Package 'lslx'

July 22, 2025

Type Package

Title Semi-Confirmatory Structural Equation Modeling via Penalized Likelihood or Least Squares

Version 0.6.11

Description
Fits semi-confirmatory structural equation modeling (SEM) via penalized likelihood (PL) or penalized least squares (PLS). For details, please see Huang (2020) <doi:10.18637/jss.v093.i07>.

License GPL-3

Encoding UTF-8

Collate lslxData-class.R lslxData-initialize-method.R
lslxModel-class.R lslxModel-initialize-method.R
lslxFitting-class.R lslxFitting-initialize-method.R
prelslx-class.R prelslx-initialize-method.R
prelslx-prefit-method.R lslx-class.R lslx-print-method.R
lslx-set-coefficient-method.R lslx-set-block-method.R
lslx-set-directed-method.R lslx-set-heterogeneity-method.R
lslx-set-undirected-method.R lslx-set-data-method.R
RcppExports.R lslx-fit-method.R lslx-get-method.R
lslx-extract-method.R lslx-test-method.R
lslx-summarize-method.R lslx-plot-method.R
lslx-validate-method.R lslx-s3-interface.R polyhedral.R
utility-function.R

Depends R (>= 3.5.0)

Imports stats, Rcpp, R6, ggplot2

Suggests knitr, rmarkdown, lavaan

VignetteBuilder knitr

LinkingTo Rcpp, RcppEigen

RoxygenNote 7.2.2

URL https://github.com/psyphh/lslx/wiki

BugReports https://github.com/psyphh/lslx/issues

NeedsCompilation yes

Author Po-Hsien Huang [cre, aut],
Wen-Hsin Hu [aut]

Maintainer Po-Hsien Huang <psyphh@gmail.com>

Repository CRAN

Date/Publication 2022-12-02 08:20:05 UTC

Contents

`coef.lslx` *S3 method to extract parameter estimate from* `lslx`

Description

`coef.lslx()` is an `S3` interface for extracting parameter estimate from a `lslx` object.

Usage

```
## S3 method for class 'lslx'
coef(object, selector, lambda, delta, ...)
```

Arguments

`object`	A fitted `lslx` object.
`selector`	A `character` to specify a selector for determining an optimal penalty level.
`lambda`	A `numeric` to specify a chosen optimal lambda value.
`delta`	A `numeric` to specify a chosen optimal lambda value.
`...`	Other arguments. For details, please see the `$extracted_coefficient()` method in `lslx`.

fitted.lslx *S3 method to extract model-implied moments from* lslx

Description

fitted.lslx() is an S3 interface for extracting model-implied moments from a lslx object.

Usage

```
## S3 method for class 'lslx'
fitted(object, selector, lambda, delta, ...)
```

Arguments

object	A fitted lslx object.
selector	A character to specify a selector for determining an optimal penalty level. Its value can be any one in "aic", "aic3", "caic", "bic", "abic", "hbic", or their robust counterparts "raic", "raic3", "rcaic", "rbic", "rabic", "rhbic" if raw data is available.
lambda	A numeric to specific a chosen optimal penalty level. If the specified lambda is not in lambda_grid, a nearest legitimate value will be used.
delta	A numeric to specific a chosen optimal convexity level. If the specified delta is not in delta_grid, a nearest legitimate value will be used.
...	Other arguments. For details, please see the $extracted_implied_mean() and the $extracted_implied_cov() methods in lslx.

lslx *R6 class for semi-confirmatory structural equation modeling via penalized likelihood*

Description

R6 class for semi-confirmatory structural equation modeling via penalized likelihood

R6 class for semi-confirmatory structural equation modeling via penalized likelihood

Value

Object of lslx R6 class for fitting semi-confirmatory structural equation modeling (SEM) with penalized likelihood (PL).

Usage

`lslx` is an `R6ClassGenerator` for constructing an `lslx` object that has methods for fitting semi-confirmatory SEM. In a simpliest case, the use of `lslx` involves three major steps

1. Initialize a new `lslx` object by specifying a model and importing a data set.
   `r6_lslx <- lslx$new(model, data)`
2. Fit the specified model to the imported data with given fitting control.
   `r6_lslx$fit(penalty_method, lambda_grid, delta_grid)`
3. Summarize the fitting results with specified selector.
   `r6_lslx$summarize(selector)`

To cite **lslx** in publications use:

Po-Hsien Huang (in press). lslx: Semi-Confirmatory Structural Equation Modeling via Penalized Likelihood. Journal of Statistical Software.

Overview

lslx is a package for fitting semi-confirmatory structural equation modeling (SEM) via penalized likelihood (PL) developed by Huang, Chen, and Weng (2017). In this semi-confirmatory method, an SEM model is distinguished into two parts: a confirmatory part and an exploratory part. The confirmatory part includes all of the freely estimated parameters and fixed parameters that are allowed for theory testing. The exploratory part is composed by a set of penalized parameters describing relationships that cannot be clearly determined by available substantive theory. By implementing a sparsity-inducing penalty and choosing an optimal penalty level, the relationships in the exploratory part can be efficiently identified by the sparsity pattern of these penalized parameters. After Version 0.6.7, **lslx** also supports penalized least squares for SEM with ordianl data under delta parameterization. The technical details of **lslx** can be found in its JSS paper (Huang, 2020) <doi:10.18637/jss.v093.i07> or Vignette for Package lslx (https://cran.r-project.org/web/packages/lslx/vignettes/vignette-lslx.pdf).

The main function `lslx` generates an object of `lslx` R6 class. R6 class is established via package **R6** (Chang, 2017) that facilitates encapsulation object-oriented programming in **R** system. Hence, the `lslx` object is self-contained. On the one hand, `lslx` object stores model, data, and fitting results. On the other hand, it has many built-in methods to respecify model, fit the model to data, and test goodness of fit and coefficients. The initialization of a new `lslx` object requires importing a model and a data set to be analyzed. After an `lslx` object is initialized, build-in methods can be used to modify the object, find the estimates, and summarize fitting result. Details of object initialization is described in the section of *Initialize Method*.

In the current semi-confirmatory approach, the model specification is quite similar to the traditional practice of SEM except that some parameters can be set as penalized. Model specification in `lslx` mainly relies on the argument model when creating a new `lslx` object. After a `lslx` object is initialized, the initialized model can be still modified by set-related methods. These set-related methods may hugely change the initialized model by just one simple command. This two-step approach allows users specifying their own models flexibly and efficiently. Details of the model specification can be found in the sections of *Model Syntax* and *Set-Related Methods*.

Given a penalty level, **lslx** finds a PL estimate by minimizing a penalized maximum likelihood (ML) loss or a least squares loss functions (including OLS, DWLS, and WLS). The penalty function

can be set as lasso (Tibshirani, 1996), ridge (Hoerl & Kennard, 1970), elastic net (Zou & Hastie, 2005), or mcp (minimax concave penalty; Zhang, 2010). **lslx** solves the optimization problem based on an improved **glmnet** method (Friedman, Hastie, & Tibshirani, 2010) made by Yuan, Ho, and Lin (2012). The underlying optimizer is written by using **Rcpp** (Eddelbuettel & Francois, 2011) and **RcppEigen** (Bates & Eddelbuettel, 2013). Our experiences show that the algorithm can efficiently find a local minimum provided that (1) the starting value is reasonable, and (2) the saturated covariance matrix is not nearly singular. Details of optimization algorithm and how to implement the algorithm can be found in the sections of *Optimization Algorithm* and *Fit-Related Methods*.

When conducting SEM, missing data are easily encountered. **lslx** can handle missing data problem by listwise deletion and two-step methods. Details of the methods for missing data can be found in the section of *Missing Data*.

After fitting the specified model to data under all of the considered penalty levels, an optimal penalty level should be chosen. A naive method for penalty level selection is using information criteria. Huang, Chen, and Weng (2017) have shown the asymptotic properties of Akaike information criterion (AIC) and Bayesian information criterion (BIC) in selecting the penalty level. In **lslx**, information criteria other an AIC and BIC can be also used. However, the empirical performances of these included criteria should be further studied. Details of choosing an optimal penalty level can be found in the section of *Penalty Level Selection*.

Given a penalty level, it is important to evaluate the goodness-of-fit of selected model and coefficients. In **lslx**, it is possible to make statistical inferences for goodness-of-fit and coefficients. However, the inference methods assume that no model selection is conducted, which is not true in the case of using PL. After version 0.6.4, several post-selection mehtods are available (Huang, 2019b). Details of statistical inference can be found in the sections of *Model Fit Evaluation* and *Coefficient Evaluation*. Implementations of these methods can be found in the sections of *Summarize Method* and *Test-Related Methods*.

Besides making statistical inference, `lslx` has methods for plotting the fitting results, include visualizing quality of optimization and the values of information criteria, fit indices, and coefficient estimates. Details of methods for plotting can be found in the section of *Plot-Related Methods*.

An object of `lslx` R6 class is composed by three R6 class objects: `lslxModel`, `lslxData`, and `lslxFitting`. `lslxModel` contains the specified model and `lslxData` object stores the imported data to be analyzed. When fitting the model to data, a reduced model and data will be sent to `lslxFitting`. After the underlying optimizer finishes its job, the fitting results will be also stored in `lslxFitting`. Since the three members are set as private, they can be only assessed by defined member functions. Other than the three members, quantities that are crucial for SEM can be also extracted, such as model-implied moments, information matrix, and etc.. Details of methods for obtaining private members and SEM-related quantities can be found in the sections of *Get-Related Methods* and *Extract-Related Methods*.

Model Syntax

With **lslx** the relationships among observed variables and latent factors are mainly specified via equation-like syntax. The creation of syntax in **lslx** is highly motivated by **lavaan** (Rosseel, 2012), a successful package for fitting SEM. However, **lslx** utilizes slightly more complex, but still intuitive operators to describe relations among variables.

Example 1: Multiple Regression Model

Consider the first example of model that specifies a multiple regression model

```
y <= x1 + x2
```

In this example, an dependent variable y is predicted by x1 and x2. These three variables are all observed variables and should appear in the given data set. The operator <= means that the regression coefficients from the right-hand side (RHS) variables to the left-hand side (LHS) variables should be freely estimated.

Although it is a very simple example, at least four important things behind this example should be addressed:

1. For any endogenous variable (i.e., an variable that is influenced by any other variable), its residual term is not required to be specified. In this example, `lslx` recognizes y is an endogenous variable and hence the variance of corresponding residual will be set as freely estimated parameter. It is also possible to explicitly specify the variance of residual of y by `y <=> y`. Here, the operator <=> indicates the covariance of the RHS and LHS variables should be freely estimated.
2. If all of the specified equations do not contain the intercept variable 1, then the intercept of each endogenous and observed variable will be freely estimated. Since the intercept variable 1 doesn't appear in this example, the intercept for y will be set as a freely estimated parameter. We can explicitly set the intercept term by `y <= 1`. However, under the situation that many equations are specified, once the intercept variable 1 appears in some equation, intercept terms in other equations should be explicitly specified. Otherwise, the intercepts of endogenous and observed variables in other equations will be fixed at zero.
3. For any set of exogeneous variables (i.e., variables that are not influenced by any other variable in the specified system), not only their variances will be freely estimated, but also their pairwise covariances will be set as freely estimated parameters. In this example, x1 and x2 are both exogeneous variables. Hence, their variance and pairwise covariances will be automatically set as freely estimated parameters. These covariances can be explicitly stated by simply `x1 + x2 <=> x1 + x2`. The syntax parser in `lslx` will consider variance/covariance of each combination of LHS and RHS variables.
4. The intercepts (or means) of exogeneous and observed variables are always set as freely estimated parameters. In this example, the intercepts of x1 and x2 will be freely estimated. It can be stated explicitly by `x1 + x2 <= 1`. Also, the `lslx` parser will know that the intercept variable 1 has effect on all of x1 and x2.

