Package: blavaan (via r-universe)

October 22, 2026

Title Bayesian Latent Variable Analysis

Version 0.6-1

Description Fit a variety of Bayesian latent variable models, including confirmatory factor analysis, structural equation models, and latent growth curve models. References: Merkle & Rosseel (2018) <doi:10.18637/jss.v085.i04>; Merkle et al. (2021) <doi:10.18637/jss.v100.i06>.

License GPL (>= 3)

Encoding UTF-8

ByteCompile true

Depends R(>= 4.1.0), methods, Rcpp(>= 0.12.15)

Imports stats, utils, graphics, lavaan(>= 0.7-1), coda, mnormt, nonnest2(>= 0.5-7), loo(>= 2.0), rstan(>= 2.26.0), rstantools(>= 1.5.0), RcppParallel (>= 5.0.1), bayesplot, Matrix, future, future.apply, tmvnsim, igraph, generics

LinkingTo StanHeaders (>= 2.26.0), rstan (>= 2.26.0), BH (>= 1.69.0), Rcpp (>= 0.12.15), RcppEigen (>= 0.3.3.4.0), RcppParallel (>= 5.0.1)

Suggests runjags(>= 2.0.4-3), modeest(>= 2.3.3), rjags, cmdstanr, posterior, semTools, blavsam, tinytest

SystemRequirements GNU make

URL https://blavaan.org, https://github.com/ecmerkle/blavaan

BugReports https://github.com/ecmerkle/blavaan/issues

Additional_repositories https://stan-dev.r-universe.dev/, https://ecmerkle.github.io/drat

Config/Needs/website brms

RoxygenNote 7.3.3

NeedsCompilation yes

Author Edgar Merkle [aut, cre] (ORCID:
<https://orcid.org/0000-0001-7158-0653>), Yves Rosseel [aut],
Ben Goodrich [aut], Mauricio Garnier-Villarreal [ctb] (ORCID:
<https://orcid.org/0000-0002-2951-6647>, R/blav_compare.R,
R/ctr_bayes_fit.R, vignettes), Terrence D. Jorgensen [ctb]
(ORCID: <https://orcid.org/0000-0001-5111-6773>,
R/ctr_bayes_fit.R, R/ctr_ppmc.R, R/blav_predict.R), Huub Hoofs
[ctb] (R/ctr_bayes_fit.R), Rens van de Schoot [ctb]
(R/ctr_bayes_fit.R), Andrew Johnson [ctb] (Makevars), Matthew
Emery [ctb] (loo moment_match), Michael S. Truong [ctb]
(R/blav_tidiers.R, options(blavaan.target))

Maintainer Edgar Merkle <merklee@missouri.edu>

Config/pak/sysreqs make

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

Date/Publication 2026-08-22 23:20:30 UTC

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

RemoteRef HEAD

RemoteSha b6fc73c2d3c7c4509e66dce9c3121f0d4287ecde

Contents

bcfa . . . 3
bgrowth . . . 6
blav_internal . . . 9
blavaan . . . 9
blavaan-class . . . 12
blavCompare . . . 14
blavFitIndices . . . 15
blavInspect . . . 18
blavPredict . . . 20
bsem . . . 22
dpriors . . . 25
glance.blavaan . . . 26
plot.blavaan . . . 27
ppmc . . . 28
ppp . . . 32
sampleData . . . 34
standardizedPosterior . . . 35
tidy.blavaan . . . 37

Index **40**

bcfa *Fit Confirmatory Factor Analysis Models*

Description

Fit a Confirmatory Factor Analysis (CFA) model.

Usage

```
bcfa(..., cp = "srs",
     dp = NULL, n.chains = 3, burnin, sample,
     adapt, mcmcfile = FALSE, mcmcextra = list(), inits = "simple",
     convergence = "manual", target = getOption("blavaan.target", "stan"),
     save.lvs = FALSE, wiggle = NULL, wiggle.sd = 0.1, prisamp = FALSE,
     jags.ic = FALSE, seed = NULL, bcontrol = list())
```

Arguments

`...`	Default lavaan arguments. See lavaan. This also accepts sampler-specific arguments that would otherwise need to be supplied via `bcontrol` – see `bcontrol` below.
`cp`	Handling of prior distributions on covariance parameters: possible values are `"srs"` (default) or `"fa"`. Option `"fa"` is only available for `target="jags"`.
`dp`	Default prior distributions on different types of parameters, typically the result of a call to `dpriors()`. See the `dpriors()` help file for more information.
`n.chains`	Number of desired MCMC chains.
`burnin`	Number of burnin/warmup iterations (not including the adaptive iterations, for target="jags"). Defaults to 4000 or target="jags" and 500 for Stan targets.
`sample`	The total number of samples to take after burnin. Defaults to 10000 for target="jags" and 1000 for Stan targets.
`adapt`	For target="jags", the number of adaptive iterations to use at the start of sampling. Defaults to 1000.
`mcmcfile`	If `TRUE`, the JAGS/Stan model will be written to file (in the lavExport directory). Can also supply a character string, which serves as the name of the directory to which files will be written.
`mcmcextra`	A list with potential names `syntax` (unavailable for `target="stan"`), `monitor`, `data`, and `llnsamp`. The `syntax` object is a text string containing extra code to insert in the JAGS/Stan model syntax. The `data` object is a list of extra data to send to the JAGS/Stan model. If `moment_match_k_threshold` is specified within `data` the looic of the model will be calculated using moment matching. The `monitor` object is a character vector containing extra JAGS/Stan parameters to monitor. The `llnsamp` object is only relevant to models with ordinal variables, and specifies the number of samples that should be drawn to approximate the model log-likelihood (larger numbers imply higher accuracy and longer time). This log-likelihood is specifically used to compute information criteria.

inits — If it is a character string, the options are currently "simple" (default), "Mplus", "prior", or "jags". In the first two cases, parameter values are set as though they will be estimated via ML (see lavaan). The starting parameter value for each chain is then perturbed from the original values through the addition of random uniform noise. If "prior" is used, the starting parameter values are obtained based on the prior distributions (while also trying to ensure that the starting values will not crash the model estimation). If "jags", no starting values are specified and JAGS will choose values on its own (and this will probably crash Stan targets). You can also supply a list of starting values for each chain, where the list format can be obtained from, e.g., blavInspect(fit, "inits"). Finally, you can specify starting values in a similar way to lavaan, using the lavaan start argument (see the lavaan documentation for all the options there). In this case, you should also set inits="simple", and be aware that the same starting values will be used for each chain.

convergence — Useful only for target="jags". If "auto", parameters are sampled until convergence is achieved (via autorun.jags()). In this case, the arguments burnin and sample are passed to autorun.jags() as startburnin and startsample, respectively. Otherwise, parameters are sampled as specified by the user (or by the run.jags defaults).

target — Desired MCMC sampling, with "stan" (pre-compiled marginal approach) as default. Also available is "vb", which calls the rstan function vb(). Other options include "jags", "stancond", and "stanclassic", which sample latent variables and provide some greater functionality (because syntax is written "on the fly"). But they are slower and less efficient.

save.lvs — Should sampled latent variables (factor scores) be saved? Logical; defaults to FALSE

wiggle — Labels of equality-constrained parameters that should be "approximately" equal. Can also be "intercepts", "loadings", "regressions", "means".

wiggle.sd — The prior sd (of normal distribution) to be used in approximate equality constraints. Can be one value, or (for target="stan") a numeric vector of values that is the same length as wiggle.

prisamp — Should samples be drawn from the prior, instead of the posterior (target="stan" only)? Logical; defaults to FALSE

jags.ic — Should DIC be computed the JAGS way, in addition to the BUGS way? Logical; defaults to FALSE

seed — A vector of length n.chains (for target "jags") or an integer (for target "stan") containing random seeds for the MCMC run. If NULL, seeds will be chosen randomly.

bcontrol — A list containing additional parameters passed to run.jags (or autorun.jags), stan, vb, or (for target="cmdstan") a cmdstanr model's $sample() method. See the manpage of those functions for an overview of the additional parameters that can be set. Recognized arguments (e.g. adapt_delta, max_treedepth, cores, thin, method, threads_per_chain, and other target-specific sampler arguments) can instead be supplied directly to blavaan()/bcfa()/bsem()/bgrowth() without wrapping them in bcontrol; for target %in% c("stan","stanclassic", "stancond"), arguments belonging to Stan's nested control= list (like adapt_delta)

are detected automatically and nested for you. Supplying the same argument both directly and inside `bcontrol` is an error. A few names remain reserved and must still be supplied via `bcontrol` because they collide with another `blavaan()`/`lavaan()` argument: `verbose`, `control` (lavaan's own optimizer control), and `cl` (lavaan's bootstrap cluster option).

Details

The `bcfa` function is a wrapper for the more general `blavaan` function, using the following default `lavaan` arguments: `int.ov.free` = `TRUE`, `int.lv.free` = `FALSE`, `auto.fix.first` = `TRUE` (unless `std.lv` = `TRUE`), `auto.fix.single` = `TRUE`, `auto.var` = `TRUE`, `auto.cov.lv.x` = `TRUE`, `auto.th` = `TRUE`, `auto.delta` = `TRUE`, and `auto.cov.y` = `TRUE`.

Value

An object that inherits from class `lavaan`, for which several methods are available, including a `summary` method.

References

Edgar C. Merkle, Ellen Fitzsimmons, James Uanhoro, & Ben Goodrich (2021). Efficient Bayesian Structural Equation Modeling in Stan. Journal of Statistical Software, 100(6), 1-22. URL http://www.jstatsoft.org/v100/i06/.

Edgar C. Merkle & Yves Rosseel (2018). blavaan: Bayesian Structural Equation Models via Parameter Expansion. Journal of Statistical Software, 85(4), 1-30. URL http://www.jstatsoft.org/v85/i04/.

