Package: sem (via r-universe)

October 28, 2026

Version 3.1-16

Date 2024-08-25

Title Structural Equation Models

Depends R (>= 3.5.0), stats

Imports MASS, boot, mi (>= 0.9-99), utils

Suggests polycor, DiagrammeR (>= 1.0.9)

LazyLoad yes

LazyData yes

ByteCompile yes

Description Functions for fitting general linear structural equation models (with observed and latent variables) using the RAM approach, and for fitting structural equations in observed-variable models by two-stage least squares.

License GPL (>= 2)

URL https://CRAN.R-project.org/package=sem

NeedsCompilation yes

Author John Fox [aut], Zhenghua Nie [aut, cre], Jarrett Byrnes [aut], Michael Culbertson [ctb], Saikat DebRoy [ctb], Michael Friendly [ctb], Benjamin Goodrich [ctb], Richard H. Jones [ctb], Adam Kramer [ctb], Georges Monette [ctb], Frederick Novomestky [ctb], R-Core [ctb]

Maintainer Zhenghua Nie <zhenghua.nie@gmail.com>

Repository https://cran.r-universe.dev

Date/Publication 2024-08-29 02:52:48 UTC

RemoteUrl https://github.com/cran/sem

RemoteRef HEAD

RemoteSha 0febabeb5fd7aab95f293549e02905d18caf33af

Contents

Bollen . . . 2
bootSem . . . 3
CNES . . . 7
effects.sem . . . 8
fscores . . . 10
HS.data . . . 12
information.criteria . . . 14
Klein . . . 15
Kmenta . . . 16
miSem . . . 17
ML.methods . . . 19
modIndices . . . 22
objective.functions . . . 24
optimizers . . . 26
pathDiagram . . . 27
ram . . . 33
rawMoments . . . 34
readMoments . . . 36
residuals.sem . . . 37
sem . . . 40
sem-deprecated . . . 62
specifyModel . . . 63
standardizedCoefficients . . . 72
Tests . . . 74
tsls . . . 75

Index **78**

`Bollen` *Bollen's Data on Industrialization and Political Democracy*

Description

This data set includes four measures of democracy at two points in time, 1960 and 1965, and three measures of industrialization in 1960, for 75 developing countries.

Usage

```
Bollen
```

Format

A data frame with 75 observations on the following 11 variables.

- `y1` freedom of the press, 1960
- `y2` freedom of political opposition, 1960
- `y3` fairness of elections, 1960
- `y4` effectivness of elected legislature, 1960
- `y5` freedom of the press, 1965
- `y6` freedom of political opposition, 1965
- `y7` fairness of elections, 1965
- `y8` effectivness of elected legislature, 1965
- `x1` GNP per capita, 1960
- `x2` energy consumption per capita, 1960
- `x3` percentage of labor force in industry, 1960

Details

Variables `y1` through `y4` are intended to be indicators of the latent variable *political democracy in 1960*; `y5` through `y8` indicators of *political democracy in 1965*; and `x1` through `x3` indicators of *industrialization in 1960*.

Source

personal communication from Ken Bollen.

References

Bollen, K. A. (1989) *Structural Equations With Latent Variables*. Wiley.

`bootSem` *Bootstrap a Structural Equation Model*

Description

Bootstraps a structural equation model in an `sem` object (as returned by the `sem` function).

Usage

```
bootSem(model, ...)

## S3 method for class 'sem'
bootSem(model, R=100, Cov=cov, data=model$data,
    max.failures=10, show.progress=TRUE, ...)

## S3 method for class 'msem'
bootSem(model, R=100, Cov=cov, data=model$data,
    max.failures=10, show.progress=TRUE, ...)

## S3 method for class 'bootsem'
print(x, digits=getOption("digits"), ...)

## S3 method for class 'bootsem'
summary(object,
    type=c("perc", "bca", "norm", "basic", "none"), level=0.95, ...)
```

Arguments

model	an sem or msem object, produced by the sem function.
R	the number of bootstrap replications; the default is 100, which should be enough for computing standard errors, but not confidence intervals (except for the normal-theory intervals).
Cov	a function to compute the input covariance or moment matrix; the default is cov. Use cor if the model is fit to the correlation matrix. The function hetcor in the polycor package will compute product-moment, polychoric, and polyserial correlations among mixed continuous and ordinal variables (see the first example below for an illustration).
data	in the case of a sem (i.e., single-group) model, a data set in a form suitable for Cov; for example, for the default Cov=cov, data may be a numeric data frame or a numeric matrix. In the case of an msem (i.e., multi-group) model, a list of data sets (again in the appropriate form), one for each group; in this case, bootstrapping is done within each group, treating the groups as strata. Note that the original observations are required, not just the covariance matrix of the observed variables in the model. The default is the data set stored in the sem object, which will be present only if the model was fit to a data set rather than to a covariance or moment matrix, and may not be in a form suitable for Cov.
max.failures	maximum number of consecutive convergence failures before bootSem gives up.
show.progress	display a text progress bar on the console tracing the bootstrap replications.
x, object	an object of class bootsem.
digits	controls the number of digits to print.
type	type of bootstrapped confidence intervals to compute; the default is "perc" (percentile); see boot.ci for details.
level	level for confidence intervals; default is 0.95.
...	in bootSem, arguments to be passed to sem; otherwise ignored.

Details

bootSem implements the nonparametric bootstrap, assuming an independent random sample. Convergence failures in the bootstrap resamples are discarded (and a warning printed); more than max.failures consecutive convergence failures (default, 10) result in an error. You can use the boot function in the **boot** package for more complex sampling schemes and additional options.

Bootstrapping is implemented by resampling the observations in data, recalculating the input covariance matrix with Cov, and refitting the model with sem, using the parameter estimates from the original sample as start-values.

Warning: the bootstrapping process can be very time-consuming.

Value

bootSem returns an object of class bootsem, which inherits from class boot, supported by the **boot** package. The returned object contains the following components:

t0	the estimated parameters in the model fit to the original data set.
t	a matrix containing the bootstrapped estimates, one bootstrap replication per row.
data	the data to which the model was fit.
seed	the value of .Random.seed when bootSem was called.
statistic	the function used to produce the bootstrap replications; this is always the local function refit from bootSem.
sim	always set to "ordinary"; see the documentation for the boot function.
stype	always set to "i"; see the documentation for the boot function.
call	the call of the bootSem function.
weights	a vector of length equal to the number of observations N, with entries $1/N$. For a multi-group model, the weights in group j are $1/N_j$, the inverse of the number of observations in the group.
strata	a vector of length N containing the number of the stratum to which each observation belongs; for a single-group model, all entries are 1.

Author(s)

John Fox <jfox@mcmaster.ca>

References

Davison, A. C., and Hinkley, D. V. (1997) *Bootstrap Methods and their Application.* Cambridge.

Efron, B., and Tibshirani, R. J. (1993) *An Introduction to the Bootstrap.* Chapman and Hall.

See Also

boot, sem

Examples

```
    ## Not run:  # because of long execution time

# A simple confirmatory factor-analysis model using polychoric correlations.
#  The polycor package is required for the hetcor function.

if (require(polycor)){

# The following function returns correlations computed by hetcor,
#   and is used for the bootstrapping.

hcor <- function(data) hetcor(data, std.err=FALSE)$correlations

model.cnes <- specifyModel(text="
F -> MBSA2, lam1
F -> MBSA7, lam2
F -> MBSA8, lam3
F -> MBSA9, lam4
F <-> F, NA, 1
MBSA2 <-> MBSA2, the1
MBSA7 <-> MBSA7, the2
MBSA8 <-> MBSA8, the3
MBSA9 <-> MBSA9, the4
")

R.cnes <- hcor(CNES)

sem.cnes <- sem(model.cnes, R.cnes, N=1529)
summary(sem.cnes)
}

#  Note: this can take a minute:

set.seed(12345) # for reproducibility

system.time(boot.cnes <- bootSem(sem.cnes, R=100, Cov=hcor, data=CNES))
summary(boot.cnes, type="norm")
# cf., standard errors to those computed by summary(sem.cnes)

## End(Not run)

    ## Not run:   # because of long execution time

# An example bootstrapping a multi-group model

mod.hs <- cfa(text="
spatial: visual, cubes, paper, flags
verbal: general, paragrap, sentence, wordc, wordm
memory: wordr, numberr, figurer, object, numberf, figurew
math: deduct, numeric, problemr, series, arithmet
")
```

```
mod.mg <- multigroupModel(mod.hs, groups=c("Female", "Male"))

sem.mg <- sem(mod.mg, data=HS.data, group="Gender",
              formula = ~ visual + cubes + paper + flags +
              general + paragrap + sentence + wordc + wordm +
              wordr + numberr + figurer + object + numberf + figurew +
              deduct + numeric + problemr + series + arithmet
              )

# Note: this example can take several minutes or more;
#       you can decrease R if you just want to see how it works:

set.seed(12345) # for reproducibility

system.time(boot.mg <- bootSem(sem.mg, R=100))
summary(boot.mg, type="norm")
# cf., standard errors to those computed by summary(sem.mg)

## End(Not run)
```

CNES *Variables from the 1997 Canadian National Election Study*

Description

These variables are from the mailback questionnaire to the 1997 Canadian National Election Study, and are intended to tap attitude towards "traditional values."

Usage

```
CNES
```

Format

A data frame with 1529 observations on the following 4 variables.

`MBSA2` an ordered factor with levels `StronglyDisagree`, `Disagree`, `Agree`, and `StronglyAgree`, in response to the statement, "We should be more tolerant of people who choose to live according to their own standards, even if they are very different from our own."

`MBSA7` an ordered factor with levels `StronglyDisagree`, `Disagree`, `Agree`, and `StronglyAgree`, in response to the statement, "Newer lifestyles are contributing to the breakdown of our society."

`MBSA8` an ordered factor with levels `StronglyDisagree`, `Disagree`, `Agree`, and `StronglyAgree`, in response to the statement, "The world is always changing and we should adapt our view of moral behaviour to these changes."

`MBSA9` an ordered factor with levels `StronglyDisagree`, `Disagree`, `Agree`, and `StronglyAgree`, in response to the statement, "This country would have many fewer problems if there were more emphasis on traditional family values."

Source

York University Institute for Social Research.

`effects.sem` *Total, Direct, and Indirect Effects for Structural Equation Models*

Description

The `sem` method for the standard generic function `effects` computes total, direct, and indirect effects for a fitted structural equation model according to the method described in Fox (1980).

Usage

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

## S3 method for class 'semeffects'
print(x, digits = getOption("digits"), ...)
## S3 method for class 'semeffectsList'
print(x, digits = getOption("digits"), ...)
```

Arguments

`object`	a fitted structural-equation model object produced by the `sem` function.
`x`	an object of class `semeffects` or `semeffectsList`, produced by `effects`.
`digits`	digits to print.
`...`	not used.

Value

`effect.sem` returns an object of class `semeffects` with `Total`, `Direct`, and `Indirect` elements.

Author(s)

John Fox <jfox@mcmaster.ca>

References

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

See Also

`sem`

Examples

```
    ## Not run:

# These examples are from Fox (1980)

# In the first pair of examples, readMoments() and specifyModel() read from the
# input stream. These examples cannot be executed via example() but can be entered
# at the command prompt. The Blau and Duncan example is repeated using file input;
# this example can be executed via example().

# The recursive Blau and Duncan basic stratification model:
#  x1 is father's education, x2 father's SES, y3 respondent's education,
#  y4 SES of respondent's first job, y5 respondent's SES in 1962

R.bd <- readMoments(names=c("x1", "x2", "y3", "y4", "y5"))
1
.516 1
.453 .438 1
.332 .417 .538 1
.322 .405 .596 .541 1

mod.bd <- specifyModel()
y3 <- x1, gam31
y3 <- x2, gam32
y4 <- x2, gam42
y4 <- y3, beta43
y5 <- x2, gam52
y5 <- y3, beta53
y5 <- y4, beta54

sem.bd <- sem(mod.bd, R.bd, N=20700, fixed.x=c("x1", "x2"))
summary(sem.bd)
effects(sem.bd)


# The nonrecursive Duncan, Haller, and Portes peer-influences model for observed variables:

R.DHP <- readMoments(diag=FALSE, names=c("ROccAsp", "REdAsp", "FOccAsp",
"FEdAsp", "RParAsp", "RIQ", "RSES", "FSES", "FIQ", "FParAsp"))
.6247
.3269  .3669
.4216  .3275  .6404
.2137  .2742  .1124  .0839
.4105  .4043  .2903  .2598  .1839
.3240  .4047  .3054  .2786  .0489  .2220
.2930  .2407  .4105  .3607  .0186  .1861  .2707
.2995  .2863  .5191  .5007  .0782  .3355  .2302  .2950
.0760  .0702  .2784  .1988  .1147  .1021  .0931 -.0438  .2087

model.dhp <- specifyModel()
RIQ      -> ROccAsp, gam51,  NA
RSES     -> ROccAsp, gam52,  NA
```

```
FSES     -> FOccAsp, gam63,  NA
FIQ      -> FOccAsp, gam64,  NA
FOccAsp  -> ROccAsp, beta56, NA
ROccAsp  -> FOccAsp, beta65, NA
ROccAsp <-> ROccAsp, ps55,   NA
FOccAsp <-> FOccAsp, ps66,   NA
ROccAsp <-> FOccAsp, ps56,   NA


# Note: The following generates a warning because not all of the variables
#       in the correlation matrix are used
sem.dhp <- sem(model.dhp, R.DHP, 329,
               fixed.x=c('RIQ', 'RSES', 'FSES', 'FIQ'))
summary(sem.dhp)
effects(sem.dhp)

## End(Not run)

## the following example may be executed via example()

etc <- system.file(package="sem", "etc") # path to data and model files

# The recursive Blau and Duncan basic stratification model:
#  x1 is father's education, x2 father's SES, y3 respondent's education,
#  y4 SES of respondent's first job, y5 respondent's SES in 1962

(R.bd <- readMoments(file=file.path(etc, "R-Blau-Duncan.txt"),
     names=c("x1", "x2", "y3", "y4", "y5")))
(mod.bd <- specifyModel(file=file.path(etc, "model-Blau-Duncan.txt")))
sem.bd <- sem(mod.bd, R.bd, N=20700, fixed.x=c("x1", "x2"))
summary(sem.bd)
effects(sem.bd)
```

`fscores` *Factor Scores for Latent Variables*

Description

Calculate factor scores or factor-score coefficients for the latent variables in a structural-equation model.

Usage

```
## S3 method for class 'sem'
fscores(model, data=model$data, center=TRUE, scale=FALSE, ...)
## S3 method for class 'msem'
fscores(model, data=model$data, center=TRUE, scale=FALSE, ...)
```

Arguments

model	an object of class "sem" or "msem", produced by the sem function.
data	an optional numeric data frame or matrix containing the observed variables in the model; if not NULL, the estimated factor scores are returned; if NULL, the factor-score *coefficients* are returned. The default is the data element of model, which is non-NULL if the model was fit to a data set rather than a covariance or moment matrix.
center	if TRUE, the default, the means of the observed variables are subtracted prior to computing factor scores. One would normally use this option if the model is estimated from a covariance or correlation matrix among the observed variables.
scale	if TRUE, the possibly centered variables are divided by their root-mean-squares; the default is FALSE. One would normally use this option if the model is estimated from a correlation matrix among the observed variables. Centering and scaling are performed by the scale function.
...	arguments to pass down.

Details

Factor-score coefficients are computed by the "regression" method as $B = C^{-1}C^*$, where C is the model-implied covariance or moment matrix among the observed variables and C^* is the matrix of model-implied covariances or moments between the observed and latent variables.

Value

Either a matrix of estimated factor scores (if the data argument is supplied) or a matrix of factor-score coefficients (otherwise). In the case of an "msem" argument, a list of matrices is returned.

Author(s)

John Fox <jfox@mcmaster.ca>

References

Bollen, K. A. (1989) *Structural Equations With Latent Variables.* Wiley.

See Also

sem, scale

Examples

```
# In the first example, readMoments() and specifyModel() read from the
# input stream. This example cannot be executed via example() but can be entered
# at the command prompt. The example is repeated using file input;
# this example can be executed via example().
    ## Not run:

S.wh <- readMoments(names=c('Anomia67','Powerless67','Anomia71',
                                    'Powerless71','Education','SEI'))
```

```
   11.834
    6.947    9.364
    6.819    5.091   12.532
    4.783    5.028    7.495    9.986
   -3.839   -3.889   -3.841   -3.625   9.610
  -21.899  -18.831  -21.748  -18.775  35.522  450.288

# This model in the SAS manual for PROC CALIS

model.wh.1 <- specifyModel()
    Alienation67   ->  Anomia67,      NA,     1
    Alienation67   ->  Powerless67,   NA,     0.833
    Alienation71   ->  Anomia71,      NA,     1
    Alienation71   ->  Powerless71,   NA,     0.833
    SES            ->  Education,     NA,     1
    SES            ->  SEI,           lamb,   NA
    SES            ->  Alienation67,  gam1,   NA
    Alienation67   ->  Alienation71,  beta,   NA
    SES            ->  Alienation71,  gam2,   NA
    Anomia67       <-> Anomia67,      the1,   NA
    Anomia71       <-> Anomia71,      the1,   NA
    Powerless67    <-> Powerless67,   the2,   NA
    Powerless71    <-> Powerless71,   the2,   NA
    Education      <-> Education,     the3,   NA
    SEI            <-> SEI,           the4,   NA
    Anomia67       <-> Anomia71,      the5,   NA
    Powerless67    <-> Powerless71,   the5,   NA
    Alienation67   <-> Alienation67,  psi1,   NA
    Alienation71   <-> Alienation71,  psi2,   NA
    SES            <-> SES,           phi,    NA


sem.wh.1 <- sem(model.wh.1, S.wh, 932)

fscores(sem.wh.1)

## End(Not run)

# The following example can be executed via example():

etc <- system.file(package="sem", "etc") # path to data and model files

(S.wh <- readMoments(file=file.path(etc, "S-Wheaton.txt"),
     names=c('Anomia67','Powerless67','Anomia71',
                            'Powerless71','Education','SEI')))
(model.wh.1 <- specifyModel(file=file.path(etc, "model-Wheaton-1.txt")))
(sem.wh.1 <- sem(model.wh.1, S.wh, 932))
fscores(sem.wh.1)
```

HS.data *Holzinger and Swineford's Data*

Description

This data set, for scores on a variety of tests, was originally in the MBESS package. A new version of the data set in that package doesn't appear to be identical to this one.

Usage

```
HS.data
```

Format

A data frame with 301 observations on the following 32 variables.

`id` a numeric vector
`Gender` a factor with levels `Female` `Male`
`grade` a numeric vector
`agey` a numeric vector
`agem` a numeric vector
`school` a factor with levels `Grant-White` `Pasteur`
`visual` a numeric vector
`cubes` a numeric vector
`paper` a numeric vector
`flags` a numeric vector
`general` a numeric vector
`paragrap` a numeric vector
`sentence` a numeric vector
`wordc` a numeric vector
`wordm` a numeric vector
`addition` a numeric vector
`code` a numeric vector
`counting` a numeric vector
`straight` a numeric vector
`wordr` a numeric vector
`numberr` a numeric vector
`figurer` a numeric vector
`object` a numeric vector
`numberf` a numeric vector
`figurew` a numeric vector
`deduct` a numeric vector
`numeric` a numeric vector
`problemr` a numeric vector
`series` a numeric vector
`arithmet` a numeric vector
`paperrev` a numeric vector
`flagssub` a numeric vector

Source

Originally from Holzinger and Swineford (1939). This copy is originally from version 4.6.0 of the MBESS package.

References

Holzinger, K. J. and Swineford, F. A. (1939). A study in factor analysis: The stability of a bi-factor solution. Supplementary Education Monographs, 48. University of Chicago.

Examples

```
summary(HS.data)
```

---

information.criteria *Additional Information Criteria*

Description

These are generic functions for computing, respectively, the AICc (second-order corrected Akaike Information Criterion) and CAIC (consistent Akaike Information Criterion).

Usage

```
AICc(object, ...)