The previous regression example can be equivalently represented by `x1 + x2 => y`. In **lslx** all of the directed operators can be reversed under the stage of model specification. Users can choose the directions of operators according to their own preference.

The unique feature of **lslx** is that all of the parameters can be set as penalized. To penalize all of the regression coefficients, the equation can be modified as

```
y <~ x1 + x2
```

Here, the operator <~ means that the regression coefficients from the RHS variables to the LHS variables should be estimated with penalization. If only the coefficient of x1 should be penalized, we can use prefix to partly modify the equation

```
y <= pen() * x1 + x2
```

or equivalently

```
y <~ x1 + free() * x2
```

Both `pen()` and `free()` are prefix to modify the **lslx** operators. `pen()` makes the corresponding parameter to be penalized and `free()` makes it to be freely estimated. Inside the parentheses, starting values can be specified. Any prefix must present before some variable name and divided by asterisk `*`. Note that prefix can appear in the either RHS or LHS of operators and its function can be 'distributed' to the variables in the other side. For example, `free() * y <~ x1 + x2` will be interpreted as that all of the coefficients should be freely estimated. However, any prefix cannot simultaneously appear on both sides of operators, which may result in an ambiguity specification.

Example 2: Factor Analysis Model

Now, we consider another example of equation specification.

```
y1 + y2 + y3 <=: f1
y4 + y5 + y6 <=: f2
y7 + y8 + y9 <=: f3
```

This example is a factor analysis model with nine observed variables and three latent factors. In **lslx**, defining a latent factor can be through the operator `<=:` which means that the RHS factor is defined by LHS observed variables. The observed variables must be presented in the given data set. Of course, `f1` can be equivalently defined by `f1 :=> y1 + y2 + y3`.

As addressed in the first example, the `lslx` parser will automatically set many parameters that are not directly presented in these equations.

1. In this example, all of the observed variables are directed by some latent factor and hence they are endogenous. The variances of their residuals will be set as freely estimated parameters. Also, their intercepts will be freely estimated since no intercept variable `1` presents in the specified equations.
2. The three latent factors `f1`, `f2`, and `f3` are exogenous variables. Their pairwise covariances will be also set as freely estimated parameters. However, because they are latent but not observed, their intercepts will be fixed at zero. If user hope to estimate the latent factor means, they should add an additional equation `f1 + f2 + f3 <= 1`. After adding this equation, on the one hand, the latent intercepts will be set as free as indicated by that equation. On the other hand, since intercept variable `1` now presents in the specified equations, the intercepts for the endogenous and observed variables will be then fixed at zero.

So far, the specification for the factor analysis model is not complete since the scales of factors are not yet determined. In SEM, there are two common ways for scale setting. The first way is to fix some loading per factor. For example, we may respecify the model via

```
fix(1) * y1 + y2 + y3 <=: f1
fix(1) * y4 + y5 + y6 <=: f2
fix(1) * y7 + y8 + y9 <=: f3
```

The prefix `fix(1)` will fix the corresponding loadings to be one. Simply using `1 * y1` will be also interpreted as fixing the loading of `y1` at one to mimic **lavaan**. The second way for scale setting is fixing the variance of latent factors, which can be achieved by specifying additional equations

```
fix(1) * f1 <=> f1
fix(1) * f2 <=> f2
fix(1) * f3 <=> f3
```

Note that in the current version of **lslx**, scale setting will be not made automatically. Users must accomplish it manually.

When conducting factor analysis, we may face the problem that each variable may not be influenced by only one latent factor. The semi-confirmatory factor analysis, which penalizes some part of loading matrix, can be applied in this situation. One possible model specification for the semi-confirmatory approach is

```
y1 + y2 + y3 <=: f1
y4 + y5 + y6 <=: f2
y7 + y8 + y9 <=: f3
y4 + y5 + y6 + y7 + y8 + y9 <~: f1
y1 + y2 + y3 + y7 + y8 + y9 <~: f2
y4 + y5 + y6 + y7 + y8 + y9 <~: f3
fix(1) * f1 <=> f1
fix(1) * f2 <=> f2
fix(1) * f3 <=> f3
```

In this specification, loadings in the non-independent cluster will be also estimated but with penalization.

After version 0.6.3, **lslx** supports basic **lavaan** operators. The previous model can be equivalently specified as

```
f1 =~ y1 + y2 + y3
f2 =~ y4 + y5 + y6
f3 =~ y7 + y8 + y9
pen() * f1 =~ y4 + y5 + y6 + y7 + y8 + y9
pen() * f2 =~ y1 + y2 + y3 + y7 + y8 + y9
pen() * f3 =~ y4 + y5 + y6 + y7 + y8 + y9
f1 ~~ 1 * f1
f2 ~~ 1 * f2
f3 ~~ 1 * f3
```

Example 3: Path Models with both Observed Variables and Latent Factors

In the third example, we consider a path model with both observed variables and latent factors

```
fix(1) * y1 + y2 + y3 <=: f1
fix(1) * y4 + y5 + y6 <=: f2
fix(1) * y7 + y8 + y9 <=: f3
f3 <= f1 + f2
f1 + f2 + f3 <~ x1 + x2
f1 <~> f2
```

The first three equations specify the measurement model for `y1 - y9` and `f1 - f3`. The forth equation describes the relations among latent factor. The fifth equation sets all the coefficients from `x1 - x2` to `f1 - f3` to be penalized. The final equation states that the covariance of residuals of `f1` and `f2` is estimated with penalization, which is achieved by the operator `<~>`.

Like Example 1 and 2, many parameters in the current example are automatically set by **lslx**.

1. Because `y1 - y9` and `f1 - f3` are all endogenous, the variances of their residuals will be treated as freely estimated parameters. Also, due to the non-presence of intercept variable `1`, the intercept of `y1 - y9` will be set as free parameters and the intercept of `f1 - f3` will be set as zero.
2. The variance, intercepts, and pairwise covariances of exogenous and observed variables `x1 - x2` will be all estimated freely.

In this example, we can see that model specification in **lslx** is quite flexible. Like usual SEM, users can specify their models according some substantive theory. If no theory is available to guide the relationships in some part of the model, the semi-confirmatory approach can set this part as exploratory by setting the corresponding parameters as penalized.

Example 4: Multi-Group Factor Analysis Model

In the fourth example, we consider a multi-group factor analysis model.

```
fix(1) * y1 + y2 + y3 <=: f1
```

```
fix(1) * y4 + y5 + y6 <=: f2
```

```
fix(1) * y7 + y8 + y9 <=: f3
```

The syntax specifies a factor analysis model with nine observed variables and three latent factors. Loadings for `y2`, `y3`, `y5`, `y6`, `y8`, and `y9` are freely estimated in both groups. Loadings for `y1`, `y4`, and `y7` are set as fixed for scale setting in both groups. You may observe that the syntax for multi-group analysis is the same as that for single group analysis. That is true because in **lslx** a multi-group analysis is mainly identified by specifying a group variable. If the imported data can be divided into several samples based on some group variable (argument `group_variable` in `new` method, please see the section of *Initialize Method*) for group labeling, **lslx** will automatically conduct multi-group analysis (see example of *Semi-Confirmatory Multi-Group Factor Analysis* in the Section of *Examples*).

Sometimes, we may hope to specify different model structures for the two groups. It can be achieved by using vector version of prefix, which is also motivated by the syntax in **lavaan**. For example, if we hope to restrict the loading for `y2` to be 1 in the first group but set it as freely estimate parameter in the second group. Then we may use

```
fix(1) * y1 + c(fix(1), free()) * y2 + y3 <=: f1
```

Note that the order of groups is important here. Since **lslx** treats the group variable as `factor`, its order is determined by the sorted name of groups. For example, if three groups `c`, `a`, and `b` are considered, then the first group is `a`, the second is `b`, and the third is `c`.

In the current version of **lslx**, coefficient constraints cannot be imposed. It seems that testing coefficient invariance across groups is impossible in **lslx**. However, the present package parameterizes group coefficients in different way compared to other SEM software (Huang, 2018). Under **lslx**, each group coefficient is decomposed into a sum of a reference component and an increment component. If the reference component is assumed to be zero, the increment component represents the group coefficient, which is equivalent to the usual parameterization in other software solutions. On the other hand, if some group is set as reference (argument `reference_group` in `new` method, please see the section of *Initialize Method*), then the reference component now represents the group coefficient of the reference group and other increment components represent the differences from the reference group. The coefficient invariance across groups can be evaluated by examining the value or sparsity of the corresponding increment component.

Optimization Algorithm

Let θ denote the vector of model parameter. **lslx** tries to find a PL estimate for θ by minimizing the objective function $objective(\theta, \lambda) = loss(\theta) + regularizer(\theta, \lambda)$ where $loss$ is the ML loss function, $regularizer$ is a regularizer, possibly lasso (Tibshirani, 1996) or mcp (Zhang, 2010), and λ is a regularization parameter. The optimization algorithm for minimizing the PL criterion is based on an improved **glmnet** method (Friedman, Hastie, & Tibshirani, 2010) made by Yuan, Ho, and Lin (2012). The algorithm can be understood as a quasi-Newton method with inner loop and outer loop. The inner loop of the algorithm derives a quasi-Newton direction by minimizing a quadratic approximated objective function via coordinate descent. To save the computation time, the Hessian matrix for the quadratic term is approximated by the identity matrix, the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method, or the expected Hessian (Fisher scoring). Although the computational cost of BFGS approximation is much smaller than calculating expected hessian, our experience shows that the two methods perform similarly in terms of computation time because more outer iterations are required for BFGS. The inner loop stops if the change of the derived direction is quite small. The outer loop of the algorithm updates the value of parameter estimate via the derived quasi-Newton direction and Armijo's rule. The outer loop stops if the maximal absolute element of subgradient of objective function is smaller than the specified tolerance. The minimizer is the so-called PL estimates. Note that the PL estimates is a function of penalty level, i.e., PL estimates can vary under different penalty levels. An optimal penalty level can be chosen by using model selection criterion.

In **lslx**, PL estimates under each penalty level and convexity level specified by user will be calculated. The convexity levels will be sorted from large to small based on the suggestion of Mazumder (2011). The previous obtained PL estimate will be used as warm start for further minimization problem. Since the solution path is continuous, the warm start can speed up the convergence of minimization (see Friedman, Hastie, & Tibshirani, 2010).

Missing Data

When conducting SEM, it is easy to encounter the problem of missing data. In **lslx**, missing data can be handled by the listwise deletion method and the two-stage method (Yuan & Bentler, 2000). The listwise deletion method only uses fully complete observations for further SEM analysis. If the missing mechanism is missing completely at random (MCAR; Rubin, 1976), the listwise deletion method can yield a consistent estimator. The two-stage method first calculates the saturated moments by minimizing the likelihoods based on all of the available observations and then use the obtained saturated moment estimates for further SEM analysis. Under the assumption of missing at random (MAR; Rubin, 1976), it has been shown that the two-stage method can yield a consistent estimate. In addition, the standard errors of coefficients can be also consistently estimated if a correct asymptotic covariance of saturated moments is used. Because the two-stage approach is generally valid and efficient compared to the listwise deletion method, **lslx** set the two-stage method as default for handling the missing data problem. The current version also supports the use of auxiliary variables (see Savalei & Bentler, 2008). If the two-stage method is implemented, the standard error formula will be corrected for the presence of missing data (see Yuan & Lu, 2008 for technical details).

So far, **lslx** doesn't include the full-information maximum likelihood (FIML) method for missing values. One reason is that PL can be computationally intensive if many penalty levels are considered. The additional E-step in each iteration of FIML makes the problem worse. Another reason is that the two-step method has been shown to outperform FIML in simulation settings (Savalei &

Falk, 2014). Therefore, we tend to believe that the implementation of FIML in PL may not bring further advantages over the two-step method.

