Package 'INLAvaan'

July 22, 2026

Type Package

Title Approximate Bayesian Latent Variable Analysis

Version 0.3.1

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 (>= 0.6-19), methods, parallel, stats, utils

Suggests blavaan, ggplot2, knitr, lme4, numDeriv, qrng, quarto, sn, testthat (>= 3.0.0), ucminf

VignetteBuilder quarto

Config/Needs/website rmarkdown

Config/roxygen2/markdown TRUE

Config/roxygen2/version 8.0.0

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 2026-07-21 22:30:09 UTC

Contents

acfa . . . 3
agrowth . . . 6
asem . . . 9
as_fun_string . . . 12
bfit_indices . . . 13
compare . . . 15
dbeta_box . . . 17
deviance . . . 18
diagnostics . . . 19
dsnorm . . . 21
fitmeasures . . . 22
fitted . . . 23
fit_skew_normal . . . 25
fit_skew_normal_samp . . . 26
get_inlavaan_internal . . . 27
inlavaan . . . 28
INLAvaan-class . . . 31
is_same_function . . . 33
logLik . . . 34
loo . . . 35
plot . . . 39
predict . . . 41
priors_for . . . 43
qsnorm_fast . . . 44
residuals . . . 45
sampling . . . 47
simulate . . . 49
standardisedsolution . . . 51
timing . . . 54
update . . . 55
vcov . . . 56
waic . . . 57

Index **60**

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,
  marginal_method = c("skewnorm", "asymgaus", "marggaus", "sampling"),
  marginal_correction = c("shortcut", "shortcut_fd", "hessian", "none"),
  nsamp = 1000,
  samp_copula = TRUE,
  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,
  ...
)
```

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().
test	Character indicating which post-estimation quantities to compute. Defaults to "standard": posterior fit indices (PPP and DIC), plus – for models supported by

the casewise machinery and fitted with a mean structure – the WAIC (reusing the fit's posterior draws, when `nsamp >= 100`) and a full leave-one-out cross-validation whenever its predicted serial cost is within a 10-second budget; both are stored with the fit (see `loo()` and `waic()`). "none" skips all of these. Include "loo" (e.g. `test = c("standard", "loo")`, or `test = "loo"` alone) to force the full LOO regardless of the budget.

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

`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. `"hessian"` computes the full `"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), with NORTA correlation adjustment. When `FALSE`, samples are drawn from the Gaussian (Laplace) approximation. Only re

`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.

`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 using parallel::mclapply() (fork-based; no parallelism on Windows).
...	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,
  marginal_method = c("skewnorm", "asymgaus", "marggaus", "sampling"),
  marginal_correction = c("shortcut", "shortcut_fd", "hessian", "none"),
  nsamp = 1000,
  samp_copula = TRUE,
  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,
  ...
)
```

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().

test — Character indicating which post-estimation quantities to compute. Defaults to "standard": posterior fit indices (PPP and DIC), plus – for models supported by the casewise machinery and fitted with a mean structure – the WAIC (reusing the fit's posterior draws, when nsamp >= 100) and a full leave-one-out cross-validation whenever its predicted serial cost is within a 10-second budget; both are stored with the fit (see loo() and waic()). "none" skips all of these. Include "loo" (e.g. test = c("standard", "loo"), or test = "loo" alone) to force the full LOO regardless of the budget.

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

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. "hessian" computes the full "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), with NORTA correlation adjustment. When FALSE, samples are drawn from the Gaussian (Laplace) approximation. Only re

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.