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

See Also

`blavaan`

Examples

```
data(HolzingerSwineford1939, package = "lavaan")

# The Holzinger and Swineford (1939) example
HS.model <- ' visual  =~ x1 + x2 + x3
              textual =~ x4 + x5 + x6
              speed   =~ x7 + x8 + x9 '

## Not run:
fit <- bcfa(HS.model, data = HolzingerSwineford1939)
summary(fit)

## End(Not run)

# A short run for rough results
fit <- bcfa(HS.model, data = HolzingerSwineford1939, burnin = 100, sample = 100,
            n.chains = 2)
summary(fit)
```

bgrowth *Fit Growth Curve Models*

Description

Fit a Growth Curve model.

Usage

```
bgrowth(..., cp = "srs", dp = NULL, n.chains = 3,
burnin, sample, adapt, mcmcfile = FALSE, mcmcextra = list(),
inits = "simple", convergence = "manual", target = getOption("blavaan.target", "stan"),
save.lvs = FALSE, wiggle = NULL, wiggle.sd = 0.1, prisamp = FALSE,
jags.ic = FALSE, seed = NULL, bcontrol = list())
```

Arguments

`...`	Default lavaan arguments. See lavaan. This also accepts sampler-specific arguments that would otherwise need to be supplied via `bcontrol` – see `bcontrol` below.
`cp`	Handling of prior distributions on covariance parameters: possible values are `"srs"` (default) or `"fa"`. Option `"fa"` is only available for `target="jags"`.
`dp`	Default prior distributions on different types of parameters, typically the result of a call to `dpriors()`. See the `dpriors()` help file for more information.
`n.chains`	Number of desired MCMC chains.
`burnin`	Number of burnin/warmup iterations (not including the adaptive iterations, for target="jags"). Defaults to 4000 or target="jags" and 500 for Stan targets.
`sample`	The total number of samples to take after burnin. Defaults to 10000 for target="jags" and 1000 for Stan targets.
`adapt`	For target="jags", the number of adaptive iterations to use at the start of sampling. Defaults to 1000.
`mcmcfile`	If `TRUE`, the JAGS/Stan model will be written to file (in the lavExport directory). Can also supply a character string, which serves as the name of the directory to which files will be written.
`mcmcextra`	A list with potential names `syntax` (unavailable for `target="stan"`), `monitor`, `data`, and `llnsamp`. The `syntax` object is a text string containing extra code to insert in the JAGS/Stan model syntax. The `data` object is a list of extra data to send to the JAGS/Stan model. If `moment_match_k_threshold` is specified within `data` the looic of the model will be calculated using moment matching. The `monitor` object is a character vector containing extra JAGS/Stan parameters to monitor. The `llnsamp` object is only relevant to models with ordinal variables, and specifies the number of samples that should be drawn to approximate the model log-likelihood (larger numbers imply higher accuracy and longer time). This log-likelihood is specifically used to compute information criteria.

`inits` If it is a character string, the options are currently `"simple"` (default), `"Mplus"`, `"prior"`, or `"jags"`. In the first two cases, parameter values are set as though they will be estimated via ML (see lavaan). The starting parameter value for each chain is then perturbed from the original values through the addition of random uniform noise. If `"prior"` is used, the starting parameter values are obtained based on the prior distributions (while also trying to ensure that the starting values will not crash the model estimation). If `"jags"`, no starting values are specified and JAGS will choose values on its own (and this will probably crash Stan targets). You can also supply a list of starting values for each chain, where the list format can be obtained from, e.g., `blavInspect(fit, "inits")`. Finally, you can specify starting values in a similar way to lavaan, using the lavaan `start` argument (see the lavaan documentation for all the options there). In this case, you should also set `inits="simple"`, and be aware that the same starting values will be used for each chain.

`convergence` Useful only for `target="jags"`. If `"auto"`, parameters are sampled until convergence is achieved (via `autorun.jags()`). In this case, the arguments `burnin` and `sample` are passed to `autorun.jags()` as `startburnin` and `startsample`, respectively. Otherwise, parameters are sampled as specified by the user (or by the `run.jags` defaults).

`target` Desired MCMC sampling, with `"stan"` (pre-compiled marginal approach) as default. Also available is `"vb"`, which calls the rstan function `vb()`. Other options include `"jags"`, `"stancond"`, and `"stanclassic"`, which sample latent variables and provide some greater functionality (because syntax is written "on the fly"). But they are slower and less efficient.

`save.lvs` Should sampled latent variables (factor scores) be saved? Logical; defaults to FALSE

`wiggle` Labels of equality-constrained parameters that should be "approximately" equal. Can also be "intercepts", "loadings", "regressions", "means".

`wiggle.sd` The prior sd (of normal distribution) to be used in approximate equality constraints. Can be one value, or (for target="stan") a numeric vector of values that is the same length as wiggle.

`prisamp` Should samples be drawn from the prior, instead of the posterior (`target="stan"` only)? Logical; defaults to FALSE

`jags.ic` Should DIC be computed the JAGS way, in addition to the BUGS way? Logical; defaults to FALSE

`seed` A vector of length `n.chains` (for target `"jags"`) or an integer (for target `"stan"`) containing random seeds for the MCMC run. If `NULL`, seeds will be chosen randomly.

`bcontrol` A list containing additional parameters passed to `run.jags` (or `autorun.jags`), `stan`, `vb`, or (for `target="cmdstan"`) a `cmdstanr` model's `$sample()` method. See the manpage of those functions for an overview of the additional parameters that can be set. Recognized arguments (e.g. `adapt_delta`, `max_treedepth`, `cores`, `thin`, `method`, `threads_per_chain`, and other target-specific sampler arguments) can instead be supplied directly to `blavaan()`/`bcfa()`/`bsem()`/`bgrowth()` without wrapping them in `bcontrol`; for `target %in% c("stan","stanclassic", "stancond")`, arguments belonging to Stan's nested `control=` list (like `adapt_delta`)

are detected automatically and nested for you. Supplying the same argument both directly and inside `bcontrol` is an error. A few names remain reserved and must still be supplied via `bcontrol` because they collide with another `blavaan()`/`lavaan()` argument: `verbose`, `control` (lavaan's own optimizer control), and `cl` (lavaan's bootstrap cluster option).

Details

The `bgrowth` function is a wrapper for the more general `blavaan` function, using the following default `lavaan` arguments: `meanstructure` = `TRUE`, `int.ov.free` = `FALSE`, `int.lv.free` = `TRUE`, `auto.fix.first` = `TRUE` (unless `std.lv` = `TRUE`), `auto.fix.single` = `TRUE`, `auto.var` = `TRUE`, `auto.cov.lv.x` = `TRUE`, `auto.th` = `TRUE`, `auto.delta` = `TRUE`, and `auto.cov.y` = `TRUE`.

Value

An object of class `blavaan`, for which several methods are available, including a `summary` method.

References

Edgar C. Merkle, Ellen Fitzsimmons, James Uanhoro, & Ben Goodrich (2021). Efficient Bayesian Structural Equation Modeling in Stan. Journal of Statistical Software, 100(6), 1-22. URL http://www.jstatsoft.org/v100/i06/.

Edgar C. Merkle & Yves Rosseel (2018). blavaan: Bayesian Structural Equation Models via Parameter Expansion. Journal of Statistical Software, 85(4), 1-30. URL http://www.jstatsoft.org/v85/i04/.

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

See Also

`blavaan`

Examples

```
## Not run:
## linear growth model with a time-varying covariate
data(Demo.growth, package = "lavaan")

model.syntax <- '
  # intercept and slope with fixed coefficients
    i =~ 1*t1 + 1*t2 + 1*t3 + 1*t4
    s =~ 0*t1 + 1*t2 + 2*t3 + 3*t4

  # regressions
    i ~ x1 + x2
    s ~ x1 + x2

  # time-varying covariates
    t1 ~ c1
    t2 ~ c2
    t3 ~ c3
    t4 ~ c4
```

```
'

fit <- bgrowth(model.syntax, data = Demo.growth)
summary(fit)

## End(Not run)
```

`blav_internal` *blavaan internal functions*

Description

Internal functions related to Bayesian model estimation. Not to be called by the user.

`blavaan` *Fit a Bayesian Latent Variable Model*

Description

Fit a Bayesian latent variable model.