Penalty Level Selection

Penalty level selection in **lslx** is based on optimizing the value of some information criterion. Many information criteria are available for this task. In the current version, available information criteria are

`aic` Akaike Information Criterion (Akaike, 1974)

$$AIC(\theta) = loss(\theta) - (2/N) * df(\theta)$$

`aic3` Akaike Information Criterion with Penalty Being 3 (Sclove, 1987)

$$AIC3(\theta) = loss(\theta) - (3/N) * df(\theta)$$

`caic` Consistent Akaike Information Criterion (Bozdogan, 1987)

$$CAIC(\theta) = loss(\theta) - ((log(N) + 1)/N) * df(\theta)$$

`bic` Bayesian Information Criterion (Schwarz, 1978)

$$BIC(\theta) = loss(\theta) - (log(N)/N) * df(\theta)$$

`abic` Adjusted Bayesian Information Criterion (Sclove, 1987)

$$ABIC(\theta) = loss(\theta) - (log((N + 2)/24)/N) * df(\theta)$$

`hbic` Haughton Bayesian Information Criterion (Haughton, 1997)

$$HBIC(\theta) = loss(\theta) - (log(N/\pi)/N) * df(\theta)$$

where

- N: total number of sample size;
- G: total number of group;
- $loss(\theta)$: the loss value under estimate θ;
- $df(\theta)$: the degree of freedom defined as (1) $G * P * (P + 3)/2 - e(\theta)$ with $e(\theta)$ being the number of non-zero elements in θ for Lasso and MCP; or (2) the expectation of likelihood ratio statistics with ridge for ridge and elastic net.

Note the formula for calculating the information criteria in **lslx** are different to other software solutions. The loss function value is used to replace the likelihood function value and hence the penalty term is also divided by sample size N. For each information criterion, a robust version is calculated if raw data is available. Their corresponding names are `raic`, `raic3`, `rcaic`, `rbic`, `rabic`, and `rhbic` with "r" standing for "robust". These robust criteria use the Satorra-Bentler scaling factor for correcting degree of freedom. For the case of normal data and correctly specified model, the two versions will be the same asymptotically.

Huang, Chen, and Weng (2017) have study the asymptotic behaviors of `aic` and `bic` under penalized estimation. They show that under suitable conditions, `aic` can select a model with minimum expected loss and `bic` can choose the most parsimonious one from models that attain the minimum expected loss. By the order of penalty term, we may expect: (1) the large sample behaviors of `aic3` and `tic` will be similar to `aic`; and (2) the asymptotic behaviors of `caic`, `abic`, and `hbic` will be similar to `bic`. However, their small-sample performances require further studies.

Model Fit Evaluation

Given a chosen penalty level, we may evaluate the overall model fit by using fit indices. In the current version, available fit indices for model evaluation are

`rmsea` Root Mean Square Error of Approximation (Steiger, 1998; Steiger & Lind, 1980)

$$RMSEA(\theta) = \sqrt{(}G * max(loss(\theta)/df(\theta) - 1/N, 0))$$

`cfi` Comparative Fit Index (Bentler, 1990)

$$CFI(\theta) = (max(loss_0 - df_0/N, 0) - max(loss(\theta) - df(\theta)/N, 0))/max(loss_0 - df_0/N, 0)$$

`nnfi` Non-Normed Fit Index (Tucker & Lewis, 1973)

$$NNFI(\theta) = (loss_0/df_0 - loss(\theta)/df(\theta))/(loss_0/df_0 - 1/N)$$

`srmr` Standardized Root Mean of Residual (Bentler, 1995)

$$SRMR(\theta) = \sqrt{(}\sum_g w_g \sum_i \sum_{j \leq i} ((\sigma_{gij} - s_{gij})^2/(\sigma_{gii} * \sigma_{gjj}))/(G * P * (P+1)/2)$$
$$+ \sum_g w_g \sum_i ((\mu_{gi} - m_{gi})^2/\sigma_{gii})/(G * P))$$

where

- N: total number of sample size;
- G: total number of groups;
- P: number of observed variables;
- w_g: sample weight of group g;
- $loss(\theta)$: the loss value under estimate θ;
- $\#(\theta)$: the number of non-zero elements in θ;
- $df(\theta)$: the degree of freedom defined by $G * P * (P+3)/2 - \#(\theta)$;
- $loss_0$: the loss value under baseline model;
- df_0: the degree of freedom under baseline model;
- σ_{gij}: the (i, j) element of model implied covariance at group g;
- s_{gij}: the (i, j) element of sample covariance at group g;
- μ_{gi}: the i element of model implied mean at group g;
- m_{gi}: the i element of sample mean at group g;

In **lslx**, the baseline model is the model that assumes a diagonal covariance matrix and a saturated mean. Hence, the baseline model may not be appropriate if users hope to evaluate the goodness-of-fit of mean structure.

It is also possible to test overall model fit by formal statistical test. In the current version, statistical tests for likelihood ratio (LR) and root mean square error of approximation (RMSEA) can be implemented. If raw data is available, **lslx** calculates mean-adjusted versions of LR statistic (Satorra & Bentler, 1994) and RMSEA intervals (Brosseau-Liard, Savalei & Li, 2012; Li & Bentler, 2006). It should be noted that the classical tests may not be valid after penalty level selection because the task of penalty level selection may destroy the sampling distribution of test statistics (see Pötscher, 1991 for discussion). Valid post model selection inference methods require further development.

Coefficient Evaluation

Given a chosen penalty level, we may evaluate the significance of coefficients (or parameters). In the current version, standard errors based on the expected/observed Fisher information matrix and the sandwich formula are available (see Yuan & Hayashi, 2006 for discussion). Because the sandwich formula is generally valid compared to the approaches based on Fisher information, **lslx** uses the sandwich formula as default whenever raw data is available. Note that sandwich covariance matrix in **lslx** is calculated based on Equation (14) in Yuan and Hayashi (2006) but not Equation (2.12a) in Browne (1984) to accommodate the potential model misspecification. Again, the significance tests may not be valid after penalty level selection. After version 0.6.4, several post-selection mehtods are available (Huang, in press), inlcuding PoSI method with Scheffe constant (Berk, Brown, Buja, Zhang, & Zhao, 2013) and Polyhedral method (Lee, Sun, Sun, & Taylor, 2016).

Initialize Method

```
$new(model, data, numeric_variable, ordered_variable,
  group_variable, reference_group, weight_variable, auxiliary_variable,
  sample_cov, sample_mean, sample_size, sample_moment_acov, verbose = TRUE)
```

Arguments

`model` A `character` with length one to represent the model specification.

`data` A `data.frame` of raw data. It must contains variables specified in `model` (and possibly the variables specified by `group_variable` and `weight_variable`).

`numeric_variable` A `character` to specify which response variables should be transfromed into `numeric`.

`ordered_variable` A `character` to specify which response variables should be transfromed into `ordered`.

`weight_variable` A `character` with length one to specify what variable is used for sampling weight.

`auxiliary_variable` A `character` to specify what variable(s) is used as auxiliary variable(s) for estimating saturated moments when missing data presents and two-step method is implemented. Auxiliary variable(s) must be numeric. If any categorical auxiliary is considered, please transform it into dummy variables before initialization.

`group_variable` A `character` with length one to specify what variable is used for labeling group.

`reference_group` A `character` with length one to specify which group is set as reference.

`sample_cov` A numeric `matrix` (single group case) or a `list` of numeric `matrix` (multi-group case) to represent sample covariance matrixs. It must have row and column names that match the variable names specified in `model`.

`sample_mean` A `numeric` (single group case) or a `list` of `numeric` (multi-group case) to represent sample mean vectors.

`sample_size` A `numeric` (single group case) with length one or a `list` of `numeric` (multi-group case) to represent the sample sizes.

`sample_moment_acov` A numeric `matrix` (single group case) or a `list` of numeric `matrix` (multi-group case) to represent asymptotic covariance for moments.

`verbose` A `logical` to specify whether messages made by `lslx` should be printed.

Details

`$new()` initializes a new object of `lslx` R6 class for fitting semi-confirmatory structural equation modeling (SEM). In most cases, a new `lslx` object is initialized by supplying `model` and `data`. For details of syntax for model specification, see the section of *Model Syntax*. By default, types of response variables (`numeric` versus `ordered`) will be infered according to the imported `data.frame`. If users hope to transform variables types in `lslx`, `numeric_variable` and `ordered_variable` can be used. When multi-group analysis is desired, argument `group_variable` should be given to specify what variable is used for labeling group. Argument `reference_group` can be used to set reference group. Note that if some group is set as reference, the coefficients in other groups will represent increments from the reference. When the missingness of data depends on some other variables, `auxiliary_variable` can be used to specify auxiliary variables for estimate saturated moments. For details of missing data, see the section of *Missing Data*. If raw data is not available, `lslx` also supports initialization via sample moments. In that case, `sample_cov` and `sample_size` are required. If `sample_mean` is missing under moment initialization, it is assumed to be zero.

Set-Related Methods

```
$free_coefficient(name, start, verbose = TRUE)
$penalize_coefficient(name, start, verbose = TRUE)
$fix_coefficient(name, start, verbose = TRUE)

$free_directed(left, right, group, verbose = TRUE)
$penalize_directed(left, right, group, verbose = TRUE)
$fix_directed(left, right, group, verbose = TRUE)

$free_undirected(both, group, verbose = TRUE)
$penalize_undirected(both, group, verbose = TRUE)
$fix_undirected(both, group, verbose = TRUE)

$free_block(block, group, type, verbose = TRUE)
$penalize_block(block, group, type, verbose = TRUE)
$fix_block(block, group, type, verbose = TRUE)

$free_heterogeneity(block, group, verbose = TRUE)
$penalize_heterogeneity(block, group, verbose = TRUE)
$fix_heterogeneity(block, group, verbose = TRUE)
```

Arguments

`name` A `character` to indicate which coefficients should be reset.

`start` A `numeric` to specify starting values. The length of `start` should be one or match the length of `name` to avoid ambiguity. If `start` is missing, the starting value will be set as (1) `NA` for free or penalized coefficient; and (2) `0` for fixed coefficient.

`left` A `character` to indicate variable names in the left-hand side of operator `"<-"`.

`right` A `character` to indicate variable names in the right-hand side of operator `"<-"`.

`both` A `character` to indicate variable names in both side of operator `"<->"`.

`group` A `character` to indicate group names that the specified relations belong to.

`block` A `character` with length one to indicate a block such that the corresponding target coefficient will be reset. Its value must be `"f<-1"`, `"y<-1"`, `"f<-f"`, `"f<-y"`, `"y<-f"`, `"y<-y"`, `"f<->f"`, `"f<->y"`, `"y<->f"`, or `"y<->y"`.

`type` A `character` to indicate which type of parameter should be changed in the given block. Its value must be `"free"`, `"fixed"`, or `"pen"`. If `type` is not specified, the types of all parameters in the given `block` will be modified.

`verbose` A `logical` to specify whether messages made by `lslx` should be printed.

Details

Set-related methods include several member functions that can be used to modify the initialized model specification. Like most encapsulation objects, set-related function is used to modify the inner members of object. So far, the set-related methods are all established to modify the model. The data are protected without any modification.

`$free_coefficient()` / `$penalize_coefficient()` / `$fix_coefficient()` sets the coefficient named `name` as FREE / PENALIZED / FIXED with starting value `start`. In the case of single group analysis, argument `name` can be replaced by relations, i.e., the group name can be omitted.

`$free_directed()` / `$penalize_directed()` / `$fix_directed()` sets all the regression coefficients from variables in `right` to variables in `left` at groups in `group` as FREE / PENALIZED / FIXED.

`$free_undirected()` / `$penalize_undirected()` / `$fix_undirected()` sets all the covariances among variables in `both` at groups in `group` as FREE / PENALIZED / FIXED. Note that this method all always not modify the variance of variables specified in `both`.

`$free_block()` / `$penalize_block()` / `$fix_block()` sets all the parameters belonging to `block` at `group` with `type` as FREE / PENALIZED / FIXED.