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 using parallel::mclapply() (fork-based; no parallelism on Windows).
...	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,
  marginal_method = c("skewnorm", "asymgaus", "marggaus", "sampling"),
  marginal_correction = c("shortcut", "shortcut_fd", "hessian", "none"),
  nsamp = 1000,
  samp_copula = TRUE,
  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,
  ...
)
```

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().
test	Character indicating which post-estimation quantities to compute. Defaults to "standard": posterior fit indices (PPP and DIC), plus – for models supported by the casewise machinery and fitted with a mean structure – the WAIC (reusing the fit's posterior draws, when nsamp >= 100) and a full leave-one-out cross-validation whenever its predicted serial cost is within a 10-second budget; both are stored with the fit (see loo() and waic()). "none" skips all of these. Include "loo" (e.g. test = c("standard", "loo"), or test = "loo" alone) to force the full LOO regardless of the budget.
vb_correction	Logical indicating whether to apply a variational Bayes correction for the posterior mean vector of estimates. Defaults to TRUE.
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. "hessian" computes the full "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), with NORTA correlation adjustment. When FALSE, samples are drawn from the Gaussian (Laplace) approximation. Only re

`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.

`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 using `parallel::mclapply()` (fork-based; no parallelism on Windows).

`...` 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	An optional INLAvaan object representing the baseline (null) model. Required for incremental fit indices (BCFI, BTLI, BNFI).

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).

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`	An INLAvaan (or `inlavaan_internal`) object used as the **baseline** (null) model. It is included in the comparison table and passed to fitMeasures() for incremental indices.
`y, ...`	One or more INLAvaan (or `inlavaan_internal`) objects to compare against the baseline.
`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

The first argument `x` serves as the **baseline** (null) model. All models (including the baseline) appear in the comparison table. The baseline is also passed to fitMeasures() when incremental fit indices (BCFI, BTLI, BNFI) are requested via `fit.measures`.

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 `test != "none"` was used during fitting).

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 second-order 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). 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 = "loo"` or `add_loo()`) are reused.

`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 `test != "none"`.
`...`	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.

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.

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

`kld_mean` Mean per-parameter KL divergence.

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

`nmad_mean` Mean NMAD across marginals.

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

`param` 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.

`nmad` Normalised max-absolute-deviation of the skew-normal fit (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 look off: the optimiser did not converge, `mode_shift_max` exceeds 0.1, any marginal has NMAD above 0.1, the VB correction shifted a posterior mean by more than 1 posterior SD, or the Hessian condition number exceeds 1e8. A healthy fit stays silent. 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()`) are included in `"all"` only when a LOO result is stored with the fit (`test = "loo"` 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()` directly) for repeated access. INLAvaan's stable spelling `fit.measures` is also accepted.
`baseline_model`	An optional INLAvaan object representing the baseline (null) model. Required for incremental fit indices (BCFI, BTLI, BNFI). Must have been fitted with `test != "none"`. INLAvaan's stable spelling `baseline.model` is also accepted.

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", "pD", "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, ...)

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

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.

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**).

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.

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.

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"`, `"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,
  marginal_method = c("skewnorm", "asymgaus", "marggaus", "sampling"),
  marginal_correction = c("shortcut", "shortcut_fd", "hessian", "none"),
  nsamp = 1000,
  samp_copula = TRUE,
  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,
  ...
)
```

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.
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().
test	Character indicating which post-estimation quantities to compute. Defaults to "standard": posterior fit indices (PPP and DIC), plus – for models supported by the casewise machinery and fitted with a mean structure – the WAIC (reusing the fit's posterior draws, when nsamp >= 100) and a full leave-one-out cross-validation whenever its predicted serial cost is within a 10-second budget; both are stored with the fit (see loo() and waic()). "none" skips all of these. Include "loo" (e.g. test = c("standard", "loo"), or test = "loo" alone) to force the full LOO regardless of the budget.
vb_correction	Logical indicating whether to apply a variational Bayes correction for the posterior mean vector of estimates. Defaults to TRUE.
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. "hessian" computes the full "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), with NORTA correlation adjustment. When FALSE, samples are drawn from the Gaussian (Laplace) approximation. Only re
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.
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 using parallel::mclapply() (fork-based; no parallelism on Windows).
...	Additional arguments to be passed to the lavaan model fitting function.