Usage

```
blavaan(..., cp = "srs",
    dp = NULL, n.chains = 3, burnin, sample,
    adapt, mcmcfile = FALSE, mcmcextra = list(), inits = "simple",
    convergence = "manual", target = getOption("blavaan.target", "stan"),
   save.lvs = FALSE, wiggle = NULL, wiggle.sd = 0.1, prisamp = FALSE, jags.ic = FALSE,
    seed = NULL, bcontrol = list())
```

Arguments

`...`	Default lavaan arguments. See `lavaan`. This also accepts sampler-specific arguments that would otherwise need to be supplied via `bcontrol` – see `bcontrol` below.
`cp`	Handling of prior distributions on covariance parameters: possible values are `"srs"` (default) or `"fa"`. Option `"fa"` is only available for `target="jags"`.
`dp`	Default prior distributions on different types of parameters, typically the result of a call to `dpriors()`. See the `dpriors()` help file for more information.
`n.chains`	Number of desired MCMC chains.
`burnin`	Number of burnin/warmup iterations (not including the adaptive iterations, for target="jags"). Defaults to 4000 or target="jags" and 500 for Stan targets.
`sample`	The total number of samples to take after burnin. Defaults to 10000 for target="jags" and 1000 for Stan targets.

adapt — For target="jags", the number of adaptive iterations to use at the start of sampling. Defaults to 1000.

mcmcfile — If TRUE, the JAGS/Stan model and data will be written to files (in the lavExport directory). Can also supply a character string, which serves as the name of the directory to which files will be written.

mcmcextra — A list with potential names syntax (unavailable for target="stan"), monitor, data, and llnsamp. The syntax object is a text string containing extra code to insert in the JAGS/Stan model syntax. The data object is a list of extra data to send to the JAGS/Stan model. If moment_match_k_threshold is specified within data the looic of the model will be calculated using moment matching. The monitor object is a character vector containing extra JAGS/Stan parameters to monitor. The llnsamp object is only relevant to models with ordinal variables, and specifies the number of samples that should be drawn to approximate the model log-likelihood (larger numbers imply higher accuracy and longer time). This log-likelihood is specifically used to compute information criteria.

inits — If it is a character string, the options are currently "simple" (default), "Mplus", "prior", or "jags". In the first two cases, parameter values are set as though they will be estimated via ML (see lavaan). The starting parameter value for each chain is then perturbed from the original values through the addition of random uniform noise. If "prior" is used, the starting parameter values are obtained based on the prior distributions (while also trying to ensure that the starting values will not crash the model estimation). If "jags", no starting values are specified and JAGS will choose values on its own (and this will probably crash Stan targets). You can also supply a list of starting values for each chain, where the list format can be obtained from, e.g., blavInspect(fit, "inits"). Finally, you can specify starting values in a similar way to lavaan, using the lavaan start argument (see the lavaan documentation for all the options there). In this case, you should also set inits="simple", and be aware that the same starting values will be used for each chain.

convergence — Useful only for target="jags". If "auto", parameters are sampled until convergence is achieved (via autorun.jags()). In this case, the arguments burnin and sample are passed to autorun.jags() as startburnin and startsample, respectively. Otherwise, parameters are sampled as specified by the user (or by the run.jags defaults).

target — Desired MCMC sampling, with "stan" (pre-compiled marginal approach) as default. Also available is "vb", which calls the rstan function vb(). Other options include "jags", "stancond", and "stanclassic", which sample latent variables and provide some greater functionality (because syntax is written "on the fly"). But they are slower and less efficient.

save.lvs — Should sampled latent variables (factor scores) be saved? Logical; defaults to FALSE

wiggle — Labels of equality-constrained parameters that should be "approximately" equal. Can also be "intercepts", "loadings", "regressions", "means".

wiggle.sd — The prior sd (of normal distribution) to be used in approximate equality constraints. Can be one value, or (for target="stan") a numeric vector of values that is the same length as wiggle.

prisamp — Should samples be drawn from the prior, instead of the posterior (target="stan" only)? Logical; defaults to FALSE

jags.ic — Should DIC be computed the JAGS way, in addition to the BUGS way? Logical; defaults to FALSE

seed — A vector of length n.chains (for target "jags") or an integer (for target "stan") containing random seeds for the MCMC run. If NULL, seeds will be chosen randomly.

bcontrol — A list containing additional parameters passed to run.jags (or autorun.jags), stan, vb, or (for target="cmdstan") a cmdstanr model's $sample() method. See the manpage of those functions for an overview of the additional parameters that can be set. Recognized arguments (e.g. adapt_delta, max_treedepth, cores, thin, method, threads_per_chain, and other target-specific sampler arguments) can instead be supplied directly to blavaan()/bcfa()/bsem()/bgrowth() without wrapping them in bcontrol; for target %in% c("stan","stanclassic", "stancond"), arguments belonging to Stan's nested control= list (like adapt_delta) are detected automatically and nested for you. Supplying the same argument both directly and inside bcontrol is an error. A few names remain reserved and must still be supplied via bcontrol because they collide with another blavaan()/lavaan() argument: verbose, control (lavaan's own optimizer control), and cl (lavaan's bootstrap cluster option).

Value

An object that inherits from class lavaan, for which several methods are available, including a summary method.

References

Edgar C. Merkle, Ellen Fitzsimmons, James Uanhoro, & Ben Goodrich (2021). Efficient Bayesian Structural Equation Modeling in Stan. Journal of Statistical Software, 100(6), 1-22. URL http://www.jstatsoft.org/v100/i06/.

Edgar C. Merkle & Yves Rosseel (2018). blavaan: Bayesian Structural Equation Models via Parameter Expansion. Journal of Statistical Software, 85(4), 1-30. URL http://www.jstatsoft.org/v85/i04/.

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

See Also

bcfa, bsem, bgrowth

Examples

```
## Not run:
data(HolzingerSwineford1939, package = "lavaan")

# The Holzinger and Swineford (1939) example
HS.model <- ' visual  =~ x1 + x2 + x3
              textual =~ x4 + x5 + x6
              speed   =~ x7 + x8 + x9 '
```

```
fit <- blavaan(HS.model, data = HolzingerSwineford1939,
               auto.var = TRUE, auto.fix.first = TRUE,
               auto.cov.lv.x = TRUE)
summary(fit)
coef(fit)

## End(Not run)
```

`blavaan-class` *Class For Representing A (Fitted) Bayesian Latent Variable Model*

Description

The `blavaan` class contains the `lavaan` class, representing a (fitted) Bayesian latent variable model. It contains a description of the model as specified by the user, a summary of the data, an internal matrix representation, and if the model was fitted, the fitting results.

Objects from the Class

Objects can be created via the `bcfa`, `bsem`, `bgrowth` or `blavaan` functions.

Slots

`version`: The lavaan package version used to create this objects

`call`: The function call as returned by `match.call()`.

`timing`: The elapsed time (user+system) for various parts of the program as a list, including the total time.

`Options`: Named list of options that were provided by the user, or filled-in automatically.

`ParTable`: Named list describing the model parameters. Can be coerced to a data.frame. In the documentation, this is called the 'parameter table'.

`pta`: Named list containing parameter table attributes.

`Data`: Object of internal class `"Data"`: information about the data.

`SampleStats`: Object of internal class `"SampleStats"`: sample statistics

`Model`: Object of internal class `"Model"`: the internal (matrix) representation of the model

`Cache`: List using objects that we try to compute only once, and reuse many times.

`Fit`: Object of internal class `"Fit"`: the results of fitting the model. No longer used.

`boot`: List. Unused for Bayesian models.

`optim`: List. Information about the optimization.

`loglik`: List. Information about the loglikelihood of the model (if maximum likelihood was used).

`implied`: List. Model implied statistics.

`vcov`: List. Information about the variance matrix (vcov) of the model parameters.

`test`: List. Different test statistics.

`h1`: List. Information about the unrestricted h1 model (if available).

`baseline`: List. Information about a baseline model (often the independence model) (if available).

`external`: List. Includes Stan or JAGS objects used for MCMC.

Methods

coef `signature(object = "blavaan", type = "free")`: Returns the estimates of the parameters in the model as a named numeric vector. If `type="free"`, only the free parameters are returned. If `type="user"`, all parameters listed in the parameter table are returned, including constrained and fixed parameters.

vcov `signature(object = "lavaan")`: returns the covariance matrix of the estimated parameters.

show `signature(object = "blavaan")`: Print a short summary of the model fit

summary `signature(object = "blavaan", header = TRUE, fit.measures = FALSE, estimates = TRUE, ci = TRUE, standardized = FALSE, rsquare = FALSE, std.nox = FALSE, psrf = TRUE, neff = FALSE, postmedian = FALSE, postmode = FALSE, priors = TRUE, bf = FALSE, level = .95, hpd = FALSE, nd = 3L)`: Print a nice summary of the model estimates. If `header = TRUE`, the header section (including fit measures) is printed. If `fit.measures = TRUE`, additional fit measures are added to the header section. If `estimates = TRUE`, print the parameter estimates section. If `ci = TRUE`, add confidence intervals to the parameter estimates section. `level` sets the credible-interval mass (default .95), and `hpd = TRUE` switches from an equal-tailed quantile interval to a highest-posterior-density interval. Both are computed directly from the MCMC draws for free *and* defined (`:=`) parameters, for `target = "stan"/"cmdstan"` fits (for other targets, the interval is unaffected by `level`/`hpd`). `psrf`, `neff`, `postmedian`, and `postmode` are likewise now reported for defined parameters under `target = "stan"/"cmdstan"`, rather than left blank. If `standardized = TRUE`, the standardized solution is also printed. Note that *SE*s and tests are still based on unstandardized estimates. Use `standardizedSolution` to obtain *SE*s and test statistics for standardized estimates. If `rsquare=TRUE`, the R-Square values for the dependent variables in the model are printed. If `std.nox = TRUE`, the `std.all` column contains the the `std.nox` column from the parameterEstimates() output. If `psrf = TRUE`, potential scale reduction factors (Rhats) are printed. If `neff = TRUE`, effective sample sizes are printed. If `postmedian` or `postmode` are TRUE, posterior medians or modes are printed instead of posterior means. If `priors = TRUE`, parameter prior distributions are printed. If `bf = TRUE`, Savage-Dickey approximations of the Bayes factor are printed for certain parameters. Nothing is returned (use `lavInspect` or another extractor function to extract information from a fitted model).

References

Edgar C. Merkle, Ellen Fitzsimmons, James Uanhoro, & Ben Goodrich (2021). Efficient Bayesian Structural Equation Modeling in Stan. Journal of Statistical Software, 100(6), 1-22. URL http://www.jstatsoft.org/v100/i06/.

Edgar C. Merkle & Yves Rosseel (2018). blavaan: Bayesian Structural Equation Models via Parameter Expansion. Journal of Statistical Software, 85(4), 1-30. URL http://www.jstatsoft.org/v85/i04/.