CAIC(object, ...)
```

Arguments

`object`	an object for which an appropriate `AICc` or `CAIC` method exists.
`...`	possible additional arguments for methods.

Author(s)

Jarrett Byrnes and John Fox <jfox@mcmaster.ca>

References

Burnham, K. P., and Anderson, D. R. (1998) *Model Selection and Inference: A Practical Information-Theoretical Approach.* Springer.

Bozdogan, H. (1987) Model selection and Akaike's information criterion (AIC). *Psychometrika* **52**, 345–370.

See Also

`AICc.objectiveML`, `CAIC.objectiveML`

`Klein` *Klein's Data on the U. S. Economy*

Description

Data for Klein's (1950) simple econometric model of the U. S. economy.

The `Klein` data frame has 22 rows and 10 columns.

Usage

```
Klein
```

Format

This data frame contains the following columns:

Year 1921–1941

C consumption.

P private profits.

Wp private wages.

I investment.

K.lag capital stock, lagged one year.

X equilibrium demand.

Wg government wages.

G government non-wage spending.

T indirect business taxes and net exports.

Source

Greene, W. H. (1993) *Econometric Analysis, Second Edition.* Macmillan.

References

Klein, L. (1950) *Economic Fluctuations in the United States 1921–1941.* Wiley.

Examples

```
Klein$P.lag <- c(NA, Klein$P[-22])
Klein$X.lag <- c(NA, Klein$X[-22])

summary(tsls(C ~ P + P.lag + I(Wp + Wg),
    instruments=~1 + G + T + Wg + I(Year - 1931) + K.lag + P.lag + X.lag,
    data=Klein))

summary(tsls(I ~ P + P.lag + K.lag,
    instruments=~1 + G + T + Wg + I(Year - 1931) + K.lag + P.lag + X.lag,
```

```
    data=Klein))

summary(tsls(Wp ~ X + X.lag + I(Year - 1931),
    instruments=~1 + G + T + Wg + I(Year - 1931) + K.lag + P.lag + X.lag,
    data=Klein))
```

`Kmenta` *Partly Artificial Data on the U. S. Economy*

Description

These are partly contrived data from Kmenta (1986), constructed to illustrate estimation of a simultaneous-equation model.

The `Kmenta` data frame has 20 rows and 5 columns.

Usage

```
Kmenta
```

Format

This data frame contains the following columns:

Q food consumption per capita.

P ratio of food prices to general consumer prices.

D disposable income in constant dollars.

F ratio of preceding year's prices received by farmers to general consumer prices.

A time in years.

Details

The exogenous variables `D`, `F`, and `A` are based on real data; the endogenous variables `P` and `Q` were generated by simulation.

Source

Kmenta, J. (1986) *Elements of Econometrics, Second Edition*, Macmillan.

miSem *Estimate a Structural Equation Model By Multiple Imputation*

Description

miSem uses the mi function in the **mi** package to generate multiple imputations of missing data, fitting the specified model to each completed data set.

Usage

```
miSem(model, ...)

## S3 method for class 'semmod'
miSem(model, ..., data, formula = ~., raw=FALSE,
        fixed.x=NULL, objective=objectiveML,
       n.imp=5, n.chains=n.imp, n.iter=30, seed=sample(1e6, 1), mi.args=list(),
        show.progress=TRUE)

## S3 method for class 'semmodList'
miSem(model, ..., data, formula = ~., group, raw=FALSE,
        fixed.x=NULL, objective=msemObjectiveML,
       n.imp=5, n.chains=n.imp, n.iter=30, seed=sample(1e6, 1), mi.args=list(),
        show.progress=TRUE)

## S3 method for class 'miSem'
print(x, ...)

## S3 method for class 'miSem'
summary(object, digits=max(3, getOption("digits") - 2), ...)
```

Arguments

model	an SEM model-description object of class semmod or semmodList, created by specifyEquations cfa, or specifyModel, in the case of a multi-group model in combination with multigroupModel.
..., formula, raw, fixed.x, objective, group	arguments to be passed to sem.
data	an R data frame, presumably with some missing data (encoded as NA), containing the data for fitting the SEM, possibly along with other variables to use to obtain multiple imputations of missing values. In the case of a multi-group model, this must be a *single* data frame.
n.imp	number of imputations (default 5).
n.chains	number of Markov chains (default is the number of imputations).
n.iter	number of iterations for the multiple-imputation process (default 30).
seed	seed for the random-number generator (default is an integer sampled from 1 to 1E6); stored in the resulting object.

mi.args	other arguments to be passed to mi.
show.progress	show a text progress bar on the console tracking model fitting to the multiple imputations; this is distinct from the progress of the multiple-imputation process, which is controlled by the verbose argument to mi (and which, although it defaults to TRUE, *fails* to produce verbose output on Windows system, as of **mi** version 1.0).
x, object	an object of class "miSem".
digits	for printing numbers.

Value

miSem returns an object of class "miSem" with the following components:

initial.fit	an sem model object produced using objectiveFIML if raw=TRUE, or the objective function given by the objective argument otherwise.
mi.fits	a list of sem model objects, one for each imputed data set.
imputation	the object produced by complete, containing the completed imputed data sets.
seed	the seed used for the random number generator.
mi.data	the object returned by mi, containing the multiple imputations, and useful, e.g., for diagnostic checking of the imputation process.

Author(s)

John Fox <jfox@mcmaster.ca>

References

Yu-Sung Su, Andrew Gelman, Jennifer Hill, Masanao Yajima. (2011). "Multiple imputation with diagnostics (mi) in R: Opening windows into the black box." *Journal of Statistical Software* 45(2).

See Also

sem, mi

Examples

```
    ## Not run:  # because of long execution time
mod.cfa.tests <- cfa(raw=TRUE, text="
verbal: x1, x2, x3
math: y1, y2, y3
")
imps <- miSem(mod.cfa.tests, data=Tests, fixed.x="Intercept",
              raw=TRUE, seed=12345)
summary(imps, digits=3)


# introduce some missing data to the HS.data data set:
HS <- HS.data[, c(2,7:10,11:15,20:25,26:30)]
set.seed(12345)
```

```
r <- sample(301, 100, replace=TRUE)
c <- sample(2:21, 100, replace=TRUE)
for (i in 1:100) HS[r[i], c[i]] <- NA

mod.hs <- cfa(text="
spatial: visual, cubes, paper, flags
verbal: general, paragrap, sentence, wordc, wordm
memory: wordr, numberr, figurer, object, numberf, figurew
math: deduct, numeric, problemr, series, arithmet
")

mod.mg <- multigroupModel(mod.hs, groups=c("Female", "Male"))
system.time( # relatively time-consuming!
  imps.mg <- miSem(mod.mg, data=HS, group="Gender", seed=12345)
)
summary(imps.mg, digits=3)

## End(Not run)
```

`ML.methods` *Methods for sem Objects Fit Using the* `objectiveML`, `objectiveGLS`, `objectiveFIML`, `msemObjectiveML`, *and* `msemObjectiveGLS` *Objective Functions*

Description

These functions are for objects fit by `sem` using the `objectiveML` (multivariate-normal full-information maximum-likelihood), `link{objectiveFIML}` (multivariate-normal full-information maximum-likihood in the presence of missing data), `objectiveGLS` (generalized least squares), and `msemObjectiveML` (multigroup multivariate-normal FIML) objective functions.

Usage

```
## S3 method for class 'objectiveML'
anova(object, model.2, robust=FALSE, ...)
## S3 method for class 'objectiveFIML'
anova(object, model.2, ...)

## S3 method for class 'objectiveML'
logLik(object, ...)
## S3 method for class 'objectiveFIML'
logLik(object, saturated=FALSE,
    intercept="Intercept", iterlim=1000, ...)
## S3 method for class 'objectiveML'
deviance(object, ...)
## S3 method for class 'objectiveFIML'
deviance(object, saturated.logLik, ...)
## S3 method for class 'msemObjectiveML'
deviance(object, ...)
```

```
## S3 method for class 'objectiveML'
AIC(object, ..., k)
## S3 method for class 'objectiveFIML'
AIC(object, saturated.logLik, ..., k)
## S3 method for class 'msemObjectiveML'
AIC(object, ..., k)
## S3 method for class 'objectiveML'
AICc(object, ...)
## S3 method for class 'objectiveFIML'
AICc(object, saturated.logLik, ...)
## S3 method for class 'msemObjectiveML'
AICc(object, ...)
## S3 method for class 'objectiveML'
BIC(object, ...)
## S3 method for class 'objectiveFIML'
BIC(object, saturated.logLik, ...)
## S3 method for class 'msemObjectiveML'
BIC(object, ...)
## S3 method for class 'objectiveML'
CAIC(object, ...)
## S3 method for class 'objectiveFIML'
CAIC(object, saturated.logLik, ...)

## S3 method for class 'objectiveML'
print(x, ...)
## S3 method for class 'objectiveGLS'
print(x, ...)
## S3 method for class 'objectiveFIML'
print(x, saturated=FALSE, ...)
## S3 method for class 'msemObjectiveML'
print(x, ...)
## S3 method for class 'msemObjectiveGLS'
print(x, ...)

## S3 method for class 'objectiveML'
summary(object, digits=getOption("digits"),
    conf.level=.90, robust=FALSE, analytic.se=object$t <= 500,
    fit.indices=c("GFI", "AGFI", "RMSEA", "NFI", "NNFI", "CFI", "RNI",
      "IFI", "SRMR", "AIC", "AICc", "BIC", "CAIC"), ...)
## S3 method for class 'objectiveFIML'
summary(object, digits=getOption("digits"), conf.level=.90,
    fit.indices=c("AIC", "AICc", "BIC", "CAIC"),
    saturated=FALSE, intercept="Intercept", saturated.logLik, ...)
## S3 method for class 'objectiveGLS'
summary(object, digits=getOption("digits"), conf.level=.90,
   fit.indices=c("GFI", "AGFI", "RMSEA", "NFI", "NNFI", "CFI", "RNI", "IFI", "SRMR"),
    ...)
## S3 method for class 'msemObjectiveML'
```

```
summary(object, digits=getOption("digits"),
    conf.level=.90, robust=FALSE,
    analytic.se=object$t <= 500,
    fit.indices=c("GFI", "AGFI", "RMSEA", "NFI", "NNFI", "CFI", "RNI",
      "IFI", "SRMR", "AIC", "AICc", "BIC"), ...)
## S3 method for class 'msemObjectiveGLS'
summary(object, digits=getOption("digits"),
    conf.level=.90,
    fit.indices=c("GFI", "AGFI", "RMSEA", "NFI", "NNFI",
      "CFI", "RNI", "IFI", "SRMR"), ...)
```

Arguments

object, model.2, x	an object inheriting from class objectiveML, objectiveGLS, objectiveFIML, msemObjectiveML, or msemObjectiveGLS.
robust	if TRUE, compute robust standard errors or test.
fit.indices	a character vector of "fit indices" to report; the allowable values are those given in **Usage** above, and vary by the objective function. If the argument isn't given then the fit indices reported are taken from the R fit.indices option; if this option isn't set, then only the AIC and BIC are reported for models fit with objectiveML, objectiveFIML, or msemObjectiveML, and no fit indices are reported for models fit with objectiveGLS or msemObjectiveGLS.
k, ...	ignored.
digits	digits to be printed.
conf.level	level for confidence interval for the RMSEA index (default is .9).
analytic.se	use analytic (as opposed to numeric) coefficient standard errors; default is TRUE where analytic standard errors are available if there are no more than 100 parameters in the model and FALSE otherwise.
saturated	if TRUE (the default is FALSE); compute the log-likelihood (and statistics that depend on it) for the saturated model when the objective function is FIML in the presence of missing data. This can be computationally costly.
intercept	the name of the intercept regressor in the raw data, to be used in calculating the saturated log-likelihood for the FIML estimator; the default is "Intercept".
saturated.logLik	the log-likelihood for the saturated model, as returned by logLik with saturated=TRUE; if absent, this will be computed and the computation can be time-consuming.
iterlim	iteration limit used by the nlm optimizer to compute the saturated log-likelihood for the FIML estimator with missing data; defaults to 1000.

Author(s)

John Fox <jfox@mcmaster.ca> and Jarrett Byrnes

References

See sem.

See Also

sem, objective.functions, modIndices.objectiveML

---

modIndices *Modification Indices for Structural Equation Models*

Description

mod.indices calculates modification indices (score tests) and estimated parameter changes for the fixed and constrained parameters in a structural equation model fit by multinormal maximum likelihood.

Usage

```
## S3 method for class 'objectiveML'
modIndices(model, duplicated, deviance=NULL, ...)
## S3 method for class 'msemObjectiveML'
modIndices(model, ...)

## S3 method for class 'modIndices'
print(x, n.largest=5, ...)
## S3 method for class 'msemModIndices'
print(x, ...)

## S3 method for class 'modIndices'
summary(object, round=2,
    print.matrices=c("both", "par.change", "mod.indices"), ...)
## S3 method for class 'msemModIndices'
summary(object, ...)
```

Arguments

model	an object of class objectiveML or msemObjectiveML, produced by the sem function.
object, x	an object of class modIndices or msemModIndices, produced by the modIndices function.
n.largest	number of modification indices to print in each of the A and P matrices of the RAM model.
round	number of places to the right of the decimal point in printing modification indices.
print.matrices	which matrices to print: estimated changes in the fixed parameters, modification indices, or both (the default).
duplicated, deviance	for internal use.
...	arguments to be passed down.

Details

Modification indices are one-df chi-square score ("Lagrange-multiplier") test statistics for the fixed and constrained parameters in a structural equation model. They may be regarded as an estimate of the improvement in the likelihood-ratio chi-square statistic for the model if the corresponding parameter is respecified as a free parameter. The `modIndices` function also estimates the change in the value of a fixed or constrained parameter if the parameter is respecified as free. When several parameters are set equal, modification indices and estimated changes are given for all but the first. Modification indices and estimated parameter changes for currently free parameters are given as `NA`.

The method employed is described in Saris, Satorra, and Sorbom (1987) and Sorbom (1989).

Value

`modIndices` returns an object of class `modIndices` with the following elements:

`mod.A`	modification indices for the elements of the A matrix.
`mod.P`	modification indices for the elements of the P matrix.
`par.A`	estimated parameter changes for the elements of the A matrix.
`par.P`	estimated parameter changes for the elements of the P matrix.

Author(s)

John Fox <jfox@mcmaster.ca> and Michael Culbertson

References

Sarris, W. E., Satorra, A., and Sorbom, D. (1987) The detection and correction of specification errors in structural equation models. Pp. 105–129 in Clogg, C. C. (ed.), *Sociological Methodology 1987.* American Sociological Association.

Sorbom, D. (1989) Model modification. *Psychometrika* **54**, 371–384.

See Also

`sem`

Examples

```
# In the first example, readMoments() and specifyModel() read from the
# input stream. This example cannot be executed via example() but can be entered
# at the command prompt. The example is repeated using file input;
# this example can be executed via example().
 ## Not run:
# This example is adapted from the SAS manual

S.wh <- readMoments(names=c('Anomia67','Powerless67','Anomia71',
                                    'Powerless71','Education','SEI'))
   11.834
    6.947    9.364
    6.819    5.091   12.532
    4.783    5.028    7.495    9.986
```

```
   -3.839   -3.889   -3.841   -3.625   9.610
  -21.899  -18.831  -21.748  -18.775  35.522  450.288

model.wh <- specifyModel()
    Alienation67   ->  Anomia67,      NA,   1
    Alienation67   ->  Powerless67,   NA,   0.833
    Alienation71   ->  Anomia71,      NA,   1
    Alienation71   ->  Powerless71,   NA,   0.833
    SES            ->  Education,     NA,   1
    SES            ->  SEI,           lamb, NA
    SES            ->  Alienation67,  gam1, NA
    Alienation67   ->  Alienation71,  beta, NA
    SES            ->  Alienation71,  gam2, NA
    Anomia67       <-> Anomia67,      the1, NA
    Anomia71       <-> Anomia71,      the1, NA
    Powerless67    <-> Powerless67,   the2, NA
    Powerless71    <-> Powerless71,   the2, NA
    Education      <-> Education,     the3, NA
    SEI            <-> SEI,           the4, NA
    Anomia67       <-> Anomia71,      the5, NA
    Powerless67    <-> Powerless71,   the5, NA
    Alienation67   <-> Alienation67,  psi1, NA
    Alienation71   <-> Alienation71,  psi2, NA
    SES            <-> SES,           phi,  NA

sem.wh <- sem(model.wh, S.wh, 932)
modIndices(sem.wh)

## End(Not run)

# The following example can be executed via example():

etc <- system.file(package="sem", "etc") # path to data and model files

(S.wh <- readMoments(file=file.path(etc, "S-Wheaton.txt"),
     names=c('Anomia67','Powerless67','Anomia71',
                            'Powerless71','Education','SEI')))
(model.wh <- specifyModel(file=file.path(etc, "model-Wheaton-1.txt")))
(sem.wh <- sem(model.wh, S.wh, 932))
modIndices(sem.wh)
```

`objective.functions` *sem Objective-Function Builders*

Description

These functions return objective functions suitable for use with `optimizers` called by `sem`. The user would not normally call these functions directly, but rather supply one of them in the `objective` argument to `sem`. Users may also write their own objective functions. `objectiveML` and `objectiveML2` are for multinormal maximum-likelihood estimation; `objectiveGLS` and `objectiveGLS2` are for

generalized least squares; and objectiveFIML2 is for so-called "full-information maximum-likelihood" estimation in the presence of missing data. The FIML estimator provides the same estimates as the ML estimator when there is no missing data; it can be slow because it iterates over the unique patterns of missing data that occur in the data set. objectiveML and objectiveGLS use compiled code and are therefore substantially faster. objectiveML2 and objectiveGLS2 are provided primarily to illustrate how to write sem objective functions in R. msemObjectiveML uses compiled code is for fitting multi-group models by multinormal maximum likelihood; msemObjectiveML2 is similar but doesn't use compiled code. msemObjectiveGLS uses compiled code and is for fitting multi-group models by generalized least squares.

Usage

