n | sample size, used for calculating standard errors. |
| fm | factoring method for estimating the factor loadings. Passed through to fa. |

## Details

Parameter estimation functions estimate the parameters of a model matrix (inner, reflective, or formative). These functions can be used as parametersInner, parametersReflective, and parametersFormative arguments for parameterEstim.separate.

When two-stage least squares regression is applied with estimator.tsls, all exogenous variables are used as instrumental varaables. There is currently no check of whether the number of instrumental variables is sufficient for estimation.

estimator.plscLoadings estimates the loadings by scaling the weights W with the correction factor c presented by Dijkstra (2011). This produces a MINRES estimator, which constraints the loadings to be proportional to the weights. The PLSc code is loosely based on code contributed by Wenjing Huang and developed with the guidance by Theo Dijkstra.

estimator.plscLoadings estimates loadings with an unconstrained single factor model, which requires at least three indicators per block. The loadings of single indicator factors are estimated as 1 and two indicator factors as estimated by the square root of the indicator correlation.

Providing C or IC allows for using disattenuated or otherwise adjusted correlation matrices. If not provided, these matrices are calculated using S and W.

A part of the code for estimator.plsreg is adopted from the **plspm** package, licenced under GPL3.

## Value

A matrix with estimated parameters or a list of two matrices containing estimates and standard errors.

## Functions

- estimator.ols: parameter estimation with OLS regression. Can be applied to inner, reflective, or formative matrix.
- estimator.tsls: parameter estimation with two-stage least squares regression. For inner matrix only.

- `estimator.plscLoadings`: parameter estimation with Dijkstra's (2011) PLSc correction for loadings. For `reflective` matrix only.
- `estimator.efaLoadings`: parameter estimation with one indicator block at at time with exploratory factor analysis. Minres estimation is implemented natively and all other estimation techniques use the `fa` function from the `psych` package. For `reflective` matrix only.
- `estimator.cfaLoadings`: Estimates a maximum likelihood confirmatory factor analysis with `lavaan`. For `reflective` matrix only.
- `estimator.plsreg`: parameter estimation with PLS regression. For `inner` matrix only.

## Author(s)

Mikko Rönkkö, Wenjing Huang, Theo Dijkstra

Mikko Rönkkö, Gaston Sanchez, Laura Trinchera, Giorgio Russolillo

## References

Huang, W. (2013). PLSe: Efficient Estimators and Tests for Partial Least Squares (Doctoral dissertation). University of California, Los Angeles.

Dijkstra, T. K. (2011). Consistent Partial Least Squares estimators for linear and polynomial factor models. A report of a belated, serious and not even unsuccessful attempt. Comments are invited. Retrieved from http://www.rug.nl/staff/t.k.dijkstra/consistent-pls-estimators.pdf

Sanchez, G. (2013). *PLS Path Modeling with R*. Retrieved from http://www.gastonsanchez.com/PLS Path Modeling with R.pdf

## Examples

```
# Run the education example from the book

# Sanchez, G. (2013) PLS Path Modeling with R
# Trowchez Editions. Berkeley, 2013.
# http://www.gastonsanchez.com/PLS Path Modeling with R.pdf

education <- read.csv("http://www.gastonsanchez.com/education.csv")

Support <- c(0, 0, 0, 0, 0, 0)
Advising <- c(0, 0, 0, 0, 0, 0)
Tutoring <- c(0, 0, 0, 0, 0, 0)
Value <- c(1, 1, 1, 0, 0, 0)
# Omit two paths (compared to teh model in the book) to achieve
# identification of the 2SLS analysis
Satisfaction <- c(0, 0, 1, 1, 0, 0)
Loyalty <- c(0, 0, 0, 0, 1, 0)

inner <- rbind(Support, Advising, Tutoring, Value, Satisfaction, Loyalty)


reflective <- diag(6)[c(rep(1,4),
                        rep(2,4),
                        rep(3,4),
```

```
                        rep(4,4),
                        rep(5,3),
                        rep(6,4)),]
formative <- matrix(0, 6, 23)

colnames(inner) <- colnames(reflective) <- rownames(formative) <- rownames(inner)
rownames(reflective) <- colnames(formative) <- colnames(education)[2:24]

education.model <- list(inner = inner,
              reflective = reflective,
              formative = formative)

# Reverse code two variables
education[,c("sup.under","loy.asha")] <- - education[,c("sup.under","loy.asha")]

S <- cor(education[,2:24])

# PLSc with OLS regression

education.out <- matrixpls(S,education.model,
                      disattenuate = TRUE,
                      parametersReflective = estimator.plscLoadings)

# PLSc with 2SLS regresssion

education.out2 <- matrixpls(S,education.model,
                      disattenuate = TRUE,
                      parametersReflective = estimator.plscLoadings,
                      parametersInner = estimator.tsls)


# Disattenuated regression with unit-weighted scales and exploratory factor analysis
# reliability estimates (with unconstrained MINRES estimator)

education.out3 <- matrixpls(S,education.model,
                       disattenuate = TRUE,
                       weightFun = weightFun.fixed,
                       parametersReflective = estimator.efaLoadings)

# Disattenuated GSCA with 2SLS regression after disattenuated based on
# confirmatory factor analysis reliability estimates


education.out4 <- matrixpls(S,education.model,
                       disattenuate = TRUE,
                       innerEstim = innerEstim.gsca,
                       outerEstim = outerEstim.gsca,
                       parametersInner = estimator.tsls,
                       parametersReflective = estimator.cfaLoadings)


# Compare the results
```

```
cbind(PLSc = education.out, PLSc_2sls = education.out2,
      DR = education.out3, GSCAc = education.out4)

# Compare the reliability estimates

cbind(PLSc = attr(education.out,"Q"), PLSc_2sls = attr(education.out2,"Q"),
      DR = attr(education.out3,"Q"), GSCAc = attr(education.out4,"Q"))
```

---

`fitSummary` *Summary of model fit of PLS model*

---

## Description

`fitSummary` computes a list of statistics that are commonly used to access the overall fit of the PLS model.

## Usage

```
fitSummary(object, ...)
```

## Arguments

| | |
|---|---|
| `object` | matrixpls estimation result object produced by the `matrixpls` function. |
| `...` | All other arguments are ignored. |

## Value

A list containing a selection of fit indices.

## See Also

Other post-estimation functions: `ave()`, `cr()`, `effects.matrixpls()`, `fitted.matrixpls()`, `gof()`, `htmt()`, `loadings()`, `predict.matrixpls()`, `r2()`, `residuals.matrixpls()`

---

`fitted.matrixpls` *Model implied covariance matrix based on matrixpls results*

---

## Description

The `matrixpls` method for generic function `fitted` computes the model implied covariance matrix by combining `inner`, `reflective`, and `formative` as a simultaneous equations system. The error terms are constrained to be uncorrelated and covariances between exogenous variables are fixed at their sample values. Defining a composite as dependent variable in both inner and formative creates an impossible model and results in an error.

### Usage

```
## S3 method for class 'matrixpls'
fitted(object, ...)
```

### Arguments

| | |
|---|---|
| object | matrixpls estimation result object produced by the matrixpls function. |
| ... | All other arguments are ignored. |

### Value

a matrix containing the model implied covariances.

### See Also

Other post-estimation functions: ave(), cr(), effects.matrixpls(), fitSummary(), gof(), htmt(), loadings(), predict.matrixpls(), r2(), residuals.matrixpls()

---

## gof *Goodness of Fit indices for matrixpls results*

---

### Description

The matrixpls method for the generic function gof computes the Goodness of Fit index for matrixpls results.

### Usage

```
gof(object, ...)
```

### Arguments

| | |
|---|---|
| object | matrixpls estimation result object produced by the matrixpls function. |
| ... | All other arguments are ignored. |

### Value

The Goodness of Fit index.

### References

Henseler, J., & Sarstedt, M. (2013). Goodness-of-fit indices for partial least squares path modeling. *Computational Statistics*, 28(2), 565–580. doi:10.1007/s00180-012-0317-1

### See Also

Other post-estimation functions: ave(), cr(), effects.matrixpls(), fitSummary(), fitted.matrixpls(), htmt(), loadings(), predict.matrixpls(), r2(), residuals.matrixpls()

GSCA *Generalized structured component analysis (GSCA) weights*

## Description

When used with `weightFun.pls`, `innerEstim.gsca` and `outerEstim.gsca` implement the generalized structured component analysis indicator weighting system. Using `weightFun.optim` with the `optimCrit.gsca` optimization criterion provides an anternative approach to calculate GSCA weights by direct numerical minimization of the GSCA criterion function.

## Details

The two step GSCA weight algorithm is designed to minimize.

`SS(ZV-ZWA)`

The parameter matrix `A` contains all model parameters including `inner`, reflective `inner`, and `formative`. The weight matrices `V` and `W`, which can contain duplicate elements, contain the indicator weights.

The first step of GSCA estimation method is calculation of regressions coefficients `A` given weights `W` and `V`. The function `innerEstim.gsca` update the part of `A` corresponding to regressions between the composites, corresponding to `E` matrix in matrixpls. The regressions between composites and indicators are estimated in `outerEstim.gsca`.