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

See Also

`bcfa`, `bsem`, `bgrowth`

Examples

```
## Not run:
```

```
HS.model <- ' visual  =~ x1 + x2 + x3
              textual =~ x4 + x5 + x6
              speed   =~ x7 + x8 + x9 '

fit <- bcfa(HS.model, data=HolzingerSwineford1939)

summary(fit, standardized=TRUE, fit.measures=TRUE, rsquare=TRUE)
coef(fit)

## End(Not run)
```

`blavCompare` *Bayesian model comparisons*

Description

Bayesian model comparisons, including WAIC, LOO, and Bayes factor approximation.

Usage

```
blavCompare(object1, object2, ...)
```

Arguments

`object1`	An object of class `blavaan`.
`object2`	A second object of class `blavaan`.
`...`	Other arguments to loo().

Details

This function computes Bayesian model comparison metrics, including a Bayes factor approximation, WAIC, and LOOIC.

The log-Bayes factor of the two models is based on the Laplace approximation to each model's marginal log-likelihood.

The WAIC and LOOIC metrics come from the loo package. The ELPD difference and SE specifically come from loo::loo_compare().

Value

A list containing separate results for log-Bayes factor, WAIC, LOOIC, and differences between WAIC and LOOIC.

References

Raftery, A. E. (1993). Bayesian model selection in structural equation models. In K. A. Bollen & J. S. Long (Eds.), Testing structural equation models (pp. 163-180). Beverly Hills, CA: Sage.

Vehtari A., Gelman A., Gabry J. (2017). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. Statistics and Computing, 27, 1413-1432.

Examples

```
## Not run:
data(HolzingerSwineford1939, package = "lavaan")

hsm1 <- ' visual  =~ x1 + x2 + x3 + x4
          textual =~ x4 + x5 + x6
          speed   =~ x7 + x8 + x9 '

fit1 <- bcfa(hsm1, data = HolzingerSwineford1939)

hsm2 <- ' visual  =~ x1 + x2 + x3
          textual =~ x4 + x5 + x6 + x7
          speed   =~ x7 + x8 + x9 '

fit2 <- bcfa(hsm2, data = HolzingerSwineford1939)

blavCompare(fit1, fit2)

## End(Not run)
```

---

blavFitIndices *SEM Fit Indices for Bayesian SEM*

Description

This function provides a posterior distribution of some χ^2-based fit indices to assess the global fit of a latent variable model.

Usage

```
blavFitIndices(object, thin = 1L, pD = c("loo","waic","dic"),
               rescale = c("devM","ppmc","mcmc"),
               fit.measures = "all", baseline.model = NULL)

## S4 method for signature 'blavFitIndices'
## S4 method for signature 'blavFitIndices'
summary(object, ...)

## S3 method for class 'bfi'
summary(object, central.tendency = c("mean","median","mode"),
        hpd = TRUE, prob = .90)
```

Arguments

`object`	An object of class `blavaan`.
`thin`	Optional `integer` indicating how much to thin each chain. Default is `1L`, indicating not to thin the chains.

pD — character indicating from which information criterion returned by fitMeasures(object) to use the estimated number of parameters. The default is from the leave-one-out information criterion (LOO-IC), which is most highly recommended by Vehtari et al. (2017).

rescale — character indicating the method used to calculate fit indices. If rescale = "devM" (default), the Bayesian analog of the χ^2 statistic (the deviance evaluated at the posterior mean of the model parameters) is approximated by rescaling the deviance at each iteration by subtracting the estimated number of parameters. If rescale = "PPMC", the deviance at each iteration is rescaled by subtracting the deviance of data simulated from the posterior predictive distribution (as in posterior predictive model checking; see Hoofs et al., 2017). If rescale = "MCMC", the fit measures are simply calculated using fitMeasures at each iteration of the Markov chain(s), based on the model-implied moments at that iteration (NOT advised when the model includes informative priors, in which case the model's estimated *pD* will deviate from the number of parameters used to calculate *df* in fitMeasures).

fit.measures — If "all", all fit measures available will be returned. If only a single or a few fit measures are specified by name, only those are computed and returned. If rescale = "devM" or "PPMC", the currently available indices are "BRMSEA", "BGammaHat", "adjBGammaHat", "BMc", "BCFI", "BTLI", or "BNFI". If rescale = "MCMC", the user may request any indices returned by fitMeasures for objects of class lavaan.

baseline.model — If not NULL, an object of class blavaan, representing a user-specified baseline model. If a baseline.model is provided, incremental fit indices (BCFI, BTLI, or BNFI) can be requested in fit.measures. Ignored if rescale = "MCMC".

... — Additional arguments to the summary method:

central.tendency — Takes values "mean", "median", "mode", indicating which statistics should be used to characterize the location of the posterior distribution. By default, all 3 statistics are returned. The posterior mean is labeled EAP for *expected a posteriori* estimate, and the mode is labeled MAP for *modal a posteriori* estimate.

hpd — A logical indicating whether to calculate the highest posterior density (HPD) credible interval for each fit index (defaults to TRUE).

prob — The "confidence" level of the credible interval(s) (defaults to 0.9).

Value

An S4 object of class blavFitIndices consisting of 2 slots:

@details — A list containing the choices made by the user (or defaults; e.g., which values of pD and rescale were set), as well as the posterior distribution of the χ^2 (deviance) statistic (rescaled, if rescale = "devM" or "PPMC").

@indices — A list containing the posterior distribution of each requested fit.measure.

The summary() method returns a data.frame containing one row for each requested fit.measure, and columns containing the specified measure(s) of central.tendency, the posterior *SD*, and (if requested) the HPD credible-interval limits.

Author(s)

Mauricio Garnier-Villareal (Vrije Universiteit Amsterdam; <mgv@pm.me>)

Terrence D. Jorgensen (University of Amsterdam; <TJorgensen314@gmail.com>)

References

`rescale = "PPMC"` based on:

Hoofs, H., van de Schoot, R., Jansen, N. W., & Kant, I. (2017). Evaluating model fit in Bayesian confirmatory factor analysis with large samples: Simulation study introducing the BRMSEA. *Educational and Psychological Measurement*. doi:10.1177/0013164417709314

`rescale = "devM"` based on:

Garnier-Villarreal, M., & Jorgensen, T. D. (2020). Adapting Fit Indices for Bayesian Structural Equation Modeling: Comparison to Maximum Likelihood. *Psychological Methods*, 25(1), 46–70. https://dx.doi.org/10.1037/met0000224 (See also https://osf.io/afkcw/)

Other references:

Vehtari, A., Gelman, A., & Gabry, J. (2017). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and Computing, 27*(5), 1413–1432. doi:10.1007/s11222-016-9696-4

Examples

```
 ## Not run:
data(HolzingerSwineford1939, package = "lavaan")

HS.model <- ' visual  =~ x1 + x2 + x3
              textual =~ x4 + x5 + x6
              speed   =~ x7 + x8 + x9 '
## fit target model
fit1 <- bcfa(HS.model, data = HolzingerSwineford1939,
             n.chains = 2, burnin = 1000, sample = 1000)

## fit null model to calculate CFI, TLI, and NFI
null.model <- c(paste0("x", 1:9, " ~~ x", 1:9), paste0("x", 1:9, " ~ 1"))
fit0 <- bcfa(null.model, data = HolzingerSwineford1939,
             n.chains = 2, burnin = 1000, sample = 1000)

## calculate posterior distributions of fit indices

## The default method mimics fit indices derived from ML estimation
ML <- blavFitIndices(fit1, baseline.model = fit0)
ML
summary(ML)

## other options:

## - use Hoofs et al.'s (2017) PPMC-based method
## - use the estimated number of parameters from WAIC instead of LOO-IC
PPMC <- blavFitIndices(fit1, baseline.model = fit0,
                       pD = "waic", rescale = "PPMC")
```

```
## issues a warning about using rescale="PPMC" with N < 1000 (see Hoofs et al.)

## - specify only the desired measures of central tendency
## - specify a different "confidence" level for the credible intervals
summary(PPMC, central.tendency = c("mean","mode"), prob = .95)



## Access the posterior distributions for further investigation
head(distML <- data.frame(ML@indices))

## For example, diagnostic plots using the bayesplot package:

## distinguish chains
nChains <- blavInspect(fit1, "n.chains")
distML$Chain <- rep(1:nChains, each = nrow(distML) / nChains)

library(bayesplot)
mcmc_pairs(distML, pars = c("BRMSEA","BMc","BGammaHat","BCFI","BTLI"),
           diag_fun = "hist")
## Indices are highly correlated across iterations in both chains

## Compare to PPMC method
distPPMC <- data.frame(PPMC@indices)
distPPMC$Chain <- rep(1:nChains, each = nrow(distPPMC) / nChains)
mcmc_pairs(distPPMC, pars = c("BRMSEA","BMc","BGammaHat","BCFI","BTLI"),
           diag_fun = "dens")
## nonlinear relation between BRMSEA, related to the floor effect of BRMSEA
## that Hoofs et al. found for larger (12-indicator) models


## End(Not run)
```

`blavInspect` *Inspect or Extract Information from a Fitted blavaan Object*

Description

The `blavInspect()` and `blavTech()` functions can be used to inspect/extract information that is stored inside (or can be computed from) a fitted blavaan object. This is similar to lavaan's `lavInspect()` function.

Usage

```
blavInspect(blavobject, what, ...)

blavTech(blavobject, what, ...)
```

Arguments

blavobject	An object of class blavaan.
what	Character. What needs to be inspected/extracted? See Details for Bayes-specific options, and see lavaan's lavInspect() for additional options. Note: the what argument is not case-sensitive (everything is converted to lower case.)
...	lavaan arguments supplied to lavInspect(); see lavaan.

Details

Below is a list of Bayesian-specific values for the what argument; additional values can be found in the lavInspect() documentation.

"start": A list of starting values for each chain, unless inits="jags" is used during model estimation. Aliases: "starting.values", "inits".

"rhat": Each parameter's potential scale reduction factor for convergence assessment. Can also use "psrf" instead of "rhat". For target = "stan"/"cmdstan", this also includes each defined (:=) parameter, appended after the free parameters.

