Package 'QAEnsemble'

January 20, 2025

Title Ensemble Quadratic and Affine Invariant Markov Chain Monte Carlo

Version 1.0.0

Description The Ensemble Quadratic and Affine Invariant Markov chain Monte Carlo algorithms provide an efficient way to perform Bayesian inference in difficult parameter space geometries. The Ensemble Quadratic Monte Carlo algorithm was developed by Militzer (2023) <doi:10.3847/1538-4357/ace1f1>. The Ensemble Affine Invariant algorithm was developed by Goodman and Weare (2010) <doi:10.2140/camcos.2010.5.65> and it was implemented in Python by Foreman-Mackey et al (2013) <doi:10.48550/arXiv.1202.3665>. The Quadratic Monte Carlo method was shown to perform better than the Affine Invariant method in the paper by Militzer (2023) <doi:10.3847/1538-4357/ace1f1> and the Quadratic Monte Carlo method is the default method used. The Chen-Shao Highest Posterior Density Estimation algorithm is used for obtaining credible intervals and the potential scale reduction factor diagnostic is used for checking the convergence of the chains.

License GPL (>= 2)

Encoding UTF-8

RoxygenNote 7.3.2

Suggests coda, diagram, expm, knitr, rmarkdown, svMisc

VignetteBuilder knitr

Imports stats

NeedsCompilation no

Author Weston Roda [aut, cre] (<https://orcid.org/0000-0001-7200-7605>),
Karsten Hempel [aut] (<https://orcid.org/0000-0003-3273-4247>),
Sasha van Katwyk [aut] (<https://orcid.org/0000-0003-3026-2063>),
Diepreye Ayabina [aut] (<https://orcid.org/0000-0002-7005-6734>),
Children's Hospital of Eastern Ontario [fnd],
Canada's Drug Agency [fnd],
Institute of Health Economics [cph]

Maintainer Weston Roda <wroda@ihe.ca>

Repository CRAN

Date/Publication 2025-01-09 14:50:05 UTC

Contents

`ensemblealg` *Ensemble MCMC algorithm (either Quadratic or Affine Invariant method)*

Description

This function runs the Ensemble Quadratic or Affine Invariant MCMC algorithm for Bayesian inference parameter estimation and it is based off of the papers by Militzer (2023), Goodman and Weare (2010), and Foreman-Mackey, Hogg, Lang, and Goodman (2013). The Ensemble Quadratic Monte Carlo algorithm was developed by Militzer (2023). The Ensemble Affine Invariant algorithm was developed by Goodman and Weare (2010) and it was implemented in Python by Foreman-Mackey, Hogg, Lang, and Goodman (2013). The Quadratic Monte Carlo method was shown to perform better than the Affine Invariant method in the paper by Militzer (2023) and the Quadratic method is the default method used in the 'ensemblealg' function.

Usage

```
ensemblealg(
  theta0,
  logfuns,
  T_iter,
  Thin_val,
  UseQuad = TRUE,
  a_par = NULL,
  ShowProgress = FALSE,
  ReturnCODA = FALSE
)
```

Arguments

`theta0`	The matrix of initial guesses for the MCMC chains
`logfuns`	A list object containing the log prior function and log likelihood function
`T_iter`	The number of iterations to run for each chain in the Ensemble MCMC algorithm
`Thin_val`	Every nth iteration is saved, where n is equal to the "Thin_val" parameter
`UseQuad`	If this bool is true, then the Ensemble Quadratic MCMC algorithm is used. Otherwise, the Ensemble Affine Invariant MCMC algorithm is used. (The default setting is true.)

a_par	The parameter 'a_par' is a performance parameter for the MCMC ensemble algorithm. (The default setting for the Quadratic algorithm is 'a_par' equal to 1.5 and the default setting for the Affine Invariant algorithm is 'a_par' equal to 2.)
ShowProgress	If this bool is true, then the progress of the algorithm is shown. Otherwise, the progress of the algorithm is not shown.(The default setting is false.)
ReturnCODA	If this bool is true, then the 'coda' package 'mcmc.list' object is returned along with the other outputs (The default setting is false.)

