Introduction to matrixpls

Mikko Rönkkö
Aalto University

Abstract

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.

Keywords: partial least squares, generalized structured component analysis, composite-based modeling, R.

1. Introduction

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.

PLS has recently gained popularity in several disciplines as an alternative approach to structural equation modeling (SEM) or as an alternative to SEM itself (Hair, Sarstedt, Pieper, et al. 2012; Hair, Sarstedt, Ringle, et al. 2012; Ringle, Sarstedt, and Straub 2012; Rönkkö and Evermann 2013) The route through which PLS emerged into the mainstream in these disciplines was rather unorthodox. The PLS method was first published in the 1966 and slowly developed through the 70's and early 80's by Herman Wold (1966; 1974; 1980; 1982; 1985). Originally developed as an iterative least squares method for calculating principal components, canonical correlations, and other similar statistic, the method was later adopted as an approximate estimation algorithm for structural equation models with latent variables. However, Dijkstra (1983) soon proved that the PLS method was not consistent when used for this purpose and the PLS method never gained much attention from other econometricians or other researchers specializing in statistical analysis. Consequently, the PLS method is currently almost completely absent from the mainstream journals on research methods (Rönkkö and Evermann 2013).

The PLS method re-emerged, however, in the marketing and information systems disciplines (Hair, Ringle, and Sarstedt 2012), in which the popularity of the method can be attributed to a number of introductory articles that present PLS as an SEM method that has less stringent assumptions concerning the data and that avoids many of the perceived difficulties of SEM. The articles published by these applied scholars, and particularly a paper by Fornell and Bookstein (1982) and a book chapter by Chin (1998), formed the core of the modern understanding of the PLS method. These publications were followed by a number of articles that provided guidelines for application of PLS in disciplines such as strategic management (Hulland 1999), operations management (Peng and Lai 2012), marketing (Hair and Ringle 2011), and information systems (Gefen, Rigdon, and Straub 2011).

Today PLS is used extensively in information systems and marketing (Hair, Sarstedt, Ringle, et al. 2012; Henseler, Ringle, and Sinkovics 2009; Ringle, Sarstedt, and Straub 2012) and is increasingly used in management and organizational research (Hair, Sarstedt, Pieper, et al. 2012; Rönkkö and Evermann 2013), and has also been introduced into psychology (Willaby et al. 2015). However, the popularity of the method has also brought with it an increasing number of arcticles critical of the method (Rönkkö and Evermann 2013; Evermann and Tate 2013; Rönkkö 2014; Rönkkö, McIntosh, and Antonakis 2015; Rönkkö and Ylitalo 2010; Antonakis et al. 2010; Goodhue, Lewis, and Thompson 2012; Goodhue, Thompson, and Lewis 2013). These critics claim that many of the beliefs about the capabilities of the PLS method as an estimator of latent variable structural equations are unsubstantiated and not true, that the method capitalizes on chance, and that it does not have valid statistical tests. Some even go as far as declaring that the PLS method should never be used (Antonakis et al. 2010).

PLS has been challenged by algorithms that are argued to be superior by their developers. Hwang and Takane (2014; 2004) proposed generalized structured component analysis (GSCA) arguing that it is superior over PLS because it has an explicit optimization criterion, which the PLS algorithm lacks. Dijkstra (2011; Dijkstra and Henseler 2015b; Dijkstra and Henseler 2015a) proposed that PLS can be made consistent by applying disattenuation, referring to this estimator as PLSc. Huang (2013; Bentler and Huang 2014) proposed two additional estimators that parameterize LISREL estimators based on Dijkstra's PLSc estimator. These estimators, referred to as PLSe1 an PLSe2 are argued to be more efficient than the consistent PLSc estimator.

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** (Sanchez, Trinchera, and Russolillo 2015) and **semPLS** (Monecke and Leisch 2012) 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**.

1.1. Overview of design principles and the package functionality

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)
```

Extending the package with more functions is explained in more detail in the end of the paper.

1.2. Model matrices

Model can be specified in the lavaan format (Rosseel 2012) 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.

1.3. Performance

This design principle and the use of covariance matrices instead of raw data make **matrixpls** substantially more computationally efficient than the other two R packages that implement PLS path modeling algorithms. Table 1 and 2 benchmark the performance of **matrixpls** 1.0.0 against **plspm** 0.4.7 and **semPLS** 1.0.10 using the `satisfaction` example distributed with **plspm** on R version 3.3.0 running on Intel(R) Core(TM) i7-3720QM CPU @ 2.60GHz. Parallel bootstrapping results are presented only for **matrixpls** because the two other packages do not support parallel computing.

Table 1: Timings for **plspm**, **semPLS**, and **matrixpls** for 100 replications

	Time elapsed (seconds)
plspm	2.82
semPLS	8.29
matrixpls	0.23

Table 2: Timings for **plspm**, **semPLS**, and **matrixpls** for 1000 bootstrap replications

	Time elapsed (seconds)
plspm	14.63
semPLS	51.25
matrixpls (single core)	2.56
matrixpls (multicore)	0.72

2. Estimation using covariance data

The following sections provide an overview of the package functionality through code examples.

2.1. Basic PLS estimation

In a typical PLS implementation, the indicator weights are calculated iteratively starting from equal weights and then recalculating the weights for each composite in two steps, called inner and outer estimation. In the inner estimation step, new composites are calculated as weighted sums of "adjacent" composites, that is, composites that are directly related to the focal composite by regression relationships. The three commonly used "schemes" for inner estimation are centroid, path, and factor schemes. In practice these results often very similar results and are thus not discussed in detail here, but detailed descriptions can be found in the **matrixpls** reference manual. The inner estimation step is calculated based on composite correlation matrix `C`, which is calculated as:

$$\boldsymbol{C} = \boldsymbol{W}^{\intercal}\boldsymbol{S}\boldsymbol{W} \tag{1}$$

where $\boldsymbol{S}$ is the $(k \times k)$ indicator covariance matrix, and $\boldsymbol{W}$ is the $(l \times k)$ weight matrix, and k is the number of indicators and l is the number of observed variables.

In the outer estimation step, new indicator weights are calculated in one of two ways. In Mode A estimation, the observed variables are regressed on the composites, whereas in Mode B estimation, the composites are regressed on the observed variables. The new indicator weights are then used to calculate new composites for the following round of inner estimation. These two steps are repeated until the changes in the indicator weights become sufficiently small to consider the model converged.

The correlations between the indicators and composites after inner estimation required for outer estimation are calculated as

$$\boldsymbol{IC} = \boldsymbol{S}\boldsymbol{W}\boldsymbol{E} \tag{2}$$

where $\boldsymbol{E}$ is a $(l \times l)$ inner weight matrix.

The **matrixpls** PLS weight algorithm implementation is summarized in Table 3

Table 3: PLS weight algorithm

Inner estimation	Inner estimation function is applied to the data covariance matrix `S`, weight matrix `W`, and composite variable model matrix `inner`. The function returns an inner weight matrix `E`.
Outer estimation	Outer estimation function is applied to the data covariance matrix `S`, weight matrix `W`, inner weight matrix `E`, and weight model matrix `W.model`. The function returns a weight matrix `W`.
Convergence check	Convergence check function is applied to the weight matrix `W` before and after outer estimation. This function returns a scalar that is compared against the tolerance value. If the scalar is smaller than the tolerance value, the algorithm converges. Otherwise, a new iteration is started.

The PLS weight algorithm is implemented with the `weightFun.pls` function:

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

The inner and outer estimation functions define the weight algorithm. The package includes all well-known and a number of less known functions. The built-in outer estimation functions are listed in Table 4 and Table 5 lists the inner estimators.

Table 4: Outer weight functions

outerEstim.modeA	PLS Mode A weights. Returns weights that are proportional to correlations between indicators and composites
outerEstim.modeB	PLS Mode B weights. Returns weights that are proportional to coefficients from regression of composites on indicators.
outerEstim.gsca	GSCA outer weights. Described later in the paper.

`outerEstim` can be either a single function or a list of functions with lenght equal to the number of composites. If just one function is provided, the same function will be used for all composites. The default is to use Mode A and Mode B based on the `reflective` and `formative` parts of `model` following the tradition in the PLS literature. If at least one of the indicators that belong to a composite is specified as being regressed on the composite in the `reflective` matrix, then Mode A is used for that composite. Otherwise Mode B is used.

Table 5: Inner weight functions

innerEstim.path	Path weighting scheme. Returns inner weights based on regressions and correlations of adjacent composites. (default)
innerEstim.centroid	Centroid weighting scheme. Returns signs of correlations of adjacent composites as weights
innerEstim.factor	Factor weighting scheme. Returns correlations of adjacent compositesas weights
innerEstim.identity	Returns identity matrix as inner weights. Each composite is used as its own inner approximation
innerEstim.gsca	GSCA inner weights. Described later in the paper.

After the weights have been calculated, the parameter estimator function defined provided in the `parameterEstim` argument is applied to the data covariance matrix `S`, the weight matrix `W`, and `model`. The default estimation command, `params.separate`, processes the three model matrices, `inner`, `reflective`, and `formative`, separately by default applying OLS regression row by row. This follows the tradition in PLS analysis. Extensions to this technique are discussed later in the paper when discussing consistent PLSc estimation.

Example: Customer satisfaction

The following code example demonstrates a PLS analysis by replicating the customer satisfaction example from the **plspm** package:

```
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)


 matrixpls parameter estimates
           Est.
EXPE~IMAG   0.56
SAT~IMAG    0.19
LOY~IMAG    0.29
QUAL~EXPE   0.85
VAL~EXPE    0.12
SAT~EXPE    0.01
VAL~QUAL    0.66
SAT~QUAL    0.14
SAT~VAL     0.58
LOY~SAT     0.47
IMAG=~imag1 0.75
IMAG=~imag2 0.89
IMAG=~imag3 0.87
IMAG=~imag4 0.63
IMAG=~imag5 0.69
EXPE=~expe1 0.79
EXPE=~expe2 0.82
EXPE=~expe3 0.73
EXPE=~expe4 0.77
EXPE=~expe5 0.82
QUAL=~qual1 0.79
QUAL=~qual2 0.87
QUAL=~qual3 0.76
QUAL=~qual4 0.82
QUAL=~qual5 0.81
VAL=~val1   0.86
VAL=~val2   0.83
VAL=~val3   0.76
VAL=~val4   0.82
```

```
SAT=~sat1   0.92
SAT=~sat2   0.91
SAT=~sat3   0.83
SAT=~sat4   0.82
LOY=~loy1   0.89
LOY=~loy2   0.72
LOY=~loy3   0.88
LOY=~loy4   0.71

 matrixpls weights
     imag1 imag2 imag3 imag4 imag5 expe1 expe2 expe3 expe4 expe5 qual1
IMAG  0.21   0.3  0.31  0.18  0.28  0.00  0.00  0.00  0.00  0.00  0.00
EXPE  0.00   0.0  0.00  0.00  0.00  0.24  0.28  0.22  0.26  0.27  0.00
QUAL  0.00   0.0  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.24
VAL   0.00   0.0  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
SAT   0.00   0.0  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
LOY   0.00   0.0  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
     qual2 qual3 qual4 qual5 val1 val2 val3 val4 sat1 sat2 sat3 sat4 loy1
IMAG  0.00  0.00  0.00  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
EXPE  0.00  0.00  0.00  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
QUAL  0.27  0.23  0.25  0.25 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
VAL   0.00  0.00  0.00  0.00 0.35 0.29 0.25 0.33 0.00 0.00 0.00 0.00 0.00
SAT   0.00  0.00  0.00  0.00 0.00 0.00 0.00 0.00 0.32 0.31 0.25 0.27 0.00
LOY   0.00  0.00  0.00  0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.38
     loy2 loy3 loy4
IMAG 0.00 0.00 0.00
EXPE 0.00 0.00 0.00
QUAL 0.00 0.00 0.00
VAL  0.00 0.00 0.00
SAT  0.00 0.00 0.00
LOY  0.25 0.37 0.22

Weight algorithm converged in 5 iterations.
```

Example: Political democracy

The following code example demonstrates a PLS analysis by replicating the political democracy example from the **lavaan** package:

```
library(lavaan)

## The industrialization and Political Democracy example
## Bollen (1989), page 332. (Adopted from the lavaan package.)
model <- '
  # latent variable definitions
     ind60 =~ x1 + x2 + x3
     dem60 =~ y1 + a*y2 + b*y3 + c*y4
     dem65 =~ y5 + a*y6 + b*y7 + c*y8

  # regressions
    dem60 ~ ind60
    dem65 ~ ind60 + dem60
  # residual correlations
    y1 ~~ y5
    y2 ~~ y4 + y6
    y3 ~~ y7
    y4 ~~ y8
    y6 ~~ y8
'

political.out <- matrixpls(cov(PoliticalDemocracy),model)
print(political.out)
```

```
 matrixpls parameter estimates
           Est.
dem60~ind60 0.40
dem65~ind60 0.20
dem65~dem60 0.79
ind60=~x1   0.95
ind60=~x2   0.97
ind60=~x3   0.92
dem60=~y1   0.88

(part of the output omitted)

 matrixpls weights
       x1   x2   x3   y1   y2   y3   y4  y5   y6   y7   y8
ind60 0.38 0.37 0.31 0.00 0.00 0.00 0.00 0.0 0.00 0.00 0.00
dem60 0.00 0.00 0.00 0.31 0.27 0.26 0.33 0.0 0.00 0.00 0.00
dem65 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.3 0.28 0.29 0.29

Weight algorithm converged in 4 iterations.
```

The `matrixpls` function returns a vector of estimates with the class `matrixpls`. The final values of the matrices used in the iterative estimation procedure, weight history, and other data about the calcuation are returned as attributes:

```
names(attributes(political.out))
```

```
 [1] "names"      "S"          "E"          "iterations" "converged"
 [6] "history"    "C"          "IC"         "inner"      "reflective"
[11] "formative"  "W"          "model"      "call"       "class"
```

Because the `matrixpls` function aggregates preliminary results from all calculation stages, the exact matrices that are returned depend on the functions and options used. All matrices are described later in the paper.

2.2. Traditional indicator weighting schemes

matrixpls implements three traditional indicator weighting systems: equal weights, factor score weights, and principal component weights. The weights are calculated blockwise using the factor analysis and principal component analysis functions form the **psych** package (Revelle 2015).

The following code example demonstrates the use of these three indicator weighting systems and compares the results with the PLS results for the political democracy example.

Example: Traditional indicator weighting systems

```
fixed.out <- matrixpls(cov(PoliticalDemocracy),model,
                       weightFun = weightFun.fixed)
factor.out <- matrixpls(cov(PoliticalDemocracy),model,
                        weightFun = weightFun.factor)
principal.out <- matrixpls(cov(PoliticalDemocracy),model,
                           weightFun = weightFun.principal)

cbind(political.out, fixed.out, factor.out, principal.out)
```

```
            political.out fixed.out factor.out principal.out
dem60~ind60          0.40      0.39       0.42          0.40
dem65~ind60          0.20      0.20       0.17          0.20
dem65~dem60          0.79      0.78       0.78          0.78
ind60=~x1            0.95      0.95       0.93          0.95
ind60=~x2            0.97      0.97       0.99          0.97
ind60=~x3            0.92      0.93       0.89          0.93
dem60=~y1            0.88      0.88       0.87          0.88

(part of the output omitted)
```

Because the results objects are vectors, we can conveniently bind them as a matrix for comparison. The differences are small, which is to be expected because indicator weighting has generally a very small effect on reliability (Cohen 1990; Raju et al. 1999; Bobko, Roth, and Buster 2007).

2.3. Optimized weights

The third alternative to PLS weight calculation and the more traditional indicator weight algorithms is optimized weights. These weights are numerically optimized to minimize a criterion value calculated based on the `matrixpls` result object. (Maximization can be done by taking a negative of the criterion.) Although PLS weights are often claimed to be optimal in the literature about PLS, the literature is unclear for which specific purpose the weights are optimal for and we consequently also lack any proofs of optimality (Rönkkö, McIntosh, and Antonakis 2015). In contrast, numerical optimization of weights can be used to produce optimal weights.

Optimized weights are calculated with the `weightFun.optim` function:

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

The function signature is very similar to the `matrixpls` function. The `optimCriterion` is a function that calculates the criterion value from `matrixpls` return object. The criterion is minimized by adjusting the weights using the `optim` function from the **stats** package. The default optimization method is Broyden–Fletcher–Goldfarb–Shannon algorithm, but this can be changed with the `method` argument and other parameters, which are passed on to `optim`.

The optimization starts by applying `parameterEstim` to the starting weights and calculates the criterion value based on the results using `optimCriterion` function. After this, the algorithm tries several alternative weights to decide the direction and step size which are used to decide on the new weights. Optimization proceeds then by calculating a new criterion value with the new weights and then calculating new direction and step size. This iterative adjustment of weights continues until convergence. **matrixpls** provides three built-in optimization criterion functions shown in Table 6.

Table 6: Weight optimization criterion functions

optimCrit.maximizeInnerR2	Returns the negative of mean of R^2 of all regressions in the `inner` matrix. Leads to maximising the mean R^2 of these regressions. (default)
optimCrit.maximizeIndicatorR2	Returns the negative of mean of R^2 of all regressions in the `reflective` matrix. Leads to maximising the mean R^2 of these regressions.
optimCrit.maximizeFullR2	Returns the negative of mean of R^2 of all regressions in the `inner` matrix and the `reflective` matrix. Leads to maximising the mean of R^2 of these regressions.
optimCrit.gsca	GSCA optimization criterion. Described later in the paper.

Example: Comparing inner model explained variance over four sets of weights

In the example below we create an arbitrary correlation matrix of six variables which we use to define three composites, A, B, and C of two indicators each. The composite C is then regressed on A and B. We compare the R^2 value of this regression using two sets of PLS weights, equal weights, and weights optimized for maximizing R^2.