```
objectiveML(gradient=TRUE, hessian=FALSE)
objectiveML2(gradient=TRUE)

objectiveGLS(gradient=FALSE)
objectiveGLS2(gradient=FALSE)

objectiveFIML(gradient=TRUE, hessian=FALSE)
objectiveFIML2(gradient=TRUE, hessian=FALSE)

msemObjectiveML(gradient=TRUE)
msemObjectiveML2(gradient=TRUE)

msemObjectiveGLS(gradient=FALSE)
```

Arguments

gradient	If TRUE, the object that's returned includes a function for computing an analytic gradient; there is at present no analytic gradient available for objectiveFIML, objectiveGLS, objectiveGLS2, or msemObjectiveGL.
hessian	If TRUE, the objected returned includes a function to compute an analytic Hessian; only avaiable for objectiveML and not generally recommended.

Value

These functions return an object of class "semObjective", with up to two elements:

objective	an objective function.
gradient	a gradient function.

Author(s)

John Fox <jfox@mcmaster.ca>

References

See sem.

See Also

sem, optimizers

---

optimizers *sem Optimizers*

Description

The default optimizer used by sem is optimizerSem, which employs compiled code and is integrated with the objectiveML and objectiveGLS objective functions; optimizerSem, written by Zhenghua Nie, is a modified version of the standard R nlm optimizer, which was written by Saikat DebRoy, R-core, and Richard H. Jones. The other functions call optimizers (nlm, optim, or nlminb), to fit structural equation models, and are called by the sem function. The user would not normally call these functions directly, but rather supply one of them in the optimizer argument to sem. Users may also write them own optimizer functions. msemOptimizerNlm is for fitting multigroup models, and also adapts the nlm code.

Usage

```
optimizerSem(start, objective=objectiveML,
 gradient=TRUE, maxiter, debug, par.size, model.description, warn, ...)

optimizerMsem(start, objective=msemObjectiveML, gradient=TRUE,
 maxiter, debug, par.size, model.description, warn=FALSE, ...)

optimizerNlm(start, objective=objectiveML, gradient=TRUE,
 maxiter, debug, par.size, model.description, warn, ...)

optimizerOptim(start, objective=objectiveML, gradient=TRUE,
 maxiter, debug, par.size, model.description, warn, method="CG", ...)

optimizerNlminb(start, objective=objectiveML, gradient=TRUE, maxiter,
 debug, par.size, model.description, warn, ...)

msemOptimizerNlm(start, objective=msemObjectiveML, gradient=TRUE,
  maxiter, debug, par.size, model.description, warn=FALSE, ...)
```

Arguments

start	a vector of start values for the parameters.
objective	the objective function to be optimized; see objective.functions.
gradient	TRUE if an analytic gradient is to be used (if one is available).
maxiter	the maximum number of iterations allowed.
debug	TRUE to show the iteration history and other available information about the optimization.

par.size	"startvalues" to have the optimizer scale the problem according to the magitudes of the start values (ignored by optimizerNlminb).
model.description	a list with elements describing the structural-equation model (see the code for details).
warn	if FALSE, suppress warnings during the optimization.
method	the method to be employed by the optim optimizer; the default is "CG" (conjugate-gradient).
...	additional arguments for the nlm, optim, or nlminb optimizer.

Value

An object of class "semResult", with elements:

convergence	TRUE if the optimization apparently converged.
iterations	the number of iterations required.
par	the vector of parameter estimates.
vcov	the estimated covariance matrix of the parameter estimates, based on a numeric Hessian; not supplied by optimizerNlminb.
criterion	the optimized value of the objective function.
C	the model-implied covariance or moment matrix at the parameter estimates.
A	the estimated A matrix.
P	the estimated P matrix.

Author(s)

John Fox <jfox@mcmaster.ca>, and Zhenghua Nie, in part adapting work by Saikat DebRoy, R-core, and Richard H. Jones.

See Also

sem, objective.functions, nlm, optim, nlminb

---

pathDiagram *Draw Path Diagram*

Description

pathDiagram creates a description of the path diagram for a structural-equation-model or SEM-specification object to be processed by the graph-drawing program *dot*.

Usage

```
pathDiagram(model, ...)

## S3 method for class 'sem'
pathDiagram(model, file = "pathDiagram",
    style = c("ram", "traditional"),
    output.type = c("html", "graphics", "dot"), graphics.fmt = "pdf",
    dot.options = NULL,
    size = c(8, 8), node.font = c("Helvetica", 14),
    edge.font = c("Helvetica", 10), digits = 2,
    rank.direction = c("LR", "TB"),
    min.rank = NULL, max.rank = NULL, same.rank = NULL,
    variables = model$var.names, var.labels, parameters, par.labels,
    ignore.double = TRUE, ignore.self = FALSE, error.nodes = TRUE,
    edge.labels = c("names", "values", "both"),
    edge.colors = c("black", "black"),
    edge.weight = c("fixed", "proportional"),
    node.colors = c("transparent", "transparent", "transparent"),
    standardize = FALSE, ...)

## S3 method for class 'semmod'
pathDiagram(model, obs.variables, ...)

math(text, html.only=FALSE, hat=FALSE)
```

Arguments

model	a structural-equation-model or SEM-specification object produced by sem, or, respectively, specifyEquations, specifyModel, or cfa.
...	arguments passed down, e.g., from the semmod method to the sem method.
file	a file name, by default "pathDiagram", given *without* an extension, to which to write the *dot* description of the path diagram if output.type "graphics" or "dot" is selected, and for the graphics output file (with appropriate extension) if "graphics" output is selected, in which case a ".dot" file and a graphics file of type specified by the graphics.fmt argument (below); file may include a path specification.
style	"ram" (the default) for a RAM path diagram including self-directed double-headed arrows representing variances, including error variances; or "traditional" for a path diagram including nodes representing error variables.
output.type	if "html" (the default), the path diagram will open in the user"s default web browser; if "dot", a file containing *dot* commands will be written; if "graphics", both .dot and graphics files will be written.
graphics.fmt	a graphics format recognized by the *dot* program; the default is "pdf"; graphics.fmt is also used for the extension of the graphics file that is created.
dot.options	options to be passed to the *dot* program, given as a character string.
size	the size of the graph, in inches.

node.font	font name and point-size for printing variable names.
edge.font	font name and point-size for printing arrow names or values.
digits	number of digits after the decimal point (default, 2) to which to round parameter estimates.
rank.direction	draw graph left-to-right, "LR", the default, or top-to-bottom, "TB".
min.rank	a character string listing names of variables to be assigned minimum rank (order) in the graph; the names should be separated by commas.
max.rank	a character string listing names of variables to be assigned maximum rank in the graph; the names should be separated by commas.
same.rank	a character string or vector of character strings of variables to be assigned equivalent rank in the graph; names in each string should be separated by commas.
variables	variable names; defaults to the variable names in model. If specified, the variable names should be in the same order as in model.
var.labels	a character vector with labels to be used in lieu of (some of) the variables names, for greater flexibility in labelling nodes in the graph — e.g., the labels can be created with the math function. The elements of the vector must have names corresponding to variables in the model.
parameters	parameter names; defaults to the parameter names in model. If specified, the parameter names should be in the same order as in model.
par.labels	a character vector with labels to be used in lieu of (some of) the parameter names, for greater flexibility in labelling edges in the graph — e.g., the labels can be created with the math function. The elements of the vector must have names corresponding to parameters in the model.
ignore.double	if TRUE, the default, double-headed arrows, representing variances and covariances, are not graphed.
ignore.self	if TRUE (the default is FALSE), and ignore.double=FALSE, self-directed double-headed arrows representing error variances are suppressed; note that if ignore.double=TRUE, *all* double-headed arrows, including self-directed arrows, are suppressed.
error.nodes	if TRUE (the default) and style="traditional", show the nodes representing error variables.
edge.labels	"names" to label arrows with parameter names; "values" to label arrows with parameter estimates, or "both".
edge.colors	two-element character vector giving colors of positive and negative arrows respectively; the default is c("black", "black").
edge.weight	if "proportional" (the default is "fixed"), the thickness of edges is proportional to the absolute size of the corresponding parameter estimate; this is generally sensible only if standardize=TRUE.
node.colors	a two- or three-element character vector giving colors of nodes representing exogenous, endogenous, and error variables (for traditional path diagrams) consecutively; the default is "transparent" for all three; if a two colors are given, error variables are colored as exogenous (the first color.
standardize	if TRUE, display standardized coefficients; default is FALSE.
obs.variables	a character vector with the names of the observed variables in the model.

`text` a character string or vector of character strings to be translated into node or edge label symbols. If a vector of character strings is supplied, then the elements of the vector should be named with the corresponding variable (node) or parameter (edge) name.

`html.only` If `TRUE` (the default is `FALSE`), the character strings in `text` are to be treated as an HTML character codes, in which case the prefix `"#"` and suffix `";"` are appended to each. Otherwise, `text` should only contain the names of lowercase or uppercase Greek letters, such as `"alpha"` or `"Alpha"`. The full set of Greek letters recognized is given in the file `Greek.txt` in the package's `etc` subdirectory – or type `sem:::Greek` at the R command prompt. In either case, the symbols may be followed by numeric subscripts in curly braces consisting of numerals (e.g., `"beta_{12}"`), and/or numeric superscripts (e.g., `"sigma^{2}"`, `"sigma_{1}^{2}"`). Superscripts and subscripts may also include most lowercase or uppercase letters. Depending upon your OS, subscripts and superscripts may only work properly with HTML output from `pathDiagram`, not with graphics output produced by *dot*.

`hat` If `TRUE` (the default is `FALSE`), a hat (circumflex) is placed over the symbols in `text`; this feature doesn't produce a visually appealing result.

Details

`pathDiagram` creates a description of the path diagram for a structural-equation-model or SEM-specification object to be processed by the graph-drawing program *dot*, which can be called automatically; see Koutsofios and North (2002) and https://www.graphviz.org/. To obtain graphics output directly, the *dot* program must be on the system search path.

Alternatively, *HTML* output can be created in a web browser without an independent installation of `dot` using facilities in the **DiagrammeR** package.

The `math` function can be used to create node (variable) and edge (arrow) labels with symbols such as Greek letters, subscripts, and superscripts.

The `semmod` method of `pathDiagram` sets up a call to the `sem` method.

The various arguments to `pathDiagram` can be used to customize the diagram, but if there are too many constraints on node placement, *dot* may fail to produce a graph or may produce a distorted graph. `pathDiagram` can create both RAM-style diagrams, in which variances are represented as self-directed arrows, and traditional path diagrams, in which error variables appear explicitly as nodes. As is conventional, latent variables (including error variables) are represented as ellipses and observed variables as rectangles; double-headed arrows represent covariances (and in RAM diagrams, variances) and single-headed arrows represent structural coefficients.

Value

`pathDiagram` invisibly returns a character vector containing *dot* commands. `math` returns a character vector containing suitable `HTML` markup.

Author(s)

John Fox <jfox@mcmaster.ca>, Adam Kramer, and Michael Friendly

References

Koutsofios, E., and North, S. C. (2002) Drawing graphs with *dot*. https://graphviz.org/documentation/.

See Also

sem, specifyEquations, specifyModel, cfa

Examples

```
 if (interactive()) {
# The Duncan, Haller, and Portes Peer-Influences Model

R.DHP <- readMoments(diag=FALSE, names=c("ROccAsp", "REdAsp", "FOccAsp",
                "FEdAsp", "RParAsp", "RIQ", "RSES", "FSES", "FIQ", "FParAsp"),
                text="
    .6247
    .3269  .3669
    .4216  .3275  .6404
    .2137  .2742  .1124  .0839
    .4105  .4043  .2903  .2598  .1839
    .3240  .4047  .3054  .2786  .0489  .2220
    .2930  .2407  .4105  .3607  .0186  .1861  .2707
    .2995  .2863  .5191  .5007  .0782  .3355  .2302  .2950
    .0760  .0702  .2784  .1988  .1147  .1021  .0931 -.0438  .2087
")

model.dhp <- specifyModel(text="
    RParAsp  -> RGenAsp, gam11,  NA
    RIQ      -> RGenAsp, gam12,  NA
    RSES     -> RGenAsp, gam13,  NA
    FSES     -> RGenAsp, gam14,  NA
    RSES     -> FGenAsp, gam23,  NA
    FSES     -> FGenAsp, gam24,  NA
    FIQ      -> FGenAsp, gam25,  NA
    FParAsp  -> FGenAsp, gam26,  NA
    FGenAsp  -> RGenAsp, beta12, NA
    RGenAsp  -> FGenAsp, beta21, NA
    RGenAsp  -> ROccAsp,  NA,       1
    RGenAsp  -> REdAsp,  lam21,  NA
    FGenAsp  -> FOccAsp,  NA,       1
    FGenAsp  -> FEdAsp,  lam42,  NA
    RGenAsp <-> RGenAsp, ps11,   NA
    FGenAsp <-> FGenAsp, ps22,   NA
    RGenAsp <-> FGenAsp, ps12,   NA
    ROccAsp <-> ROccAsp, theta1, NA
    REdAsp  <-> REdAsp,  theta2, NA
    FOccAsp <-> FOccAsp, theta3, NA
    FEdAsp  <-> FEdAsp,  theta4, NA
")

sem.dhp <- sem(model.dhp, R.DHP, 329,
    fixed.x=c("RParAsp", "RIQ", "RSES", "FSES", "FIQ", "FParAsp"))
```

```
pathDiagram(sem.dhp, min.rank="RIQ, RSES, RParAsp, FParAsp, FSES, FIQ",
                     max.rank="ROccAsp, REdAsp, FEdAsp, FOccAsp",
                     same.rank="RGenAsp, FGenAsp",
                     edge.labels="values")

pathDiagram(model.dhp,
    obs.variables=c("RParAsp", "RIQ", "RSES", "FSES", "FIQ",
        "FParAsp", "ROccAsp", "REdAsp", "FOccAsp", "FEdAsp"),
    style="traditional",
    node.colors=c("pink", "lightblue", "lightgreen"),
    min.rank="RIQ, RSES, RParAsp, FParAsp, FSES, FIQ",
    max.rank="ROccAsp, REdAsp, FEdAsp, FOccAsp",
    same.rank="RGenAsp, FGenAsp",
    var.labels=c(RParAsp="Respondent Parental Aspiration",
        RIQ="Respondent IQ",
        RSES="Respondent SES",
        FSES="Friend SES",
        FIQ="Friend IQ",
        FParAsp="Friend Parental Aspiration",
        ROccAsp="Respondent Occupational Aspiration",
        REdAsp="Respondent Educational Aspiration",
        RGenAsp="Respondent General Aspiration",
        FOccAsp="Friend Occupational Aspiration",
        FEdAsp="Friend Educational Aspiration",
        FGenAsp="Friend General Aspiration",
        math(c(RGenAsp.error="xi_{1}",
        FGenAsp.error="xi_{2}",
        ROccAsp.error="epsilon_{1}",
        REdAsp.error="epsilon_{2}",
        FOccAsp.error="epsilon_{3}",
        FEdAsp.error="epsilon_{4}"))),
    par.labels=math(c(gam11="gamma_{11}",
        gam12="gamma_{12}",
        gam13="gamma_{13}",
        gam14="gamma_{14}",
        gam23="gamma_{23}",
        gam24="gamma_{24}",
        gam25="gamma_{25}",
        gam26="gamma_{26}",
        beta12="beta_{12}",
        beta21="beta_{21}",
        lam21="lambda_{21}",
        lam42="lambda_{42}",
        ps11="psi_{11}",
        ps22="psi_{22}",
        ps12="psi_{12}",
        theta1="theta_{1}",
        theta2="theta_{2}",
        theta3="theta_{3}",
        theta4="theta_{4}")))

    # the following example contributed by Michael Friendly:
```

```
union <- readMoments(diag=TRUE,
    names=c('y1', 'y2', 'y3', 'x1', 'x2'), text="
14.610
-5.250  11.017
-8.057  11.087   31.971
-0.482   0.677    1.559   1.021
-18.857  17.861   28.250   7.139  215.662
")

union.mod <- specifyEquations(covs=c("x1, x2"), text="
y1 = gam12*x2
y2 = beta21*y1 + gam22*x2
y3 = beta31*y1 + beta32*y2 + gam31*x1
")

union.sem <- sem(union.mod, union, N=173)

dot <- pathDiagram(union.sem, style="traditional",
    ignore.double=FALSE, error.nodes=FALSE,
    edge.labels="values",
    min.rank=c("Years", "Age"),
    max.rank=c("Sentiment", "Sentiment.error"),
    same.rank=c("Deference, Deference.error", "Activism, Activism.error"),
    variables=c("Deference", "Activism", "Sentiment", "Years", "Age"),
    edge.colors=c("black", "red"),
    node.colors = c("pink", "lightblue"))

cat(paste(dot, collapse="\n")) # dot commands

    }
```

---

ram *RAM Matrix for a Structural-Equation Model*

Description

Print the labelled RAM definition matrix for a structural-equation model fit by sem.

Usage

```
ram(object, digits=getOption("digits"), startvalues=FALSE)
```

Arguments

object	an object of class sem returned by the sem function.
digits	number of digits for printed output.
startvalues	if TRUE, start values for parameters are printed; otherwise, the parameter estimates are printed; the default is FALSE.

Value

A data frame containing the labelled RAM definition matrix, which is normally just printed.

Author(s)

John Fox <jfox@mcmaster.ca>

See Also

sem

Examples

```
# ------------- assumes that Duncan, Haller and Portes peer-influences model
# -------------     has been fit and is in sem.dhp

    ## Not run:
ram(sem.dhp)

## End(Not run)
```

---

rawMoments *Compute Raw Moments Matrix*

Description

Computes the "uncorrected" sum-of-squares-and-products matrix divided by the number of observations.

Usage

```
## S3 method for class 'formula'
rawMoments(formula, data, subset, na.action,
    contrasts=NULL, ...)