`$free_heterogeneity()` / `$penalize_heterogeneity()` / `$fix_heterogeneity()` sets every target coefficient as FREE / PENALIZED / FIXED. A target coefficient should satisfy that (1) it belongs to `block` at `group`, and (2) it has either free or penalized reference component. The method is only available in the case of multi-group analysis with specified reference group.

Inside `lslx` object, every coefficient (or parameter) has its own name and belongs to some block.

- The coefficient name is constructed by a relation and a group name. A relation is defined by combining a left variable name, an operator, and a right variable name. For example, `"y1<-f1"` is a relation to represent the coefficient from `"f1"` to `"y1"`. Note that in relation we only use operators `"<-"` and `"<->"`. `"->"` is illegal and no distinction between `"="` and `"~"` are made. In multi-group analysis cases, a coefficient name requires explicitly specifying group name. For example, `"y1<-f1/G1"` is the parameter name for the path coefficient from `"f1"` to `"y1"` in group `"G1"`.
- Block is defined by the types of left variable and right variable, and the operator. In **lslx**, `"y"` is used to indicate observed response, `"f"` is used for latent factor, and `"1"` is for intercept. Hence, `"y<-f"` is the block that contains all the coefficients from latent factors to observed responses. There are 10 possible distinct blocks: `"f<-1"`, `"y<-1"`, `"f<-f"`, `"f<-y"`, `"y<-f"`, `"y<-y"`, `"f<->f"`, `"f<->y"`, `"y<->f"`, and `"y<->y"`

Arguments in set-related methods may rely on the naming rule of coefficent name and block to modify model specification.

Fit-Related Methods

```
$fit(penalty_method = "lasso", lambda_grid = "default", delta_grid = "default",
  step_grid = "default", loss = "default", algorithm = "default",
  missing_method = "default", start_method = "default",
  lambda_direction = "default", lambda_length = 50L, delta_length = 3L,
  threshold_value = 0.3, iter_out_max = 100L, iter_in_max = 50L,
  iter_other_max = 500L, iter_armijo_max = 100L, tol_out = 1e-3,
  tol_in = 1e-3, tol_other = 1e-7, step_size = 0.5, momentum = 0,
  armijo = 1e-5, ridge_cov = 0, ridge_hessian = 1e-4, warm_start = TRUE,
  positive_variance = TRUE, minimum_variance = 1e-4, enforce_cd = FALSE,
  random_update = TRUE, weight_matrix = NULL, verbose = TRUE)
$fit_none(...)
$fit_lasso(lambda_grid = "default", ...)
$fit_ridge(lambda_grid = "default", ...)
$fit_elastic_net(lambda_grid = "default", delta_grid = "default", ...)
$fit_mcp(lambda_grid = "default", delta_grid = "default", ...)
$fit_forward(step_grid = "default", ...)
$fit_backward(step_grid = "default", ...)
```

Arguments

`penalty_method` A `character` to specify the penalty method. There are two class penalty methods can be specified. For penalized estimation with a regularizer, `"lasso"`, `"ridge"`, `"elastic_net"`, and `"mcp"` can be used. For penalized estimation with stepwise search, `"forward"` and `"backward"` can be implemented. If no penalty is considered, we can set `penalty_method` as `"none"`.

`lambda_grid` A non-negative `numeric` to specify penalty levels for the regularizer. If it is set as `"default"`, its value will be generated automatically based on the variable scales.

`delta_grid` A non-negative `numeric` to specify the combination weight for `"elastic_net"` or the convexity level for `"mcp"`. If it is set as `"default"`, its value will be generated automatically.

`step_grid` A non-negative `numeric` to specify a grid for stepwise search with `"forward"` and `"backward"`.

`loss` A `character` to determine the loss function. The current version supports `"ml"` (maximum likelihood), `"uls"` (unweighted least squares), `"dwls"` (diagonal weighted least squres), and `"wls"` (weighted least squares). The maximum likelihood is only available for all continuous response variables. If the argument is set as `"default"`, then (1) `"ml"` will be implemented for all continuous response variables; (2) `"dwls"` will be implemented for ordinal or mixed types response variables.

`algorithm` A `character` to determine the method of optimization. The current version supports `"gd"` (gradient descent), `"bfgs"` (Broyden-Fletcher-Goldfarb-Shanno), `"fisher"` (Fisher scoring), and `"dynamic"` (an adaptive algorithm). If the argument is set as `"default"`, then `"dynamic"` will be implemented.

`missing_method` A `character` to determine the method for handling missing data (or `NA`). The current version supports `"two_stage"` and `"listwise_deletion"`. If the argument is set as `"default"` and a raw data set is available, the `"two_stage"` will be implemented. If the argument is set as `"default"` and only moment data is available, the `"listwise_deletion"` will be used (actually, in this case no missing presences).

start_method A character to determine the method for calculating unspecified starting values. The current version supports "mh" (McDonald & Hartmann, 1992) and "heuristic". If the argument is set as "default", the "mh" will be implemented.

lambda_direction A character to determine the "direction" of lambda_grid. "decrease" sorts lambda_grid from large to small. On the contrary, "increase" sorts lambda_grid from small to large. If the argument is set as "default", "increase" will be used when the smallest element of lambda_grid is larger than zero; otherwise, "decrease" is assumed.

lambda_length A numeric to specify the length of automatically generated lambda_grid under lambda_grid = "default".

delta_length A numeric to specify the length of automatically generated delta_grid under delta_grid = "default".

threshold_value A numeric to specify the "largest threshold value" for lambda_grid initialization.

iter_out_max A positive integer to specify the maximal iterations for outer loop of the modified glmnet algorithm.

iter_in_max A positive integer to specify the maximal iterations for inner loop of the modified glmnet algorith.

iter_other_max A positive integer to specify the maximal iterations for other loop.

iter_armijo_max A positive integer to specify the maximal iterations for searching step-size via Armijo rule.

tol_out A small positive numeric to specify the tolerance (convergence criterion) for outer loop of the modified glmnet algorithm.

tol_in A small positive numeric to specify the tolerance (convergence criterion) for inner loop of the modified glmnet algorithm.

tol_other A small positive numeric to specify the tolerance (convergence criterion) for other loop.

step_size A positive numeric smaller than one to specify the step-size.

momentum A numeric between 0 and 1 for momentum parameter.

armijo A small positive numeric for the constant in Armijo rule.

ridge_cov A small positive numeric for the ridge of sample covariance matrix.

ridge_hessian A small positive numeric for the ridge of approximated hessian in optimization.

ridge_weight A small positive numeric for the ridge of weight matrix in weighted least squares.

warm_start A logical to specify whether the warm start approach should be used.

positive_variance A logical to specify whether the variance estimate should be constrained to be larger than minimum_variance.

minimum_variance A numeric to specify the minimum value of variance if positive_variance = TRUE.

enforce_cd A logic to specify whether coordinate descent should be used when no penalty function is used. Its default value is TRUE.

random_update A logic to specify whether coordinate descent should be conducted in a random order. Its default value is TRUE.

`weight_matrix` A `list` with length equaling to the number of groups to specify a user-defined weight matrix for least squares loss.

`verbose` A `logical` to specify whether messages made by `lslx` should be printed.

`...` Other passing arguments for calling `$fit()`.

Details

Fit-related methods are used for fitting model to data. The success of these methods may depend on the specified fitting control. For details of optimization algorithm, see the section of *Optimization Algorithm*.

`$fit()` fits the specified model to data by minimizing a penalized loss function. It is the most comprehensive fit method and hence many arguments can be specified.

`$fit_none()` fits the specified model to data by minimizing a loss function without penalty. It is a user convinient wrapper of `$fit()` with `penalty_method = "none"`.

`$fit_lasso()` fits the specified model to data by minimizing a loss function with lasso penalty (Tibshirani, 1996). It is a user convinient wrapper of `$fit()` with `penalty_method = "lasso"`.

`$fit_ridge()` fits the specified model to data by minimizing a loss function with ridge penalty (Hoerl & Kennard, 1970). It is a user convinient wrapper of `$fit()` with `penalty_method = "ridge"`.

`$fit_elastic_net()` fits the specified model to data by minimizing a loss function with elastic net penalty (Zou & Hastie, 2005). It is a user convinient wrapper of `$fit()` with `penalty_method = "elastic_net"`.

`$fit_mcp()` method fits the specified model to data by minimizing a loss function with mcp (Zhang, 2010). It is a user convinient wrapper of `$fit()` with `penalty_method = "mcp"`.

`$fit_forward()` method fits the specified model to data by minimizing a loss function with forward searching. It is a user convinient wrapper of `$fit()` with `penalty_method = "forward"`.

`$fit_backward()` method fits the specified model to data by minimizing a loss function with backward searching. It is a user convinient wrapper of `$fit()` with `penalty_method = "backward"`.

Summarize Method

```
$summarize(selector, lambda, delta, step, standard_error = "default",
  debias = "default", inference = "default", alpha_level = .05,
  include_faulty = FALSE, style = "default", mode = "default", digit = 3,
  interval = TRUE, output)
```

Arguments

`selector` A `character` to specify a selector for determining an optimal penalty level. Its value can be any one in `"aic"`, `"aic3"`, `"caic"`, `"bic"`, `"abic"`, `"hbic"`, or their robust counterparts `"raic"`, `"raic3"`, `"rcaic"`, `"rbic"`, `"rabic"`, `"rhbic"` if raw data is available.

`lambda` A `numeric` to specific a chosen optimal penalty level. If the specified `lambda` is not in `lambda_grid`, a nearest legitimate value will be used.

`delta` A `numeric` to specific a chosen optimal convexity level. If the specified `delta` is not in `delta_grid`, a nearest legitimate value will be used.

`step` A `numeric` to specific a chosen step for stepwise searching. If the specified `step` is not in `step_grid`, a nearest legitimate value will be used.

`standard_error` A `character` to specify the standard error to be used for hypothesis testing. The argument can be either `"sandwich"`, `"expected_information"`, and `"observed_information"`. If it is specified as `"default"`, it will be set as (1) `"sandwich"` when raw data is available; (2) `"observed_information"` when only moment data is available.

`debias` A `character` to specify a debias method for obtaining a debiased estimator. Its value can be either `"none"` or `"one_step"`. If it is specified as `"default"`, `"none"` will be used unless `post = "polyhedral"` is used.

`inference` A `character` to specify the method for post selection inference. The current version supports `"naive"`, `"polyhedral"`, and `"scheffe"`. If it is specified as `"default"`, `"naive"` will be used.

`alpha_level` A `numeric` to specify the alpha level for constructing 1 - alpha confidence intervals.

`include_faulty` A `logical` to specify whether non-convergence or non-convexity results should be removed for penalty level selection. Non-convergence result determined by examining the maximal elements of absolute objective gradient and the number of iterations. non-convexity result is determined by checking the minimum of univariate approximate hessian.

`style` A `character` to specify whether the style of summary. Its value must be either `"default"`, `"manual"`, `"minimal"`, or `"maximal"`.

`mode` A `character` to specify the mode of summary. Its value must be `"print"` or `"return"`. `"print"` will print the summary result and `"return"` will return a `list` of the summary result. If it is specified as `"default"`, it will be set as `"print"`.

`digit` An `integer` to specify the number of digits to be displayed.

`interval` A `logical` to specify whether the confidence interval should be printed.

Details

`$summarize()` prints a summary for the fitting result under a selected peanlty/convexity level. It requires users to specify which selector should be used or a combination of `gamma` and `lambda`.. By default, the summary includes model information, numerical conditions, fit indices, coefficient estimates, and related statistical inference. It can be modified by the `style` argument. For details of evaluation and inference methods, see the sections of *Model Fit Evaluation* and *Coefficient Evaluation*.

