### Package ‘bridgesampling’

February 26, 2020

<table>
<thead>
<tr>
<th>Type</th>
<th>Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Bridge Sampling for Marginal Likelihoods and Bayes Factors</td>
</tr>
<tr>
<td>Version</td>
<td>1.0-0</td>
</tr>
<tr>
<td>Depends</td>
<td>R (&gt;= 3.0.0)</td>
</tr>
<tr>
<td>Imports</td>
<td>mvtnorm, Matrix, Brobdingnag, stringr, coda, parallel, scales, utils, methods</td>
</tr>
<tr>
<td>Suggests</td>
<td>testthat, Rcpp, RcppEigen, R2jags, rjags, runjags, knitr, rmarkdown, R.rsp, BayesFactor, rstan, rstanarm, nimble, MCMCpack</td>
</tr>
<tr>
<td>License</td>
<td>GPL (&gt;= 2)</td>
</tr>
<tr>
<td>LazyData</td>
<td>true</td>
</tr>
<tr>
<td>RoxygenNote</td>
<td>7.0.2</td>
</tr>
<tr>
<td>VignetteBuilder</td>
<td>knitr, R.rsp</td>
</tr>
<tr>
<td>URL</td>
<td><a href="https://github.com/quentingronau/bridgesampling">https://github.com/quentingronau/bridgesampling</a></td>
</tr>
<tr>
<td>NeedsCompilation</td>
<td>no</td>
</tr>
<tr>
<td>Author</td>
<td>Quentin F. Gronau [aut, cre] (<a href="https://orcid.org/0000-0001-5510-6943">https://orcid.org/0000-0001-5510-6943</a>), Henrik Singmann [aut] (<a href="https://orcid.org/0000-0002-4842-3657">https://orcid.org/0000-0002-4842-3657</a>), Jonathan J. Forster [ctb], Eric-Jan Wagenmakers [ths], The JASP Team [ctb], Jiqiang Guo [ctb], Jonah Gabry [ctb], Ben Goodrich [ctb], Kees Mulder [ctb], Perry de Valpine [ctb]</td>
</tr>
<tr>
<td>Maintainer</td>
<td>Quentin F. Gronau <a href="mailto:Quentin.F.Gronau@gmail.com">Quentin.F.Gronau@gmail.com</a></td>
</tr>
</tbody>
</table>
bayes_factor() is simply an (S3 generic) alias for bf().

## Default S3 method:
bayes_factor(x1, x2, log = FALSE, ...)

## S3 method for class 'bridge'
bf(x1, x2, log = FALSE, ...)

## S3 method for class 'bridge_list'
bf(x1, x2, log = FALSE, ...)

## Default S3 method:
bf(x1, x2, log = FALSE, ...)

Arguments

x1 Object of class "bridge" or "bridge_list" as returned from bridge_sampler. Additionally, the default method assumes that x1 is a single numeric log marginal likelihood (e.g., from logml) and will throw an error otherwise.
### bf

x2 Object of class "bridge" or "bridge_list" as returned from `bridge_sampler`. Additionally, the default method assumes that x2 is a single numeric log marginal likelihood (e.g., from `logml`) and will throw an error otherwise.

log Boolean. If TRUE, the function returns the log of the Bayes factor. Default is FALSE.

... currently not used here, but can be used by other methods.

### Details

Computes the Bayes factor (Kass & Raftery, 1995) in favor of the model associated with x1 over the model associated with x2.

### Value

For the default method returns a list of class "bf_default" with components:

- `bf`: (scalar) value of the Bayes factor in favor of the model associated with x1 over the model associated with x2.
- `log`: Boolean which indicates whether `bf` corresponds to the log Bayes factor.

For the method for "bridge" objects returns a list of class "bf_bridge" with components:

- `bf`: (scalar) value of the Bayes factor in favor of the model associated with x1 over the model associated with x2.
- `log`: Boolean which indicates whether `bf` corresponds to the log Bayes factor.

For the method for "bridge_list" objects returns a list of class "bf_bridge_list" with components:

- `bf`: a numeric vector consisting of Bayes factors where each element gives the Bayes factor for one set of logmls in favor of the model associated with x1 over the model associated with x2. The length of this vector is given by the "bridge_list" element with the most repetitions. Elements with fewer repetitions will be recycled (with warning).
- `bf_median_based`: (scalar) value of the Bayes factor in favor of the model associated with x1 over the model associated with x2 that is based on the median values of the logml estimates.
- `log`: Boolean which indicates whether `bf` corresponds to the log Bayes factor.

### Note

For examples, see `bridge_sampler` and the accompanying vignettes:

```r
vignette("bridgesampling_example_jags")
vignette("bridgesampling_example_stan")
```

### Author(s)

Quentin F. Gronau

### References

Methods for bridge and bridge_list objects

Description

Methods defined for objects returned from the generic `bridge_sampler` function.

Usage

```r
## S3 method for class 'bridge'
summary(object, na.rm = TRUE, ...)

## S3 method for class 'bridge_list'
summary(object, na.rm = TRUE, ...)

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

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

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

## S3 method for class 'bridge_list'
print(x, na.rm = TRUE, ...)
```

Arguments

- `object, x` object of class bridge or bridge_list as returned from `bridge_sampler`.
- `na.rm` logical. Should NA estimates in bridge_list objects be removed? Passed to `error_measures`.
- `...` further arguments, currently ignored.