```
S <- diag(6)
S[upper.tri(S, diag = FALSE)] <- c(.3,
                                   -.4,-.4,
                                   .4,.4,.3,
                                   .3,.3,.3,.3,
                                   .3,.3,.3,.3,.3)
S[lower.tri(S, diag = FALSE)] <- t(S)[lower.tri(S, diag = FALSE)]

inner <- matrix(c(0,0,1,
                  0,0,1,
                  0,0,0),3,3)

reflective <- diag(3)[rep(1:3, each = 2),]
```

```
formative <- matrix(0,3,6)

colnames(inner) <- rownames(inner) <- colnames(reflective) <-
  rownames(formative) <- c("A","B","C")
colnames(S) <- rownames(S) <- colnames(formative) <-
  rownames(reflective) <- c("a1","a2","b1","b2","c1","c2")

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

modeA <- matrixpls(S,model, outerEstim = outerEstim.modeA)
modeB <- matrixpls(S,model, outerEstim = outerEstim.modeB)
fixed <- matrixpls(S,model, weightFun = weightFun.fixed)
optimR2 <- matrixpls(S,model, weightFun = weightFun.optim)

rbind(ModeA = r2(modeA),
      ModeB = r2(modeB),
      Fixed = r2(fixed),
      Optim = r2(optimR2))

      A B    C
ModeA 0 0 0.43
ModeB 0 0 0.43
Fixed 0 0 0.43
Optim 0 0 0.93
```

In this particular scenario, the optimized weights result in 118% higher R^2 value than either of the PLS weights.

2.4. GSCA estimation

Generalized structured component analysis differes from PLS by defining an explicit optimization criterion and an algorithm for weight calculation (Hwang and Takane 2004). However, the initially presented algorith contained a scaling error that was corrected only after six years in 2010 (Hwang et al. 2010; Henseler 2010; Hwang, Malhotra, and Kim 2010; Henseler 2012). **matrixpls** addresses the scaling issue by always standardizing composites and unless the `standardize` option is set to false also the indicators.

The optimization criterion for the corrected GSCA is to minimize the sum of squares of all regressions specified in `inner` and `reflective`, which is equivalent to maximixing the sum of R^2 of these regressions. In the original GSCA specification, the formative model, if included, always overlap completely with the indicators forming the composite and therefore the R^2 of regressions in the `formative` part of the model is always 1 and therefore the `formative` matrix is ignored when calculating the GSCA weights.

The GSCA estimation algorithm consists of two steps. In the first step, all model regressions are estimated by minimizing the sum of squared residuals keeping the weights fixed. In the second step, the sum of squared residuals of the same regressions are minimized for weights keeping the regression coefficients fixed.

matrixpls implementes GSCA using the functions `innerEstim.gsca` and `outerEstim.gsca` that are used with the `weightFun.pls` weight function. The algorithm starts by generating initial composites using the starting weights. The intial composites are first used to estimate the inner weight matrix $\boldsymbol{E}$ consisting of the regression coefficients between the composites using the function `innerEstim.gsca`. Because some of the composites are exogenous to other composites, the $\boldsymbol{E}$ matrix from GSCA inner estimation should not be used with PLS outer estimation functions.

The outer estimation function `outerEstim.gsca` starts by estimating the regression models between indicators and composites using the `reflective` matrix. These regression coefficients and the $\boldsymbol{E}$ matrix can be combined to form the $\boldsymbol{A}$ matrix in the Hwang and Takane (2004) article. This completes the first step of the GSCA estimation algorithm.

In the second step, the weights are updated holding the regression coeffiecients fixed. Because one weight can be used in multiple equations, the sum of squares cannot be minimized by estimating each regression equation separately. Rather, the weights are updated one composite at a time taking into consideration all equations where the composite is used. In Hwang and Takane (2004), the weights for one composite are defined by specifying a series of regression analyses where the indicators are independent variables. These regressions are then estimated simultaneously by stacking the data so that the system of equations can be estimated with OLS estimator in one go. This is equivalent to collecting all covariances between the independent variables and dependent variables into a matrix and then taking a mean over all dependent variables so that we have a vector of mean covariances for each independent variable. This aggregated covariance vector is then used with the sample covariance matrix of the independent variables obtain new least squares estimates of the weights for the composite.

Hwang and Takane present the above mention algorithm as a way to minimize the GSCA estimation criterion. However, they note that the algorithm many not result in global optimum (Hwang and Takane 2004, 87–88). In `matrixpls` these scenarios can be addressed by using numerical optimization with the `weightFun.optim` weight function and `optimCrit.gsca` optimization criterion, which together maximize the GSCA estimation criterion by using numerical optimization of the weights. Because the GSCA optimization criterion does not have known closed-form derivatives, the hessian matrix used by the optimization algorithm must be calculated numerically. This makes the weight optimization technique much slower than the alternating least squares technique.

The example below demonstrates this and compares the results against GSCA implementation provided by the **ASGSCA** package (Romdhani et al. 2014) available throuh Bioconductor.

Example: GSCA estimation with alternating least squares and direct numerical optimization

```
# 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]))

     ASGSCA matrixpls_als matrixpls_optim
 [1,]  -0.31         -0.31           -0.31
 [2,]   1.06          1.06            1.06
 [3,]   0.08          0.08            0.08
 [4,]   0.95          0.95            0.95
 [5,]   0.34          0.34            0.34
 [6,]   0.49          0.49            0.49
 [7,]   0.53          0.53            0.53
 [8,]   0.66          0.66            0.66
 [9,]   0.59          0.59            0.59
```

```
# Check the criterion function values

optimCrit.gsca(matrixpls.res1)

[1] 1.9

optimCrit.gsca(matrixpls.res2)

[1] 1.9
```

2.5. PLSc and other disattenuation techniques

Recent research by Dijkstra and coauthors (2011; Dijkstra and Henseler 2015b; Dijkstra and Henseler 2015a) introduced the correction for attenuation to PLS literature. Labeled as PLSc, Dijkstra's technique consists of four enhancements to the traditional PLS analysis:

1. Consistent estimation of factor loadings
2. Consistent estimation of the reliabilities of weighted composite
3. Application of correction for attenuation using the reliability estimates
4. (optionally) two-stage least square (2SLS) estimation of the `inner` matrix regressions

`matrixpls` implements this full collection of techniques with a collection of functions. Table 7 list the included factor loading estimation functions.

Table 7: Factor loading estimation functions

estimator.regression	Estimates loadings as regressions of indicators on composites. (default)
estimator.plscLoadings	Dijkstra's technique. Estimates loadings one block at a time using MINRES estimator and constraining the loadings to be proportinal to the weights.
estimator.efaLoadings	Estimates loadings one block at a time with using the `fa` function of the **psych** package. Defaults to MINRES estimator, but this can be changed with the `fm` parameter.
estimator.cfaLoadings	Estimates all loadings simultaneously with using the `cfa` function of the **lavaan** package. Defaults to ML estimator, but this can be changed with the `estimator` parameter.

The consistency of the loading estimates produced by Dijkstra's technique (`estimator.plscLoadings`) requires that the weights are are asymptotically proportional to the loadings. With PLS Mode A weights this is the case. The more traditional factor analysis techniques, implemented in the `estimator.efaLoadings` and `estimator.cfaLoadings` do not depend on any particular set of weights and are therefore more general.

Setting the `disattenuate` parameter to `TRUE` enables disattenuation of the composite correlation matrix. The composite reliabilities required for disattenuation are calculated with the function provided as the `reliabilities` parameter. The default reliability function is a general function for calculating the reliability of a weighted composites, of which Dijkstra's reliability function is a special case. The estimated reliabilities are available as the `Q` vector after estimation.

The following code example demonstrates the use of disattenuation.

Example: PLSc and other dissattenuation techniques

```
# 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)

                       PLSc PLSc_2sls    DR GSCAc
Value~Support          0.96      0.96  0.96  0.95
Value~Advising        -0.01     -0.01 -0.03  0.00
Value~Tutoring        -0.02     -0.02 -0.01 -0.01
Satisfaction~Tutoring  0.26      0.23  0.26  0.22
Satisfaction~Value     0.62      0.67  0.61  0.68
Loyalty~Satisfaction   0.89      0.83  0.90  0.82
Support=~sup.help      0.77      0.77  0.83  0.77

(part of the output omitted)

# 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"))

             PLSc PLSc_2sls   DR GSCAc
Support      0.77      0.77 0.75  0.76
```

```
Advising     0.93       0.93 0.93  0.93
Tutoring     0.86       0.86 0.86  0.86
Value        0.91       0.91 0.91  0.91
Satisfaction 0.90       0.90 0.90  0.90
Loyalty      0.84       0.84 0.83  0.87
```

3. Postestimation tools

matrixpls package contains a collection of postestimation functions that can be applied to `matrixpls` objects produced by the `matrixpls` functions. These postestimation functions can be used to calculate predictions, model indices, and perform diagnostics on the results.

3.1. Predictions

The `predict` function can be used to calculate predictions based on the `matrixpls` object and a dataset. Although originally developed with a strong focus on prediction of indicators, the current partial least squares literature uses the term prediction for multiple different meanings (Shmueli et al. 2016). In **matrixpls**, the default technique for calculating predictions follows Wold's original approach (Wold 1985).

1. Composites that are exogenous in the `inner` model matrix are calculated as weighted sums of their indicators
2. The remaining composites are predicted using the equations in `inner` model matrix
3. The indicators are predicted using the equations in the `reflective` model matrix

Also two alternative prediction strategies, labeled as "redundancy" and "communality" by Chin (2010) are implemented.

The `predict` function then returns the matrix of indicators. The following example demonstrates calculating a model with a sample, calculating predictions from a hold-out samples, blindfolding, assessing prediction error, and calculating the Q^2 predictive relevance statistics.

Example: Predictions using the ECSI mobi dataset

```
library(semPLS)
data(ECSImobi)

# Construct the model based on the ECSImobi example

model <- list(inner = t(ECSImobi$D),
     reflective = ECSImobi$M,
     formative = t(ECSImobi$M))
model$formative[] <- 0


# Estimation using covariance matrix

matrixpls.out <- matrixpls(cov(mobi),
                           model = model,
                           standardize = FALSE)

# Calculate within-sample predictions

predictions <- predict(matrixpls.out, mobi)

# Calculate root mean squared prediction errors

sqrt(apply((predictions-mobi[61:75,])**2,2,mean))

CUEX1 CUEX2 CUEX3 CUSA1 CUSA2 CUSA3 CUSCO CUSL1 CUSL2 CUSL3 IMAG1 IMAG2
  1.8   2.2   2.1   1.6   2.6   1.7   2.3   3.8   3.4   3.0   1.5   1.6
IMAG3 IMAG4 IMAG5 PERQ1 PERQ2 PERQ3 PERQ4 PERQ5 PERQ6 PERQ7 PERV1 PERV2
  2.9   1.6   1.9   1.8   2.0   2.2   1.6   1.4   1.4   1.8   3.1   2.6

