Package 'thamesblock'

July 4, 2026

Type Package

Title Truncated Harmonic Mean Estimator of the Marginal Likelihood for Block Models

Version 0.1.0

Description Implements the truncated harmonic mean estimator (THAMES) and other estimators of the reciprocal marginal likelihood for block models. This is done via reciprocal importance sampling, using posterior samples and unnormalized log posterior values. For further information see Metodiev, Perrot-Dockès, Fouetilou, Latouche & Raftery (2026).

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.2

Imports mclust, stats, combinat, withr, Matrix, label.switching

NeedsCompilation no

Author Martin Metodiev [aut, cre, cph] (ORCID: <https://orcid.org/0009-0000-9432-3756>)

Maintainer Martin Metodiev <m.metodiev@tutanota.com>

Repository CRAN

Date/Publication 2026-07-04 08:40:08 UTC

Contents

`ChibPartition` *Computes the ChibPartition estimator*

Description

Computes the ChibPartition estimator

Usage

```
ChibPartition(lps, clusterList, num_clusters)
```

Arguments

`lps`	log-prior + log-likelihood values (collapsed sample)
`clusterList`	the posterior sample of cluster allocations
`num_clusters`	the number of clusters

Value

an estimator of the reciprocal log marginal likelihood

References

Hairault, A., Robert, C. P., Rousseau, J. (2022). Evidence estimation in finite and infinite mixture models and applications. arXiv preprint arXiv:2205.05416.

Examples

```
num_clusters = 2 # number of clusters
iterations = 6 # size of the MCMC sample
size_dataset = 10 # size of the data
# the MCMC sample (usually it is much larger than this)
clustermat = rbind(c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
c(0, 0, 0, 0, 1, 0, 0, 0, 0, 0),
c(1, 0, 0, 0, 1, 0, 0, 1, 0, 0))
clusterList = list(c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
c(0, 0, 0, 0, 1, 0, 0, 0, 0, 0),
c(1, 0, 0, 0, 1, 0, 0, 1, 0, 0))

# an example dataset with one edge between nodes 1 and 2, no edge otherwise
dataset = matrix(0, nrow=size_dataset, ncol=size_dataset)
dataset[1,2] = 1
dataset[2,1] = 1
```

```
# should be implemented in Rcpp for optimal performance

logprior = function(clustermat){
dirichlet_hyperparameter_vector = rep(1, num_clusters)
logprior_values = numeric(iterations)

for(iter in (1:iterations)){
update_dirichlet_hyperparameter_vector = numeric(num_clusters)
for(cluster_index in 0:(num_clusters-1)){
 update_dirichlet_hyperparameter_vector[cluster_index + 1] =
   dirichlet_hyperparameter_vector[cluster_index + 1] +
   sum(clustermat[iter,] == cluster_index)
 }
 logprior_values[iter] = lgamma(sum(dirichlet_hyperparameter_vector)) +
 sum(lgamma(dirichlet_hyperparameter_vector)) -
 lgamma(sum(update_dirichlet_hyperparameter_vector)) -
 sum(lgamma(update_dirichlet_hyperparameter_vector))
 }
   return(logprior_values)
   }

loglik = function(clustermat){
alpha_hyperparameters = matrix(1, nrow=num_clusters, ncol=num_clusters)
beta_hyperparameters = matrix(1, nrow=num_clusters, ncol=num_clusters)
loglik_values = numeric(iterations)
for(iter in iterations){
updated_alpha_hyperparameters = matrix(nrow=num_clusters, ncol=num_clusters)
updated_beta_hyperparameters = matrix(nrow=num_clusters, ncol=num_clusters)
for(cluster_index_1 in 0:(num_clusters-1)){
 for(cluster_index_2 in 0:(num_clusters-1)){
   updated_alpha_hyperparameters[cluster_index_1 + 1,
                                 cluster_index_2 + 1] =
   alpha_hyperparameters[cluster_index_1 + 1,
                         cluster_index_2 + 1] +
   sum(dataset[which(clustermat[iter,] == cluster_index_1),
               which(clustermat[iter,] == cluster_index_2)])
   updated_beta_hyperparameters[cluster_index_1 + 1,
                                cluster_index_2 + 1] =
       beta_hyperparameters[cluster_index_1 + 1,
                            cluster_index_2 + 1] +
         sum(1 - dataset[which(clustermat[iter,] == cluster_index_1),
         which(clustermat[iter,] == cluster_index_2)])
   loglik_values[iter] = loglik_values[iter] +
        lgamma(alpha_hyperparameters[cluster_index_1 + 1,
                                     cluster_index_2 + 1] +
                  beta_hyperparameters[cluster_index_1 + 1,
                                       cluster_index_2 + 1]) +
       lgamma(updated_alpha_hyperparameters[cluster_index_1 + 1,
                                            cluster_index_2 + 1]) +
       lgamma(updated_beta_hyperparameters[cluster_index_1 + 1,
                                           cluster_index_2 + 1]) -
       lgamma(updated_alpha_hyperparameters[cluster_index_1 + 1,
```

```
                                                cluster_index_2 + 1] +
              updated_beta_hyperparameters[cluster_index_1 + 1,
                                           cluster_index_2 + 1]) -
        lgamma(alpha_hyperparameters[cluster_index_1 + 1,
                                     cluster_index_2 + 1]) -
        lgamma(beta_hyperparameters[cluster_index_1 + 1,
                                    cluster_index_2 + 1])
      }
    }
  }
 return(loglik_values)
 }
 logpost = function(clustermat) logprior(clustermat) + loglik(clustermat)
 lps = logpost(clustermat)
 ChibPartition(lps=lps, clusterList=clusterList, num_clusters=num_clusters)
```