Value

The summary methods return a data.frame which contains the log marginal likelihood plus the result returned from invoking `error_measures`.

The print methods simply print and return nothing.
Description

Computes log marginal likelihood via bridge sampling.

Usage

bridge_sampler(samples, ...)

## S3 method for class 'stanfit'
bridge_sampler(samples = NULL,
               stanfit_model = samples, repetitions = 1, method = "normal",
               cores = 1, use_neff = TRUE, maxiter = 1000, silent = FALSE,
               verbose = FALSE, ...)

## S3 method for class 'mcmc.list'
bridge_sampler(samples = NULL,
               log_posterior = NULL, ..., data = NULL, lb = NULL, ub = NULL,
               repetitions = 1, param_types = rep("real", ncol(samples[[1]])),
               method = "normal", cores = 1, use_neff = TRUE, packages = NULL,
               varlist = NULL, envir = .GlobalEnv, rcppFile = NULL,
               maxiter = 1000, silent = FALSE, verbose = FALSE)

## S3 method for class 'mcmc'
bridge_sampler(samples = NULL, log_posterior = NULL,
               ..., data = NULL, lb = NULL, ub = NULL, repetitions = 1,
               method = "normal", cores = 1, use_neff = TRUE, packages = NULL,
               varlist = NULL, envir = .GlobalEnv, rcppFile = NULL,
               maxiter = 1000, param_types = rep("real", ncol(samples)),
               silent = FALSE, verbose = FALSE)

## S3 method for class 'matrix'
bridge_sampler(samples = NULL, log_posterior = NULL,
               ..., data = NULL, lb = NULL, ub = NULL, repetitions = 1,
               method = "normal", cores = 1, use_neff = TRUE, packages = NULL,
               varlist = NULL, envir = .GlobalEnv, rcppFile = NULL,
               maxiter = 1000, param_types = rep("real", ncol(samples)),
               silent = FALSE, verbose = FALSE)

## S3 method for class 'stanreg'
bridge_sampler(samples, repetitions = 1,
               method = "normal", cores = 1, use_neff = TRUE, maxiter = 1000,
               silent = FALSE, verbose = FALSE, ...)

## S3 method for class 'rjags'

bridge_sampler(samples, repetitions = 1, method = "normal", cores = 1, use_neff = TRUE, maxiter = 1000, silent = FALSE, verbose = FALSE, ...)

## S3 method for class 'data.frame'
bridge_sampler(samples, repetitions = 1, method = "normal", cores = 1, use_neff = TRUE, maxiter = 1000, silent = FALSE, verbose = FALSE, ...)
bridge_sampler(samples = NULL, log_posterior = NULL,
..., data = NULL, lb = NULL, ub = NULL, repetitions = 1,
method = "normal", cores = 1, use_neff = TRUE, packages = NULL,
varlist = NULL, envir = .GlobalEnv, rcppFile = NULL,
maxiter = 1000, silent = FALSE, verbose = FALSE)

## S3 method for class 'runjags'
bridge_sampler(samples = NULL, log_posterior = NULL,
..., data = NULL, lb = NULL, ub = NULL, repetitions = 1,
method = "normal", cores = 1, use_neff = TRUE, packages = NULL,
varlist = NULL, envir = .GlobalEnv, rcppFile = NULL,
maxiter = 1000, silent = FALSE, verbose = FALSE)

## S3 method for class 'MCMC_refClass'
bridge_sampler(samples, repetitions = 1,
method = "normal", cores = 1, use_neff = TRUE, maxiter = 1000,
silent = FALSE, verbose = FALSE, ...)

**Arguments**

- **samples**
  - an mcmc.list object, a fitted stanfit object, a stanreg object, an rjags object, a runjags object, or a matrix with posterior samples (colnames need to correspond to parameter names in lb and ub) with posterior samples.
  - ... additional arguments passed to log_posterior. Ignored for the stanfit and stanreg methods.
  - **stanfit_model** for the stanfit method, an additional object of class "stanfit" with the same model as samples, which will be used for evaluating the log_posterior (i.e., it does not need to contain any samples). The default is to use samples. In case samples was compiled in a different R session or on another computer with a different OS or setup, the samples model usually cannot be used for evaluation. In this case, one can compile the model on the current computer with iter = 0 and pass it here (this usually needs to be done before samples is loaded).

- **repetitions** number of repetitions.

- **method** either "normal" or "warp3".

- **cores** number of cores used for evaluating log_posterior. On unix-like systems (where .Platform$OS.type == "unix" evaluates to TRUE; e.g., Linux and Mac OS) forking via mclapply is used. Hence elements needed for evaluation should be in the .GlobalEnv. For other systems (e.g., Windows) makeCluster is used and further arguments specified below will be used.

- **use_neff** Boolean which determines whether the effective sample size is used in the optimal bridge function. Default is TRUE. If FALSE, the number of samples is used instead. If samples is a matrix, it is assumed that the matrix contains the samples of one chain in order. If samples come from more than one chain, we recommend to use an mcmc.list object for optimal performance.

