Package 'INLAvaan'

October 26, 2026

Type Package

Title Approximate Bayesian Latent Variable Analysis

Version 0.3.3

Description Implements approximate Bayesian inference for Structural Equation Models (SEM) using a custom adaptation of the Integrated Nested Laplace Approximation (Rue et al., 2009) <doi:10.1111/j.1467-9868.2008.00700.x> as described in Jamil and Rue (2026a) <doi:10.48550/arXiv.2603.25690>. Provides a computationally efficient alternative to Markov Chain Monte Carlo (MCMC) for Bayesian estimation, allowing users to fit latent variable models using the 'lavaan' syntax. See also the companion paper on implementation and workflows, Jamil and Rue (2026b) <doi:10.48550/arXiv.2604.00671>.

License GPL (>= 3)

URL https://inlavaan.haziqj.ml/, https://github.com/haziqj/INLAvaan

BugReports https://github.com/haziqj/INLAvaan/issues

Depends R (>= 3.5)

Imports cli, graphics, lavaan (>= 0.7-2), methods, parallel, stats, utils

Suggests blavaan, ggplot2, knitr, lme4, numDeriv, qrng, quarto, sn, testthat (>= 3.2.0), ucminf

VignetteBuilder quarto

Config/Needs/website rmarkdown

Config/roxygen2/markdown TRUE

Config/roxygen2/version 8.1.0

Config/testthat/edition 3

Encoding UTF-8

NeedsCompilation yes

Author Haziq Jamil [aut, cre, cph] (ORCID: <https://orcid.org/0000-0003-3298-1010>), Håvard Rue [ctb] (ORCID: <https://orcid.org/0000-0002-0222-1881>,

Statistical and computational methodology),
Alvin Bong [ctb] (Initial site build)

Maintainer Haziq Jamil <haziq.jamil@gmail.com>

Repository CRAN

Date/Publication 2026-10-26 17:30:08 UTC

Contents

acfa . . . 3
agrowth . . . 7
asem . . . 12
as_fun_string . . . 16
bfit_indices . . . 17
compare . . . 19
dbeta_box . . . 21
deviance . . . 22
diagnostics . . . 23
dsnorm . . . 26
fitmeasures . . . 27
fitted . . . 29
fit_skew_normal . . . 30
fit_skew_normal_samp . . . 32
get_inlavaan_internal . . . 33
inlavaan . . . 34
INLAvaan-class . . . 40
is_same_function . . . 42
logLik . . . 42
loo . . . 44
plot . . . 48
predict . . . 49
priors_for . . . 51
qsnorm_fast . . . 53
residuals . . . 54
sampling . . . 55
simulate . . . 58
standardisedsolution . . . 60
timing . . . 63
update . . . 64
vcov . . . 65
waic . . . 66

Index **70**

acfa *Fit an Approximate Bayesian Confirmatory Factor Analysis Model*

Description

Fit an Approximate Bayesian Confirmatory Factor Analysis Model

Usage

```
acfa(
  model,
  data,
  dp = priors_for(),
  test = "standard",
  vb_correction = TRUE,
  n_qmc = 64L,
  vb_method = c("sobol", "gauss_hermite"),
  marginal_method = c("skewnorm", "asymgaus", "marggaus", "sampling"),
  marginal_correction = c("shortcut", "shortcut_fd", "hessian", "none"),
  nsamp = 1000,
  samp_copula = TRUE,
  samp_norta = FALSE,
  cov_as_cor = FALSE,
  sn_fit_ngrid = 21,
  sn_fit_logthresh = -6,
  sn_fit_temp = 1,
  sn_fit_sample = TRUE,
  control = list(),
  verbose = TRUE,
  debug = FALSE,
  add_priors = TRUE,
  optim_method = c("nlminb", "ucminf", "optim"),
  numerical_grad = FALSE,
  cores = NULL,
  ppp_method = c("onestep", "em"),
  ppp_nsamp = 250L,
  ...
)
```

Arguments

model	A description of the user-specified model. Typically, the model is described using the lavaan model syntax. See model.syntax for more information. Alternatively, a parameter table (e.g., the output of the lavParTable() function) is also accepted.

data — An optional data frame containing the observed variables used in the model. If some variables are declared as ordered factors, lavaan will treat them as ordinal variables.

dp — Default prior distributions for the different types of model parameters; a named character vector as returned by priors_for(). Types left out take their default priors.

test — Character vector naming the post-estimation quantities to compute and store with the fit. The atoms are "ppp" (posterior predictive p-value), "dic" (deviance information criterion and its pD), "loo" (leave-one-out cross-validation, see loo()) and "waic" (see waic()). Three aliases stand for sets of atoms: "standard" (the default) and its synonym "default" give c("ppp", "dic"); "full" gives all four; "none" gives nothing. Aliases and atoms may be mixed and are unioned, so test = c("standard", "loo") adds the LOO to the default set. The LOO and the WAIC come from one Taylor pass, so asking for either stores both. They run only when asked for, with no time budget. On a model the casewise machinery does not support (PML or ordinal data, conditional.x = TRUE, multigroup two-level) they are skipped with a warning and the rest of the fit proceeds. For a two-level model the PPP follows blavaan: each of ppp_nsamp posterior draws generates replicate data, which are scored against the saturated model (see ppp_method). The two-level PPP is experimental. The fit records what was requested and what was computed (get_inlavaan_internal(fit, "test")); summary(), fitmeasures(), deviance(), logLik() and timing() report only what was computed. add_loo() stores the LOO and WAIC post hoc; loo() and waic() compute on demand. For a random-slope model "ppp" is dropped, with a warning when it was asked for and a message otherwise (see the Random slopes section of inlavaan()). "dic", "loo" and "waic" are unaffected.

vb_correction — Logical indicating whether to apply a variational Bayes correction for the posterior mean vector of estimates. Defaults to TRUE.

n_qmc — Number of quasi-Monte Carlo nodes used by the VB mean correction. Defaults to 64; see the Details section of inlavaan(). Values above 128 (the size of the stored Sobol table) require the **qrng** package. Ignored when vb_correction = FALSE or vb_method = "gauss_hermite".

vb_method — Integration rule for the VB mean correction. "sobol" (default) averages over n_qmc scrambled Sobol nodes. "gauss_hermite" uses a deterministic rule instead: a three-point Gauss-Hermite rule along each principal axis of the Laplace covariance, 2m + 1 nodes in all for m free parameters. It is exact whenever the log-posterior is quartic in whitened coordinates, and it gives the same shift on every run. Having no node sets to compare, it reports no quadrature error, so vb_mcse_sigma in diagnostics() is NA. Its cost grows with m: it is cheaper than the default below about 30 free parameters and dearer above. Experimental.

marginal_method — The method for approximating the marginal posterior distributions. Options include "skewnorm" (skew-normal), "asymgaus" (two-piece asymmetric Gaussian), "marggaus" (marginalising the Laplace approximation), and "sampling" (sampling from the joint Laplace approximation).

`marginal_correction`
: Which type of correction to use when fitting the skew-normal or two-piece Gaussian marginals. `"shortcut"` (default) computes only diagonals via central differences (full z-trace plus Schur complement correction), `"shortcut_fd"` is the same formula using forward differences (roughly half the cost, less accurate), `"hessian"` computes the full Hessian-based correction (slow), and `"none"` (or `FALSE`) applies no correction.

`nsamp`
: The number of samples to draw for all sampling-based approaches (including posterior sampling for model fit indices).

`samp_copula`
: Logical. When `TRUE` (default), posterior samples are drawn using the copula method with the fitted marginals (e.g. skew-normal or asymmetric Gaussian). When `FALSE`, samples are drawn from the Gaussian (Laplace) approximation.

`samp_norta`
: Logical. When `TRUE`, the latent correlation matrix of the skew-normal copula is adjusted by the NORmal-To-Anything (NORTA) scheme of Cario and Nelson (1997) so that the Pearson correlations of the copula draws match those of the Laplace approximation after the nonlinear quantile transform. The adjustment never changes a marginal; it affects only summaries that involve several parameters at once, and in practice moves the correlations very little. Default `FALSE`. Only used when `samp_copula = TRUE` and `marginal_method = "skewnorm"`.

`cov_as_cor`
: Logical. Residual and latent-disturbance covariance parameters (`~~` between two observed or two latent variables) are always estimated on the correlation scale internally (an `atanh` link, the same as for `std.ov`/`std.lv`-standardised parameters); by default their reported marginal is then re-derived on the covariance scale $\sigma_i \sigma_j \rho$ from a posterior sample (see `samp_copula`), because that is the scale lavaan/blavaan report by default. When `TRUE`, that re-derivation is skipped and each such parameter's own directly profiled correlation-scale marginal $\rho \in (-1, 1)$ is reported instead – useful for comparing the profiling machinery (skew-normal fit, VB, ...) against a correlation-scale reference without the sampling/copula step in between. Model estimation is identical either way; only what is reported for these parameters changes (and, correspondingly, their `mat` classification in the returned partable, `theta_cov`/`psi_cov` vs. `theta_cor`/`psi_cor`). Not the same as lavaan's `std.ov`/`std.lv`, which re-parameterises the whole model on a standardised scale. Defaults to `FALSE`.

`sn_fit_ngrid`
: Number of grid points to lay out per dimension when fitting the skew-normal marginals. A finer grid gives a better fit at the cost of more joint-log-posterior evaluations. Defaults to `21`.

`sn_fit_logthresh`
: The log-threshold for fitting the skew-normal. Points with log-posterior drop below this threshold (relative to the maximum) will be excluded from the fit. Defaults to `-6`.

`sn_fit_temp`
: Temperature parameter for fitting the skew-normal. Defaults to `1` (weights are the density values themselves). If `NA`, the temperature is included as an additional optimisation parameter.

`sn_fit_sample`
: Logical. When `TRUE` (default), a parametric skew-normal is fitted to the posterior samples for covariance and defined parameters. When `FALSE`, these are summarised using kernel density estimation instead.

control	A list of control parameters for the optimiser. For the default "nlminb", INLAvaan raises the stock iteration ceilings to iter.max = 1000 and eval.max = 2000 (complex models can exhaust nlminb()'s own defaults of 150 and 200); any value supplied here overrides these.
verbose	Logical indicating whether to print progress messages.
debug	Logical indicating whether to return debug information.
add_priors	Logical indicating whether to include prior densities in the posterior computation.
optim_method	The optimisation method to use for finding the posterior mode. Options include "nlminb" (default), "ucminf", and "optim" (BFGS).
numerical_grad	Logical indicating whether to use numerical gradients for the optimisation. Defaults to FALSE to use analytical gradients.
cores	Integer or NULL. Number of cores for parallel marginal fitting. When NULL (default), serial execution is used unless the number of free parameters exceeds 120, in which case parallelisation is enabled automatically using all available physical cores. Set to 1L to force serial execution. If cores > 1, marginal fits are distributed across cores – forked via parallel::mclapply() where that is safe, or over a PSOCK cluster (separate R processes) inside IDE R sessions (RStudio, Positron) and on Windows.
ppp_method	How the PPP of a two-level model scores the observed and the replicate data against the saturated model. "onestep" (default) takes one Fisher-scoring step from the moments of each posterior draw towards the saturated fit, and fits by EM where the step would leave the valid covariance matrices (with few clusters or a small between variance). "em" always fits the saturated model by EM, as blavaan does. Fits with missing = "ml" always use "em". Ignored for single-level models.
ppp_nsamp	The number of posterior draws, each with one replicate data set, that the PPP of a two-level model uses. Defaults to 250, and is capped at nsamp. Draws that cannot be scored are left out, with a warning. Ignored for single-level models.
...	Additional arguments to be passed to the lavaan model fitting function.

Details

The acfa() function is a wrapper for the more general inlavaan() function, using the following default 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.efa = TRUE
- auto.th = TRUE

- `auto.delta = TRUE`
- `auto.cov.y = TRUE`

For further information regarding these arguments, please refer to the `lavaan::lavOptions()` documentation.

Value

An S4 object of class `INLAvaan` which is a subclass of the lavaan class.

See Also

Typically, users will interact with the specific latent variable model functions instead, including `acfa()`, `asem()`, and `agrowth()`.

Examples

```
# The famous Holzinger and Swineford (1939) example
HS.model <- "
  visual  =~ x1 + x2 + x3
  textual =~ x4 + x5 + x6
  speed   =~ x7 + x8 + x9
"
utils::data("HolzingerSwineford1939", package = "lavaan")

# Fit a CFA model with standardised latent variables
fit <- acfa(HS.model, data = HolzingerSwineford1939, std.lv = TRUE, nsamp = 100)
summary(fit)
```

---

agrowth *Fit an Approximate Bayesian Growth Curve Model*

Description

Fit an Approximate Bayesian Growth Curve Model

Usage

```
agrowth(
  model,
  data,
  dp = priors_for(),
  test = "standard",
  vb_correction = TRUE,
  n_qmc = 64L,
  vb_method = c("sobol", "gauss_hermite"),
  marginal_method = c("skewnorm", "asymgaus", "marggaus", "sampling"),
  marginal_correction = c("shortcut", "shortcut_fd", "hessian", "none"),
  nsamp = 1000,
```

```
  samp_copula = TRUE,
  samp_norta = FALSE,
  cov_as_cor = FALSE,
  sn_fit_ngrid = 21,
  sn_fit_logthresh = -6,
  sn_fit_temp = 1,
  sn_fit_sample = TRUE,
  control = list(),
  verbose = TRUE,
  debug = FALSE,
  add_priors = TRUE,
  optim_method = c("nlminb", "ucminf", "optim"),
  numerical_grad = FALSE,
  cores = NULL,
  ppp_method = c("onestep", "em"),
  ppp_nsamp = 250L,
  ...
)
```

Arguments

model	A description of the user-specified model. Typically, the model is described using the lavaan model syntax. See model.syntax for more information. Alternatively, a parameter table (e.g., the output of the lavParTable() function) is also accepted.
data	An optional data frame containing the observed variables used in the model. If some variables are declared as ordered factors, lavaan will treat them as ordinal variables.
dp	Default prior distributions for the different types of model parameters; a named character vector as returned by priors_for(). Types left out take their default priors.
test	Character vector naming the post-estimation quantities to compute and store with the fit. The atoms are "ppp" (posterior predictive p-value), "dic" (deviance information criterion and its pD), "loo" (leave-one-out cross-validation, see loo()) and "waic" (see waic()). Three aliases stand for sets of atoms: "standard" (the default) and its synonym "default" give c("ppp", "dic"); "full" gives all four; "none" gives nothing. Aliases and atoms may be mixed and are unioned, so test = c("standard", "loo") adds the LOO to the default set. The LOO and the WAIC come from one Taylor pass, so asking for either stores both. They run only when asked for, with no time budget. On a model the casewise machinery does not support (PML or ordinal data, conditional.x = TRUE, multigroup two-level) they are skipped with a warning and the rest of the fit proceeds. For a two-level model the PPP follows blavaan: each of ppp_nsamp posterior draws generates replicate data, which are scored against the saturated model (see ppp_method). The two-level PPP is experimental. The fit records what was requested and what was computed (get_inlavaan_internal(fit, "test")); summary(), fitmeasures(), deviance(), logLik() and timing() report only what was computed. add_loo() stores the LOO and WAIC post

hoc; loo() and waic() compute on demand. For a random-slope model "ppp" is dropped, with a warning when it was asked for and a message otherwise (see the Random slopes section of inlavaan()). "dic", "loo" and "waic" are unaffected.

vb_correction Logical indicating whether to apply a variational Bayes correction for the posterior mean vector of estimates. Defaults to TRUE.

n_qmc Number of quasi-Monte Carlo nodes used by the VB mean correction. Defaults to 64; see the Details section of inlavaan(). Values above 128 (the size of the stored Sobol table) require the **qrng** package. Ignored when vb_correction = FALSE or vb_method = "gauss_hermite".

vb_method Integration rule for the VB mean correction. "sobol" (default) averages over n_qmc scrambled Sobol nodes. "gauss_hermite" uses a deterministic rule instead: a three-point Gauss-Hermite rule along each principal axis of the Laplace covariance, 2m + 1 nodes in all for m free parameters. It is exact whenever the log-posterior is quartic in whitened coordinates, and it gives the same shift on every run. Having no node sets to compare, it reports no quadrature error, so vb_mcse_sigma in diagnostics() is NA. Its cost grows with m: it is cheaper than the default below about 30 free parameters and dearer above. Experimental.

marginal_method
The method for approximating the marginal posterior distributions. Options include "skewnorm" (skew-normal), "asymgaus" (two-piece asymmetric Gaussian), "marggaus" (marginalising the Laplace approximation), and "sampling" (sampling from the joint Laplace approximation).

marginal_correction
Which type of correction to use when fitting the skew-normal or two-piece Gaussian marginals. "shortcut" (default) computes only diagonals via central differences (full z-trace plus Schur complement correction), "shortcut_fd" is the same formula using forward differences (roughly half the cost, less accurate), "hessian" computes the full Hessian-based correction (slow), and "none" (or FALSE) applies no correction.