"ac.10": Each parameter's estimated lag-10 autocorrelation.

"neff": Each parameters effective sample size, taking into account autocorrelation. For target = "stan"/"cmdstan", this also includes each defined (:=) parameter, appended after the free parameters.

"mcmc": An object of class mcmc containing the individual parameter draws from the MCMC run. Aliases: "draws", "samples".

"mcobj": The underlying run.jags or stan object that resulted from the MCMC run.

"n.chains": The number of chains sampled.

"cp": The approach used for estimating covariance parameters ("srs" or "fa"); these are only relevant if using JAGS.

"dp": Default prior distributions used for each type of model parameter.

"postmode": Estimated posterior mode of each free parameter (and, for target = "stan"/"cmdstan", each defined (:=) parameter, appended after the free parameters).

"postmean": Estimated posterior mean of each free parameter (and, for target = "stan"/"cmdstan", each defined (:=) parameter, appended after the free parameters).

"postmedian": Estimated posterior median of each free parameter (and, for target = "stan"/"cmdstan", each defined (:=) parameter, appended after the free parameters).

"lvs": An object of class mcmc containing latent variable (factor score) draws. In two-level models, use level = 1 or level = 2 to specify which factor scores you want.

"lvmeans": A matrix of mean factor scores (rows are observations, columns are variables). Use the additional level argument in the same way.

"hpd": HPD interval of each free parameter, plus each defined (:=) parameter for target = "stan"/"cmdstan". In this case, the prob (or level) argument can be used to specify a number in (0,1) reflecting the desired percentage of the interval.

"ci": Equal-tailed quantile credible interval of each free parameter, plus each defined (:=) parameter for target = "stan"/"cmdstan", computed directly from the MCMC draws (as opposed to a Gaussian approximation). The prob (or level) argument specifies a number in (0,1) reflecting the desired percentage of the interval; default is .95.

See Also

lavInspect, bcfa, bsem, bgrowth

Examples

```
## Not run:
# The Holzinger and Swineford (1939) example
data(HolzingerSwineford1939, package = "lavaan")

HS.model <- ' visual  =~ x1 + x2 + x3
              textual =~ x4 + x5 + x6
              speed   =~ x7 + x8 + x9 '

fit <- bcfa(HS.model, data = HolzingerSwineford1939,
            bcontrol = list(method = "rjparallel"))

# extract information
blavInspect(fit, "psrf")
blavInspect(fit, "hpd", prob = .9)
blavInspect(fit, "ci", level = .9)

## End(Not run)
```

---

blavPredict *Predict the values of latent variables, observed variables, and missing variables.*

Description

The purpose of the blavPredict() function is to compute various types of model predictions, conditioned on observed data. This differs somewhat from lavPredict() in lavaan.

Usage

```
blavPredict(object, newdata = NULL, type = "lv", level = 1L)
```

Arguments

object	An object of class blavaan.
newdata	An optional data.frame, containing the same variables as the data.frame used when fitting the model in object.
type	A character string. If "lv", estimated values for the latent variables in the model are computed. If "ov" or "yhat", predicted means for the observed variables in the model are computed. If "ypred" or "ydist", predicted values for the observed variables (including residual noise) are computed. If "ymis" or "ovmis", model predicted values ("imputations") for the missing data are computed. See details for further information.
level	For type = "lv", used to specify whether one desires the level 1 latent variables or level 2 latent variables.

Details

The `predict()` function calls the `blavPredict()` function with its default options.

Below, we provide more information about each `type` option. Most options only work for target="stan", and "number of samples" is defined as the number of posterior samples across all chains.

`type="lv"`: The posterior distribution of latent variables conditioned on observed variables. Returns a list with "number of samples" entries, where each entry is a matrix where rows are observations and columns are latent variables.

`type="yhat"`: The posterior expected value of observed variables conditioned on the sampled latent variables. Returns a list with "number of samples" entries, where each entry is a matrix where rows are observations and columns are observed variables.

`type="ypred"`: The posterior predictive distribution of observed variables conditioned on the sampled latent variables (including residual variability). Returns a list with "number of samples" entries, where each entry is a data frame where rows are observations and columns are observed variables.

`type="ymis"`: The posterior predictive distribution of missing values conditioned on observed variables. Returns a matrix with "number of samples" rows and "number of missing variables" columns.

See Also

Users may also wish to generate the posterior predictive distribution of observed data, not conditioned on the latent variables. This would often be viewed as data from new clusters (people) that were not observed in the original dataset. For that, see `sampleData()`.

Examples

```
## Not run:
data(HolzingerSwineford1939, package = "lavaan")

## fit model
HS.model <- ' visual  =~ x1 + x2 + x3
              textual =~ x4 + x5 + x6
              speed   =~ x7 + x8 + x9 '

fit <- bcfa(HS.model, data = HolzingerSwineford1939, save.lvs = TRUE)
lapply(blavPredict(fit)[1:2], head) # first 6 rows of first 10 posterior samples
head(blavPredict(fit, type = "yhat")[[1]]) # top of first posterior sample

## multigroup models return a list of factor scores (one per group)
mgfit <- bcfa(HS.model, data = HolzingerSwineford1939, group = "school",
              group.equal = c("loadings","intercepts"), save.lvs = TRUE)

lapply(blavPredict(fit)[1:2], head)
head(blavPredict(fit, type = "ypred")[[1]])

## End(Not run)
```

bsem *Fit Structural Equation Models*

Description

Fit a Structural Equation Model (SEM).

Usage

```
bsem(..., cp = "srs",
     dp = NULL, n.chains = 3, burnin, sample,
     adapt, mcmcfile = FALSE, mcmcextra = list(), inits = "simple",
     convergence = "manual", target = getOption("blavaan.target", "stan"),
     save.lvs = FALSE, wiggle = NULL, wiggle.sd = 0.1, prisamp = FALSE,
     jags.ic = FALSE, seed = NULL, bcontrol = list())
```

Arguments

`...`	Default lavaan arguments. See lavaan. This also accepts sampler-specific arguments that would otherwise need to be supplied via `bcontrol` – see `bcontrol` below.
`cp`	Handling of prior distributions on covariance parameters: possible values are `"srs"` (default) or `"fa"`. Option `"fa"` is only available for `target="jags"`.
`dp`	Default prior distributions on different types of parameters, typically the result of a call to `dpriors()`. See the `dpriors()` help file for more information.
`n.chains`	Number of desired MCMC chains.
`burnin`	Number of burnin/warmup iterations (not including the adaptive iterations, for target="jags"). Defaults to 4000 or target="jags" and 500 for Stan targets.
`sample`	The total number of samples to take after burnin. Defaults to 10000 for target="jags" and 1000 for Stan targets.
`adapt`	For target="jags", the number of adaptive iterations to use at the start of sampling. Defaults to 1000.
`mcmcfile`	If `TRUE`, the JAGS/Stan model will be written to file (in the lavExport directory). Can also supply a character string, which serves as the name of the directory to which files will be written.
`mcmcextra`	A list with potential names `syntax` (unavailable for `target="stan"`), `monitor`, `data`, and `llnsamp`. The `syntax` object is a text string containing extra code to insert in the JAGS/Stan model syntax. The `data` object is a list of extra data to send to the JAGS/Stan model. If `moment_match_k_threshold` is specified within `data` the looic of the model will be calculated using moment matching. The `monitor` object is a character vector containing extra JAGS/Stan parameters to monitor. The `llnsamp` object is only relevant to models with ordinal variables, and specifies the number of samples that should be drawn to approximate the model log-likelihood (larger numbers imply higher accuracy and longer time). This log-likelihood is specifically used to compute information criteria.

inits — If it is a character string, the options are currently "simple" (default), "Mplus", "prior", or "jags". In the first two cases, parameter values are set as though they will be estimated via ML (see lavaan). The starting parameter value for each chain is then perturbed from the original values through the addition of random uniform noise. If "prior" is used, the starting parameter values are obtained based on the prior distributions (while also trying to ensure that the starting values will not crash the model estimation). If "jags", no starting values are specified and JAGS will choose values on its own (and this will probably crash Stan targets). You can also supply a list of starting values for each chain, where the list format can be obtained from, e.g., blavInspect(fit, "inits"). Finally, you can specify starting values in a similar way to lavaan, using the lavaan start argument (see the lavaan documentation for all the options there). In this case, you should also set inits="simple", and be aware that the same starting values will be used for each chain.

convergence — Useful only for target="jags". If "auto", parameters are sampled until convergence is achieved (via autorun.jags()). In this case, the arguments burnin and sample are passed to autorun.jags() as startburnin and startsample, respectively. Otherwise, parameters are sampled as specified by the user (or by the run.jags defaults).

target — Desired MCMC sampling, with "stan" (pre-compiled marginal approach) as default. Also available is "vb", which calls the rstan function vb(). Other options include "jags", "stancond", and "stanclassic", which sample latent variables and provide some greater functionality (because syntax is written "on the fly"). But they are slower and less efficient.

save.lvs — Should sampled latent variables (factor scores) be saved? Logical; defaults to FALSE

wiggle — Labels of equality-constrained parameters that should be "approximately" equal. Can also be "intercepts", "loadings", "regressions", "means".

wiggle.sd — The prior sd (of normal distribution) to be used in approximate equality constraints. Can be one value, or (for target="stan") a numeric vector of values that is the same length as wiggle.

prisamp — Should samples be drawn from the prior, instead of the posterior (target="stan" only)? Logical; defaults to FALSE

jags.ic — Should DIC be computed the JAGS way, in addition to the BUGS way? Logical; defaults to FALSE

seed — A vector of length n.chains (for target "jags") or an integer (for target "stan") containing random seeds for the MCMC run. If NULL, seeds will be chosen randomly.

bcontrol — A list containing additional parameters passed to run.jags (or autorun.jags), stan, vb, or (for target="cmdstan") a cmdstanr model's $sample() method. See the manpage of those functions for an overview of the additional parameters that can be set. Recognized arguments (e.g. adapt_delta, max_treedepth, cores, thin, method, threads_per_chain, and other target-specific sampler arguments) can instead be supplied directly to blavaan()/bcfa()/bsem()/bgrowth() without wrapping them in bcontrol; for target %in% c("stan","stanclassic", "stancond"), arguments belonging to Stan's nested control= list (like adapt_delta)

are detected automatically and nested for you. Supplying the same argument both directly and inside `bcontrol` is an error. A few names remain reserved and must still be supplied via `bcontrol` because they collide with another `blavaan()`/`lavaan()` argument: `verbose`, `control` (lavaan's own optimizer control), and `cl` (lavaan's bootstrap cluster option).