- **maxiter** maximum number of iterations for the iterative updating scheme. Default is 1,000 to avoid infinite loops.
silent  Boolean which determines whether to print the number of iterations of the updating scheme to the console. Default is FALSE.

verbose Boolean. Should internal debug information be printed to console? Default is FALSE.

log_posterior function or name of function that takes a parameter vector and the data as input and returns the log of the unnormalized posterior density (i.e., a scalar value). If the function name is passed, the function should exist in the .GlobalEnv. For special behavior if cores > 1 see Details.

data  data object which is used in log_posterior.

lb  named vector with lower bounds for parameters.

ub  named vector with upper bounds for parameters.

param_types  character vector of length ncol(samples) with "real", "simplex" or "circular". For all regular bounded or unbounded continuous parameters, this should just be "real". However, if there are parameters which lie on a simplex or on the circle, this should be noted here. Simplex parameters are parameters which are bounded below by zero and collectively sum to one, such as weights in a mixture model. For these, the stick-breaking transformation is performed as described in the Stan reference manual. The circular variables are given a numerical representation to which the normal distribution is most likely a good fit. Only possible to use with bridge_sampler.matrix.

packages  character vector with names of packages needed for evaluating log_posterior in parallel (only relevant if cores > 1 and .Platform$OS.type != "unix").

varlist  character vector with names of variables needed for evaluating log_posterior (only needed if cores > 1 and .Platform$OS.type != "unix" as these objects will be exported to the nodes). These objects need to exist in envir.

evir  specifies the environment for varlist (only needed if cores > 1 and .Platform$OS.type != "unix" as these objects will be exported to the nodes). Default is .GlobalEnv.

cppFile  in case cores > 1 and log_posterior is an Rcpp function, cppFile specifies the path to the cpp file (will be compiled on all cores).

Details

Bridge sampling is implemented as described in Meng and Wong (1996, see equation 4.1) using the "optimal" bridge function. When method = "normal", the proposal distribution is a multivariate normal distribution with mean vector equal to the sample mean vector of samples and covariance matrix equal to the sample covariance matrix of samples. For a recent tutorial on bridge sampling, see Gronau et al. (in press).

When method = "warp3", the proposal distribution is a standard multivariate normal distribution and the posterior distribution is "warped" (Meng & Schilling, 2002) so that it has the same mean vector, covariance matrix, and skew as the samples. method = "warp3" takes approximately twice as long as method = "normal".

Note that for the matrix method, the lower and upper bound of a parameter cannot be a function of the bounds of another parameter. Furthermore, constraints that depend on multiple parameters of the model are not supported. This usually excludes, for example, parameters that constitute a covariance matrix or sets of parameters that need to sum to one.
However, if the retransformations are part of the model itself and the log_posterior accepts parameters on the real line and performs the appropriate Jacobian adjustments, such as done for stanfit and stanreg objects, such constraints are obviously possible (i.e., we currently do not know of any parameter supported within Stan that does not work with the current implementation through a stanfit object).

**Parallel Computation:** On unix-like systems forking is used via `mclapply`. Hence elements needed for evaluation of log_posterior should be in the `.GlobalEnv`.

On other OSes (e.g., Windows), things can get more complicated. For normal parallel computation, the log_posterior function can be passed as both function and function name. If the latter, it needs to exist in the environment specified in the `envir` argument. For parallel computation when using an Rcpp function, log_posterior can only be passed as the function name (i.e., character). This function needs to result from calling `sourceCpp` on the file specified in `rcppFile`.

Due to the way rstan currently works, parallel computations with stanfit and stanreg objects only work with forking (i.e., NOT on Windows).

**Value**

if `repetitions = 1`, returns a list of class "bridge" with components:

- logml: estimate of log marginal likelihood.
- niter: number of iterations of the iterative updating scheme.
- method: bridge sampling method that was used to obtain the estimate.
- q11: log posterior evaluations for posterior samples.
- q12: log proposal evaluations for posterior samples.
- q21: log posterior evaluations for samples from proposal.
- q22: log proposal evaluations for samples from proposal.

if `repetitions > 1`, returns a list of class "bridge_list" with components:

- logml: numeric vector with estimates of log marginal likelihood.
- niter: numeric vector with number of iterations of the iterative updating scheme for each repetition.
- method: bridge sampling method that was used to obtain the estimates.
- repetitions: number of repetitions.

**Warning**

Note that the results depend strongly on the parameter priors. Therefore, it is strongly advised to think carefully about the priors before calculating marginal likelihoods. For example, the prior choices implemented in rstanarm or brms might not be optimal from a testing point of view. We recommend to use priors that have been chosen from a testing and not a purely estimation perspective.

Also note that for testing, the number of posterior samples usually needs to be substantially larger than for estimation.
Note

To be able to use a stanreg object for samples, the user crucially needs to have specified the diagnostic_file when fitting the model in rstanarm.

Author(s)

Quentin F. Gronau and Henrik Singmann. Parallel computing (i.e., cores > 1) and the stanfit method use code from rstan by Jiaqing Guo, Jonah Gabry, and Ben Goodrich. Ben Goodrich added the stanreg method. Kees Mulder added methods for simplex and circular variables.