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 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 PPP).
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 test != "none".
...	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,
  Sigma = 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,
  Sigma = NULL,
  cores = NULL,
  verbose = FALSE,
  ...
)

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

Arguments

x	A fitted INLAvaan object (or its inlavaan_internal list).
...	Not used.
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, as recorded in the fit – for multigroup fits the full results are stacked by group, but a unit is always addressed by its case number); for LOCO, cluster positions.
second_order	Logical; compute the second-order correction (default TRUE). FALSE skips the Hessian stage entirely and reports first-order estimates.
theta, Sigma	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.

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^*, Σ), the log conditional predictive ordinate is approximated to first and second order by

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

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

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` $= -2\,$elpd. The effective number of parameters is $p_{\mathrm{loo}} = \sum_u (\mathrm{lpd}_u - \log \mathrm{CPO}_u)$, where lpd_u is the analogous Taylor approximation of the full-posterior pointwise log predictive density. Units whose second-order curvature matrix is not positive definite fall back to first order for that unit only (flagged in `per_unit$ok`).

The type is resolved automatically: per-cluster (`"loco"`) when the model was fitted with a `cluster` argument, per-subject (`"loso"`) otherwise. For a two-level model these are the two estimands of Merkle, Furr & Rabe-Hesketh (2019): the default per-cluster `"loco"` is the *marginal* predictive (leave-one-cluster-out – prediction for a *new* cluster), while `type = "loso"` forces the *conditional* predictive (leave-one-unit-out – prediction for a new observation within an *observed* cluster), where row i of cluster j contributes $\ell_i = \ell_j(\text{full}) - \ell_j(\text{minus row } i)$, the conditional density of the row given the rest of its cluster. The two answer different questions and are easily conflated, so the per-cluster marginal is the default and `type = "loso"` warns. It works with and without missing data, costs one cluster evaluation per row per Hessian direction, and is best subset with `units`.

Multigroup models. Groups are independent, so each unit is scored against its own group's implied moments; without a mean structure the exchangeability transformation applies per group, and cross-group equality constraints (`group.equal`) flow through the packed parameter space automatically. The per-unit results are stacked by group (a `group` column records the membership), and units are identified by *case number* – the row number of the analysed dataset – so a unit keeps its identity across fits that assign or order groups differently (e.g. a pooled fit versus a grouped fit of the same data, which `compare()` pairs unit by unit). This makes `compare(..., loo = TRUE)` the instrument of choice for the measurement-invariance ladder: configural, metric, and scalar fits are compared on a proper predictive scale with paired standard errors.

Supplying `theta` and/or `Sigma` scores the model at an *arbitrary* Gaussian posterior summary instead of the fit's own, without refitting. This is the building block for refit-free model exploration: for example, conditioning the encompassing model's summary on a parameter being zero (a rank-one update of `theta` and `Sigma`) and scoring the result gives the LOO of that submodel from a single fit. INLAvaan provides only this evaluation API; search strategies are left to the user. A conditioned `Sigma` may be singular; the computation automatically restricts to the non-degenerate block, which is exact.

Parallelism is strictly opt-in: the default `cores = NULL` runs serially, and `cores > 1` parallelises the Hessian stage via forking (not available on Windows).

Calling `loo()` never modifies the fitted object. Under the default `test = "standard"`, `inlavaan()` already computes and stores the full LOO at fit time whenever the model is supported, has a mean structure, and the predicted serial cost is within a 10-second budget (measured by timing one score evaluation); `test = "loo"` forces the computation regardless of the budget, and `fit <- add_loo(fit)` stores it post hoc. A stored result is returned directly by `loo(fit)` when called with default arguments, and is reused by `fitmeasures()` and `compare()` without recomputation.