nsamp The number of samples to draw for all sampling-based approaches (including posterior sampling for model fit indices).

samp_copula Logical. When TRUE (default), posterior samples are drawn using the copula method with the fitted marginals (e.g. skew-normal or asymmetric Gaussian). When FALSE, samples are drawn from the Gaussian (Laplace) approximation.

samp_norta Logical. When TRUE, the latent correlation matrix of the skew-normal copula is adjusted by the NORmal-To-Anything (NORTA) scheme of Cario and Nelson (1997) so that the Pearson correlations of the copula draws match those of the Laplace approximation after the nonlinear quantile transform. The adjustment never changes a marginal; it affects only summaries that involve several parameters at once, and in practice moves the correlations very little. Default FALSE. Only used when samp_copula = TRUE and marginal_method = "skewnorm".

cov_as_cor Logical. Residual and latent-disturbance covariance parameters (~~ between two observed or two latent variables) are always estimated on the correlation scale internally (an atanh link, the same as for std.ov/std.lv-standardised parameters); by default their reported marginal is then re-derived on the covariance scale $\sigma_i \sigma_j \rho$ from a posterior sample (see samp_copula), because that

is the scale lavaan/blavaan report by default. When TRUE, that re-derivation is skipped and each such parameter's own directly profiled correlation-scale marginal $\rho \in (-1, 1)$ is reported instead – useful for comparing the profiling machinery (skew-normal fit, VB, ...) against a correlation-scale reference without the sampling/copula step in between. Model estimation is identical either way; only what is reported for these parameters changes (and, correspondingly, their mat classification in the returned partable, theta_cov/psi_cov vs. theta_cor/psi_cor). Not the same as lavaan's std.ov/std.lv, which re-parameterises the whole model on a standardised scale. Defaults to FALSE.

sn_fit_ngrid
: Number of grid points to lay out per dimension when fitting the skew-normal marginals. A finer grid gives a better fit at the cost of more joint-log-posterior evaluations. Defaults to 21.

sn_fit_logthresh
: The log-threshold for fitting the skew-normal. Points with log-posterior drop below this threshold (relative to the maximum) will be excluded from the fit. Defaults to -6.

sn_fit_temp
: Temperature parameter for fitting the skew-normal. Defaults to 1 (weights are the density values themselves). If NA, the temperature is included as an additional optimisation parameter.

sn_fit_sample
: Logical. When TRUE (default), a parametric skew-normal is fitted to the posterior samples for covariance and defined parameters. When FALSE, these are summarised using kernel density estimation instead.

control
: A list of control parameters for the optimiser. For the default "nlminb", INLAvaan raises the stock iteration ceilings to iter.max = 1000 and eval.max = 2000 (complex models can exhaust nlminb()'s own defaults of 150 and 200); any value supplied here overrides these.

verbose
: Logical indicating whether to print progress messages.

debug
: Logical indicating whether to return debug information.

add_priors
: Logical indicating whether to include prior densities in the posterior computation.

optim_method
: The optimisation method to use for finding the posterior mode. Options include "nlminb" (default), "ucminf", and "optim" (BFGS).

numerical_grad
: Logical indicating whether to use numerical gradients for the optimisation. Defaults to FALSE to use analytical gradients.

cores
: Integer or NULL. Number of cores for parallel marginal fitting. When NULL (default), serial execution is used unless the number of free parameters exceeds 120, in which case parallelisation is enabled automatically using all available physical cores. Set to 1L to force serial execution. If cores > 1, marginal fits are distributed across cores – forked via parallel::mclapply() where that is safe, or over a PSOCK cluster (separate R processes) inside IDE R sessions (RStudio, Positron) and on Windows.

ppp_method
: How the PPP of a two-level model scores the observed and the replicate data against the saturated model. "onestep" (default) takes one Fisher-scoring step from the moments of each posterior draw towards the saturated fit, and fits by EM where the step would leave the valid covariance matrices (with few clusters

	or a small between variance). "em" always fits the saturated model by EM, as blavaan does. Fits with missing = "ml" always use "em". Ignored for single-level models.
ppp_nsamp	The number of posterior draws, each with one replicate data set, that the PPP of a two-level model uses. Defaults to 250, and is capped at nsamp. Draws that cannot be scored are left out, with a warning. Ignored for single-level models.
...	Additional arguments to be passed to the lavaan model fitting function.

Details

The asem() function is a wrapper for the more general inlavaan() function, using the following default 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.efa = TRUE
- auto.th = TRUE
- auto.delta = TRUE
- auto.cov.y = TRUE

Value

An S4 object of class INLAvaan which is a subclass of the lavaan class.

See Also

Typically, users will interact with the specific latent variable model functions instead, including acfa(), asem(), and agrowth().

Examples

```
# Linear growth model with a time-varying covariate
mod <- "
  # Intercept and slope with fixed coefficients
    i =~ 1*t1 + 1*t2 + 1*t3 + 1*t4
    s =~ 0*t1 + 1*t2 + 2*t3 + 3*t4

  # (Latent) regressions
    i ~ x1 + x2
    s ~ x1 + x2

  # Time-varying covariates
```

```
    t1 ~ c1
    t2 ~ c2
    t3 ~ c3
    t4 ~ c4
"
utils::data("Demo.growth", package = "lavaan")
str(Demo.growth)

fit <- agrowth(mod, data = Demo.growth, nsamp = 100)
summary(fit)
```

---

asem *Fit an Approximate Bayesian Structural Equation Model*

Description

Fit an Approximate Bayesian Structural Equation Model

Usage

```
asem(
  model,
  data,
  dp = priors_for(),
  test = "standard",
  vb_correction = TRUE,
  n_qmc = 64L,
  vb_method = c("sobol", "gauss_hermite"),
  marginal_method = c("skewnorm", "asymgaus", "marggaus", "sampling"),
  marginal_correction = c("shortcut", "shortcut_fd", "hessian", "none"),
  nsamp = 1000,
  samp_copula = TRUE,
  samp_norta = FALSE,
  cov_as_cor = FALSE,
  sn_fit_ngrid = 21,
  sn_fit_logthresh = -6,
  sn_fit_temp = 1,
  sn_fit_sample = TRUE,
  control = list(),
  verbose = TRUE,
  debug = FALSE,
  add_priors = TRUE,
  optim_method = c("nlminb", "ucminf", "optim"),
  numerical_grad = FALSE,
  cores = NULL,
  ppp_method = c("onestep", "em"),
  ppp_nsamp = 250L,
  ...
)
```

Arguments

model	A description of the user-specified model. Typically, the model is described using the lavaan model syntax. See model.syntax for more information. Alternatively, a parameter table (e.g., the output of the lavParTable() function) is also accepted.
data	An optional data frame containing the observed variables used in the model. If some variables are declared as ordered factors, lavaan will treat them as ordinal variables.
dp	Default prior distributions for the different types of model parameters; a named character vector as returned by priors_for(). Types left out take their default priors.
test	Character vector naming the post-estimation quantities to compute and store with the fit. The atoms are "ppp" (posterior predictive p-value), "dic" (deviance information criterion and its pD), "loo" (leave-one-out cross-validation, see loo()) and "waic" (see waic()). Three aliases stand for sets of atoms: "standard" (the default) and its synonym "default" give c("ppp", "dic"); "full" gives all four; "none" gives nothing. Aliases and atoms may be mixed and are unioned, so test = c("standard", "loo") adds the LOO to the default set. The LOO and the WAIC come from one Taylor pass, so asking for either stores both. They run only when asked for, with no time budget. On a model the casewise machinery does not support (PML or ordinal data, conditional.x = TRUE, multigroup two-level) they are skipped with a warning and the rest of the fit proceeds. For a two-level model the PPP follows blavaan: each of ppp_nsamp posterior draws generates replicate data, which are scored against the saturated model (see ppp_method). The two-level PPP is experimental. The fit records what was requested and what was computed (get_inlavaan_internal(fit, "test")); summary(), fitmeasures(), deviance(), logLik() and timing() report only what was computed. add_loo() stores the LOO and WAIC post hoc; loo() and waic() compute on demand. For a random-slope model "ppp" is dropped, with a warning when it was asked for and a message otherwise (see the Random slopes section of inlavaan()). "dic", "loo" and "waic" are unaffected.
vb_correction	Logical indicating whether to apply a variational Bayes correction for the posterior mean vector of estimates. Defaults to TRUE.
n_qmc	Number of quasi-Monte Carlo nodes used by the VB mean correction. Defaults to 64; see the Details section of inlavaan(). Values above 128 (the size of the stored Sobol table) require the **qrng** package. Ignored when vb_correction = FALSE or vb_method = "gauss_hermite".
vb_method	Integration rule for the VB mean correction. "sobol" (default) averages over n_qmc scrambled Sobol nodes. "gauss_hermite" uses a deterministic rule instead: a three-point Gauss-Hermite rule along each principal axis of the Laplace covariance, 2m + 1 nodes in all for m free parameters. It is exact whenever the log-posterior is quartic in whitened coordinates, and it gives the same shift on every run. Having no node sets to compare, it reports no quadrature error, so vb_mcse_sigma in diagnostics() is NA. Its cost grows with m: it is cheaper than the default below about 30 free parameters and dearer above. Experimental.

marginal_method
: The method for approximating the marginal posterior distributions. Options include "skewnorm" (skew-normal), "asymgaus" (two-piece asymmetric Gaussian), "marggaus" (marginalising the Laplace approximation), and "sampling" (sampling from the joint Laplace approximation).

marginal_correction
: Which type of correction to use when fitting the skew-normal or two-piece Gaussian marginals. "shortcut" (default) computes only diagonals via central differences (full z-trace plus Schur complement correction), "shortcut_fd" is the same formula using forward differences (roughly half the cost, less accurate), "hessian" computes the full Hessian-based correction (slow), and "none" (or FALSE) applies no correction.

nsamp
: The number of samples to draw for all sampling-based approaches (including posterior sampling for model fit indices).

samp_copula
: Logical. When TRUE (default), posterior samples are drawn using the copula method with the fitted marginals (e.g. skew-normal or asymmetric Gaussian). When FALSE, samples are drawn from the Gaussian (Laplace) approximation.

samp_norta
: Logical. When TRUE, the latent correlation matrix of the skew-normal copula is adjusted by the NORmal-To-Anything (NORTA) scheme of Cario and Nelson (1997) so that the Pearson correlations of the copula draws match those of the Laplace approximation after the nonlinear quantile transform. The adjustment never changes a marginal; it affects only summaries that involve several parameters at once, and in practice moves the correlations very little. Default FALSE. Only used when samp_copula = TRUE and marginal_method = "skewnorm".

cov_as_cor
: Logical. Residual and latent-disturbance covariance parameters (~~ between two observed or two latent variables) are always estimated on the correlation scale internally (an atanh link, the same as for std.ov/std.lv-standardised parameters); by default their reported marginal is then re-derived on the covariance scale $\sigma_i \sigma_j \rho$ from a posterior sample (see samp_copula), because that is the scale lavaan/blavaan report by default. When TRUE, that re-derivation is skipped and each such parameter's own directly profiled correlation-scale marginal $\rho \in (-1, 1)$ is reported instead – useful for comparing the profiling machinery (skew-normal fit, VB, ...) against a correlation-scale reference without the sampling/copula step in between. Model estimation is identical either way; only what is reported for these parameters changes (and, correspondingly, their mat classification in the returned partable, theta_cov/psi_cov vs. theta_cor/psi_cor). Not the same as lavaan's std.ov/std.lv, which re-parameterises the whole model on a standardised scale. Defaults to FALSE.

sn_fit_ngrid
: Number of grid points to lay out per dimension when fitting the skew-normal marginals. A finer grid gives a better fit at the cost of more joint-log-posterior evaluations. Defaults to 21.

sn_fit_logthresh
: The log-threshold for fitting the skew-normal. Points with log-posterior drop below this threshold (relative to the maximum) will be excluded from the fit. Defaults to -6.

sn_fit_temp
: Temperature parameter for fitting the skew-normal. Defaults to 1 (weights are the density values themselves). If NA, the temperature is included as an additional optimisation parameter.

sn_fit_sample	Logical. When TRUE (default), a parametric skew-normal is fitted to the posterior samples for covariance and defined parameters. When FALSE, these are summarised using kernel density estimation instead.
control	A list of control parameters for the optimiser. For the default "nlminb", INLAvaan raises the stock iteration ceilings to iter.max = 1000 and eval.max = 2000 (complex models can exhaust nlminb()'s own defaults of 150 and 200); any value supplied here overrides these.
verbose	Logical indicating whether to print progress messages.
debug	Logical indicating whether to return debug information.
add_priors	Logical indicating whether to include prior densities in the posterior computation.
optim_method	The optimisation method to use for finding the posterior mode. Options include "nlminb" (default), "ucminf", and "optim" (BFGS).
numerical_grad	Logical indicating whether to use numerical gradients for the optimisation. Defaults to FALSE to use analytical gradients.
cores	Integer or NULL. Number of cores for parallel marginal fitting. When NULL (default), serial execution is used unless the number of free parameters exceeds 120, in which case parallelisation is enabled automatically using all available physical cores. Set to 1L to force serial execution. If cores > 1, marginal fits are distributed across cores – forked via parallel::mclapply() where that is safe, or over a PSOCK cluster (separate R processes) inside IDE R sessions (RStudio, Positron) and on Windows.
ppp_method	How the PPP of a two-level model scores the observed and the replicate data against the saturated model. "onestep" (default) takes one Fisher-scoring step from the moments of each posterior draw towards the saturated fit, and fits by EM where the step would leave the valid covariance matrices (with few clusters or a small between variance). "em" always fits the saturated model by EM, as blavaan does. Fits with missing = "ml" always use "em". Ignored for single-level models.
ppp_nsamp	The number of posterior draws, each with one replicate data set, that the PPP of a two-level model uses. Defaults to 250, and is capped at nsamp. Draws that cannot be scored are left out, with a warning. Ignored for single-level models.
...	Additional arguments to be passed to the lavaan model fitting function.

Details

The asem() function is a wrapper for the more general inlavaan() function, using the following default 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.efa = TRUE
- auto.th = TRUE
- auto.delta = TRUE
- auto.cov.y = TRUE

For further information regarding these arguments, please refer to the lavaan::lavOptions() documentation.

Value

An S4 object of class INLAvaan which is a subclass of the lavaan class.

See Also

Typically, users will interact with the specific latent variable model functions instead, including acfa(), asem(), and agrowth().

Examples

```
# The industrialization and Political Democracy Example from Bollen (1989), page
# 332
model <- "
  # Latent variable definitions
     ind60 =~ x1 + x2 + x3
     dem60 =~ y1 + a*y2 + b*y3 + c*y4
     dem65 =~ y5 + a*y6 + b*y7 + c*y8

  # (Latent) regressions
    dem60 ~ ind60
    dem65 ~ ind60 + dem60

  # Residual correlations
    y1 ~~ y5
    y2 ~~ y4 + y6
    y3 ~~ y7
    y4 ~~ y8
    y6 ~~ y8
"
utils::data("PoliticalDemocracy", package = "lavaan")

fit <- asem(model, PoliticalDemocracy, test = "none")
summary(fit)
```

---

as_fun_string *Convert function to single string*

Description

Convert function to single string

Usage

```
as_fun_string(f)
```

Arguments

f	Function to convert.

Value

A single character vector representing the function.

Examples

```
f <- function(x) { x^2 + 1 }
as_fun_string(f)
```

---

bfit_indices *Bayesian Fit Indices*

Description

Compute posterior distributions of Bayesian fit indices for an INLAvaan model, analogous to `blavaan::blavFitIndices()`.

