Package ‘regsem’

February 19, 2020

Type Package
Title Regularized Structural Equation Modeling
Version 1.5.2
Author Ross Jacobucci[aut,cre],
        Kevin J. Grimm [ctb],
        Andreas M. Brandmaier [ctb],
        Sarfaraz Serang [ctb],
        Rogier A. Kievit [ctb],
        Florian Scharf [ctb],
        Xiaobei Li [ctb]
Maintainer Ross Jacobucci <rcjacobuc@gmail.com>
Description Uses both ridge and lasso penalties (and extensions) to penalize
        specific parameters in structural equation models. The package offers additional
        cost functions, cross validation, and other extensions beyond traditional structural
        equation models. Also contains a function to perform exploratory mediation (XMed).
License GPL (>= 2)
LazyData TRUE
VignetteBuilder knitr
Depends lavaan, Rcpp, Rsolnp
Suggests snowfall, MASS, GA, caret, glmnet, ISLR, lbfgs, numDeriv,
        psych, knitr, nloptr, NlcOptim, optimx, semPlot, colorspace,
        plyr, matrixStats, stringr
LinkingTo Rcpp, RcppArmadillo
RoxygenNote 6.1.1
NeedsCompilation yes
Repository CRAN
Date/Publication 2020-02-19 12:00:03 UTC

R topics documented:

    cv_regsem ................................................................. 2
cv_regsem

The main function that runs multiple penalty values.

Description

The main function that runs multiple penalty values.

Usage

cv_regsem(model, n.lambda = 40, pars_pen = "regressions",
metric = ifelse(fit.ret2 == "train", "BIC", "chisq"),
mult.start = FALSE, multi.iter = 10, jump = 0.01,
lambda.start = 0, alpha = 0.5, gamma = 3.7, type = "lasso",
random.alpha = 0.5, fit.ret = c("rmsea", "BIC", "chisq"),
fit.ret2 = "train", n.boot = 20, data = NULL,
optMethod = "rsolnp", gradFun = "ram", hessFun = "none",
test.cov = NULL, test.n.obs = NULL, prerun = FALSE,
parallel = FALSE, ncore = 2, Start = "lavaan", subOpt = "nlminb",
diff_par = NULL, LB = -Inf, UB = Inf, par.lim = c(-Inf, Inf),
block = TRUE, full = TRUE, calc = "normal", max.iter = 2000,
tol = 1e-05, round = 3, solver = FALSE, quasi = FALSE,
solver.maxit = 5, alpha.inc = FALSE, step = 0.1,
momentum = FALSE, step.ratio = FALSE, line.search = FALSE,
nlminb.control = list(), warm.start = FALSE, missing = "listwise",
verbose = TRUE, ...)

Arguments

model: Lavaan output object. This is a model that was previously run with any of the lavaan main functions: cfa(), lavaan(), sem(), or growth(). It also can be from the efaUnrotate() function from the semTools package. Currently, the parts of the model which cannot be handled in regsem is the use of multiple group models, missing other than listwise, thresholds from categorical variable models, the use of additional estimators other than ML, most notably WLSMV for categorical variables. Note: the model does not have to actually run (use do.fit=FALSE), converge etc... regsem() uses the lavaan object as more of a parser and to get sample covariance matrix.

n.lambda: number of penalization values to test.

pars_pen: Parameter indicators to penalize. There are multiple ways to specify. The default is to penalize all regression parameters ("regressions"). Additionally, one can specify all loadings ("loadings"), or both c("regressions","loadings"). Next, parameter labels can be assigned in the lavaan syntax and passed to pars_pen. See the example. Finally, one can take the parameter numbers from the A or S matrices and pass these directly. See extractMatrices(lav.object)$A.

metric: Which fit index to use to choose a final model? Note that it chooses the best fit that also achieves convergence (conv=0).

mult.start: Logical. Whether to use multi_optim() (TRUE) or regsem() (FALSE).

multi.iter: maximum number of random starts for multi_optim

jump: Amount to increase penalization each iteration.

lambda.start: What value to start the penalty at

alpha: Mixture for elastic net. 1 = ridge, 0 = lasso

gamma: Additional penalty for MCP and SCAD

type: Penalty type. Options include "none", "lasso", "ridge", "enet" for the elastic net, "alasso" for the adaptive lasso and "diff_lasso". diff_lasso penalizes the discrepancy between parameter estimates and some pre-specified values. The values to take the deviation from are specified in diff_par. Two methods for sparser results than lasso are the smooth clipped absolute deviation, "scad", and the minimum concave penalty, "mcp". Last option is "rlasso" which is the randomised lasso to be used for stability selection.

random.alpha: Alpha parameter for randomised lasso. Has to be between 0 and 1, with a default of 0.5. Note this is only used for "rlasso", which pairs with stability selection.

fit.ret: Fit indices to return.

fit.ret2: Return fits using only dataset "train" or bootstrap "boot"? Have to do 2 sample CV manually.

n.boot: Number of bootstrap samples if fit.ret2="boot"

data: Optional dataframe. Only required for missing="fiml".

optMethod: Solver to use. Two main options for use: rsolnp and coord_desc. Although slightly slower, rsolnp works much better for complex models. coord_desc uses gradient descent with soft thresholding for the type of of penalty. Rsolnp is a nonlinear solver that doesn't rely on gradient information. There is a similar type
of solver also available for use, slsqp from the nloptr package. coord_desc can
also be used with hessian information, either through the use of quasi=TRUE,
or specifying a hess_fun. However, this option is not recommended at this time.

