Package ‘regsem’

May 22, 2019

Type Package
Title Regularized Structural Equation Modeling
Version 1.3.9
Author Ross Jacobucci[aut,cre],
       Kevin J. Grimm [ctb],
       Andreas M. Brandmaier [ctb],
       Sarfaraz Serang [ctb],
       Rogier A. Kievit [ctb],
       Florian Scharf [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
LinkingTo Rcpp, RcppArmadillo
RoxygenNote 6.1.1
NeedsCompilation yes
Repository CRAN
Date/Publication 2019-05-22 17:10:03 UTC

R topics documented:

  cv_regsem .................................................. 2
efaModel ..................................................... 6
extractMatrices ............................................. 7
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, ...)
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 de-
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 ran-
domised 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.
grdFun 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.
<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>test.cov</code></td>
<td>Covariance matrix from test dataset. Necessary for CV=T</td>
</tr>
<tr>
<td><code>test.n.obs</code></td>
<td>Number of observations in test set. Used when CV=T</td>
</tr>
<tr>
<td><code>prerun</code></td>
<td>Logical. Use rsolnp to first optimize before passing to gradient descent? Only for use with coord_desc</td>
</tr>
<tr>
<td><code>parallel</code></td>
<td>Logical. whether to parallelize the processes running models for all values of lambda.</td>
</tr>
<tr>
<td><code>ncore</code></td>
<td>Number of cores to use when parallel=TRUE</td>
</tr>
<tr>
<td><code>start</code></td>
<td>type of starting values to use.</td>
</tr>
<tr>
<td><code>subOpt</code></td>
<td>type of optimization to use in the optimx package.</td>
</tr>
<tr>
<td><code>diff_par</code></td>
<td>parameter values to deviate from.</td>
</tr>
<tr>
<td><code>LB</code></td>
<td>lower bound vector.</td>
</tr>
<tr>
<td><code>UB</code></td>
<td>upper bound vector</td>
</tr>
<tr>
<td><code>par.lim</code></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><code>block</code></td>
<td>Whether to use block coordinate descent</td>
</tr>
<tr>
<td><code>full</code></td>
<td>Whether to do full gradient descent or block</td>
</tr>
<tr>
<td><code>calc</code></td>
<td>Type of calc function to use with means or not. Not recommended for use.</td>
</tr>
<tr>
<td><code>max.iter</code></td>
<td>Number of iterations for coordinate descent</td>
</tr>
<tr>
<td><code>tol</code></td>
<td>Tolerance for coordinate descent</td>
</tr>
<tr>
<td><code>round</code></td>
<td>Number of digits to round results to</td>
</tr>
<tr>
<td><code>solver</code></td>
<td>Whether to use solver for coord_desc</td>
</tr>
<tr>
<td><code>quasi</code></td>
<td>Whether to use quasi-Newton</td>
</tr>
<tr>
<td><code>solver.maxit</code></td>
<td>Max iterations for solver in coord_desc</td>
</tr>
<tr>
<td><code>alpha.inc</code></td>
<td>Whether alpha should increase for coord_desc</td>
</tr>
<tr>
<td><code>step</code></td>
<td>Step size</td>
</tr>
<tr>
<td><code>momentum</code></td>
<td>Momentum for step sizes</td>
</tr>
<tr>
<td><code>step.ratio</code></td>
<td>Ratio of step size between A and S. Logical</td>
</tr>
<tr>
<td><code>line.search</code></td>
<td>Use line search for optimization. Default is no, use fixed step size</td>
</tr>
<tr>
<td><code>nlminb.control</code></td>
<td>list of control values to pass to nlminb</td>
</tr>
<tr>
<td><code>warm.start</code></td>
<td>Whether start values are based on previous iteration. This is not recommended.</td>
</tr>
<tr>
<td><code>missing</code></td>
<td>How to handle missing data. Current options are &quot;listwise&quot; and &quot;fiml&quot;.</td>
</tr>
<tr>
<td><code>verbose</code></td>
<td>Print progress bar?</td>
</tr>
<tr>
<td>...</td>
<td>Any additional arguments to pass to regsem() or multi_optim().</td>
</tr>
</tbody>
</table>
Examples

```r
## Not run:
library(regsem)
libraryHregsemI
vignetteHoverviewI
# 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 <- 1*x1 + 11*x2 + 12*x3 + 13*x4 + 14*x5 + 15*x6 + 16*x7 + 17*x8 + 18*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)
**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**

`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",
                    multi.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