Usage

```
bfit_indices(
  object,
  baseline.model = NULL,
  rescale = c("devM", "MCMC"),
  nsamp = NULL,
  samp_copula = TRUE
)

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

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

Arguments

object	An object of class INLAvaan.
baseline.model	The baseline (null) model that the incremental fit indices (BCFI, BTLI, BNFI) are scaled against. NULL (default) fits the independence model on the same data and options automatically, as lavaan does: every observed variable keeps its variance (and intercept) and nothing correlates. That fit uses Gaussian marginals

	and no VB shift, since only its posterior draws and pD are needed, and takes a fraction of a second. Supply an INLAvaan object to use another baseline, or FALSE to skip the incremental indices.
rescale	Character string controlling how the Bayesian chi-square is rescaled. "devM" (default) subtracts pD from the deviance at each sample. "MCMC" uses the classical chi-square and classical df at each sample.
nsamp	Number of posterior samples to draw. Defaults to the value used when fitting the model.
samp_copula	Logical. When TRUE (default), posterior samples are drawn using the copula method with the fitted marginals. When FALSE, samples are drawn from the Gaussian (Laplace) approximation.
...	Additional arguments passed to methods.
x	An object of class bfit_indices (for print).

Details

For a random-slope model, which has no saturated model, the chi-square and the number of sample moments come from the unrestricted random-coefficient model with the same random-effects design, fitted by maximum likelihood. This reference is INLAvaan's own construction. It is available on the closed-form route only, for models without between-only outcomes, and needs enough clusters for its parameters.

Value

An S3 object of class "bfit_indices" containing:

indices Named list of numeric vectors (one per posterior sample) for each computed fit index.

details List with chisq (per-sample deviance), df, pD, rescale, and nsamp.

Use summary() to obtain a table of posterior summaries (Mean, SD, quantiles, Mode) for each index.

See Also

lavaan::fitMeasures(), blavaan::blavFitIndices(), fitmeasures(), compare()

Examples

```
HS.model <- "
  visual  =~ x1 + x2 + x3
  textual =~ x4 + x5 + x6
  speed   =~ x7 + x8 + x9
"
utils::data("HolzingerSwineford1939", package = "lavaan")
fit <- acfa(HS.model, HolzingerSwineford1939, std.lv = TRUE, nsamp = 100,
            verbose = FALSE)

# Absolute fit indices
bf <- bfit_indices(fit)
```

```
bf
summary(bf)
```

`compare` *Compare Bayesian Models Fitted with INLAvaan*

Description

Compare two or more Bayesian SEM fitted with INLAvaan, reporting model-fit statistics and (optionally) fit indices side by side.

Usage

```
compare(x, y, ..., fit.measures = NULL, loo = FALSE)

## S4 method for signature 'INLAvaan'
compare(x, y, ..., fit.measures = NULL, loo = FALSE)

## S4 method for signature 'INLAvaan'
anova(object, ...)
```

Arguments

`x, y, ...`	Two or more INLAvaan (or `inlavaan_internal`) objects fitted to the same data.
`fit.measures`	Character vector of additional fit-measure names to include (e.g. `"BRMSEA"`, `"BCFI"`). Use `"all"` to include every measure returned by fitMeasures(). The default (`NULL`) shows only the core comparison statistics.
`loo`	Logical; if `TRUE`, compare models by leave-one-out cross-validation with paired standard errors (see Details). Defaults to `FALSE`.
`object`	An INLAvaan object (the `anova()` method, which is disabled and redirects to `compare()`).

Details

All models appear in the comparison table. When incremental fit indices (BCFI, BTLI, BNFI) are requested via `fit.measures`, they are scaled against the independence (null) model, fitted once on the data of the first model and shared by every model in the table (see `bfit_indices()`).

The default table always includes:

- **npar**: Number of free parameters.
- **Marg.Loglik**: Approximated marginal log-likelihood.
- **logBF**: Natural-log Bayes factor relative to the best model.
- **DIC / pD**: Deviance Information Criterion and effective number of parameters (when the fit computed the DIC, i.e. `test` included `"dic"` during fitting; the default `"standard"` does).

Fit all models with the same vb_correction setting. compare() warns when they differ.

Marginal likelihoods, Bayes factors and DIC of fits with composites (<~) treat the indicator (co)variances that lavaan fixes at their sample values as known, so compare() warns unless all models fix the same ones. The LOO comparison (loo = TRUE) remains valid across such models.

Set fit.measures to a character vector of measure names (anything returned by fitMeasures()) to append extra columns. Use fit.measures = "all" to include every available measure.

Set loo = TRUE to compare models by leave-one-out cross-validation (see loo()). This appends **ELPD / SE** (the Taylor expected log predictive density and its standard error), **p_loo**, and, against the best-ELPD model, the difference **elpd_diff** with its *paired* standard error **se_diff** computed from the pointwise contributions (the appropriate uncertainty for nested or same-data comparisons). Every model is scored at one common Taylor order, the lowest any of them can supply: if some unit of some model has no second-order term, all models are compared at first order, since otherwise a change of estimator between models would read as a difference between the models themselves. The order used is stated when the table is printed. The table is then sorted by descending ELPD. All models must be fitted to the same data with matching units; units are paired by id rather than by row order, so fits that stack groups differently – a pooled fit against a multigroup fit, or multigroup fits with different group orderings – still pair up unit by unit. For missing-data (FIML) fits, "the same data" also means the same observed entries: each unit is scored on the entries it has, so comparisons require identical missingness patterns across models. All models must also share the score flavour (see loo()): mixing fits with modelled covariates (fixed.x = FALSE, joint scores) and fixed covariates (fixed.x = TRUE, conditional scores) is refused. Joint scores additionally require identical variable sets across models, while conditional scores require only matching outcome variables – covariate sets may differ, which is the covariate-selection setting. Stored LOO results (test including "loo" or "full", or add_loo()) are reused.

When any of the models has random slopes (lavaan's rv() modifier; see inlavaan()), compare() aborts unless all the models were fitted with fixed.x = TRUE, score the same outcome variables, and share one integration.ngh on the quadrature route. Their covariates may differ. The fixed-slope model (the same path without rv()) is a valid comparator for testing a random slope.

anova() is disabled for INLAvaan fits – there is no direct Bayesian analogue of the classical likelihood-ratio test – and points here instead.

Value

A data frame of class compare.inlavaan_internal containing model fit statistics, sorted by descending marginal log-likelihood (or by descending ELPD when loo = TRUE).

References

https://lavaan.ugent.be/tutorial/groups.html

See Also

fitmeasures(), bfit_indices()

Examples

```
# Model comparison on multigroup analysis (measurement invariance)
HS.model <- '
```

```
  visual  =~ x1 + x2 + x3
  textual =~ x4 + x5 + x6
  speed   =~ x7 + x8 + x9
"
utils::data("HolzingerSwineford1939", package = "lavaan")

# Configural invariance
fit1 <- acfa(HS.model, data = HolzingerSwineford1939, group = "school")

# Weak invariance
fit2 <- acfa(
  HS.model,
  data = HolzingerSwineford1939,
  group = "school",
  group.equal = "loadings"
)

# Strong invariance
fit3 <- acfa(
  HS.model,
  data = HolzingerSwineford1939,
  group = "school",
  group.equal = c("intercepts", "loadings")
)

# Compare models (fit1 = configural = baseline, always first argument)
compare(fit1, fit2, fit3)

# With extra fit measures
compare(fit1, fit2, fit.measures = c("BRMSEA", "BMc"))

# With incremental indices (baseline = fit1, passed to fitMeasures())
compare(fit1, fit2, fit3, fit.measures = c("BCFI", "BTLI"))
```

`dbeta_box` *Density of a Beta distribution on a bounded interval*

Description

Density of a Beta distribution on a bounded interval

Usage

```
dbeta_box(x, shape1, shape2, a, b, log = FALSE)
```

Arguments

`x`	A numeric vector of quantiles.
`shape1, shape2`	non-negative parameters of the Beta distribution.

a	The lower bound of the interval.
b	The upper bound of the interval.
log	Logical; if TRUE, probabilities p are given as log(p).

Value

A numeric vector of density values.

Examples

```
# Beta(2,5) on (0,100)
x <- seq(0, 100, length.out = 100)
y <- dbeta_box(x, shape1 = 2, shape2 = 5, a = 0, b = 100)
plot(x, y, type = "l", main = "Beta(2,5) on (0,100)")

# Beta(1,1) i.e. uniform on (-1, 1)
x <- seq(-1, 1, length.out = 100)
y <- dbeta_box(x, shape1 = 1, shape2 = 1, a = -1, b = 1)
plot(x, y, type = "l", main = "Beta(1,1) on (-1,1)")
```

---

deviance *Deviance for INLAvaan Models*

Description

Extract the (Bayesian) deviance of a fitted INLAvaan model. Unlike **lavaan**, which has no deviance() method, this follows the BUGS/JAGS/Stan convention: "deviance" is -2 times the log-likelihood, summarised over the posterior.

Usage

```
## S3 method for class 'INLAvaan'
deviance(object, type = c("mean", "plugin"), ...)
```

Arguments

object	An object of class INLAvaan.
type	Character. "mean" (default) returns the posterior mean deviance $\bar{D} = E[-2 \log p(y \mid \theta)]$, averaged over posterior draws. "plugin" returns the deviance evaluated at the posterior mean point estimate, $\hat{D} = -2 \log p(y \mid \hat{\theta})$ (matching -2 * logLik(object, type = "plugin")). Both require the model to have been fitted with a test that includes "dic" (the default "standard" does).
...	Currently unused.

Details

$\bar{D}$ and $\hat{D}$ are the two ingredients of the Deviance Information Criterion, $DIC = \bar{D} + p_D$ where $p_D = \bar{D} - \hat{D}$ is the effective number of parameters. Use `compare()` to compare models by DIC (or Bayes factors, or LOO/WAIC) rather than comparing raw deviances directly.

This p_D is estimated from the posterior draws. `loo()` reaches the same quantity in closed form as `pd_trace`, the trace $\mathrm{tr}(\Sigma\mathcal{I})$; the two routes estimate one target but do not agree exactly in a finite sample. Neither is `p_loo`, which estimates something else entirely (see `loo()`).

Value

A length-one numeric of class `inlavaan_deviance`, with the effective number of parameters (`pD`) and `DIC` attached as attributes.

See Also

`logLik()`, `compare()`

Examples

```
HS.model <- "
  visual  =~ x1 + x2 + x3
  textual =~ x4 + x5 + x6
  speed   =~ x7 + x8 + x9
"
utils::data("HolzingerSwineford1939", package = "lavaan")
fit <- acfa(HS.model, HolzingerSwineford1939, std.lv = TRUE, nsamp = 100,
            test = "standard", verbose = FALSE)

deviance(fit)
attr(deviance(fit), "DIC")
```

---

diagnostics *Convergence and Approximation Diagnostics for INLAvaan Models*

Description

Extract convergence and approximation-quality diagnostics from a fitted `INLAvaan` model.

Usage

```
diagnostics(object, ...)

## S4 method for signature 'INLAvaan'
diagnostics(object, type = c("global", "param"), ...)
```

Arguments

`object`	An object of class INLAvaan.
`...`	Currently unused.
`type`	Character. `"global"` (default) returns a named numeric vector of scalar diagnostics. `"param"` returns a data frame with one row per free parameter containing per-parameter diagnostics.

Details

Global diagnostics (`type = "global"`):

`npar` Number of free parameters.

`nsamp` Number of posterior samples drawn.

`converged` 1 if the optimiser converged, 0 otherwise.

`iterations` Number of optimiser iterations.

`grad_inf` L-infinity norm of the analytic gradient at the mode (max |grad|). Should be ~0 at convergence.

`grad_inf_rel` Relative L-infinity norm of the analytic gradient (max |grad| / (|par| + 1e-6)).

`grad_l2` L2 (Euclidean) norm of the analytic gradient at the mode.

`mode_shift_max` Maximum, across parameters, of the Newton step at the reported mode in posterior-SD units (max $|\Sigma_\theta$ grad| / se). Unlike the raw gradient norms this is scale-free: it estimates how far the reported mode sits from the true posterior mode relative to the posterior uncertainty. Should be ~0 at convergence.

`hess_cond` Condition number of the Hessian (precision matrix) computed from Σ_θ. Large values indicate near-singularity.

`vb_kld_global` Global KL divergence from the VB mean correction (NA if VB correction was not applied).

`vb_applied` 1 if VB correction was applied, 0 otherwise.

`vb_shift_max` Maximum, across parameters, of the absolute VB correction in posterior-SD units (max |`vb_shift_sigma`|). This is the quantity the fit-time check tests. NA if the VB correction was not applied.

`kld_max` Maximum per-parameter KL divergence from the VB correction.

`kld_mean` Mean per-parameter KL divergence.

`vb_mcse_max` Maximum, across parameters, of the estimated quadrature error of the VB shift, in posterior-SD units. See `vb_mcse_sigma` below. NA under `vb_method = "gauss_hermite"`.

`vb_mcse_mean` Mean estimated quadrature error of the VB shift, in posterior-SD units.

`nmad_max` Maximum normalised max-absolute-deviation across marginals (skew-normal method only; NA otherwise).

`nmad_mean` Mean NMAD across marginals.

`scan_end_mass_max` Maximum, across parameters, of `scan_end_mass` (see below). NA unless the skew-normal marginal method was used.

Per-parameter diagnostics (`type = "param"`): A data frame with columns:

`names` Parameter name.

`grad` Analytic gradient of the negative log-posterior at the mode. Should be ~0 at convergence.

`grad_num` Numerical (finite-difference) gradient at the mode. Should agree with `grad`; large discrepancies indicate a bug in the analytic gradient.

`grad_diff` Difference `grad_num - grad`: should be ~0.

`grad_abs` Absolute analytic gradient.

`grad_rel` Relative analytic gradient |grad| / (|par| + 1e-6).

`mode_shift_sigma` Newton step at the reported mode in posterior-SD units. Should be ~0 at convergence.

`kld` Per-parameter KL divergence from the VB correction.

`vb_shift` VB correction shift (in original scale).

`vb_shift_sigma` VB shift in units of posterior SD.

`vb_mcse_sigma` Estimated quadrature error of the VB shift, in posterior-SD units. The shift is the solution of an integral evaluated by quasi-Monte Carlo over a finite node set, so it carries an integration error of its own. This estimates that error by splitting the node set in half and taking half the disagreement between the two half-set solutions. Read it as an error bar on `vb_shift_sigma`: a value of 0.05 means the reported posterior mean of that parameter could move by roughly that much, in SD units, purely from the choice of node set. It runs conservative, because the two halves are negatively correlated and quasi-Monte Carlo error falls faster than root-n. It is exactly zero for parameters whose shift is pinned by the saturated-means fast path, since no quadrature is used there. It is NA for every parameter under `vb_method = "gauss_hermite"`, whose rule is deterministic.

`nmad` Normalised max-absolute-deviation of the skew-normal fit (NA when not using the skewnorm method).

`alpha` Shape parameter of the fitted skew-normal marginal, on the unconstrained scale. Zero is a Gaussian marginal, and the sign gives the direction of the skew (NA when not using the skewnorm method).

`scan_end_mass` Mass the fitted marginal puts outside the window that was scanned to fit it, $F(\hat{\theta}_j - 4s_j) + 1 - F(\hat{\theta}_j + 4s_j)$, where F is the fitted skew-normal distribution function, $\hat{\theta}_j$ the posterior mode and s_j the Laplace posterior SD. The marginal is fitted inside that window only, so mass outside it is extrapolation, and when this is large the 2.5\ 97.5\ gives $2\Phi(-4)$ = 6.3e-05, and healthy fits sit between 1e-03 and 1e-02 (NA when not using the skewnorm method).

Fit-time warnings: `inlavaan()` runs these checks once at the end of every fit and emits a single consolidated warning (condition class `"inlavaan_diagnostics_warning"`) when any of them looks off:

- the optimiser did not converge;
- `mode_shift_max` above 0.1 posterior SDs;
- any marginal with `nmad` above 0.1;
- any marginal with `scan_end_mass` above 0.05;
- `vb_shift_max` above 1 posterior SD;
- `hess_cond` above 1e8.

Three of these checks have calibration behind them. Over 1,046 simulated fits with MCMC references, the Spearman correlation with the worst-case posterior-mean discrepancy was 0.64 for the VB shift in SD units, 0.59 for the scan-endpoint mass and 0.48 for the NMAD. The convergence, mode-shift and condition-number checks are rules of thumb. Every number above is a package default, chosen so that a healthy fit stays silent, and not a calibrated cut-off: a tripped check is a prompt to look again, and silence is not a certificate of accuracy. Silence the check with `suppressWarnings()`, or selectively by handling the condition class.

Value

For `type = "global"`, a named numeric vector (class `"diagnostics.INLAvaan"`). For `type = "param"`, a data frame (class `c("diagnostics.INLAvaan.param", "data.frame")`).

See Also

`timing()`, `fitmeasures()`, `plot()`

Examples

```
HS.model <- "
  visual  =~ x1 + x2 + x3
  textual =~ x4 + x5 + x6
  speed   =~ x7 + x8 + x9
"
utils::data("HolzingerSwineford1939", package = "lavaan")
fit <- acfa(HS.model, HolzingerSwineford1939, std.lv = TRUE, nsamp = 100,
            test = "none", verbose = FALSE)

# Global convergence summary
diagnostics(fit)

# Per-parameter table
diagnostics(fit, type = "param")
```

---

dsnorm *The Skew Normal Distribution*

Description

Density for the skew normal distribution with location `xi`, scale `omega`, and shape `alpha`.

Usage

```
dsnorm(x, xi, omega, alpha, logC = 0, log = FALSE)
```

Arguments

x	Vector of quantiles.
xi	Location parameter.
omega	Scale parameter.
alpha	Shape parameter.
logC	Log-normalization constant.
log	Logical; if TRUE, returns the log density.

Value

A numeric vector of (log) density values.

References

https://en.wikipedia.org/wiki/Skew_normal_distribution

Examples

```
x <- seq(-2, 5, length.out = 100)
y <- dsnorm(x, xi = 0, omega = 1, alpha = 5)
plot(x, y, type = "l", main = "Skew Normal Density")
```

---

fitmeasures *Fit Measures for a Latent Variable Model estimated using INLA*

Description

Fit Measures for a Latent Variable Model estimated using INLA

Usage

```
## S4 method for signature 'INLAvaan'
fitMeasures(object, fit_measures = "all",
  baseline_model = NULL, h1_model = NULL, fm_args,
  output = "vector", level = NULL, ...)