This algorithm for estimating the relationships between the composites is therefore identical to the PLS path weighting scheme with the exception that correlations are not used for inverse relationships and there is no falling back to identity scheme for composites that are not connected to other composites The second step of GSCA is calculating a new set of weights conditional on the regression coeffcients `A` to minimize the sum of error terms in the regressions. This step is implemented in `outerEstim.gsca` after updating the regresions between indicators and composites. The implementation of GSCA in **matrixpls** differs from the Hwang & Takane (2004) version in that during the first step, only regressions between composites are estimated. The regressions between composites and indicators are estimated by the second stage the indicators and compositess. Since these covariances need to be calculated in the second step, it is more efficient to not calculate them during the first step.

A part of the code for `outerEstim.gsca` is adopted from the **ASGCA** package, licenced under GPL3.

## Author(s)

Mikko Rönkkö, Hela Romdhani, Stepan Grinek, Heungsun Hwang, Aurelie Labbe.

## References

Hwang, H., & Takane, Y. (2004). Generalized structured component analysis. Psychometrika, 69(1), 81–99. doi:10.1007/BF02295841

Hela Romdhani, Stepan Grinek, Heungsun Hwang and Aurelie Labbe. (2014). ASGSCA: Association Studies for multiple SNPs and multiple traits using Generalized Structured Equation Models. R package version 1.4.0.

## Examples

```
if(!require(ASGSCA)){
    print("This example requires the ASGSCA package from Bioconductor")
} else{
# Run the example from ASGSCA package using GSCA estimation

data(GenPhen)
W0 <- matrix(c(rep(1,2),rep(0,8),rep(1,2),rep(0,8),rep(1,3),rep(0,7),rep(1,2)),
             nrow=8,ncol=4)
B0 <- matrix(c(rep(0,8),rep(1,2),rep(0,3),1,rep(0,2)),nrow=4,ncol=4)

# Set seed becayse ASGSCA uses random numbers as starting values
set.seed(1)

GSCA.res <-GSCA(GenPhen,W0, B0,estim=TRUE,path.test=FALSE,
                 latent.names=c("Gene1","Gene2",
                                "Clinical pathway 1",
                                "Clinical pathway 2"))

# Setup matrixpls to estimate the same model. Note that ASGSCA places dependent
# variables on columns but matrixpls uses rows for dependent variables

inner <- t(B0)
formative <- t(W0)
reflective <- matrix(0,8,4)

colnames(formative) <- rownames(reflective) <- names(GenPhen)

colnames(inner) <- rownames(inner) <-
  rownames(formative) <- colnames(reflective) <-
  c("Gene1","Gene2","Clinical pathway 1","Clinical pathway 2")

model <- list(inner = inner,
              reflective = reflective,
              formative = formative)

# Estimate using alternating least squares

matrixpls.res1 <- matrixpls(cov(GenPhen),  model,
                            outerEstim = outerEstim.gsca,
                            innerEstim = innerEstim.gsca)

# Estimate using direct minimization of the estimation criterion
# Set the convergence criterion to be slightly stricter than normally
# to get indentical results

matrixpls.res2 <- matrixpls(cov(GenPhen),  model,
                            weightFun = weightFun.optim,
                            optimCrit = optimCrit.gsca,
                            control = list(reltol = 1e-12))

# Compare the weights
```

```
do.call(cbind,lapply(list(ASGSCA =GSCA.res[["Weight"]],
                          matrixpls_als = t(attr(matrixpls.res1,"W")),
                          matrixpls_optim =t(attr(matrixpls.res2,"W"))),
                     function(W) W[W!=0]))


# Check the criterion function values

optimCrit.gsca(matrixpls.res1)
optimCrit.gsca(matrixpls.res2)

}
```

---

`htmt` *Heterotrait-monotrait ratio*

---

## Description

The `matrixpls` method for the generic function `htmt` computes Heterotrait-monotrait ratio for the model using the formula presented by Henseler et al (2014).

## Usage

```
htmt(object, ...)
```

## Arguments

| | |
|---|---|
| `object` | matrixpls estimation result object produced by the `matrixpls` function. |
| `...` | All other arguments are ignored. |

## Value

Heterotrait-monotrait ratio as a scalar

## References

Henseler, J., Ringle, C. M., & 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.

## See Also

Other post-estimation functions: `ave()`, `cr()`, `effects.matrixpls()`, `fitSummary()`, `fitted.matrixpls()`, `gof()`, `loadings()`, `predict.matrixpls()`, `r2()`, `residuals.matrixpls()`

`innerEstim` *PLS inner estimation*

## Description

Calculates a set of inner weights.

## Usage

```
innerEstim.centroid(S, W, inner.mod, ignoreInnerModel = FALSE, ...)

innerEstim.path(S, W, inner.mod, ...)

innerEstim.factor(S, W, inner.mod, ignoreInnerModel = FALSE, ...)

innerEstim.identity(S, W, inner.mod, ...)

innerEstim.gsca(S, W, inner.mod, ...)
```

## Arguments

| | |
|---|---|
| `S` | Covariance matrix of the data. |
| `W` | Weight matrix, where the indicators are on colums and composites are on the rows. |
| `inner.mod` | A square matrix specifying the relationships of the composites in the model. |
| `ignoreInnerModel` | Should the inner model be ignored and all correlations be used. |
| `...` | Other arguments are ignored. |

## Details

In the centroid scheme, inner weights are set to the signs (1 or -1) of correlations between composites that are connected in the model specified in `inner.mod` and zero otherwise.

In the path scheme, inner weights are based on regression estimates of the relationships between composites that are connected in the model specified in `inner.mod`, and correlations for the inverse relationships. If a relationship is reciprocal, regression is used for both directions.

In the factor scheme, inner weights are set to the correlations between composites that are connected in the model specified in `inner.mod` and zero otherwise.

In the identity scheme identity matrix is used as the inner weight matrix `E`.

Centroid, innner, and path schemes fall back to to identity scheme for composites that are not connected to any other composites.

For information about GSCA weights, see GSCA.

## Value

A matrix of unscaled inner weights `E` with the same dimesions as `inner.mod`.

### Functions

- `innerEstim.centroid`: inner estimation with centroid scheme.
- `innerEstim.path`: inner estimation with path scheme.
- `innerEstim.factor`: inner estimation with factor scheme.
- `innerEstim.identity`: inner estimation with identity scheme.
- `innerEstim.gsca`: inner estimation with generalized structured component analysis.

### References

Lohmöller J.-B. (1989) *Latent variable path modeling with partial least squares.* Heidelberg: Physica-Verlag.

---

## `loadings` *Factor loadings matrix from matrixpls results*

---

### Description

The `matrixpls` method for the generic function `loadings` computes standardized factor loading matrix from the model results.

### Usage

```
loadings(object, ...)
```

### Arguments

| | |
|---|---|
| `object` | matrixpls estimation result object produced by the `matrixpls` function. |
| `...` | All other arguments are ignored. |

### Value

A matrix of estimated factor loadings.

### See Also

Other post-estimation functions: `ave()`, `cr()`, `effects.matrixpls()`, `fitSummary()`, `fitted.matrixpls()`, `gof()`, `htmt()`, `predict.matrixpls()`, `r2()`, `residuals.matrixpls()`

matrixpls *Partial Least Squares and other composite variable models.*

## Description

Estimates a weight matrix using Partial Least Squares or a related algorithm and then uses the weights to estimate the parameters of a statistical model.

## Usage

```
matrixpls(
  S,
  model,
  W.model = NULL,
  weightFun = weightFun.pls,
  parameterEstim = parameterEstim.separate,
  weightSign = NULL,
  ...,
  validateInput = TRUE,
  standardize = TRUE
)
```

## Arguments

| | |
|---|---|
| S | Covariance matrix of the data. |
| model | There are two options for this argument: 1. lavaan script or lavaan parameter table, or 2. a list containing three matrices inner, reflective, and formative defining the free regression paths in the model. |
| W.model | An optional numeric matrix representing the weight pattern and starting weights (i.e. the how the indicators are combined to form the composite variables). If this argument is not specified, the weight patter is defined based on the relationships in the reflective and formative elements of model. |
| weightFun | A function for calculating indicator weights using the data covariance matrix S, a model specification model, and a weight pattern W.model. Returns a weigth matrix W. The default is weightFun.pls |
| parameterEstim | A function for estimating the model parameters using the data covariance matrix S, model specification model, and weight matrix W. Returns a named vector of parameter estimates. The default is parameterEstim.separate |
| weightSign | A function for resolving weight sign ambiquity based on the data covariance matrix S and a weight matrix W. Returns a weigth matrix W. See weightSign for details. |
| ... | All other arguments are passed through to weightFun and parameterEstim. |
| validateInput | If TRUE, the arguments are validated. |
| standardize | If TRUE, S is converted to a correlation matrix before analysis. |

## Details

matrixpls is the main function of the matrixpls package. This function parses a model object and then uses the results to call weightFun to to calculate indicator weight. After this the parameterEstim function is applied to the indicator weights, the data covariance matrix, and the model object and the resulting parameter estimates are returned.