Test-Related Methods

```
$test_lr(selector, lambda, delta, step, include_faulty = FALSE)
$test_rmsea(selector, lambda, delta, step,
  alpha_level = .05, include_faulty = FALSE)
$test_coefficient(selector, lambda, delta, step,
  standard_error = "default", debias = "default", inference = "default",
  alpha_level = .05, include_faulty = FALSE)
```

Arguments

`selector` A `character` to specify a selector for determining an optimal penalty level. Its value can be any one in `"aic"`, `"aic3"`, `"caic"`, `"bic"`, `"abic"`, `"hbic"`, or their robust counterparts `"raic"`, `"raic3"`, `"rcaic"`, `"rbic"`, `"rabic"`, `"rhbic"` if raw data is available.

`delta` A `numeric` to specific a chosen optimal weight for elastic net or convexity level for mcp. If the specified `delta` is not in `delta_grid`, a nearest legitimate value will be used.

`step` A `numeric` to specific a chosen step for stepwise searching. If the specified `step` is not in `step_grid`, a nearest legitimate value will be used.

`standard_error` A `character` to specify the standard error to be used for hypothesis testing. The argument can be either `"sandwich"`, `"expected_information"`, and `"observed_information"`. If it is specified as `"default"`, it will be set as (1) `"sandwich"` when raw data is available; (2) `"observed_information"` when only moment data is available.

`debias` A `character` to specify a debias method for obtaining a debiased estimator. Its value can be either `"none"` or `"one_step"`. If it is specified as `"default"`, `"none"` will be used unless `post = "polyhedral"` is used.

`inference` A `character` to specify the method for post selection inference. The current version supports `"naive"`, `"polyhedral"`, and `"scheffe"`. If it is specified as `"default"`, `"naive"` will be used.

`alpha_level` A `numeric` to specify the alpha level for constructing 1 - alpha confidence intervals.

`include_faulty` A `logical` to specify whether non-convergence or non-convexity results should be removed for penalty level selection. Non-convergence result determined by examining the maximal elements of absolute objective gradient and the number of iteration. non-convexity result is determined by checking the minimum of univariate approximate hessian.

Details

Test-related methods are used to obtain the result of specific statistical test. So far, only tests for likelihood ratio (LR), root mean square error of approximation (RMSEA), and coefficients are available.

`$test_lr()` returns a `data.frame` of result for likelihood ratio test. If raw data is available, it also calculates a mean-adjusted statistic. For details of significance test method for LR, see the section of *Model Fit Evaluation*.

`$test_rmsea()` returns a `data.frame` of result for rmsea confidence intervals. If raw data is available, it also calculates a mean-adjusted confidence interval (Brosseau-Liard, Savalei & Li, 2012; Li & Bentler, 2006). For details of confidence interval construction for RMSEA, see the section of *Model Fit Evaluation*.

`$test_coefficient()` returns a `data.frame` of result for coefficient significance and confidence interval. For details of standard error formula for coefficients, see the section of *Coefficient Evaluation*.

Plot-Related Methods

```
$plot_numerical_condition(condition, x_scale = "default",
  mode = "default")
$plot_information_criterion(criterion, x_scale = "default",
  mode = "default")
$plot_fit_index(index, x_scale = "default", mode = "default")
$plot_coefficient(block, left, right, both, x_scale = "default",
  mode = "default")
```

Arguments

`condition` A `character` to specify which numerical conditions should be plotted. Its value must be `"objective_value"`, `"objective_gradient_abs_max"`, `"objective_hessian_convexity"`, `"n_iter_out"`, `"loss_value"`, `"n_nonzero_coefficient"`, or their combination.

`criterion` A `character` to specify which information criteria should be plotted. Its value must be `"aic"`, `"aic3"`, `"caic"`, `"bic"`, `"abic"`, `"hbic"`, `"raic"`, `"raic3"`, `"rcaic"`, `"rbic"`, `"rabic"`, `"rhbic"`, or their combination.

`index` A `character` to specify which fit indices should be plotted. Its value must be `"rmsea"`, `"cfi"`, `"nnfi"`, `"srmr"`, or their combination.

`block` A `character` with length one to indicate a block such that the corresponding target coefficient will be reset. Its value must be one of `"f<-1"`, `"y<-1"`, `"f<-f"`, `"f<-y"`, `"y<-f"`, `"y<-y"`, `"f<->f"`, `"f<->y"`, `"y<->f"`, or `"y<->y"`.

`left` A `character` to specify the variables in the left-hand side of operator in `block`.

`right` A `character` to specify the variables in the right-hand side of operator in `block`.

`both` A `character` to specify the variables in both sides of operator in `block`.

`lambda_scale` A `character` to specify the scale of lambda (x-axis) for `coord_trans()` in **ggplot2**.

`mode` A `character` to specify the mode of plot. Its value must be `"plot"` or `"return"`. `"plot"` will plot the result and `"return"` will return a `data.frame` for plot. If it is specified as `"default"`, it will be set as `"plot"`.

Details

Plot-related methods are used for visualizing the fitting results.

`$plot_numerical_condition()` plots the values of selected numerical conditions. It can be used to assess the quality of optimization. By default, `"n_iter_out"`, `"objective_gradient_abs_max"`, and `"objective_hessian_convexity"` across the given penalty levels are plotted.

`$plot_information_criterion()` shows how the values of information criteria vary with penalty levels. By default, `"aic"`, `"aic3"`, `"caic"`, `"bic"`, `"abic"`, and `"hbic"` are plotted.

`$plot_fit_index()` shows how the values of fit indices vary with penalty levels. By default, `"rmsea"`, `"cfi"`, `"nnfi"`, and `"srmr"` are plotted.

`$plot_coefficient()` visualizes the solution paths of coefficients belonging to the intersection of `block`, `left`, `right`, and `both` arguments. By default, all of the coefficients are plotted.

Get-Related Methods

```
$get_model()
$get_data()
$get_fitting()
```

Details

Get-related methods are defined to obtain a deep copy of members inside `lslx`. Note that all of the data members of `lslx` are set as private to protect the inner data structure. They cannot be assessed directly via `$`.

`$get_model()` returns a deep copy of `model` member in the current `lslx` object.

`$get_data()` returns a deep copy of `data` member in the current `lslx` object.

`$get_fitting()` returns a deep copy of `fitting` member in the current `lslx` object.

Extract-Related Methods

```
$extract_specification()
$extract_saturated_cov()
$extract_saturated_mean()
$extract_saturated_moment_acov()

$extract_penalty_level(selector, lambda, delta, step,
  include_faulty = FALSE)

$extract_numerical_condition(selector, lambda, delta, step,
  include_faulty = FALSE)
$extract_information_criterion(selector, lambda, delta, step,
  include_faulty = FALSE)
$extract_fit_index(selector, lambda, delta, step, include_faulty = FALSE)
$extract_cv_error(selector, lambda, delta, step, include_faulty = FALSE)

$extract_coefficient(selector, lambda, delta, step, type = "default",
  include_faulty = FALSE)
$extract_debiased_coefficient(selector, lambda, delta, step, type = "default",
  include_faulty = FALSE)

$extract_implied_cov(selector, lambda, delta, step,
  include_faulty = FALSE)
$extract_implied_mean(selector, lambda, delta, step,
  include_faulty = FALSE)
$extract_residual_cov(selector, lambda, delta, step,
  include_faulty = FALSE)
$extract_residual_mean(selector, lambda, delta, step,
  include_faulty = FALSE)

$extract_coefficient_matrix(selector, lambda, delta, step, block,
  include_faulty = FALSE)

$extract_moment_jacobian(selector, lambda, delta, step,
  type = "default", include_faulty = FALSE)

$extract_expected_information(selector, lambda, delta, step,
  type = "default", include_faulty = FALSE)
$extract_observed_information(selector, lambda, delta, step,
  type = "default", include_faulty = FALSE)

$extract_bfgs_hessian(selector, lambda, delta, step, type = "default",
  include_faulty = FALSE)
$extract_score_acov(selector, lambda, delta, step, type = "default",
  include_faulty = FALSE)
$extract_coefficient_acov(selector, lambda, delta, step,
  standard_error = "default", type = "default", include_faulty = FALSE)
```

```
$extract_loss_gradient(selector, lambda, delta, step, type = "default",
  include_faulty = FALSE)
$extract_regularizer_gradient(selector, lambda, delta, step,
  type = "default", include_faulty = FALSE)
$extract_objective_gradient(selector, lambda, delta, step,
  type = "default", include_faulty = FALSE)
```

Arguments

`selector` A `character` to specify a selector for determining an optimal penalty level. Its value can be any one in `"aic"`, `"aic3"`, `"caic"`, `"bic"`, `"abic"`, `"hbic"`, or their robust counterparts `"raic"`, `"raic3"`, `"rcaic"`, `"rbic"`, `"rabic"`, `"rhbic"` if raw data is available.

`lambda` A `numeric` to specific a chosen optimal penalty level. If the specified `lambda` is not in `lambda_grid`, a nearest legitimate value will be used.

`delta` A `numeric` to specific a chosen optimal weight for elastic net or convexity level for mcp. If the specified `delta` is not in `delta_grid`, a nearest legitimate value will be used.

`step` A `numeric` to specific a chosen step for stepwise searching. If the specified `step` is not in `step_grid`, a nearest legitimate value will be used.

`standard_error` A `character` to specify the standard error to be used for hypothesis testing. The argument can be either `"sandwich"`, `"expected_information"`, and `"observed_information"`. If it is specified as `"default"`, it will be set as (1) `"sandwich"` when raw data is available; (2) `"observed_information"` when only moment data is available.

`block` A `character` with length one to indicate a block such that the corresponding target coefficient will be reset. Its value must be `"f<-1"`, `"y<-1"`, `"f<-f"`, `"f<-y"`, `"y<-f"`, `"y<-y"`, `"f<->f"`, `"f<->y"`, `"y<->f"`, or `"y<->y"`.

`type` A `character` to specify the type of parameters that will be used to compute the extracted quantity. The argument can be either `"all"`, `"fixed"`, `"free"`, `"pen"`, `"effective"` (include `"free"` + `"selected"`), and `"selected"` (non-zero element of `"pen"`). If it is specified as `"default"`, it will be set as `all`.

`include_faulty` A `logical` to specify whether non-convergence or non-convexity results should be removed for penalty level selection. Non-convergence result determined by examining the maximal elements of absolute objective gradient and the number of iterations. non-convexity result is determined by checking the minimum of univariate approximate hessian.

Details

Many extract-related methods are defined to obtain quantities that can be used for further SEM applications or model diagnosis. Some of these quantities only depend on data (e.g., saturated sample covariance matrix), but some of them relies on a penalty level (e.g., gradient of objective function). An optimal penalty level can be determined by specifying a `selector` or a combination of `gamma` and `lambda`. When implementing fit-related methods, `lslx` only records necessary results for saving memory. Therefore, the extract-related methods not only extract objects but may possibly re-compute some of them. If the extracted quantity is a function of model coefficients, note that fixed coefficient is still considered as a valid variable. For example, the sub-gradient of objective function is calculated by considering both estimated coefficients and fixed coefficients. Hence, it is not appropriate to use all the elements of the sub-gradient to evaluate the optimality of solution because the values of fixed coefficients are not optimized. Fortunately, the `type` argument can be used to choose desired parameters by their types.

`$extract_specification()` returns a `data.frame` of model specification.

`$extract_saturated_cov()` returns a `list` of saturated sample covariance matrix(s).

`$extract_saturated_mean()` returns a `list` of saturated sample mean vector(s).

`$extract_saturated_moment_acov()` returns a `list` of asymptotic covariance matrix(s) of saturated moments. Note that if raw data is not available, asymptotic covariance matrix is calculated by assuming normality for data.

`$extract_penalty_level()` returns a `character` of the index name of the optimal penalty level.

`$extract_numerical_condition()` returns a `numeric` of the numerical conditions.

`$extract_information_criterion()` returns a `numeric` of the values of information criteria.

`$extract_fit_indice()` returns a `numeric` of the values of fit indices.

`$extract_coefficient()` returns a `numeric` of estimates of the coefficients.

`$extract_implied_cov()` returns a `list` of model-implied covariance matrix(s).

`$extract_implied_mean()` returns a `list` of model-implied mean vector(s).