References


See Also

`bf` allows the user to calculate Bayes factors and `post_prob` allows the user to calculate posterior model probabilities from bridge sampling estimates. `bridge-methods` lists some additional methods that automatically invoke the `error_measures` function.

Examples

```r
## Example 1: Estimating the Normalizing Constant of a Two-Dimensional Standard Normal Distribution

library(bridgesampling)
library(mvtnorm)
samples <- rmvnorm(1e4, mean = rep(0, 2), sigma = diag(2))
```
colnames(samples) <- c("x1", "x2")
log_density <- function(samples.row, data) {
  -.5* t(samples.row) %*% samples.row
}

lb <- rep(-Inf, 2)
ub <- rep(Inf, 2)
names(lb) <- names(ub) <- colnames(samples)
bridge_result <- bridge_sampler(samples = samples, log_posterior = log_density,
data = NULL, lb = lb, ub = ub, silent = TRUE)

# compare to analytical value
analytical <- log(2*pi)
print(cbind(bridge_result$logml, analytical))

## Not run:
## ------------------------------------------------------------------------
## Example 2: Hierarchical Normal Model
## ------------------------------------------------------------------------

# for a full description of the example, see
vignette("bridgesampling_example_jags")

library(R2jags)

### generate data ###
set.seed(12345)

mu <- 0
tau2 <- 0.5
sigma2 <- 1

n <- 20
theta <- rnorm(n, mu, sqrt(tau2))
y <- rnorm(n, theta, sqrt(sigma2))

### set prior parameters
alpha <- 1
beta <- 1
mu0 <- 0
tau0 <- 1

### functions to get posterior samples ###

### H0: mu = 0

getsamplesModelH0 <- function(data, niter = 52000, nburnin = 2000, nchains = 3) {
  model <- "
  model {

for (i in 1:n) {
    theta[i] ~ dnorm(0, invTau2)
    y[i] ~ dnorm(theta[i], 1/sigma2)
}
invTau2 ~ dgamma(alpha, beta)
tau2 <- 1/invTau2
"

s <- jags(data, parameters.to.save = c("theta", "invTau2"),
           model.file = textConnection(model),
           n.chains = nchains, n.iter = niter,
           n.burnin = nburnin, n.thin = 1)

return(s)
}

### H1: mu != 0

getSamplesModelH1 <- function(data, niter = 52000, nburnin = 2000,
                               nchains = 3) {

    model <- "
    model {
        for (i in 1:n) {
            theta[i] ~ dnorm(mu, invTau2)
            y[i] ~ dnorm(theta[i], 1/sigma2)
        }
        mu ~ dnorm(mu0, 1/tau20)
        invTau2 ~ dgamma(alpha, beta)
        tau2 <- 1/invTau2
    }"

    s <- jags(data, parameters.to.save = c("theta", "mu", "invTau2"),
              model.file = textConnection(model),
              n.chains = nchains, n.iter = niter,
              n.burnin = nburnin, n.thin = 1)

    return(s)
}

### get posterior samples ###

# create data lists for Jags

data_H0 <- list(y = y, n = length(y), alpha = alpha, beta = beta, sigma2 = sigma2)
data_H1 <- list(y = y, n = length(y), mu0 = mu0, tau20 = tau20, alpha = alpha,
                beta = beta, sigma2 = sigma2)

# fit models

samples_H0 <- getSamplesModelH0(data_H0)
samples_H1 <- getSamplesModelH1(data_H1)
## functions for evaluating the unnormalized posteriors on log scale ##

```r
log_posterior_H0 <- function(samples.row, data) {

  mu <- 0
  invTau2 <- samples.row["invTau2"]
  theta <- samples.row[paste0("theta[", seq_along(data$y), "]")]

  sum(dnorm(data$y, theta, data$sigma2, log = TRUE)) +
  sum(dnorm(theta, mu, 1/sqrt(invTau2), log = TRUE)) +
  dgamma(invTau2, data$alpha, data$beta, log = TRUE)
}

log_posterior_H1 <- function(samples.row, data) {

  mu <- samples.row["mu"]
  invTau2 <- samples.row["invTau2"]
  theta <- samples.row[paste0("theta[", seq_along(data$y), "]")]

  sum(dnorm(data$y, theta, data$sigma2, log = TRUE)) +
  sum(dnorm(theta, mu, 1/sqrt(invTau2), log = TRUE)) +
  dnorm(mu, data$mu0, sqrt(data$tau20), log = TRUE) +
  dgamma(invTau2, data$alpha, data$beta, log = TRUE)
}
```

# specify parameter bounds H0
```r
cn <- colnames(samples_H0$BUGSoutput$sims.matrix)
cn <- cn[cn != "deviance"]
lb_H0 <- rep(-Inf, length(cn))
ub_H0 <- rep(Inf, length(cn))
names(lb_H0) <- names(ub_H0) <- cn
lb_H0["invTau2"] <- 0
```

# specify parameter bounds H1
```r
cn <- colnames(samples_H1$BUGSoutput$sims.matrix)
cn <- cn[cn != "deviance"]
lb_H1 <- rep(-Inf, length(cn))
ub_H1 <- rep(Inf, length(cn))
names(lb_H1) <- names(ub_H1) <- cn
lb_H1["invTau2"] <- 0
```