Exogenous covariates. The flavour of the score follows the fitted likelihood. Under `fixed.x = FALSE` the covariates are modelled jointly and each unit is scored by the joint predictive density of its outcomes *and* covariates (`flavour = "joint"`). Under `fixed.x = TRUE` (the lavaan default) the fitted likelihood is the conditional one, and each unit is scored by the predictive density of its outcomes *given* its covariates (`flavour = "conditional"`); since the conditional likelihood is exactly invariant to the frozen covariate moments, this involves no additional approximation. The two flavours estimate different quantities whose scales differ by the covariate predictive density, so a joint and a conditional elpd must never be compared (`compare()` refuses mixed-flavour comparisons). Conditional scores of models conditioning on *different* covariate sets are comparable provided the outcome variables match – the natural setting for covariate selection. Both flavours support any covariate placement: single-level covariates, and cluster-level (between) and/or within-level covariates in two-level models.

Missing data. Fits estimated by full-information maximum likelihood (`missing = "ml"`) are scored on the *observed-data* predictive: each unit contributes the density of the entries it actually has, with its full row (single-level) or whole cluster (two-level) removed from the conditioning set. For single-level fits the casewise kernels operate on each unit's observed subset, grouping rows by missing pattern, so a unit with fewer observed entries contributes a smaller log-likelihood term *and* a smaller score and thus self-weights in the elpd. Two-level fits are scored per cluster (`"loco"`): each cluster contributes its observed-data marginal likelihood, evaluated by lavaan's raw-data cluster kernels (no per-cluster sufficient statistics are needed, since leave-one-cluster-out deletes the whole cluster). All carry the same missing-at-random assumption as the FIML fit itself. Because the score is the observed-entry predictive, a `compare()` of two missing-data fits is meaningful only when they share the same observed entries (the same data *and* the same holes). The two-level conditional predictive (`type = "loso"`) is available under missing data too, on the same kernels.

Supported models: continuous-indicator models fitted with the `ML` estimator (including FIML, `missing = "ml"`, single- and two-level), single-group or multigroup (multigroup two-level models are not supported yet). If the `loo` package is attached it masks this generic, but `loo(fit)` continues to dispatch correctly because the method is registered by generic name.

Value

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

`per_unit` Data frame of pointwise results: `unit` (case number for LOSO, cluster position for LOCO), `group` (multigroup fits only), `nobs` (1 for LOSO, the cluster size for LOCO), `l_star` (unit log-likelihood at the summary), `score_norm`, `lpd_1`/`lpd_2` (pointwise log predictive density), `log_cpo_1`/`log_cpo_2` (pointwise LOO contributions), `det_term`, and `ok` (second-order success flag).

`estimates` Matrix with rows `elpd_loo`, `p_loo`, `looic` and columns `Estimate`, `SE` (headline second-order values).

`elpd_1`, `elpd_2`, `se_1`, `se_2`, `p_loo_1`, `p_loo_2` First- and second-order aggregates.

`type`, `flavour`, `n_units`, `n_groups`, `n_ok`, `second_order`, `theta_overridden` Metadata; `flavour` records whether units were scored jointly with their covariates (`"joint"`) or conditionally on them (`"conditional"`, for `fixed.x` fits).

`add_loo()` returns a copy of `object` with the LOO result 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 result always matches `loo(fit)`.

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$theta_star
Sigma <- int$Sigma_theta
p <- which(names(coef(fit)) == "visual~~speed")
theta_c <- theta - Sigma[, p] * (theta[p] / Sigma[p, p])
Sigma_c <- Sigma - tcrossprod(Sigma[, p]) / Sigma[p, p]
loo(fit, theta = theta_c, Sigma = Sigma_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, or "all" (default) to plot all free 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 $\eta \vert y, \theta$. `"yhat"`, `"ov"` Predicted means for observed variables $E(y \vert \eta, \theta) = \nu + \Lambda\eta$; no residual noise. `"ypred"`, `"ydist"` Predicted observed values including residual noise $y = \nu + \Lambda\eta + \varepsilon, \varepsilon \sim N(0, \Theta)$. `"ymis"`, `"ovmis"` Imputed values for missing observations, drawn from the conditional distribution $y_{mis} \vert y_{obs}, \theta$.
`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 multilevel models, specifies whether level 1 or level 2 latent variables are desired (default `1L`).
`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
- `theta` = `"gamma(1,.5)[sd]"`: Residual precisions
- `psi` = `"gamma(1,.5)[sd]"`: Latent variable precisions
- `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, ...)