**gradFun**
Gradient function to use. Recommended to use “ram”, which refers to the
method specified in von Oertzen & Brick (2014). Only for use with optMethod="coord_desc".

**hessFun**
hessian function to use. Currently not recommended.

**test.cov**
Covariance matrix from test dataset. Necessary for CV=T

**test.n.obs**
Number of observations in test set. Used when CV=T

**prerun**
 Logical. Use rsolnp to first optimize before passing to gradient descent? Only
for use with coord_desc

**parallel**
Logical. whether to parallelize the processes running models for all values of
lambda.

**ncore**
Number of cores to use when parallel=TRUE

**Start**
type of starting values to use.

**subOpt**
type of optimization to use in the optimx package.

**diff_par**
parameter values to deviate from.

**LB**
lower bound vector.

**UB**
upper bound vector

**par.lim**
Vector of minimum and maximum parameter estimates. Used to stop optimiza-
tion and move to new starting values if violated.

**block**
Whether to use block coordinate descent

**full**
Whether to do full gradient descent or block

**calc**
Type of calc function to use with means or not. Not recommended for use.

**max.iter**
Number of iterations for coordinate descent

**tol**
Tolerance for coordinate descent

**round**
Number of digits to round results to

**solver**
Whether to use solver for coord_desc

**quasi**
Whether to use quasi-Newton

**solver.maxit**
Max iterations for solver in coord_desc

**alpha.inc**
Whether alpha should increase for coord_desc

**step**
Step size

**momentum**
Momentum for step sizes

**step.ratio**
Ratio of step size between A and S. Logical

**line.search**
Use line search for optimization. Default is no, use fixed step size

**nlminb.control**
list of control values to pass to nlminb

**warm.start**
Whether start values are based on previous iteration. This is not recommended.

**missing**
How to handle missing data. Current options are "listwise" and "fiml".

**verbose**
Print progress bar?

... Any additional arguments to pass to regsem() or multi_optim().
Examples

## Not run:

```r
library(regsem)
# put variables on same scale for regsem
HS <- data.frame(scale(HolzingerSwineford1939[,7:15]))
mod <- '
  f =~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9
',
  outt = cfa(mod, HS)
# increase to > 25
cv.out = cv_regsem(outt, type="lasso", pars_pen = c(1:2, 6:8),
  n.lambda = 5, jump = 0.01)
# check parameter numbers
extractMatrices(outt)["A"]
# equivalent to
mod <- '
  f =~ l1*x1 + l2*x2 + l3*x3 + l4*x4 + l5*x5 + l6*x6 + l7*x7 + l8*x8 + l9*x9
',
  outt = cfa(mod, HS)
# increase to > 25
cv.out = cv_regsem(outt, type="lasso", pars_pen = c("l1", "l2", "l6", "l7", "l8"),
  n.lambda = 5, jump = 0.01)
summary(cv.out)
plot(cv.out, show.minimum = "BIC")
mod <- '
  f =~ x1 + x2 + x3 + x4 + x5 + x6
',
  outt = cfa(mod, HS)
# can penalize all loadings
cv.out = cv_regsem(outt, type="lasso", pars_pen = "loadings",
  n.lambda = 5, jump = 0.01)
mod2 <- '
  f =~ x4+x5+x3
  x1 ~ x7 + x8 + x9 + x2
  x1 ~ f
  x2 ~ f
',
  outt2 = cfa(mod2, HS)
extractMatrices(outt2)$A
# if no pars_pen specification, defaults to all
# regressions
cv.out = cv_regsem(outt2, type="lasso",
  n.lambda = 15, jump = 0.03)
# check
cv.out$pars_pen

## End(Not run)
```
det_range

Determine the initial range for stability selection

Description
This function perform regsem on bootstrap samples to determine the initial range for stability selection. Interquartile range of the bootstrap optimal regularization amounts are used as the final range.

Usage
```
det_range(data, model, times = 50, ...)
```

Arguments
- `data` data frame
- `model` lavaan output object.
- `times` number of bootstrap samples used.
- `...` Any additional arguments to pass to regsem() or cv_regsem().

Value
result the lambda values and the upper bound and lower bound of the interquartile range.

---

det_range_par

Determine the initial range for stability selection, parallel version

Description
This function perform regsem on bootstrap samples to determine the initial range for stability selection. Interquartile range of the bootstrap optimal regularization amounts are used as the final range. Parallelization is used to achieve faster performance.