`$extract_residual_cov()` returns a `list` of residual matrix(s) of covariance.

`$extract_residual_mean()` returns a `list` of residual vector(s) of mean.

`$extract_coefficient_matrix()` returns a `list` of coefficient matrix(s) specified by `block`.

`$extract_moment_jacobian()` returns a `matrix` of Jacobian of moment structure.

`$extract_expected_information()` returns a `matrix` of the expected Fisher information matrix.

`$extract_observed_information()` returns a `matrix` of the observed Fisher information matrix. Note that the observed information matrix is calculated via numerical differentiation for the gradient of loss.

`$extract_bfgs_hessian()` returns a `matrix` of the BFGS Hessian matrix.

`$extract_score_acov()` returns a `matrix` of the asymptotic covariance of scores.

`$extract_coefficient_acov()` returns a `matrix` of the asymptotic covariance of coefficients. For details of standard error formula, see the section of *Coefficient Evaluation*.

`$extract_loss_gradient()` returns a `matrix` of the gradient of loss function.

`$extract_regularizer_gradient()` returns a `matrix` of the sub-gradient of regularizer.

`$extract_objective_gradient()` returns a `matrix` of the sub-gradient of objective function.

Super class

`lslx::prelslx` -> `lslx`

Methods

Public methods:

- `lslx$extract_specification()`
- `lslx$extract_saturated_cov()`
- `lslx$extract_saturated_mean()`
- `lslx$extract_saturated_moment_acov()`
- `lslx$extract_lambda_grid()`

- lslx$extract_delta_grid()
- lslx$extract_weight_matrix()
- lslx$extract_penalty_level()
- lslx$extract_coefficient_indicator()
- lslx$extract_numerical_condition()
- lslx$extract_information_criterion()
- lslx$extract_fit_index()
- lslx$extract_cv_error()
- lslx$extract_coefficient()
- lslx$extract_debiased_coefficient()
- lslx$extract_implied_cov()
- lslx$extract_implied_mean()
- lslx$extract_residual_cov()
- lslx$extract_residual_mean()
- lslx$extract_coefficient_matrix()
- lslx$extract_moment_jacobian()
- lslx$extract_expected_information()
- lslx$extract_observed_information()
- lslx$extract_score_acov()
- lslx$extract_coefficient_acov()
- lslx$extract_loss_gradient()
- lslx$extract_regularizer_gradient()
- lslx$extract_objective_gradient()
- lslx$fit()
- lslx$fit_lasso()
- lslx$fit_ridge()
- lslx$fit_elastic_net()
- lslx$fit_mcp()
- lslx$fit_forward()
- lslx$fit_backward()
- lslx$fit_none()
- lslx$get_model()
- lslx$get_data()
- lslx$get_fitting()
- lslx$plot_numerical_condition()
- lslx$plot_information_criterion()
- lslx$plot_fit_index()
- lslx$plot_coefficient()
- lslx$print()
- lslx$free_block()
- lslx$fix_block()
- lslx$penalize_block()

- lslx$free_coefficient()
- lslx$fix_coefficient()
- lslx$penalize_coefficient()
- lslx$set_coefficient_type()
- lslx$set_coefficient_start()
- lslx$set_data()
- lslx$free_directed()
- lslx$fix_directed()
- lslx$penalize_directed()
- lslx$free_heterogeneity()
- lslx$fix_heterogeneity()
- lslx$penalize_heterogeneity()
- lslx$free_undirected()
- lslx$fix_undirected()
- lslx$penalize_undirected()
- lslx$summarize()
- lslx$test_lr()
- lslx$test_rmsea()
- lslx$test_coefficient()
- lslx$validate()
- lslx$clone()

Method `extract_specification()`**:**

Usage:

```
lslx$extract_specification()
```

Method `extract_saturated_cov()`**:**

Usage:

```
lslx$extract_saturated_cov()
```

Method `extract_saturated_mean()`**:**

Usage:

```
lslx$extract_saturated_mean()
```

Method `extract_saturated_moment_acov()`**:**

Usage:

```
lslx$extract_saturated_moment_acov()
```

Method `extract_lambda_grid()`**:**

Usage:

```
lslx$extract_lambda_grid()
```

Method `extract_delta_grid()`**:**

Usage:

```
lslx$extract_delta_grid()
```

Method `extract_weight_matrix()`**:**

Usage:

```
lslx$extract_weight_matrix()
```

Method `extract_penalty_level()`**:**

Usage:

```
lslx$extract_penalty_level(
  selector,
  lambda,
  delta,
  step,
  include_faulty = FALSE
)
```

Method `extract_coefficient_indicator()`**:**

Usage:

```
lslx$extract_coefficient_indicator(
  selector,
  lambda,
  delta,
  step,
  type = "default",
  include_faulty = FALSE
)
```

Method `extract_numerical_condition()`**:**

Usage:

```
lslx$extract_numerical_condition(
  selector,
  lambda,
  delta,
  step,
  include_faulty = FALSE
)
```

Method `extract_information_criterion()`**:**

Usage:

```
lslx$extract_information_criterion(
  selector,
  lambda,
  delta,
  step,
  include_faulty = FALSE
)
```

Method `extract_fit_index()`**:**

Usage:

```
lslx$extract_fit_index(selector, lambda, delta, step, include_faulty = FALSE)
```

Method `extract_cv_error()`**:**

Usage:

```
lslx$extract_cv_error(selector, lambda, delta, step, include_faulty = FALSE)
```

Method `extract_coefficient()`**:**

Usage:

```
lslx$extract_coefficient(
  selector,
  lambda,
  delta,
  step,
  type = "default",
  include_faulty = FALSE
)
```

Method `extract_debiased_coefficient()`**:**

Usage:

```
lslx$extract_debiased_coefficient(
  selector,
  lambda,
  delta,
  step,
  type = "default",
  include_faulty = FALSE
)
```

Method `extract_implied_cov()`**:**

Usage:

```
lslx$extract_implied_cov(selector, lambda, delta, step, include_faulty = FALSE)
```

Method `extract_implied_mean()`**:**

Usage:

```
lslx$extract_implied_mean(
  selector,
  lambda,
  delta,
  step,
  include_faulty = FALSE
)
```

Method `extract_residual_cov()`**:**

Usage:

```
lslx$extract_residual_cov(
  selector,
  lambda,
  delta,
  step,
  include_faulty = FALSE
)
```

Method `extract_residual_mean()`**:**

Usage:

```
lslx$extract_residual_mean(
  selector,
  lambda,
  delta,
  step,
  include_faulty = FALSE
)
```

Method `extract_coefficient_matrix()`**:**

Usage:

```
lslx$extract_coefficient_matrix(
  selector,
  lambda,
  delta,
  step,
  block,
  include_faulty = FALSE
)
```

Method `extract_moment_jacobian()`**:**

Usage:

```
lslx$extract_moment_jacobian(
  selector,
  lambda,
  delta,
  step,
  type = "default",
  include_faulty = FALSE
)
```

Method `extract_expected_information()`**:**

Usage:

```
lslx$extract_expected_information(
  selector,
  lambda,
  delta,
  step,
  type = "default",
```

```
  include_faulty = FALSE
)
```

Method `extract_observed_information()`**:**

Usage:

```
lslx$extract_observed_information(
  selector,
  lambda,
  delta,
  step,
  type = "default",
  include_faulty = FALSE
)
```

Method `extract_score_acov()`**:**

Usage:

```
lslx$extract_score_acov(
  selector,
  lambda,
  delta,
  step,
  type = "default",
  include_faulty = FALSE
)
```

Method `extract_coefficient_acov()`**:**

Usage:

```
lslx$extract_coefficient_acov(
  selector,
  lambda,
  delta,
  step,
  standard_error = "default",
  ridge_penalty = "default",
  type = "default",
  include_faulty = FALSE
)
```

Method `extract_loss_gradient()`**:**

Usage:

```
lslx$extract_loss_gradient(
  selector,
  lambda,
  delta,
  step,
  type = "default",
  include_faulty = FALSE
)
```

Method `extract_regularizer_gradient()`**:**

Usage:

```
lslx$extract_regularizer_gradient(
  selector,
  lambda,
  delta,
  step,
  type = "default",
  include_faulty = FALSE
)
```

Method `extract_objective_gradient()`**:**

Usage:

```
lslx$extract_objective_gradient(
  selector,
  lambda,
  delta,
  step,
  type = "default",
  include_faulty = FALSE
)
```

Method `fit()`**:**

Usage:

```
lslx$fit(
  penalty_method = "mcp",
  lambda_grid = "default",
  delta_grid = "default",
  step_grid = "default",
  loss = "default",
  algorithm = "default",
  missing_method = "default",
  start_method = "default",
  lambda_direction = "default",
  lambda_length = 50L,
  delta_length = 3L,
  threshold_value = 0.3,
  subset = NULL,
  cv_fold = 1L,
  iter_out_max = 100L,
  iter_in_max = 50L,
  iter_other_max = 500L,
  iter_armijo_max = 20L,
  tol_out = 0.001,
  tol_in = 0.001,
  tol_other = 1e-07,
  step_size = 1,
  momentum = 0,
```

```
  armijo = 1e-05,
  ridge_cov = 0,
  ridge_hessian = 1e-04,
  ridge_weight = 1e-04,
  warm_start = TRUE,
  positive_variance = TRUE,
  minimum_variance = 1e-04,
  armijo_rule = TRUE,
  enforce_cd = TRUE,
  random_update = TRUE,
  weight_matrix = NULL,
  verbose = TRUE
)
```

Method `fit_lasso()`**:**

Usage:

```
lslx$fit_lasso(lambda_grid = "default", ...)
```

Method `fit_ridge()`**:**

Usage:

```
lslx$fit_ridge(lambda_grid = "default", ...)
```

Method `fit_elastic_net()`**:**

Usage:

```
lslx$fit_elastic_net(lambda_grid = "default", delta_grid = "default", ...)
```

Method `fit_mcp()`**:**

Usage:

```
lslx$fit_mcp(lambda_grid = "default", delta_grid = "default", ...)
```

Method `fit_forward()`**:**

Usage:

```
lslx$fit_forward(step_grid = "default", ...)
```

Method `fit_backward()`**:**

Usage:

```
lslx$fit_backward(step_grid = "default", ...)
```

Method `fit_none()`**:**

Usage:

```
lslx$fit_none(...)
```

Method `get_model()`**:**

Usage:

```
lslx$get_model()
```

Method `get_data()`**:**

Usage:

```
lslx$get_data()
```

Method `get_fitting()`**:**

Usage:

```
lslx$get_fitting()
```

Method `plot_numerical_condition()`**:**

Usage:

```
lslx$plot_numerical_condition(
  condition,
  x_scale = "default",
  x_reverse = "default",
  mode = "default"
)
```

Method `plot_information_criterion()`**:**

Usage:

```
lslx$plot_information_criterion(
  criterion,
  x_scale = "default",
  x_reverse = "default",
  mode = "default"
)
```

Method `plot_fit_index()`**:**

Usage:

```
lslx$plot_fit_index(
  index,
  x_scale = "default",
  x_reverse = "default",
  mode = "default"
)
```

Method `plot_coefficient()`**:**

Usage:

```
lslx$plot_coefficient(
  block,
  left,
  right,
  both,
  x_scale = "default",
  x_reverse = "default",
  mode = "default"
)
```

Method `print()`**:**

Usage:

```
lslx$print()
```

Method `free_block()`**:**

Usage:

```
lslx$free_block(block, group, type, verbose = TRUE)
```

Method `fix_block()`**:**

Usage:

```
lslx$fix_block(block, group, type, verbose = TRUE)
```

Method `penalize_block()`**:**

Usage:

```
lslx$penalize_block(block, group, penalty, set, type, verbose = TRUE)
```