```r
extractMatrices(model)
```

### Arguments

- **model**: Lavaan model object.

### Value

The RAM matrices from `model`.

### Examples

```r
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

```r
fit_indices(model, CV = F, CovMat = NULL, data = NULL, n.obs = NULL)
```
multi_optim

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

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

### Usage

```r
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 = "rm",
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, 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
\texttt{multi_optim}

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

\textbf{Examples}

\begin{verbatim}
## Not run:
# 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)
\end{verbatim}
### 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.

### plot.cvregsem

Plot function for cv_regsem.

#### Description

Plot function for cv_regsem.

#### Usage

```r
## 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.
rcpp_fit_fun

Calculates the objective function values.

Description

Calculates the objective function values.

Usage

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

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ImpCov</td>
<td>expected covariance matrix.</td>
</tr>
<tr>
<td>SampCov</td>
<td>Sample covariance matrix.</td>
</tr>
<tr>
<td>type2</td>
<td>penalty type.</td>
</tr>
<tr>
<td>lambda</td>
<td>penalty value.</td>
</tr>
<tr>
<td>gamma</td>
<td>additional penalty for mcp and scad</td>
</tr>
<tr>
<td>pen_vec</td>
<td>vector of penalized parameters.</td>
</tr>
<tr>
<td>pen_diff</td>
<td>Vector of values to take deviation from.</td>
</tr>
<tr>
<td>e_alpha</td>
<td>Alpha for elastic net</td>
</tr>
<tr>
<td>rlasso_pen</td>
<td>Alpha for rlasso2</td>
</tr>
</tbody>
</table>
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

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

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>par</td>
<td>vector with parameters.</td>
</tr>
<tr>
<td>ImpCov</td>
<td>expected covariance matrix.</td>
</tr>
<tr>
<td>SampCov</td>
<td>Sample covariance matrix.</td>
</tr>
<tr>
<td>Areg</td>
<td>A matrix with current parameter estimates.</td>
</tr>
<tr>
<td>Sreg</td>
<td>S matrix with current parameter estimates.</td>
</tr>
<tr>
<td>A</td>
<td>A matrix with parameter labels.</td>
</tr>
<tr>
<td>S</td>
<td>S matrix with parameter labels.</td>
</tr>
<tr>
<td>F</td>
<td>F matrix.</td>
</tr>
<tr>
<td>lambda</td>
<td>penalty value.</td>
</tr>
<tr>
<td>type2</td>
<td>penalty type.</td>
</tr>
<tr>
<td>pen_vec</td>
<td>parameter indicators to be penalized.</td>
</tr>
<tr>
<td>diff_par</td>
<td>parameter values to take deviations from.</td>
</tr>
</tbody>
</table>

rcpp_quasi_calc

Compute quasi Hessian

Description

Compute quasi Hessian

Usage

rcpp_quasi_calc(I, s, y, H)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>identity matrix.</td>
</tr>
<tr>
<td>s</td>
<td>s vector.</td>
</tr>
<tr>
<td>y</td>
<td>y vector.</td>
</tr>
<tr>
<td>H</td>
<td>previous Hessian.</td>
</tr>
</tbody>
</table>
rcpp_RAMmult