# Mimic the blindfolding procedure used in semPLS

predictions.blindfold <- matrixpls.crossvalidate(mobi,
                                                 model = model,
                                                 blindfold = TRUE,
                                                 predictionType = "redundancy",
                                                 groups = 4)

# Q2 predictive relevance

q2(mobi, predictions.blindfold, model)


 Q2 predictive relevance statistics

 Overall Q2
0.21
 Block Q2
      Image  Expectation      Quality        Value Satisfaction
       0.20         0.11         0.17         0.26         0.46
  Complaints      Loyalty
       0.26         0.17

 Indicator Q2
CUEX1 CUEX2 CUEX3 CUSA1 CUSA2 CUSA3 CUSCO CUSL1 CUSL2 CUSL3 IMAG1 IMAG2
 0.12  0.14  0.08  0.36  0.45  0.52  0.26  0.21  0.01  0.40  0.24  0.13
IMAG3 IMAG4 IMAG5 PERQ1 PERQ2 PERQ3 PERQ4 PERQ5 PERQ6 PERQ7 PERV1 PERV2
 0.12  0.31  0.20  0.27  0.10  0.18  0.15  0.16  0.20  0.17  0.22  0.33

# The results are similar to semPLS q2 values, but not exactly identical due to differences
# in how the two packages apply standardization when calculating predictions.

ecsi <- sempls(model=ECSImobi, data=mobi, E="C")

All 250 observations are valid.
Converged after 6 iterations.
Tolerance: 1e-07
Scheme: centroid
```

```
qSquared(ecsi, d=4, dlines = FALSE)

             Q-Squared
Image                .
Expectation       0.11
Quality           0.18
Value             0.26
Satisfaction      0.44
Complaints        0.26
Loyalty           0.21
```

3.2. Residual analysis

The `residuals` function calculates two kinds of residuals. The observed residuals are empirical residuals as presented by Lohmöller (1989, ch 2.4) or model implied residuals as used by Henseler et al. (2014). The model implied residuals are calculated by comparing the fitted covariance matrix, obtained with the `fitted` function, against the observed covariance matrix. `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.

Two versions of the SRMR index are provided for both types of residuals, 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.

Example: Calculating observed and implied residuals

```
residuals(political.out)


 Inner model (composite) residual covariance matrix
      dem60 dem65
dem60  0.84  0.66
dem65  0.66  0.22

 Outer model (indicator) residual covariance matrix
      x1    x2    x3    y1    y2    y3    y4    y5    y6    y7    y8
x1  0.09 -0.03 -0.08  0.04 -0.10  0.02  0.12  0.16 -0.07 -0.04  0.03
x2 -0.03  0.06 -0.04 -0.02 -0.07  0.00  0.09  0.11 -0.06 -0.03  0.01
x3 -0.08 -0.04  0.15 -0.08 -0.09 -0.07  0.06  0.04 -0.07 -0.06 -0.05
y1  0.04 -0.02 -0.08  0.22 -0.11 -0.02 -0.10  0.10  0.01  0.01 -0.02
y2 -0.10 -0.07 -0.09 -0.11  0.34 -0.20 -0.01 -0.05  0.11 -0.03 -0.03
y3  0.02  0.00 -0.07 -0.02 -0.20  0.37 -0.11  0.00 -0.15  0.05 -0.09
y4  0.12  0.09  0.06 -0.10 -0.01 -0.11  0.19  0.00  0.00  0.00  0.04
y5  0.16  0.11  0.04  0.10 -0.05  0.00  0.00  0.30 -0.14 -0.05 -0.12
y6 -0.07 -0.06 -0.07  0.01  0.11 -0.15  0.00 -0.14  0.29 -0.13  0.00
y7 -0.04 -0.03 -0.06  0.01 -0.03  0.05  0.00 -0.05 -0.13  0.24 -0.07
y8  0.03  0.01 -0.05 -0.02 -0.03 -0.09  0.04 -0.12  0.00 -0.07  0.19
```

```
 Residual-based fit indices
                              Value
Communality                    0.30
Redundancy                     0.27
SMC                            0.47
RMS outer residual covariance  0.08
RMS inner residual covariance  0.09
SRMR                           0.07
SRMR (Henseler)                0.07
```

residuals(political.out, observed = FALSE)

```
 Model implied residual covariance matrix
      x1    x2    x3    y1    y2    y3    y4    y5    y6    y7    y8
x1  0.00 -0.03 -0.08  0.04 -0.10  0.02  0.12  0.16 -0.07 -0.04  0.03
x2 -0.03  0.00 -0.04 -0.02 -0.07  0.00  0.09  0.11 -0.06 -0.03  0.01
x3 -0.08 -0.04  0.00 -0.08 -0.09 -0.07  0.06  0.04 -0.07 -0.06 -0.05
y1  0.04 -0.02 -0.08  0.00 -0.11 -0.02 -0.10  0.10  0.01  0.01 -0.02
y2 -0.10 -0.07 -0.09 -0.11  0.00 -0.20 -0.01 -0.05  0.11 -0.03 -0.03
y3  0.02  0.00 -0.07 -0.02 -0.20  0.00 -0.11  0.00 -0.15  0.05 -0.09
y4  0.12  0.09  0.06 -0.10 -0.01 -0.11  0.00  0.00  0.00  0.00  0.04
y5  0.16  0.11  0.04  0.10 -0.05  0.00  0.00  0.00 -0.14 -0.05 -0.12
y6 -0.07 -0.06 -0.07  0.01  0.11 -0.15  0.00 -0.14  0.00 -0.13  0.00
y7 -0.04 -0.03 -0.06  0.01 -0.03  0.05  0.00 -0.05 -0.13  0.00 -0.07
y8  0.03  0.01 -0.05 -0.02 -0.03 -0.09  0.04 -0.12  0.00 -0.07  0.00

 Residual-based fit indices
                Value
SRMR             0.07
SRMR (Henseler)  0.07
```

3.3. Commonly used model quality indices

matrixpls implements a number of model quality indices. Each index or set of indices is implemented as a separate function. The `summary` method of `matrixpls` object prints out a collection of these indicess.

Example: Calculating model quality indices

summary(political.out)

```
 matrixpls parameter estimates
            Est.
dem60~ind60 0.40
dem65~ind60 0.20
dem65~dem60 0.79
ind60=~x1   0.95
ind60=~x2   0.97
ind60=~x3   0.92
```

```
dem60=~y1   0.88

(part of the output omitted)

 matrixpls weights
       x1   x2   x3   y1   y2   y3   y4  y5   y6   y7   y8
ind60 0.38 0.37 0.31 0.00 0.00 0.00 0.00 0.0 0.00 0.00 0.00
dem60 0.00 0.00 0.00 0.31 0.27 0.26 0.33 0.0 0.00 0.00 0.00
dem65 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.3 0.28 0.29 0.29

Weight algorithm converged in 4 iterations.

 Total Effects (column on row)
      ind60 dem60
dem60  0.40  0.00
dem65  0.51  0.79

 Direct Effects
      ind60 dem60
dem60   0.4  0.00
dem65   0.2  0.79

 Indirect Effects
      ind60 dem60
dem60  0.00     0
dem65  0.32     0

 Inner model squared multiple correlations (R2)
ind60 dem60 dem65
 0.00  0.16  0.78

 Inner model (composite) residual covariance matrix
      dem60 dem65
dem60  0.84  0.66
dem65  0.66  0.22

 Outer model (indicator) residual covariance matrix
      x1    x2    x3    y1    y2    y3    y4    y5    y6    y7    y8
x1  0.09 -0.03 -0.08  0.04 -0.10  0.02  0.12  0.16 -0.07 -0.04  0.03
x2 -0.03  0.06 -0.04 -0.02 -0.07  0.00  0.09  0.11 -0.06 -0.03  0.01
x3 -0.08 -0.04  0.15 -0.08 -0.09 -0.07  0.06  0.04 -0.07 -0.06 -0.05
y1  0.04 -0.02 -0.08  0.22 -0.11 -0.02 -0.10  0.10  0.01  0.01 -0.02
y2 -0.10 -0.07 -0.09 -0.11  0.34 -0.20 -0.01 -0.05  0.11 -0.03 -0.03
y3  0.02  0.00 -0.07 -0.02 -0.20  0.37 -0.11  0.00 -0.15  0.05 -0.09
y4  0.12  0.09  0.06 -0.10 -0.01 -0.11  0.19  0.00  0.00  0.00  0.04
y5  0.16  0.11  0.04  0.10 -0.05  0.00  0.00  0.30 -0.14 -0.05 -0.12
y6 -0.07 -0.06 -0.07  0.01  0.11 -0.15  0.00 -0.14  0.29 -0.13  0.00
y7 -0.04 -0.03 -0.06  0.01 -0.03  0.05  0.00 -0.05 -0.13  0.24 -0.07
y8  0.03  0.01 -0.05 -0.02 -0.03 -0.09  0.04 -0.12  0.00 -0.07  0.19

 Residual-based fit indices
                              Value
Communality                    0.30
```

```
Redundancy                     0.27
SMC                            0.47
RMS outer residual covariance  0.08
RMS inner residual covariance  0.09
SRMR                           0.07
SRMR (Henseler)                0.07