## Default S3 method:
rawMoments(object, na.rm=FALSE, ...)

cov2raw(cov, mean, N, sd)

## S3 method for class 'rawmoments'
print(x, ...)
```

Arguments

object	a one-sided model formula or an object coercible to a numeric matrix.
formula	a one-sided model formula specifying the model matrix for which raw moments are to be computed; note that a constant is included if it is not explicitly suppressed by putting -1 in the formula.
data	an optional data frame containing the variables in the formula. By default the variables are taken from the environment from which rawMoments is called.
subset	an optional vector specifying a subset of observations to be used in computing moments.
na.action	a function that indicates what should happen when the data contain NAs. The default is set by the na.action option.
contrasts	an optional list. See the contrasts.arg argument of model.matrix.default
.	
na.rm	if TRUE, any data rows with missing data will be removed.
cov	a covariance or correlation matrix.
mean	a vector of means.
N	the number of observations on which the covariances or correlations are based.
sd	an optional vector of standard deviations, to be given if cov is a correlation matrix.
x	an object of class rawmoments to print.
...	arguments passed down.

Value

rawMoments and cov2raw return an object of class rawmoments, which is simply a matrix with an attribute "N" that contains the number of observations on which the moments are based.

Author(s)

John Fox <jfox@mcmaster.ca>

See Also

sem

Examples

```
# the following are all equivalent (with the exception of the name of the intercept):

rawMoments(cbind(1, Kmenta))

rawMoments(~ Q + P + D + F + A, data=Kmenta)

Cov <- with(Kmenta, cov(cbind(Q, P, D, F, A)))
cov2raw(Cov, colMeans(Kmenta), nrow(Kmenta))
```

readMoments *Input a Covariance, Correlation, or Raw Moment Matrix*

Description

This functions makes it simpler to input covariance, correlation, and raw-moment matrices to be analyzed by the sem function. The matrix is input in lower-triangular form on as many lines as is convenient, omitting the above-diagonal elements. The elements on the diagonal may also optionally be omitted, in which case they are taken to be 1.

Usage

```
readMoments(file="", text, diag=TRUE,
    names=as.character(paste("X", 1:n, sep = "")))
```

Arguments

file	The (quoted) file from which to read the moment matrix, including the path to the file if it is not in the current directory. If "" (the default) and the text argument is absent, then the moment matrix is read from the standard input stream, and is terminated by a blank line.
text	The moment matrix given as a character string, as an alternative to specifying the file argument or reading the moments from the input stream — e.g., when the session is not interactive and there is no standard input.
diag	If TRUE (the default), then the input matrix includes diagonal elements.
names	a character vector containing the names of the variables, to label the rows and columns of the moment matrix.

Value

Returns a lower-triangular matrix (i.e., with zeroes above the main diagonal) suitable for input to sem.

Author(s)

John Fox <jfox@mcmaster.ca>

See Also

sem

Examples

```
R.DHP <- readMoments(diag=FALSE, names=c("ROccAsp", "REdAsp", "FOccAsp",
                "FEdAsp", "RParAsp", "RIQ", "RSES", "FSES", "FIQ", "FParAsp"),
                text="
    .6247
    .3269  .3669
    .4216  .3275  .6404
    .2137  .2742  .1124  .0839
    .4105  .4043  .2903  .2598  .1839
    .3240  .4047  .3054  .2786  .0489  .2220
    .2930  .2407  .4105  .3607  .0186  .1861  .2707
    .2995  .2863  .5191  .5007  .0782  .3355  .2302  .2950
    .0760  .0702  .2784  .1988  .1147  .1021  .0931 -.0438  .2087
")
R.DHP

#the following will work only in an interactive sessions:
    ## Not run:
R.DHP <- readMoments(diag=FALSE, names=c("ROccAsp", "REdAsp", "FOccAsp",
                "FEdAsp", "RParAsp", "RIQ", "RSES", "FSES", "FIQ", "FParAsp"))
    .6247
    .3269  .3669
    .4216  .3275  .6404
    .2137  .2742  .1124  .0839
    .4105  .4043  .2903  .2598  .1839
    .3240  .4047  .3054  .2786  .0489  .2220
    .2930  .2407  .4105  .3607  .0186  .1861  .2707
    .2995  .2863  .5191  .5007  .0782  .3355  .2302  .2950
    .0760  .0702  .2784  .1988  .1147  .1021  .0931 -.0438  .2087

R.DHP

## End(Not run)
```

---

residuals.sem *Residual Covariances for a Structural Equation Model*

Description

These functions compute residual covariances, variance-standardized residual covariances, and normalized residual covariances for the observed variables in a structural-equation model fit by sem.

Usage

```
## S3 method for class 'sem'
residuals(object, ...)
## S3 method for class 'msem'
residuals(object, ...)
```

```
## S3 method for class 'sem'
standardizedResiduals(object, ...)
## S3 method for class 'msem'
standardizedResiduals(object, ...)

## S3 method for class 'objectiveML'
normalizedResiduals(object, ...)
## S3 method for class 'objectiveGLS'
normalizedResiduals(object, ...)
## S3 method for class 'msemObjectiveML'
normalizedResiduals(object, ...)
```

Arguments

`object`	an object inheriting from class `sem` or `msem` returned by the `sem` function.
`...`	not for the user.

Details

Residuals are defined as $S - C$, where S is the sample covariance matrix of the observed variables and C is the model-reproduced covariance matrix. The *standardized* residual covariance for a pair of variables divides the residual covariance by the product of the sample standard deviations of the two variables, $(s_{ij} - c_{ij})/(s_{ii}s_{jj})^{1/2}$. The *normalized* residual is given by

$$\frac{s_{ij} - c_{ij}}{[(c_{ii}c_{jj} - c_{ij}^2)/N^*]^{1/2}}$$

where N^* is the number of observations minus one if the model is fit to a covariance matrix, or the number of observations if it is fit to a raw moment matrix.

Value

Each function returns a matrix of residuals.

Author(s)

John Fox <jfox@mcmaster.ca>

References

Bollen, K. A. (1989) *Structural Equations With Latent Variables.* Wiley.

See Also

`sem`

Examples

```
# In the first example, readMoments() and specifyModel() read from the
# input stream. This example cannot be executed via example() but can be entered
# at the command prompt. The example is repeated using file input;
# this example can be executed via example().
    ## Not run:
# Duncan, Haller, and Portes peer-influences model

R.DHP <- readMoments(diag=FALSE, names=c("ROccAsp", "REdAsp", "FOccAsp",
                "FEdAsp", "RParAsp", "RIQ", "RSES", "FSES", "FIQ", "FParAsp"))
    .6247
    .3269  .3669
    .4216  .3275  .6404
    .2137  .2742  .1124  .0839
    .4105  .4043  .2903  .2598  .1839
    .3240  .4047  .3054  .2786  .0489  .2220
    .2930  .2407  .4105  .3607  .0186  .1861  .2707
    .2995  .2863  .5191  .5007  .0782  .3355  .2302  .2950
    .0760  .0702  .2784  .1988  .1147  .1021  .0931 -.0438  .2087

model.dhp <- specifyModel()
    RParAsp  -> RGenAsp, gam11,  NA
    RIQ      -> RGenAsp, gam12,  NA
    RSES     -> RGenAsp, gam13,  NA
    FSES     -> RGenAsp, gam14,  NA
    RSES     -> FGenAsp, gam23,  NA
    FSES     -> FGenAsp, gam24,  NA
    FIQ      -> FGenAsp, gam25,  NA
    FParAsp  -> FGenAsp, gam26,  NA
    FGenAsp  -> RGenAsp, beta12, NA
    RGenAsp  -> FGenAsp, beta21, NA
    RGenAsp  -> ROccAsp,  NA,     1
    RGenAsp  -> REdAsp,  lam21,  NA
    FGenAsp  -> FOccAsp,  NA,     1
    FGenAsp  -> FEdAsp,  lam42,  NA
    RGenAsp <-> RGenAsp, ps11,   NA
    FGenAsp <-> FGenAsp, ps22,   NA
    RGenAsp <-> FGenAsp, ps12,   NA
    ROccAsp <-> ROccAsp, theta1, NA
    REdAsp  <-> REdAsp,  theta2, NA
    FOccAsp <-> FOccAsp, theta3, NA
    FEdAsp  <-> FEdAsp,  theta4, NA

sem.dhp <- sem(model.dhp, R.DHP, 329,
    fixed.x=c('RParAsp', 'RIQ', 'RSES', 'FSES', 'FIQ', 'FParAsp'))
residuals(sem.dhp)
normalizedResiduals(sem.dhp)
standardizedResiduals(sem.dhp) # same as residuals because model is fit to correlations

## End(Not run)
# The following example can be executed via example():
```

```
etc <- system.file(package="sem", "etc") # path to data and model files

(R.DHP <- readMoments(file=file.path(etc, "R-DHP.txt"),
    diag=FALSE, names=c("ROccAsp", "REdAsp", "FOccAsp",
                "FEdAsp", "RParAsp", "RIQ", "RSES", "FSES", "FIQ", "FParAsp")))
(model.dhp <- specifyModel(file=file.path(etc, "model-DHP.txt")))
(sem.dhp <- sem(model.dhp, R.DHP, 329,
    fixed.x=c('RParAsp', 'RIQ', 'RSES', 'FSES', 'FIQ', 'FParAsp')))

residuals(sem.dhp)

normalizedResiduals(sem.dhp)

standardizedResiduals(sem.dhp)  # same as residuals because model is fit to correlations
```

---

sem *General Structural Equation Models*

Description

sem fits general structural equation models (with both observed and unobserved variables). Observed variables are also called *indicators* or *manifest variables*; unobserved variables are also called *factors* or *latent variables*. Normally, the generic function (sem) is called directly with a semmod first argument produced by specifyModel, specifyEquations, or cfa, invoking the sem.semmod method, which in turn sets up a call to the sem.default method; thus, the user may wish to specify arguments accepted by the semmod and default methods. Similarly, for a multigroup model, sem would normally be called with a semmodList object produced by multigroupModel as its first argument, and would then generate a call to the code msemmod method.

Usage

```
## S3 method for class 'semmod'
sem(model, S, N, data,
    raw=identical(na.action, na.pass), obs.variables=rownames(S),
   fixed.x=NULL, formula= ~ ., na.action=na.omit,
    robust=!missing(data), debug=FALSE,
    optimizer=optimizerSem, objective=objectiveML, ...)

## Default S3 method:
sem(model, S, N, raw=FALSE, data=NULL, start.fn=startvalues,
    pattern.number=NULL, valid.data.patterns=NULL,
    use.means=TRUE, param.names,
   var.names, fixed.x=NULL, robust=!is.null(data), semmod=NULL, debug=FALSE,
  analytic.gradient=!identical(objective, objectiveFIML),
        warn=FALSE, maxiter=1000, par.size=c("ones", "startvalues"),
  start.tol=1E-6, optimizer=optimizerSem, objective=objectiveML, cls, ...)

## S3 method for class 'semmodList'
```

```
sem(model, S, N, data, raw=FALSE, fixed.x=NULL,
  robust=!missing(data), formula, group="Group", debug=FALSE, ...)

## S3 method for class 'msemmod'
sem(model, S, N, start.fn=startvalues,
        group="Group", groups=names(model), raw=FALSE, fixed.x,
  param.names, var.names, debug=FALSE, analytic.gradient=TRUE, warn=FALSE,
  maxiter=5000, par.size = c("ones", "startvalues"), start.tol = 1e-06,
  start=c("initial.fit", "startvalues"), initial.maxiter=1000,
  optimizer = optimizerMsem, objective = msemObjectiveML, ...)

startvalues(S, ram, debug=FALSE, tol=1E-6)
startvalues2(S, ram, debug=FALSE, tol=1E-6)

## S3 method for class 'sem'
coef(object, standardized=FALSE, ...)
## S3 method for class 'msem'
coef(object, ...)
## S3 method for class 'sem'
vcov(object, robust=FALSE,
 analytic=inherits(object, "objectiveML") && object$t <= 500, ...)
## S3 method for class 'msem'
vcov(object, robust=FALSE,
    analytic=inherits(object, "msemObjectiveML") && object$t <= 500, ...)
## S3 method for class 'sem'
df.residual(object, ...)
## S3 method for class 'msem'
df.residual(object, ...)
```

Arguments

model	RAM specification, which is a simple encoding of the path diagram for the model. The model may be given either in symbolic form (as a semmod object, as returned by the specifyModel, specifyEquations, or cfa function, or as a character matrix), invoking sem.semmod, which calls sem.default after setting up the model, or (less conveniently) in numeric form, invoking sem.default directly, which is not recommended (see **Details** below). The model argument may also be a multigroup-model specification, as produced by multigroupModel.
S	covariance matrix among observed variables; may be input as a symmetric matrix, or as a lower- or upper-triangular matrix. S may also be a raw (i.e., "uncorrected") moment matrix — that is, a sum-of-squares-and-products matrix divided by N. This form of input is useful for fitting models with intercepts, in which case the moment matrix should include the mean square and cross-products for a unit variable all of whose entries are 1; of course, the raw mean square for the unit variable is 1. Raw-moment matrices may be computed by rawMoments. If the ram argument is given in symbolic form, then the observed-variable covariance or raw-moment matrix may contain variables that do not appear in the model, in which case a warning is printed. S may also be a list

of covariance or moment matrices for each group in a multigroup model. As an alternative to specifying S the user may supply a data frame containing the data for the model (see the argument data).

N — number of observations on which the covariance matrix is based; for a multigroup model, a vector of group Ns.

data — As a generally preferable alternative to specifying S and N, the user may supply a data frame containing the data to which the model is to be fit. In a multigroup model, the data argument may be a list of data frames or a single data frame; in the later event, the factor given as the group argument is used to split the data into groups.

start.fn — a function to compute startvalues for the free parameters of the model; two functions are supplied, startvalues and a older version, startvalues2, the first of which is the default.

na.action — a function to process missing data, if raw data are supplied in the data argument. The default is na.omit, which returns only complete cases; specify na.action=na.pass to get FIML estimates in the presence of missing data from the objectiveFIML and objectiveFIML2 objective functions.

raw — TRUE if S is a raw moment matrix or if a raw moment matrix — as opposed to a covariance matrix — is to be computed from data; the default is FALSE unless the na.action argument is set to na.pass.

pattern.number, valid.data.patterns — these arguments pass information about valid (i.e., non-missing) data patterns and normally would not be specified directly by the user.

use.means — When raw data are supplied and intercepts are included in the model, use the observed-variable means as start values for the intercepts; the default is TRUE.

obs.variables — names of observed variables, by default taken from the row names of the covariance or moment matrix S, which may be given directly or generated according to the data and formula arguments.

fixed.x — names (if the ram matrix is given in symbolic form) or indices (if it is in numeric form) of fixed exogenous variables. Specifying these obviates the necessity of having to fix the variances and covariances among these variables (and produces correct degrees of freedom for the model chisquare).

formula — a one-sided formula, to be applied to data to generate the variables for which covariances or raw moments are computed. The default formula is ~., i.e., all of the variables in the data, including an implied intercept; if a covariance matrix is to be computed, the constant is suppressed. In a multigroup model, alternatively a list one one-sided formulas as be given, to be applied individually to the groups.

robust — In sem: if TRUE, then quantities are calculated that can be used to compute robust estimates of coefficient standard errors and robust tests when the model is fit by multinormal maximum likelihood; the default is TRUE when the data argument is TRUE, and this option is only available when the data argument is given. In vcov: if TRUE, return a robust coefficient covariance matrix (if object contains the requisite information).

semmod — a semmod object containing the description of the model; optional, and normally supplied not directly by the user but via the semmod method for sem.

debug — if TRUE, some information is printed to help you debug the symbolic model specification; for example, if a variable name is misspelled, sem will assume that the variable is a (new) latent variable. Information about the optimization will also be printed, but details will vary with the optimizer employed. The default is FALSE.

... — arguments to be passed down, including from sem.default to the optimizer.

param.names — names of the t free parameters, given in their numerical order; default names are Param1, ..., Param$t$. Note: Should not be specified when the model is given in symbolic form.

var.names — names of the m entries of the v vector (typically the observed and latent variables — see below), given in their numerical order; default names are Var1, ..., Var$m$. Note: Should not be specified when the model is given in symbolic form.

analytic.gradient — if TRUE (the default, except for the objectiveFIML objective function, where, at present, an analytic gradient slows down the computation), then analytic first derivatives are used in the maximization of the likelihood if the optimzer employed will accept them; otherwise numeric derivatives are used, again if the optimizer will compute them.

warn — if TRUE, warnings produced by the optimization function will be printed. This should generally not be necessary, since sem prints its own warning, and saves information about convergence. The default is FALSE.

maxiter — the maximum number of iterations for the optimization of the objective function, to be passed to the optimizer.

par.size — the anticipated size of the free parameters; if "ones", a vector of ones is used; if "startvalues", taken from the start values. You can try changing this argument if you encounter convergence problems. The default is "startvalues" if the largest input variance is at least 100 times the smallest, and "ones" otherwise. Whether this argument is actually used depends upon the optimizer employed.

start.tol, tol — if the magnitude of an automatic start value is less than start.tol, then it is set to start.tol; defaults to 1E-6.

optimizer — a function to be used to minimize the objective function; the default for single-group models is optimizerSem. Alternatives are nlm, which employs the standard R optimizer nlm; optimizerOptim, which employs optim; and optimizerNlminb, which uses nlminb — or the user can supply an optimizer. For multigroup model, the default is optimizerMsem, and msemOptimizerNlm, based on nlm, is provided as an alternative.

objective — An objective function to be minimized, sometimes called a "fit" function in the SEM literature. The default for single-group models is objectiveML, which produces maximum-likelihood estimates assuming multinormality. An alternative is objectiveGLS, which produced generalized least squares estimates — or the user can supply an objective function to be minimized. For multigroup models, the default is available is msemObjectiveML for ML estimates and an alternative is msemObjectiveGLS for GLS estiamtes.

cls — primary class to be assigned to the result; normally this is not specified directly, but raither is inferred from the objective function.

ram	numeric RAM matrix.
object	an object of class "sem" or "msem", returned by sem.
standardized	if TRUE, return standardized coefficients.
analytic	return an analytic (as opposed to numeric) estimate of the coefficient covariance matrix; at present only available for the objectiveML objective function. The default is FALSE for this objective function if there are no more than 100 parameters and FALSE otherwise.
group	for a multigroup model, the quoted name of the group variable; if the data argument is given, snd is a single data frame, then this should be a factor in the data set or a variable coercible to a factor, to be used to split the data into groups; otherwise, the name is arbitrary.
groups	a character vector giving the names of the groups; will be ignored if group is a factor in data.
start	if "initial.fit" (the default), start values for a multi-group model are computed by first fitting the intra-group models separately by group; if "startvalues", then start values are computed as for a single-group model. In some cases, the intra-group models may not be identified even if the multi-group model is, and then start="initial.fit" should not be used.
initial.maxiter	if start="initial.fit" for a multi-group model, then initial.maxiter gives the maximum number of iterations for each initial intra-group fit.

Details

The model is set up using either RAM ("reticular action model" – don't ask!) notation – a simple format for specifying general structural equation models by coding the "arrows" in the path diagram for the model (see, e.g., McArdle and McDonald, 1984) – typically using the specifyModel function; in equation format using the specifyEquations function; or, for a simple confirmatory factor analysis model, via the cfa function. In any case, the model is represented internally in RAM format.

The variables in the v vector in the model (typically, the observed and unobserved variables, but not error variables) are numbered from 1 to m. the RAM matrix contains one row for each (free or constrained) parameter of the model, and may be specified either in symbolic format or in numeric format.

A symbolic ram matrix consists of three columns, as follows:

1. **Arrow specification:** This is a simple formula, of the form "A -> B" or, equivalently, "B <- A" for a regression coefficient (i.e., a single-headed or directional arrow); "A <-> A" for a variance or "A <-> B" for a covariance (i.e., a double-headed or bidirectional arrow). Here, A and B are variable names in the model. If a name does not correspond to an observed variable, then it is assumed to be a latent variable. Spaces can appear freely in an arrow specification, and there can be any number of hyphens in the arrows, including zero: Thus, e.g., "A->B", "A --> B", and "A>B" are all legitimate and equivalent.
2. **Parameter name:** The name of the regression coefficient, variance, or covariance specified by the arrow. Assigning the same name to two or more arrows results in an equality constraint. Specifying the parameter name as NA produces a fixed parameter.

3. **Value:** start value for a free parameter or value of a fixed parameter. If given as NA, sem will compute the start value.

It is simplest to construct the RAM matrix with the specifyModel, specifyEquations, or cfa function, all of which return an object of class semmod, and also incorporate some model-specification convenience shortcuts. This process is illustrated in the examples below.

A numeric ram matrix consists of five columns, as follows:

1. **Number of arrow heads:** 1 (directed arrow) or 2 (covariance).
2. **Arrow *to*:** index of the variable at the head of a directional arrow, or at one end of a bidirectional arrow. Observed variables should be assigned the numbers 1 to n, where n is the number of rows/columns in the covariance matrix S, with the indices corresponding to the variables' positions in S. Variable indices above n represent latent variables.
3. **Arrow *from*:** the index of the variable at the tail of a directional arrow, or at the other end of a bidirectional arrow.
4. **Parameter number:** free parameters are numbered from 1 to t, but do not necessarily appear in consecutive order. Fixed parameters are given the number 0. Equality contraints are specified by assigning two or more parameters the same number.
5. **Value:** start value for a free parameter, or value of a fixed parameter. If given as NA, the program will compute a start value, by a slight modification of the method described by McDonald and Hartmann (1992). *Note:* In some circumstances, some start values are selected randomly; this might produce small differences in the parameter estimates when the program is rerun.

The numeric ram matrix is normally generated automatically, not specified directly by the user.

For specifyEquations, each input line is either a regression equation or the specification of a variance or covariance. Regression equations are of the form

y = par1*x1 + par2*x2 + ... + park*xk

where y and the xs are variables in the model (either observed or latent), and the pars are parameters. If a parameter is given as a numeric value (e.g., 1) then it is treated as fixed. Note that no "error" variable is included in the equation; "error variances" are specified via either the covs argument, via V(y) = par (see immediately below), or are added automatically to the model when, as by default, endog.variances=TRUE.

Variances are specified in the form V(var) = par and covariances in the form C(var1, var2) = par, where the vars are variables (observed or unobserved) in the model. The symbols V and C may be in either lower- or upper-case. If par is a numeric value (e.g., 1) then it is treated as fixed. In conformity with the RAM model, a variance or covariance for an endogenous variable in the model is an "error" variance or covariance.

To set a start value for a free parameter, enclose the numeric start value in parentheses after the parameter name, as parameter(value).

sem fits the model by calling the optimizer specified in the optimizer argument to minimize the objective function specified in the objective argument. If the optimization fails to converge, a warning message is printed.

The RAM formulation of the general structural equation model is given by the basic equation

$$v = Av + u$$

where v and u are vectors of random variables (observed or unobserved), and the parameter matrix A contains regression coefficients, symbolized by single-headed arrows in a path diagram. Another parameter matrix,

$$P = E(uu')$$

contains covariances among the elements of u (assuming that the elements of u have zero means). Usually v contains endogenous and exogenous observed and unobserved variables, but not error variables (see the examples below).

The `startvalues` function may be called directly, but is usually called by `sem.default`; `startvalues2` is an older version of this function that may be used alternatively; see the `startvalues` argument to `sem`.

Value

`sem` returns an object of class `c(`*objective*`, "sem")`, where *objective* is the name of the objective function that was optimized (e.g., `"objectiveML"`), with the following elements:

`var.names`	vector of variable names.
`ram`	RAM matrix, including any rows generated for covariances among fixed exogenous variables; column 5 includes computed start values.
`S`	observed covariance matrix.
`J`	RAM selection matrix, J, which picks out observed variables.
`n.fix`	number of fixed exogenous variables.
`n`	number of observed variables.
`N`	number of observations.
`m`	number of variables (observed plus unobserved).
`t`	number of free parameters.
`raw`	`TRUE` if the model is fit to a raw moment matrix, `FALSE` otherwise.
`data`	the observed-variable data matrix, or `NULL` if data are not supplied.
`semmod`	the `semmod` specification object for the model, if one was supplied; otherwise `NULL`.
`optimizer`	the optimizer function.
`objective`	the objective function.
`coeff`	estimates of free parameters.
`vcov`	estimated asymptotic covariance matrix of parameter estimates, based on a numeric Hessian, if supplied by the optimizer; otherwise `NULL`.
`par.posn`	indices of free parameters.
`convergence`	`TRUE` or `FALSE`, depending upon whether the optimization apparently converged.
`iterations`	number of iterations performed.
`criterion`	value of the objective function at the minimum.
`C`	model-reproduced covariance matrix.
`A`	RAM A matrix.

P	RAM P matrix.
adj.obj	robust adjusted value of the objective function; NULL if robust is FALSE.
robust.vcov	robust estimated coefficient covariance matrix; NULL if robust is FALSE.

For multigroup models, sem returns an object of class c("msemObjectiveML", "msem").

Warning

A common error is to fail to specify variance or covariance terms in the model, which are denoted by double-headed arrows, <->.

In general, every observed or latent variable in the model should be associated with a variance or error variance. This may be a free parameter to estimate or a fixed constant (as in the case of a latent exogenous variable for which you wish to fix the variance, e.g., to 1). Again in general, there will be an *error variance* associated with each endogenous variable in the model (i.e., each variable to which at least one single-headed arrow points — including observed indicators of latent variables), and a *variance* associated with each exogenous variable (i.e., each variable that appears only at the tail of single-headed arrows, never at the head).

To my knowledge, the only *apparent* exception to this rule is for observed variables that are declared to be fixed exogenous variables. In this case, the program generates the necessary (fixed-constant) variances and covariances automatically.

If there are missing variances, a warning message will be printed, and estimation will almost surely fail in some manner. Missing variances might well indicate that there are missing covariances too, but it is not possible to deduce this in a mechanical manner. The specifyModel funciton will by default supply error-variance parameters if these are missing.

Author(s)

John Fox <jfox@mcmaster.ca>, Zhenghua Nie, and Jarrett Byrnes

References

Fox, J. (2006) Structural equation modeling with the sem package in R. *Structural Equation Modeling* **13**:465–486.

Bollen, K. A. (1989) *Structural Equations With Latent Variables.* Wiley.

Bollen, K. A. and Long, J. S. (eds.) *Testing Structural Equation Models*, Sage.

McArdle, J. J. and Epstein, D. (1987) Latent growth curves within developmental structural equation models. *Child Development* **58**, 110–133.

McArdle, J. J. and McDonald, R. P. (1984) Some algebraic properties of the reticular action model. *British Journal of Mathematical and Statistical Psychology* **37**, 234–251.

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

Raftery, A. E. (1993) Bayesian model selection in structural equation models. In Bollen, K. A. and Long, J. S. (eds.) *Testing Structural Equation Models*, Sage.

Raftery, A. E. (1995) Bayesian model selection in social research (with discussion). *Sociological Methodology* **25**, 111–196.

Satorra, A. (2000) Scaled and adjusted restricted tests in multi-sample analysis of moment structures. pp. 233–247 in Heijmans, R.D.H., Pollock, D.S.G. & Satorra, A. (eds.) *Innovations in Multivariate Statistical Analysis. A Festschrift for Heinz Neudecker* , Kluwer.

See Also

rawMoments, startvalues, objectiveML, objectiveGLS, optimizerNlm, optimizerOptim, optimizerNlminb, nlm, optim, nlminb, specifyModel, specifyEquations, cfa

Examples