Model can be specified in the lavaan format or the native matrixpls format. The native model format is a list of three binary matrices, inner, reflective, and formative specifying the free parameters of a model: inner (l x l) specifies the regressions between composites, reflective (k x l) specifies the regressions of observed data on composites, and formative (l x k) specifies the regressions of composites on the observed data. Here k is the number of observed variables and l is the number of composites.

If the model is specified in lavaan format, the native format model is derived from this model by assigning all regressions between latent variables to inner, all factor loadings to reflective, and all regressions of latent variables on observed variables to formative. Regressions between observed variables and all free covariances are ignored. All parameters that are specified in the model will be treated as free parameters.

The original papers about Partial Least Squares, as well as many of the current PLS implementations, impose restrictions on the matrices inner, reflective, and formative: inner must be a lower triangular matrix, reflective must have exactly one non-zero value on each row and must have at least one non-zero value on each column, and formative must only contain zeros. Some PLS implementations allow formative to contain non-zero values, but impose a restriction that the sum of reflective and t(formative) must satisfy the original restrictions of reflective. The only restrictions that matrixpls imposes on inner, reflective, and formative is that these must be binary matrices and that the diagonal of inner must be zeros.

The argument W.model is a (l x k) matrix that indicates how the indicators are combined to form the composites. The original papers about Partial Least Squares as well as all current PLS implementations define this as t(reflective) | formative, which means that the weight patter must match the model specified in reflective and formative. Matrixpls does not require that W.model needs to match reflective and formative, but accepts any numeric matrix. If this argument is not specified, all elements of W.model that correspond to non-zero elements in the reflective or formative formative matrices receive the value 1.

## Value

A named numeric vector of class matrixpls containing the parameter estimates followed by weights.

matrixpls returns the following as attributes:

| | |
|---|---|
| W | the weight matrix. |
| model | the model specification in native format. |
| call | the function call. |

Additionally, all attributes returned by functions called by matrixpls are returned. This can include:

| | |
|---|---|
| S | the sample covariance matrix. |
| E | inner weight matrix. |
| iterations | the number of iterations used by the weight algorithm. |

| | |
|---|---|
| converged | TRUE if the weight algorithm converged and FALSE otherwise. |
| history | weight optimization history as a matrix. |
| C | the composite correlation matrix (after disattenuation, if requested). |
| IC | the indicator-composite covariance matrix (after disattenuation, if requested). |
| inner | the inner model matrix with estimated parameters. |
| reflective | the reflective model matrix with estimated parameters. |
| formative | the formative model matrix with estimated parameters. |
| Q | the reliability estimates used in dissattenuation. |
| c | the PLSc loading estimate correction factors. |

## References

Rosseel, Y. (2012). lavaan: An R Package for Structural Equation Modeling. *Journal of Statistical Software*, 48(2), 1–36. Retrieved from http://www.jstatsoft.org/v48/i02

Wold, H. (1982). Soft modeling - The Basic Design And Some Extensions. In K. G. Jöreskog & S. Wold (Eds.),*Systems under indirect observation: causality, structure, prediction* (pp. 1–54). Amsterdam: North-Holland.

## See Also

Weight algorithms: weightFun.pls; weightFun.fixed; weightFun.optim; weightFun.principal; weightFun.factor

Weight sign corrections:weightSign.Wold1985; weightSign.dominantIndicator

Parameter estimators: parameterEstim.separate

---

# matrixpls-common *Commonly used arguments*

---

## Description

The following describes commonly used arguments in the **matrixpls** package.

## Arguments

| | |
|---|---|
| S | Covariance matrix of the data. |
| model | There are two options for this argument: 1. lavaan script or lavaan parameter table, or 2. a list containing three matrices inner, reflective, and formative defining the free regression paths in the model. |
| W | Weight matrix, where the indicators are on colums and composites are on the rows. |
| C | Correlation matrix of the composites. |
| IC | Correlation matrix of the composites and indicators. |
| E | Inner weight matrix. A square matrix of inner estimates between the composites. |
| W.model | A matrix specifying the weight relationships and their starting values. |

matrixpls-functions *All estimation function types*

## Description

The following describes all estimation function types used in **matrixpls** package.

## Arguments

| | |
|---|---|
| weightFun | A function for calculating indicator weights using the data covariance matrix S, a model specification model, and a weight pattern W.model. Returns a weigth matrix W. The default is weightFun.pls |
| parameterEstim | A function for estimating the model parameters using the data covariance matrix S, model specification model, and weight matrix W. Returns a named vector of parameter estimates. The default is parameterEstim.separate |
| estimator | A function for estimating the parameters of one model matrix using the data covariance matrix S, a model matrix modelMatrix, and a weight matrix W. Disattenuated composite correlation matrix C and indicator composite covariance matrix IC are optional. Returns matrix of parameter estimates. The default is estimator.ols |
| weightSign | A function for resolving weight sign ambiquity based on the data covariance matrix S and a weight matrix W. Returns a weigth matrix W. See weightSign for details. |
| outerEstim | A function for calculating outer weights using the data covariance matrix S, a weight matrix W, an inner weight matrix E, and a weight pattern W.model. Returns a weigth matrix W. See outerEstim. |
| innerEstim | A function for calculating inner weights using the data covariance matrix S, a weight matrix W, and an inner model matrix inner.mod. Returns an inner weigth matrix E. The default is innerEstim.path. |
| convCheck | A function that takes the old Wold and new weight Wold matrices and returns a scalar that is compared against tol to check for convergence. The default is convCheck.absolute. |
| optimCrit | A function for calculating value for an optimization criterion based on a matrixpls result object. Returns a scalar. The default is optimCrit.maximizeInnerR2. |
| reliabilities | A function for calculating reliability estimates based on the data covariance matrix S, factor loading matrix loadings, and a weight matrix W. Returns a vector of reliability estimates. The default is reliabilityEstim.weightLoadingProduct |

`matrixpls.crossvalidate` *Cross-validation of predictions from matrixpls results*

## Description

`matrixpls.crossvalidate` Calculates cross-validation predictions using `matrixpls`.

## Usage

```
matrixpls.crossvalidate(
  data,
  model,
  ...,
  predictFun = stats::predict,
  nGroup = 4,
  blindfold = FALSE,
  imputationFun = NULL
)
```

## Arguments

| | |
|---|---|
| `data` | Matrix or data frame containing the raw data. |
| `model` | There are two options for this argument: 1. lavaan script or lavaan parameter table, or 2. a list containing three matrices `inner`, `reflective`, and `formative` defining the free regression paths in the model. |
| `...` | All other arguments are passed through to `matrixpls` and `predictFun`. |
| `predictFun` | The function used to calculate the predictions. |
| `nGroup` | The number of groups to divide the data into. |
| `blindfold` | Whether blindfolding should be used instead of holdout sample cross-validation. |
| `imputationFun` | The function used to impute missing data before blindfold prediction. If `NULL`, simple mean substitution is used. |

## Details

In cross-validation, some elements of the data matrix are set to missing and then predicted based on the remaining observations. The process is repeated until all elements of the data have been predicted.

Cross-validation is typically applied by dividing the data into two groups, the training sample and the validation sample. The prediction model is calculated based on the training sample and used to calculate predictions for the validation sample.

In blindfolding, the data are not omitted case wise, but elements of the data are omitted diagonally. After this, imputation is applied to missing data and the prediction model is calibrated with the dataset containing also the imputations. The imputed values are then predicted with the model.

## Value

A matrix containing predictions calculated with cross-validation.

## References

Chin, W. W. (2010). How to write up and report PLS analyses. In V. Esposito Vinzi, W. W. Chin, J. Henseler, & H. Wang (Eds.), Handbook of partial least squares (pp. 655–690). Berlin Heidelberg: Springer.

Chin, W. W. (1998). The partial least squares approach to structural equation modeling. In G. A. Marcoulides (Ed.), Modern methods for business research (pp. 295–336). Mahwah, NJ: Lawrence Erlbaum Associates Publishers.

---

`matrixpls.plspm` *A plspm compatibility wrapper for matrixpls*

---

## Description

`matrixpls.plspm` mimics `plspm` function of the `plspm` package. The arguments and their default values and the output of the function are identical with `plspm` function, but internally the function uses matrixpls estimation.