 Absolute goodness of fit: 0.61

 Composite Reliability indices
ind60 dem60 dem65
 0.96  0.91  0.92

 Average Variance Extracted indices
ind60 dem60 dem65
 0.90  0.72  0.74

 AVE - largest squared correlation
 ind60  dem60  dem65
 0.636 -0.027 -0.004

 Heterotrait-monotrait matrix
      ind60 dem60 dem65
ind60  0.00
dem60  0.43  0.00
dem65  0.56  0.98  0.00
```

4. Bootstrapping

The variance of the PLS results is typically estimated with boostrapping. Bootstrapping is needed because the sampling distribution of PLS and other model-dependent indicator weights is unknown and therefore there is no closed form solutions to the standard errors of the parameter estimates.

matrixpls implements bootstrapping by integrating with the **boot** package (Canty and Ripley 2016; Davison and Hinkley 1997) with the `matrixpls.boot` function. The function takes the following arguments:

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

`data` is a data.frame or a matrix of raw data, from which `R` bootstrap samples are drawn. In addition to the data and the number of samples, the function requires also that `model` is specified. As with most functions in **matrixpls**, `matrixpls.boot` passes most of its arguments down to other functions and accepts any additional arguments used by `matrixpls` or `boot`. The arguments of the `boot` function allow parallel and ncpus can be used to enable parallel processing of bootstrap replications. By default, all bootstrap replications are calculated on a single processor core. Computational time can be reduced significantly by dividing the bootstrap replications on multiple processor cores. The only drawback in using parallel processing is that if any of the bootstrap replications produced an error, the full error message is discarded. Therefore, when troubleshooting an analysis, it is always a good idea to start by disabling parallel processing.

By default, only the parameter estimates are boostrapped, but boostrapping other statistics is possible with the extraFun argument.

Example: Boostrapping estimates

The following example demonstrates bootstrapping the estimates. Because the summary method prints out the full matrixpls summary, including all estimates and model quality indices, only the parts relevant to bootstrapping are shown.

```
set.seed(1)

boot.out <- matrixpls.boot(satisfaction, model = satisfaction.model, R = 500,
                           parallel = "multicore", ncpus = parallel::detectCores())

# Summary method prints confidence intervals and p values
summary(boot.out)
```

```
Calculating confidence intervals.


(part of the output omitted)

 Bootstrap SEs and significance tests
          Estimate   SE     t p (regression) p (Hair) p (Henseler) p (z)
EXPE~IMAG     0.56 0.05 11.60            0.0     0.00         0.00  0.00
SAT~IMAG      0.19 0.05  3.62            0.0     0.00         0.00  0.00
LOY~IMAG      0.29 0.07  4.22            0.0     0.00         0.00  0.00
QUAL~EXPE     0.85 0.02 41.86            0.0     0.00         0.00  0.00
VAL~EXPE      0.12 0.07  1.68            0.1     0.09         0.09  0.09
SAT~EXPE      0.01 0.07  0.13            0.9     0.90         0.90  0.90
VAL~QUAL      0.66 0.08  8.72            0.0     0.00         0.00  0.00

(part of the output omitted)

 Bootstrap confidence intervals
           Estimate         Norm         Basic          Perc          BCa
EXPE~IMAG      0.56  ( 0.46 0.65) ( 0.47  0.66) ( 0.46 0.65) ( 0.44 0.64)
SAT~IMAG       0.19  ( 0.08 0.28) ( 0.07  0.28) ( 0.10 0.30) ( 0.09 0.28)
LOY~IMAG       0.29  ( 0.14 0.41) ( 0.15  0.41) ( 0.17 0.43) ( 0.13 0.41)
QUAL~EXPE      0.85  ( 0.80 0.88) ( 0.81  0.89) ( 0.80 0.88) ( 0.80 0.88)
VAL~EXPE       0.12  (-0.02 0.25) (-0.03  0.25) (-0.02 0.27) (-0.03 0.25)
SAT~EXPE       0.01  (-0.12 0.14) (-0.12  0.13) (-0.12 0.13) (-0.12 0.13)
VAL~QUAL       0.66  ( 0.51 0.81) ( 0.51  0.83) ( 0.49 0.81) ( 0.49 0.81)

(part of the output omitted)
```

The first output block shows the parameter estimates, boostrap standard errors, t statistics (estimate/SE), and four different p values.

The significance of OLS regression coefficients, used as the default parameter estimation technique, is tested by the one-sample t test with n – k – 1 degrees of freedom, where n is the number of observations and k is the number of independent variables (Wooldridge 2009, sec. 4.2). However, introductory texts on PLS argue that the degrees of freedom should be n – 1 (Hair et al. 2014, 134) or n + m – 2, where m is always 1 and n is the number of bootstrap samples (Henseler, Ringle, and Sinkovics 2009, 305). The cited sources do not explain how these alternative references distributions are derived. Finally, p values can also be calculated with the z test. The four p values "regression", "Hair", "Henseler", and "z" refer to these four techniques in this order.

The second output block shows confidence intervals, calculated with the `boot.ci` function of the **boot** package.

Example: Boostrap-based model testing

The following example demonstrates how bootstrapping can be used to generate an empical reference distribution for a model test statistic for the education analysis presented earlier in the paper. The approach is based on the idea presented by Yuan and Hayashi (2003) and its adaptation to PLS by Dijkstra and Henseler (2015b)[1].

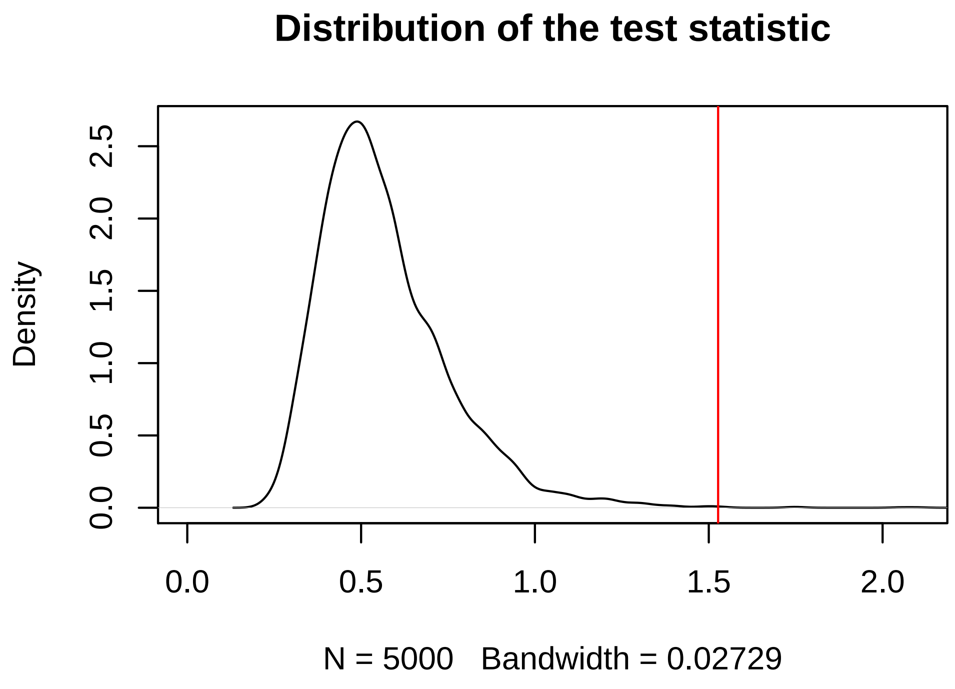
```
# Define a function that calculates the test statistic as the squared Euclidean distance
# between the empirical and the model implied correlation matrix

calculateTestStat <- function(matrixpls.res){
  S <- attr(matrixpls.res,"S")
  Sigma <- fitted(matrixpls.res)
  sum((S-Sigma)[lower.tri(S, diag = FALSE)]**2)
}

# Function that tranforms the data sets in the way proposed by Yuan & Hayashi (2003)

scaleDataSet <- function(data, Sigma){

  S <- cor(data)

  # Ensure that the variables are in the same order
  if(! identical(colnames(S), colnames(Sigma))) stop("data is inconsistent with Sigma")

  # singular value decomposition
  Ssvd=svd(S); Sigsvd=svd(Sigma)
  dS=Ssvd$d; dSig=Sigsvd$d
  uS=Ssvd$u; uSig=Sigsvd$u
  vS=Ssvd$v; vSig=Sigsvd$v
  #S**(-1/2)
  S_half=uS%*%(diag(dS**(-1/2)))%*%t(vS)
  # Sigma_hat**(1/2)
  Sig_half=uSig%*%(diag(dSig**(1/2)))%*%t(vSig)
  # scale factor Sig_hat**(1/2)%*%S**(-1/2)
```

[1]This example is based on code contributed by Florian Schuberth.