Method `free_coefficient()`**:**

Usage:

```
lslx$free_coefficient(name, start, verbose = TRUE)
```

Method `fix_coefficient()`**:**

Usage:

```
lslx$fix_coefficient(name, start, verbose = TRUE)
```

Method `penalize_coefficient()`**:**

Usage:

```
lslx$penalize_coefficient(name, start, penalty, set, weight, verbose = TRUE)
```

Method `set_coefficient_type()`**:**

Usage:

```
lslx$set_coefficient_type(name, type)
```

Method `set_coefficient_start()`**:**

Usage:

```
lslx$set_coefficient_start(name, start)
```

Method `set_data()`**:**

Usage:

```
lslx$set_data(data, sample_cov, sample_mean, sample_size, sample_moment_acov)
```

Method `free_directed()`**:**

Usage:

```
lslx$free_directed(left, right, group, verbose = TRUE)
```

Method `fix_directed()`**:**

Usage:

```
lslx$fix_directed(left, right, group, verbose = TRUE)
```

Method `penalize_directed()`**:**

Usage:

```
lslx$penalize_directed(left, right, group, penalty, set, verbose = TRUE)
```

Method `free_heterogeneity()`**:**

Usage:

```
lslx$free_heterogeneity(block, group, hold_fixed = TRUE, verbose = TRUE)
```

Method `fix_heterogeneity()`**:**

Usage:

```
lslx$fix_heterogeneity(block, group, hold_fixed = TRUE, verbose = TRUE)
```

Method `penalize_heterogeneity()`**:**

Usage:

```
lslx$penalize_heterogeneity(
  block,
  group,
  penalty,
  set,
  hold_fixed = TRUE,
  verbose = TRUE
)
```

Method `free_undirected()`**:**

Usage:

```
lslx$free_undirected(both, group, verbose = TRUE)
```

Method `fix_undirected()`**:**

Usage:

```
lslx$fix_undirected(both, group, verbose = TRUE)
```

Method `penalize_undirected()`**:**

Usage:

```
lslx$penalize_undirected(both, group, penalty, set, verbose = TRUE)
```

Method `summarize()`**:**

Usage:

```
lslx$summarize(
  selector,
  lambda,
  delta,
  step,
  standard_error = "default",
  ridge_penalty = "default",
  debias = "default",
  inference = "default",
```

```
  alpha_level = 0.05,
  include_faulty = FALSE,
  style = "default",
  mode = "default",
  interval = TRUE,
  digit = 3L,
  output = list(general_information = TRUE, fitting_information = FALSE,
    saturated_model_information = FALSE, baseline_model_information = FALSE,
   numerical_condition = TRUE, information_criterion = FALSE, fit_index = TRUE, cv_error
    = TRUE, lr_test = TRUE, rmsea_test = TRUE, coefficient_test = TRUE)
)
```

Method `test_lr()`**:**

Usage:

```
lslx$test_lr(selector, lambda, delta, step, include_faulty = FALSE)
```

Method `test_rmsea()`**:**

Usage:

```
lslx$test_rmsea(
  selector,
  lambda,
  delta,
  step,
  alpha_level = 0.05,
  include_faulty = FALSE
)
```

Method `test_coefficient()`**:**

Usage:

```
lslx$test_coefficient(
  selector,
  lambda,
  delta,
  step,
  standard_error = "default",
  ridge_penalty = "default",
  debias = "default",
  inference = "default",
  alpha_level = 0.05,
  include_faulty = FALSE
)
```

Method `validate()`**:**

Usage:

```
lslx$validate(
  selector,
  lambda,
  delta,
```

```
  data,
  subset = NULL,
  do_fit = "default",
  standard_error = "default",
  alpha_level = 0.05,
  include_faulty = FALSE,
  style = "default",
  mode = "default",
  interval = TRUE,
  digit = 3L
)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
lslx$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Akaike, H. (1974). A new look at the statistical model identification. IEEE Transactionson Automatic Control, 19(6), 716–723.

Bates, D., & Eddelbuettel, D. (2013). Fast and Elegant Numerical Linear Algebra Using the RcppEigen Package. Journal of Statistical Software, 52(5), 1–24.

Bentler, P. M. (1995). EQS structural equations program manual. Encino, CA: Multivariate Software.

Bentler, P. (1990). Comparative fit indices in structural models. Psychological Bulletin, 107(2), 238–246.

Berk, R., Brown, L., Buja, A., Zhang, K., & Zhao, L. (2013). Valid postselection inference. The Annals of Statistics, 41(2), 802–837.

Bozdogan, H. (1987). Model selection and Akaike's Information Criterion (AIC): The general theory and its analytical extensions. Psychometrika, 52(3), 345–370.

Browne, M. W. (1984). Asymptotic distribution-free methods for the analysis of covariance structures. British Journal of Mathematical and Statistical Psychology, 37(1), 62–83.

Chang, W. (2017). R6: Classes with Reference Semantics.

Eddelbuettel, D., & François, R. (2011). Rcpp: Seamless R and C++ Integration. Journal of Statistical Software, 40(8), 1–18.

Friedman, J., Hastie, T., & Tibshirani, R. (2010). Regularization Paths for Generalized Linear Models via Coordinate Descent. Journal of Statistical Software, 33(1), 1–22.

Haughton, D. M. A., Oud, J. H. L., & Jansen, R. A. R. G. (1997). Information and other criteria in structural equation model selection. Communications in Statistics - Simulation and Computation, 26(4), 1477–1516.

Hoerl, A. E., & Kennard, R. W. (1970). Ridge Regression: Biased Estimation for Nonorthogonal Problems. Technometrics, 12(1), 55–67.

Huang, P. H. (2018). A Penalized Likelihood Method for Multi-Group Structural Equation Modeling. British Journal of Mathematical and Statistical Psychology, 71(3), 499-522.

Huang, P. H. (2020). lslx: Semi-Confirmatory Structural Equation Modeling via Penalized Likelihood. Journal of Statistical Software. 93(7), 1-37.

Huang, P. H. (in press). Post-selection inference in Structural Equation Modeling. Multivariate Behavioral Research.

Huang, P. H., Chen, H., & Weng, L. J. (2017). A Penalized Likelihood Method for Structural Equation Modeling. Psychometrika, 82(2), 329–354.

Brosseau-Liard, P. E., Savalei, V., & Li, L. (2012). An Investigation of the Sample Performance of Two Nonnormality Corrections for RMSEA. Multivariate Behavioral Research, 47(6), 904-930.

Lee, J. D., Sun, D. L., Sun, Y., & Taylor, J. E. (2016). Exact postselection inference, with application to the lasso. The Annals of Statistics, 44(3), 907–927.

Li, L., & Bentler, P. M. (2006). Robust statistical tests for evaluating the hypothesis of close fit of misspecified mean and covariance structural models. UCLA Statistics Preprint #506. Los Angeles: University of California.

Mazumder, R., Friedman, J. H., & Hastie, T. (2011). SparseNet: Coordinate Descent With Nonconvex Penalties. Journal of the American Statistical Association, 106(495), 1125–1138.

McDonald, R. P., & Hartmann, W. M. (1992). A procedure for obtaining initial values of parameters in the RAM model. Multivariate Behavioral Research, 27(1), 57–76.

Pötscher, B. M. (1991). Effects of model selection on inference. Econometric Theory, 7(2), 163-185.

Rosseel, Y. (2012). lavaan: An R Package for Structural Equation Modeling. Journal of Statistical Software, 48(2), 1–36.

Rubin, D. B. (1976). Inference and Missing Data. Biometrika, 63(3), 581–592.

Satorra, A., & Bentler, P. M. (1994). Corrections to test statistics and standard errors in covariance structure analysis. In A. von Eye & C. C. Clogg (Eds.), Latent variable analysis: Applications to developmental research (pp. 399–419). Thousand Oaks, CA: Sage.

Savalei, V. & Falk, C. F. (2014). Robust two-stage approach outperforms robust full information maximum likelihood with incomplete nonnormal data. Structural Equation Modeling: A Multidisciplinary Journal, 21(2), 280-302.

Savalei, V. & Bentler, P. M. (2009). A Two-Stage Approach to Missing Data: Theory and Application to Auxiliary Variables, Structural Equation Modeling: A Multidisciplinary Journal, 16(3), 477-497.

Schwarz, G. (1978). Estimating the dimension of a model. The Annals of Statistics, 6(2), 461–464.

Sclove, S. L. (1987). Application of model-selection criteria to some problems in multivariate analysis. Psychometrika, 52(3), 333–343.

Steiger, J. H. (1998). A Note on Multiple Sample Extensions of the RMSEA Fit Index. Structural Equation Modeling-a Multidisciplinary Journal, 5(4), 411–419.

Steiger, J. H., & Lind, J. C. (1980). Statistically-based tests for the number of common factors. In Paper presented at the annual meeting of the Psychometric Society.

Tibshirani, R. (1996). Regression Selection and Shrinkage via the Lasso. Journal of the Royal Statistical Society B, 58(1), 267–288.

Tucker, L. R., & Lewis, C. (1973). A reliability coefficient for maximum likelihood factor analysis. Psychometrika, 38(1), 1–10.

Yuan, K.-H., & Bentler, P. M. (2000). Three likelihood-based methods for mean and covariance structure analysis with nonnormal missing data. Sociological Methodology, 30(1), 165–200.

Yuan, K. H., & Hayashi, K. (2006). Standard errors in covariance structure models: Asymptotics versus bootstrap. British Journal of Mathematical and Statistical Psychology, 59(2), 397–417.

Yuan, K.-H., & Lu, L. (2008). SEM with missing data and unknown population distributions using two-stage ML: Theory and its application. Multivariate Behavioral Research, 43(4), 621–652.

Yuan, G. X., Ho, C. H., & Lin, C. J. (2012). An Improved GLMNET for L1-regularized Logistic Regression. Journal of Machine Learning Research, 13(1), 1999–2030.

Zhang, C. H. (2010). Nearly unbiased variable selection under minimax concave penalty. Annals of Statistics, 38(2), 894–942.

Zou, H., & Hastie, T. (2005). Regularization and Variable Selection via the Elastic Net. Journal of the Royal Statistical Society B, 67(2), 301–320.

Examples

```
## EXAMPLE: Regression Analysis with Lasso Penalty ##
# run `vignette("regression-analysis")` to see the vignette
# generate data for regression analysis
set.seed(9487)
x <- matrix(rnorm(2000), 200, 10)
colnames(x) <- paste0("x", 1:10)
y <- matrix(rnorm(200), 200, 1)
data_reg <- data.frame(y, x)

# specify regression model with penalized coefficients
model_reg <- "y <= x1 + x2 + x3 + x4
              y <~ x5 + x6 + x7 + x8 + x9 + x10"

# initialize lslx object via specified model and raw data
lslx_reg <- lslx$new(model = model_reg,
                     data = data_reg)

# fit specified model to data with lasso under specified penalty levels
lslx_reg$fit(penalty_method = "lasso",
             lambda_grid = seq(.00, .30, .02))

# summarize fitting result under penalty level selected by 'aic'
lslx_reg$summarize(selector = "aic")


## EXAMPLE: Semi-Confirmatory Factor Analysis ##
# run `vignette("factor-analysis")` to see the vignette
# specify semi-confirmatory factor analysis model
model_fa <- "visual  :=> x1 + x2 + x3
             textual :=> x4 + x5 + x6
             speed   :=> x7 + x8 + x9
             visual  :~> x4 + x5 + x6 + x7 + x8 + x9
             textual :~> x1 + x2 + x3 + x7 + x8 + x9
```

```
              speed   :~> x1 + x2 + x3 + x4 + x5 + x6
              visual  <=> 1 * visual
              textual <=> 1 * textual
              speed   <=> 1 * speed"

# initialize lslx object via specified model and raw data
lslx_fa <- lslx$new(model = model_fa,
                    data = lavaan::HolzingerSwineford1939)

# fit with mcp under specified penalty levels and convexity levels
lslx_fa$fit(penalty_method = "mcp",
            lambda_grid = seq(.02, .60, .02),
            delta_grid = c(1.5, 3.0, Inf))