## Usage

```
matrixpls.plspm(
  Data,
  path_matrix,
  blocks,
  modes = NULL,
  scheme = "centroid",
  scaled = TRUE,
  tol = 1e-06,
  maxiter = 100,
  boot.val = FALSE,
  br = NULL,
  dataset = TRUE
)
```

## Arguments

| | |
|---|---|
| `Data` | matrix or data frame containing the manifest variables. |
| `path_matrix` | A square (lower triangular) boolean matrix representing the inner model (i.e. the path relationships between latent variables). |
| `blocks` | list of vectors with column indices or column names from `Data` indicating the sets of manifest variables forming each block (i.e. which manifest variables correspond to each block). |

| | |
|---|---|
| modes | character vector indicating the type of measurement for each block. Possible values are: "A","B","newA","PLScore","PLScow". The length of modes must be equal to the length of blocks. |
| scheme | string indicating the type of inner weighting scheme. Possible values are "centroid", "factorial", or "path". |
| scaled | whether manifest variables should be standardized. Only used when scaling = NULL. When (TRUE, data is scaled to standardized values (mean=0 and variance=1). The variance is calculated dividing by N instead of N-1). |
| tol | decimal value indicating the tolerance criterion for the iterations (tol=0.000001). Can be specified between 0 and 0.001. |
| maxiter | integer indicating the maximum number of iterations (maxiter=100 by default). The minimum value of maxiter is 100. |
| boot.val | whether bootstrap validation should be performed. (FALSE by default). |
| br | number bootstrap resamples. Used only when boot.val=TRUE. When boot.val=TRUE, the default number of re-samples is 100. |
| dataset | whether the data matrix used in the computations should be retrieved (TRUE by default). |

## Details

The function matrixpls.plspm calculates indicator weights and estimates a model identically to the plspm function. In contrast to the matrixpls function that provides only weights and parameter estimates, this function also reports multiple post-estimation statistics. Because of this matrixpls.plspm is substantially less efficient than the matrixpls function.

The argument path_matrix is a matrix of zeros and ones that indicates the structural relationships between composites. This must be a lower triangular matrix. path_matrix will contain a 1 when column j affects row i, 0 otherwise.

## Value

An object of class plspm.

## References

Sanchez, G. (2013). *PLS Path Modeling with R*. Retrieved from http://www.gastonsanchez.com/PLS Path Modeling with R.pdf

## See Also

plspm

## Examples

```
if(!require(plspm)){
    print("This example requires the plspm package")
} else{
```

```
library(plspm)

cores <- getOption("mc.cores")
options(mc.cores=2)

# Run the example from plspm package

# load dataset satisfaction
data(satisfaction)
# inner model matrix
IMAG = c(0,0,0,0,0,0)
EXPE = c(1,0,0,0,0,0)
QUAL = c(0,1,0,0,0,0)
VAL = c(0,1,1,0,0,0)
SAT = c(1,1,1,1,0,0)
LOY = c(1,0,0,0,1,0)
sat_inner = rbind(IMAG, EXPE, QUAL, VAL, SAT, LOY)
# outer model list
sat_outer = list(1:5, 6:10, 11:15, 16:19, 20:23, 24:27)
# vector of modes (reflective indicators)
sat_mod = rep("A", 6)

# apply plspm
plspm.res <- plspm(satisfaction, sat_inner, sat_outer, sat_mod,
                   scaled=FALSE, boot.val=FALSE)

# apply matrixpls
matrixpls.res <- matrixpls.plspm(satisfaction, sat_inner, sat_outer, sat_mod,
                                 scaled=FALSE, boot.val=FALSE)

# If RUnit is installed check that the results are identical

if(is.element("RUnit", installed.packages()[,1])){
  library(RUnit)
  checkEquals(plspm.res, matrixpls.res, tol = 0.001)
}

# print the resuls

print(summary(plspm.res))
print(summary(matrixpls.res))

options(mc.cores=cores)

}
```

---

`matrixpls.sempls` *A semPLS compatibility wrapper for matrixpls*

---

## Description

matrixpls.sempls mimics sempls function of the semPLS package. The arguments and their default values and the output of the function are identical with sempls function, but internally the function uses matrixpls estimation.

## Usage

```
matrixpls.sempls(
  model,
  data,
  maxit = 20,
  tol = 1e-07,
  scaled = TRUE,
  sum1 = FALSE,
  wscheme = "centroid",
  pairwise = FALSE,
  method = c("pearson", "kendall", "spearman"),
  convCrit = c("relative", "square"),
  verbose = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| model | An object inheriting from class plsm as returned from plsm or read.splsm. |
| data | A data.frame containing the observed variables (MVs). The storage mode for all the MVs included in the model must be numeric. |
| maxit | A numeric value, which determines the maximum number of iterations performed by the PLS algorithm. The default is 20 iterations. |
| tol | A numeric value, specifying the tolerance for the maximum relative differences in the outer weights. The default value is $10^{-7}$. |
| scaled | A logical value indicating, whether the observed data shall be scaled to zero mean and unit variance. The default is TRUE. |
| sum1 | A logical value indicating, whether the outer weights foreach latent variable (LV) shall be standardized to sum up to one. The default is FALSE. Since the factor scores are scaled in each step of the PLS algorithm, changing this value to TRUE does not affect the results. |
| wscheme | A character naming the weighting scheme to use. Possible values are:<br>• "A" or "centroid" for the centroid scheme, the default,<br>• "B" or "factorial"for the factorial scheme and<br>• "C", "pw" or "pathWeighting" for the path weighting scheme. |
| pairwise | A logical value indicating, whether correlations shall be calculated pairwise. If the observed data does not contain missing values, the results are not affected. The default is FALSE. For more details the R help, ?cor, can be consulted. |
| method | A character naming the method to calculate the correlations. Possible values are: |

- "pearson" , the default,
- "kendall",
- "spearman".

For more details on the method, the R help, ?cor, can be consulted. Note, that despite of the method argument, pearson correlations are always used for the inner approximation (step 2).

convCrit — The convergence criteria to use:

- "relative", the default,
- "square".

verbose — Logical: If FALSE no status messages are printed.

... — Other arguments are ignored

## Value

An object of class sempls.

## References

Monecke, A., & Leisch, F. (2012). semPLS: Structural Equation Modeling Using Partial Least Squares. *Journal of Statistical Software*, 48(3), 1–32.

## See Also

sempls

## Examples

```
if(!require(semPLS)){
    print("This example requires the semPLS package")
} else{
data(ECSImobi)

ecsi.sempls <- sempls(model=ECSImobi, data=mobi, wscheme="pathWeighting")
ecsi <- matrixpls.sempls(model=ECSImobi, data=mobi, wscheme="pathWeighting")

# If RUnit is installed check that the results are identical

if(require(RUnit)){
  checkEquals(ecsi.sempls,ecsi, check.attributes = FALSE)
}

ecsi

## create plots
densityplot(ecsi)
densityplot(ecsi, use="prediction")
densityplot(ecsi, use="residuals")

## Values of 'sempls' objects
```

```
names(ecsi)
ecsi$outer_weights
ecsi$outer_loadings
ecsi$path_coefficients
ecsi$total_effects


### using convenience methods to sempls results
## path coefficients
pathCoeff(ecsi)

## total effects
totalEffects(ecsi)

## get loadings and check for discriminant validity
(l <- plsLoadings(ecsi))
# outer loadings
print(l, type="outer", digits=2)
# outer loadings greater than 0.5
print(l,type="outer", cutoff=0.5, digits=2)
# cross loadings greater than 0.5
print(l, type="cross", cutoff=0.5, digits=2)


### R-squared
rSquared(ecsi)

}
```

---

`matrixpls.sim` *Monte Carlo simulations with matrixpls*

---

## Description

Performs Monte Carlo simulations of `matrixpls` with the `sim` function of the `simsem` package. The standard errors and confidence intervals are estimated with the `boot` and `boot.ci` functions of the `boot` package.

## Usage

```
matrixpls.sim(
  nRep = NULL,
  model = NULL,
  n = NULL,
  ...,
  cilevel = 0.95,
  citype = c("norm", "basic", "stud", "perc", "bca"),
  boot.R = 500,
  fitIndices = fitSummary,
```

```
  outfundata = NULL,
  outfun = NULL,
  prefun = NULL
)
```

## Arguments

| | |
|---|---|
| nRep | Number of replications. If any of the n, pmMCAR, or pmMAR arguments are specified as lists, the number of replications will default to the length of the list(s), and nRep need not be specified. |
| model | There are two options for this argument: 1. lavaan script or lavaan parameter table, or 2. a list containing three matrices inner, reflective, and formative defining the free regression paths in the model. |
| n | Sample size. Either a single value, or a list of values to vary sample size across replications. The n argument can also be specified as a random distribution object; if any resulting values are non-integers, the decimal will be rounded. |
| ... | All other arguments are passed through to sim, matrixpls.boot, or matrixpls. |
| cilevel | Confidence level. This argument will be forwarded to the boot.ci when calculating the confidence intervals. |
| citype | Type of confidence interval. This argument will be forwarded to the boot.ci when calculating the confidence intervals. |
| boot.R | Number of bootstrap replications to use to estimate standard errors or FALSE to disable bootstrapping. |
| fitIndices | A function that returns a list of fit indices for the model. Setting this argument to NULL disables fit indices. |
| outfundata | A function to be applied to the matrixpls output and the generated data after each replication. Users can get the characteristics of the generated data and also compare the characteristics with the generated output. The output from this function in each replication will be saved in the simulation output (SimResult), and can be obtained using the getExtraOutput function. |
| outfun | A function to be applied to the matrixpls output at each replication. Output from this function in each replication will be saved in the simulation output (SimResult), and can be obtained using the getExtraOutput function. |
| prefun | A function to be applied to the dataset before each replication. The output of this function is passed as arguments to matrixpls |

## Details

This funtion calls the sim function from the simsem package to perform Monte Carlo simulations with matrixpls. The function parses the model parameters and replaces it with a function call that estimates the model and bootstrapped standard errors and confidence intervals with matrixpls.boot.

If the generate or rawdata arguments are not specified in the sim arguments then the model argument will be used for data generation and must be specified in lavaan format.

## Value

An object of class SimResult-class.

## See Also

matrixpls, matrixpls.boot, sim, SimResult-class

## Examples