# compute log marginal likelihood via bridge sampling for H0
```r
H0.bridge <- bridge_sampler(samples = samples_H0, data = data_H0,
  log_posterior = log_posterior_H0, lb = lb_H0,
  ub = ub_H0, silent = TRUE)
print(H0.bridge)
```

# compute log marginal likelihood via bridge sampling for H1
```r
H1.bridge <- bridge_sampler(samples = samples_H1, data = data_H1,
  log_posterior = log_posterior_H1, lb = lb_H1,
  ub = ub_H1, silent = TRUE)
```
# compute percentage error
print(error_measures(H0.bridge)$percentage)
print(error_measures(H1.bridge)$percentage)

# compute Bayes factor
BF01 <- bf(H0.bridge, H1.bridge)
print(BF01)

# compute posterior model probabilities (assuming equal prior model probabilities)
post1 <- post_prob(H0.bridge, H1.bridge)
print(post1)

# compute posterior model probabilities (using user-specified prior model probabilities)
post2 <- post_prob(H0.bridge, H1.bridge, prior_prob = c(.6, .4))
print(post2)

## End(Not run)

## Not run:

## Example 3: rstanarm

library(rstanarm)
# N.B.: remember to specify the diagnostic_file

fit_1 <- stan_glm(mpg ~ wt + qsec + am, data = mtcars,
                   chains = 2, cores = 2, iter = 5000,
                   diagnostic_file = file.path(tempdir(), "df.csv"))
bridge_1 <- bridge_sampler(fit_1)
fit_2 <- update(fit_1, formula = . ~ . + cyl)
bridge_2 <- bridge_sampler(fit_2, method = "warp3")
bf(bridge_1, bridge_2)

## End(Not run)

---

## error_measures

### Error Measures for Estimated Marginal Likelihood

#### Description

Computes error measures for estimated marginal likelihood.
error_measures

Usage

```r
error_measures(bridge_object, ...)
## S3 method for class 'bridge'
error_measures(bridge_object, ...)
## S3 method for class 'bridge_list'
error_measures(bridge_object, na.rm = TRUE, ...)
```

Arguments

- `bridge_object` an object of class "bridge" or "bridge_list" as returned from `bridge_sampler`.
- `...` additional arguments (currently ignored).
- `na.rm` a logical indicating whether missing values in logml estimates should be removed. Ignored for the bridge method.

Details

Computes error measures for marginal likelihood bridge sampling estimates. The approximate errors for a `bridge_object` of class "bridge" that has been obtained with `method = "normal"` and `repetitions = 1` are based on Fruehwirth-Schnatter (2004). Not applicable in case the object of class "bridge" has been obtained with `method = "warp3"` and `repetitions = 1`. To assess the uncertainty of the estimate in this case, it is recommended to run the "warp3" procedure multiple times.

Value

If `bridge_object` is of class "bridge" and has been obtained with `method = "normal"` and `repetitions = 1`, returns a list with components:

- `re2`: approximate relative mean-squared error for marginal likelihood estimate.
- `cv`: approximate coefficient of variation for marginal likelihood estimate (assumes that bridge estimate is unbiased).
- `percentage`: approximate percentage error of marginal likelihood estimate.

If `bridge_object` is of class "bridge_list", returns a list with components:

- `min`: minimum of the log marginal likelihood estimates.
- `max`: maximum of the log marginal likelihood estimates.
- `IQR`: interquartile range of the log marginal likelihood estimates.

Note

For examples, see `bridge_sampler` and the accompanying vignettes:

```r
deprecated_vignette("bridgesampling_example_jags")
deprecated_vignette("bridgesampling_example_stan")
```
**ier**

**Author(s)**

Quentin F. Gronau

**References**


**See Also**

The summary methods for `bridge` and `bridge_list` objects automatically invoke this function, see `bridge-methods`.

---

<table>
<thead>
<tr>
<th>ier</th>
<th><em>Standardized International Exchange Rate Changes from 1975 to 1986</em></th>
</tr>
</thead>
</table>

**Description**

This data set contains the changes in monthly international exchange rates for pounds sterling from January 1975 to December 1986 obtained from West and Harrison (1997, pp. 612-615). Currencies tracked are US Dollar (column `us_dollar`), Canadian Dollar (column `canadian_dollar`), Japanese Yen (column `yen`), French Franc (column `franc`), Italian Lira (column `lira`), and the (West) German Mark (column `mark`). Each series has been standardized with respect to its sample mean and standard deviation.

**Usage**

`ier`

**Format**

A matrix with 143 rows and 6 columns.