```
# The following example illustrates the use the text argument to
#   readMoments() and specifyEquations():

R.DHP <- readMoments(diag=FALSE, names=c("ROccAsp", "REdAsp", "FOccAsp",
                "FEdAsp", "RParAsp", "RIQ", "RSES", "FSES", "FIQ", "FParAsp"),
                text="
    .6247
    .3269  .3669
    .4216  .3275  .6404
    .2137  .2742  .1124  .0839
    .4105  .4043  .2903  .2598  .1839
    .3240  .4047  .3054  .2786  .0489  .2220
    .2930  .2407  .4105  .3607  .0186  .1861  .2707
    .2995  .2863  .5191  .5007  .0782  .3355  .2302  .2950
    .0760  .0702  .2784  .1988  .1147  .1021  .0931 -.0438  .2087
")

model.dhp.1 <- specifyEquations(covs="RGenAsp, FGenAsp", text="
RGenAsp = gam11*RParAsp + gam12*RIQ + gam13*RSES + gam14*FSES + beta12*FGenAsp
FGenAsp = gam23*RSES + gam24*FSES + gam25*FIQ + gam26*FParAsp + beta21*RGenAsp
ROccAsp = 1*RGenAsp
REdAsp = lam21(1)*RGenAsp  # to illustrate setting start values
FOccAsp = 1*FGenAsp
FEdAsp = lam42(1)*FGenAsp
")

sem.dhp.1 <- sem(model.dhp.1, R.DHP, 329,
    fixed.x=c('RParAsp', 'RIQ', 'RSES', 'FSES', 'FIQ', 'FParAsp'))
summary(sem.dhp.1)

# Note: The following set of examples can't be run via example() because the default file
#  argument of specifyeEquations, specifyModel(), and readMoments() requires that the model
#  specification and covariances, correlations, or raw moments be entered in an interactive
#   session at the command prompt. The examples can be copied and run in the R console,
#   however. See ?specifyModel and ?readMoments for further information.
#   These examples are repeated below using file input to specifyModel() and
#   readMoments(). The second version of the examples may be executed through example().

    ## Not run:

# ------------- Duncan, Haller and Portes peer-influences model ----------------------
```

```
# A nonrecursive SEM with unobserved endogenous variables and fixed exogenous variables

R.DHP <- readMoments(diag=FALSE, names=c("ROccAsp", "REdAsp", "FOccAsp",
                "FEdAsp", "RParAsp", "RIQ", "RSES", "FSES", "FIQ", "FParAsp"))
    .6247
    .3269  .3669
    .4216  .3275  .6404
    .2137  .2742  .1124  .0839
    .4105  .4043  .2903  .2598  .1839
    .3240  .4047  .3054  .2786  .0489  .2220
    .2930  .2407  .4105  .3607  .0186  .1861  .2707
    .2995  .2863  .5191  .5007  .0782  .3355  .2302  .2950
    .0760  .0702  .2784  .1988  .1147  .1021  .0931 -.0438  .2087

# Fit the model using a symbolic ram specification

model.dhp <- specifyModel()
    RParAsp  -> RGenAsp, gam11,  NA
    RIQ      -> RGenAsp, gam12,  NA
    RSES     -> RGenAsp, gam13,  NA
    FSES     -> RGenAsp, gam14,  NA
    RSES     -> FGenAsp, gam23,  NA
    FSES     -> FGenAsp, gam24,  NA
    FIQ      -> FGenAsp, gam25,  NA
    FParAsp  -> FGenAsp, gam26,  NA
    FGenAsp  -> RGenAsp, beta12, NA
    RGenAsp  -> FGenAsp, beta21, NA
    RGenAsp  -> ROccAsp,  NA,     1
    RGenAsp  -> REdAsp,  lam21,  NA
    FGenAsp  -> FOccAsp,  NA,     1
    FGenAsp  -> FEdAsp,  lam42,  NA
    RGenAsp <-> RGenAsp, ps11,   NA
    FGenAsp <-> FGenAsp, ps22,   NA
    RGenAsp <-> FGenAsp, ps12,   NA
    ROccAsp <-> ROccAsp, theta1, NA
    REdAsp  <-> REdAsp,  theta2, NA
    FOccAsp <-> FOccAsp, theta3, NA
    FEdAsp  <-> FEdAsp,  theta4, NA

# an equivalent specification, allowing specifyModel() to generate
#  variance parameters for endogenous variables (and suppressing the
#  unnecessary NAs):

model.dhp <- specifyModel()
RParAsp  -> RGenAsp, gam11
RIQ      -> RGenAsp, gam12
RSES     -> RGenAsp, gam13
FSES     -> RGenAsp, gam14
RSES     -> FGenAsp, gam23
FSES     -> FGenAsp, gam24
FIQ      -> FGenAsp, gam25
FParAsp  -> FGenAsp, gam26
FGenAsp  -> RGenAsp, beta12
```

```
RGenAsp  -> FGenAsp, beta21
RGenAsp  -> ROccAsp,  NA,     1
RGenAsp  -> REdAsp,  lam21
FGenAsp  -> FOccAsp,  NA,     1
FGenAsp  -> FEdAsp,  lam42
RGenAsp <-> FGenAsp, ps12

# Another equivalent specification, telling specifyModel to add paths for
#   variances and covariance of RGenAsp and FGenAsp:

model.dhp <- specifyModel(covs="RGenAsp, FGenAsp")
RParAsp  -> RGenAsp, gam11
RIQ      -> RGenAsp, gam12
RSES     -> RGenAsp, gam13
FSES     -> RGenAsp, gam14
RSES     -> FGenAsp, gam23
FSES     -> FGenAsp, gam24
FIQ      -> FGenAsp, gam25
FParAsp  -> FGenAsp, gam26
FGenAsp  -> RGenAsp, beta12
RGenAsp  -> FGenAsp, beta21
RGenAsp  -> ROccAsp,  NA,     1
RGenAsp  -> REdAsp,  lam21
FGenAsp  -> FOccAsp,  NA,     1
FGenAsp  -> FEdAsp,  lam42

# Yet another equivalent specification using specifyEquations():

model.dhp.1 <- specifyEquations(covs="RGenAsp, FGenAsp")
RGenAsp = gam11*RParAsp + gam12*RIQ + gam13*RSES + gam14*FSES + beta12*FGenAsp
FGenAsp = gam23*RSES + gam24*FSES + gam25*FIQ + gam26*FParAsp + beta21*RGenAsp
ROccAsp = 1*RGenAsp
REdAsp = lam21(1)*RGenAsp  # to illustrate setting start values
FOccAsp = 1*FGenAsp
FEdAsp = lam42(1)*FGenAsp

sem.dhp.1 <- sem(model.dhp.1, R.DHP, 329,
    fixed.x=c('RParAsp', 'RIQ', 'RSES', 'FSES', 'FIQ', 'FParAsp'))
summary(sem.dhp.1)

# Fit the model using a numerical ram specification (not recommended!)

ram.dhp <- matrix(c(
#               heads   to      from    param  start
                1,       1,     11,      0,     1,
                1,       2,     11,      1,     NA, # lam21
                1,       3,     12,      0,     1,
                1,       4,     12,      2,     NA, # lam42
                1,      11,      5,      3,     NA, # gam11
                1,      11,      6,      4,     NA, # gam12
                1,      11,      7,      5,     NA, # gam13
                1,      11,      8,      6,     NA, # gam14
                1,      12,      7,      7,     NA, # gam23
```

```
                1,      12,      8,      8,     NA, # gam24
                1,      12,      9,      9,     NA, # gam25
                1,      12,     10,     10,     NA, # gam26
                1,      11,     12,     11,     NA, # beta12
                1,      12,     11,     12,     NA, # beta21
                2,       1,      1,     13,     NA, # theta1
                2,       2,      2,     14,     NA, # theta2
                2,       3,      3,     15,     NA, # theta3
                2,       4,      4,     16,     NA, # theta4
                2,      11,     11,     17,     NA, # psi11
                2,      12,     12,     18,     NA, # psi22
                2,      11,     12,     19,     NA  # psi12
                ), ncol=5, byrow=TRUE)

params.dhp <- c('lam21', 'lam42', 'gam11', 'gam12', 'gam13', 'gam14',
                'gam23',  'gam24',  'gam25',  'gam26',
                'beta12', 'beta21', 'theta1', 'theta2', 'theta3', 'theta4',
                'psi11', 'psi22', 'psi12')

vars.dhp <- c('ROccAsp', 'REdAsp', 'FOccAsp', 'FEdAsp', 'RParAsp', 'RIQ',
              'RSES', 'FSES', 'FIQ', 'FParAsp', 'RGenAsp', 'FGenAsp')

sem.dhp.2 <- sem(ram.dhp, R.DHP, 329, param.names=params.dhp, var.names=vars.dhp,
 fixed.x=5:10)
summary(sem.dhp.2)


# -------------------- Wheaton et al. alienation data ----------------------


S.wh <- readMoments(names=c('Anomia67','Powerless67','Anomia71',
                                'Powerless71','Education','SEI'))
   11.834
    6.947    9.364
    6.819    5.091   12.532
    4.783    5.028    7.495    9.986
   -3.839   -3.889   -3.841   -3.625   9.610
  -21.899  -18.831  -21.748  -18.775  35.522  450.288

# This is the model in the SAS manual for PROC CALIS: A Recursive SEM with
# latent endogenous and exogenous variables.
# Curiously, both factor loadings for two of the latent variables are fixed.

model.wh.1 <- specifyModel()
    Alienation67   ->  Anomia67,      NA,     1
    Alienation67   ->  Powerless67,   NA,     0.833
    Alienation71   ->  Anomia71,      NA,     1
    Alienation71   ->  Powerless71,   NA,     0.833
    SES            ->  Education,     NA,     1
    SES            ->  SEI,           lamb,   NA
    SES            ->  Alienation67,  gam1,   NA
    Alienation67   ->  Alienation71,  beta,   NA
    SES            ->  Alienation71,  gam2,   NA
```

```
    Anomia67       <-> Anomia67,      the1,   NA
    Anomia71       <-> Anomia71,      the1,   NA
    Powerless67    <-> Powerless67,   the2,   NA
    Powerless71    <-> Powerless71,   the2,   NA
    Education      <-> Education,     the3,   NA
    SEI            <-> SEI,           the4,   NA
    Anomia67       <-> Anomia71,      the5,   NA
    Powerless67    <-> Powerless71,   the5,   NA
    Alienation67   <-> Alienation67,  psi1,   NA
    Alienation71   <-> Alienation71,  psi2,   NA
    SES            <-> SES,           phi,    NA

sem.wh.1 <- sem(model.wh.1, S.wh, 932)
summary(sem.wh.1)

# The same model in equation format:

model.wh.1 <- specifyEquations()
Anomia67 = 1*Alienation67
Powerless67 = 0.833*Alienation67
Anomia71 = 1*Alienation71
Powerless71 = 0.833*Alienation71
Education = 1*SES
SEI = lamb*SES
Alienation67 = gam1*SES
Alienation71 = gam2*SES + beta*Alienation67
V(Anomia67) = the1
V(Anomia71) = the1
V(Powerless67) = the2
V(Powerless71) = the2
V(SES) = phi
C(Anomia67, Anomia71) = the5
C(Powerless67, Powerless71) = the5

# The same model, but treating one loading for each latent variable as free
# (and equal to each other).

model.wh.2 <- specifyModel()
    Alienation67   ->  Anomia67,      NA,        1
    Alienation67   ->  Powerless67,   lamby,    NA
    Alienation71   ->  Anomia71,      NA,        1
    Alienation71   ->  Powerless71,   lamby,    NA
    SES            ->  Education,     NA,        1
    SES            ->  SEI,           lambx,    NA
    SES            ->  Alienation67,  gam1,     NA
    Alienation67   ->  Alienation71,  beta,     NA
    SES            ->  Alienation71,  gam2,     NA
    Anomia67       <-> Anomia67,      the1,     NA
    Anomia71       <-> Anomia71,      the1,     NA
    Powerless67    <-> Powerless67,   the2,     NA
    Powerless71    <-> Powerless71,   the2,     NA
    Education      <-> Education,     the3,     NA
    SEI            <-> SEI,           the4,     NA
```