Details

The `bsem` function is a wrapper for the more general `blavaan` function, using the following default `lavaan` arguments: `int.ov.free` = `TRUE`, `int.lv.free` = `FALSE`, `auto.fix.first` = `TRUE` (unless `std.lv` = `TRUE`), `auto.fix.single` = `TRUE`, `auto.var` = `TRUE`, `auto.cov.lv.x` = `TRUE`, `auto.th` = `TRUE`, `auto.delta` = `TRUE`, and `auto.cov.y` = `TRUE`.

Value

An object of class lavaan, for which several methods are available, including a `summary` method.

References

Edgar C. Merkle, Ellen Fitzsimmons, James Uanhoro, & Ben Goodrich (2021). Efficient Bayesian Structural Equation Modeling in Stan. Journal of Statistical Software, 100(6), 1-22. URL http://www.jstatsoft.org/v100/i06/.

Edgar C. Merkle & Yves Rosseel (2018). blavaan: Bayesian Structural Equation Models via Parameter Expansion. Journal of Statistical Software, 85(4), 1-30. URL http://www.jstatsoft.org/v85/i04/.

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

See Also

`blavaan`

Examples

```
# The industrialization and Political Democracy Example
# Bollen (1989), page 332
data(PoliticalDemocracy, package = "lavaan")

model <- '
  # latent variable definitions
     ind60 =~ x1 + x2 + x3
     dem60 =~ y1 + a*y2 + b*y3 + c*y4
     dem65 =~ y5 + a*y6 + b*y7 + c*y8

  # regressions
    dem60 ~ ind60
    dem65 ~ ind60 + dem60

  # residual correlations
    y1 ~~ y5
    y2 ~~ y4 + y6
    y3 ~~ y7
```

```
    y4 ~~ y8
    y6 ~~ y8
'

## Not run:
# mildly informative priors for mv intercepts and loadings
fit <- bsem(model, data = PoliticalDemocracy,
            dp = dpriors(nu = "normal(5,10)", lambda = "normal(1,.5)"))
summary(fit)

## End(Not run)

# A short run for rough results
fit <- bsem(model, data = PoliticalDemocracy, burnin = 100, sample = 100,
            dp = dpriors(nu = "normal(5,10)", lambda = "normal(1,.5)"),
            n.chains = 2)
summary(fit)
```

`dpriors` *Specify Default Prior Distributions*

Description

Specify "default" prior distributions for classes of model parameters.

Usage

```
dpriors(..., target = "stan")
```

Arguments

`...`	Parameter names paired with desired priors (see example below).
`target`	Are the priors for jags, stan (default), or stanclassic?

Details

The prior distributions always use JAGS/Stan syntax and parameterizations. For example, the normal distribution in JAGS is parameterized via the precision, whereas the normal distribution in Stan is parameterized via the standard deviation.

User-specified prior distributions for specific parameters (using the `prior()` operator within the model syntax) always override prior distributions set using `dpriors()`.

The parameter names are:

- nu: Observed variable intercept parameters.
- alpha: Latent variable intercept parameters.
- lambda: Loading parameters.
- beta: Regression parameters.

- itheta: Observed variable precision parameters.
- ipsi: Latent variable precision parameters.
- rho: Correlation parameters (associated with covariance parameters).
- ibpsi: Inverse covariance matrix of blocks of latent variables (used for `target="jags"`).
- tau: Threshold parameters (ordinal data only).
- delta: Delta parameters (ordinal data only).

Value

A character vector containing the prior distribution for each type of parameter.

References

Edgar C. Merkle, Ellen Fitzsimmons, James Uanhoro, & Ben Goodrich (2021). Efficient Bayesian Structural Equation Modeling in Stan. Journal of Statistical Software, 100(6), 1-22. URL http://www.jstatsoft.org/v100/i06/.

Edgar C. Merkle & Yves Rosseel (2018). blavaan: Bayesian Structural Equation Models via Parameter Expansion. Journal of Statistical Software, 85(4), 1-30. URL http://www.jstatsoft.org/v85/i04/.

See Also

bcfa, bsem, bgrowth

Examples

```
dpriors(nu = "normal(0,10)", lambda = "normal(0,1)", rho = "beta(3,3)")
```

---

glance.blavaan *Glance at a blavaan object*

Description

Glance at a blavaan object

Usage

```
## S3 method for class 'blavaan'
glance(x, fit.indices = FALSE, ...)
```

Arguments

x	A blavaan object
fit.indices	Character vector of fit indices to compute from fitMeasures and blavFitIndices. Use "TRUE" to also compute blavFitIndices measures. Default is "FALSE".
...	Additional arguments (currently ignored).

Value

A single-row `data.frame` with columns:

`npar`	Number of estimated parameters
`ngroups`	Number of groups
`ppp`	Posterior predictive p-value
`bic`	Bayesian information criterion
`dic`	Deviance information criterion
`waic`	Widely applicable information criterion
`looic`	Leave-one-out information criterion
`margloglik`	Marginal log-likelihood

Additional EAP of fit measures from `blavFitIndices` are included if requested.

Examples

```
## Not run:
library(blavaan)

HS.model <- 'visual =~ x1 + x2 + x3'
fit <- bcfa(HS.model, data = HolzingerSwineford1939, seed = 123)
glance(fit)
glance(fit, fit.indices = TRUE)  # includes BRMSEA

## End(Not run)
```

`plot.blavaan` *blavaan Diagnostic Plots*

Description

Convenience functions to create plots of blavaan objects, via the bayesplot package.

Usage

```
## S3 method for class 'blavaan'
plot(x, pars = NULL, plot.type = "trace", showplot = TRUE, ...)
```

Arguments

`x`	An object of class `blavaan`.
`pars`	Parameter numbers to plot, where the numbers correspond to the order of parameters as reported by `coef()` (also as shown in the 'free' column of the parTable). If no numbers are provided, all free parameters will be plotted.

plot.type	The type of plot desired. This should be the name of a MCMC function, without the mcmc_ prefix.
showplot	Should the plot be sent to the graphic device? Defaults to TRUE.
...	Other arguments sent to the bayesplot function.

Details

In previous versions of blavaan, the plotting functionality was handled separately for JAGS and for Stan (using plot functionality in packages runjags and rstan, respectively). For uniformity, all plotting functionality is now handled by bayesplot. If users desire additional functionality that is not immediately available, they can extract the matrix of MCMC draws via `as.matrix(blavInspect(x, 'mcmc'))`.

Value

An invisible ggplot object that, if desired, can be further customized.

Examples

```
## Not run:
data(HolzingerSwineford1939, package = "lavaan")

HS.model <- ' visual  =~ x1 + x2 + x3
              textual =~ x4 + x5 + x6
              speed   =~ x7 + x8 + x9 '

fit <- bcfa(HS.model, data = HolzingerSwineford1939)

# trace plots of free loadings
plot(fit, pars = 1:6)

## End(Not run)
```

---

ppmc *Posterior Predictive Model Checks*

Description

This function allows users to conduct a posterior predictive model check to assess the global or local fit of a latent variable model using any discrepancy function that can be applied to a lavaan model.

Usage

```
ppmc(object, thin = 1, fit.measures = c("srmr","chisq"), discFUN = NULL,
     conditional = FALSE)

## S4 method for signature 'blavPPMC'
```

```
summary(object, ...)

## S3 method for class 'ppmc'
summary(object, discFUN, dist = c("obs","sim"),
        central.tendency = c("mean","median","mode"),
        hpd = TRUE, prob = .95, to.data.frame = FALSE, diag = TRUE,
        sort.by = NULL, decreasing = FALSE)

## S3 method for class 'blavPPMC'
plot(x, ..., discFUN, element, central.tendency = "",
     hpd = TRUE, prob = .95, nd = 3)

## S3 method for class 'blavPPMC'
hist(x, ..., discFUN, element, hpd = TRUE, prob = .95,
     printLegend = TRUE, legendArgs = list(x = "topleft"),
     densityArgs = list(), nd = 3)

## S3 method for class 'blavPPMC'
pairs(x, discFUN, horInd = 1:DIM, verInd = 1:DIM,
      printLegend = FALSE, ...)
```