**Source**


Examples

```r
## Not run:

# BAYESIAN FACTOR ANALYSIS (AS PROPOSED BY LOPES & WEST, 2004)
library(bridgesampling)
library(rstan)
cores <- 4
options(mc.cores = cores)
data("ier")

# plot data

currency <- colnames(ier)
label <- c("US Dollar", "Canadian Dollar", "Yen", "Franc", "Lira", "Mark")
op <- par(mfrow = c(3, 2), mar = c(6, 6, 3, 3))
for (i in seq_along(currency)) {
  plot(ier[,currency[i]], type = "l", col = "darkblue", axes = FALSE,
       ylim = c(-4, 4), ylab = "", xlab = "", lwd = 2)
  axis(1, at = 0:12*12, labels = 1975:1987, cex.axis = 1.7)
  axis(2, at = pretty(c(-4, 4)), las = 1, cex.axis = 1.7)
  mtext("Year", 1, cex = 1.5, line = 3.2)
  mtext("Exchange Rate Changes", 2, cex = 1.4, line = 3.2)
  mtext(label[i], 3, cex = 1.6, line = .1)
}
par(op)

# stan model

model_code <-
  "data {
    int<lower=1> T; // number of observations
    int<lower=1> m; // number of variables
    int<lower=1> k; // number of factors
    matrix[T,m] Y; // data matrix
  }
  transformed data {
    int<lower = 1> r;
    vector[m] zeros;
    r = m * k - k * (k - 1) / 2; // number of non-zero factor loadings
    zeros = rep_vector(0.0, m);
  }
  parameters {
    matrix[k,m] Beta;
    real<lower=0> alpha;
    real<lower=0> beta;
    real<lower=0> sigma;
  }
  transformed parameters {
    matrix[k,m] Beta_t;
    real<lower=0> sigma_t;
    matrix[k,m] Sigma = sigma^2 * eye_array(k, m);
    real<lower=0> R:
    Beta_t = Beta / sigma_t;
    sigma_t = (1.0 / (1.0 + R * R)) * (1.0 / (1.0 + R * R))
  }
  model {
    Y ~ normal(Beta_t * Y, sigma_t * Sigma);
    alpha ~ normal(0, 1);
    beta ~ normal(0, 1);
    sigma ~ normal(1, 1);
    R ~ normal(0, 1);
    Beta ~ normal(Beta_t, Sigma);
  }
  generated quantities {
    correlation[k,m] correlation_matrix;
    for (i in 1:k) {
      for (j in 1:m) {
        correlation_matrix[i,j] = Beta[i,j] / sqrt(Beta[i,i] * Beta[j,j]);
      }
    }
  }"
```

```
parameters {
  real beta_lower[r - k]; // lower-diagonal elements of beta
  real<lower = 0> beta_diag[k]; // diagonal elements of beta
  vector<lower = 0>[m] sigma2; // residual variances
}

transformed parameters {
  matrix[m,k] beta;
  cov_matrix[m] Omega;
  // construct lower-triangular factor loadings matrix
  {
    int index_lower = 1;
    for (j in 1:k) {
      for (i in 1:m) {
        if (i == j) {
          beta[j,j] = beta_diag[j];
        } else if (i >= j) {
          beta[i,j] = beta_lower[index_lower];
          index_lower = index_lower + 1;
        } else {
          beta[i,j] = 0.0;
        }
      }
    }
  }
  Omega = beta * beta' + diag_matrix(sigma2);
}

model {
  // priors
  target += normal_lpdf(beta_diag | 0, 1) - k * normal_lccdf(0 | 0, 1);
  target += normal_lpdf(beta_lower | 0, 1);
  target += inv_gamma_lpdf(sigma2 | 2.2 / 2.0, 0.1 / 2.0);
  // likelihood
  for(t in 1:T) {
    target += multi_normal_lpdf(Y[t] | zeros, Omega);
  }
}

# compile model
model <- stan_model(model_code = model_code)

# function for generating starting values
init_fun <- function(nchains, k, m) {
  r <- m * k - k * (k - 1) / 2
  out <- vector("list", nchains)
  for (i in seq_len(nchains)) {
    beta_lower <- array(runif(r - k, 0.05, 1), dim = r - k)
  }
  out[[1]] <- list(beta_lower = beta_lower)
  return(out)
}
```r
beta_diag <- array(runif(k, .05, 1), dim = k)
sigma2 <- array(runif(m, .05, 1.5), dim = m)
out[[i]] <- list(beta_lower = beta_lower,
               beta_diag = beta_diag,
               sigma2 = sigma2)
}
return(out)

set.seed(1)
stanfit <- bridge <- vector("list", 3)
for (k in 1:3) {
  stanfit[[k]] <- sampling(model,
    data = list(Y = ier, T = nrow(ier),
                m = ncol(ier), k = k),
    iter = 11000, warmup = 1000, chains = 4,
    init = init_fun(nchains = 4, k = k, m = ncol(ier)),
    cores = cores, seed = 1)
  bridge[[k]] <- bridge_sampler(stanfit[[k]], method = "warp3",
                            repetitions = 10, cores = cores)
}
# example output
summary(bridge[[2]])

# compute posterior model probabilities
pp <- post_prob(bridge[[1]], bridge[[2]], bridge[[3]],
               model_names = c("k = 1", "k = 2", "k = 3"))
pp

op <- par(mar = c(6, 6, 3, 3))
boxplot(pp, axes = FALSE,
       ylim = c(0, 1), ylab = "",
       xlab = "")
axis(1, at = 1:3, labels = colnames(pp), cex.axis = 1.7)
axis(2, cex.axis = 1.1)
mtext("Posterior Model Probability", 2, cex = 1.5, line = 3.2)
mtext("Number of Factors", 1, cex = 1.4, line = 3.2)
par(op)