```
    Anomia67       <-> Anomia71,      the5,     NA
    Powerless67    <-> Powerless71,   the5,     NA
    Alienation67   <-> Alienation67,  psi1,     NA
    Alienation71   <-> Alienation71,  psi2,     NA
    SES            <-> SES,           phi,      NA


sem.wh.2 <- sem(model.wh.2, S.wh, 932)
summary(sem.wh.2)

# And again, in equation format:

model.wh <- specifyEquations()
Anomia67 = 1*Alienation67
Powerless67 = lamby*Alienation67
Anomia71 = 1*Alienation71
Powerless71 = lamby*Alienation71
Education = 1*SES
SEI = lambx*SES
Alienation67 = gam1*SES
Alienation71 = gam2*SES + beta*Alienation67
V(Anomia67) = the1
V(Anomia71) = the1
V(Powerless67) = the2
V(Powerless71) = the2
V(SES) = phi
C(Anomia67, Anomia71) = the5
C(Powerless67, Powerless71) = the5


# Compare the two models by a likelihood-ratio test:

anova(sem.wh.1, sem.wh.2)


# ---------------------- Thurstone data --------------------------------------
#  Second-order confirmatory factor analysis, from the SAS manual for PROC CALIS

R.thur <- readMoments(diag=FALSE, names=c('Sentences','Vocabulary',
        'Sent.Completion','First.Letters','4.Letter.Words','Suffixes',
        'Letter.Series','Pedigrees', 'Letter.Group'))
    .828
    .776   .779
    .439   .493    .46
    .432   .464    .425   .674
    .447   .489    .443   .59    .541
    .447   .432    .401   .381    .402   .288
    .541   .537    .534   .35    .367   .32   .555
    .38   .358    .359   .424    .446   .325   .598   .452

model.thur <- specifyModel()
    F1 -> Sentences,                    lam11
    F1 -> Vocabulary,                   lam21
```

```
    F1 -> Sent.Completion,               lam31
    F2 -> First.Letters,                 lam42
    F2 -> 4.Letter.Words,                lam52
    F2 -> Suffixes,                      lam62
    F3 -> Letter.Series,                 lam73
    F3 -> Pedigrees,                     lam83
    F3 -> Letter.Group,                  lam93
    F4 -> F1,                            gam1
    F4 -> F2,                            gam2
    F4 -> F3,                            gam3
    F1 <-> F1,                           NA,     1
    F2 <-> F2,                           NA,     1
    F3 <-> F3,                           NA,     1
    F4 <-> F4,                           NA,     1

sem.thur <- sem(model.thur, R.thur, 213)
summary(sem.thur)

# The model in equation format:

model.thur <- specifyEquations()
Sentences = lam11*F1
Vocabulary = lam21*F1
Sent.Completion = lam31*F1
First.Letters = lam42*F2
4.Letter.Words = lam52*F2
Suffixes = lam62*F2
Letter.Series = lam73*F3
Pedigrees = lam83*F3
Letter.Group = lam93*F3
F1 = gam1*F4
F2 = gam2*F4
F3 = gam3*F4
V(F1) = 1
V(F2) = 1
V(F3) = 1
V(F4) = 1


#------------------------ Kerchoff/Kenney path analysis ---------------------
# An observed-variable recursive SEM from the LISREL manual

R.kerch <- readMoments(diag=FALSE, names=c('Intelligence','Siblings',
                       'FatherEd','FatherOcc','Grades','EducExp','OccupAsp'))
    -.100
     .277  -.152
     .250  -.108  .611
     .572  -.105  .294   .248
     .489  -.213  .446   .410   .597
     .335  -.153  .303   .331   .478   .651

model.kerch <- specifyModel()
    Intelligence -> Grades,       gam51
```

```
    Siblings -> Grades,           gam52
    FatherEd -> Grades,           gam53
    FatherOcc -> Grades,          gam54
    Intelligence -> EducExp,      gam61
    Siblings -> EducExp,          gam62
    FatherEd -> EducExp,          gam63
    FatherOcc -> EducExp,         gam64
    Grades -> EducExp,            beta65
    Intelligence -> OccupAsp,     gam71
    Siblings -> OccupAsp,         gam72
    FatherEd -> OccupAsp,         gam73
    FatherOcc -> OccupAsp,        gam74
    Grades -> OccupAsp,           beta75
    EducExp -> OccupAsp,          beta76

sem.kerch <- sem(model.kerch, R.kerch, 737,
    fixed.x=c('Intelligence', 'Siblings', 'FatherEd', 'FatherOcc'))
summary(sem.kerch)

# The model in equation format:

model.kerch <- specifyEquations()
Grades = gam51*Intelligence + gam52*Siblings + gam53*FatherEd
          + gam54*FatherOcc
EducExp = gam61*Intelligence + gam62*Siblings + gam63*FatherEd
           + gam64*FatherOcc + beta65*Grades
OccupAsp = gam71*Intelligence + gam72*Siblings + gam73*FatherEd
           + gam74*FatherOcc + beta75*Grades + beta76*EducExp


#------------------- McArdle/Epstein latent-growth-curve model -----------------
# This model, from McArdle and Epstein (1987, p.118), illustrates the use of a
# raw moment matrix to fit a model with an intercept. (The example was suggested
# by Mike Stoolmiller.)

M.McArdle <- readMoments(
    names=c('WISC1', 'WISC2', 'WISC3', 'WISC4', 'UNIT'))
    365.661
    503.175     719.905
    675.656     958.479    1303.392
    890.680    1265.846    1712.475    2278.257
     18.034      25.819      35.255      46.593     1.000

mod.McArdle <- specifyModel()
    C -> WISC1, NA, 6.07
    C -> WISC2, B2, NA
    C -> WISC3, B3, NA
    C -> WISC4, B4, NA
    UNIT -> C, Mc, NA
    C <-> C, Vc, NA,
    WISC1 <-> WISC1, Vd, NA
    WISC2 <-> WISC2, Vd, NA
    WISC3 <-> WISC3, Vd, NA
```

```
    WISC4 <-> WISC4, Vd, NA

sem.McArdle <- sem(mod.McArdle, M.McArdle, 204, fixed.x="UNIT", raw=TRUE)
summary(sem.McArdle)

# The model in equation format:

mod.McArdle <- specifyEquations()
WISC1 = 6.07*C
WISC2 = B2*C
WISC3 = B3*C
WISC4 = b4*C
C = Mc*UNIT
v(C) = Vc
v(WISC1) = Vd
v(WISC2) = Vd
v(WISC3) = Vd
v(WISC4) = Vd


#------------ Bollen industrialization and democracy example -----------------
# This model, from Bollen (1989, Ch. 8), illustrates the use in sem() of a
# case-by-variable data (see ?Bollen) set rather than a covariance or moment matrix

model.bollen <- specifyModel()
 Demo60 -> y1, NA, 1
 Demo60 -> y2, lam2,
 Demo60 -> y3, lam3,
 Demo60 -> y4, lam4,
 Demo65 -> y5, NA, 1
 Demo65 -> y6, lam2,
 Demo65 -> y7, lam3,
 Demo65 -> y8, lam4,
 Indust -> x1, NA, 1
 Indust -> x2, lam6,
 Indust -> x3, lam7,
 y1 <-> y5, theta15
 y2 <-> y4, theta24
 y2 <-> y6, theta26
 y3 <-> y7, theta37
 y4 <-> y8, theta48
 y6 <-> y8, theta68
 Indust -> Demo60, gamma11,
 Indust -> Demo65, gamma21,
 Demo60 -> Demo65, beta21,
 Indust <-> Indust, phi

sem.bollen <- sem(model.bollen, data=Bollen)
summary(sem.bollen)
summary(sem.bollen, robust=TRUE) # robust SEs and tests
summary(sem.bollen, analytic.se=FALSE) # uses numeric rather than analytic Hessian

  # GLS rather than ML estimator:
```

```
sem.bollen.gls <- sem(model.bollen, data=Bollen, objective=objectiveGLS)
summary(sem.bollen.gls)

# The model in equation format:

model.bollen <- specifyEquations()
y1 = 1*Demo60
y2 = lam2*Demo60
y3 = lam3*Demo60
y4 = lam4*Demo60
y5 = 1*Demo65
y6 = lam2*Demo65
y7 = lam3*Demo65
y8 = lam4*Demo65
x1 = 1*Indust
x2 = lam6*Indust
x3 = lam7*Indust
c(y1, y5) = theta15
c(y2, y4) = theta24
c(y2, y6) = theta26
c(y3, y7) = theta37
c(y4, y8) = theta48
c(y6, y8) = theta68
Demo60 = gamma11*Indust
Demo65 = gamma21*Indust + beta21*Demo60
v(Indust) = phi


# -------------- A simple CFA model for the Thurstone mental tests data --------------

R.thur <- readMoments(diag=FALSE,
  names=c('Sentences','Vocabulary',
          'Sent.Completion','First.Letters','4.Letter.Words','Suffixes',
          'Letter.Series','Pedigrees', 'Letter.Group'))
.828
.776   .779
.439   .493    .46
.432   .464    .425   .674
.447   .489    .443   .59    .541
.447   .432    .401   .381    .402   .288
.541   .537    .534   .35    .367   .32   .555
.38   .358    .359   .424    .446   .325   .598   .452

 #  (1) in CFA format:

mod.cfa.thur.c <- cfa(reference.indicators=FALSE)
FA: Sentences, Vocabulary, Sent.Completion
FB: First.Letters, 4.Letter.Words, Suffixes
FC: Letter.Series, Pedigrees, Letter.Group

cfa.thur.c <- sem(mod.cfa.thur.c, R.thur, 213)
summary(cfa.thur.c)
```

```
 #  (2) in equation format:

mod.cfa.thur.e <- specifyEquations(covs="F1, F2, F3")
Sentences = lam11*F1
Vocabulary = lam21*F1
Sent.Completion = lam31*F1
First.Letters = lam42*F2
4.Letter.Words = lam52*F2
Suffixes = lam62*F2
Letter.Series = lam73*F3
Pedigrees = lam83*F3
Letter.Group = lam93*F3
V(F1) = 1
V(F2) = 1
V(F3) = 1

cfa.thur.e <- sem(mod.cfa.thur.e, R.thur, 213)
summary(cfa.thur.e)

 #  (3) in path format:

mod.cfa.thur.p <- specifyModel(covs="F1, F2, F3")
F1 -> Sentences,                    lam11
F1 -> Vocabulary,                   lam21
F1 -> Sent.Completion,              lam31
F2 -> First.Letters,                lam41
F2 -> 4.Letter.Words,               lam52
F2 -> Suffixes,                     lam62
F3 -> Letter.Series,                lam73
F3 -> Pedigrees,                    lam83
F3 -> Letter.Group,                 lam93
F1 <-> F1,                          NA,     1
F2 <-> F2,                          NA,     1
F3 <-> F3,                          NA,     1

cfa.thur.p <- sem(mod.cfa.thur.p, R.thur, 213)
summary(cfa.thur.p)

# -----  a CFA model fit by FIML to the mental-tests dataset with missing data -----

mod.cfa.tests <- cfa(raw=TRUE)
verbal: x1, x2, x3
math: y1, y2, y3

cfa.tests <- sem(mod.cfa.tests, data=Tests, na.action=na.pass,
               objective=objectiveFIML, fixed.x="Intercept")
summary(cfa.tests)
summary(cfa.tests, saturated=TRUE) # takes time to fit saturated model for comparison


# ---  a multigroup CFA model fit to the Holzinger-Swineford mental-tests data  -----

mod.hs <- cfa()
```

```
spatial: visual, cubes, paper, flags
verbal: general, paragrap, sentence, wordc, wordm
memory: wordr, numberr, figurer, object, numberf, figurew
math: deduct, numeric, problemr, series, arithmet

mod.mg <- multigroupModel(mod.hs, groups=c("Female", "Male"))

sem.mg <- sem(mod.mg, data=HS.data, group="Gender",
              formula = ~ visual + cubes + paper + flags +
              general + paragrap + sentence + wordc + wordm +
              wordr + numberr + figurer + object + numberf + figurew +
              deduct + numeric + problemr + series + arithmet
              )
summary(sem.mg)

 # with cross-group equality constraints:

mod.mg.eq <- multigroupModel(mod.hs, groups=c("Female", "Male"), allEqual=TRUE)

sem.mg.eq <- sem(mod.mg.eq, data=HS.data, group="Gender",
              formula = ~ visual + cubes + paper + flags +
                general + paragrap + sentence + wordc + wordm +
                wordr + numberr + figurer + object + numberf + figurew +
                deduct + numeric + problemr + series + arithmet
              )
summary(sem.mg.eq)

anova(sem.mg, sem.mg.eq) # test equality constraints

## End(Not run)

## ============================================================================

# The following examples use file input and may be executed via example():

etc <- system.file(package="sem", "etc") # path to data and model files

#   to get all fit indices (not recommended, but for illustration):

opt <- options(fit.indices = c("GFI", "AGFI", "RMSEA", "NFI", "NNFI",
         "CFI", "RNI", "IFI", "SRMR", "AIC", "AICc", "BIC", "CAIC"))

# ------------- Duncan, Haller and Portes peer-influences model ----------------------
# A nonrecursive SEM with unobserved endogenous variables and fixed exogenous variables

(R.DHP <- readMoments(file=file.path(etc, "R-DHP.txt"),
    diag=FALSE, names=c("ROccAsp", "REdAsp", "FOccAsp",
                "FEdAsp", "RParAsp", "RIQ", "RSES", "FSES", "FIQ", "FParAsp")))
(model.dhp <- specifyModel(file=file.path(etc, "model-DHP.txt")))
sem.dhp.1 <- sem(model.dhp, R.DHP, 329,
    fixed.x=c('RParAsp', 'RIQ', 'RSES', 'FSES', 'FIQ', 'FParAsp'))
summary(sem.dhp.1)
```

```
# -------------------- Wheaton et al. alienation data ----------------------

(S.wh <- readMoments(file=file.path(etc, "S-Wheaton.txt"),
     names=c('Anomia67','Powerless67','Anomia71',
                           'Powerless71','Education','SEI')))

# This is the model in the SAS manual for PROC CALIS: A Recursive SEM with
# latent endogenous and exogenous variables.
# Curiously, both factor loadings for two of the latent variables are fixed.

(model.wh.1 <- specifyModel(file=file.path(etc, "model-Wheaton-1.txt")))
sem.wh.1 <- sem(model.wh.1, S.wh, 932)
summary(sem.wh.1)

# The same model, but treating one loading for each latent variable as free
# (and equal to each other).

(model.wh.2 <- specifyModel(file=file.path(etc, "model-Wheaton-2.txt")))
sem.wh.2 <- sem(model.wh.2, S.wh, 932)
summary(sem.wh.2)

# Compare the two models by a likelihood-ratio test:

anova(sem.wh.1, sem.wh.2)


# ---------------------- Thurstone data --------------------------------------

#  Second-order confirmatory factor analysis, from the SAS manual for PROC CALIS

(R.thur <- readMoments(file=file.path(etc, "R-Thurstone.txt"),
  diag=FALSE, names=c('Sentences','Vocabulary',
        'Sent.Completion','First.Letters','4.Letter.Words','Suffixes',
        'Letter.Series','Pedigrees', 'Letter.Group')))
(model.thur <- specifyModel(file=file.path(etc, "model-Thurstone.txt")))
sem.thur <- sem(model.thur, R.thur, 213)
summary(sem.thur)


#------------------------ Kerchoff/Kenney path analysis ---------------------

# An observed-variable recursive SEM from the LISREL manual

(R.kerch <- readMoments(file=file.path(etc, "R-Kerchoff.txt"),
        diag=FALSE, names=c('Intelligence','Siblings',
                      'FatherEd','FatherOcc','Grades','EducExp','OccupAsp')))
(model.kerch <- specifyModel(file=file.path(etc, "model-Kerchoff.txt")))
sem.kerch <- sem(model.kerch, R.kerch, 737,
    fixed.x=c('Intelligence', 'Siblings', 'FatherEd', 'FatherOcc'))
summary(sem.kerch)
```

```
#------------------- McArdle/Epstein latent-growth-curve model -----------------

# This model, from McArdle and Epstein (1987, p.118), illustrates the use of a
# raw moment matrix to fit a model with an intercept. (The example was suggested
# by Mike Stoolmiller.)

(M.McArdle <- readMoments(file=file.path(etc, "M-McArdle.txt"),
    names=c('WISC1', 'WISC2', 'WISC3', 'WISC4', 'UNIT')))
(mod.McArdle <- specifyModel(file=file.path(etc, "model-McArdle.txt")))
sem.McArdle <- sem(mod.McArdle, M.McArdle, 204, fixed.x="UNIT", raw=TRUE)
summary(sem.McArdle)


#------------ Bollen industrialization and democracy example -----------------

# This model, from Bollen (1989, Ch. 8), illustrates the use in sem() of a
# case-by-variable data set (see ?Bollen) rather than a covariance or moment matrix

(model.bollen <- specifyModel(file=file.path(etc, "model-Bollen.txt")))
sem.bollen <- sem(model.bollen, data=Bollen)
summary(sem.bollen)
summary(sem.bollen, robust=TRUE) # robust SEs and tests
summary(sem.bollen, analytic.se=FALSE) # uses numeric rather than analytic Hessian

  # GLS rather than ML estimator:
sem.bollen.gls <- sem(model.bollen, data=Bollen, objective=objectiveGLS)
summary(sem.bollen.gls)

# -----  a CFA model fit by FIML to the mental-tests dataset with missing data -----

(mod.cfa.tests <- cfa(file=file.path(etc, "model-Tests.txt"), raw=TRUE))
cfa.tests <- sem(mod.cfa.tests, data=Tests, na.action=na.pass,
                optimizer=optimizerNlm, objective=objectiveFIML, fixed.x="Intercept")
summary(cfa.tests)

#------------ Holzinger and Swineford muiltigroup CFA example ----------------

mod.hs <- cfa(file=file.path(etc, "model-HS.txt"))

mod.mg <- multigroupModel(mod.hs, groups=c("Female", "Male"))

sem.mg <- sem(mod.mg, data=HS.data, group="Gender",
              formula = ~ visual + cubes + paper + flags +
              general + paragrap + sentence + wordc + wordm +
              wordr + numberr + figurer + object + numberf + figurew +
              deduct + numeric + problemr + series + arithmet
              )
summary(sem.mg)

 # with cross-group equality constraints:

mod.mg.eq <- multigroupModel(mod.hs, groups=c("Female", "Male"), allEqual=TRUE)
```

```
sem.mg.eq <- sem(mod.mg.eq, data=HS.data, group="Gender",
              formula = ~ visual + cubes + paper + flags +
                general + paragrap + sentence + wordc + wordm +
                wordr + numberr + figurer + object + numberf + figurew +
                deduct + numeric + problemr + series + arithmet
              )
summary(sem.mg.eq)

anova(sem.mg, sem.mg.eq) # test equality constraints

options(opt) # restore fit.indices option
```

`sem-deprecated` *Deprecated Functions in the sem Package*

Description

These functions are provided for compatibility with older versions of the **sem** package only, and may be removed eventually. Although an effort has been made to insure backwards-compatibility, commands that worked in versions of the **sem** package prior to version 2.0-0 will not necessarily work in version 2.0-0 and beyond, or may not work in the same manner.

Usage

```
boot.sem(...)
mod.indices(...)
normalized.residuals(...)
path.diagram(...)
raw.moments(...)
read.moments(...)
specify.model(...)
standardized.coefficients(...)
standardized.residuals(...)
std.coef(...)
```

Arguments

`...`	pass arguments down to replacements for deprecated functions.

Details

`boot.sem` is now a synonym for the `bootSem` function.

`mod.indices` is now a synonym for `modIndices`.

`normalized.residuals` is now a synonym for `normalizedResiduals`.

`path.diagram` is now a synonym for `pathDiagram`.

`raw.moments` is now a synonym for `rawMoments`.

`read.moments` is now a synonym for `readMoments`.

`specify.model` is now a synonym for `specifyModel`.

`standardized.coefficients` and `std.coef` are now synonyms for the `standardizedCoefficients` and `stdCoef` functions.

`standardized.residuals` is now a synonym for `standardizedResiduals`.

`specifyModel` *Specify a Structural Equation Model*

Description

Create the RAM specification of a structural equation model.

Usage

```
specifyModel(file="", text, exog.variances=FALSE, endog.variances=TRUE, covs,
 suffix="", quiet=FALSE)

specifyEquations(file="", text, ...)

cfa(file="", text, covs=paste(factors, collapse=","),
    reference.indicators=TRUE, raw=FALSE,
    subscript=c("name", "number"), ...)

multigroupModel(..., groups=names(models), allEqual=FALSE)

classifyVariables(model)

removeRedundantPaths(model, warn=TRUE)
## S3 method for class 'semmod'
combineModels(..., warn=TRUE)
## S3 method for class 'semmod'
update(object, file = "", text, ...)
## S3 method for class 'semmod'
edit(name, ...)