Usage
```
det_range_par(data, model, times = 50, ...)
```

Arguments
- `data` data frame
- `model` lavaan output object.
- `times` number of bootstrap samples used.
- `...` Any additional arguments to pass to regsem() or cv_regsem().

Value
result the lambda values and the upper bound and lower bound of the interquartile range.
efaModel

Generates an EFA model to be used by lavaan and regsem Function created by Florian Scharf for the paper Should regularization replace simple structure rotation in Exploratory Factor Analysis – Scharf & Nestler (in press at SEM)

**Description**

Generates an EFA model to be used by lavaan and regsem Function created by Florian Scharf for the paper Should regularization replace simple structure rotation in Exploratory Factor Analysis – Scharf & Nestler (in press at SEM)

**Usage**

```r
efaModel(nFactors, variables)
```

**Arguments**

- `nFactors`: Number of latent factors to generate.
- `variables`: Names of variables to be used as indicators

**Examples**

```r
## Not run:
HS <- data.frame(scale(HolzingerSwineford1939[,7:15]))
# Note to find number of factors, recommended to use
# fa.parallel() from the psych package
# using the wrong number of factors can distort the results
mod = efaModel(3, colnames(HS))
semFit = sem(mod, data = HS, int.ov.free = FALSE, int.lv.free = FALSE,
std.lv = TRUE, std.ov = TRUE, auto.fix.single = FALSE, se = "none")

# note it requires smaller penalties than other applications
reg.out2 = cv_regsem(model = semFit, pars_pen = "loadings",
mult.start = TRUE, multi.iter = 10,
n.lambda = 100, type = "lasso", jump = 10^-5, lambda.start = 0.001)
reg.out2
plot(reg.out2) # note that the solution jumps around -- make sure best fit makes sense
```

## End(Not run)
extractMatrices  This function extracts RAM matrices from a lavaan object.

Description

This function extracts RAM matrices from a lavaan object.

Usage

extractMatrices(model)

Arguments

model  Lavaan model object.

Value

The RAM matrices from model.

Examples

library(lavaan)
data(HolzingerSwineford1939)
HS.model <- 'visual =~ x1 + x2 + x3
textual =~ x4 + x5 + x6
speed =~ x7 + x8 + x9'
mod <- cfa(HS.model, data=HolzingerSwineford1939)
mats = extractMatrices(mod)

fit_indices  Calculates the fit indices

Description

Calculates the fit indices

Usage

fit_indices(model, CV = F, CovMat = NULL, data = NULL, n.obs = NULL)
Arguments

model: regsem model object.

CV: cross-validation. Note that this requires splitting the dataset into a training and test set prior to running the model. The model should be run on the training set, with the test set held out and then passed to CovMat=.

CovMat: If CV=T then test covariance matrix must be supplied. Note that this should be done before running the lavaan model and should not overlap with the data or covariance matrix used to run the model.

data: supply the dataset?
n.obs: Number of observations in the test set for CV.

Examples

## Not run:
fit_indices()
## End(Not run)

multi_optim

Multiple starts for Regularized Structural Equation Modeling

Description

Multiple starts for Regularized Structural Equation Modeling

Usage

multi_optim(model, max.try = 10, lambda = 0, alpha = 0.5,
gamma = 3.7, random.alpha = 0.5, LB = -Inf, UB = Inf,
par.lim = c(-Inf, Inf), block = TRUE, full = TRUE,
type = "lasso", optMethod = "rsolnp", gradFun = "ram",
pars_pen = "regressions", diff_par = NULL, hessFun = "none",
tol = 1e-05, round = 3, solver = FALSE, quasi = FALSE,
solver.maxit = 50000, alpha.inc = FALSE, line.search = FALSE,
prerun = FALSE, step = 0.1, momentum = FALSE, step.ratio = FALSE,
verbose = FALSE, warm.start = FALSE, Start2 = NULL,
nlminb.control = NULL, max.iter = 500)

Arguments

model: Lavaan output object. This is a model that was previously run with any of the lavaan main functions: cfa(), lavaan(), sem(), or growth(). It also can be from the efaUnrotate() function from the semTools package. Currently, the parts of the model which cannot be handled in regsem is the use of multiple group models, missing other than listwise, thresholds from categorical variable models, the use of additional estimators other than ML, and most notably WLSMV for categorical
variables. Note: the model does not have to actually run (use do.fit=FALSE), converge etc... regsem() uses the lavaan object as more of a parser and to get sample covariance matrix.

max.try number of starts to try before convergence.
lambda Penalty value. Note: higher values will result in additional convergence issues.
alpha Mixture for elastic net.
gamma Additional penalty for MCP and SCAD
random.alpha Alpha parameter for randomised lasso. Has to be between 0 and 1, with a default of 0.5. Note this is only used for "rlasso", which pairs with stability selection.
LB lower bound vector. Note: This is very important to specify when using regularization. It greatly increases the chances of converging.
UB Upper bound vector
par.lim Vector of minimum and maximum parameter estimates. Used to stop optimization and move to new starting values if violated.
block Whether to use block coordinate descent
full Whether to do full gradient descent or block
type Penalty type. Options include "none", "lasso", "enet" for the elastic net, "alasso" for the adaptive lasso and "diff_lasso". If ridge penalties are desired, use type="enet" and alpha=1. diff_lasso penalizes the discrepancy between parameter estimates and some pre-specified values. The values to take the deviation from are specified in diff_par. Two methods for sparser results than lasso are the smooth clipped absolute deviation, "scad", and the minimum concave penalty, "mcp". Last option is "rlasso" which is the randomised lasso to be used for stability selection.