```
if(!require(simsem)){
    print("This example requires the simsem package")
} else{

#
# Runs the second model from
#
# Aguirre-Urreta, M., & Marakas, G. (2013). Partial Least Squares and Models with Formatively
# Specified Endogenous Constructs: A Cautionary Note. Information Systems Research.
# doi:10.1287/isre.2013.0493

library(MASS)

X <- diag(15)
X[upper.tri(X, diag=TRUE)] <- c(
  1.000,
  0.640, 1.000,
  0.640, 0.640, 1.000,
  0.640, 0.640, 0.640, 1.000,
  0.096, 0.096, 0.096, 0.096, 1.000,
  0.096, 0.096, 0.096, 0.096, 0.640, 1.000,
  0.096, 0.096, 0.096, 0.096, 0.640, 0.640, 1.000,
  0.096, 0.096, 0.096, 0.096, 0.640, 0.640, 0.640, 1.000,
  0.115, 0.115, 0.115, 0.115, 0.192, 0.192, 0.192, 0.192, 1.000,
  0.115, 0.115, 0.115, 0.115, 0.192, 0.192, 0.192, 0.192, 0.640, 1.000,
  0.115, 0.115, 0.115, 0.115, 0.192, 0.192, 0.192, 0.192, 0.640, 0.640,
  1.000,
  0.115, 0.115, 0.115, 0.115, 0.192, 0.192, 0.192, 0.192, 0.640, 0.640,
  0.640, 1.000,
  0.000, 0.000, 0.000, 0.000, 0.271, 0.271, 0.271, 0.271, 0.325, 0.325,
  0.325, 0.325, 1.000,
  0.000, 0.000, 0.000, 0.000, 0.271, 0.271, 0.271, 0.271, 0.325, 0.325,
  0.325, 0.325, 0.300, 1.000,
  0.000, 0.000, 0.000, 0.000, 0.271, 0.271, 0.271, 0.271, 0.325, 0.325,
  0.325, 0.325, 0.300, 0.300, 1.000
)
X <- X + t(X) - diag(diag(X))

colnames(X) <- rownames(X) <- c(paste("Y",1:12,sep=""),paste("X",1:3,sep=""))

# Print the population covariance matrix X to see that it is correct

X

# The estimated model in Lavaan syntax

analyzeModel <- "
```

```
ksi =~ Y1 + Y2 + Y3 + Y4
eta1 <~ X1 + X2 + X3
eta2 =~ Y5 + Y6 + Y7 + Y8
eta3 =~ Y9 + Y10 + Y11 + Y12

eta1 ~ ksi
eta2 ~ eta1
eta3 ~ eta1
"

# Only run 100 replications without bootstrap replications each so that the
# example runs faster. Generate the data outside simsem because simsem
# does not support drawing samples from population matrix

dataSets <- lapply(1:100, function(x){
  mvrnorm(300,                 # Sample size
          rep(0,15),           # Means
          X)                   # Population covarancematrix
})

Output <- matrixpls.sim(model = analyzeModel, rawData = dataSets, boot.R=FALSE,
                        multicore = FALSE, stopOnError = TRUE)

summary(Output)


}
```

---

`matrixplsboot` *Bootstrapping of matrixpls function*

---

## Description

`matrixpls.boot` is a convenience method that implements bootstrapping of `matrixpls` with `boot` method of the `boot` package.

## Usage

```
matrixpls.boot(
  data,
  model,
  ...,
  R = 500,
  signChange = NULL,
  parallel = c("no", "multicore", "snow"),
  ncpus = getOption("boot.ncpus", 1L),
  dropInadmissible = FALSE,
  stopOnError = FALSE,
  extraFun = NULL
```

```
)

## S3 method for class 'matrixplsboot'
summary(object, ..., ci.type = "all")
```

## Arguments

| | |
|---|---|
| data | Matrix or data frame containing the raw data. |
| model | There are two options for this argument: 1. lavaan script or lavaan parameter table, or 2. a list containing three matrices inner, reflective, and formative defining the free regression paths in the model. |
| ... | All other arguments are passed through to matrixpls and boot. |
| R | Number of bootstrap samples to draw. |
| signChange | Sign change correction function. |
| parallel | The type of parallel operation to be used (if any). If missing, the default is taken from the option "boot.parallel" (and if that is not set, "no"). |
| ncpus | integer: number of processes to be used in parallel operation: typically one would chose this to the number of available CPUs. |
| dropInadmissible | A logical indicating whether non-convergent and inadmissible replications should be discarded. |
| stopOnError | A logical indicating whether boostrapping should be continued when error occurs in a replication. |
| extraFun | A function that takes a matrixpls object and a boostrap sample and returns a numeric vector. The vector is appended to bootstrap replication. Can be used for boostrapping additional statistics calculated based on the estimation results. |
| object | object of class matrixplsboot |
| ci.type | A vector of character strings representing the type of intervals required. Passed on to boot.ci. If "none", no confidence intervals are calculated. |

## Value

An object of class matrixplsboot and boot.

## Analyzing the boostrap results

The output can be analyzed with any of the functions provided by the boot package. For example, the boot.ci function can be used for calculating confidence intervals and the empinf function can be used to calculate influence values that may be useful for identifying outliers.

The class matrixplsboot provides only a summary function, which calculates a set of statistics that are commonly of interest after boostrapping. This includes standard errors, t statistics (estimate / SE) and p-values based on Student's t distribution and standard normal distribution. Because the sampling distribution of the parameter estimates calculated by matrixpls are not always known, the p-values cannot be expected to be unbiased.

This concern applies particularly when using PLS weights. Because the PLS literature provides conflicting advice on which probability distribution to use as a reference, the summary method

of matrixplsboot produces two-tailed p-values based on four different probalility distributions. The *regression* p values are based on comparing the t statistic against the references distribution used in regression analysis, namely Student's t distribution with n -k -1 degrees of freedom. The *Hair* p values are based on Hair et al's (2014, p. 134) recommendation to ignore the number of independent variables k and set the degrees of freedom to n -1. The *Henseler* p values are based on the recommendation by Henseler et al (2009, p. 305) that the degrees of freedom should be set as n + m -2, where mis always 1 and n is the number of bootstrap samples. The z p values are based on comparing the t statistic against the standard normal distibution. This choice can be motivated by asymptotic normality of the PLS estimates in certain conditions.

## References

Hair, J. F., Hult, G. T. M., Ringle, C. M., & Sarstedt, M. (2014). *A primer on partial least squares structural equations modeling (PLS-SEM)*. Los Angeles: SAGE.

Henseler, J., Ringle, C. M., & Sinkovics, R. R. (2009). The use of partial least squares path modeling in international marketing. *Advances in International Marketing*, 20, 277–319.

Rönkkö, M., & Evermann, J. (2013). A critical examination of common beliefs about partial least squares path modeling. *Organizational Research Methods*, 16(3), 425–448. http://doi.org/10.1177/1094428112474693

Rönkkö, M., McIntosh, C. N., & Antonakis, J. (2015). On the adoption of partial least squares in psychological research: Caveat emptor. *Personality and Individual Differences*, (87), 76–84. http://doi.org/10.1016/j.paid.2015.07.019

## See Also

boot

Sign change corrections: signChange.individual; signChange.construct

## Examples

```
library(plspm)

# Run the customer satisfaction example form plspm

# load dataset satisfaction
data(satisfaction)
# inner model matrix
IMAG = c(0,0,0,0,0,0)
EXPE = c(1,0,0,0,0,0)
QUAL = c(0,1,0,0,0,0)
VAL = c(0,1,1,0,0,0)
SAT = c(1,1,1,1,0,0)
LOY = c(1,0,0,0,1,0)
inner = rbind(IMAG, EXPE, QUAL, VAL, SAT, LOY)
colnames(inner) <- rownames(inner)

# Reflective model

reflective<- matrix(
  c(1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1),
  27,6, dimnames = list(colnames(satisfaction)[1:27],colnames(inner)))

# empty formative model

formative <- matrix(0, 6, 27, dimnames = list(colnames(inner),
                                              colnames(satisfaction)[1:27]))

# Only 100 replications to make the example faster when running on single processor.
# Increase the number of replications (e.g. to 500) to get the BCa intervals

matrixpls.boot.out <- matrixpls.boot(satisfaction[,1:27],
                           model = list(inner = inner,
                                        reflective = reflective,
                                        formative = formative),
                           R = 100)

print(matrixpls.boot.out)
summary(matrixpls.boot.out)
```

---

`optimCrit` *Optimization criteria functions*

---

## Description

Optimization criterion functions calculate various optimization criterion values from `matrixpls` objects.

## Usage