`compute_nobile_identity` *Nobile's identity for the marginal likelihood*

Description

This function, partially copied from the thamesmix package, uses the identity from Nobile (2004, 2007) to compute an estimate of the marginal likelihood of a stochastic block model with G clusters given an estimate of the marginal likelihood of a stochastic block model with G-1 clusters and an estimate of the proportion of empty components.

Usage

```
compute_nobile_identity(
  logZhatGminus1,
  clustermat,
  dirichlet_vec,
  size_dataset
)
```

Arguments

`logZhatGminus1`	marginal likelihood estimate with one component less
`clustermat`	the posterior sample of cluster allocations
`dirichlet_vec`	hyperparameter-vector of the dirichlet prior
`size_dataset`	size of the data

Details

NOTE: It is important to verify that an estimate of the logarithm, not an estimate of the negative logarithm is used.

Value

estimate of the marginal likelihood for G

References

Nobile, A. (2004). On the posterior distribution of the number of components in a finite mixture. The Annals of Statistics 32(5), 2044–2073.

Nobile, A. (2007). Bayesian finite mixtures: a note on prior specification and posterior computation.arXiv preprint arXiv:0711.0458.

Martin Metodiev, Marie Perrot-Dockès, Guilhem Fouetilou, Pierre Latouche, Adrian E. Raftery. "Simulation-consistent Estimation of the Marginal Likelihood for Block Models."

Examples

```
num_clusters = 2 # number of clusters
iterations = 6 # size of the MCMC sample
size_dataset = 10 # size of the data
# the MCMC sample (usually it is much larger than this)
clustermat = rbind(c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
c(0, 0, 0, 0, 1, 0, 0, 0, 0, 0),
c(1, 0, 0, 0, 1, 0, 0, 1, 0, 0))

compute_nobile_identity(logZhatGminus1 = -909.49,
clustermat = clustermat,
dirichlet_vec = rep(1,num_clusters),
size_dataset=size_dataset)
```

---

harmonic_mean_estimator

Computes the harmonic mean estimator

Description

Computes the harmonic mean estimator

Usage

```
harmonic_mean_estimator(logliks)
```

Arguments

logliks	the log-likelihood values

Value

an estimator of the reciprocal log marginal likelihood

References

Newton, M. A., & Raftery, A. E. (1994). Approximate Bayesian inference with the weighted likelihood bootstrap. Journal of the Royal Statistical Society Series B: Statistical Methodology, 56(1), 3-26.