optMethod Solver to use. Two main options for use: rsolnp and coord_desc. Although slightly slower, rsolnp works much better for complex models. coord_desc uses gradient descent with soft thresholding for the type of of penalty. Rsolnp is a nonlinear solver that doesn’t rely on gradient information. There is a similar type of solver also available for use, slsqp from the nloptr package. coord_desc can also be used with hessian information, either through the use of quasi=TRUE, or specifying a hess_fun. However, this option is not recommended at this time.

gradFun Gradient function to use. Recommended to use "ram", which refers to the method specified in von Oertzen & Brick (2014). Only for use with optMethod="coord_desc".
pars_pen Parameter indicators to penalize. There are multiple ways to specify. The default is to penalize all regression parameters ("regressions"). Additionally, one can specify all loadings ("loadings"), or both c("regressions","loadings"). Next, parameter labels can be assigned in the lavaan syntax and passed to pars_pen. See the example. Finally, one can take the parameter numbers from the A or S matrices and pass these directly. See extractMatrices(lav.object)$A.

diff_par Parameter values to deviate from. Only used when type="diff_lasso".
hessFun Hessian function to use. Currently not recommended.
tol Tolerance for coordinate descent
round Number of digits to round results to
multi_optim

solver  Whether to use solver for coord_desc
quasi   Whether to use quasi-Newton. Currently not recommended.
solver.maxit Max iterations for solver in coord_desc
alpha.inc Whether alpha should increase for coord_desc
line.search Use line search for optimization. Default is no, use fixed step size
prerun   Logical. Use rsolnp to first optimize before passing to gradient descent? Only for use with coord_desc.
step    Step size
momentum Momentum for step sizes
step.ratio Ratio of step size between A and S. Logical
verbose Whether to print iteration number.
warm.start Whether start values are based on previous iteration. This is not recommended.
Start2  Provided starting values. Not required
nlminb.control list of control values to pass to nlminb
max.iter Number of iterations for coordinate descent

Examples

## Not run:
# Note that this is not currently recommended. Use cv_regsem() instead
library(regsem)
# put variables on same scale for regsem
HS <- data.frame(scale(HolzingerSwineford1939[,7:15]))
mod <- '
  f =~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9
',
outt = cfa(mod, HS, meanstructure=TRUE)

fit1 <- multi_optim(outt, max.try=40,
  lambda=0.1, type="lasso")

# growth model
model <- ' i =- 1*t1 + 1*t2 + 1*t3 + 1*t4
  s =- 0*t1 + s1*t2 + s2*t3 + 3*t4 '
fit <- growth(model, data=Demo.growth)
summary(fit)
fitmeasures(fit)
fit3 <- multi_optim(fit, lambda=0.2, type="lasso")
summary(fit3)

## End(Not run)
**parse_parameters**

*Takes either a vector of parameter ids or a vector of named parameters and returns a vector of parameter ids*

**Description**

Takes either a vector of parameter ids or a vector of named parameters and returns a vector of parameter ids

**Usage**

```r
parse_parameters(x, model)
```

**Arguments**

- `x` Parameter labels
- `model` Lavaan model

**Value**

NULL if undefined input. Else vector of parameter ids

---

**pen_mod**

*Penalized model syntax.*

**Description**

This function create a lavaan model syntax with paths corresponding to paremeters penalized to 0 removed.

**Usage**

```r
pen_mod(model, nm = NULL, pars_pen = NULL)
```

**Arguments**

- `model` lavaan output object.
- `nm` names(regsemOutput$coefficients).
- `pars_pen` a vector of numbers corresponding to paths to be removed (same sequence as regsemOutput$coefficients).

**Value**

new.mod new model in lavaan syntax.
Plot function for cv_regsem

Description

Plot function for cv_regsem

Usage

## S3 method for class 'cvregsem'
plot(x, ..., pars = NULL, show.minimum = "BIC",
col = NULL, type = "l", lwd = 3, h_line = 0, lty = 1,
xlab = NULL, ylab = NULL, legend.x = NULL, legend.y = NULL,
legend.cex = 1, legend.bg = par("bg"), grey.out = FALSE)

Arguments

x An x from cv_regsem.
... Other arguments.
pars Which parameters to plot
show.minimum What fit index to use
col A specification for the default plotting color.
type what type of plot should be drawn. Possible types are "p" for points, "l" for lines, or "b" for both
lwd line width
h_line Where to draw horizontal line
lty line type
xlab X axis label
ylab Y axis label
legend.x x-coordinate of legend. See ?legend
legend.y y-coordinate of legend. See ?legend
legend.cex cex of legend. See ?legend
legend.bg legend background color. See ?legend
grey.out Add grey to background
**rcpp_fit_fun**

*Calculates the objective function values.*

**Description**

Calculates the objective function values.