```
optimCrit.maximizeInnerR2(matrixpls.res)

optimCrit.maximizeIndicatorR2(matrixpls.res)

optimCrit.maximizeFullR2(matrixpls.res)

optimCrit.gsca(matrixpls.res)
```

## Arguments

| | |
|---|---|
| `matrixpls.res` | An object of class `matrixpls` from which the criterion function is calculated |

## Value

Value of the optimization criterion.

### Functions

- `optimCrit.maximizeInnerR2`: maximizes the sum of R2 statistics of the `inner` matrix
- `optimCrit.maximizeIndicatorR2`: maximizes the sum of R2 statistics of the `reflective` matrix.
- `optimCrit.maximizeFullR2`: maximizes the sum of R2 statistics of the `inner` and `reflective` matrices.
- `optimCrit.gsca`: minimies the generalized structured component analysis criterion. See GSCA

### See Also

`weightFun.optim`

---

## `outerEstim` *PLS outer estimation*

---

### Description

Calculates a set of unstandardized outer weights.

Mode A outer weights are correlations between indicators and composites. Mode B outer weights are regression coefficients of composites on indicators.

For information about GSCA weights, see GSCA.

### Usage

```
outerEstim.modeA(S, W, E, W.model, ...)

outerEstim.modeB(S, W, E, W.model, ...)

outerEstim.gsca(S, W, E, W.model, model, ...)

outerEstim.fixed(S, W, E, W.model, ...)
```

### Arguments

| | |
|---|---|
| `S` | Covariance matrix of the data. |
| `W` | Weight matrix, where the indicators are on colums and composites are on the rows. |
| `E` | Inner weight matrix. A square matrix of inner estimates between the composites. |
| `W.model` | A matrix specifying the weight relationships and their starting values. |
| `...` | All other arguments are ignored. |
| `model` | There are two options for this argument: 1. lavaan script or lavaan parameter table, or 2. a list containing three matrices `inner`, `reflective`, and `formative` defining the free regression paths in the model. |

## Value

A matrix of unscaled outer weights `W` with the same dimesions as `W.model`.

## Functions

- `outerEstim.modeA`: Mode A outer estimation.
- `outerEstim.modeB`: Mode B outerestimation.
- `outerEstim.gsca`: outer estimation with generalized structured component analysis.
- `outerEstim.fixed`: Fixed weights. Returns the starting weights specified in `W.model`

## References

Lohmöller J.-B. (1989) *Latent variable path modeling with partial least squares.* Heidelberg: Physica-Verlag.

---

# `parameterEstim` *Parameter estimation of full model*

---

## Description

`paramsEstimator` functions estimates the statistical model described by `model`

## Usage

```
parameterEstim.separate(
  S,
  model,
  W,
  ...,
  parametersInner = estimator.ols,
  parametersReflective = estimator.ols,
  parametersFormative = estimator.ols,
  disattenuate = FALSE,
  reliabilities = reliabilityEstim.weightLoadingProduct
)
```

## Arguments

| | |
|---|---|
| `S` | Covariance matrix of the data. |
| `model` | There are two options for this argument: 1. lavaan script or lavaan parameter table, or 2. a list containing three matrices `inner`, `reflective`, and `formative` defining the free regression paths in the model. |
| `W` | Weight matrix, where the indicators are on colums and composites are on the rows. |

`...` All other arguments are passed through to `parametersInner`, `parametersReflective`, and`parametersFormative`

`parametersInner`
A function used to estimate the `inner` model matrix. The default is `estimator.ols`

`parametersReflective`
A function used to estimate the `reflective` model matrix. The default is `estimator.ols`

`parametersFormative`
A function used to estimate the `formative` model matrix. The default is `estimator.ols`

`disattenuate` If `TRUE`, `C` is disattenuated before applying `parametersInner`.

`reliabilities` A function for calculating reliability estimates based on the data covariance matrix `S`, factor loading matrix `loadings`, and a weight matrix `W`. Returns a vector of reliability estimates. The default is `reliabilityEstim.weightLoadingProduct`

## Details

Model can be specified in the lavaan format or the native matrixpls format. The native model format is a list of three binary matrices, `inner`, `reflective`, and `formative` specifying the free parameters of a model: `inner` (`l` x `l`) specifies the regressions between composites, `reflective` (`k` x `l`) specifies the regressions of observed data on composites, and `formative` (`l` x `k`) specifies the regressions of composites on the observed data. Here `k` is the number of observed variables and `l` is the number of composites.

If the model is specified in lavaan format, the native format model is derived from this model by assigning all regressions between latent variables to `inner`, all factor loadings to `reflective`, and all regressions of latent variables on observed variables to `formative`. Regressions between observed variables and all free covariances are ignored. All parameters that are specified in the model will be treated as free parameters.

The original papers about Partial Least Squares, as well as many of the current PLS implementations, impose restrictions on the matrices `inner`, `reflective`, and `formative`: `inner` must be a lower triangular matrix, `reflective` must have exactly one non-zero value on each row and must have at least one non-zero value on each column, and `formative` must only contain zeros. Some PLS implementations allow `formative` to contain non-zero values, but impose a restriction that the sum of `reflective` and `t(formative)` must satisfy the original restrictions of `reflective`. The only restrictions that matrixpls imposes on `inner`, `reflective`, and `formative` is that these must be binary matrices and that the diagonal of `inner` must be zeros.

Model estimation proceeds as follows. The weights `W` and the data covariance matrix `S` are used to calculate the composite covariance matrix `C` and the indicator-composite covariance matrix `IC`. These are matrices are used to separately estimate each of teh three model matrices `inner`, `reflective`, and `formative`. This approach of estimating the parameter matrices separately is the standard way of estimation in the PLS literature.

The default estimation approach is to estimate all parameters with a series of OLS regressions using `estimator.ols`.

## Value

A named vector of parameter estimates.

`parameterEstim.separate` returns the following as attributes:

| | |
|---|---|
| C | the composite correlation matrix (after disattenuation, if requested). |
| IC | the indicator-composite covariance matrix (after disattenuation, if requested). |
| inner | the inner model matrix with estimated parameters. |
| reflective | the reflective model matrix with estimated parameters. |
| formative | the formative model matrix with estimated parameters. |
| Q | the reliability estimates used in dissattenuation. |

Additionally, all attributes returned by functions called by parameterEstim.separate are returned. This can include:

| | |
|---|---|
| c | the PLSc loading estimate correction factors. |

## Functions

- parameterEstim.separate: Estimates the model parameters in inner, reflective, and formative separately.

## References

Rosseel, Y. (2012). lavaan: An R Package for Structural Equation Modeling. *Journal of Statistical Software*, 48(2), 1–36. Retrieved from http://www.jstatsoft.org/v48/i02

---

# predict.matrixpls *Predict method for matrixpls results*

---

## Description

The matrixpls method for the generic function predict predict. Predicts the reflective indicators of endogenous latent variables using estimated model and data for the indicators of exogenous latent variables

## Usage

```
## S3 method for class 'matrixpls'
predict(
  object,
  newData,
  predictionType = c("exogenous", "redundancy", "communality", "composites"),
  means = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| object | matrixpls estimation result object produced by the matrixpls function. |
| newData | A data frame or a matrix containing data used for prediction. |
| predictionType | "exogenous" (default) predicts indicators from exogenous composites. "redundancy" and "communality" are alternative strategies described by Chin (2010). "composites" returns the composites calculated by multiplying the data with the weight matrix. |
| means | A vector of means of the original data used to calculate intercepts for the linear prediciton equations. If not provided, calculated from the new data or assumed zero. |
| ... | All other arguments are ignored. |

## Value

a matrix of predicted values for reflective indicators of endogenous latent variables or weighted composites of the indicators.

## References

Wold, H. (1974). Causal flows with latent variables: Partings of the ways in the light of NIPALS modelling. *European Economic Review*, 5(1), 67–86. doi:10.1016/0014-2921(74)90008-7

Chin, W. W. (2010). How to write up and report PLS analyses. In V. Esposito Vinzi, W. W. Chin, J. Henseler, & H. Wang (Eds.), Handbook of partial least squares (pp. 655–690). Berlin Heidelberg: Springer.

## See Also

Other post-estimation functions: ave(), cr(), effects.matrixpls(), fitSummary(), fitted.matrixpls(), gof(), htmt(), loadings(), r2(), residuals.matrixpls()

## Examples

```
library(plspm)

# Run the customer satisfaction example form plspm

# load dataset satisfaction
data(satisfaction)
# inner model matrix
IMAG = c(0,0,0,0,0,0)
EXPE = c(1,0,0,0,0,0)
QUAL = c(0,1,0,0,0,0)
VAL = c(0,1,1,0,0,0)
SAT = c(1,1,1,1,0,0)
LOY = c(1,0,0,0,1,0)
inner = rbind(IMAG, EXPE, QUAL, VAL, SAT, LOY)
colnames(inner) <- rownames(inner)

# Reflective model
```

```
reflective<- matrix(
  c(1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1),
  27,6, dimnames = list(colnames(satisfaction)[1:27],colnames(inner)))

# empty formative model

formative <- matrix(0, 6, 27, dimnames = list(colnames(inner),
                                              colnames(satisfaction)[1:27]))

satisfaction.model <- list(inner = inner,
                           reflective = reflective,
                           formative = formative)

# Estimation using covariance matrix


satisfaction.out <- matrixpls(cov(satisfaction[,1:27]),
                           model = satisfaction.model)

print(satisfaction.out)



# Predict indicators using means from the data
predict(satisfaction.out,
        newData = satisfaction,
        means= sapply(satisfaction, mean))

# Calculate composite scores
predict(satisfaction.out,
        newData = satisfaction,
        predictionType = "composites")
```

---

q2 *Q2 predictive relevance statistics*

---

## Description

Calculates Q2 predictive relevance statistics based on comparing predictions and real data.

