Package 'bridgesampling'

June 27, 2017

Type Package

Title Bridge Sampling for Marginal Likelihoods and Bayes Factors

Version 0.2-2

Depends R (>= 3.0.0)

Imports mvtnorm, Matrix, Brobdingnag, stringr, coda, parallel, scales, utils, methods

Suggests testthat, Rcpp, RcppEigen, R2jags, knitr, rmarkdown, R.rsp, BayesFactor, rstan

Description Provides functions for estimating marginal likelihoods, Bayes factors, posterior model probabilities, and normalizing constants in general, via different versions of bridge sampling (Meng & Wong, 1996, <http://www3.stat.sinica.edu.tw/statistica/j6n4/j6n43/j6n43.htm>).

License GPL (>= 2)

LazyData true

RoxygenNote 6.0.1

VignetteBuilder knitr, R.rsp

URL https://github.com/quentingronau/bridgesampling

NeedsCompilation no

Author Quentin F. Gronau [aut, cre],
Henrik Singmann [aut],
Jonathan J. Forster [ctb],
Eric-Jan Wagenmakers [ths],
The JASP Team [ctb],
Jiqiang Guo [ctb],
Jonah Gabry [ctb],
Ben Goodrich [ctb]

Maintainer Quentin F. Gronau <Quentin.F.Gronau@gmail.com>

Repository CRAN

Date/Publication 2017-06-27 08:24:15 UTC

R topics documented:

---

bf *Bayes Factor(s) from Marginal Likelihoods*

Description

Generic function that computes Bayes factor(s) from marginal likelihoods.

Usage

```
bf(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.
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.

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 vignette:
vignette("bridgesampling_example")

Author(s)

Quentin F. Gronau

References

Kass, R. E., & Raftery, A. E. (1995). Bayes factors. *Journal of the American Statistical Association*, 90(430), 773-795. http://dx.doi.org/10.1080/01621459.1995.10476572

---

bridge_sampler *Log Marginal Likelihood via Bridge Sampling*

Description

Computes log marginal likelihood via bridge sampling.

Usage

```
bridge_sampler(samples, ...)

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

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

Arguments

samples	a matrix with posterior samples (colnames need to correspond to parameter names in lb and ub) or a fitted stanfit object with posterior samples.
...	additional arguments passed to log_posterior for the matrix method. Ignored for the stanfit method.
log_posterior	function or name of function that takes a single row of samples and the data 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.
repetitions	number of repetitions.
method	either "normal" or "warp3".
cores	number of cores used for evaluating log_posterior. On unix-like systems (where .Platform$OS.type == "unix" evaluates to TRUE; e.g., Linux and Mac OS) forking via mclapply is used. Hence elements needed for evaluation should be in the .GlobalEnv. For other systems (e.g., Windows) makeCluster is used and further arguments specified below will be used.
packages	character vector with names of packages needed for evaluating log_posterior in parallel (only relevant if cores > 1 and .Platform$OS.type != "unix").
varlist	character vector with names of variables needed for evaluating log_posterior (only needed if cores > 1 and .Platform$OS.type != "unix" as these objects will be exported to the nodes). These objects need to exist in envir.
envir	specifies the environment for varlist (only needed if cores > 1 and .Platform$OS.type != "unix" as these objects will be exported to the nodes). Default is .GlobalEnv.
rcppFile	in case cores > 1 and log_posterior is an Rcpp function, rcppFile specifies the path to the cpp file (will be compiled on all cores).

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.
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).

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 column means of samples and covariance matrix equal to the sample covariance matrix of samples. For a recent tutorial on bridge sampling, see Gronau et al. (2017).

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 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 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.

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.