```
  sf=S_half%*%Sig_half

  sdata <- as.matrix(data)%*%sf
  colnames(sdata) <- colnames(data)
  sdata
}

# Calculate the empirical reference distribution

scaledEducation <- scaleDataSet(education[,2:24], fitted(education.out))

set.seed(1)

boot.out <- matrixpls.boot(scaledEducation,
                           model = education.model,
                           disattenuate = TRUE,
                           parametersReflective = estimator.plscLoadings,
                           R = 5000, parallel = "multicore", ncpus = parallel::detectCores(),
                           # Calculate the test statistic for each bootstrap replication
                           extraFun = calculateTestStat)

# The reference distribution is stored as the last entry in the boostrap results

ref <- boot.out$t[,ncol(boot.out$t)]
testStat <- calculateTestStat(education.out)

# Get the p value
sum(ref>testStat)/length(ref)
```

```
[1] 0.001
```

```
# Present the disribution and the value of the statistic graphically
plot(density(ref), xlim = c(0, max(c(ref,testStat))), main = "Distribution of the test statistic")
abline(v=testStat, col = "red")
```



In this example, the bootstrap-based model test conclusively rejects the model.

5. Monte Carlo simulations

matrixpls integratates with **simsem** (Pornprasertmanit, Miller, and Schoemann 2016) Monte Carlo simulation framework. Monte Carlo simulation involves drawing multiple independent samples of random variables from a population model chosen by the researcher, using each of these samples to estimate a model, and collecting the results (Boomsma 2013). After a sufficient number of replications have been generated, the results of the replications are analyzed, typically by calculating summary statistics, such as the bias or mean squared error, or by plotting the sampling distributions of the estimates. Monte Carlo simulations are commonly used to analyze the robustness of estimators under violation of their assumptions (e.g., small sample size, non-normal data), to compare estimators under different conditions, and to analyze procedures for which there are no theoretical results that could be used (Boomsma 2013).

Given that the sampling distribution of PLS weights remains unknown, most methodological research on the algorithm is based on simulations. However, simulations can be useful for applied researchers as well. Besides important for power analysis (Aguirre-Urreta and Rönkkö 2015), simulations can be useful in teaching or self-learning of statistical techniques (Antonakis et al. 2010).

Monte Carlo simulations are implemented with the `matrixpls.sim` function:

```
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)
```

The nRep argument defines the number of Monte Carlo replications. model is a **matrixpls** model specification, either in the native format or as **lavaan** syntax. If **lavaan** syntax is used, then that syntax can be used for defining the population model. alternatively, the generate argument defining the data generation model is required. This argument is not part of the function definition, but is passed through to sim function of **simsem** package.

The cilevel, citype, and boot.R define how bootstrapped confidence intervals are calculated. The outfundata, outfun, and prefun are functions that can be used for post-processing individual replications or pre-processing datasets. fitIndices a function that is used for calculating model quality indices after estimation.

matrixpls.sim integrates tightly with the sim function of **simsem** package. Monte Carlo simulation can be thought of consisting of four types of tasks: 1) controlling the simulation, 2) generating the datasets, 3) calculating statistics based on the datasets, and 4) summarizing the results. Out of these four tasks, only calculation of statistics is done by **matrixpls** and other three tasks are handled by **simsem**. Planning of simulation studies should therefore be started by studying the **simsem** package documentation.

5.1. Example: Power analysis

This example demonstrates power analysis by implementing the non-normal condition from the article by Aguirre-Urreta and Rönkkö (2015). The simulation runs 1000 replications with 500 bootstrap samples each for a total of 501 000 PLS analyses.

```
library(simsem)

model<-"! factor loadings
A=~0.7*x1 + 0.7*x2 + 0.7*x3
B=~0.7*x4 + 0.7*x5 + 0.8*x6 + 0.8*x7
C=~0.6*x8 + 0.6*x9 + 0.6*x10 + 0.8*x11 + 0.8*x12
D=~0.8*x13 + 0.8*x14 + 0.8*x15

!latent regressions
D ~ 0.3*A + 0.3*C
C ~ 0.1*B + 0.5*A

! error variances, variances and covariances
A ~~ 1.0*A
B ~~ 1.0*B
C ~~ 0.71*C
D ~~ 0.725*D
B ~~ 0.3*A
x1 ~~ 0.51*x1
x2 ~~ 0.51*x2
x3 ~~ 0.51*x3
x4 ~~ 0.51*x4
x5 ~~ 0.51*x5
x6 ~~ 0.36*x6
x7 ~~ 0.36*x7
```

```
x8 ~~ 0.64*x8
x9 ~~ 0.64*x9
x10 ~~ 0.64*x10
x11 ~~ 0.36*x11
x12 ~~ 0.36*x12
x13 ~~ 0.36*x13
x14 ~~ 0.36*x14
x15 ~~ 0.36*x15"


# Non-normal data

skewness <- c(0.50, 0.50, 0.50, 0.50, 0.50,
              0.50, 0.50, 0.50, 0.75, 0.75,
              0.75, 0.75, 0.75, 0.75, 0.75)

kurtosis <-  c(1, 1, 1, 1, 1,
               1, 1, 1, 1.50, 1.50,
               1.50, 1.50, 1.50, 1.50, 1.50)

# Simulate

nonNorm <- matrixpls.sim(1000, model,                         # Basic arguments
                         generate = list(model = model,       # Data generation
                                         skewness = skewness,
                                         kurtosis = kurtosis),
                         n=100,
                         multicore = TRUE,                    # Additional arguments
                         completeRep = TRUE)

Progress tracker is not available when 'multicore' is TRUE.

# Print the results
summary(nonNorm)

RESULT OBJECT
Model Type
[1] "function"
========= Fit Indices Cutoffs ============
           Alpha
Fit Indices  0.1 0.05 0.01 0.001 Mean   SD
       srmr 0.09 0.09 0.14  0.16 0.08 0.01
========= Parameter Estimates and Standard Errors ============
       Est. Avg Est. SD Avg SE Power Param  Bias Coverage
C~A        0.40    0.09   0.08  0.99   0.5 -0.10     0.75
D~A        0.26    0.10   0.10  0.70   0.3 -0.04     0.92
C~B        0.14    0.09   0.11  0.31   0.1  0.04     0.93
D~C        0.28    0.10   0.10  0.78   0.3 -0.02     0.92
A=~x1      0.81    0.05   0.05  1.00   0.7  0.11     0.38
A=~x2      0.81    0.05   0.06  1.00   0.7  0.11     0.43
A=~x3      0.81    0.05   0.06  1.00   0.7  0.11     0.43
B=~x4      0.75    0.15   0.17  0.94   0.7  0.05     0.88
B=~x5      0.76    0.14   0.18  0.94   0.7  0.06     0.87
```

```
B=~x6       0.82    0.12   0.17  0.95   0.8  0.02     0.92
B=~x7       0.82    0.13   0.17  0.96   0.8  0.02     0.92
C=~x8       0.69    0.07   0.07  1.00   0.6  0.09     0.64
C=~x9       0.69    0.07   0.07  1.00   0.6  0.09     0.63
C=~x10      0.69    0.07   0.07  1.00   0.6  0.09     0.62
C=~x11      0.83    0.04   0.04  1.00   0.8  0.03     0.71
C=~x12      0.84    0.03   0.04  1.00   0.8  0.04     0.71
D=~x13      0.87    0.03   0.04  1.00   0.8  0.07     0.43
D=~x14      0.87    0.03   0.04  1.00   0.8  0.07     0.40
D=~x15      0.87    0.03   0.04  1.00   0.8  0.07     0.40
================= Replications ====================
Number of replications = 1000
Number of converged replications = 1000
Number of nonconverged replications:
   1. Nonconvergent Results = 0
   2. Nonconvergent results from multiple imputation = 0
   3. At least one SE were negative or NA = 0
   4. At least one variance estimates were negative = 0
   5. At least one correlation estimates were greater than 1 or less than -1 = 0
   6. Model-implied covariance matrices of any groups of latent variables are not
positive definite = 0
```

The timings of the simulation demonstrate what kind of computation times can be expected for this kind of analysis.

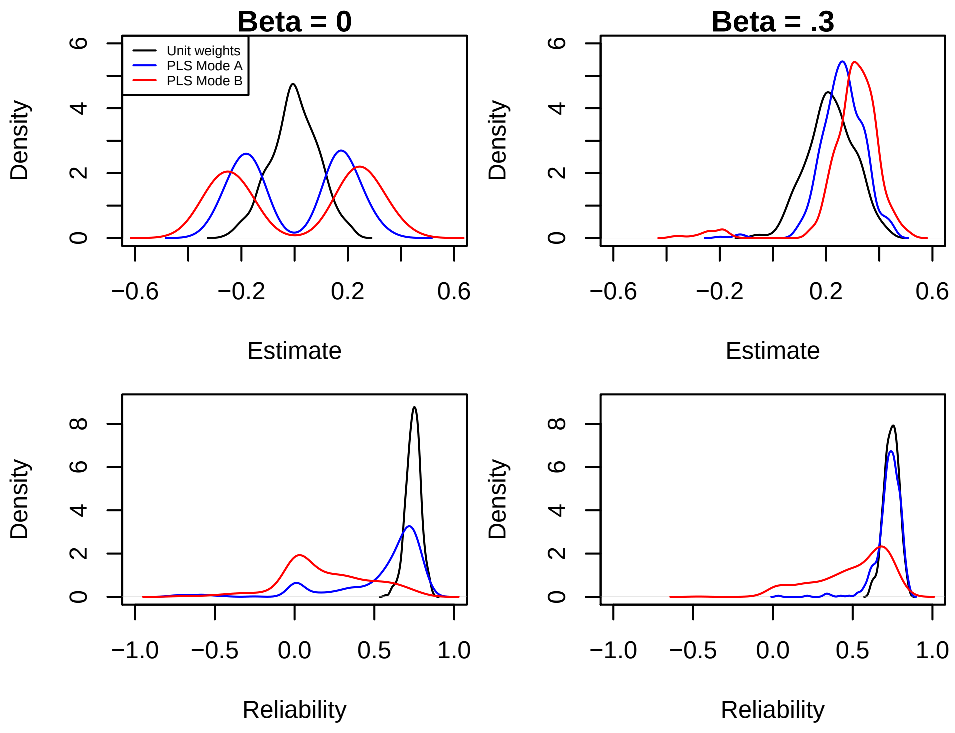
```
summaryTime(nonNorm)

============ Wall Time ============
  1. Error Checking and setting up data-generation and analysis template:       0.001
  2. Set combinations of n, pmMCAR, and pmMAR:                                  0.000
  3. Setting up simulation conditions for each replication:                     0.073
  4. Total time elapsed running all replications:                            1003.993
  5. Combining outputs from different replications:                             0.163

============ Average Time in Each Replication ============
  1. Data Generation:                                     0.138
  2. Impose Missing Values:                               0.000
  3. User-defined Data-Transformation Function:           0.000
  4. Main Data Analysis:                                  7.854
  5. Extracting Outputs:                                  0.000

============ Summary ============
  Start Time:                     2016-06-11 23:08:28
  End Time:                       2016-06-11 23:25:12
  Wall (Actual) Time:               1004.230
  System (Processors) Time:         7993.112
  Units:                          seconds
```

5.2. Example: True reliabilities and distribution of estimates

The following example replicates a simulation study presented by Rönkkö and Evermann (2013). The example demonstrates comparing the sampling distribution of reliability of a composite and its correlation with another composite over three different indicator weighting techniques: PLS Mode A, PLS Mode B, and unit weights.