## S4 method for signature 'INLAvaan'
fitmeasures(object, fit_measures = "all",
  baseline_model = NULL, h1_model = NULL, fm_args,
  output = "vector", level = NULL, ...)
```

Arguments

object	An object of class INLAvaan.
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. The LOO measures "elpd_loo", "se_loo", "p_loo" and "looic" (see loo()), and the WAIC measures "elpd_waic", "se_waic", "p_waic" and "waic" (see waic()), are included in "all" only when stored with the fit (test including "loo", "waic" or "full" in inlavaan(), or add_loo()); otherwise they are computed on demand when requested by name, and recomputed on every call – store the result with fit <- add_loo(fit) (or call loo()/waic() directly) for repeated access. INLAvaan's stable spelling fit.measures is also accepted.
baseline_model	The baseline (null) model for the incremental fit indices (BCFI, BTLI, BNFI). NULL (default) fits the independence model automatically, as lavaan does. Supply an INLAvaan object to use another baseline, or FALSE to skip the incremental indices. INLAvaan's stable spelling baseline.model is also accepted; see bfit_indices().
h1_model	Ignored (included for compatibility with the lavaan generic).
fm_args	Ignored (included for compatibility with the lavaan generic).
output	Ignored (included for compatibility with the lavaan generic).
level	Ignored (included for compatibility with the lavaan generic).
...	Additional arguments. Currently supports:
	rescale Character string controlling how the Bayesian chi-square is computed, following blavaan::blavFitIndices(). Options are "devM" (default) which uses the deviance rescaled by pD from DIC, or "MCMC" which uses the classical chi-square ((N-1) * F_ML) and classical degrees of freedom (p - npar) at each posterior sample.

Value

A named numeric vector of fit measures.

See Also

bfit_indices(), compare(), diagnostics()

Examples

```
HS.model <- "
  visual  =~ x1 + x2 + x3
  textual =~ x4 + x5 + x6
  speed   =~ x7 + x8 + x9
"
utils::data("HolzingerSwineford1939", package = "lavaan")
fit <- acfa(HS.model, HolzingerSwineford1939, std.lv = TRUE, nsamp = 100,
            verbose = FALSE)

# All available fit measures
```

```
fitMeasures(fit)

# Specific measures
fitMeasures(fit, c("npar", "dic", "p_dic", "ppp"))
```

`fitted` *Model-Implied Moments for INLAvaan Models*

Description

Extract the model-implied (fitted) sample statistics from a fitted `INLAvaan` model. As in **lavaan** and **blavaan**, the moments are the model-implied covariance matrix (and mean vector, when a mean structure is present) evaluated at the parameter estimates – here the posterior means.

Usage

```
## S4 method for signature 'INLAvaan'
fitted(object, type = "moments", labels = TRUE, ..., per_cluster = FALSE)

## S4 method for signature 'INLAvaan'
fitted.values(
  object,
  type = "moments",
  labels = TRUE,
  ...,
  per_cluster = FALSE
)
```

Arguments

`object`	An object of class INLAvaan.
`type`	Character. `"moments"` (default) returns the model-implied variance-covariance matrix and, when relevant, the mean vector (plus thresholds for ordinal data). `"casewise"` (aliases `"obs"`, `"ov"`) returns the model-predicted values for each observation.
`labels`	Logical. Attach variable names to the output. Default `TRUE`.
`...`	Currently unused.
`per_cluster`	Logical. For a random-slope model, return the moments of each cluster instead of their average. Default `FALSE`.

Details

This delegates to **lavaan**'s own `fitted()` machinery, so the return structure matches lavaan exactly. Because INLAvaan stores the posterior means as the point estimates of the fitted object, the implied moments are the posterior-mean model-implied moments (mirroring **blavaan**).

For a model with random slopes (lavaan's `rv()`), the moments are averaged over the covariates and include the mean and the variance of each slope. With `per_cluster = TRUE`, each cluster gets the expected cluster mean and within-cluster covariance (divisor n_j) of its outcomes and covariates at its own covariate values, and the mean also holds the expected values of any between-only outcomes. This is not available for a slope on a latent or split covariate. `type = "casewise"` gives the outcomes' means given the covariates (closed-form route only).

Value

For `type = "moments"`, a list (or list of lists, for multiple groups) with elements such as `cov`, `mean`, and `th`. For `type = "casewise"`, a numeric matrix of predicted observed-variable values. With `per_cluster = TRUE`, a list with one element per cluster, each holding `cov` and `mean`.

See Also

`predict()`, `coef()`, fitMeasures()

Examples

```
HS.model <- "
  visual  =~ x1 + x2 + x3
  textual =~ x4 + x5 + x6
  speed   =~ x7 + x8 + x9
"
utils::data("HolzingerSwineford1939", package = "lavaan")
fit <- acfa(HS.model, HolzingerSwineford1939, std.lv = TRUE, nsamp = 100,
            test = "none", verbose = FALSE)

# Model-implied covariance matrix (posterior means)
fitted(fit)

# Casewise model-predicted observed values
head(fitted(fit, type = "ov"))
```

`fit_skew_normal` *Fit a skew normal distribution to log-density evaluations*

Description

Fit a skew normal distribution to log-density evaluations

Usage

```
fit_skew_normal(x, y, threshold_log_drop = -6, temp = NA)
```

Arguments

x	A numeric vector of points where the density is evaluated.
y	A numeric vector of log-density evaluations at points x.
threshold_log_drop	A negative numeric value indicating the log-density drop threshold below which points are ignored in the fitting. Default is -6.
temp	A numeric value for the temperature parameter k. If NA (default), it is included in the optimisation.

Details

This skew normal fitting function uses a weighted least squares approach to fit the log-density evaluations provided in y at points x. The weights are set to be the density evaluations raised to the power of the temperature parameter k. This has somewhat an interpretation of finding the skew normal fit that minimises the Kullback-Leibler divergence from the true density to it.

In R-INLA, the C code implementation from which this was translated from can be found here.

Value

A list with fitted parameters:

- xi: location parameter
- omega: scale parameter
- alpha: shape parameter
- logC: log-normalization constant
- k: temperature parameter
- rsq: R-squared of the fit

Note that logC and k are not used when fitting from a sample.

Examples

```
# Fit a SN curve to gamma log-density
logdens <- function(x) dgamma(x, shape = 3, rate = 1, log = TRUE)

x_grid <- seq(0.1, 8, length.out = 21)
y_log  <- sapply(x_grid, logdens)
y_log  <- y_log - max(y_log)  # normalise to have maximum at zero

res <- fit_skew_normal(x_grid, y_log, temp = 10)
unlist(res)

# Compare truth vs skew-normal approximation
x_fine <- seq(0.1, 8, length.out = 200)
```

```
y_true <- exp(logdens(x_fine))
y_sn   <- dsnorm(x_fine, xi = res$xi, omega = res$omega, alpha = res$alpha)

plot(x_fine, y_true, type = "n", xlab = "x", ylab = "Density", bty = "n")
polygon(c(x_fine, rev(x_fine)), c(y_true, rep(0, length(x_fine))),
        col = adjustcolor("#131516", 0.25), border = NA)
lines(x_fine, y_sn, col = "#00A6AA", lwd = 2)
legend("topright", legend = c("Truth", "SN Approx."),
       fill = c(adjustcolor("#131516", 0.25), NA), border = NA,
       col = c(NA, "#00A6AA"), lwd = c(NA, 2), lty = c(NA, 1),
       bty = "n")
```

`fit_skew_normal_samp` *Fit a skew normal distribution to a sample*

Description

Fit a skew normal distribution to a sample

Usage

```
fit_skew_normal_samp(x)
```

Arguments

`x`	A numeric vector of sample data.

Details

Uses maximum likelihood estimation to fit a skew normal distribution to the provided numeric vector `x`. The fit is computed on standardised draws, so it does not depend on the scale of `x`.

Value

A list with fitted parameters:

- `xi`: location parameter
- `omega`: scale parameter
- `alpha`: shape parameter
- `logC`: log-normalization constant
- `k`: temperature parameter
- `rsq`: R-squared of the fit

Note that `logC` and `k` are not used when fitting from a sample.

Examples

```
x <- rnorm(100, mean = 5, sd = 1)
unlist(fit_skew_normal_samp(x))
```

`get_inlavaan_internal` *Extract the Internal INLAvaan Object*

Description

Returns the `inlavaan_internal` list stored inside a fitted INLAvaan object, optionally extracting a single named element.

Usage

```
get_inlavaan_internal(object, what)
```

Arguments

`object`	An object of class INLAvaan.
`what`	Character. Name of the element to extract from the internal list. If missing, the entire list is returned. Common elements include `"coefficients"`, `"summary"`, `"Sigma_theta"`, `"vcov_x"`, `"theta_star"`, `"approx_data"`, `"pdf_data"`, `"partable"`, `"marginal_method"`, `"nsamp"`, `"mloglik"`, `"DIC"`, `"ppp"`, `"loo"`, `"waic"`, `"test"`, `"vb"`, `"opt"`, `"timing"`, `"visual_debug"`.

Value

The full `inlavaan_internal` list, or the named element when `what` is supplied.

See Also

INLAvaan, `diagnostics()`, `timing()`

Examples

```
HS.model <- "
  visual  =~ x1 + x2 + x3
  textual =~ x4 + x5 + x6
  speed   =~ x7 + x8 + x9
"
utils::data("HolzingerSwineford1939", package = "lavaan")
fit <- acfa(HS.model, HolzingerSwineford1939, std.lv = TRUE, nsamp = 100,
            test = "none", verbose = FALSE)

# Full internal object
int <- get_inlavaan_internal(fit)
names(int)

# Extract a specific element
get_inlavaan_internal(fit, "coefficients")
```

`inlavaan` *Fit an Approximate Bayesian Latent Variable Model*

Description

This function fits a Bayesian latent variable model by approximating the posterior distributions of the model parameters using various methods, including skew-normal, asymmetric Gaussian, marginal Gaussian, or sampling-based approaches. It leverages the lavaan package for model specification and estimation.

Usage

```
inlavaan(
  model,
  data,
  model.type = "sem",
  dp = priors_for(),
  test = "standard",
  vb_correction = TRUE,
  n_qmc = 64L,
  vb_method = c("sobol", "gauss_hermite"),
  marginal_method = c("skewnorm", "asymgaus", "marggaus", "sampling"),
  marginal_correction = c("shortcut", "shortcut_fd", "hessian", "none"),
  nsamp = 1000,
  samp_copula = TRUE,
  samp_norta = FALSE,
  cov_as_cor = FALSE,
  sn_fit_ngrid = 21,
  sn_fit_logthresh = -6,
  sn_fit_temp = 1,
  sn_fit_sample = TRUE,
  control = list(),
  verbose = TRUE,
  debug = FALSE,
  add_priors = TRUE,
  optim_method = c("nlminb", "ucminf", "optim"),
  numerical_grad = FALSE,
  start = NULL,
  cores = NULL,
  ppp_method = c("onestep", "em"),
  ppp_nsamp = 250L,
  ...
)
```

Arguments

`model` A description of the user-specified model. Typically, the model is described using the lavaan model syntax. See `model.syntax` for more information. Alter-

natively, a parameter table (e.g., the output of the `lavParTable()` function) is also accepted.

`data` An optional data frame containing the observed variables used in the model. If some variables are declared as ordered factors, lavaan will treat them as ordinal variables.

`model.type` The lavaan entry point used to fit `model`: `"cfa"`, `"sem"`, or `"growth"` (matching lavaan's model-specific wrapper functions), or `"lavaan"` for the general-purpose interface. Set automatically by `acfa()`, `asem()`, and `agrowth()`; documented explicitly here because lavaan >= 0.7-1 renamed the corresponding `simulateData()` argument to `model_type`, so it can no longer be inherited from there.

`dp` Default prior distributions for the different types of model parameters; a named character vector as returned by `priors_for()`. Types left out take their default priors.

`test` Character vector naming the post-estimation quantities to compute and store with the fit. The atoms are `"ppp"` (posterior predictive p-value), `"dic"` (deviance information criterion and its `pD`), `"loo"` (leave-one-out cross-validation, see `loo()`) and `"waic"` (see `waic()`). Three aliases stand for sets of atoms: `"standard"` (the default) and its synonym `"default"` give `c("ppp", "dic")`; `"full"` gives all four; `"none"` gives nothing. Aliases and atoms may be mixed and are unioned, so `test = c("standard", "loo")` adds the LOO to the default set. The LOO and the WAIC come from one Taylor pass, so asking for either stores both. They run only when asked for, with no time budget. On a model the casewise machinery does not support (PML or ordinal data, `conditional.x = TRUE`, multigroup two-level) they are skipped with a warning and the rest of the fit proceeds. For a two-level model the PPP follows blavaan: each of `ppp_nsamp` posterior draws generates replicate data, which are scored against the saturated model (see `ppp_method`). The two-level PPP is experimental. The fit records what was requested and what was computed (`get_inlavaan_internal(fit, "test")`); `summary()`, `fitmeasures()`, `deviance()`, `logLik()` and `timing()` report only what was computed. `add_loo()` stores the LOO and WAIC post hoc; `loo()` and `waic()` compute on demand. For a random-slope model `"ppp"` is dropped, with a warning when it was asked for and a message otherwise (see the Random slopes section of `inlavaan()`). `"dic"`, `"loo"` and `"waic"` are unaffected.

`vb_correction` Logical indicating whether to apply a variational Bayes correction for the posterior mean vector of estimates. Defaults to `TRUE`.

`n_qmc` Number of quasi-Monte Carlo nodes used by the VB mean correction. Defaults to `64`; see the Details section of `inlavaan()`. Values above `128` (the size of the stored Sobol table) require the **qrng** package. Ignored when `vb_correction = FALSE` or `vb_method = "gauss_hermite"`.

`vb_method` Integration rule for the VB mean correction. `"sobol"` (default) averages over `n_qmc` scrambled Sobol nodes. `"gauss_hermite"` uses a deterministic rule instead: a three-point Gauss-Hermite rule along each principal axis of the Laplace covariance, `2m + 1` nodes in all for `m` free parameters. It is exact whenever the log-posterior is quartic in whitened coordinates, and it gives the same shift on every run. Having no node sets to compare, it reports no quadrature error,

so vb_mcse_sigma in diagnostics() is NA. Its cost grows with m: it is cheaper than the default below about 30 free parameters and dearer above. Experimental.

marginal_method
: The method for approximating the marginal posterior distributions. Options include "skewnorm" (skew-normal), "asymgaus" (two-piece asymmetric Gaussian), "marggaus" (marginalising the Laplace approximation), and "sampling" (sampling from the joint Laplace approximation).

marginal_correction
: Which type of correction to use when fitting the skew-normal or two-piece Gaussian marginals. "shortcut" (default) computes only diagonals via central differences (full z-trace plus Schur complement correction), "shortcut_fd" is the same formula using forward differences (roughly half the cost, less accurate), "hessian" computes the full Hessian-based correction (slow), and "none" (or FALSE) applies no correction.

nsamp
: The number of samples to draw for all sampling-based approaches (including posterior sampling for model fit indices).

samp_copula
: Logical. When TRUE (default), posterior samples are drawn using the copula method with the fitted marginals (e.g. skew-normal or asymmetric Gaussian). When FALSE, samples are drawn from the Gaussian (Laplace) approximation.

samp_norta
: Logical. When TRUE, the latent correlation matrix of the skew-normal copula is adjusted by the NORmal-To-Anything (NORTA) scheme of Cario and Nelson (1997) so that the Pearson correlations of the copula draws match those of the Laplace approximation after the nonlinear quantile transform. The adjustment never changes a marginal; it affects only summaries that involve several parameters at once, and in practice moves the correlations very little. Default FALSE. Only used when samp_copula = TRUE and marginal_method = "skewnorm".

cov_as_cor
: Logical. Residual and latent-disturbance covariance parameters (~~ between two observed or two latent variables) are always estimated on the correlation scale internally (an atanh link, the same as for std.ov/std.lv-standardised parameters); by default their reported marginal is then re-derived on the covariance scale $\sigma_i \sigma_j \rho$ from a posterior sample (see samp_copula), because that is the scale lavaan/blavaan report by default. When TRUE, that re-derivation is skipped and each such parameter's own directly profiled correlation-scale marginal $\rho \in (-1, 1)$ is reported instead – useful for comparing the profiling machinery (skew-normal fit, VB, ...) against a correlation-scale reference without the sampling/copula step in between. Model estimation is identical either way; only what is reported for these parameters changes (and, correspondingly, their mat classification in the returned partable, theta_cov/psi_cov vs. theta_cor/psi_cor). Not the same as lavaan's std.ov/std.lv, which re-parameterises the whole model on a standardised scale. Defaults to FALSE.

sn_fit_ngrid
: Number of grid points to lay out per dimension when fitting the skew-normal marginals. A finer grid gives a better fit at the cost of more joint-log-posterior evaluations. Defaults to 21.

sn_fit_logthresh
: The log-threshold for fitting the skew-normal. Points with log-posterior drop below this threshold (relative to the maximum) will be excluded from the fit. Defaults to -6.

sn_fit_temp	Temperature parameter for fitting the skew-normal. Defaults to 1 (weights are the density values themselves). If NA, the temperature is included as an additional optimisation parameter.
sn_fit_sample	Logical. When TRUE (default), a parametric skew-normal is fitted to the posterior samples for covariance and defined parameters. When FALSE, these are summarised using kernel density estimation instead.
control	A list of control parameters for the optimiser. For the default "nlminb", INLAvaan raises the stock iteration ceilings to iter.max = 1000 and eval.max = 2000 (complex models can exhaust nlminb()'s own defaults of 150 and 200); any value supplied here overrides these.
verbose	Logical indicating whether to print progress messages.
debug	Logical indicating whether to return debug information.
add_priors	Logical indicating whether to include prior densities in the posterior computation.
optim_method	The optimisation method to use for finding the posterior mode. Options include "nlminb" (default), "ucminf", and "optim" (BFGS).
numerical_grad	Logical indicating whether to use numerical gradients for the optimisation. Defaults to FALSE to use analytical gradients.
start	Optional numeric vector of starting values for the optimiser, given as a full vector of free parameters in the internal (unconstrained) parameterisation. Mainly for internal use by update(), which warm-starts mode-finding from a previous fit's posterior mode; supplying a hand-built vector requires knowledge of the internal parameter ordering. Its length must equal the number of free parameters or an error is raised.
cores	Integer or NULL. Number of cores for parallel marginal fitting. When NULL (default), serial execution is used unless the number of free parameters exceeds 120, in which case parallelisation is enabled automatically using all available physical cores. Set to 1L to force serial execution. If cores > 1, marginal fits are distributed across cores – forked via parallel::mclapply() where that is safe, or over a PSOCK cluster (separate R processes) inside IDE R sessions (RStudio, Positron) and on Windows.
ppp_method	How the PPP of a two-level model scores the observed and the replicate data against the saturated model. "onestep" (default) takes one Fisher-scoring step from the moments of each posterior draw towards the saturated fit, and fits by EM where the step would leave the valid covariance matrices (with few clusters or a small between variance). "em" always fits the saturated model by EM, as blavaan does. Fits with missing = "ml" always use "em". Ignored for single-level models.
ppp_nsamp	The number of posterior draws, each with one replicate data set, that the PPP of a two-level model uses. Defaults to 250, and is capped at nsamp. Draws that cannot be scored are left out, with a warning. Ignored for single-level models.
...	Additional arguments to be passed to the lavaan model fitting function.

Details

The VB mean correction integrates over `n_qmc` quasi-Monte Carlo nodes, so it carries a quadrature error that falls as `n_qmc` rises. The default of `64` keeps this error at roughly 0.05 posterior SDs – on par with the Monte Carlo error of a routine MCMC run, and small against the shifts being corrected. Users may increase `n_qmc` to reduce the error further, at a proportional cost in computation time; `diagnostics()` reports the realised error per fit as `vb_mcse_sigma` per parameter and `vb_mcse_max` globally, both in posterior-SD units. Setting `vb_method = "gauss_hermite"` removes the random node set altogether; see the `vb_method` argument.

Value

An S4 object of class `INLAvaan` which is a subclass of the lavaan class.

Random slopes

Wrapping a level-1 regression coefficient in lavaan's `rv()` modifier turns it into a level-2 latent variable – a random slope – which can then be given a mean, a variance and cross-level regressions like any other level-2 latent variable:

```
level: 1
  fw =~ y1 + y2 + y3
  fw ~ rv('s1')*x1
level: 2
  fb =~ y1 + y2 + y3
  s1 ~ w1
  s1 ~~ s1
  s1 ~ 1
```

The slope's variance and intercept are added automatically, so in practice only the cross-level regression `s1 ~ w1` need be written out. `summary()` marks the level-1 carrier row as `x1 (s1)`: that row is fixed at zero, the slope itself being reported under Level 2.

The likelihood takes one of two routes. When the covariate carrying the slope is observed and purely within-cluster, the slope integrates out in closed form. When it is latent, or split across both levels (the same variable entering at level 1 and at level 2), the integral is done by Gauss-Hermite quadrature instead; that route warns at fit time and is governed by `integration.ngh`, passed through to lavaan. The node count is an accuracy setting as much as a cost setting, so give every fit that is to be compared with another the same `integration.ngh`.

A model with observed exogenous covariates requires `fixed.x = TRUE` (the default): the likelihood is the density of the outcomes *given* those covariates, so their own means and (co)variances are unidentified and would be reported back as their priors. A `fixed.x = FALSE` fit is refused. A model whose covariates are all latent or modelled has nothing to hold fixed, and lavaan reports `fixed.x = FALSE` for it of its own accord; such a fit is accepted as it stands.

Equality constraints work as for other models, except that covariances cannot be held equal. Composites (`<~`) cannot be combined with random slopes yet.

A random-slope model implies no single within-cluster covariance matrix: the covariance of the outcomes depends on the covariate values. `fitted()` and `residuals()` therefore use the implied moments averaged over the covariates, which include the mean and the variance of each slope, or

with `per_cluster = TRUE` the moments of each cluster at its own covariate values (closed-form route only). `standardisedsolution()` and the standardised columns and R-square of `summary()` are scaled by the averaged variances. The carrier row `x1 (s1)` then gives the standardised mean slope, and the slope's own rows are on the same standardised scale. An outcome observed at level 1 only, whose slope covariate has a non-zero mean, has no place in lavaan's two-level layout, and these outputs abort for it.

`simulate()` draws each cluster's slopes and other level-2 effects and then its outcomes, at the cluster's own covariates (closed-form route only). `sampling()` gives latent, observed and implied draws. `predict()` gives `type = "lv"` and, on the closed-form route, the cluster-specific `"yhat"` and `"ypred"` from the empirical Bayes random effects, while `fitted(type = "casewise")` gives the outcomes' means given the covariates. `bfit_indices()` scales the Bayesian fit indices against the unrestricted random-coefficient model with the same random-effects design, because a saturated model does not exist here. This reference is INLAvaan's own construction (closed-form route only).

The posterior predictive p-value (`test = "ppp"`) is dropped, because a random-slope model has no saturated model to score replicate data against. What aborts with an explanation: the residual types scaled by standard errors, `predict(type = "ymis")`, and `loo(type = "loso")`, which would need a cluster's sufficient statistics downdated by one row.

The model-comparison side works throughout. `compare()` reports the marginal likelihood, Bayes factors, the DIC and its p_D, and `loo()` and `waic()` score the fit leave-one-cluster-out on the conditional likelihood.

To ask whether there is a random slope at all, compare the fit with the fixed-slope model (`fw ~ x1` at level 1) using `compare()`. On the closed-form route, fixing the slope variance at zero (`s1 ~~ 0*s1`) and dropping any cross-level regression on the slope gives the same model. Keeping `s1 ~ w1` gives a cross-level interaction model instead. The quadrature route refuses a slope variance fixed at zero.

See Also

Typically, users will interact with the specific latent variable model functions instead, including `acfa()`, `asem()`, and `agrowth()`.

Examples

```
# The Holzinger and Swineford (1939) example
HS.model <- "
  visual  =~ x1 + x2 + x3
  textual =~ x4 + x5 + x6
  speed   =~ x7 + x8 + x9
"
utils::data("HolzingerSwineford1939", package = "lavaan")

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

INLAvaan-class *Class For Representing a (Fitted) Latent Variable Model*

Description

This is a class that extends the lavaan class. Several S4 methods are available.

Usage

```
## S4 method for signature 'INLAvaan'
coef(object, ...)

## S4 method for signature 'INLAvaan'
nobs(object, ...)

## S4 method for signature 'INLAvaan'
show(object)

## S4 method for signature 'INLAvaan'
summary(
  object,
  header = TRUE,
  fit.measures = TRUE,
  estimates = TRUE,
  ci = TRUE,
  standardized = FALSE,
  standardised = standardized,
  rsquare = FALSE,
  postmedian = FALSE,
  postmode = FALSE,
  nmad = TRUE,
  kld = FALSE,
  vb_shift = FALSE,
  priors = TRUE,
  nd = 3L,
  ...
)
```

Arguments

object	An object of class INLAvaan.
...	Additional arguments passed to methods.
header	Logical; if TRUE, print model fit information header.
fit.measures	Logical; if TRUE, print fit measures (DIC and pD; PPP is printed by the header regardless). Both require the fit to have computed them, see the test argument of inlavaan().

estimates	Logical; if TRUE, print parameter estimates table.
ci	Logical; if TRUE (default), include 95% credible intervals (2.5% and 97.5% posterior quantiles) in the estimates table.
standardized	Logical; if TRUE, include standardized estimates.
standardised	Alias of standardized; either spelling is accepted (the alias wins if both are supplied).
rsquare	Logical; if TRUE, include R-square values for the observed dependent variables, computed at the posterior mean point estimates.
postmedian	Logical; if TRUE, include posterior median in estimates.
postmode	Logical; if TRUE, include posterior mode in estimates.
nmad	Logical; if TRUE (default), include the NMAD column for skew-normal marginal fit quality.
kld	Logical; if FALSE (default), omit the per-parameter KLD column. Set to TRUE to show it.
vb_shift	Logical; if FALSE (default), omit the VB shift column (shift in units of posterior SD). Set to TRUE to show it.
priors	Logical; if TRUE, include prior information in estimates.
nd	Integer; number of decimal places to print for numeric values.

Slots

external A list containing an inlavaan_internal object.

See Also

lavaan, inlavaan(), acfa(), asem(), agrowth()

Examples

```
HS.model <- "
  visual  =~ x1 + x2 + x3
  textual =~ x4 + x5 + x6
  speed   =~ x7 + x8 + x9
"
utils::data("HolzingerSwineford1939", package = "lavaan")
fit <- acfa(HS.model, HolzingerSwineford1939, std.lv = TRUE, nsamp = 100,
            test = "none", verbose = FALSE)

# Print basic info
fit

# Detailed summary
summary(fit)

# Extract coefficients
coef(fit)
```

is_same_function *Helper function to check if two functions are the same*

Description

Helper function to check if two functions are the same

Usage

```
is_same_function(f, g)
```

Arguments

f, g	Functions to compare.

Value

Logical.

Examples

```
f1 <- function(x) { x^2 + 1 }
f2 <- function(x) { x^2 + 1 }
is_same_function(f1, f2)  # TRUE
```

logLik *Log-Likelihood for INLAvaan Models*

Description

Extract a log-likelihood-flavoured summary from a fitted INLAvaan model. Two distinct quantities are available, deliberately not conflated: the Bayesian marginal log-likelihood (the default) and the classical log-likelihood evaluated at the posterior mean.

Usage

```
## S4 method for signature 'INLAvaan'
logLik(object, type = c("marginal", "plugin"), ...)

## S4 method for signature 'INLAvaan'
AIC(object, ..., k = 2)

## S4 method for signature 'INLAvaan'
BIC(object, ...)
```

Arguments

object	An object of class INLAvaan.
type	Character. "marginal" (default) returns the Laplace-approximated marginal log-likelihood (log evidence), the same quantity compare() uses for Bayes factors. "plugin" returns the classical log-likelihood evaluated at the posterior mean point estimate, with df/nobs attributes and class "logLik" so it supports AIC/BIC at the point estimate. Requires the model to have been fitted with a test that includes "dic" (the default "standard" does).
...	Currently unused.
k	Numeric penalty per parameter passed to the (disabled) AIC() method; see stats::AIC(). Defaults to 2.

Details

The marginal log-likelihood already integrates over the (Laplace- approximated) posterior, so it is not on the same scale as a classical log-likelihood and should not be passed to AIC()/BIC() – doing so would double-penalise model complexity that the evidence has already accounted for. Use compare() to compare models via Bayes factors, DIC, or LOO/WAIC. The plug-in variant exists for users who specifically want a point-estimate-based classical comparison.

AIC()/BIC() on an INLAvaan fit are themselves disabled (mirroring anova()): both are large-sample asymptotic approximations to quantities INLAvaan already computes directly – AIC approximates predictive accuracy, which loo()/waic() already estimate more rigorously; BIC approximates -2 times the log marginal likelihood, which logLik() already returns directly (up to the Laplace approximation). Point-estimate AIC/BIC remain available for reporting-convention purposes via AIC(logLik(object, type = "plugin")) / BIC(logLik(object, type = "plugin")).

Value

For type = "marginal", a length-one numeric of class inlavaan_logLik that prints with a note on its interpretation. For type = "plugin", a standard "logLik" object.

See Also

deviance(), compare(), loo(), waic()

Examples

```
HS.model <- "
  visual  =~ x1 + x2 + x3
  textual =~ x4 + x5 + x6
  speed   =~ x7 + x8 + x9
"
utils::data("HolzingerSwineford1939", package = "lavaan")
fit <- acfa(HS.model, HolzingerSwineford1939, std.lv = TRUE, nsamp = 100,
            test = "standard", verbose = FALSE)

# Marginal log-likelihood (log evidence)
logLik(fit)
```

```
# Classical log-likelihood at the posterior mean, AIC/BIC-compatible
ll <- logLik(fit, type = "plugin")
AIC(ll)
```

`loo` *Approximate Leave-One-Out Cross-Validation for INLAvaan Models*

Description

Computes leave-one-out (LOO) cross-validation for a fitted INLAvaan model from a single fit, with no refitting and no sampling, via a Taylor approximation of the case-deletion posterior around the Laplace summary. Single-level models are scored per subject (leave-one-subject-out, LOSO); two-level models are scored per cluster (leave-one-cluster-out, LOCO).

Usage

```
loo(x, ...)

## S3 method for class 'INLAvaan'
loo(
  x,
  type = c("auto", "loso", "loco"),
  units = NULL,
  second_order = TRUE,
  theta = NULL,
  Omega = NULL,
  cores = NULL,
  verbose = FALSE,
  ...
)

## S3 method for class 'inlavaan_internal'
loo(
  x,
  type = c("auto", "loso", "loco"),
  units = NULL,
  second_order = TRUE,
  theta = NULL,
  Omega = NULL,
  cores = NULL,
  verbose = FALSE,
  ...
)

add_loo(object, cores = NULL, verbose = FALSE)
```

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

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

Arguments

x	A fitted INLAvaan object (or its inlavaan_internal list).
...	Not used, beyond catching the deprecated argument name Sigma (see Omega).
type	Unit type: "auto" (default) resolves to "loso" (per-subject) for single-level models and "loco" (per-cluster, marginal predictive) for two-level models. "loco" cannot be forced on a model without clusters; "loso" on a two-level model scores the conditional (leave-one-unit-out) predictive instead (with a warning; see Details).
units	Optional integer vector of unit indices to score; defaults to all units. For LOSO these are case numbers (row numbers of the analysed dataset); for LOCO, cluster positions. For a random-slope fit the per-cluster kernels return every cluster on each call, so this trims the reported table rather than the work.
second_order	Logical; compute the second-order correction (default TRUE). FALSE skips the Hessian stage and reports first-order estimates, which cannot be compared across models of different dimension (see Details).
theta, Omega	Optional posterior mean vector and covariance matrix (in the unconstrained parameter space, as stored in theta_star and Sigma_theta) at which to evaluate the LOO instead of the fit's own Laplace summary. See Details.
cores	Number of cores for the Hessian stage. The default NULL runs serially; parallelism must be requested explicitly.
verbose	Logical; print progress (default FALSE).
object	A fitted INLAvaan object, or an inlavaan_loo result for summary().

Details

For a unit u (a subject for LOSO, a cluster for LOCO) with log-likelihood contribution $\ell_u(\theta)$, score s_u and Hessian H_u evaluated at the posterior summary (θ^*, Ω), the log conditional predictive ordinate is approximated to first and second order by

$$\log \mathrm{CPO}_u^{(1)} = \ell_u - \tfrac{1}{2} s_u' \Omega s_u,$$

$$\log \mathrm{CPO}_u^{(2)} = \ell_u - \tfrac{1}{2} s_u' (\Omega^{-1} + H_u)^{-1} s_u + \tfrac{1}{2} \log |I + \Omega H_u|.$$

The reported elpd_loo is the sum of the second-order terms (first-order when second_order = FALSE), with standard error $\sqrt{n \operatorname{var}(\log \mathrm{CPO}_u)}$ and looic $= -2\,\mathrm{elpd}$. p_loo is the **loo** package's effective number of parameters, $p_{\mathrm{loo}} = \sum_u (\mathrm{lpd}_u - \log \mathrm{CPO}_u)$, where lpd_u is the analogous Taylor approximation of the full-posterior pointwise log predictive density – the same definition loo::loo() reports, and *not* the p_D of the DIC.