Arguments

object, x	An object of class blavaan.
thin	Optional integer indicating how much to thin each chain. Default is 1L, indicating not to thin the chains in object.
fit.measures	character vector indicating the names of global discrepancy measures returned by fitMeasures. Ignored unless discFUN is NULL, but users may include fitMeasures in the list of discrepancy functions in discFUN. For ordinal models, the "logl" or "chisq" computations are done via lavaan.
discFUN	function, or a list of functions, that can be called on an object of class lavaan. Each function must return an object whose mode is numeric, but may be a vector, matrix, or multidimensional array. In the summary and plot methods, discFUN is a character indicating which discrepancy function to summarize.
conditional	logical indicating whether or not, during artificial data generation, we should condition on the estimated latent variables. Requires the model to be estimated with save.lvs = TRUE.
element	numeric or character indicating the index (in each dimension of the discFUN output, if multiple) to plot.
horInd, verInd	Similar to element, but a numeric or character vector indicating the indices of a matrix to plot in a scatterplot matrix. If horInd==verInd, histograms will be plotted in the upper triangle.
dist	character indicating whether to summarize the distribution of discFUN on either the observed or simulated data.
central.tendency	character indicating which statistics should be used to characterize the location of the posterior (predictive) distribution. By default, all 3 statistics are returned

	for the summary method, but none for the plot method. The posterior mean is labeled EAP for *expected a posteriori* estimate, and the mode is labeled MAP for *modal a posteriori* estimate.
hpd	logical indicating whether to calculate the highest posterior density (HPD) credible interval for discFUN.
prob	The "confidence" level of the credible interval(s).
nd	The number of digits to print in the scatterplot.
to.data.frame	logical indicating whether the summary of a symmetric 2-dimensional matrix returned by discFUN should have its unique elements stored in rows of a data.frame that can be sorted for convenience of identifying large discrepancies. If discFUN returns an asymmetric 2-dimensional matrix, the list of matrices returned by the summary can also be converted to a data.frame.
diag	Passed to lower.tri if to.data.frame=TRUE.
sort.by	character. If summary returns a data.frame, it can be sorted by this column name using order. Note that if discFUN returns an asymmetric 2-dimensional matrix, each data.frame in the returned list will be sorted independently, so the rows are unlikely to be consistent across summary statistics.
decreasing	Passed to order if !is.null(sort.by).
...	Additional graphical parameters to be passed to plot.default.
printLegend	logical. If TRUE (default), a legend will be printed with the histogram
legendArgs	list of arguments passed to the legend function. The default argument is a list placing the legend at the top-left of the figure.
densityArgs	list of arguments passed to the density function, used to obtain densities for the hist method.

Value

An S4 object of class blavPPMC consisting of 5 list slots:

@discFUN	The user-supplied discFUN, or the call to fitMeasures that returns fit.measures.
@dims	The dimensions of the object returned by each discFUN.
@PPP	The posterior predictive *p* value for each discFUN element.
@obsDist	The posterior distribution of realize values of discFUN applied to observed data.
@simDist	The posterior predictive distribution of values of discFUN applied to data simulated from the posterior samples.

The summary() method returns a numeric vector if discFUN returns a scalar, a data.frame with one discrepancy function per row if discFUN returns a numeric vector, and a list with one summary statistic per element if discFUN returns a matrix or multidimensional array.

The plot and pairs methods invisibly return NULL, printing a plot (or scatterplot matrix) to the current device.

The hist method invisibly returns a list or arguments that can be passed to the function for which the list element is named. Users can edit the arguments in the list to customize their histograms.

Author(s)

Terrence D. Jorgensen (University of Amsterdam; <TJorgensen314@gmail.com>)

References

Levy, R. (2011). Bayesian data–model fit assessment for structural equation modeling. *Structural Equation Modeling, 18*(4), 663–685. doi:10.1080/10705511.2011.607723

Examples

```
 ## Not run:
data(HolzingerSwineford1939, package = "lavaan")

HS.model <- ' visual  =~ x1 + x2 + x3
              textual =~ x4 + x5 + x6
              speed   =~ x7 + x8 + x9 '
## fit single-group model
fit <- bcfa(HS.model, data = HolzingerSwineford1939,
            n.chains = 2, burnin = 1000, sample = 500)
## fit multigroup model
fitg <- bcfa(HS.model, data = HolzingerSwineford1939,
             n.chains = 2, burnin = 1000, sample = 500, group = "school")


## Use fit.measures as a shortcut for global fitMeasures only
## - Note that indices calculated from the "df" are only appropriate under
##   noninformative priors, such that pD approximates the number of estimated
##   parameters counted under ML estimation; incremental fit indices
##   introduce further complications)

AFIs <- ppmc(fit, thin = 10, fit.measures = c("srmr","chisq","rmsea","cfi"))
summary(AFIs)                 # summarize the whole vector in a data.frame
hist(AFIs, element = "rmsea") # only plot one discrepancy function at a time
plot(AFIs, element = "srmr")


## define a list of custom discrepancy functions
## - (global) fit measures
## - (local) standardized residuals

discFUN <- list(global = function(fit) {
                  fitMeasures(fit, fit.measures = c("cfi","rmsea","srmr","chisq"))
                },
                std.cov.resid = function(fit) lavResiduals(fit, zstat = FALSE,
                                                           summary = FALSE)$cov,
                std.mean.resid = function(fit) lavResiduals(fit, zstat = FALSE,
                                                            summary = FALSE)$mean)
out1g <- ppmc(fit, discFUN = discFUN)

## summarize first discrepancy by default (fit indices)
summary(out1g)
## some model-implied correlations look systematically over/underestimated
```

```
summary(out1g, discFUN = "std.cov.resid", central.tendency = "EAP")
hist(out1g, discFUN = "std.cov.resid", element = c(1, 7))
plot(out1g, discFUN = "std.cov.resid", element = c("x1","x7"))
## For ease of investigation, optionally export summary as a data.frame,
## sorted by size of average residual
summary(out1g, discFUN = "std.cov.resid", central.tendency = "EAP",
        to.data.frame = TRUE, sort.by = "EAP")
## or sorted by size of PPP
summary(out1g, discFUN = "std.cov.resid", central.tendency = "EAP",
        to.data.frame = TRUE, sort.by = "PPP_sim_LessThan_obs")

## define a list of custom discrepancy functions for multiple groups
## (return each group's numeric output using a different function)

disc2g <- list(global = function(fit) {
                 fitMeasures(fit, fit.measures = c("cfi","rmsea","mfi","srmr","chisq"))
               },
               cor.resid1 = function(fit) lavResiduals(fit, zstat = FALSE,
                                                       type = "cor.bollen",
                                                       summary = FALSE)[[1]]$cov,
               cor.resid2 = function(fit) lavResiduals(fit, zstat = FALSE,
                                                       type = "cor.bollen",
                                                       summary = FALSE)[[2]]$cov)
out2g <- ppmc(fitg, discFUN = disc2g, thin = 2)
## some residuals look like a bigger problem in one group than another
pairs(out2g, discFUN = "cor.resid1", horInd = 1:3, verInd = 7:9) # group 1
pairs(out2g, discFUN = "cor.resid2", horInd = 1:3, verInd = 7:9) # group 2

## print all to file: must be a LARGE picture. First group 1 ...
png("cor.resid1.png", width = 1600, height = 1200)
pairs(out2g, discFUN = "cor.resid1")
dev.off()
## ... then group 2
png("cor.resid2.png", width = 1600, height = 1200)
pairs(out2g, discFUN = "cor.resid2")
dev.off()

## End(Not run)
```

---

ppp *Posterior Predictive P-value*

Description

Obtain the posterior predictive p-value (ppp) for a fitted blavaan object, computing it on demand if it was not already computed during model estimation.

Usage

```
ppp(object, ...)
```

Arguments

`object`	An object of class `blavaan`.
`...`	Other arguments passed to the underlying computation for single-level models with ordinal variables, or for two-level (multilevel) models (currently only `thin` and `parallel` have an effect for either case, plus `em_control` for two-level models; see Details).

Details

For most models, `fitMeasures(object, "ppp")` already returns the ppp computed during estimation, and `ppp(object)` simply returns that same value. The exceptions are single-level models (with a single group or multiple groups) that include ordinal variables, and two-level (multilevel) models, both fit with `target = "stan"` or `"cmdstan"`: for these models, `test` defaults to `"none"` and ppp is *not* computed automatically, because the computation takes noticeably longer than for other models. Calling `ppp(object)` on such a fit computes it as a separate step. To have ppp computed automatically during estimation instead, pass `test = "standard"` (or, for two-level models, any value other than `"none"`) to `bcfa()`/`bsem()`/`bgrowth()`/ `blavaan()`.

For single-level ordinal/mixed models, the on-demand computation is a limited-information, pairwise comparison of observed vs model-implied associations, which better reflects fit for ordinal data than the previous Stan-computed default.

For two-level (multilevel) models, the on-demand computation is a saturated-model likelihood-ratio comparison (the same statistic previously computed inside the compiled Stan program), now computed in R by calling **lavaan**'s own saturated-model EM algorithm once per retained posterior draw. Because this per-draw computation is comparatively slow, increasing `thin` and/or setting `parallel = TRUE` (together with `future::plan("multicore")` or `future::plan("multisession")`) is recommended for larger two-level fits; an `em_control` argument (a list with `tol`, `max_iter`, and `acceleration` elements) is also available to tune the per-draw EM refit's convergence settings. `fixed.x` variables are supported at the within level, the between level, or both (in different variables): they are held fixed at their observed values in every simulated replicate rather than being resimulated, matching how fixed.x is already handled for single-level on-demand computations. A variable that is `fixed.x` at *both* levels simultaneously is not supported (a pre-existing **lavaan** limitation, not specific to this computation) and will raise an error.

Value

A numeric ppp value (or vector, for multiple imputations/groups, matching whatever `fitMeasures(object, "ppp")` would otherwise return).

See Also

`ppmc`, for more flexible posterior predictive model checks using arbitrary discrepancy functions.

Examples

```
## Not run:
data(HolzingerSwineford1939, package = "lavaan")

HS.model <- ' visual  =~ x1 + x2 + x3
```

```
              textual =~ x4 + x5 + x6
              speed   =~ x7 + x8 + x9 '

## make some indicators ordinal
HSord <- HolzingerSwineford1939
HSord$x1 <- as.integer(cut(HSord$x1, 3))
HSord$x2 <- as.integer(cut(HSord$x2, 3))

## ppp is not computed by default for this ordinal model
fit <- bcfa(HS.model, data = HSord, ordered = c("x1", "x2"))

## compute it on demand
ppp(fit)

## End(Not run)
```

`sampleData` *Sample data from the posterior (or prior) distribution.*

Description

The purpose of the `sampleData()` function is to simulate new data from a model that has already been estimated. This can faciliate posterior predictive checks, as well as prior predictive checks (setting prisamp = TRUE during model estimation).