## S3 method for class 'semmod'
print(x, ...)
## S3 method for class 'semmodList'
print(x, ...)
```

Arguments

`file`	The (quoted) file from which to read the model specification, including the path to the file if it is not in the current directory. If `""` (the default) and the `text` argument is not supplied, then the specification is read from the standard input stream, and is terminated by a blank line.

`text` The model specification given as a character string, as an alternative to specifying the `]file` argument or reading the model specification from the input stream — e.g., when the session is not interactive and there is no standard input.

`exog.variances` If `TRUE` (the default is `FALSE`), free variance parameters are added for the exogenous variables that lack them.

`endog.variances`
If `TRUE` (the default), free error-variance parameters are added for the endogenous variables that lack them.

`covs` optional: a character vector of one or more elements, with each element giving a string of variable names, separated by commas. Variances and covariances among all variables in each such string are added to the model. For confirmatory factor analysis models specified via `cfa`, `covs` defaults to all of the factors in the model, thus specifying all variances and covariances among these factors. *Warning*: `covs="x1, x2"` and `covs=c("x1", "x2")` are *not* equivalent: `covs="x1, x2"` specifies the variance of `x1`, the variance of `x2`, *and* their covariance, while `covs=c("x1", "x2")` specifies the variance of `x1` and the variance of `x2` *but not* their covariance.

`suffix` a character string (defaulting to an empty string) to be appended to each parameter name; this can be convenient for specifying multiple-group models.

`reference.indicators`
if `FALSE`, the default, variances of factors are set to 1 by `cfa`; if `TRUE`, variances of factors are free parameters to estimate from the data, and instead the first factor loading for each factor is set to 1 to identify the model.

`raw` if `TRUE` (the default is `FALSE`), a path from `Intercept` to each observed variable is added to the model, and the raw second moment for `Intercept` is fixed to `1`. The `sem` function should then be called with `raw=TRUE`, and either supplied with a data set (via the `data` argument) or a raw-moment matrix (via the `S` argument).

`subscript` The "subscripts" to be appended to `lam` to name factor-loading parameters, either `"name"` (the default) to use the names of observed variables, or `"number"` to number the parameters serially within each factor. Using `"number"` produces shorter parameter names.

`quiet` if `FALSE`, the default, then the number of input lines is reported and a message is printed suggesting that `specifyEquations` or `cfa` be used.

`x`, `model`, `object`, `name`
An object of class `semmod` or `semmodList`, as produced by `specifyModel` or `multigroupModel`.

`warn` print a warning if redundant paths are detected.

`...` For `multigroupModel`, one or more optionally named arguments each of which is a `semmod` object produced, e.g., by `specifyModel`, `specifyEquations`, or `cfa`; if only one such model is given, then it will be used for all groups defined by the `groups` argument. If parameters have the same name in different groups, then they will be constrained to be equal. For `specifyEquations` and `cfa`, arguments (such as `covs`, in the case of `specifyEquations`) to be passed to `specifyModel`; for `combineModels`, `sem` objects; ignored in the `update` and `print` methods.

groups — a character vector of names for the groups in a multigroup model; taken by default from the names of the ... arguments.

allEqual — if FALSE (the default), then if only one model object is given for a multigroup model, all corresponding parameters in the groups will be distinct; if TRUE, all corresponding parameters will be constrained to be equal.

Details

The principal functions for model specification are specifyModel, to specify a model in RAM (path) format via single- and double-headed arrows; specifyEquations, to specify a model in equation format, which is then translated by the function into RAM format; and cfa, for compact specification of simple confirmatory factor analysis models.

specifyModel:

Each line of the RAM specification for specifyModel consists of three (unquoted) entries, separated by commas:

1. **Arrow specification:** This is a simple formula, of the form A -> B or, equivalently, B <- A for a regression coefficient (i.e., a single-headed or directional arrow); A <-> A for a variance or A <-> B for a covariance (i.e., a double-headed or bidirectional arrow). Here, A and B are variable names in the model. If a name does not correspond to an observed variable, then it is assumed to be a latent variable. Spaces can appear freely in an arrow specification, and there can be any number of hyphens in the arrows, including zero: Thus, e.g., A->B, A --> B, and A>B are all legitimate and equivalent.
2. **Parameter name:** The name of the regression coefficient, variance, or covariance specified by the arrow. Assigning the same name to two or more arrows results in an equality constraint. Specifying the parameter name as NA produces a fixed parameter.
3. **Value:** start value for a free parameter or value of a fixed parameter. If given as NA (or simply omitted), sem will compute the start value.

Lines may end in a comment following #.

specifyEquations:

For specifyEquations, each input line is either a regression equation or the specification of a variance or covariance. Regression equations are of the form

y = par1*x1 + par2*x2 + ... + park*xk

where y and the xs are variables in the model (either observed or latent), and the pars are parameters. If a parameter is given as a numeric value (e.g., 1) then it is treated as fixed. Note that no "error" variable is included in the equation; "error variances" are specified via either the covs argument, via V(y) = par (see immediately below), or are added automatically to the model when, as by default, endog.variances=TRUE. A regression equation may be split over more than one input by breaking at a +, so that + is either the last non-blank character on a line or the first non-blank character on the subsequent line.

Variances are specified in the form V(var) = par and covariances in the form C(var1, var2) = par, where the vars are variables (observed or unobserved) in the model. The symbols V and C may be in either lower- or upper-case. If par is a numeric value (e.g., 1) then it is treated as fixed. In conformity with the RAM model, a variance or covariance for an endogenous variable in the model is an "error" variance or covariance.

Warning: If the covs argument to specifyEquations is used to specify variances and covariances, please be aware that covs="x1, x2" and covs=c("x1", "x2") are *not* equivalent: covs="x1, x2" specifies the variance of x1, the variance of x2, *and* their covariance, while covs=c("x1", "x2") specifies the variance of x1 and the variance of x2 *but not* their covariance.

To set a start value for a free parameter, enclose the numeric start value in parentheses after the parameter name, as parameter(value).

cfa:

For cfa, each input line includes the names of the variables, separated by commas, that load on the corresponding factor; the name of the factor is given optionally at the beginning of the line, followed by a colon. If necessary, the variables that load on a factor may be continued across two or more input lines; in this case, each such line but the last must end in a comma. A variable may load on more than one factor (as long as the resulting model is identified, of course), but each factor may appear in only one input line (or set of input lines, if the variable list is continued onto the next line).

Equality constraints for factor loadings can be set by using equal-signs (=) rather than commas to separate observed variable names. For example, fac1: x1=x2=x3, x4=x5 sets the loadings for x1, x2, and x3 equal to each other, and the loadings for x4 and x5 equal to each other.

Equality constraints among error variances can similarly be specified by using var: or variance: at the beginning of a line (actually, any character string beginning with var will do, and thus no factor name may begin with the characters var). For example, var: x1=x2=x3, x4=x5 sets the error variances for x1, x2, and x3 equal to each other, and the error variances for x4 and x5 equal to each other. There may be several lines beginning with var:.

If the argument reference.indicators=FALSE, the default, cfa will fix the variance of each factor to 1, and by default include covariances (i.e., correlations) among all pairs of factors. Alternatively, if reference.indicators=TRUE, then the factor variances are free parameters to be estimated from the data, and the first loading for each factor is set to 1 to identify the model. These two approaches produce equivalent models, with the same fit to the data, but alternative parametrizations. Specifying the argument covs=NULL implicitly fixes the factor intercorrelations to 0.

See sem and the examples for further details on model specification.

Other Functions:

classifyVariables classifies the variables in a model as endogenous or exogenous.

combineModels and removeRedundantPaths take semmod objects as arguments and do what their names imply.

The file input argument to the update method for semmod objects, which by default comes from standard input, is a set of update directives, one per line. There are five kinds of directives. In each case the directive begins with the directive name, followed by one or more fields separated by commas.

1. **delete:** Remove a path from the model. Example: delete, RSES -> FGenAsp
2. **add:** Add a path to the model. Example (the NA for the start value is optional): add, RSES -> FGenAsp, gam14, NA
3. **replace:** Replace every occurrence of the first string with the second in the variables and parameters of the model. This directive may be used, for example, to change one variable to another or to rename a parameter. Example: replace, gam, gamma, substitutes the string "gamma" for "gam" wherever the latter appears, presumably in parameter names.

4. **fix:** Fix a parameter that was formerly free. Example: `fix, RGenAsp -> REdAsp, 1`
5. **free:** Free a parameter that was formerly fixed. Example (the `NA` for the start value is optional): `free, RGenAsp -> ROccAsp, lam11, NA`

The `edit` method for `semmod` objects opens the model in the R editor.

Value

`specifyModel`, `specifyEquations`, `cfa`, `removeRedundantPaths`, `combineModels`, `update`, and `edit` return an object of class `semmod`, suitable as input for `sem`.

`multigroupModel` returns an object of class `semmodList`, also suitable as input for `sem`.

`classifyVariables` returns a list with two character vectors: `endogenous`, containing the names of endogenous variables in the model; and `exogenous`, containing the names of exogenous variables.

Author(s)

John Fox <jfox@mcmaster.ca> and Jarrett Byrnes

See Also

`sem`

Examples

```
# example using the text argument:

model.dhp <- specifyModel(text="
    RParAsp  -> RGenAsp, gam11,  NA
    RIQ      -> RGenAsp, gam12,  NA
    RSES     -> RGenAsp, gam13,  NA
    FSES     -> RGenAsp, gam14,  NA
    RSES     -> FGenAsp, gam23,  NA
    FSES     -> FGenAsp, gam24,  NA
    FIQ      -> FGenAsp, gam25,  NA
    FParAsp  -> FGenAsp, gam26,  NA
    FGenAsp  -> RGenAsp, beta12, NA
    RGenAsp  -> FGenAsp, beta21, NA
    RGenAsp  -> ROccAsp,  NA,     1
    RGenAsp  -> REdAsp,  lam21,  NA
    FGenAsp  -> FOccAsp,  NA,     1
    FGenAsp  -> FEdAsp,  lam42,  NA
    RGenAsp <-> RGenAsp, ps11,   NA
    FGenAsp <-> FGenAsp, ps22,   NA
    RGenAsp <-> FGenAsp, ps12,   NA
    ROccAsp <-> ROccAsp, theta1, NA
    REdAsp  <-> REdAsp,  theta2, NA
    FOccAsp <-> FOccAsp, theta3, NA
    FEdAsp  <-> FEdAsp,  theta4, NA
")
model.dhp
```

```
   # same model in equation form:
model.dhp.1 <- specifyEquations(covs="RGenAsp, FGenAsp", text="
RGenAsp = gam11*RParAsp + gam12*RIQ + gam13*RSES + gam14*FSES + beta12*FGenAsp
FGenAsp = gam23*RSES + gam24*FSES + gam25*FIQ + gam26*FParAsp + beta21*RGenAsp
ROccAsp = 1*RGenAsp
REdAsp = lam21(1)*RGenAsp  # to illustrate setting start values
FOccAsp = 1*FGenAsp
FEdAsp = lam42(1)*FGenAsp
")
model.dhp

# Note: The following examples can't be run via example() because the
#  default file argument requires that the model specification be entered
#  at the command prompt. The examples can be copied and run in an interactive
#  session in the R console, however.

    ## Not run:
model.dhp <- specifyModel()
    RParAsp  -> RGenAsp, gam11,  NA
    RIQ      -> RGenAsp, gam12,  NA
    RSES     -> RGenAsp, gam13,  NA
    FSES     -> RGenAsp, gam14,  NA
    RSES     -> FGenAsp, gam23,  NA
    FSES     -> FGenAsp, gam24,  NA
    FIQ      -> FGenAsp, gam25,  NA
    FParAsp  -> FGenAsp, gam26,  NA
    FGenAsp  -> RGenAsp, beta12, NA
    RGenAsp  -> FGenAsp, beta21, NA
    RGenAsp  -> ROccAsp,  NA,     1
    RGenAsp  -> REdAsp,  lam21,  NA
    FGenAsp  -> FOccAsp,  NA,     1
    FGenAsp  -> FEdAsp,  lam42,  NA
    RGenAsp <-> RGenAsp, ps11,   NA
    FGenAsp <-> FGenAsp, ps22,   NA
    RGenAsp <-> FGenAsp, ps12,   NA
    ROccAsp <-> ROccAsp, theta1, NA
    REdAsp  <-> REdAsp,  theta2, NA
    FOccAsp <-> FOccAsp, theta3, NA
    FEdAsp  <-> FEdAsp,  theta4, NA

model.dhp

# an equivalent specification, allowing specifyModel() to generate
#  variance parameters for endogenous variables (and suppressing
#  the unnecessary trailing NAs):

model.dhp <- specifyModel()
RParAsp  -> RGenAsp, gam11
RIQ      -> RGenAsp, gam12
RSES     -> RGenAsp, gam13
FSES     -> RGenAsp, gam14
RSES     -> FGenAsp, gam23
FSES     -> FGenAsp, gam24
```

```
FIQ      -> FGenAsp, gam25
FParAsp  -> FGenAsp, gam26
FGenAsp  -> RGenAsp, beta12
RGenAsp  -> FGenAsp, beta21
RGenAsp  -> ROccAsp,  NA,     1
RGenAsp  -> REdAsp,  lam21
FGenAsp  -> FOccAsp,  NA,     1
FGenAsp  -> FEdAsp,  lam42
RGenAsp <-> FGenAsp, ps12

model.dhp

# Another equivalent specification, telling specifyModel to add paths for
#   variances and covariance of RGenAsp and FGenAsp:

model.dhp <- specifyModel(covs="RGenAsp, FGenAsp")
RParAsp  -> RGenAsp, gam11
RIQ      -> RGenAsp, gam12
RSES     -> RGenAsp, gam13
FSES     -> RGenAsp, gam14
RSES     -> FGenAsp, gam23
FSES     -> FGenAsp, gam24
FIQ      -> FGenAsp, gam25
FParAsp  -> FGenAsp, gam26
FGenAsp  -> RGenAsp, beta12
RGenAsp  -> FGenAsp, beta21
RGenAsp  -> ROccAsp,  NA,     1
RGenAsp  -> REdAsp,  lam21
FGenAsp  -> FOccAsp,  NA,     1
FGenAsp  -> FEdAsp,  lam42

model.dhp

# The same model in equation format:

model.dhp.1 <- specifyEquations(covs="RGenAsp, FGenAsp")
RGenAsp = gam11*RParAsp + gam12*RIQ + gam13*RSES + gam14*FSES + beta12*FGenAsp
FGenAsp = gam23*RSES + gam24*FSES + gam25*FIQ + gam26*FParAsp + beta21*RGenAsp
ROccAsp = 1*RGenAsp
REdAsp = lam21(1)*RGenAsp  # to illustrate setting start values
FOccAsp = 1*FGenAsp
FEdAsp = lam42(1)*FGenAsp

model.dhp

classifyVariables(model.dhp)

# updating the model to impose equality constraints
#  and to rename the latent variables and gamma parameters

model.dhp.eq <- update(model.dhp)
delete, RSES -> FGenAsp
delete, FSES -> FGenAsp
```

```
delete, FIQ  -> FGenAsp
delete, FParAsp -> FGenAs
delete, RGenAsp  -> FGenAsp
add, RSES     -> FGenAsp, gam14,  NA
add, FSES     -> FGenAsp, gam13,  NA
add, FIQ      -> FGenAsp, gam12,  NA
add, FParAsp  -> FGenAsp, gam26,  NA
add, RGenAsp  -> FGenAsp, beta12, NA
replace, gam, gamma
replace, Gen, General

model.dhp.eq

# A three-factor CFA model for the Thurstone mental-tests data,
#    specified three equivalent ways:

R.thur <- readMoments(diag=FALSE,
    names=c('Sentences','Vocabulary',
            'Sent.Completion','First.Letters','4.Letter.Words','Suffixes',
            'Letter.Series','Pedigrees', 'Letter.Group'))
.828
.776   .779
.439   .493    .46
.432   .464    .425   .674
.447   .489    .443   .59    .541
.447   .432    .401   .381    .402   .288
.541   .537    .534   .35    .367   .32   .555
.38   .358    .359   .424    .446   .325   .598   .452

 #  (1a) in CFA format:

mod.cfa.thur.c <- cfa(reference.indicators=FALSE)
FA: Sentences, Vocabulary, Sent.Completion
FB: First.Letters, 4.Letter.Words, Suffixes
FC: Letter.Series, Pedigrees, Letter.Group

cfa.thur.c <- sem(mod.cfa.thur.c, R.thur, 213)
summary(cfa.thur.c)

 #  (1b) in CFA format, using reference indicators:

mod.cfa.thur.r <- cfa()
FA: Sentences, Vocabulary, Sent.Completion
FB: First.Letters, 4.Letter.Words, Suffixes
FC: Letter.Series, Pedigrees, Letter.Group

cfa.thur.r <- sem(mod.cfa.thur.r, R.thur, 213)
summary(cfa.thur.r)

 #  (2) in equation format:

mod.cfa.thur.e <- specifyEquations(covs="F1, F2, F3")
Sentences = lam11*F1
```

```
Vocabulary = lam21*F1
Sent.Completion = lam31*F1
First.Letters = lam42*F2
4.Letter.Words = lam52*F2
Suffixes = lam62*F2
Letter.Series = lam73*F3
Pedigrees = lam83*F3
Letter.Group = lam93*F3
V(F1) = 1
V(F2) = 1
V(F3) = 1

cfa.thur.e <- sem(mod.cfa.thur.e, R.thur, 213)
summary(cfa.thur.e)