## End(Not run)
```

---

**logml**  
Log Marginal Likelihoods from Bridge Objects
Description

Generic function that returns log marginal likelihood from bridge objects. For objects of class "bridge_list", which contains multiple log marginal likelihoods, fun is performed on the vector and its result returned.

Usage

\[
\text{logml}(x, \ldots)
\]

## S3 method for class 'bridge'
\[
\text{logml}(x, \ldots)
\]

## S3 method for class 'bridge_list'
\[
\text{logml}(x, \text{fun} = \text{median}, \ldots)
\]

Arguments

\(x\) Object of class "bridge" or "bridge_list" as returned from \texttt{bridge_sampler}.

\(\ldots\) Further arguments passed to \texttt{fun}.

\(\text{fun}\) Function which returns a scalar value and is applied to the \texttt{logml} vector of "bridge_list" objects. Default is \texttt{median}.

Value

scalar numeric

---

\textbf{post_prob} \hspace{1cm} \textit{Posterior Model Probabilities from Marginal Likelihoods}

Description

Generic function that computes posterior model probabilities from marginal likelihoods.

Usage

\[
\text{post_prob}(x, \ldots, \text{prior_prob} = \text{NULL}, \text{model_names} = \text{NULL})
\]

## S3 method for class 'bridge'
\[
\text{post_prob}(x, \ldots, \text{prior_prob} = \text{NULL}, \text{model_names} = \text{NULL})
\]

## S3 method for class 'bridge_list'
\[
\text{post_prob}(x, \ldots, \text{prior_prob} = \text{NULL}, \text{model_names} = \text{NULL})
\]

## Default S3 method:
\[
\text{post_prob}(x, \ldots, \text{prior_prob} = \text{NULL}, \text{model_names} = \text{NULL})
\]
Arguments

- `x`: Object of class "bridge" or "bridge_list" as returned from `bridge_sampler`. Additionally, the default method assumes that all passed objects are numeric log marginal likelihoods (e.g., from `logml`) and will throw an error otherwise.

- `...`: further objects of class "bridge" or "bridge_list" as returned from `bridge_sampler`. Or numeric values for the default method.

- `prior_prob`: numeric vector with prior model probabilities. If omitted, a uniform prior is used (i.e., all models are equally likely a priori). The default `NULL` corresponds to equal prior model weights.

- `model_names`: If `NULL` (the default) will use model names derived from deparsing the call. Otherwise will use the passed values as model names.

Value

For the default method and the method for "bridge" objects, a named numeric vector with posterior model probabilities (i.e., which sum to one).

For the method for "bridge_list" objects, a matrix consisting of posterior model probabilities where each row sums to one and gives the model probabilities for one set of logmls. The (named) columns correspond to the models and the number of rows is given by the "bridge_list" element with the most repetitions. Elements with fewer repetitions will be recycled (with warning).

Note

For realistic examples, see `bridge_sampler` and the accompanying vignettes:

- `vignette("bridgesampling_example_jags")`
- `vignette("bridgesampling_example_stan")`

Author(s)

Quentin F. Gronau and Henrik Singmann

Examples

```r
H0 <- structure(list(logml = -20.8084543022433, niter = 4, method = "normal"),
   .Names = c("logml", "niter", "method"), class = "bridge")
H1 <- structure(list(logml = -17.9623077558729, niter = 4, method = "normal"),
   .Names = c("logml", "niter", "method"), class = "bridge")
H2 <- structure(list(logml = -19, niter = 4, method = "normal"),
   .Names = c("logml", "niter", "method"), class = "bridge")

post_prob(H0, H1, H2)
post_prob(H1, H0)

## all produce the same (only names differ):
post_prob(H0$logml, H1$logml, H2$logml)
post_prob(c(H0$logml, H1$logml, H2$logml))
```
turtles

21

post_prob(H0$logml, c(H1$logml, H2$logml))
post_prob(H0$logml, c(H1$logml, H2$logml), model_names = c("H0", "H1", "H2"))

### with bridge list elements:
H0L <- structure(list(logml = c(-20.8088381186739, -20.8072772698116,
-20.808454454621, -20.808419072281, -20.808780541247, -20.8084887398113,
-20.8086023582344, -20.8079083169745, -20.8083048489095, -20.8090050811436
), niter = c(4, 4, 4, 4, 4, 4, 4, 4, 3, 4), method = "normal",
repetitions = 10), .Names = c("logml", "niter", "method",
"repetitions"), class = "bridge_list")

H1L <- structure(list(logml = c(-17.961665507006, -17.9611290723151,
-17.9607509604499, -17.9608629535992, -17.9602093576442, -17.9600223300432,
-17.9610157118017, -17.9615557696561, -17.9608437034849, -17.9606743200309
), niter = c(4, 4, 4, 4, 4, 4, 4, 3, 4), method = "normal",
repetitions = 10), .Names = c("logml", "niter", "method",
"repetitions"), class = "bridge_list")

post_prob(H1L, H0L)
post_prob(H1L, H0L, H0) # last element recycled with warning.

---

### Turtles Data from Janzen, Tucker, and Paukstis (2000)

#### Description

This data set contains information about 244 newborn turtles from 31 different clutches. For each
turtle, the data set includes information about survival status (column y; 0 = died, 1 = survived),
birth weight in grams (column x), and clutch (family) membership (column clutch; an integer
between one and 31). The clutches have been ordered according to mean birth weight.

#### Usage

turtles

#### Format

A data.frame with 244 rows and 3 variables.

#### Source


Examples