```
library(simsem)

# Specify the data generation model using SimSem syntax

# Loadings
LY <- bind(matrix(c(.6, .7, .8, 0, 0, 0,
                  0, 0, 0, .6, .7, .8), 6,2))

# Factor correlations (2 different values)
RPS1 <- binds(matrix(c(1, 0, 0, 1), 2, 2))
RPS2 <- binds(matrix(c(1, .3, .3, 1), 2, 2))

# Error terms
RTE <- binds(diag(c(.64, .51, .36, .64, .51, .36)))

generateModels <- list(model.cfa(LY = LY, RPS = RPS1, RTE = RTE),
                       model.cfa(LY = LY, RPS = RPS2, RTE = RTE))

# Specify the computed model with lavaan syntax
analyzeModel <- "f1=~ y1 + y2 + y3
                f2=~ y4 + y5 + y6
                f2~f1"


coefficients <- list()
reliabilities <- list()

# We have 3 weight techniques and 2 populations: 6 conditions

for(i in 1:6){
  population <- rep(1:2, each=3)[i]
  technique <-  rep(1:3, 2)[i]
  out <- matrixpls.sim(nRep = 500, model = analyzeModel,
                       weightFun = ifelse(technique == 1, weightFun.fixed, weightFun.pls),
                       # Use Mode B for the third estimator, otherwise use Mode A
                       # This argument is ignored by fixed weights
                       outerEstim = ifelse(technique == 3, outerEstim.modeB, outerEstim.modeA),
                       n = 100, generate = generateModels[[population]],
                       sequential = TRUE, saveLatentVar = TRUE,
                       multicore = TRUE, boot.R = FALSE, fitIndices = NULL)

  coefficients[[i]] <- out@coef[,1]
  # Extract the reliability of the first composite from the matrixpls
  # results objects stored as extra output
  reliabilities[[i]] <- unlist(lapply(out@extraOut, function(x) attr(x,"R")[1]))
}

par(mfrow = c(2,2))
par(mar = c(4,4,1,0.5))
```

```
plot(density(coefficients[[1]]), main = "Beta = 0", xlab = "Estimate",
     xlim = c(-.6,.6), ylim = c(0,6))
lines(density(coefficients[[2]]), col = "blue")
lines(density(coefficients[[3]]), col = "red")
legend("topleft", c("Unit weights", "PLS Mode A", "PLS Mode B"),
       col=c("black","blue","red"), lty = 1, cex=0.5)

plot(density(coefficients[[4]]), main = "Beta = .3", xlab = "Estimate",
     xlim = c(-.6,.6), ylim = c(0,6))
lines(density(coefficients[[5]]), col = "blue")
lines(density(coefficients[[6]]), col = "red")

plot(density(reliabilities[[1]]), main = "", xlab = "Reliability",
     xlim = c(-1,1), ylim = c(0,9))
lines(density(reliabilities[[2]]), col = "blue")
lines(density(reliabilities[[3]]), col = "red")