**Usage**

```r
rcpp_fit_fun(ImpCov, SampCov, type2, lambda, gamma, pen_vec, pen_diff,
            e_alpha, rlasso_pen)
```

**Arguments**

- **ImpCov**: expected covariance matrix.
- **SampCov**: Sample covariance matrix.
- **type2**: penalty type.
- **lambda**: penalty value.
- **gamma**: additional penalty for mcp and scad
- **pen_vec**: vector of penalized parameters.
- **pen_diff**: Vector of values to take deviation from.
- **e_alpha**: Alpha for elastic net
- **rlasso_pen**: Alpha for rlasso2

---

**rcpp_grad_ram**

*Calculates the gradient vector based on Von Oertzen \\ Brick, 2014*

**Description**

Calculates the gradient vector based on Von Oertzen \\ Brick, 2014

**Usage**

```r
rcpp_grad_ram(par, ImpCov, SampCov, Areg, Sreg, A, S, F, lambda, type2,
              pen_vec, diff_par)
```
Arguments

par vector with parameters.
ImpCov expected covariance matrix.
SampCov Sample covariance matrix.
Areg A matrix with current parameter estimates.
Sreg S matrix with current parameter estimates.
A A matrix with parameter labels.
S S matrix with parameter labels.
F F matrix.
lambda penalty value.
type2 penalty type.
pen_vec parameter indicators to be penalized.
diff_par parameter values to take deviations from.

rcpp_quasi_calc Compute quasi Hessian

Description

Compute quasi Hessian

Usage

rcpp_quasi_calc(I, s, y, H)

Arguments

I identity matrix.
 s s vector.
 y y vector.
 H previous Hessian.
**rcpp_RAMmult**

*Take RAM matrices, multiplies, and returns Implied Covariance matrix.*

**Description**

Take RAM matrices, multiplies, and returns Implied Covariance matrix.

**Usage**

```r
rcpp_RAMmult(par, A, S, S_fixed, A_fixed, A_est, S_est, F, I)
```

**Arguments**

- `par`: parameter estimates.
- `A`: A matrix with parameter labels.
- `S`: S matrix with parameter labels.
- `S_fixed`: S matrix with fixed indicators.
- `A_fixed`: A matrix with fixed indicators.
- `A_est`: A matrix with parameter estimates.
- `S_est`: S matrix with parameter estimates.
- `F`: F matrix.
- `I`: Diagonal matrix of ones.

**regsem**

*Regularized Structural Equation Modeling. Tests a single penalty. For testing multiple penalties, see cv_regsem().*

**Description**

Regularized Structural Equation Modeling. Tests a single penalty. For testing multiple penalties, see `cv_regsem()`.

**Usage**

```r
regsem(model, lambda = 0, alpha = 0.5, gamma = 3.7, type = "lasso",
        random.alpha = 0.5, data = NULL, optMethod = "rsolnp",
        estimator = "ML", gradFun = "ram", hessFun = "none",
        prerun = FALSE, parallel = "no", Start = "lavaan",
        subOpt = "nlminb", longMod = F, pars_pen = "regressions",
        diff_par = NULL, LB = -Inf, UB = Inf, par.lim = c(-Inf, Inf),
        block = TRUE, full = TRUE, calc = "normal", max.iter = 500,
        tol = 1e-05, round = 3, solver = FALSE, quasi = FALSE,
        solver.maxit = 5, alpha.inc = FALSE, line.search = FALSE,
        step = 0.1, momentum = FALSE, step.ratio = FALSE,
        nlminb.control = list(), missing = "listwise")
```
Arguments

**model**
Lavaan output object. This is a model that was previously run with any of the lavaan main functions: cfa(), lavaan(), sem(), or growth(). It also can be from the efaUnrotate() function from the semTools package. Currently, the parts of the model which cannot be handled in regsem is the use of multiple group models, missing other than listwise, thresholds from categorical variable models, the use of additional estimators other than ML, most notably WLSMV for categorical variables. Note: the model does not have to actually run (use do.fit=FALSE), converge etc... regsem() uses the lavaan object as more of a parser and to get sample covariance matrix.

**lambda**
Penalty value. Note: higher values will result in additional convergence issues. If using values > 0.1, it is recommended to use multi_optim() instead. See multi_optim for more detail.

**alpha**
Mixture for elastic net. 1 = ridge, 0 = lasso

**gamma**
Additional penalty for MCP and SCAD

**type**
Penalty type. Options include "none", "lasso", "enet" for the elastic net, "alasso" for the adaptive lasso and "diff_lasso". If ridge penalties are desired, use type="enet" and alpha=1. diff_lasso penalizes the discrepancy between parameter estimates and some pre-specified values. The values to take the deviation from are specified in diff_par. Two methods for sparser results than lasso are the smooth clipped absolute deviation, "scad", and the minimum concave penalty, "mcp". Last option is "rlasso" which is the randomised lasso to be used for stability selection.