# summarize fitting result under penalty level selected by 'bic'
lslx_fa$summarize(selector = "bic")


## EXAMPLE: Semi-Confirmatory Structural Equation Modeling ##
# run `vignette("structural-equation-modeling")` to see the vignette
# specify structural equation modeling model
model_sem <- "fix(1) * x1 + x2 + x3   <=: ind60
              fix(1) * y1 + y2 + y3 + y4 <=: dem60
              fix(1) * y5 + y6 + y7 + y8 <=: dem65
              dem60 <= ind60
              dem65 <= ind60 + dem60"

# initialize lslx object via specified model and sample moments
lslx_sem <- lslx$new(model = model_sem,
                     sample_cov = cov(lavaan::PoliticalDemocracy),
                     sample_size = nrow(lavaan::PoliticalDemocracy))

# set some covariances of errors as penalized
lslx_sem$penalize_coefficient(name = c("y1<->y5",
                                       "y2<->y4",
                                       "y2<->y6",
                                       "y3<->y7",
                                       "y4<->y8",
                                       "y6<->y8"))

# fit with lasso under default penalty levels
lslx_sem$fit_lasso(lambda_length = 25)

# summarize fitting result under penalty level selected by 'abic'
lslx_sem$summarize(selector = "abic")


## EXAMPLE: Factor Analysis with Missing Data ##
# run `vignette("missing-data-analysis")` to see the vignette
# create missing values for x5 and x9 by the code in package semTools
data_miss <- lavaan::HolzingerSwineford1939
data_miss$x5 <- ifelse(data_miss$x1 <= quantile(data_miss$x1, .3),
                       NA, data_miss$x5)
```

```
data_miss$age <- data_miss$ageyr + data_miss$agemo/12
data_miss$x9 <- ifelse(data_miss$age <= quantile(data_miss$age, .3),
                       NA, data_miss$x9)

# specify confirmatory factor analysis model
model_miss <- "visual  :=> x1 + x2 + x3
               textual :=> x4 + x5 + x6
               speed   :=> x7 + x8 + x9
               visual  <=> 1 * visual
               textual <=> 1 * textual
               speed   <=> 1 * speed"

# "ageyr" and "agemo" are set as auxiliary variables
lslx_miss <- lslx$new(model = model_miss,
                      data = data_miss,
                      auxiliary_variable = c("ageyr", "agemo"))

# penalize all covariances among residuals
lslx_miss$penalize_block(block = "y<->y",
                         type = "fixed",
                         verbose = FALSE)

# fit with lasso under default penalty levels
lslx_miss$fit_lasso(lambda_length = 25)

# summarize fitting result under penalty level selected by 'raic'
lslx_miss$summarize(selector = "raic")


## EXAMPLE: Multi-Group Factor Analysis ##
# run `vignette("multi-group-analysis")` to see the vignette
# specify multi-group factor analysis model
model_mgfa <- "visual  :=> 1 * x1 + x2 + x3
               textual :=> 1 * x4 + x5 + x6
               speed   :=> 1 * x7 + x8 + x9"

# "school" is set as group variable and "Pasteur" is specified as reference
lslx_mgfa <- lslx$new(model = model_mgfa,
                      data = lavaan::HolzingerSwineford1939,
                      group_variable = "school",
                      reference_group = "Pasteur")

# penalize increment components of loadings and intercepts in 'Grant-White'
lslx_mgfa$penalize_heterogeneity(block = c("y<-1", "y<-f"),
                                 group = "Grant-White")

# free increment components of means of latent factors in 'Grant-White'
lslx_mgfa$free_block(block = "f<-1",
                     group = "Grant-White")

# fit with mcp under default penalty levels and specified convexity levels
lslx_mgfa$fit_mcp(lambda_length = 25)
```

```
# summarize fitting result under penalty level selected by 'hbic'
lslx_mgfa$summarize(selector = "hbic")
```

---

plsem *S3 interface for semi-confirmatory SEM via PL*

Description

`plsem()` is an S3 interface for obaining a fitted `lslx` object.

Usage

```
plsem(
  model,
  data,
  penalty_method = "mcp",
  lambda_grid = "default",
  delta_grid = "default",
  numeric_variable,
  ordered_variable,
  weight_variable,
  auxiliary_variable,
  group_variable,
  reference_group,
  sample_cov,
  sample_mean,
  sample_size,
  sample_moment_acov,
  verbose = TRUE,
  ...
)
```

Arguments

`model`	A `character` with length one to represent the model specification.
`data`	A `data.frame` of raw data. It must contains variables specified in `model` (and possibly the variables specified by `group_variable` and `weight_variable`).
`penalty_method`	A `character` to specify the penalty method. The current version supports `"none"`, `"lasso"`, `"ridge"`, `"elastic"`, and `"mcp"`.
`lambda_grid`	A non-negative `numeric` to specify penalty levels for both `"lasso"` and `"mcp"`. If it is set as `"default"`, its value will be generated automatically based on the variable scales.

`delta_grid` A non-negative `numeric` to specify the convexity level for `"mcp"`. If it is set as `"default"`, its value will be generated automatically based on the variable scales.

`numeric_variable`
A `character` to specify which response variables should be transfromed into `numeric`.

`ordered_variable`
A `character` to specify which response variables should be transfromed into `ordered`.

`weight_variable`
A `character` with length one to specify what variable is used for sampling weight.

`auxiliary_variable`
A `character` to specify what variable(s) is used as auxiliary variable(s) for estimating saturated moments when missing data presents and two-step method is implemented. Auxiliary variable(s) must be numeric. If any categorical auxiliary is considered, please transform it into dummy variables before initialization.

`group_variable` A `character` with length one to specify what variable is used for labeling group.

`reference_group`
A `character` with length one to specify which group is set as reference.

`sample_cov` A numeric `matrix` (single group case) or a `list` of numeric `matrix` (multi-group case) to represent sample covariance matrixs. It must have row and column names that match the variable names specified in `model`.

`sample_mean` A `numeric` (single group case) or a `list` of `numeric` (multi-group case) to represent sample mean vectors.

`sample_size` A `numeric` (single group case) with length one or a `list` of `numeric` (multi-group case) to represent the sample sizes.

`sample_moment_acov`
A numeric `matrix` (single group case) or a `list` of numeric `matrix` (multi-group case) to represent asymptotic covariance for moments.

`verbose` A `logical` to specify whether messages made by `lslx` should be printed.

`...` Other arguments. For details, please see the documentation of `lslx`.

Value

A fitted `lslx` object

Examples

```
## EXAMPLE: Semi-Confirmatory Factor Analysis with lavaan Style ##
# specify a factor analysis model with lavaan style
model_fa <- "visual  =~ x1 + x2 + x3
             textual =~ x4 + x5 + x6
             speed   =~ x7 + x8 + x9
             pen() * visual  =~ x4 + x5 + x6 + x7 + x8 + x9
             pen() * textual =~ x1 + x2 + x3 + x7 + x8 + x9
             pen() * speed   =~ x1 + x2 + x3 + x4 + x5 + x6
```

```
              visual  ~~ 1 * visual
              textual ~~ 1 * textual
              speed   ~~ 1 * speed"

# fit with mcp under specified penalty levels and convexity levels
lslx_fa <- plsem(model = model_fa,
                 data = lavaan::HolzingerSwineford1939,
                 penalty_method = "mcp",
                 lambda_grid = seq(.02, .60, .02),
                 delta_grid = c(1.5, 3.0, Inf))

# summarize fitting result under the penalty level selected by 'bic'
summary(lslx_fa, selector = "bic")
```

`prelslx` *R6 class to obtain preliminary result for semi-confirmatory structural equation modeling*

Description

R6 class to obtain preliminary result for semi-confirmatory structural equation modeling

R6 class to obtain preliminary result for semi-confirmatory structural equation modeling

Value

Object of `prelslx` R6 class.

Methods

Public methods:

- `prelslx$new()`
- `prelslx$prefit()`
- `prelslx$clone()`

Method `new()`:

Usage:

```
prelslx$new(
  model,
  data,
  numeric_variable,
  ordered_variable,
  weight_variable,
  auxiliary_variable,
  group_variable,
  reference_group,
  sample_cov,
```

```
  sample_mean,
  sample_size,
  sample_moment_acov,
  verbose = TRUE
)
```

Method `prefit()`**:**

Usage:

```
prelslx$prefit(
  penalty_method = "mcp",
  lambda_grid = "default",
  delta_grid = "default",
  step_grid = "default",
  loss = "default",
  algorithm = "default",
  missing_method = "default",
  start_method = "default",
  lambda_direction = "default",
  lambda_length = 50L,
  delta_length = 3L,
  threshold_value = 0.3,
  subset = NULL,
  cv_fold = 1L,
  iter_out_max = 100L,
  iter_in_max = 50L,
  iter_other_max = 500L,
  iter_armijo_max = 100L,
  tol_out = 0.001,
  tol_in = 0.001,
  tol_other = 1e-07,
  step_size = 0.5,
  momentum = 0,
  armijo = 1e-05,
  ridge_cov = 0,
  ridge_hessian = 1e-04,
  ridge_weight = 1e-04,
  warm_start = TRUE,
  positive_variance = TRUE,
  minimum_variance = 1e-04,
  armijo_rule = TRUE,
  enforce_cd = TRUE,
  random_update = TRUE,
  weight_matrix = NULL,
  verbose = TRUE
)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
prelslx$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

`residuals.lslx` *S3 method to extract residual moments from* `lslx`

Description

`residuals.lslx()` is an `S3` interface for extracting residuals from a `lslx` object.

Usage

```
## S3 method for class 'lslx'
residuals(object, selector, lambda, delta, ...)
```

Arguments

`object`	A fitted `lslx` object.
`selector`	A `character` to specify a selector for determining an optimal penalty level. Its value can be any one in `"aic"`, `"aic3"`, `"caic"`, `"bic"`, `"abic"`, `"hbic"`, or their robust counterparts `"raic"`, `"raic3"`, `"rcaic"`, `"rbic"`, `"rabic"`, `"rhbic"` if raw data is available.
`lambda`	A `numeric` to specific a chosen optimal penalty level. If the specified `lambda` is not in `lambda_grid`, a nearest legitimate value will be used.
`delta`	A `numeric` to specific a chosen optimal convexity level. If the specified `delta` is not in `delta_grid`, a nearest legitimate value will be used.
`...`	Other arguments. For details, please see the `$extracted_residual_mean()` and the `$extracted_residual_cov()` methods in `lslx`.

`summary.lslx` *S3 method to summarize* `lslx` *fitting results*

Description

`summary.lslx()` is an `S3` interface for summarizing `lslx` fitting results.

Usage

```
## S3 method for class 'lslx'
summary(object, selector, lambda, delta, ...)
```

Arguments

object	A fitted lslx object.
selector	A character to specify a selector for determining an optimal penalty level.
lambda	A numeric to specify a chosen optimal lambda value.
delta	A numeric to specify a chosen optimal lambda value.
...	Other arguments. For details, please see the $summarize() method in lslx.

---

vcov.lslx *S3 method to extract covariance matrix of estimates from* lslx

---

Description

vcov.lslx() is an S3 interface for extracting covariance matrix of parameter estimate from a lslx object.

Usage

```
## S3 method for class 'lslx'
vcov(object, selector, lambda, delta, ...)
```

Arguments

object	A fitted lslx object.
selector	A character to specify a selector for determining an optimal penalty level. Its value can be any one in "aic", "aic3", "caic", "bic", "abic", "hbic", or their robust counterparts "raic", "raic3", "rcaic", "rbic", "rabic", "rhbic" if raw data is available.
lambda	A numeric to specific a chosen optimal penalty level. If the specified lambda is not in lambda_grid, a nearest legitimate value will be used.
delta	A numeric to specific a chosen optimal convexity level. If the specified delta is not in delta_grid, a nearest legitimate value will be used.
...	Other arguments. For details, please see the $extracted_coefficient_acov() method in lslx.

Index