## Usage

```
q2(originalData, predictedData, model = NULL)
```

### Arguments

| | |
|---|---|
| originalData | A matrix or a data.frame containing the original data. |
| predictedData | A matrix or a data.frame containing the predicted data that are compared against the original data to calculate the predictive relevance statistic. |
| model | There are two options for this argument: 1. lavaan script or lavaan parameter table, or 2. a list containing three matrices inner, reflective, and formative defining the free regression paths in the model. |

### Details

The Q2 statistic is calculated as 1-sse/sso where sse is the sum of squared prediction errors based on comparison of the originalData and predictedData and sso is based on prediction with mean. If the predicted data contain the groups attribute, which indicates the groups used in blindfolding or cross-validation, the means are calculated separately for each group excluding the predicted group from the calculation.

### Value

A list with total, block, and indicator elements containing the Q2 predictive relevance statistics for the full dataset, for each indicator block, and for each indicator

---

## r2 *R2 for matrixpls results*

---

### Description

The matrixpls method for the generic function r2 computes the squared multiple correlation (R2) for composites predicted by other composites in the model.

### Usage

```
r2(object, ...)
```

### Arguments

| | |
|---|---|
| object | matrixpls estimation result object produced by the matrixpls function. |
| ... | All other arguments are ignored. |

### Value

A named numberic vector containing the R2 values.

### See Also

Other post-estimation functions: ave(), cr(), effects.matrixpls(), fitSummary(), fitted.matrixpls(), gof(), htmt(), loadings(), predict.matrixpls(), residuals.matrixpls()

## reliabilityEstim *Reliabilities as products of weights and loadings*

### Description

Calculates reliabilities as a matrix product of loadings and weights.

### Usage

```
reliabilityEstim.weightLoadingProduct(S, loadings, W, ...)

reliabilityEstim.alpha(S, loadings, W, ...)
```

### Arguments

| | |
|---|---|
| S | the data covariance matrix |
| loadings | matrix of factor loading estimates |
| W | matrix of weights |
| ... | All other arguments are ignored. |

### Value

a named vector of estimated composite reliabilities.

### Functions

- `reliabilityEstim.weightLoadingProduct`: Reliability estimation based on weights and loadings.
- `reliabilityEstim.alpha`: Reliability estimation with Cronbach's alpha

## residuals.matrixpls *Residual diagnostics for matrixpls results*

### Description

The `matrixpls` method for generic function `residuals` computes the residual covariance matrix and various fit indices.

### Usage

```
## S3 method for class 'matrixpls'
residuals(object, ..., observed = TRUE)
```

## Arguments

| | |
|---|---|
| `object` | matrixpls estimation result object produced by the `matrixpls` function. |
| `...` | All other arguments are ignored. |
| `observed` | If `TRUE` (default) the observed residuals from the outerEstim.model regressions (indicators regressed on composites) are returned. If `FALSE`, the residuals are calculated by combining `inner`, `reflective`, and `formative` as a simultaneous equations system and subtracting the covariances implied by this system from the observed covariances. The error terms are constrained to be uncorrelated and covariances between exogenous observed values are fixed at their sample values. |

## Details

The residuals can be either observed residuals from the regressions of indicators on composites and composites on composites (i.e. the `reflective` and `inner` models) as presented by Lohmöller (1989, ch 2.4) or model implied residuals calculated by subtracting model implied covariance matrix from the sample covariance matrix as done by Henseler et al. (2014).

The root mean squared residual indices (Lohmöller, 1989, eq 2.118) are calculated from the off diagonal elements of the residual covariance matrix. The standardized root mean squared residual (SRMR) is calculated based on the standardized residuals of the `reflective` model matrix.

Following Hu and Bentler (1999, Table 1), the SRMR index is calculated by dividing with $p(p+1)/2$, where $p$ is the number of indicator variables. In typical SEM applications, the diagonal of residual covariance matrix consistes of all zeros because error term variances are freely estimated. To make the SRMR more comparable with the index produced by SEM software, the SRMR is calculated by summing only the squares of off-diagonal elements, which is equivalent to including a diagonal of all zeros.

Two versions of the SRMR index are rovided, the traditional SRMR that includes all residual covariances, and the version proposed by Henseler et al. (2014) where the within-block residual covariances are ignored.

## Value

A list with three elements: `inner`, `outer`, and `indices` elements containing the residual covariance matrix of regressions of composites on other composites, the residual covariance matrix of indicators on composites, and various indices calculated based on the residuals.

## References

Henseler, J., Dijkstra, T. K., Sarstedt, M., Ringle, C. M., Diamantopoulos, A., Straub, D. W., ... Calantone, R. J. (2014). Common Beliefs and Reality About PLS Comments on Rönkkö and Evermann (2013). *Organizational Research Methods*, 17(2), 182–209. doi:10.1177/1094428114526928

Hu, L., & Bentler, P. M. (1999). Cutoff criteria for fit indexes in covariance structure analysis: Conventional criteria versus new alternatives. *Structural Equation Modeling: A Multidisciplinary Journal*, 6(1), 1–55.

Lohmöller J.-B. (1989) *Latent variable path modeling with partial least squares.* Heidelberg: Physica-Verlag.

## See Also

Other post-estimation functions: `ave()`, `cr()`, `effects.matrixpls()`, `fitSummary()`, `fitted.matrixpls()`, `gof()`, `htmt()`, `loadings()`, `predict.matrixpls()`, `r2()`

---

`signChange` *Sign change corrections for bootstrap*

---

## Description

These functions selectively reverse the signs of the weights in boostrap samples to be consistent with the weights calculated based on the original sample.

## Usage

```
signChange.individual(Worig, W)

signChange.construct(Worig, W)
```

## Arguments

| | |
|---|---|
| `Worig` | The original weight matrix. |
| `W` | a Weight matrix of a bootstrap sample. |

## Details

Sign change corrections are a controversial and inconsistently implemented feature in PLS analysis. The two corrections described in the literature are the individual sign chance correction and the construct level sign chance corrections. The individual correction changes the signs of `W` to match `Worig`.

The construct level correction changes the signs of `W` on all rows where the sign of the sum of the row differs between `Worig` and `W`.

The sign chance corrections are described ambiquosly and sometimes implemented inconsistently between software. **matrixpls** implements the corrections by adjusting the weights before calculating parameter estimates in each bootstrap replication. Some software implement the correction post-hoc by adjusting the bootstrap estimates directly. Moreover, the literature present at least two different formulas for the construct level correction. **matrixpls** implements the version described by Tenenhaus et al. (2005).

The sign chance corrections should not be confused with sign indeterminacy corrections applied to individual analyses (See `weightSign`).

## Value

A weight matrix with the same dimensions as `W` after applying the correction.

### Functions

- `signChange.individual`: individual sign change correction
- `signChange.construct`: individual sign change correction

### References

Tenenhaus, M., Esposito Vinzi, V., Chatelin, Y.-M., & Lauro, C. (2005). PLS Path Modeling. *Computational Statistics & Data Analysis*, 48(1), 159–205. doi:10.1016/j.csda.2004.03.005

Rönkkö, M., McIntosh, C. N., & Antonakis, J. (2015). On the adoption of partial least squares in psychological research: Caveat emptor. *Personality and Individual Differences*, (87), 76–84. DOI:10.1016/j.paid.2015.07.019

### See Also

`matrixpls.boot`

---

## `weightFun` *Indicator weigth algoritms*

---

### Description

Estimates a weight matrix using Partial Least Squares or a related algorithm.

`weightFun.factor` calculates weights by estimating a common factor analysis model with a single factor for each indicator block and using the resulting estimates to calculate factor score weights

`weightFun.principal` calculates weights by calculating a principal component analysis for each indicator block and returning the weights for the first principal component.