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

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.

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).

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).

"observed" Observed variables generated from the full model. Returns an nsamp by nobs_vars matrix.

"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.

"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.

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. 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.

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	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().

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"`, `"optim"`, `"vb"`, `"loglik"`, `"marginals"`, `"norta"`, `"sampling"`, `"covariances"`, `"definedpars"`, `"deltapars"`, `"test"`, `"total"`.

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 (matching lavaan output). `"theta"` returns the Laplace approximation covariance in the internal parameterisation.
`...`	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 from unit log-likelihoods evaluated over posterior draws. 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,
  nsamp = NULL,
  cores = NULL,
  verbose = FALSE,
  ...
)

## S3 method for class 'inlavaan_internal'
```

```
waic(
  x,
  type = c("auto", "loso", "loco"),
  units = NULL,
  nsamp = NULL,
  cores = NULL,
  verbose = FALSE,
  ...
)
```

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.
nsamp	Number of posterior draws. Defaults to the nsamp used when fitting the model.
cores	Number of cores for evaluating draws. The default NULL runs serially; parallelism must be requested explicitly.
verbose	Logical; print progress (default FALSE).

Details

For each posterior draw θ_s (drawn with the same copula sampler used for the fit's posterior summaries) and unit u, the log-likelihood $\log p(y_u \mid \theta_s)$ is evaluated; then $\mathrm{lpd}_u = \log \frac{1}{S} \sum_s p(y_u \mid \theta_s)$, $p_{\mathrm{waic},u} = \mathrm{var}_s \log p(y_u \mid \theta_s)$, and $\mathrm{elpd}_{\mathrm{waic}} = \sum_u (\mathrm{lpd}_u - p_{\mathrm{waic},u})$ with $\mathrm{WAIC} = -2\,\mathrm{elpd}_{\mathrm{waic}}$. Unlike loo(), this is a sampling-based estimate: results vary with the random draws, and units with $p_{\mathrm{waic},u} > 0.4$ trigger a reliability warning (also annotated when printing). 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()).

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 (each row contributes the conditional density of its observed entries given the rest of its 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 compute the same estimand by sampling and by Taylor expansion.

Under the default test = "standard", inlavaan() computes the WAIC at fit time by reusing the posterior draws the fit already produced (when the model is supported and nsamp >= 100), and stores it 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. 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`, `n_units`, `n_groups`, and `nsamp`.

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(), *5*, *8*, *12*, *29*, *31*, *32*, *44*, *55*