 #  (3) in path format:

mod.cfa.thur.p <- specifyModel(covs="F1, F2, F3")
F1 -> Sentences,                    lam11
F1 -> Vocabulary,                   lam21
F1 -> Sent.Completion,              lam31
F2 -> First.Letters,                lam41
F2 -> 4.Letter.Words,               lam52
F2 -> Suffixes,                     lam62
F3 -> Letter.Series,                lam73
F3 -> Pedigrees,                    lam83
F3 -> Letter.Group,                 lam93
F1 <-> F1,                          NA,     1
F2 <-> F2,                          NA,     1
F3 <-> F3,                          NA,     1

cfa.thur.p <- sem(mod.cfa.thur.p, R.thur, 213)
summary(cfa.thur.p)

# The Thursstone CFA model with equality constraints on the
#  factor loadings and error variances

mod.cfa.thur.ceq <- cfa(reference.indicators=FALSE)
FA: Sentences = Vocabulary = Sent.Completion
FB: First.Letters = 4.Letter.Words = Suffixes
FC: Letter.Series = Pedigrees = Letter.Group
var: Sentences = Vocabulary = Sent.Completion
var: First.Letters = 4.Letter.Words = Suffixes
var: Letter.Series = Pedigrees = Letter.Group

cfa.thur.ceq <- sem(mod.cfa.thur.ceq, R.thur, 213)
summary(cfa.thur.ceq)
anova(cfa.thur.c, cfa.thur.ceq)
pathDiagram(cfa.thur.ceq, ignore.double=FALSE, ignore.self=TRUE,
    min.rank="FA, FB, FC", edge.labels="values")

# a multigroup CFA model fit to the Holzinger-Swineford
#   mental-tests data
```

```
mod.hs <- cfa()
spatial: visual, cubes, paper, flags
verbal: general, paragrap, sentence, wordc, wordm
memory: wordr, numberr, figurer, object, numberf, figurew
math: deduct, numeric, problemr, series, arithmet

mod.mg <- multigroupModel(mod.hs, groups=c("Female", "Male"))

sem.mg <- sem(mod.mg, data=HS.data, group="Gender",
              formula = ~ visual + cubes + paper + flags +
              general + paragrap + sentence + wordc + wordm +
              wordr + numberr + figurer + object + numberf + figurew +
              deduct + numeric + problemr + series + arithmet
              )
summary(sem.mg)

 # with cross-group equality constraints:

mod.mg.eq <- multigroupModel(mod.hs, groups=c("Female", "Male"), allEqual=TRUE)

sem.mg.eq <- sem(mod.mg.eq, data=HS.data, group="Gender",
              formula = ~ visual + cubes + paper + flags +
                general + paragrap + sentence + wordc + wordm +
                wordr + numberr + figurer + object + numberf + figurew +
                deduct + numeric + problemr + series + arithmet
              )
summary(sem.mg.eq)

## End(Not run)
```

`standardizedCoefficients` *Standardized Coefficients for Structural Equation Models*

Description

These functions calculate standardized regression coefficients for structural equation models. The function `stdCoef` is simply an abbreviation for `standardizedCoefficients`.

Usage

```
standardizedCoefficients(object, ...)
## S3 method for class 'sem'
standardizedCoefficients(object,
    digits = getOption("digits"), oneheaded = TRUE, twoheaded = TRUE, ...)
## S3 method for class 'msem'
standardizedCoefficients(object, ...)

stdCoef(...)
```

Arguments

object	an object of class sem or msem returned by the sem function.
digits	number of digits for printed output.
oneheaded	standardize path coefficients? Default is TRUE.
twoheaded	standardize variances and covariances? Default is TRUE.
...	arguments to pass down.

Value

Returns a data frame with the coefficients, labelled both by parameter names and by arrows in the path diagram for the model. The msem (multigroup) method computes and prints the standardized coefficients for each group; it does not return a useful result.

Author(s)

John Fox <jfox@mcmaster.ca> and Adam Kramer

References

Bollen, K. A. (1989) *Structural Equations With Latent Variables.* Wiley.

See Also

sem

Examples

```
# In the first example, readMoments() and specifyModel() read from the
# input stream. This example cannot be executed via example() but can be entered
# at the command prompt. The example is repeated using file input;
# this example can be executed via example().
    ## Not run:
# Duncan, Haller, and Portes peer-influences model

R.DHP <- readMoments(diag=FALSE, names=c("ROccAsp", "REdAsp", "FOccAsp",
                "FEdAsp", "RParAsp", "RIQ", "RSES", "FSES", "FIQ", "FParAsp"))
    .6247
    .3269  .3669
    .4216  .3275  .6404
    .2137  .2742  .1124  .0839
    .4105  .4043  .2903  .2598  .1839
    .3240  .4047  .3054  .2786  .0489  .2220
    .2930  .2407  .4105  .3607  .0186  .1861  .2707
    .2995  .2863  .5191  .5007  .0782  .3355  .2302  .2950
    .0760  .0702  .2784  .1988  .1147  .1021  .0931 -.0438  .2087

model.dhp <- specifyModel()
    RParAsp  -> RGenAsp, gam11,  NA
    RIQ      -> RGenAsp, gam12,  NA
    RSES     -> RGenAsp, gam13,  NA
```

```
    FSES     -> RGenAsp, gam14,  NA
    RSES     -> FGenAsp, gam23,  NA
    FSES     -> FGenAsp, gam24,  NA
    FIQ      -> FGenAsp, gam25,  NA
    FParAsp  -> FGenAsp, gam26,  NA
    FGenAsp  -> RGenAsp, beta12, NA
    RGenAsp  -> FGenAsp, beta21, NA
    RGenAsp  -> ROccAsp,  NA,     1
    RGenAsp  -> REdAsp,  lam21,  NA
    FGenAsp  -> FOccAsp,  NA,     1
    FGenAsp  -> FEdAsp,  lam42,  NA
    RGenAsp <-> RGenAsp, ps11,   NA
    FGenAsp <-> FGenAsp, ps22,   NA
    RGenAsp <-> FGenAsp, ps12,   NA
    ROccAsp <-> ROccAsp, theta1, NA
    REdAsp  <-> REdAsp,  theta2, NA
    FOccAsp <-> FOccAsp, theta3, NA
    FEdAsp  <-> FEdAsp,  theta4, NA

sem.dhp <- sem(model.dhp, R.DHP, 329,
    fixed.x=c('RParAsp', 'RIQ', 'RSES', 'FSES', 'FIQ', 'FParAsp'))
standardizedCoefficients(sem.dhp)

## End(Not run)
# The following example can be executed via example():

etc <- system.file(package="sem", "etc") # path to data and model files

(R.DHP <- readMoments(file=file.path(etc, "R-DHP.txt"),
    diag=FALSE, names=c("ROccAsp", "REdAsp", "FOccAsp",
                "FEdAsp", "RParAsp", "RIQ", "RSES", "FSES", "FIQ", "FParAsp")))
(model.dhp <- specifyModel(file=file.path(etc, "model-DHP.txt")))
(sem.dhp <- sem(model.dhp, R.DHP, 329,
    fixed.x=c('RParAsp', 'RIQ', 'RSES', 'FSES', 'FIQ', 'FParAsp')))
standardizedCoefficients(sem.dhp)
```

`Tests` *Six Mental Tests*

Description

These data are from the SAS manual and consist of six mental tests for 32 students, with some missing data. The three `x` variables are intended to load on a verbal factor, and the three `y` variables on a math factor. The data can be used to illustrate the estimation of a confirmatory factor analysis model by multinormal full-information maximum-likelihood in the presence of missing data.

Usage

```
Tests
```

Format

A data frame with 32 observations on the following 6 variables.

`x1` score on verbal test 1.

`x2` score on verbal test 2.

`x3` score on verbal test 3.

`y1` score on math test 1.

`y2` score on math test 2.

`y3` score on math test 3.

Source

Example 25.13 from *SAS/STAT 9.22 User's Guide*, SAS Institute, 2010.

`tsls` *Two-Stage Least Squares*

Description

Fits a regression equation, such as an equation in a structural-equation model, by two-stage least squares. This is equivalent to direct instrumental-variables estimation when the number of instruments is equal to the number of predictors.

Usage

```
## S3 method for class 'formula'
tsls(formula, instruments, data, subset, weights,
 na.action, contrasts=NULL, ...)
## Default S3 method:
tsls(y, X, Z, w, names=NULL, ...)

## S3 method for class 'tsls'
print(x, ...)
## S3 method for class 'tsls'
summary(object, digits=getOption("digits"), ...)
## S3 method for class 'summary.tsls'
print(x, ...)
## S3 method for class 'tsls'
anova(object, model.2, s2, dfe, ...)

## S3 method for class 'tsls'
fitted(object, ...)
## S3 method for class 'tsls'
residuals(object, ...)
## S3 method for class 'tsls'
coef(object, ...)
## S3 method for class 'tsls'
vcov(object, ...)
```

Arguments

formula	model formula for structural equation to be estimated; a regression constant is implied if not explicitly omitted.
instruments	one-sided model formula specifying instrumental variables.
data	an optional data frame containing the variables in the model. By default the variables are taken from the environment from which tsls is called.
subset	an optional vector specifying a subset of observations to be used in fitting the model.
weights, w	an optional vector of weights to be used in the fitting process; if specified should be a non-negative numeric vector with one entry for each observation, to be used to compute weighted 2SLS estimates.
na.action	a function that indicates what should happen when the data contain NAs. The default is set by the na.action option.
contrasts	an optional list. See the contrasts.arg argument of model.matrix.default.
y	Response-variable vector.
X	Matrix of predictors, including a constant (if one is in the model).
Z	Matrix of instrumental variables, including a constant (if one is in the model).
names	optional character vector of names for the columns of the X matrix.
x, object, model.2	objects of class tsls returned by tsls.formula (or of class summary.tsls), for anova containing nested models to be compared by an incremental F-test. One model should be nested in the other; the order of models is immaterial.
s2	an optional estimate of error variance for the denominator of the F-test. If missing, the error-variance estimate is taken from the larger model.
dfe	optional error degrees of freedom, to be specified when an estimate of error variance is given.
digits	number of digits for summary output.
...	arguments to be passed down.

Value

tsls.formula returns an object of class tsls, with the following components:

n	number of observations.
p	number of parameters.
coefficients	parameter estimates.
V	estimated covariance matrix of coefficients.
s	residual standard error.
residuals	vector of residuals.
response	vector of response values.
X	model matrix.

Z	instrumental-variables matrix.
response.name	name of response variable, or expression evaluating to response.
formula	model formula.
instruments	one-sided formula for instrumental variables.

Author(s)

John Fox <jfox@mcmaster.ca>

References

Fox, J. (1979) Simultaneous equation models and two-stage least-squares. In Schuessler, K. F. (ed.) *Sociological Methodology 1979*, Jossey-Bass.

Greene, W. H. (1993) *Econometric Analysis, Second Edition*, Macmillan.

See Also

sem

Examples

```
summary(tsls(Q ~ P + D, ~ D + F + A, data=Kmenta))     # demand equation

summary(tsls(Q ~ P + F + A, ~ D + F + A, data=Kmenta)) # supply equation

anova(tsls(Q ~ P + F + A, ~ D + F + A, data=Kmenta),
    tsls(Q ~ 1, ~ D + F + A, data=Kmenta))
```

Index

∗ **datasets**
  Bollen, 2
  CNES, 7
  HS.data, 12
  Klein, 15
  Kmenta, 16
  Tests, 74
∗ **dplot**
  pathDiagram, 27
∗ **htest**
  bootSem, 3
∗ **manip**
  fscores, 10
  rawMoments, 34
  readMoments, 36
∗ **models**
  bootSem, 3
  effects.sem, 8
  fscores, 10
  information.criteria, 14
  miSem, 17
  ML.methods, 19
  modIndices, 22
  objective.functions, 24
  optimizers, 26
  pathDiagram, 27
  ram, 33
  residuals.sem, 37
  sem, 40
  specifyModel, 63
  standardizedCoefficients, 72
  tsls, 75

AIC.msemObjectiveML *(*ML.methods*)*, 19
AIC.objectiveFIML *(*ML.methods*)*, 19
AIC.objectiveML *(*ML.methods*)*, 19
AICc *(*information.criteria*)*, 14
AICc.msemObjectiveML *(*ML.methods*)*, 19
AICc.objectiveFIML *(*ML.methods*)*, 19
AICc.objectiveML, *14*
AICc.objectiveML *(*ML.methods*)*, 19
anova.msemObjectiveML *(*ML.methods*)*, 19
anova.objectiveFIML *(*ML.methods*)*, 19
anova.objectiveML *(*ML.methods*)*, 19
anova.tsls *(*tsls*)*, 75

BIC.msemObjectiveML *(*ML.methods*)*, 19
BIC.objectiveFIML *(*ML.methods*)*, 19
BIC.objectiveML *(*ML.methods*)*, 19
Bollen, 2
boot, *5*
boot.ci, *4*
boot.sem *(*sem-deprecated*)*, 62
bootSem, 3, *62*

CAIC *(*information.criteria*)*, 14
CAIC.objectiveFIML *(*ML.methods*)*, 19
CAIC.objectiveML, *14*
CAIC.objectiveML *(*ML.methods*)*, 19
cfa, *17*, *31*, *40*, *41*, *44*, *45*, *48*
cfa *(*specifyModel*)*, 63
classifyVariables *(*specifyModel*)*, 63
CNES, 7
coef.msem *(*sem*)*, 40
coef.sem *(*sem*)*, 40
coef.tsls *(*tsls*)*, 75
combineModels *(*specifyModel*)*, 63
complete, *18*
cor, *4*
cov, *4*
cov2raw *(*rawMoments*)*, 34

deviance.msemObjectiveML *(*ML.methods*)*, 19
deviance.objectiveFIML *(*ML.methods*)*, 19
deviance.objectiveML *(*ML.methods*)*, 19
df.residual.msem *(*sem*)*, 40
df.residual.sem *(*sem*)*, 40

edit.semmod *(*specifyModel*)*, 63

effects.msem (effects.sem), 8
effects.sem, 8

fitted.tsls (tsls), 75
fscores, 10

GLS.methods (ML.methods), 19

hetcor, *4*
HS.data, 12

information.criteria, 14

Klein, 15
Kmenta, 16

logLik.msemObjectiveML (ML.methods), 19
logLik.objectiveFIML (ML.methods), 19
logLik.objectiveML (ML.methods), 19

math (pathDiagram), 27
mi, *17, 18*
miSem, 17
ML.methods, 19
mod.indices (sem-deprecated), 62
model.matrix.default, *35*, *76*
modIndices, 22, *62*
modIndices.objectiveML, *22*
msemObjectiveGLS, *43*
msemObjectiveGLS (objective.functions), 24
msemObjectiveML, *19*, *43*
msemObjectiveML (objective.functions), 24
msemObjectiveML2 (objective.functions), 24
msemOptimizerNlm, *43*
msemOptimizerNlm (optimizers), 26
multigroupModel, *17*, *40, 41*
multigroupModel (specifyModel), 63

nlm, *26, 27*, *43*, *48*
nlminb, *26, 27*, *43*, *48*
normalized.residuals (sem-deprecated), 62
normalizedResiduals, *62*
normalizedResiduals (residuals.sem), 37

objective.functions, *22*, 24, *26, 27*
objectiveFIML, *18*
objectiveFIML (objective.functions), 24
objectiveFIML2 (objective.functions), 24
objectiveGLS, *19*, *26*, *43*, *48*
objectiveGLS (objective.functions), 24
objectiveGLS2 (objective.functions), 24
objectiveML, *19*, *26*, *43, 44*, *48*
objectiveML (objective.functions), 24
objectiveML2 (objective.functions), 24
optim, *26, 27*, *43*, *48*
optimizerMsem, *43*
optimizerMsem (optimizers), 26
optimizerNlm, *48*
optimizerNlm (optimizers), 26
optimizerNlminb, *43*, *48*
optimizerNlminb (optimizers), 26
optimizerOptim, *43*, *48*
optimizerOptim (optimizers), 26
optimizers, *24*, 26, *26*
optimizerSem, *43*
optimizerSem (optimizers), 26

path.diagram (sem-deprecated), 62
pathDiagram, 27, *62*
print.bootsem (bootSem), 3
print.miSem (miSem), 17
print.modIndices (modIndices), 22
print.msemModIndices (modIndices), 22
print.msemObjectiveGLS (ML.methods), 19
print.msemObjectiveML (ML.methods), 19
print.objectiveFIML (ML.methods), 19
print.objectiveGLS (ML.methods), 19
print.objectiveML (ML.methods), 19
print.rawmoments (rawMoments), 34
print.semeffects (effects.sem), 8
print.semeffectsList (effects.sem), 8
print.semmod (specifyModel), 63
print.semmodList (specifyModel), 63
print.summary.bootsem (bootSem), 3
print.summary.objectiveML (ML.methods), 19
print.summary.tsls (tsls), 75
print.tsls (tsls), 75

ram, 33
raw.moments (sem-deprecated), 62
rawMoments, 34, *41*, *48*, *62*
read.moments (sem-deprecated), 62
readMoments, 36, *63*
removeRedundantPaths (specifyModel), 63

residuals.msem *(residuals.sem)*, 37
residuals.sem, 37
residuals.tsls *(tsls)*, 75

scale, *11*
sem, *3–5*, *8*, *11*, *17–19*, *21–27*, *31*, *34–36*, *38*, 40, *66, 67*, *73*, *77*
sem-deprecated, 62
specify.model *(sem-deprecated)*, 62
specifyEquations, *17*, *31*, *40, 41*, *44, 45*, *48*
specifyEquations *(specifyModel)*, 63
specifyModel, *17*, *31*, *40, 41*, *44, 45*, *47, 48*, 63, *63*
standardized.coefficients *(sem-deprecated)*, 62
standardized.residuals *(sem-deprecated)*, 62
standardizedCoefficients, *63*, 72
standardizedResiduals, *63*
standardizedResiduals *(residuals.sem)*, 37
startvalues, *48*
startvalues *(sem)*, 40
startvalues2 *(sem)*, 40
std.coef *(sem-deprecated)*, 62
stdCoef, *63*
stdCoef *(standardizedCoefficients)*, 72
summary.bootsem *(bootSem)*, 3
summary.miSem *(miSem)*, 17
summary.modIndices *(modIndices)*, 22
summary.msemModIndices *(modIndices)*, 22
summary.msemObjectiveGLS *(ML.methods)*, 19
summary.msemObjectiveML *(ML.methods)*, 19
summary.objectiveFIML *(ML.methods)*, 19
summary.objectiveGLS *(ML.methods)*, 19
summary.objectiveML *(ML.methods)*, 19
summary.tsls *(tsls)*, 75

Tests, 74
tsls, 75

update.semmod *(specifyModel)*, 63

vcov.msem *(sem)*, 40
vcov.sem *(sem)*, 40
vcov.tsls *(tsls)*, 75