$\log \mathrm{CPO}_u^{(2)}$ exists exactly when $\Omega^{-1} + H_u$ is positive definite (recorded in `per_unit$ok`) and $\mathrm{lpd}_u^{(2)}$ exactly when $\Omega^{-1} - H_u$ is. A unit failing the former drops every estimate to first order over all units (with a warning), while one failing only the latter contributes its first-order difference to `p_loo`.

The leverages `k_max`, `k_min`, and `k_sum` read these conditions off the spectrum of $-\Omega H_u$ (`k_max` < 1, `k_min` > -1), with `k_sum` summing across units to the trace form of p_D. `p_loo` (cross-product form) and p_D (second-derivative form) agree only in the correct-specification limit, and printing a result reports the first-to-second-order gap against its limit $p_D/2$ as a free check on the Taylor truncation. Because the first-order elpd overstates the truth by $\frac{1}{2}p_D$ in the limit, keep `second_order = TRUE` whenever models of different dimension are compared.

`type = "auto"` resolves to the marginal per-cluster `"loco"` for two-level fits and per-subject `"loso"` otherwise. Forcing `"loso"` on a two-level model scores the *conditional* predictive of Merkle, Furr & Rabe-Hesketh (2019) instead, and warns. Multigroup units are scored against their own group's implied moments and identified by case number, so `compare()` pairs them across fits.

The score follows the fitted likelihood's treatment of exogenous covariates, i.e. joint under `fixed.x = FALSE`, conditional under `fixed.x = TRUE` (recorded as `"joint"` or `"conditional"` in the result's `flavour` field), and the two flavours are never comparable (`compare()` refuses to mix them).

For composites (<~), lavaan fixes the (co)variances of the indicators at their sample values. Each unit is scored with them recomputed without it (to first order under missing data), so `p_loo` counts them as parameters.

A random-slope fit (lavaan's `rv()` modifier; see `inlavaan()`) is scored leave-one-cluster-out only, conditional on the exogenous covariates (`flavour` is `"conditional"`).

Supplying `theta`/`Omega` evaluates the LOO at an arbitrary Gaussian posterior summary (a singular `Omega` is restricted to its non-degenerate block), the building block for refit-free submodel scoring. `Sigma` is the deprecated former name of `Omega`, still accepted through `...` with a warning.

The LOO is stored with the fit when `inlavaan()`'s `test` includes `"loo"` or `"waic"` (e.g. `test = "full"`), which also stores the WAIC (see `waic()`) from the same Taylor pass, or afterwards with `add_loo()`; `loo(fit)` with default arguments then returns the stored result. Under the default `test = "standard"` nothing is stored and `loo(fit)` computes it on demand.

The default `cores = NULL` runs serially, and `cores > 1` parallelises the Hessian stage. Supported models are continuous-indicator models fitted with the `ML` estimator, single- or two-level, single-group or multigroup (multigroup two-level models are not supported yet).

Value

An object of class `inlavaan_loo`: a list with elements

`per_unit` Data frame of pointwise results, one row per unit:

- `unit` Case number for LOSO, cluster position for LOCO.
- `group` Group membership (multigroup fits only).
- `nobs` 1 for LOSO, the cluster size for LOCO.
- `l_star` Unit log-likelihood at the summary.
- `score_norm` Norm of the unit score s_u.
- `lpd_1`, `lpd_2` Pointwise log predictive density, at first and second order.
- `log_cpo_1`, `log_cpo_2` Pointwise LOO contributions, at first and second order.
- `det_term` $\frac{1}{2}\log|I + \Omega H_u|$, the determinant term of the second-order score.

`k_max`, `k_min`, `k_sum` Leverage diagnostics (see Details).

`k_ssq` $\mathrm{tr}[(\Omega H_u)^2]$, consumed by the closed-form `waic()` penalty.

`ok` Whether the second-order $\log \mathrm{CPO}$ exists.

`t_delta` Composite fits only: the drop in ℓ_u when the fixed indicator (co)variances are recomputed without the unit, already subtracted from the log CPO terms.

`estimates` Matrix with rows `elpd_loo`, `p_loo`, `looic` and columns `Estimate`, `SE`, at the highest order available to each.

`elpd_1`, `elpd_2`, `se_1`, `se_2`, `p_loo_1`, `p_loo_2` First- and second-order aggregates; the second-order ones are `NA` when any $\log \mathrm{CPO}_u^{(2)}$ does not exist.

`elpd_gap`, `pd_trace` The two sides of the curvature check (see Details); both are `NA` at first order and partial totals under a `units` subset.

`type`, `flavour`, `n_units`, `n_groups`, `n_ok`, `n_lpd_ok`, `second_order`, `use_second`, `theta_overridden` Metadata.

`summary()` is an alias for `print()`: it prints the same output and returns the result invisibly.

`add_loo()` returns a copy of `object` with the LOO and WAIC results stored alongside the fit (the input object is unchanged); reassign it, e.g. `fit <- add_loo(fit)`. Only the default LOO is stored, so the stored results always match `loo(fit)` and `waic(fit)`.

References

Alhyari, M., Jamil, H., Montcho, H., & Rue, H. (2026). *Deterministic leave-one-cluster-out cross-validation for multilevel Bayesian structural equation models*. arXiv:2609.00670. doi:10.48550/arXiv.2609.00670

Merkle, E. C., Furr, D., & Rabe-Hesketh, S. (2019). Bayesian comparison of latent variable models: Conditional versus marginal likelihoods. *Psychometrika*, *84*(3), 802–829. doi:10.1007/s11336019-096790

See Also

`fitmeasures()`, `compare()`, `inlavaan()`

Examples

```
HS.model <- "
  visual  =~ x1 + x2 + x3
  textual =~ x4 + x5 + x6
  speed   =~ x7 + x8 + x9
"
utils::data("HolzingerSwineford1939", package = "lavaan")
fit <- acfa(HS.model, HolzingerSwineford1939, meanstructure = TRUE)

# Leave-one-subject-out (LOSO) from the single fit -- no refitting
res <- loo(fit)
res
head(res$per_unit)

# Score a submodel without refitting: condition the Laplace summary on the
```

```
# visual ~~ speed covariance being zero, then evaluate at that summary
int <- get_inlavaan_internal(fit)
theta <- int$theta_star
Omega <- int$Sigma_theta
p <- which(names(coef(fit)) == "visual~~speed")
theta_c <- theta - Omega[, p] * (theta[p] / Omega[p, p])
Omega_c <- Omega - tcrossprod(Omega[, p]) / Omega[p, p]
loo(fit, theta = theta_c, Omega = Omega_c)

# Two-level models are scored per cluster (LOCO) automatically
utils::data("Demo.twolevel", package = "lavaan")
model2l <- "
  level: 1
    fw =~ y1 + y2 + y3
    fw ~ x1 + x2 + x3
  level: 2
    fb =~ y1 + y2 + y3
    fb ~ w1 + w2
"
fit2l <- asem(model2l, Demo.twolevel, cluster = "cluster",
              meanstructure = TRUE, fixed.x = FALSE)
loo(fit2l)
```

---

plot *Plot an INLAvaan Object*

Description

Generates diagnostic plots for a fitted INLAvaan model.

Usage

```
## S4 method for signature 'INLAvaan,ANY'
plot(
  x,
  y,
  type = c("marg_pdf", "sn_fit", "sn_fit_log"),
  params = "all",
  nrow = NULL,
  ncol = NULL,
  use_ggplot = TRUE,
  points = FALSE,
  ...
)
```

Arguments

x	An object of class INLAvaan.
y	Not used.
type	Character. One of "marg_pdf" (default; posterior marginal densities), "sn_fit" (skew-normal fit diagnostic on natural scale), or "sn_fit_log" (same on log scale).
params	Character vector of parameter names to plot, as in coef() or := labels, or "all" (default) to plot all parameters.
nrow, ncol	Integer. Number of rows/columns for the facet grid when use_ggplot = TRUE. If NULL (default), layout is chosen automatically.
use_ggplot	Logical. When TRUE (default) and **ggplot2** is available, a **ggplot2** object is returned. Set to FALSE for a base-R plot.
points	Logical. When TRUE, individual grid points are overlaid on the curves. Defaults to FALSE.
...	Additional arguments (currently unused).

See Also

diagnostics(), summary()

Examples

```
HS.model <- "
  visual  =~ x1 + x2 + x3
  textual =~ x4 + x5 + x6
  speed   =~ x7 + x8 + x9
"
utils::data("HolzingerSwineford1939", package = "lavaan")
fit <- acfa(HS.model, HolzingerSwineford1939, std.lv = TRUE, nsamp = 100,
            test = "none", verbose = FALSE)

# Posterior marginal densities (default)
plot(fit)

# Skew-normal fit diagnostic for a single parameter
plot(fit, type = "sn_fit", params = "visual=~x1")
```

---

predict *Posterior Predictions for INLAvaan Models*

Description

Compute posterior predictions from a fitted INLAvaan model, including latent variable scores, predicted observed values, and imputed missing data.

Usage

```
## S4 method for signature 'INLAvaan'
predict(
  object,
  type = c("lv", "yhat", "ov", "ypred", "ydist", "ymis", "ovmis"),
  newdata = NULL,
  level = 1L,
  nsamp = 1000,
  ymis_only = FALSE,
  summary = FALSE,
  ...
)
```

Arguments

`object`	An object of class INLAvaan.
`type`	Character string specifying the type of prediction: `"lv"` (default) Posterior draws of latent variable scores $\eta \mid y, \theta$. A composite (`<~`) score is the weighted sum of its indicators (centred without a mean structure). `"yhat"`, `"ov"` Predicted means for observed variables $E(y \mid \eta, \theta) = \nu + \Lambda\eta$; no residual noise. An observed outcome is predicted from its regressors, and an indicator of a composite from the composite. `"ypred"`, `"ydist"` Predicted observed values including residual noise $y = \nu + \Lambda\eta + \varepsilon$, $\varepsilon \sim N(0, \Theta)$, with the residual variances that `lavInspect(fit, "theta")` reports (observed outcomes included, observed covariates excluded), except that the drawn indicators of a composite keep its weighted sum. `"ymis"`, `"ovmis"` Imputed values for missing observations, drawn from the conditional distribution $y_{mis} \mid y_{obs}, \theta$.
`newdata`	An optional data frame of new observations. If supplied, predictions are computed for `newdata` rather than the original training data. Not supported for `type = "ymis"`.
`level`	Integer; for `type = "lv"` in two-level models, specifies whether level 1 or level 2 latent variables are desired (default `1L`). Other types ignore it: for two-level models, `"yhat"` and `"ypred"` give the total within plus between prediction, and a `"ypred"` draw keeps its cluster's between-level values, so only the observation-level residuals are new. For a random-slope model (see `inlavaan()`), each level 2 latent variable, slopes included, is drawn on its own, so draws of different variables are independent given the parameters, and level 1 values are conditional means. Its `"yhat"` and `"ypred"` use these empirical Bayes values (closed-form route only), and `"ypred"` adds the level-1 and between-level residuals.
`nsamp`	Integer; number of posterior samples to use for prediction. Defaults to `1000`.
`ymis_only`	Logical; only applies when `type = "ymis"`. When `TRUE`, returns only the imputed values as a named numeric vector per sample (names of the form `"varname[rowindex]"`, matching the blavaan convention). When `FALSE` (default), returns the full data matrix with missing values filled in.

summary — Logical. When TRUE, collapse the posterior draws with summary() and return summary statistics (mean, SD, quantiles, mode) instead of the raw draws – equivalent to calling summary(predict(object, ...)) but without materialising the intermediate draws object. Default FALSE.

... — Currently unused.

Value

A list of nsamp posterior draws, each a matrix (or data frame, for multiple groups) with rows corresponding to cases and columns to variables or latent factors. When summary = TRUE, instead returns a summary.predict.inlavaan_internal object with the posterior mean, SD, quantiles, and mode for each case/variable.

See Also

sampling(), simulate(), summary()

Examples

```
HS.model <- "
  visual  =~ x1 + x2 + x3
  textual =~ x4 + x5 + x6
  speed   =~ x7 + x8 + x9
"
utils::data("HolzingerSwineford1939", package = "lavaan")
fit <- acfa(HS.model, HolzingerSwineford1939, std.lv = TRUE, nsamp = 100,
            test = "none", verbose = FALSE)

# Posterior latent variable scores
lv_scores <- predict(fit)
head(lv_scores)

# Predicted observed variable means
yhat <- predict(fit, type = "yhat")
head(yhat)

# Point estimates only, skipping the manual summary() step
predict(fit, type = "yhat", summary = TRUE)
```

---

priors_for *Specify priors for a SEM*

Description

Specify priors for a SEM, similar to how blavaan::dpriors() works.

Usage

```
priors_for(...)
```

Arguments

...	Named arguments specifying prior distributions for lavaan parameter types.

Details

This function provides a convenient way to specify prior distributions for different types of parameters in a structural equation model (SEM). It uses a registry of default priors for common lavaan parameter types (e.g., loadings, regressions, residuals, etc.) and allows users to override these defaults by passing named arguments.

The parameter names, and default settings, are:

- `nu` = `"normal(0,32)"`: Observed variable intercepts
- `alpha` = `"normal(0,10)"`: Latent variable intercepts
- `lambda` = `"normal(0,10)"`: Factor loadings
- `beta` = `"normal(0,10)"`: Regression coefficients
- `wmat` = `"normal(0,10)"`: Composite weights (<~)
- `theta` = `"gamma(1,.5)[sd]"`: Residual variances (prior on the standard deviation)
- `psi` = `"gamma(1,.5)[sd]"`: Latent variable variances (prior on the standard deviation)
- `rho` = `"beta(1,1)"`: Correlations (both latent and observed)
- `tau` = `"normal(0,1.5)"`: Thresholds for ordinal variables

Note that the normal distributions are parameterised using standard deviations, and not variances. For example, `normal(0,10)` means a normal distribution with mean 0 and standard deviation 10 (not variance 10).

Value

A named character vector of prior specifications, where names correspond to lavaan parameter types (e.g., "lambda", "beta", "theta", etc.) and values are character strings specifying the prior distribution (e.g., `"normal(0,10)"`, `"gamma(1,0.5)[sd]"`, `"gamma(1,1)[prec]"`, etc.).

Scale qualifiers

For variance parameters (`theta`, `psi`), the prior distribution can be placed on a transformed scale by appending a qualifier:

- `[sd]`: Prior is on the standard deviation σ. Example: `"gamma(1,0.5)[sd]"` places a Gamma(1, 0.5) prior on $\sigma = \sqrt{\text{variance}}$.
- `[prec]`: Prior is on the precision $\tau = 1/\sigma^2$. Example: `"gamma(1,1)[prec]"` places a Gamma(1, 1) prior on $\tau = 1/\text{variance}$. This is the parameterisation used by blavaan and corresponds to an Inverse-Gamma prior on the variance.

The necessary Jacobian adjustment is applied automatically in both cases.