add_loo *(*loo*)*, 35
add_loo(), *16*, *22*
agrowth, 6
agrowth(), *5*, *8*, *12*, *29*, *31*, *32*, *44*, *55*
AIC, *34*
AIC,INLAvaan-method *(*logLik*)*, 34
anova *(*compare*)*, 15
anova(), *34*
anova,INLAvaan-method *(*compare*)*, 15
as_fun_string, 12
asem, 9
asem(), *5*, *8*, *11*, *12*, *29*, *31*, *32*, *44*, *55*

bfit_indices, 13
bfit_indices(), *16*, *23*, *48*
BIC, *34*
BIC,INLAvaan-method *(*logLik*)*, 34
blavaan::blavFitIndices(), *13*, *14*
blavaan::dpriors(), *43*

coef *(*INLAvaan-class*)*, 31
coef(), *24*, *53*, *57*
coef,INLAvaan-method *(*INLAvaan-class*)*, 31
compare, 15
compare(), *14*, *18*, *19*, *23*, *34*, *35*, *37–39*, *56*
compare,INLAvaan-method *(*compare*)*, 15

dbeta_box, 17
deviance, 18
deviance(), *35*
diagnostics, 19
diagnostics(), *23*, *27*, *40*, *54*
diagnostics,INLAvaan-method *(*diagnostics*)*, 19
dsnorm, 21

fit_skew_normal, 25
fit_skew_normal_samp, 26
fitmeasures, 22
fitMeasures(), *15*, *16*, *24*, *46*
fitmeasures(), *14*, *16*, *21*, *37*, *39*, *58*, *59*
fitMeasures,INLAvaan-method *(*fitmeasures*)*, 22
fitmeasures,INLAvaan-method *(*fitmeasures*)*, 22
fitted, 23
fitted(), *46*
fitted,INLAvaan-method *(*fitted*)*, 23
fitted.values,INLAvaan-method *(*fitted*)*, 23

get_inlavaan_internal, 27

INLAvaan, *13*, *15*, *18*, *19*, *22*, *24*, *27*, *34–36*, *40*, *41*, *46*, *47*, *49*, *52*, *54–58*
inlavaan, 28
inlavaan(), *5*, *8*, *11*, *20*, *22*, *32*, *37*, *39*, *44*, *55*, *56*, *58*
INLAvaan-class, 31
is_same_function, 33

lavaan, *5*, *8*, *11*, *12*, *30–32*
lavaan::fitMeasures(), *14*
lavaan::lavOptions(), *5*, *12*
logLik, 34
logLik(), *19*
logLik,INLAvaan-method *(*logLik*)*, 34
loo, 35
loo(), *4*, *7*, *10*, *16*, *22*, *29*, *34*, *35*, *57–59*

model.syntax, *3*, *6*, *10*, *29*

nobs *(*INLAvaan-class*)*, 31
nobs,INLAvaan-method *(*INLAvaan-class*)*, 31

parallel::mclapply(), *5*, *8*, *11*, *30*
plot, 39

plot(), *21*
plot,INLAvaan,ANY-method *(*plot*)*, 39
predict, 41
predict(), *24*, *46*, *48*
predict,INLAvaan-method *(*predict*)*, 41
print.bfit_indices *(*bfit_indices*)*, 13
priors_for, 43
priors_for(), *3*, *7*, *10*, *29*

qsnorm_fast, 44

resid,INLAvaan-method *(*residuals*)*, 45
residuals, 45
residuals,INLAvaan-method *(*residuals*)*, 45

sampling, 47
sampling(), *42*, *50*
sampling,INLAvaan-method *(*sampling*)*, 47
set.seed(), *49*
show *(*INLAvaan-class*)*, 31
show,INLAvaan-method *(*INLAvaan-class*)*, 31
simulate, 49
simulate(), *42*, *48*
simulate,INLAvaan-method *(*simulate*)*, 49
standardisedSolution *(*standardisedsolution*)*, 51
standardisedsolution, 51
standardisedsolution(), *57*
standardizedSolution *(*standardisedsolution*)*, 51
standardizedsolution *(*standardisedsolution*)*, 51
stats::AIC(), *34*
stats::update(), *55*
summary *(*INLAvaan-class*)*, 31
summary(), *14*, *40*, *42*, *53*, *54*, *57*
summary,INLAvaan-method *(*INLAvaan-class*)*, 31
summary.bfit_indices *(*bfit_indices*)*, 13

timing, 54
timing(), *21*, *27*
timing,INLAvaan-method *(*timing*)*, 54

update, 55
update(), *30*
update,INLAvaan-method *(*update*)*, 55

vcov, 56
vcov(), *53*
vcov,INLAvaan-method *(*vcov*)*, 56

waic, 57
waic(), *4*, *7*, *10*, *29*, *34*, *35*