References

Gronau, Q. F., Sarafoglou, A., Matzke, D., Ly, A., Boehm, U., Marsman, M., Leslie, D. S., Forster, J. J., Wagenmakers, E.-J., & Steingroever, H. (2017). *A tutorial on bridge sampling*. Manuscript submitted for publication. https://arxiv.org/abs/1703.05984
`vignette("bridgesampling_tutorial")`

Meng, X.-L., & Wong, W. H. (1996). Simulating ratios of normalizing constants via a simple identity: A theoretical exploration. *Statistica Sinica*, 6, 831-860. http://www3.stat.sinica.edu.tw/statistica/j6n4/j6n43/j6n43.htm

Meng, X.-L., & Schilling, S. (2002). Warp bridge sampling. *Journal of Computational and Graphical Statistics*, 11(3), 552-586. http://dx.doi.org/10.1198/106186002457

Overstall, A. M., & Forster, J. J. (2010). Default Bayesian model determination methods for generalised linear mixed models. *Computational Statistics & Data Analysis*, 54, 3269-3288. http://dx.doi.org/10.1016/j.csda.2010.03.008

Examples

```
## ------------------------------------------------------------------------
## 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")

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
tau20 <- 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)
  cn <- colnames(s$BUGSoutput$sims.matrix)
  samples_matrix <- s$BUGSoutput$sims.matrix[ ,-which(cn == "deviance")] # cut off deviance

  return(samples_matrix)

}

### 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)
  cn <- colnames(s$BUGSoutput$sims.matrix)
  samples_matrix <- s$BUGSoutput$sims.matrix[ ,-which(cn == "deviance")] # cut off deviance

  return(samples_matrix)

}

### 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 ###
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
lb_H0 <- rep(-Inf, ncol(samples_H0))
ub_H0 <- rep(Inf, ncol(samples_H0))
names(lb_H0) <- names(ub_H0) <- colnames(samples_H0)
lb_H0[[ "invTau2" ]] <- 0

# specify parameter bounds H1
lb_H1 <- rep(-Inf, ncol(samples_H1))
ub_H1 <- rep(Inf, ncol(samples_H1))
names(lb_H1) <- names(ub_H1) <- colnames(samples_H1)
lb_H1[[ "invTau2" ]] <- 0


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

# 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)
```

`error_measures` *Error Measures for Estimated Marginal Likelihood*

Description

Computes error measures for estimated marginal likelihood.

Usage

```
error_measures(bridge_object)
```

Arguments

`bridge_object`	an object of class `"bridge"` as returned from `bridge_sampler`.

Details

Computes approximate error measures for marginal likelihood bridge sampling estimates. Based on Fruehwirth-Schnatter (2004).

Value

a list with the objects:

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

Note

For examples, see bridge_sampler and the accompanying vignette:
vignette("bridgesampling_example")

Author(s)

Quentin F. Gronau

References

Fruehwirth-Schnatter, S. (2004). Estimating marginal likelihoods for mixture and Markov switching models using bridge sampling techniques. *The Econometrics Journal*, 7, 143-167. http://dx.doi.org/10.1111/j.1368-423X.2004.00125.x

---

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

```
logml(x, ...)

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

## S3 method for class 'bridge_list'
logml(x, fun = median, ...)
```

Arguments

x	Object of class "bridge" or "bridge_list" as returned from bridge_sampler.
...	Further arguments passed to fun.
fun	Function which returns a scalar value and is applied to the logml vector of "bridge_list" objects. Default is median.

Value

scalar numeric

---

post_prob *Posterior Model Probabilities from Marginal Likelihoods*

Description

Generic function that computes posterior model probabilities from marginal likelihoods.

Usage

```
post_prob(x, ..., prior_prob = NULL, model_names = NULL)

## S3 method for class 'bridge'
post_prob(x, ..., prior_prob = NULL, model_names = NULL)

## S3 method for class 'bridge_list'
post_prob(x, ..., prior_prob = NULL,
  model_names = NULL)

## Default S3 method:
post_prob(x, ..., prior_prob = NULL, model_names = 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 vignette:
vignette("bridgesampling_example")

Author(s)

Quentin F. Gronau and Henrik Singmann

Examples

```
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, H1, H2)
post_prob(H0$logml, H1$logml, H2$logml)
post_prob(c(H0$logml, H1$logml, H2$logml))
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, -20.8083419072281, -20.8087870541247, -20.8084887398113,
-20.8086023582344, -20.8079083169745, -20.8083048489095, -20.8090050811436
), niter = c(4, 4, 4, 4, 4, 4, 4, 4, 4, 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, 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.
```

Index