plot(density(reliabilities[[4]]), main = "", xlab = "Reliability",
     xlim = c(-1,1), ylim = c(0,9))
lines(density(reliabilities[[5]]), col = "blue")
lines(density(reliabilities[[6]]), col = "red")
```



6. Technical details

As explained in Section 1.1, **matrixpls** works by applying a series of functions on matrices containing covariances, weights, and model specifications. The matrices are listed in Table 9 and functions in Table 8. The package can be extended by using user written functions in place of any of the built-in functions. Reference manual provides nore details on the built-in function implementations.

Table 8: Types of funcions used by **matrixpls**

Function	Description
`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`.
`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.
`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 with the same dimensions as `modelMatrix`.
`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`.
`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`.
`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`.
`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.
`optimCrit`	A function for calculating value for an optimization criterion based on a `matrixpls` result object. Returns a scalar.
`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.

Table 9: Matrices and other objects used and returned by **matrixpls** functions

Matrix or attribute	Matrix order	Description
`S`	k x k	the sample covariance matrix
`reflective`	k x l	the `reflective` model matrix with estimated parameters
`c`	l	the PLSc loading estimate correction factors
`Q`	l	the reliability estimates used in dissattenuation
`R`	l	true reliabilities of composites (squared correlation between composites and simulated latent variable values)
`formative`	l x k	the `formative` model matrix with estimated parameters
`IC`	l x k	the indicator-composite covariance matrix (after disattenuation, if requested)
`W`	l x k	the weight matrix
`C`	l x l	the composite correlation matrix (after disattenuation, if requested)
`E`	l x l	inner weight matrix
`inner`	l x l	the `inner` model matrix with estimated parameters
`call`		the function call
`converged`		`TRUE` if the weight algorithm converged and `FALSE` otherwise
`history`		weight optimization history as a matrix
`iterations`		the number of iterations used by the weight algorithm
`model`		the model specification in native format

`k` is the number of observed variables and `l` is the number of composites.

References

Aguirre-Urreta, Miguel I., and Mikko Rönkkö. 2015. "Sample Size Determination and Statistical Power Analysis in PLS Using R: An Annotated Tutorial." *Communications of the Association for Information Systems* 36 (1). `http://aisel.aisnet.org/cais/vol36/iss1/3`.

Antonakis, John, Samuel Bendahan, Philippe Jacquart, and Rafael Lalive. 2010. "On Making Causal Claims: A Review and Recommendations." *The Leadership Quarterly* 21 (6): 1086–1120. doi:10.1016/j.leaqua.2010.10.010.

Bentler, Peter M., and Wenjing Huang. 2014. "On Components, Latent Variables, PLS and Simple Methods: Reactions to Rigdon's Rethinking of PLS." *Long Range Planning*, Rethinking Partial Least Squares Path Modeling: Looking Back and Moving Forward, 47 (3): 138–45. doi:10.1016/j.lrp.2014.02.005.

Bobko, Philip, Philip L. Roth, and Maury A. Buster. 2007. "The Usefulness of Unit Weights in Creating Composite Scores: A Literature Review, Application to Content Validity, and Meta-Analysis." *Organizational Research Methods* 10 (4): 689–709. doi:10.1177/1094428106294734.

Boomsma, Anne. 2013. "Reporting Monte Carlo Studies in Structural Equation Modeling." *Structural Equation Modeling: A Multidisciplinary Journal* 20 (3): 518–40. doi:10.1080/10705511.2013.79783

Canty, Angelo, and B. D. Ripley. 2016. *Boot: Bootstrap R (S-Plus) Functions*.

Chin, Wynne W. 1998. "The Partial Least Squares Approach to Structural Equation Modeling." In *Modern Methods for Business Research*, edited by George A. Marcoulides, 295–336. Mahwah, NJ: Lawrence Erlbaum Associates Publishers.

———. 2010. "How to Write up and Report PLS Analyses." In *Handbook of Partial Least Squares*, edited by Vincenzo Esposito Vinzi, Wynne W. Chin, Jörg Henseler, and Huiwen Wang, 655–90. Berlin Heidelberg: Springer.

Cohen, Jacob. 1990. "Things I Have Learned (so Far)." *American Psychologist* 45 (12): 1304–12. `http://psycnet.apa.org/psycinfo/1991-11596-001`.

Davison, A. C., and D. V. Hinkley. 1997. *Bootstrap Methods and Their Applications*. Cambridge: Cambridge University Press. `http://statwww.epfl.ch/davison/BMA/`.

Dijkstra, Theo K. 1983. "Some Comments on Maximum Likelihood and Partial Least Squares Methods." *Journal of Econometrics* 22 (1-2): 67–90.

———. 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." `http://www.rug.nl/staff/t.k.dijkstra/consistent-pls-estimators.pdf`.

Dijkstra, Theo K., and Jörg Henseler. 2015a. "Consistent Partial Least Squares Path Modeling." *MIS Quarterly* 39 (2): 297–316.

———. 2015b. "Consistent and Asymptotically Normal PLS Estimators for Linear Structural Equations." *Computational Statistics & Data Analysis* 81 (January): 10–23. doi:10.1016/j.csda.2014.07.008.

Evermann, Joerg, and Mary Tate. 2013. "Is My Model Right? Model Quality and Model Misspecification in PLS – Recommendations for IS Research." Social Science Research Network. `http://papers.ssrn.com/abstract=2337699`.

Fornell, Claes, and Fred L. Bookstein. 1982. "Two Structural Equation Models: LISREL and PLS Applied to Consumer Exit-Voice Theory." *Journal of Marketing Research* 19 (4): 440–52.

Gefen, David, E. E Rigdon, and Detmar W. Straub. 2011. "An Update and Extension to SEM

Guidelines for Administrative and Social Science Research." *MIS Quarterly* 35 (2): iii–xiv.

Goodhue, Dale L., William Lewis, and Ron Thompson. 2012. "Does PLS Have Advantages for Small Sample Size or Non-Normal Data." *MIS Quarterly* 36 (3): 981–1001.

Goodhue, Dale L., Ron Thompson, and William Lewis. 2013. "Why You Shouldn't Use PLS: Four Reasons to Be Uneasy About Using PLS in Analyzing Path Models." In *2013 46th Hawaii International Conference on System Sciences (HICSS)*, 4739–48. doi:10.1109/HICSS.2013.612.

Hair, Joseph F., and Christian M. Ringle. 2011. "The Use of Partial Least Squares (PLS) to Address Marketing Management Topics: From the Special Issue Guest Editors." *Journal of Marketing Theory and Practice* 19: 135–38.

Hair, Joseph F., G. Tomas M. Hult, Christian M. Ringle, and Marko Sarstedt. 2014. *A Primer on Partial Least Squares Structural Equations Modeling (PLS-SEM)*. Los Angeles, CA: SAGE Publications.

Hair, Joseph F., Christian M. Ringle, and Marko Sarstedt. 2012. "Partial Least Squares: The Better Approach to Structural Equation Modeling?" *Long Range Planning* 45 (5–6): 312–19. doi:10.1016/j.lrp.2012.09.011.

Hair, Joseph F., Marko Sarstedt, Torsten M. Pieper, and Christian M. Ringle. 2012. "The Use of Partial Least Squares Structural Equation Modeling in Strategic Management Research: A Review of Past Practices and Recommendations for Future Applications." *Long Range Planning* 45 (5-6): 320–40. doi:10.1016/j.lrp.2012.09.008.

Hair, Joseph F., Marko Sarstedt, Christian M. Ringle, and Jeannette A. Mena. 2012. "An Assessment of the Use of Partial Least Squares Structural Equation Modeling in Marketing Research." *Journal of the Academy of Marketing Science* 40 (3): 414–33. doi:10.1007/s11747-011-0261-6.

Henseler, Jörg. 2010. "A Comparative Study on Parameter Recovery of Three Approaches to Structural Equation Modeling: A Rejoinder." SSRN Scholarly Paper ID 1585305. Rochester, NY: Social Science Research Network. `http://papers.ssrn.com/abstract=1585305`.

———. 2012. "Why Generalized Structured Component Analysis Is Not Universally Preferable to Structural Equation Modeling." *Journal of the Academy of Marketing Science* 40 (3): 402–13. doi:10.1007/s11747-011-0298-6.

Henseler, Jörg, Theo K. Dijkstra, Marko Sarstedt, Christian M. Ringle, Adamantios Diamantopoulos, Detmar W. Straub, David J. Ketchen, Joseph F. Hair, G. Tomas M. Hult, and Roger J. Calantone. 2014. "Common Beliefs and Reality About PLS: Comments on Rönkkö and Evermann (2013)." *Organizational Research Methods* 17 (2): 182–209. doi:10.1177/1094428114526928.

Henseler, Jörg, Christian M. Ringle, and R. R Sinkovics. 2009. "The Use of Partial Least Squares Path Modeling in International Marketing." *Advances in International Marketing* 20: 277–319.

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

Hulland, J. 1999. "Use of Partial Least Squares (PLS) in Strategic Management Research: A Review of Four Recent Studies." *Strategic Management Journal* 20 (2): 195–204.

Hwang, H., N. K. Malhotra, and Y. Kim. 2010. "Web Errata for ,'a Comparative Study on Parameter Recovery of Three Approaches to Structural Equation Modeling (Hwang, Naresh, Kim, Tomiuk, & Hong, 2010, Vol. XLVII, August, Pp. 699-712.'" `https://www.ama.org/`

publications/JournalOfMarketingResearch/Documents/erratum_comparative_study_on_parameter_recovery.pdf.

Hwang, Heungsun, and Yoshio Takane. 2004. "Generalized Structured Component Analysis." *Psychometrika* 69 (1): 81–99. doi:10.1007/BF02295841.

———. 2014. *Generalized Structured Component Analysis: A Component-Based Approach to Structural Equation Modeling.* CRC Press.

Hwang, Heungsun, Naresh K Malhotra, Youngchan Kim, Marc A Tomiuk, and Sungjin Hong. 2010. "A Comparative Study on Parameter Recovery of Three Approaches to Structural Equation Modeling." *Journal of Marketing Research (JMR)* 47 (4): 699–712. doi:10.1509/jmkr.47.4.699.

Lohmöller, Jan-Bernd. 1989. *Latent Variable Path Modeling with Partial Least Squares.* Heidelberg: Physica-Verlag.

Monecke, Armin, and Friedrich Leisch. 2012. "semPLS: Structural Equation Modeling Using Partial Least Squares." *Journal of Statistical Software* 48 (3): 1–32. http://www.jstatsoft.org/v48/i03/.

Peng, David Xiaosong, and Fujun Lai. 2012. "Using Partial Least Squares in Operations Management Research: A Practical Guideline and Summary of Past Research." *Journal of Operations Management* 30 (6): 467–80. doi:10.1016/j.jom.2012.06.002.

Pornprasertmanit, Sunthud, Patrick Miller, and Alexander Schoemann. 2016. *Simsem: SIMulated Structural Equation Modeling.* https://CRAN.R-project.org/package=simsem.

Raju, Nambury S., Reyhan Bilgic, Jack E. Edwards, and Paul F. Fleer. 1999. "Accuracy of Population Validity and Cross-Validity Estimation: An Empirical Comparison of Formula-Based, Traditional Empirical, and Equal Weights Procedures." *Applied Psychological Measurement* 23 (2): 99–115. http://apm.sagepub.com/content/23/2/99.short.

Revelle, William. 2015. "Psych: Procedures for Psychological, Psychometric, and Personality Research." http://cran.r-project.org/web/packages/psych/index.html.

Ringle, Christian M., Marko Sarstedt, and Detmar W. Straub. 2012. "A Critical Look at the Use of PLS-SEM in MIS Quarterly." *MIS Quarterly* 36 (1): iiv–8.

Romdhani, Hela, Stepan Grinek, Heungsun Hwang, and Aurelie Labbe. 2014. *ASGSCA: Association Studies for Multiple SNPs and Multiple Traits Using Generalized Structured Equation Models.*

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

Rönkkö, Mikko. 2014. "The Effects of Chance Correlations on Partial Least Squares Path Modeling." *Organizational Research Methods* 17 (2): 164–81. doi:10.1177/1094428114525667.

Rönkkö, Mikko, and Joerg Evermann. 2013. "A Critical Examination of Common Beliefs About Partial Least Squares Path Modeling." *Organizational Research Methods* 16 (3): 425–48. doi:10.1177/1094428112474693.

Rönkkö, Mikko, and Jukka Ylitalo. 2010. "Construct Validity in Partial Least Squares Path Modeling." In *ICIS 2010 Proceedings.* http://aisel.aisnet.org/icis2010_submissions/155.

Rönkkö, Mikko, Cameron N. McIntosh, and John Antonakis. 2015. "On the Adoption of Partial Least Squares in Psychological Research: Caveat Emptor." *Personality and Individual Differences* forthcoming. https://www.researchgate.net/publication/280083330_

On_the_Adoption_of_Partial_Least_Squares_in_Psychological_Research_Caveat_Emptor.

Sanchez, Gaston, Laura Trinchera, and Giorgio Russolillo. 2015. *Plspm: Tools for Partial Least Squares Path Modeling (PLS-PM)*. http://CRAN.R-project.org/package=plspm.

Shmueli, Galit, Soumya Ray, Juan Manuel Velasquez Estrada, and Suneel Babu Chatla. 2016. "The Elephant in the Room: Predictive Performance of PLS Models." *Journal of Business Research*. doi:10.1016/j.jbusres.2016.03.049.

Willaby, Harold W., Daniel S. J. Costa, Bruce D. Burns, Carolyn MacCann, and Richard D. Roberts. 2015. "Testing Complex Models with Small Sample Sizes: A Historical Overview and Empirical Demonstration of What Partial Least Squares (PLS) Can Offer Differential Psychology." *Personality and Individual Differences*, Theory and Measurement in Personality and Individual Differences, 84 (October): 73–78. doi:10.1016/j.paid.2014.09.008.

Wold, Herman. 1966. "Nonlinear Estimation by Iterative Least Squares Procedures." In *Research Papers in Statistics: Festschrift for J. Neyman*, edited by F. N. David, 411–44. London: Wiley.

———. 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.

———. 1980. "Model Construction and Evaluation When Theoretical Knowledge Is Scarce." In *Model Evaluation in Econometrics*, edited by J. Kmenta and J. B. Ramsey, 47–74. New York, NY: Academic Press.

———. 1982. "Soft Modeling: The Basic Design and Some Extensions." In *Systems Under Indirect Observation : Causality, Structure, Prediction*, edited by K. G. Jöreskog and Svante Wold, 1–54. Amsterdam: North-Holland.

———. 1985. "Systems Analysis by Partial Least Squares." In *Measuring the Unmeasurable*, edited by P. Nijkamp, L Leitner, and N Wrigley, 221–52. Dordrecht, Germany: Marinus Nijhoff.

Wooldridge, Jeffrey M. 2009. *Introductory Econometrics: A Modern Approach*. 4th ed. Mason, OH: South Western, Cengage Learning.

Yuan, Ke-Hai, and Kentaro Hayashi. 2003. "Bootstrap Approach to Inference and Power Analysis Based on Three Test Statistics for Covariance Structure Models." *British Journal of Mathematical and Statistical Psychology* 56 (1): 93–110. doi:10.1348/000711003321645368.

Affiliation:

Mikko Rönkkö
Aalto University
Aalto University, School of Science Department of Industrial Management, Institute of Strategy and Venturing PO Box 15500, FI-00076 Aalto, Finland
E-mail: mikko.ronkko@aalto.fi