Examples

```
num_clusters = 2 # number of clusters
iterations = 6 # size of the MCMC sample
size_dataset = 10 # size of the data
# the MCMC sample (usually it is much larger than this)
clustermat = rbind(c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
c(0, 0, 0, 0, 1, 0, 0, 0, 0, 0),
c(1, 0, 0, 0, 1, 0, 0, 1, 0, 0))

# an example dataset with one edge between nodes 1 and 2, no edge otherwise
dataset = matrix(0, nrow=size_dataset, ncol=size_dataset)
dataset[1,2] = 1
dataset[2,1] = 1

# should be implemented in Rcpp for optimal performance

loglik = function(clustermat){
alpha_hyperparameters = matrix(1, nrow=num_clusters, ncol=num_clusters)
beta_hyperparameters = matrix(1, nrow=num_clusters, ncol=num_clusters)
loglik_values = numeric(iterations)
for(iter in iterations){
updated_alpha_hyperparameters = matrix(nrow=num_clusters, ncol=num_clusters)
updated_beta_hyperparameters = matrix(nrow=num_clusters, ncol=num_clusters)
for(cluster_index_1 in 0:(num_clusters-1)){
 for(cluster_index_2 in 0:(num_clusters-1)){
   updated_alpha_hyperparameters[cluster_index_1 + 1,
                                 cluster_index_2 + 1] =
   alpha_hyperparameters[cluster_index_1 + 1,
                         cluster_index_2 + 1] +
   sum(dataset[which(clustermat[iter,] == cluster_index_1),
               which(clustermat[iter,] == cluster_index_2)])
   updated_beta_hyperparameters[cluster_index_1 + 1,
                                cluster_index_2 + 1] =
       beta_hyperparameters[cluster_index_1 + 1,
                            cluster_index_2 + 1] +
         sum(1 - dataset[which(clustermat[iter,] == cluster_index_1),
         which(clustermat[iter,] == cluster_index_2)])
   loglik_values[iter] = loglik_values[iter] +
        lgamma(alpha_hyperparameters[cluster_index_1 + 1,
```

```
                                    cluster_index_2 + 1] +
                beta_hyperparameters[cluster_index_1 + 1,
                                      cluster_index_2 + 1]) +
        lgamma(updated_alpha_hyperparameters[cluster_index_1 + 1,
                                             cluster_index_2 + 1]) +
        lgamma(updated_beta_hyperparameters[cluster_index_1 + 1,
                                            cluster_index_2 + 1]) -
        lgamma(updated_alpha_hyperparameters[cluster_index_1 + 1,
                                               cluster_index_2 + 1] +
              updated_beta_hyperparameters[cluster_index_1 + 1,
                                           cluster_index_2 + 1]) -
        lgamma(alpha_hyperparameters[cluster_index_1 + 1,
                                     cluster_index_2 + 1]) -
        lgamma(beta_hyperparameters[cluster_index_1 + 1,
                                    cluster_index_2 + 1])
      }
    }
  }
  return(loglik_values)
  }
  logliks = loglik(clustermat)
  harmonic_mean_estimator(logliks=logliks)
```

`risvb` *Computes the RISVB estimator for block models*

Description

Computes the RISVB estimator for block models

Usage

```
risvb(num_clusters, lps, clustermat, split = TRUE)
```

Arguments

`num_clusters`	the number of clusters
`lps`	log-prior + log-likelihood values (collapsed sample)
`clustermat`	the posterior sample of cluster allocations
`split`	splits the sample in two parts if true Literature: "Simulation-consistent Estimation of the Marginal Likelihood for Block Models" by Martin Metodiev, Marie Perrot-Dockès, Guilhem Fouetilou, Pierre Latouche, and Adrian E. Raftery "Accurate Computation of Marginal Data Densities Using Variational Bayes" by Gholamreza Hajargasht and Tomasz Wozniak