**random.alpha**
Alpha parameter for randomised lasso. Has to be between 0 and 1, with a default of 0.5. Note this is only used for "rlasso", which pairs with stability selection.

**data**
Optional dataframe. Only required for missing="fiml" which is not currently working.

**optMethod**
Solver to use. Two main options for use: rsolnp and coord_desc. Although slightly slower, rsolnp works much better for complex models. coord_desc uses gradient descent with soft thresholding for the type of of penalty. Rsolnp is a nonlinear solver that doesn't rely on gradient information. There is a similar type of solver also available for use, slsqp from the nloptr package. coord_desc can also be used with hessian information, either through the use of quasi=TRUE, or specifying a hess_fun. However, this option is not recommended at this time.

**estimator**
Whether to use maximum likelihood (ML) or unweighted least squares (ULS) as a base estimator.

**gradFun**
Gradient function to use. Recommended to use "ram", which refers to the method specified in von Oertzen & Brick (2014). Only for use with optMethod="coord_desc".

**hessFun**
Hessian function to use. Recommended to use "ram", which refers to the method specified in von Oertzen & Brick (2014). This is currently not recommended.

**prerun**
Logical. Use rsolnp to first optimize before passing to gradient descent? Only for use with coord_desc.

**parallel**
Logical. Whether to parallelize the processes?
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Start</strong></td>
<td>Type of starting values to use. Only recommended to use &quot;default&quot;. This sets factor loadings and variances to 0.5. Start = &quot;lavaan&quot; uses the parameter estimates from the lavaan model object. This is not recommended as it can increase the chances in getting stuck at the previous parameter estimates.</td>
</tr>
<tr>
<td><strong>subOpt</strong></td>
<td>Type of optimization to use in the optimx package.</td>
</tr>
<tr>
<td><strong>longMod</strong></td>
<td>If TRUE, the model is using longitudinal data? This changes the sample covariance used.</td>
</tr>
<tr>
<td><strong>pars_pen</strong></td>
<td>Parameter indicators to penalize. There are multiple ways to specify. The default is to penalize all regression parameters (&quot;regressions&quot;). Additionally, one can specify all loadings (&quot;loadings&quot;), or both c(&quot;regressions&quot;,&quot;loadings&quot;). Next, parameter labels can be assigned in the lavaan syntax and passed to pars_pen. See the example. Finally, one can take the parameter numbers from the A or S matrices and pass these directly. See extractMatrices(lav.object)$A.</td>
</tr>
<tr>
<td><strong>diff_par</strong></td>
<td>Parameter values to deviate from. Only used when type=&quot;diff_lasso&quot;.</td>
</tr>
<tr>
<td><strong>LB</strong></td>
<td>Lower bound vector. Note: This is very important to specify when using regularization. It greatly increases the chances of converging.</td>
</tr>
<tr>
<td><strong>UB</strong></td>
<td>Upper bound vector</td>
</tr>
<tr>
<td><strong>par.lim</strong></td>
<td>Vector of minimum and maximum parameter estimates. Used to stop optimization and move to new starting values if violated.</td>
</tr>
<tr>
<td><strong>block</strong></td>
<td>Whether to use block coordinate descent</td>
</tr>
<tr>
<td><strong>full</strong></td>
<td>Whether to do full gradient descent or block</td>
</tr>
<tr>
<td><strong>calc</strong></td>
<td>Type of calc function to use with means or not. Not recommended for use.</td>
</tr>
<tr>
<td><strong>max.iter</strong></td>
<td>Number of iterations for coordinate descent</td>
</tr>
<tr>
<td><strong>tol</strong></td>
<td>Tolerance for coordinate descent</td>
</tr>
<tr>
<td><strong>round</strong></td>
<td>Number of digits to round results to</td>
</tr>
<tr>
<td><strong>solver</strong></td>
<td>Whether to use solver for coord_desc</td>
</tr>
<tr>
<td><strong>quasi</strong></td>
<td>Whether to use quasi-Newton</td>
</tr>
<tr>
<td><strong>solver.maxit</strong></td>
<td>Max iterations for solver in coord_desc</td>
</tr>
<tr>
<td><strong>alpha.inc</strong></td>
<td>Whether alpha should increase for coord_desc</td>
</tr>
<tr>
<td><strong>line.search</strong></td>
<td>Use line search for optimization. Default is no, use fixed step size</td>
</tr>
<tr>
<td><strong>step</strong></td>
<td>Step size</td>
</tr>
<tr>
<td><strong>momentum</strong></td>
<td>Momentum for step sizes</td>
</tr>
<tr>
<td><strong>step.ratio</strong></td>
<td>Ratio of step size between A and S. Logical</td>
</tr>
<tr>
<td><strong>nlminb.control</strong></td>
<td>List of control values to pass to nlminb</td>
</tr>
<tr>
<td><strong>missing</strong></td>
<td>How to handle missing data. Current options are &quot;listwise&quot; and &quot;fiml&quot;. &quot;fiml&quot; is not currently working well.</td>
</tr>
</tbody>
</table>
Value

out List of return values from optimization program

convergence Convergence status. 0 = converged, 1 or 99 means the model did not converge.

par.ret Final parameter estimates

Imp_Cov Final implied covariance matrix

grad Final gradient.