See Also

inlavaan(), acfa(), asem(), agrowth()

Examples

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

# Precision-scale prior for residual variances (blavaan-style)
priors_for(theta = "gamma(1,1)[prec]")
```

---

qsnorm_fast *Fast Approximation of Skew-Normal Quantile Function*

Description

A fast approximation of skew-normal quantiles using the high-performance approximation algorithm from the INLA GMRFLib C source, and originally by Thomas Luu (see details for reference).

Usage

```
qsnorm_fast(p, xi = 0, omega = 1, alpha = 0)
```

Arguments

p	Vector of probabilities.
xi	Location parameter (numeric vector).
omega	Scale parameter (numeric vector).
alpha	Shape parameter (numeric vector).

Details

This function implements a high-performance approximation for the skew-normal quantile function based on the algorithm described by Luu (2016). The method uses a domain decomposition strategy to achieve high accuracy ($< 10^{-7}$ relative error) without iterative numerical inversion.

The domain is split into two regions:

- **Tail Regions:** For extreme probabilities where $|u|$ is large, the quantile is approximated using the Lambert W-function, $W(z)$, solving $z = \Phi(q)$ via asymptotic expansion:

$$q \approx \sqrt{2W\left(\frac{1}{2\pi(1-p)^2}\right)}$$

- **Central Region:** For the main body of the distribution, the function uses a high-order Taylor expansion of the inverse error function around a carefully selected expansion point x_0:

$$\Phi^{-1}(p) \approx \sum_{k=0}^{5} c_k (z - x_0)^k$$

This approach is significantly faster than standard numerical inversion (e.g., uniroot) while maintaining sufficient precision for most statistical applications.

Value

Vector of quantiles.

References

Luu, T. (2016). *Fast and accurate parallel computation of quantile functions for random number generation* (Doctoral thesis). UCL (University College London). https://discovery.ucl.ac.uk/1482128/

Examples

```
qsnorm_fast(c(0.025, 0.5, 0.975))
qsnorm_fast(c(0.025, 0.5, 0.975), xi = 2, omega = 0.5, alpha = 1)
```

`residuals` *Model Residuals for INLAvaan Models*

Description

Extract the difference between the observed and model-implied (fitted) sample statistics from a fitted `INLAvaan` model. As in **lavaan** and **blavaan**, residuals are computed at the parameter estimates – here the posterior means – not as a posterior distribution over residuals.

Usage

```
## S4 method for signature 'INLAvaan'
residuals(object, type = "raw", labels = TRUE, ..., per_cluster = FALSE)

## S4 method for signature 'INLAvaan'
resid(object, type = "raw", ..., per_cluster = FALSE)
```

Arguments

`object`	An object of class INLAvaan.
`type`	Character. `"raw"` (default) returns the unscaled difference between the observed and model-implied covariance matrix (and mean vector, when a mean structure is present). `"cor"` (or `"cor.bollen"`) first rescales both matrices to a correlation matrix. `"cor.bentler"` rescales both by the observed variances (the basis of the SRMR). `"normalized"` and `"standardized"` divide the raw residuals by their asymptotic standard errors. `"casewise"` (aliases `"case"`, `"obs"`, `"observations"`, `"ov"`) returns observed-minus-fitted values for each observation.
`labels`	Logical. Attach variable names to the output. Default `TRUE`.
`...`	Currently unused.
`per_cluster`	Logical. For a random-slope model, return the residuals of each cluster instead of the averaged ones. Default `FALSE`.

Details

This delegates to **lavaan**'s own `residuals()` machinery, so the return structure matches lavaan exactly. Because INLAvaan stores the posterior means as the point estimates of the fitted object, the residuals are the observed statistics minus the posterior-mean model-implied statistics (mirroring **blavaan**, which likewise inherits lavaan's `residuals()` without overriding it).

For a model with random slopes (lavaan's `rv()`), the residuals compare the sample moments with the averaged moments of `fitted()`. With `per_cluster = TRUE`, they compare each cluster's sample mean and within-cluster covariance with its expected ones. A covariance of two variables observed together in fewer than two rows of a cluster is `NA`. The raw and correlation types are available, and `type = "casewise"` gives the observed values minus the outcomes' means given the covariates (closed-form route only).

Value

For moment-based `types`, a list with elements `type`, `cov`, and (when relevant) `mean`. For `type = "casewise"`, a numeric matrix of observed-minus-fitted values.

See Also

`fitted()`, `predict()`, fitMeasures()

Examples

```
HS.model <- "
  visual  =~ x1 + x2 + x3
  textual =~ x4 + x5 + x6
  speed   =~ x7 + x8 + x9
"
utils::data("HolzingerSwineford1939", package = "lavaan")
fit <- acfa(HS.model, HolzingerSwineford1939, std.lv = TRUE, nsamp = 100,
            test = "none", verbose = FALSE)

# Raw residual covariance matrix (posterior means)
residuals(fit)

# SRMR-basis residuals
residuals(fit, type = "cor.bentler")

# Casewise observed-minus-fitted values
head(residuals(fit, type = "casewise"))
```

`sampling` *Draw Samples from the Generative Model*

Description

Sample model parameters, latent variables, or observed variables from the generative model underlying a fitted INLAvaan model. By default, parameters are drawn from the **posterior** distribution; set `prior = TRUE` to draw from the **prior** instead (useful for prior predictive checks).

Usage

```
sampling(object, ...)

## S4 method for signature 'INLAvaan'
sampling(
  object,
  type = c("lavaan", "theta", "latent", "observed", "implied", "all"),
  nsamp = 1000L,
  samp_copula = TRUE,
  prior = FALSE,
  silent = FALSE,
  ...
)
```

Arguments

`object`	An object of class INLAvaan (or `inlavaan_internal`).
`...`	Additional arguments (currently unused).
`type`	Character string specifying what to sample:

`"lavaan"` (Default) The lavaan-side (constrained) model parameters. Returns an `nsamp` by `npar` matrix.

`"theta"` The INLAvaan-side unconstrained parameters. Returns an `nsamp` by `npar` matrix.

`"latent"` Latent variables from the model-implied distribution. Returns an `nsamp` by `nlv` matrix (one draw per posterior sample, not tied to any individual). For two-level models the matrix holds the within- *and* between-level latent variables, the level-2 columns carrying the `.l2` suffix when the same latent variable also exists at level 1.

`"observed"` Observed variables generated from the full model. Returns an `nsamp` by `nobs_vars` matrix. For two-level models each row is a draw from the two-level generative model, $\mathbf{y} = \mathbf{y}^B + \mathbf{y}^W$: variables that live at both levels sum their between- and within-level draws, and within-only or between-only variables take the single level available to them.

`"implied"` Model-implied moments. Returns a length-`nsamp` list, each element a list with `cov` (model-implied covariance matrix) and, when `meanstructure = TRUE`, `mean` (model-implied mean vector). For multi-group models each element is itself a list of groups. For two-level models each element is a list with a `within` and a `cluster` block, each holding a `cov` and a `mean`, as `lavaan::lavInspect()` reports them.

`"all"` A named list with elements `lavaan`, `theta`, `latent`, `observed`, and `implied`.

`nsamp`	Number of samples to draw.
`samp_copula`	Logical. When `TRUE` (default), posterior parameter samples use the copula method with the fitted marginals. When `FALSE`, samples are drawn from the joint Gaussian (Laplace) approximation. Ignored when `prior = TRUE`.
`prior`	Logical. When `TRUE`, parameters are drawn from the prior distribution and then propagated through the generative model. When `FALSE` (default), parameters come from the posterior.
`silent`	Logical. When `TRUE`, suppresses the informational message about rejected non-PD draws during prior rejection sampling. Default `FALSE`.

Details

Each row of the output corresponds to a **fresh parameter draw**: a new $\boldsymbol{\theta}^{(s)}$ is sampled and then propagated through the generative chain to produce one latent vector and one observed vector. This makes `sampling()` ideal for **prior and posterior predictive checks** (e.g., density overlays, test statistic distributions).

The generative chain is:

$$\boldsymbol{\theta}^{(s)} \sim \pi(\boldsymbol{\theta} \mid \mathbf{y})$$
$$\boldsymbol{\eta}^{(s)} \sim N((\mathbf{I} - \mathbf{B})^{-1}\boldsymbol{\alpha},\, \boldsymbol{\Phi})$$
$$\mathbf{y}^{*(s)} \sim N(\boldsymbol{\Lambda}\boldsymbol{\eta}^{(s)} + \boldsymbol{\nu},\, \boldsymbol{\Theta})$$

If you need **complete replicate datasets** (many observations from a single parameter draw) — for example, for simulation-based calibration (SBC) — use `simulate()` instead.

This is distinct from `predict()`, which computes individual-specific factor scores $\boldsymbol{\eta} \mid \mathbf{y}, \boldsymbol{\theta}$ conditional on observed data.

For a random-slope model, each draw takes the level-2 variables, slopes included, first and puts the drawn slopes into their level-1 paths, and `type = "implied"` gives the moments averaged over the covariates (see `fitted()`). As for other models, the covariates of a latent or observed draw come from their fitted moments, whereas `simulate()` keeps the observed covariates of a random-slope model.

Value

A matrix or named list, depending on `type`.

See Also

`simulate()` for generating complete replicate datasets (e.g., for SBC); `predict()` for individual-specific factor scores; `bfit_indices()` for Bayesian fit indices.

Examples

```
utils::data("HolzingerSwineford1939", package = "lavaan")
fit <- acfa("visual =~ x1 + x2 + x3", HolzingerSwineford1939)

# Posterior samples of lavaan-side parameters
samps <- sampling(fit, nsamp = 500)
```

```
head(samps)

# Compare copula vs Gaussian sampling
s_cop <- sampling(fit, nsamp = 500, samp_copula = TRUE)
s_gaus <- sampling(fit, nsamp = 500, samp_copula = FALSE)

# Prior predictive samples
y_prior <- sampling(fit, type = "observed", nsamp = 500, prior = TRUE)
```

`simulate` *Simulate Datasets from the Generative Model*

Description

Generate complete synthetic datasets from a fitted INLAvaan model. For each simulation, a single parameter vector is drawn (from the posterior or prior), and then `sample.nobs` observations are generated from the model-implied distribution at that parameter value.

Usage

```
## S4 method for signature 'INLAvaan'
simulate(
  object,
  nsim = 1L,
  seed = NULL,
  sample.nobs = NULL,
  prior = FALSE,
  samp_copula = TRUE,
  silent = FALSE,
  ...
)
```

Arguments

`object`	An object of class INLAvaan.
`nsim`	Number of replicate datasets to generate (default 1).
`seed`	Optional random seed (passed to `set.seed()`).
`sample.nobs`	Number of observations per dataset, per group for multigroup models (a single number is used for every group). Defaults to the sample size of the original data.
`prior`	Logical. When `TRUE`, parameters are drawn from the prior; when `FALSE` (default), from the posterior.
`samp_copula`	Logical. When `TRUE` (default) and `prior = FALSE`, posterior parameter draws use the copula method. Ignored when `prior = TRUE`.
`silent`	Logical. When `TRUE`, suppresses the informational message about rejected non-PD draws. Default `FALSE`.
`...`	Additional arguments (currently unused).

Details

This function is designed for tasks that require **full replicate datasets** from a single parameter draw, such as simulation-based calibration (SBC) and posterior predictive p-values. It differs from `sampling()` which generates one observation per parameter draw (useful for prior/posterior predictive density overlays).

For each simulation $s = 1, \ldots, S$:

1. Draw $\boldsymbol{\theta}^{(s)}$ from the posterior (or prior).
2. Compute the model-implied covariance $\boldsymbol{\Sigma}(\boldsymbol{\theta}^{(s)})$. If it is not positive-definite, reject and redraw.
3. Generate a dataset of `sample.nobs` rows from $N(\boldsymbol{\mu}(\boldsymbol{\theta}^{(s)}),\ \boldsymbol{\Sigma}(\boldsymbol{\theta}^{(s)}))$.

Parameter draws reuse the same internal machinery as `sampling()` (`sample_params_prior` / `sample_params_posterior`), so the prior specification is consistent.

A two-level dataset keeps the clusters of the data, with their labels and sizes, unless `sample.nobs` is given. For a random-slope model (closed-form route only), each cluster keeps its own covariates and size, its slopes and other level-2 effects are drawn first, and then its outcomes. `sample.nobs` is not available. Other models draw their covariates from their fitted moments.

Value

A list of length `nsim`. Each element is a data frame with `sample.nobs` rows and two attributes:

- `"truth"` — named numeric vector of lavaan-side (x-space, constrained) parameter values used to generate the dataset.
- `"truth_theta"` — named numeric vector of the corresponding unconstrained (theta-space) parameter values.

See Also

`sampling()` for single-observation draws from the predictive distribution (prior/posterior predictive checks).

Examples

```
utils::data("HolzingerSwineford1939", package = "lavaan")
fit <- acfa("visual =~ x1 + x2 + x3", HolzingerSwineford1939)

# Simulate one replicate dataset from the posterior
sims <- simulate(fit, nsim = 1)
head(sims[[1]])                    # data frame
attr(sims[[1]], "truth")           # true lavaan-side (x-space) parameters
attr(sims[[1]], "truth_theta")     # corresponding unconstrained (theta-space) parameters

# Simulate from the prior (e.g., for SBC)
sims_prior <- simulate(fit, nsim = 5, prior = TRUE)
lapply(sims_prior, nrow)
```

`standardisedsolution` *Standardised solution of a latent variable model*

Description

Standardised solution of a latent variable model

Usage

```
standardisedsolution(
  object,
  type = "std.all",
  se = TRUE,
  ci = TRUE,
  level = 0.95,
  postmedian = FALSE,
  postmode = FALSE,
  cov.std = TRUE,
  remove.eq = TRUE,
  remove.ineq = TRUE,
  remove.def = FALSE,
  nsamp = 250,
  ...
)

standardisedSolution(
  object,
  type = "std.all",
  se = TRUE,
  ci = TRUE,
  level = 0.95,
  postmedian = FALSE,
  postmode = FALSE,
  cov.std = TRUE,
  remove.eq = TRUE,
  remove.ineq = TRUE,
  remove.def = FALSE,
  nsamp = 250,
  ...
)

standardizedsolution(
  object,
  type = "std.all",
  se = TRUE,
  ci = TRUE,
  level = 0.95,
```

```
  postmedian = FALSE,
  postmode = FALSE,
  cov.std = TRUE,
  remove.eq = TRUE,
  remove.ineq = TRUE,
  remove.def = FALSE,
  nsamp = 250,
  ...
)

standardizedSolution(
  object,
  type = "std.all",
  se = TRUE,
  ci = TRUE,
  level = 0.95,
  postmedian = FALSE,
  postmode = FALSE,
  cov.std = TRUE,
  remove.eq = TRUE,
  remove.ineq = TRUE,
  remove.def = FALSE,
  nsamp = 250,
  ...
)
```

Arguments

object	An object of class INLAvaan.
type	Character (dots and underscores are interchangeable: "std.all" and "std_all" are the same type). If "std_lv", the standardized estimates are based on the variances of the (continuous) latent variables only. If "std_all", the standardized estimates are based on the variances of both (continuous) observed and latent variables. If "std_nox", the standardized estimates are based on the variances of both (continuous) observed and latent variables, but not the variances of exogenous covariates. Note that "std_nox" only differs from "std_all" if fixed_x = TRUE; if fixed_x = FALSE, the exogenous covariates are treated as random variables (and standardized) just like any other variable, and a warning is issued. Alternatively, type may be a vector of (observed) variable names (for example type = c("x1", "x2")); in that case only the parameters involving these variables are standardized (the other observed variables are left unstandardized). This is a generalization of "std_nox", where the (observed) exogenous x variables are the ones left unstandardized.
se	Logical. If TRUE, standard errors for the standardized parameters will be computed, together with a z-statistic and a p-value.
ci	If TRUE, confidence intervals are added to the output.
level	The confidence level required.

postmedian	Logical; if TRUE, include posterior median in estimates.
postmode	Logical; if TRUE, include posterior mode in estimates.
cov.std	Logical. If TRUE, the (residual) observed covariances are scaled by the square root of the Theta diagonal elements, and the (residual) latent covariances are scaled by the square root of the Psi diagonal elements. If FALSE, the (residual) observed covariances are scaled by the square root of the diagonal elements of the model-implied observed covariance matrix, and the (residual) latent covariances are scaled similarly using the model-implied covariance matrix of the latent variables. Documented explicitly here (rather than inherited) because lavaan >= 0.7-1 renamed this and the next three arguments to snake_case.
remove.eq	Logical. If TRUE, filter the output by removing all rows containing equality constraints, if any.
remove.ineq	Logical. If TRUE, filter the output by removing all rows containing inequality constraints, if any.
remove.def	Logical. If TRUE, filter the output by removing all rows containing parameter definitions, if any.
nsamp	The number of samples to draw from the approximate posterior distribution for the calculation of standardised estimates.
...	Additional arguments sent to lavaan::standardizedSolution().

Details

For a model with random slopes (lavaan's rv()), the estimates are scaled by the implied variances averaged over the covariates, which include the mean and the variance of each slope. The level-1 row that carries a slope gives the standardised mean slope. The slope's own rows are on the same standardised-slope scale: its intercept is a standardised slope, and its (residual) variance is the share of the outcome's level-1 variance that the slope's (residual) variation adds.

Value

A data.frame containing standardised model parameters.

See Also

summary(), coef(), vcov()

Examples

```
HS.model <- "
  visual  =~ x1 + x2 + x3
  textual =~ x4 + x5 + x6
"
utils::data("HolzingerSwineford1939", package = "lavaan")

# Fit a CFA model with standardised latent variables
fit <- acfa(
  HS.model,
  data = HolzingerSwineford1939,
```

```
  test = "none",
  nsamp = 10,
  vb_correction = FALSE,
  verbose = FALSE
)
standardisedsolution(fit, nsamp = 10, se = FALSE, ci = FALSE)
```

`timing` *Timing Information for INLAvaan Models*

Description

Extract wall-clock timings for individual computation stages of a fitted `INLAvaan` model.

Usage

```
timing(object, ...)