Value

an estimator of the reciprocal log marginal likelihood

Examples

```
num_clusters = 2 # number of clusters
iterations = 6 # size of the MCMC sample
size_dataset = 10 # size of the data
# the MCMC sample (usually it is much larger than this)
clustermat = rbind(c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
c(0, 0, 0, 0, 1, 0, 0, 0, 0, 0),
c(1, 0, 0, 0, 1, 0, 0, 1, 0, 0))

# an example dataset with one edge between nodes 1 and 2, no edge otherwise
dataset = matrix(0, nrow=size_dataset, ncol=size_dataset)
dataset[1,2] = 1
dataset[2,1] = 1

# should be implemented in Rcpp for optimal performance

logprior = function(clustermat){
dirichlet_hyperparameter_vector = rep(1, num_clusters)
logprior_values = numeric(iterations)

for(iter in (1:iterations)){
update_dirichlet_hyperparameter_vector = numeric(num_clusters)
for(cluster_index in 0:(num_clusters-1)){
 update_dirichlet_hyperparameter_vector[cluster_index + 1] =
   dirichlet_hyperparameter_vector[cluster_index + 1] +
   sum(clustermat[iter,] == cluster_index)
 }
 logprior_values[iter] = lgamma(sum(dirichlet_hyperparameter_vector)) +
 sum(lgamma(dirichlet_hyperparameter_vector)) -
 lgamma(sum(update_dirichlet_hyperparameter_vector)) -
 sum(lgamma(update_dirichlet_hyperparameter_vector))
 }
   return(logprior_values)
   }

loglik = function(clustermat){
alpha_hyperparameters = matrix(1, nrow=num_clusters, ncol=num_clusters)
beta_hyperparameters = matrix(1, nrow=num_clusters, ncol=num_clusters)
loglik_values = numeric(iterations)
for(iter in iterations){
updated_alpha_hyperparameters = matrix(nrow=num_clusters, ncol=num_clusters)
updated_beta_hyperparameters = matrix(nrow=num_clusters, ncol=num_clusters)
for(cluster_index_1 in 0:(num_clusters-1)){
 for(cluster_index_2 in 0:(num_clusters-1)){
   updated_alpha_hyperparameters[cluster_index_1 + 1,
                                 cluster_index_2 + 1] =
   alpha_hyperparameters[cluster_index_1 + 1,
                         cluster_index_2 + 1] +
   sum(dataset[which(clustermat[iter,] == cluster_index_1),
```

```
            which(clustermat[iter,] == cluster_index_2)])
    updated_beta_hyperparameters[cluster_index_1 + 1,
                                 cluster_index_2 + 1] =
        beta_hyperparameters[cluster_index_1 + 1,
                             cluster_index_2 + 1] +
          sum(1 - dataset[which(clustermat[iter,] == cluster_index_1),
          which(clustermat[iter,] == cluster_index_2)])
    loglik_values[iter] = loglik_values[iter] +
         lgamma(alpha_hyperparameters[cluster_index_1 + 1,
                                      cluster_index_2 + 1] +
                    beta_hyperparameters[cluster_index_1 + 1,
                                         cluster_index_2 + 1]) +
        lgamma(updated_alpha_hyperparameters[cluster_index_1 + 1,
                                             cluster_index_2 + 1]) +
        lgamma(updated_beta_hyperparameters[cluster_index_1 + 1,
                                            cluster_index_2 + 1]) -
        lgamma(updated_alpha_hyperparameters[cluster_index_1 + 1,
                                               cluster_index_2 + 1] +
              updated_beta_hyperparameters[cluster_index_1 + 1,
                                           cluster_index_2 + 1]) -
        lgamma(alpha_hyperparameters[cluster_index_1 + 1,
                                     cluster_index_2 + 1]) -
        lgamma(beta_hyperparameters[cluster_index_1 + 1,
                                    cluster_index_2 + 1])
      }
    }
  }
  return(loglik_values)
  }
  logpost = function(clustermat) logprior(clustermat) + loglik(clustermat)
  lps = logpost(clustermat)
  risvb(num_clusters=num_clusters, lps=lps, clustermat=clustermat)
```

`thamesblock` *Computes the THAMES for the stochastic block model*

Description

Computes the THAMES for the stochastic block model

Usage

```
thamesblock(num_clusters, logpost, clustermat, split = TRUE, seed = 1)
```

Arguments

`num_clusters`	the number of clusters
`logpost`	log-prior + log-likelihood values (collapsed sample)
`clustermat`	the posterior sample of cluster allocations

split	splits the sample in two parts if true
seed	the seed

Value

an estimator of the reciprocal log marginal likelihood

References

Martin Metodiev, Marie Perrot-Dockès, Guilhem Fouetilou, Pierre Latouche, Adrian E. Raftery. "Simulation-consistent Estimation of the Marginal Likelihood for Block Models."