KKT1 Were final gradient values close enough to 0.

KKT2 Was the final Hessian positive definite.

df Final degrees of freedom. Note that df changes with lasso penalties.

npar Final number of free parameters. Note that this can change with lasso penalties.

SampCov Sample covariance matrix.

fit Final F_ml fit. Note this is the final parameter estimates evaluated with the F_ml fit function.

coefficients Final parameter estimates

nvar Number of variables.

N sample size.

nfac Number of factors

baseline.chisq Baseline chi-square.

baseline.df Baseline degrees of freedom.

Examples

# Note that this is not currently recommended. Use cv_regsem() instead
library(lavaan)
# put variables on same scale for regsem
HS <- data.frame(scale(HolzingerSwineford1939[,7:15]))
mod <- '
f =~ 1*x1 + 11*x2 + 12*x3 + 13*x4 + 14*x5 + 15*x6 + 16*x7 + 17*x8 + 18*x9
',
# Recommended to specify meanstructure in lavaan
outt = cfa(mod, HS, meanstructure=TRUE)

fit1 <- regsem(outt, lambda=0.05, type="lasso",
  pars_pen=c("l1", "l2", "l6", "l7", "l8"))
#equivalent to pars_pen=c(1:2, 6:8)
#summary(fit1)
stabsel  

**Stability selection**

**Description**

Stability selection

**Usage**

```r
stabsel(data, model, det.range = FALSE, from, to, times = 50,
jump = 0.01, detr.nlambda = 20, n.lambda = 40, n.boot = 100,
det.thr = FALSE, p = 0.8, p.from = 0.5, p.to = 1,
p.jump = 0.05, p.method = "aic", type = "lasso",
pars_pen = "regressions", ...)
```

**Arguments**

- `data`: data frame
- `model`: lavaan syntax model.
- `det.range`: Whether to determine the range of penalization values for stability selection through bootstrapping. Default is FALSE, from and to arguments are then needed. If set to TRUE, then jump, times and detr.nlambda arguments will be needed.
- `from`: Minimum value of penalization values for stability selection.
- `to`: Maximum value of penalization values for stability selection.
- `times`: Number of bootstrapping sample used to determine the range. Default is 50.
- `jump`: Amount to increase penalization each iteration. Default is 0.01
- `detr.nlambda`: Number of penalization values to test for determing range.
- `n.lambda`: Number of penalization values to test for stability selection.
- `n.boot`: Number of bootstrap samples needed for stability selection.
- `det.thr`: Whether to determine the probability threshold value. Default is FALSE, p is then needed. If set to TRUE, p.from, p.to, p.method arguments will be needed.
- `p`: Probability threshold: above which selection probability is the path kept in the model. Default value is 0.8.
- `p.from`: Lower bound of probability threshold to test. Default is 0.5.
- `p.to`: Upper bound of probability threshold to test. Default is 1.
- `p.jump`: Amount to increase threshold each iteration. Default is 0.05.
- `p.method`: Which fit index to use to choose a final model?
- `type`: Penalty type
- `pars_pen`: Parameter indicators to penalize.
- `...`: Any additional arguments to pass to regsem() or cv_regsem().
Examples

```r
## Not run:
library(regsem)
# put variables on same scale for regsem
HS <- data.frame(scale(HolzingerSwineford1939[,7:15]))
mod <- '
  f =~ 1*x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9
  x1 ~~ r1*x2;x1 ~~ r2*x3;x1 ~~ r3*x4;x1 ~~ r4*x5
',
  outt = cfa(mod, HS)

stabsel.out = stabsel(data=HS,model=mod,det.range=T,detr.nlambda=20,n.lambda=5,
  n.boot=10,p=0.9,type="alasso",p.method="aic",
  pars_pen=c("r1","r2","r3","r4"))
stabsel.out$selection_results

## End(Not run)
```

### Description

Stability selection, parallelized version

#### Usage

```r
stabsel_par(data, model, det.range = FALSE, from, to, times = 50,
  jump = 0.01, detr.nlambda = 20, n.lambda = 40, n.boot = 100,
  det.thr = FALSE, p = 0.8, p.from = 0.5, p.to = 1,
  p.jump = 0.05, p.method = "aic", type = "lasso",
  pars_pen = "regressions", ...)
```

#### Arguments

- `data` : data frame
- `model` : lavaan syntax model.
- `det.range` : Whether to determine the range of penalization values for stability selection through bootstrapping. Default is FALSE, from and to arguments are then needed. If set to TRUE, then jump, times and detr.nlambda arguments will be needed.
- `from` : Minimum value of penalization values for stability selection.
- `to` : Maximum value of penalization values for stability selection.
- `times` : Number of bootstrapping sample used to determine the range. Default is 50.
- `jump` : Amount to increase penalization each iteration. Default is 0.01
stabsel_thr

Description
This function tune the probability threshold parameter.