## S4 method for signature 'INLAvaan'
timing(object, what = "total", ...)
```

Arguments

`object`	An object of class INLAvaan.
`...`	Currently unused.
`what`	Character vector of timing segment names to return, or `"all"` to return every segment. Defaults to `"total"`. Available segments (depending on model options): `"init"` (which includes the lavaan model setup), `"optim"`, `"vb"`, `"loglik"`, `"marginals"`, `"norta"`, `"sampling"`, `"covariances"`, `"definedpars"`, `"deltapars"`, `"test"`, `"loo"`, `"waic"`, `"total"`. The segments are disjoint and `"total"` is their sum. `"loo"` and `"waic"` are recorded only when the fit computed them, i.e. `test` (see `inlavaan()`) included `"loo"`, `"waic"` or `"full"` and the model is supported (see `loo()`); requesting either otherwise gives an error explaining why, distinct from requesting a misspelled segment name.

Value

A named numeric vector (class `c("timing.INLAvaan", "numeric")`) of elapsed times in seconds. Printing formats short durations as seconds, longer ones as minutes or hours.

See Also

`diagnostics()`, `summary()`

Examples

```
HS.model <- "
  visual  =~ x1 + x2 + x3
  textual =~ x4 + x5 + x6
  speed   =~ x7 + x8 + x9
"
utils::data("HolzingerSwineford1939", package = "lavaan")
fit <- acfa(HS.model, HolzingerSwineford1939, std.lv = TRUE, nsamp = 100,
            test = "none", verbose = FALSE)

# Total elapsed time
timing(fit)

# All stages
timing(fit, what = "all")

# Specific stages
timing(fit, what = c("optim", "marginals"))
```

`update` *Update and re-fit an INLAvaan model*

Description

Re-fit an INLAvaan model with modified arguments, in the style of `stats::update()`. This is convenient for prior-sensitivity analyses (vary `dp` or the `prior()` modifiers), iterative respecification (swap the `model` or `add` syntax), and swapping approximation settings (e.g. `marginal_method`, `nsamp`) without retyping the whole call.

Usage

```
## S4 method for signature 'INLAvaan'
update(object, model, add, ..., evaluate = TRUE)
```

Arguments

`object`	An object of class INLAvaan.
`model`	Optional replacement model, in the same form accepted by `inlavaan()` (lavaan model syntax or a parameter table).
`add`	Optional lavaan syntax appended to the original model. Requires the original model to have been specified as a syntax string, and is ignored when `model` is supplied.
`...`	Further arguments passed to `inlavaan()`, overriding the values in the original call (e.g. `dp`, `marginal_method`, `nsamp`, `data`).
`evaluate`	Logical. If `TRUE` (default) the updated call is evaluated and the re-fitted INLAvaan object is returned; if `FALSE` the updated, unevaluated call is returned for inspection.

Details

The method edits the recorded `inlavaan()` call, replacing `model` (or extending it via `add`) and overriding any argument supplied through `...`, then re-evaluates it. Because INLAvaan extends the **lavaan** class, a dedicated method is required: without it, `update()` would dispatch to **lavaan**'s method and silently return a frequentist **lavaan** fit instead of a Bayesian one.

When the parameter structure is preserved (no change to `model`, `add`, or `model.type`), the previous posterior mode is passed as the optimiser's starting value (a warm start), so mode-finding is typically much faster than a cold fit. This makes prior sweeps particularly cheap. The warm start only changes the optimiser's starting point, not the fitted result. Supply `start` explicitly to override it.

Fits produced by INLAvaan versions that did not record their call cannot be updated; re-fit once with `inlavaan()` (or `acfa()`, `asem()`, `agrowth()`) and `update()` will work on the new object.

Value

A re-fitted INLAvaan object, or the updated call when `evaluate = FALSE`.

See Also

`inlavaan()`, `compare()`

Examples

```
## Not run:
model <- "visual =~ x1 + x2 + x3"
fit <- acfa(model, data = lavaan::HolzingerSwineford1939)

# Prior sensitivity: refit under a tighter loading prior (warm-started)
fit2 <- update(fit, dp = priors_for(lambda = "normal(0,1)"))

# Respecify: add a residual covariance
fit3 <- update(fit, add = "x1 ~~ x2")
compare(fit, fit3)

# Swap approximation settings only
fit4 <- update(fit, marginal_method = "sampling", nsamp = 4000)

## End(Not run)
```

---

vcov *Variance-Covariance Matrix for INLAvaan Models*

Description

Extract the posterior variance-covariance matrix of model parameters from a fitted `INLAvaan` model.

Usage

```
## S4 method for signature 'INLAvaan'
vcov(object, type = c("lavaan", "theta"), ...)
```

Arguments

object	An object of class INLAvaan.
type	Character. "lavaan" (default) returns the posterior covariance matrix of the model parameters computed from posterior samples, with one row per entry of coef() (matching lavaan output). "theta" returns the Laplace approximation covariance in the internal parameterisation, with one row per distinct free parameter.
...	Currently unused.

Value

A square numeric matrix.

See Also

summary(), coef(), standardisedsolution()

Examples

```
HS.model <- "
  visual  =~ x1 + x2 + x3
  textual =~ x4 + x5 + x6
  speed   =~ x7 + x8 + x9
"
utils::data("HolzingerSwineford1939", package = "lavaan")
fit <- acfa(HS.model, HolzingerSwineford1939, std.lv = TRUE, nsamp = 100,
            test = "none", verbose = FALSE)

# Default: posterior covariance of lavaan parameters
vcov(fit)

# Internal parameterisation (Laplace approximation)
vcov(fit, type = "theta")
```

---

waic *Widely Applicable Information Criterion for INLAvaan Models*

Description

Computes the WAIC of a fitted INLAvaan model in closed form from the fit's Laplace summary – the same per-unit Taylor quantities behind loo(), with no posterior draws and no Monte Carlo error. Single-level models are scored per subject; two-level models are scored per cluster by default, matching the units used by loo(). For a two-level model type = "loso" instead scores the *conditional* (leave-one-unit-out) WAIC; see Details.

Usage

```
waic(x, ...)

## S3 method for class 'INLAvaan'
waic(
  x,
  type = c("auto", "loso", "loco"),
  units = NULL,
  second_order = TRUE,
  cores = NULL,
  verbose = FALSE,
  ...
)

## S3 method for class 'inlavaan_internal'
waic(
  x,
  type = c("auto", "loso", "loco"),
  units = NULL,
  second_order = TRUE,
  cores = NULL,
  verbose = FALSE,
  ...
)

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

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

Arguments

x	A fitted INLAvaan object (or its inlavaan_internal list).
...	Not used.
type	Unit type: "auto" (default) resolves to per-subject for single-level models and per-cluster (marginal WAIC) for two-level models. "loso" on a two-level model scores the conditional (leave-one-unit-out) WAIC instead (with a warning; see Details); "loco" cannot be forced on a model without clusters.
units	Optional integer vector of unit indices to score; defaults to all units.
second_order	Logical; include the second-order (Hessian) terms (default TRUE). FALSE gives the first-order WAIC, which equals the first-order LOO exactly – and inherits its bias: a first-order score overstates the elpd by $\frac{1}{2}p_D$ in the limit, so candidates of different dimension cannot be compared on it (see loo()).
cores	Number of cores for differentiating the unit scores. The default NULL runs serially; parallelism must be requested explicitly.
verbose	Logical; print progress (default FALSE).

`object` A fitted INLAvaan object, or an `inlavaan_waic` result.

Details

Writing $\ell_u(\theta) = \log p(y_u \mid \theta)$ and expanding it to second order about the posterior mode, with the posterior taken as $N(\theta^*, \Omega)$, both WAIC terms are available in closed form: the pointwise log predictive density lpd_u is the same Gaussian integral `loo()` computes, and the penalty is the polynomial

$$p_{\mathrm{waic},u} = \mathrm{Var}[\ell_u(\theta)] = s_u^\top \Omega\, s_u + \tfrac{1}{2}\mathrm{tr}\big[(H_u\Omega)^2\big]\,,$$

with s_u and H_u the unit's score and Hessian. Then $\mathrm{elpd}_{\mathrm{waic}} = \sum_u(\mathrm{lpd}_u - p_{\mathrm{waic},u})$ and $\mathrm{WAIC} = -2\,\mathrm{elpd}_{\mathrm{waic}}$.

Existence. $p_{\mathrm{waic},u}$ is a polynomial in the posterior moments, so unlike the lpd and log CPO integrals of `loo()` it is finite for every unit and carries no condition of its own. The second-order WAIC therefore exists exactly where its lpd term does: where $\Omega^{-1} - H_u$ is positive definite, equivalently $k_{\min} > -1$ for the spectrum k of $-\Omega H_u$. The log CPO condition $k_{\max} < 1$ is irrelevant here, since the WAIC reads no case-deletion term: a unit whose deleted posterior is improper can still carry an exact second-order WAIC. Where the lpd term fails, every estimate is reported at *first order over all units* and warns – $\mathrm{elpd}_{\mathrm{waic}}$ is a headline predictive score, and mixing two Taylor orders within one reported number is exactly what `loo()` refuses for $\mathrm{elpd}_{\mathrm{loo}}$ (the mixed alternative is reserved for `p_loo`, a secondary diagnostic). That fallback is exact rather than merely lower-order: the identity $\mathrm{lpd}_u^{(1)} - p_{\mathrm{waic},u}^{(1)} = \log \mathrm{CPO}_u^{(1)}$ holds pointwise, so the first-order WAIC *is* the first-order LOO score. No other threshold is applied: as in `loo()`, existence is the only condition the package acts on.

The same model restrictions as `loo()` apply, and so does the flavour rule: fits with `fixed.x = TRUE` are scored conditionally on the exogenous covariates, fits with `fixed.x = FALSE` jointly (see `loo()`). For composites, p_{waic} also counts the indicator (co)variances that lavaan fixes at their sample values, as `p_loo` does.

Marginal vs conditional WAIC (two-level models). The default per-cluster scoring is the *marginal* WAIC, which corresponds to leave-one-cluster-out cross-validation – prediction for a *new* cluster. Setting `type = "loso"` scores the *conditional* WAIC, corresponding to leave-one-unit-out – prediction for a new observation within an *observed* cluster. The two answer different questions and are easily conflated (Merkle, Furr & Rabe-Hesketh, 2019); the per-cluster marginal is the usual model-comparison target, so it is the default, and `type = "loso"` warns. This matches `loo(type = "loso")` – the two read the same estimand off the same expansion.

When `inlavaan()`'s `test` includes `"loo"` or `"waic"` (e.g. `test = "full"`), the WAIC is derived at fit time from the same Taylor pass as the LOO at no extra cost and stored with the fit: `waic(fit)` then returns the stored result when called with default arguments, and `fitmeasures()` reports `waic`, `p_waic`, `se_waic` as part of `"all"` for free. Under the default `test = "standard"` nothing is stored and `waic(fit)` computes it on demand. If the `loo` package is attached it masks this generic, but dispatch on INLAvaan objects continues to work.

Value

An object of class `inlavaan_waic`: a list with `per_unit` (pointwise `lpd`, `p_waic`, `elpd_waic`, with the same `unit`/`group` identification as `loo()`), `estimates` (matrix with rows `elpd_waic`, `p_waic`, `waic` and columns `Estimate`, `SE`), `type`, `flavour`, `n_units`, `n_groups`, `n_lpd_ok` (units whose second-order lpd exists), `second_order` (whether it was requested) and `use_second` (whether it

was used). summary() is an alias for print(): it prints the same output and returns the result invisibly.

See Also

loo(), fitmeasures()

Examples

```
HS.model <- '
  visual  =~ x1 + x2 + x3
  textual =~ x4 + x5 + x6
  speed   =~ x7 + x8 + x9
'
utils::data("HolzingerSwineford1939", package = "lavaan")
fit <- acfa(HS.model, HolzingerSwineford1939, meanstructure = TRUE)
waic(fit)
```

Index

acfa, 3
acfa(), *6, 7*, *11*, *16*, *35*, *39*, *41*, *53*, *65*
add_loo *(*loo*)*, 44
add_loo(), *4*, *8*, *13*, *20*, *28*, *35*, *46*
agrowth, 7
agrowth(), *7*, *11*, *16*, *35*, *39*, *41*, *53*, *65*
AIC, *43*
AIC,INLAvaan-method *(*logLik*)*, 42
anova *(*compare*)*, 19
anova(), *43*
anova,INLAvaan-method *(*compare*)*, 19
as_fun_string, 16
asem, 12
asem(), *7*, *11*, *15, 16*, *35*, *39*, *41*, *53*, *65*

bfit_indices, 17
bfit_indices(), *19, 20*, *28*, *39*, *57*
BIC, *43*
BIC,INLAvaan-method *(*logLik*)*, 42
blavaan::blavFitIndices(), *17, 18*
blavaan::dpriors(), *51*

coef *(*INLAvaan-class*)*, 40
coef(), *30*, *62*, *66*
coef,INLAvaan-method *(*INLAvaan-class*)*, 40
compare, 19
compare(), *18*, *23*, *28*, *39*, *43*, *46, 47*, *65*
compare,INLAvaan-method *(*compare*)*, 19

dbeta_box, 21
deviance, 22
deviance(), *4*, *8*, *13*, *35*, *43*
diagnostics, 23
diagnostics(), *4*, *9*, *13*, *28*, *33*, *36*, *49*, *63*
diagnostics,INLAvaan-method *(*diagnostics*)*, 23
dsnorm, 26

fit_skew_normal, 30
fit_skew_normal_samp, 32
fitmeasures, 27
fitMeasures(), *19, 20*, *30*, *55*
fitmeasures(), *4*, *8*, *13*, *18*, *20*, *26*, *35*, *47*, *68, 69*
fitMeasures,INLAvaan-method *(*fitmeasures*)*, 27
fitmeasures,INLAvaan-method *(*fitmeasures*)*, 27
fitted, 29
fitted(), *38*, *55*, *57*
fitted,INLAvaan-method *(*fitted*)*, 29
fitted.values,INLAvaan-method *(*fitted*)*, 29

get_inlavaan_internal, 33

INLAvaan, *17–19*, *22*, *24*, *28, 29*, *33*, *43–45*, *49, 50*, *54*, *56*, *58*, *61*, *63–68*
inlavaan, 34
inlavaan(), *4*, *6*, *9*, *11*, *13*, *15*, *20*, *25*, *28*, *35*, *40, 41*, *46, 47*, *50*, *53*, *63–65*, *68*
INLAvaan-class, 40
is_same_function, 42

lavaan, *6, 7*, *11*, *15, 16*, *37, 38*, *40, 41*
lavaan::fitMeasures(), *18*
lavaan::lavInspect(), *56*
lavaan::lavOptions(), *7*, *16*
logLik, 42
logLik(), *4*, *8*, *13*, *23*, *35*
logLik,INLAvaan-method *(*logLik*)*, 42
loo, 44
loo(), *4*, *8, 9*, *13*, *20*, *23*, *28*, *35*, *39*, *43*, *63*, *66–69*

model.syntax, *3*, *8*, *13*, *34*

nobs *(*INLAvaan-class*)*, 40
nobs,INLAvaan-method *(*INLAvaan-class*)*, 40

parallel::mclapply(), *6*, *10*, *15*, *37*
plot, 48
plot(), *26*
plot,INLAvaan,ANY-method *(*plot*)*, 48
predict, 49
predict(), *30*, *39*, *55*, *57*
predict,INLAvaan-method *(*predict*)*, 49
print.bfit_indices *(*bfit_indices*)*, 17
print.inlavaan_loo *(*loo*)*, 44
print.inlavaan_waic *(*waic*)*, 66
priors_for, 51
priors_for(), *4*, *8*, *13*, *35*

qsnorm_fast, 53

resid,INLAvaan-method *(*residuals*)*, 54
residuals, 54
residuals(), *38*
residuals,INLAvaan-method *(*residuals*)*, 54

sampling, 55
sampling(), *39*, *51*, *59*
sampling,INLAvaan-method *(*sampling*)*, 55
set.seed(), *58*
show *(*INLAvaan-class*)*, 40
show,INLAvaan-method *(*INLAvaan-class*)*, 40
simulate, 58
simulate(), *39*, *51*, *57*
simulate,INLAvaan-method *(*simulate*)*, 58
standardisedSolution *(*standardisedsolution*)*, 60
standardisedsolution, 60
standardisedsolution(), *39*, *66*
standardizedSolution *(*standardisedsolution*)*, 60
standardizedsolution *(*standardisedsolution*)*, 60
stats::AIC(), *43*
stats::update(), *64*
summary *(*INLAvaan-class*)*, 40
summary(), *4*, *8*, *13*, *18*, *35*, *38*, *39*, *49*, *51*, *62*, *63*, *66*
summary,INLAvaan-method *(*INLAvaan-class*)*, 40
summary.bfit_indices *(*bfit_indices*)*, 17
summary.inlavaan_loo *(*loo*)*, 44
summary.inlavaan_waic *(*waic*)*, 66

timing, 63
timing(), *4*, *8*, *13*, *26*, *33*, *35*
timing,INLAvaan-method *(*timing*)*, 63

update, 64
update(), *37*
update,INLAvaan-method *(*update*)*, 64

vcov, 65
vcov(), *62*
vcov,INLAvaan-method *(*vcov*)*, 65

waic, 66
waic(), *4*, *8*, *9*, *13*, *28*, *35*, *39*, *43*, *46*, *47*