Examples

```
num_clusters = 2 # number of clusters
iterations = 6 # size of the MCMC sample
size_dataset = 10 # size of the data
# the MCMC sample (usually it is much larger than this)
clustermat = rbind(c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
c(0, 0, 0, 0, 1, 0, 0, 0, 0, 0),
c(1, 0, 0, 0, 1, 0, 0, 1, 0, 0))

# an example dataset with one edge between nodes 1 and 2, no edge otherwise
dataset = matrix(0, nrow=size_dataset, ncol=size_dataset)
dataset[1,2] = 1
dataset[2,1] = 1

# should be implemented in Rcpp for optimal performance

logprior = function(clustermat){
dirichlet_hyperparameter_vector = rep(1, num_clusters)
logprior_values = numeric(iterations)

for(iter in (1:iterations)){
update_dirichlet_hyperparameter_vector = numeric(num_clusters)
for(cluster_index in 0:(num_clusters-1)){
 update_dirichlet_hyperparameter_vector[cluster_index + 1] =
   dirichlet_hyperparameter_vector[cluster_index + 1] +
   sum(clustermat[iter,] == cluster_index)
 }
 logprior_values[iter] = lgamma(sum(dirichlet_hyperparameter_vector)) +
 sum(lgamma(dirichlet_hyperparameter_vector)) -
 lgamma(sum(update_dirichlet_hyperparameter_vector)) -
 sum(lgamma(update_dirichlet_hyperparameter_vector))
 }
   return(logprior_values)
   }

loglik = function(clustermat){
```

```
alpha_hyperparameters = matrix(1, nrow=num_clusters, ncol=num_clusters)
beta_hyperparameters = matrix(1, nrow=num_clusters, ncol=num_clusters)
loglik_values = numeric(iterations)
for(iter in iterations){
updated_alpha_hyperparameters = matrix(nrow=num_clusters, ncol=num_clusters)
updated_beta_hyperparameters = matrix(nrow=num_clusters, ncol=num_clusters)
for(cluster_index_1 in 0:(num_clusters-1)){
 for(cluster_index_2 in 0:(num_clusters-1)){
   updated_alpha_hyperparameters[cluster_index_1 + 1,
                                 cluster_index_2 + 1] =
   alpha_hyperparameters[cluster_index_1 + 1,
                         cluster_index_2 + 1] +
   sum(dataset[which(clustermat[iter,] == cluster_index_1),
               which(clustermat[iter,] == cluster_index_2)])
   updated_beta_hyperparameters[cluster_index_1 + 1,
                                cluster_index_2 + 1] =
       beta_hyperparameters[cluster_index_1 + 1,
                            cluster_index_2 + 1] +
         sum(1 - dataset[which(clustermat[iter,] == cluster_index_1),
         which(clustermat[iter,] == cluster_index_2)])
   loglik_values[iter] = loglik_values[iter] +
        lgamma(alpha_hyperparameters[cluster_index_1 + 1,
                                     cluster_index_2 + 1] +
                  beta_hyperparameters[cluster_index_1 + 1,
                                       cluster_index_2 + 1]) +
       lgamma(updated_alpha_hyperparameters[cluster_index_1 + 1,
                                            cluster_index_2 + 1]) +
       lgamma(updated_beta_hyperparameters[cluster_index_1 + 1,
                                           cluster_index_2 + 1]) -
       lgamma(updated_alpha_hyperparameters[cluster_index_1 + 1,
                                              cluster_index_2 + 1] +
             updated_beta_hyperparameters[cluster_index_1 + 1,
                                          cluster_index_2 + 1]) -
       lgamma(alpha_hyperparameters[cluster_index_1 + 1,
                                    cluster_index_2 + 1]) -
       lgamma(beta_hyperparameters[cluster_index_1 + 1,
                                   cluster_index_2 + 1])
     }
   }
 }
return(loglik_values)
}
logpost = function(clustermat) logprior(clustermat) + loglik(clustermat)
thamesblock(num_clusters=num_clusters, logpost=logpost, clustermat=clustermat)
```

Index