Usage
stabsel_thr(stabsel = NULL, data = NULL, model = NULL,
est_model = NULL, prob = NULL, nm = NULL, pars.pen = NULL,
from = 0.5, to = 1, jump = 0.01, method = "aic")

Arguments
stabsel output object from stabsel function. If specified, data, model, est_model, prob, nm, and pars.pen parameters are not needed.
data data frame
model lavaan syntax model.
est_model lavaan output object.
prob matrix of selection probabilities.
nm names(regsemOutput$coefficients).
pars.pen a vector of numbers corresponding to paths to be removed (same sequence as regsemOutput$coefficients).
from starting value of the threshold parameter.
to end value of the threshold parameter.
jump increment of the threshold parameter.
method fit indices used to tune the parameter.
Value

rtn results using the optimal threshold.

Description

print information about cvregsem object

Usage

## S3 method for class 'cvregsem'
summary(object, ...)

Arguments

object cv_regsem object
...
Additional arguments

Summary results from regsem.

Description

Summary results from regsem.

Usage

## S3 method for class 'regsem'
summary(object, ...)

Arguments

object An object from regsem.
...
Other arguments.
Function to performed exploratory mediation with continuous and categorical variables

Usage

`xmed(data, iv, mediators, dv, covariates = NULL, type = "lasso", nfolds = 10, epsilon = 0.001, seed = NULL)`

Arguments

- `data`: Name of the dataset
- `iv`: Name of independent variable
- `mediators`: Name of mediators
- `dv`: Name of dependent variable
- `covariates`: Name of covariates to be included in model.
- `type`: What type of penalty. Options include lasso, ridge, and enet.
- `nfolds`: Number of cross-validation folds.
- `epsilon`: Threshold for determining whether effect is 0 or not.
- `seed`: Set seed to control CV results

Examples

```r
## Not run:
# example
library(ISLR)
College1 = College[which(College$Private=="Yes"),]
Data = data.frame(scale(College1[c("Grad.Rate","Accept","Outstate","Room.Board","Books","Expend")]))
Data$Grad.Rate <- ifelse(Data$Grad.Rate > 0,1,0)
Data$Grad.Rate <- as.factor(Data$Grad.Rate)
# lavaan model with all mediators
model1 <-
  ' # direct effect (c_prime)
  Grad.Rate ~ c_prime*Accept
  # mediators
  Outstate ~ a1*Accept
  Room.Board ~ a2*Accept
  Books ~ a3*Accept
  Expend ~ a6*Accept
  Grad.Rate ~ b1*Outstate + b2*Room.Board + b3*Books + b6*Expend
  # indirect effects (a*b)
  a1b1 := a1*b1
```
\[
a_{2b2} := a_2 \times b_2 \\
a_{3b3} := a_3 \times b_3 \\
a_{6b6} := a_6 \times b_6
\]

# total effect (c)
c := c_{prime} + (a_{1b1}) + (a_{2b2}) + (a_{3b3}) + (a_{6b6})

# p-value approach using delta method standard errors
fit.delta = sem(model1, data=Data, fixed.x=TRUE, ordered="Grad.Rate")
summary(fit.delta)

xmed()

iv <- "Accept"
dv <- "Grad.Rate"
mediators <- c("Outstate", "Room.Board", "Books", "Expend")

out <- xmed(Data, iv, mediators, dv)
out

## End(Not run)
Index

+ Topic **analysis**
  - efaModel, 7
+ Topic **calc**
  - cv_regsem, 2
  - regsem, 16
+ Topic **chisq**
  - fit_indices, 8
+ Topic **extract**
  - extractMatrices, 8
+ Topic **factor**
  - efaModel, 7
+ Topic **fa**
  - efaModel, 7
+ Topic **fit**
  - fit_indices, 8
+ Topic **multiple**
  - multi_optim, 9
+ Topic **ncp**
  - fit_indices, 8
+ Topic **optim**
  - cv_regsem, 2
  - multi_optim, 9
  - regsem, 16
+ Topic **rmsea**
  - fit_indices, 8
  - cv_regsem, 2
  - det_range, 6
  - det_range_par, 6
  - efaModel, 7
  - extractMatrices, 8
  - fit_indices, 8
  - multi_optim, 9, 17
  - parse_parameters, 12
  - pen_mod, 12
  - plot.cvregsem, 13
  - rcpp_fit_fun, 14
  - rcpp_grad_ram, 14
  - rcpp_quasi_calc, 15
  - rcpp_RAMmult, 16
  - regsem, 16
  - stabsel, 20
  - stabsel_par, 21
  - stabsel_thr, 22
  - summary.cvregsem, 23
  - summary.regsem, 23
  - xmed, 24