```r
## Not run:
#
##########################################################
# BAYESIAN GENERALIZED LINEAR MIXED MODEL (PROBIT REGRESSION)
##########################################################

library(bridgesampling)
library(rstan)
data("turtles")

# plot data

# reproduce Figure 1 from Sinharay & Stern (2005)
xticks <- pretty(turtles$clutch)
yticks <- pretty(turtles$x)
plot(1, type = "n", axes = FALSE, ylab = "", xlab = "", xlim = range(xticks),
     ylim = range(yticks))
points(turtles$clutch, turtles$x, pch = ifelse(turtles$y, 21, 4), cex = 1.3,
       col = ifelse(turtles$y, "black", "darkred"), bg = "grey", lwd = 1.3)
axis(1, cex.axis = 1.4)
mtext("Clutch Identifier", side = 1, line = 2.9, cex = 1.8)
axis(2, las = 1, cex.axis = 1.4)
mtext("Birth Weight (Grams)", side = 2, line = 2.6, cex = 1.8)

# Analysis: Natural Selection Study (compute same BF as Sinharay & Stern, 2005)

### H0 (model without random intercepts) ###
H0_code <-
  "data {
    int<lower = 1> N;
    int<lower = 0, upper = 1> y[N];
    real<lower = 0> x[N];
  }
parameters {
    real alpha0_raw;
    real alpha1_raw;
  }
transformed parameters {
    real alpha0 = sqrt(10.0) * alpha0_raw;
    real alpha1 = sqrt(10.0) * alpha1_raw;
  }

```

turtles

```r
model {
    // priors
    target += normal_lpdf(alpha0_raw | 0, 1);
    target += normal_lpdf(alpha1_raw | 0, 1);

    // likelihood
    for (i in 1:N) {
        target += bernoulli_lpmf(y[i] | Phi(alpha0 + alpha1 * x[i]));
    }
}
```

### H1 (model with random intercepts) ###

```r
H1_code <-
"data {
    int<lower = 1> N;
    int<lower = 0, upper = 1> y[N];
    real<lower = 0> x[N];
    int<lower = 1> C;
    int<lower = 1, upper = C> clutch[N];
}
parameters {
    real alpha0_raw;
    real alpha1_raw;
    vector[C] b_raw;
    real<lower = 0> sigma2;
}
transformed parameters {
    vector[C] b;
    real<lower = 0> sigma = sqrt(sigma2);
    real alpha0 = sqrt(10.0) * alpha0_raw;
    real alpha1 = sqrt(10.0) * alpha1_raw;
    b = sigma * b_raw;
}
model {
    // priors
    target += - 2 * log(1 + sigma2); // p(sigma2) = 1 / (1 + sigma2) ^ 2
    target += normal_lpdf(alpha0_raw | 0, 1);
    target += normal_lpdf(alpha1_raw | 0, 1);

    // random effects
    target += normal_lpdf(b_raw | 0, 1);

    // likelihood
    for (i in 1:N) {
        target += bernoulli_lpmf(y[i] | Phi(alpha0 + alpha1 * x[i] + b[clutch[i]]));
    }
}
```

set.seed(1)

### fit models ###

```r
stanfit_H0 <- stan(model_code = H0_code,
    data = list(y = turtles$y, x = turtles$x, N = nrow(turtles)),
```
iter = 15500, warmup = 500, chains = 4, seed = 1)

stanfit_H1 <- stan(model_code = H1_code,
                   data = list(y = turtles$y, x = turtles$x, N = nrow(turtles),
                               C = max(turtles$clutch), clutch = turtles$clutch),
                   iter = 15500, warmup = 500, chains = 4, seed = 1)

set.seed(1)
### compute (log) marginal likelihoods ###
bridge_H0 <- bridge_sampler(stanfit_H0)
bridge_H1 <- bridge_sampler(stanfit_H1)

### compute approximate percentage errors ###
error_measures(bridge_H0)$percentage
error_measures(bridge_H1)$percentage

### summary ###
summary(bridge_H0)
summary(bridge_H1)

### compute Bayes factor ("true" value: BF01 = 1.273) ###
bf(bridge_H0, bridge_H1)

## End(Not run)
Index

+Topic dataset
  ier, 15
  turtles, 21
  .GlobalEnv, 6–8

bayes_factor (bf), 2
bf, 2, 9
bridge-methods, 4
bridge_sampler, 2–4, 5, 14, 19, 20

error_measures, 4, 9, 13

ier, 15

logml, 2, 3, 18, 20

makeCluster, 6
mclapply, 6, 8
median, 19

post_prob, 9, 19
print.bridge (bridge-methods), 4
print.bridge_list (bridge-methods), 4
print.summary.bridge (bridge-methods), 4
print.summary.bridge_list (bridge-methods), 4

summary.bridge (bridge-methods), 4
summary.bridge_list (bridge-methods), 4

turtles, 21