Usage

```
sampleData(object, nrep = NULL, conditional = FALSE, type = "response",
           simplify = FALSE, ...)
```

Arguments

`object`	An object of class `blavaan`.
`nrep`	How many datasets to generate? If not supplied, defaults to the total number of posterior samples.
`conditional`	Logical indicating whether to sample from the distribution that is marginal over latent variables (`FALSE`; default) or from the distribution that conditions on latent variables (`TRUE`). For `TRUE`, you must set `save.lvs = TRUE` during model estimation.
`type`	The type of data desired (only relevant to ordinal data). The `type = "response"` option generates ordinal data. The `type = "link"` option generates continuous variables underlying ordinal data (which would be cut by thresholds to yield ordinal data).
`simplify`	For single-group models, should the list structure be simplified? This makes each dataset a single list entry, instead of a list within a list (which reflects group 1 of dataset 1). Defaults to `FALSE`.
`...`	Other arguments, which for now is only `parallel`. Parallelization via `future_lapply()` is available by setting `parallel = TRUE`.

Details

This is a convenience function to generate data for posterior or prior predictive checking. The underlying code is also used to generate data for posterior predictive p-value computation.

See Also

This function overlaps with `blavPredict()`. The `blavPredict()` function is more focused on generating pieces of data conditioned on other pieces of observed data (i.e., latent variables conditioned on observed variables; missing variables conditioned on observed variables). In contrast, the `sampleData()` function is more focused on generating new data given the sampled model parameters.

Examples

```
## Not run:
data(HolzingerSwineford1939, package = "lavaan")

## fit model
HS.model <- ' visual  =~ x1 + x2 + x3
              textual =~ x4 + x5 + x6
              speed   =~ x7 + x8 + x9 '

fit <- bcfa(HS.model, data = HolzingerSwineford1939)

## 1 dataset generated from the posterior
out <- sampleData(fit, nrep = 1)

## nested lists: 1 list entry per nrep.
## then, within a rep, 1 list entry per group
## so our dataset is here:
dim(out[[1]][[1]])

## 1 posterior dataset per posterior sample:
out <- sampleData(fit)

## obtain the data on x1 across reps and summarize:
x1dat <- sapply(out, function(x) x[[1]][,1])
summary( as.numeric(x1dat) )

## End(Not run)
```

---

standardizedPosterior *Standardized Posterior*

Description

Standardized posterior distribution of a latent variable model.

Usage

```
standardizedPosterior(object, ...)
```

Arguments

`object`	An object of class `blavaan`.
`...`	Additional arguments passed to lavaan's `standardizedSolution()`

Value

A matrix containing standardized posterior draws, where rows are draws and columns are parameters.

Note

The only allowed `standardizedSolution()` arguments are type, cov.std, remove.eq, remove.ineq, and remove.def. Other arguments are not immediately suited to posterior distributions.

Examples

```
## Not run:
data(PoliticalDemocracy, package = "lavaan")

model <- '
  # latent variable definitions
     ind60 =~ x1 + x2 + x3
     dem60 =~ y1 + a*y2 + b*y3 + c*y4
     dem65 =~ y5 + a*y6 + b*y7 + c*y8

  # regressions
    dem60 ~ ind60
    dem65 ~ ind60 + dem60

  # residual correlations
    y1 ~~ y5
    y2 ~~ y4 + y6
    y3 ~~ y7
    y4 ~~ y8
    y6 ~~ y8
'

fit <- bsem(model, data = PoliticalDemocracy,
            dp = dpriors(nu = "normal(5, 10)"))

standardizedPosterior(fit)

## End(Not run)
```

`tidy.blavaan` *Tidy a blavaan object*

Description

This function repackages the output from `summary()` into a friendly `data.frame`.

Usage

```
## S3 method for class 'blavaan'
tidy(
  x,
  estimate.method = c("mean", "median", "mode"),
  conf.int = TRUE,
  level = 0.95,
  hpd = FALSE,
  rhat = TRUE,
  ess = TRUE,
  priors = TRUE,
  ...
)
```

Arguments

`x`	A `blavaan` object.
`estimate.method`	The posterior summary statistic to compute the point estimates. One of `"mean"` (posterior mean, default), `"median"` (posterior median), or `"mode"` (posterior mode). Only free parameters are summarized by the requested statistic; fixed parameters retain their fixed values. Defined/constrained parameters (`:=`) keep the point estimate from the standard `summary()` output regardless of `estimate.method`.
`conf.int`	Logical indicating whether to include credible intervals. Default is `TRUE`.
`level`	The credible-interval mass to use when `conf.int = TRUE`, e.g. `.90` for a 90% interval. Default is `.95`. Computed from the MCMC draws (via `blavInspect`) for both free and defined (`:=`) parameters; only applies for `target = "stan"`/`"cmdstan"` fits.
`hpd`	Logical. If `TRUE`, the credible interval is a highest-posterior-density interval instead of the default equal-tailed quantile interval. Default is `FALSE`.
`rhat`	Logical indicating whether to include the Rhat convergence diagnostic. Default is `TRUE`.
`ess`	Logical indicating whether to include the effective sample size. Default is `TRUE`.
`priors`	Logical indicating whether to include the prior distribution specification. Default is `TRUE`.
`...`	Additional arguments (currently ignored).

Value

A `data.frame` with columns:

`term`	Parameter name (lhs, op, rhs combined)
`op`	Operator from the model syntax
`level`	The level of a parameter estimate
`group`	Group number (for multigroup models)
`estimate`	Posterior summary statistic determined by `estimate.method`
`std.error`	Posterior standard deviation
`conf.low`	Lower bound of the credible interval, width set by `level` (if `conf.int = TRUE`)
`conf.high`	Upper bound of the credible interval, width set by `level` (if `conf.int = TRUE`)
`std.lv`	Standardized estimates based on the variances of the (continuous) latent variables only
`std.all`	Standardized estimates based on both the variances of both (continuous) observed and latent variables.
`rhat`	Rhat convergence diagnostic (if `rhat = TRUE`)
`ess`	Effective sample size (if `ess = TRUE`)
`prior`	Prior distribution specification (if `priors = TRUE`)

Examples

```
## Not run:
data(HolzingerSwineford1939, package = "lavaan")

HS.model <- 'visual =~ x1 + x2 + x3'
fit <- bcfa(HS.model, data = HolzingerSwineford1939, seed = 123,
            n.chains = 1, sample = 300)
tidy(fit)
tidy(fit, estimate.method = "median")

data(Demo.twolevel, package = "lavaan")
model <- "
    level: within
        fw =~ y1 + y2 + y3
        fw ~ x1 + x2 + x3
    level: between
        fb =~ y1 + y2 + y3
        fb ~ w1 + w2
"
bfit <- bsem(
  model = model,
  data = Demo.twolevel,
  cluster = "cluster",
  seed = 123,
  n.chains = 1,
  sample = 300,
  target = "stan"
```

```
)
tidy(fit)

## End(Not run)
```

Index

bcfa, 3, *11–13*, *20*, *26*
BF *(*blavCompare*)*, 14
bgrowth, 6, *11–13*, *20*, *26*
blav_internal, 9
blav_model_test *(*blav_internal*)*, 9
blavaan, *5*, *8*, 9, *12*, *15, 16*, *20*, *24*, *29*, *32–34*, *36*
blavaan-class, 12
blavCompare, 14
blavFitIndices, 15
blavFitIndices-class *(*blavFitIndices*)*, 15
blavInspect, 18, *37*
blavPPMC-class *(*ppmc*)*, 28
blavPredict, 20
blavpredict *(*blavPredict*)*, 20
blavTech *(*blavInspect*)*, 18
bsem, *11–13*, *20*, 22, *26*

coef,blavaan-method *(*blavaan-class*)*, 12
coeffun *(*blav_internal*)*, 9

density, *30*
dpriors, 25

fitMeasures, *16*, *29, 30*

glance.blavaan, 26

hist.blavPPMC *(*ppmc*)*, 28

labelfun *(*blav_internal*)*, 9
lavaan, *3–11*, *16*, *19*, *22–24*, *28, 29*
lavInspect, *20*
legend, *30*
lower.tri, *30*

MCMC, *28*
mode, *29*

order, *30*

pairs.blavPPMC *(*ppmc*)*, 28
par, *30*
plot.blavaan, 27
plot.blavPPMC *(*ppmc*)*, 28
plot.default, *30*
ppmc, 28, *33*
ppp, 32
predict,blavaan-method *(*blavaan-class*)*, 12

sampleData, 34
sampledata *(*sampleData*)*, 34
set_inits *(*blav_internal*)*, 9
set_phantoms *(*blav_internal*)*, 9
set_priors *(*blav_internal*)*, 9
show,blavaan-method *(*blavaan-class*)*, 12
show,blavFitIndices-method *(*blavFitIndices*)*, 15
show,blavPPMC-method *(*ppmc*)*, 28
standardizedPosterior, 35
standardizedposterior *(*standardizedPosterior*)*, 35
standardizedSolution, *13*
summary,blavaan-method *(*blavaan-class*)*, 12
summary,blavFitIndices-method *(*blavFitIndices*)*, 15
summary,blavPPMC-method *(*ppmc*)*, 28
summary.bfi *(*blavFitIndices*)*, 15
summary.ppmc *(*ppmc*)*, 28

tidy.blavaan, 37