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

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

Usage
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
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: rsoolnp 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>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</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>subOpt</td>
<td>Type of optimization to use in the optimx package.</td>
</tr>
<tr>
<td>longMod</td>
<td>If TRUE, the model is using longitudinal data? This changes the sample covariance used.</td>
</tr>
<tr>
<td>pars_pen</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>diff_par</td>
<td>Parameter values to deviate from. Only used when type=&quot;diff_lasso&quot;.</td>
</tr>
<tr>
<td>LB</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>UB</td>
<td>Upper bound vector</td>
</tr>
<tr>
<td>par.lim</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>block</td>
<td>Whether to use block coordinate descent</td>
</tr>
<tr>
<td>full</td>
<td>Whether to do full gradient descent or block</td>
</tr>
<tr>
<td>calc</td>
<td>Type of calc function to use with means or not. Not recommended for use.</td>
</tr>
<tr>
<td>max.iter</td>
<td>Number of iterations for coordinate descent</td>
</tr>
<tr>
<td>tol</td>
<td>Tolerance for coordinate descent</td>
</tr>
<tr>
<td>round</td>
<td>Number of digits to round results to</td>
</tr>
<tr>
<td>solver</td>
<td>Whether to use solver for coord_desc</td>
</tr>
<tr>
<td>quasi</td>
<td>Whether to use quasi-Newton</td>
</tr>
<tr>
<td>solver.maxit</td>
<td>Max iterations for solver in coord_desc</td>
</tr>
<tr>
<td>alpha.inc</td>
<td>Whether alpha should increase for coord_desc</td>
</tr>
<tr>
<td>line.search</td>
<td>Use line search for optimization. Default is no, use fixed step size</td>
</tr>
<tr>
<td>step</td>
<td>Step size</td>
</tr>
<tr>
<td>momentum</td>
<td>Momentum for step sizes</td>
</tr>
<tr>
<td>step.ratio</td>
<td>Ratio of step size between A and S. Logical</td>
</tr>
<tr>
<td>nlminb.control</td>
<td>List of control values to pass to nlminb</td>
</tr>
<tr>
<td>missing</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
# vignette("overview", package="regsem")
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("11", "12", "16", "17", "18"))
#equivalent to pars_pen=c(1:2, 6:8)
#summary(fit1)
summary.cvregsem

`summary.cvregsem` is a method for class `cvregsem` that prints information about the `cvregsem` object.

### Description

`summary.cvregsem` is a method for class `cvregsem` that prints information about the `cvregsem` object.

### Usage

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

### Arguments

- **object**: `cv_regsem` object
- **...**: Additional arguments

summary.regsem

`summary.regsem` is a method for class `regsem` that prints `Summary results from regsem`.

### Description

`summary.regsem` is a method for class `regsem` that prints `Summary results from regsem`.

### Usage

```r
## 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

**Description**

Function to performed exploratory mediation with continuous and categorical variables

**Usage**

```r
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

```
```r
a2b2 := a2*b2
a3b3 := a3*b3
a6b6 := a6*b6
# total effect (c)
c := c_prime + (a1*b1) + (a2*b2) + (a3*b3) + (a6*b6)

# p-value approach using delta method standard errors
fit.delta = sem(modell, 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, 6

*Topic calc
  cv_regsem, 2
  regsem, 14

*Topic chisq
  fit_indices, 7

*Topic extract
  extractMatrices, 7

*Topic factor
  efaModel, 6

*Topic fa
  efaModel, 6

*Topic fit
  fit_indices, 7

*Topic multiple
  multi_optim, 8

*Topic ncp
  fit_indices, 7

*Topic optim
  cv_regsem, 2
  multi_optim, 8
  regsem, 14

*Topic rmsea
  fit_indices, 7

  cv_regsem, 2
  efaModel, 6
  extractMatrices, 7
  fit_indices, 7
  multi_optim, 8, 15
  parse_parameters, 11
  plot.cvregsem, 11
  rcpp_fit_fun, 12
  rcpp_grad_ram, 13
  rcpp_quasi_calc, 13
  rcpp_RAMmult, 14
  regsem, 14
  summary.cvregsem, 18
  summary.regsem, 18
  xmed, 19