Value

A list object is returned that contains three matrices: theta_sample, log_like_sample, and log_prior_sample.

theta_sample: this is the matrix of parameter samples returned from the Ensemble MCMC algorithm, the matrix dimensions are given by (Number of parameters) x (Number of chains) x (Number of iterations)

log_like_sample: this is the matrix of log likelihood samples returned from the Ensemble MCMC algorithm, the matrix dimensions are given by (Number of chains) x (Number of iterations)

log_prior_sample: this is the matrix of log prior samples returned from the Ensemble MCMC algorithm, the matrix dimensions are given by (Number of chains) x (Number of iterations)

mcmc_list_coda: (optional) this is the 'coda' package 'mcmc.list' object that can be used with various MCMC diagnostic functions in the 'coda' package

References

Militzer B (2023) Study of Jupiter's Interior with Quadratic Monte Carlo Simulations. ApJ 953(111):20pp. https://doi.org/10.3847/1538-4357/ace1f1

Goodman J and Weare J (2010) Ensemble samplers with affine invariance. Commun Appl Math Comput Sci 5(1):65-80. https://doi.org/10.2140/camcos.2010.5.65

Foreman-Mackey D, Hogg DW, Lang D, Goodman J (2013) emcee: The MCMC Hammer. PASP 125(925):306. https://doi.org/10.48550/arXiv.1202.3665

Examples

```
#Ensemble Quadratic MCMC algorithm example for fitting a Weibull
#distribution

#Assume the true parameters are

a_shape = 20
sigma_scale = 900
```

```
#Random sample from the Weibull distribution with a = 20 and sigma = 900,
#Y~WEI(a = 20, sigma = 900)

num_ran_samples = 50

data_weibull = matrix(NA, nrow = 1, ncol = num_ran_samples)

#Set the seed for this example
set.seed(10)

data_weibull = rweibull(num_ran_samples, shape = a_shape, scale = sigma_scale)

#We want to estimate a_shape and sigma_scale

#Log prior function for a_shape and sigma_scale
#(assumed priors a_shape ~ U(1e-2, 1e2) and sigma_scale ~ U(1, 1e4))
logp <- function(param)
{
 a_shape_use = param[1]
 sigma_scale_use = param[2]

 logp_val = dunif(a_shape_use, min = 1e-2, max = 1e2, log = TRUE) +
 dunif(sigma_scale_use, min = 1, max = 1e4, log = TRUE)

 return(logp_val)
}

#Log likelihood function for a_shape and sigma_scale
logl <- function(param)
{
 a_shape_use = param[1]
 sigma_scale_use = param[2]

 logl_val = sum(dweibull(data_weibull, shape = a_shape_use, scale = sigma_scale_use, log = TRUE))

 return(logl_val)
}

logfuns = list(logp = logp, logl = logl)

num_par = 2

#It is recommended to use at least twice as many chains as the number of
#parameters to be estimated.
num_chains = 2*num_par

#Generate initial guesses for the MCMC chains
theta0 = matrix(0, nrow = num_par, ncol = num_chains)

temp_val = 0
j = 0

while(j < num_chains)
```

```
{
 initial = c(runif(1, 1e-2, 1e2), runif(1, 1, 1e4))
 temp_val = logl(initial) + logp(initial)

 while(is.na(temp_val) || is.infinite(temp_val))
 {
   initial = c(runif(1, 1e-2, 1e2), runif(1, 1, 1e4))
   temp_val = logl(initial) + logp(initial)
 }

 j = j + 1

 message(paste('j:', j))

 theta0[1,j] = initial[1]
 theta0[2,j] = initial[2]

}

num_chain_iterations = 1e4
thin_val_par = 10

#The total number of returned samples is given by
#(num_chain_iterations/thin_val_par)*num_chains = 4e3

#Ensemble Quadratic MCMC algorithm

Weibull_Quad_result = ensemblealg(theta0, logfuns,
T_iter = num_chain_iterations, Thin_val = thin_val_par)

my_samples = Weibull_Quad_result$theta_sample

my_log_prior = Weibull_Quad_result$log_prior_sample

my_log_like = Weibull_Quad_result$log_like_sample

#Burn-in 25% of each chain
my_samples_burn_in = my_samples[,,-c(1:floor((num_chain_iterations/thin_val_par)*0.25))]

my_log_prior_burn_in = my_log_prior[,-c(1:floor((num_chain_iterations/thin_val_par)*0.25))]

my_log_like_burn_in = my_log_like[,-c(1:floor((num_chain_iterations/thin_val_par)*0.25))]

#Calculate potential scale reduction factors
diagnostic_result = psrfdiagnostic(my_samples_burn_in, 0.05)

diagnostic_result$p_s_r_f_vec

#log unnormalized posterior samples
log_un_post_vec = as.vector(my_log_prior_burn_in + my_log_like_burn_in)

#a_shape posterior samples
k1 = as.vector(my_samples_burn_in[1,,])
```

```
#sigma_scale posterior samples
k2 = as.vector(my_samples_burn_in[2,,])

#Calculate posterior median, 95% credible intervals, and maximum posterior for
#the parameters
median(k1)
hpdparameter(k1, 0.05)

median(k2)
hpdparameter(k2, 0.05)

k1[which.max(log_un_post_vec)]

k2[which.max(log_un_post_vec)]

#These plots display the silhouette of the unnormalized posterior surface from
#the chosen parameter's perspective

plot(k1, exp(log_un_post_vec), xlab="a_shape", ylab="unnormalized posterior density")

plot(k2, exp(log_un_post_vec), xlab="sigma_scale", ylab="unnormalized posterior density")
```