### Usage

```
weightFun.pls(
  S,
  model,
  W.model,
  outerEstim = NULL,
  innerEstim = innerEstim.path,
  ...,
  convCheck = convCheck.absolute,
  variant = "lohmoller",
  tol = 1e-05,
  iter = 100,
  validateInput = TRUE
)

weightFun.optim(
  S,
```

```
  model,
  W.model,
  parameterEstim = parameterEstim.separate,
  optimCrit = optimCrit.maximizeInnerR2,
  method = "BFGS",
  ...,
  validateInput = TRUE,
  standardize = TRUE
)

weightFun.fixed(S, model, W.model = NULL, ..., standardize = TRUE)

weightFun.factor(
  S,
  model,
  W.model = NULL,
  ...,
  fm = "minres",
  standardize = TRUE
)

weightFun.principal(S, model, W.model = NULL, ..., standardize = TRUE)
```

## Arguments

| | |
|---|---|
| S | Covariance matrix of the data. |
| model | There are two options for this argument: 1. lavaan script or lavaan parameter table, or 2. a list containing three matrices inner, reflective, and formative defining the free regression paths in the model. |
| W.model | A matrix specifying the weight relationships and their starting values. |
| outerEstim | A function or a list of functions used for outer estimation. If the value of this parameter is a function, the same function is applied to all composites. If the value is a list, the composite n is estimated with the estimator in the nth position in the list. If this argument is NULL the outerEstim.modeA is used for all composites that are linked to at least one indicator in the reflective matrix.outerEstim.modeB is used for all other composites. See outerEstim. |
| innerEstim | A function for calculating inner weights using the data covariance matrix S, a weight matrix W, and an inner model matrix inner.mod. Returns an inner weigth matrix E. The default is innerEstim.path. |
| ... | All other arguments are passed through to other estimation functions. |
| convCheck | A function that takes the old Wold and new weight Wold matrices and returns a scalar that is compared against tol to check for convergence. The default is convCheck.absolute. |
| variant | Choose either Lohmöller's ("lohmoller", default) or Wold's ("wold") variant of PLS. In Wold's variant the inner and outer estimation steps are repeated for each indicator block whereas in Lohmöller's variant the weights for all composites are calculated simultaneously. |

| | |
|---|---|
| tol | Decimal value indicating the tolerance criterion for convergence. |
| iter | An integer indicating the maximum number of iterations. |
| validateInput | A boolean indicating whether the validity of the parameter values should be tested. |
| parameterEstim | A function for estimating the model parameters using the data covariance matrix S, model specification model, and weight matrix W. Returns a named vector of parameter estimates. The default is parameterEstim.separate |
| optimCrit | A function for calculating value for an optimization criterion based on a matrixpls result object. Returns a scalar. The default is optimCrit.maximizeInnerR2. |
| method | The minimization algorithm to be used. See optim for details. Default is "BFGS". |
| standardize | A boolean indicating whether S the weights should be scaled to produce standardized composites. |
| fm | factoring method for estimating the common factor model. Possible values are minres, wls, gls, pa, and ml. The parameter is passed through to to fa. |

## Details

Model can be specified in the lavaan format or the native matrixpls format. The native model format is a list of three binary matrices, inner, reflective, and formative specifying the free parameters of a model: inner (l x l) specifies the regressions between composites, reflective (k x l) specifies the regressions of observed data on composites, and formative (l x k) specifies the regressions of composites on the observed data. Here k is the number of observed variables and l is the number of composites.

If the model is specified in lavaan format, the native format model is derived from this model by assigning all regressions between latent variables to inner, all factor loadings to reflective, and all regressions of latent variables on observed variables to formative. Regressions between observed variables and all free covariances are ignored. All parameters that are specified in the model will be treated as free parameters.

The original papers about Partial Least Squares, as well as many of the current PLS implementations, impose restrictions on the matrices inner, reflective, and formative: inner must be a lower triangular matrix, reflective must have exactly one non-zero value on each row and must have at least one non-zero value on each column, and formative must only contain zeros. Some PLS implementations allow formative to contain non-zero values, but impose a restriction that the sum of reflective and t(formative) must satisfy the original restrictions of reflective. The only restrictions that matrixpls imposes on inner, reflective, and formative is that these must be binary matrices and that the diagonal of inner must be zeros.

The argument W.model is a (l x k) matrix that indicates how the indicators are combined to form the composites. The original papers about Partial Least Squares as well as all current PLS implementations define this as t(reflective) | formative, which means that the weight patter must match the model specified in reflective and formative. Matrixpls does not require that W.model needs to match reflective and formative, but accepts any numeric matrix. If this argument is not specified, all elements of W.model that correspond to non-zero elements in the reflective or formative formative matrices receive the value 1.

`weightFun.pls` calculates indicator weights by calling the `innerEstim` and `outerEstim` iteratively until either the convergence criterion or maximum number of iterations is reached and provides the results in a matrix.

`weightFun.optim` calculates indicator weights by optimizing the indicator weights against the criterion function using `optim`. The algoritmh works by first estimating the model with the starting weights. The resulting `matrixpls` object is passed to the `optimCrit` function, which evaluates the optimization criterion for the weights. The weights are adjusted and new estimates are calculated until the optimization criterion converges.

## Value

An object of class `"matrixplsweights"`, which is a matrix containing the weights with the following attributes:

| | |
|---|---|
| `iterations` | Number of iterations performed |
| `converged` | A boolean indicating if the algorithm converged |
| `history` | A data.frame containing the weights for each iteration |

`weightFun.pls` returns the following as attributes:

| | |
|---|---|
| `S` | the sample covariance matrix. |
| `E` | inner weight matrix. |
| `iterations` | the number of iterations used by the weight algorithm. |
| `converged` | `TRUE` if the weight algorithm converged and `FALSE` otherwise. |
| `history` | weight optimization history as a matrix. |

## Functions

- `weightFun.pls`: partial Least Squares and other iterative two-stage weight algorithms.
- `weightFun.optim`: calculates a set of weights to minimize an optimization criterion.
- `weightFun.fixed`: returns the starting weights.
- `weightFun.factor`: blockwise factor score weights.
- `weightFun.principal`: blockwise principal component weights.

## References

Rosseel, Y. (2012). lavaan: An R Package for Structural Equation Modeling. *Journal of Statistical Software*, 48(2), 1–36. Retrieved from http://www.jstatsoft.org/v48/i02

## Examples

```
library(plspm)

# Run the customer satisfaction examle form plspm

# load dataset satisfaction
data(satisfaction)
```

```
# inner model matrix
IMAG = c(0,0,0,0,0,0)
EXPE = c(1,0,0,0,0,0)
QUAL = c(0,1,0,0,0,0)
VAL = c(0,1,1,0,0,0)
SAT = c(1,1,1,1,0,0)
LOY = c(1,0,0,0,1,0)
inner = rbind(IMAG, EXPE, QUAL, VAL, SAT, LOY)
colnames(inner) <- rownames(inner)

# Reflective model
list(1:5, 6:10, 11:15, 16:19, 20:23, 24:27)

reflective<- matrix(
  c(1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1),
  27,6, dimnames = list(colnames(satisfaction)[1:27],colnames(inner)))

# empty formative model

formative <- matrix(0, 6, 27, dimnames = list(colnames(inner), colnames(satisfaction)[1:27]))

# Estimation using covariance matrix
model <- list(inner = inner,
              reflective = reflective,
              formative = formative)

S <- cov(satisfaction[,1:27])

matrixpls.ModeA <- matrixpls(S, model)
matrixpls.ModeB <- matrixpls(S, model, outerEstim = outerEstim.modeB)
matrixpls.MaxR2 <- matrixpls(S, model, weightFun = weightFun.optim)

# Compare the R2s from the different estimations

R2s <- cbind(r2(matrixpls.ModeA), r2(matrixpls.ModeB), r2(matrixpls.MaxR2))
print(R2s)
apply(R2s,2,mean)

# Optimization against custom function

maximizeSumOfCorrelations <- function(matrixpls.res){
  C <- attr(matrixpls.res,"C")
  model <- attr(matrixpls.res,"model")
  - sum(C[model$inner != 0])
}

matrixpls.MaxCor <- matrixpls(S, model, weightFun = weightFun.optim,
                              optimCrit = maximizeSumOfCorrelations)
```

```
# Compare the Mode B and optimized solutions

C <- attr(matrixpls.ModeB,"C")
print(C)
print(sum(C[inner != 0]))
C <- attr(matrixpls.MaxCor,"C")
print(C)
print(sum(C[inner != 0]))
```

---

`weightSign` *Sign ambiquity corrections*

---

## Description

Sign ambiquity corrections adjust the signs of the weights to satisfy a criterion.

## Usage

```
weightSign.Wold1985(W, S)

weightSign.dominantIndicator(W, S)
```

## Arguments

| | |
|---|---|
| `W` | Weight matrix, where the indicators are on colums and composites are on the rows. |
| `S` | Covariance matrix of the data. |

## Details

Instead of fixing a weight to a particular value, composite variables are typically provided a scale by standardization. This leads to sign indeterminacy because standardized weights `W` and `-W` both satisfy the scalign constraint. The sing ambiquity corrections add additional constraints that make

The sign indeterminacy corrections should not be confused with sign chance corrections applied to boostrap samples (See `signChange`).

## Value

`W` after sign correction.

## Functions

- `weightSign.Wold1985`: Adjust the signs of W so that the majority of the indicators are positively correlated with the composite as proposed by Wold (1985).
- `weightSign.dominantIndicator`: Adjust the signs of W so that the first indicator of each composite has positive weight.

## References

Wold, H. (1985). Partial Least Squares. In S. Kotz & N. L. Johnson (Eds.), Encyclopedia of statistical sciences (Vol. 6, pp. 581–591). New York: Wiley.

## See Also

matrixpls;

# Index