`hpdparameter` *Highest Posterior Density (HPD) for a parameter*

Description

This function returns the upper and lower bound of the Highest Posterior Density (HPD) for a parameter based on the Chen-Shao Highest Posterior Density (HPD) Estimation Algorithm found in the book by Chen, Shao, and Ibrahim (2000). (The smallest 95% credible interval will be given by the HPD using alpha = 0.05)

Usage

```
hpdparameter(parameter_MCMC, alpha = 0.05)
```

Arguments

`parameter_MCMC`	a vector of the parameter samples for a single estimated parameter
`alpha`	100(1 - alpha)% credible interval with the default value as alpha = 0.05

Value

A vector is returned that contains the lower and upper bound of the Highest Posterior Density (HPD) for a parameter (this will be the smallest 95% credible interval using alpha = 0.05)

References

Chen M, Shao Q, Ibrahim JG (2000) Monte Carlo Methods in Bayesian Computation. New York-New York: Springer-Verlag.

Examples

```
x_parameter = rnorm(75, mean = 0, sd = 1)

hpdparameter(x_parameter, 0.05)
```

`psrfdiagnostic` *Potential Scale Reduction Factor computation*

Description

This function computes the potential scale reduction factor for each parameter to formally test the convergence of the MCMC sampling to the estimated posterior distribution which was developed by Gelman and Brooks (1998). This potential scale reduction factor is based on empirical interval lengths with the following formula: $\hat{R} = \frac{S}{\sum_{i=1}^{K} \frac{s_i}{K}}$, where S is the distance between the upper and lower values of the $100(1-\alpha)\%$ interval for the pooled samples, s_i is the distance between the upper and lower values of the $100(1-\alpha)\%$ interval for the i^{th} chain, and K is the total number of chains used. When the potential scale reduction factor is close to 1 for all the estimated parameters, this indicates that the MCMC sampling converged to the estimated posterior distribution for each parameter.

Usage

```
psrfdiagnostic(my_samples_burn_in, alpha = 0.05)
```

Arguments

`my_samples_burn_in`	This parameter is a matrix of parameter samples returned from the Ensemble MCMC algorithm 'ensemblealg', the matrix dimensions are given by (Number of parameters) x (Number of chains) x (Number of iterations - Number of burn in iterations). It is recommended to burn-in the parameter samples from the starting iterations before running the 'psrfdiagnostic' to assess the convergence.
`alpha`	the alpha value here corresponds to the 100(1 - alpha)% credible intervals to be estimated, with the default value as alpha = 0.05

Value

A list object is returned that contains two vectors and one matrix: p_s_r_f_vec, L_vec, and d_matrix.

p_s_r_f_vec: this is the vector of potential scale reduction factors in order of the parameters

L_vec: this is the vector of distances between the upper and lower values of the 95% interval for the pooled samples and these distances are in order of the parameters

d_matrix: this is the matrix of distances between the upper and lower values of the 95% interval for the samples in each of the chains, the matrix dimensions are given by (Number of parameters) x (Number of chains)

References

Brooks SP and Gelman A (1998) General methods for monitoring convergence of iterative simulations. J Comp Graph Stat 7(4):434-455.

Examples

```
#Take 100 random samples from a multivariate normal distribution
#with mean c(1, 2) and covariance matrix matrix(c(1, 0.75, 0.75, 1), nrow = 2, ncol = 2)
#for each of four chains.

my_samples_example = array(0, dim=c(2, 4, 100))

for(j in 1:4)
{
  for(i in 1:100)
  {
    my_samples_example[,j,i] = solve(matrix(c(1, 0.75, 0.75, 1), nrow = 2, ncol = 2))%*%
    rnorm(2, mean = 0, sd = 1) +  matrix(c(1, 2), nrow = 2, ncol = 1, byrow = TRUE)
  }
}

#The potential scale reduction factors for each parameter are close to 1
psrfdiagnostic(my_samples_example)$p_s_r_f_vec
```

Index