Package 'scalablebayesm'

February 25, 2025

Type Package

Title Distributed Markov Chain Monte Carlo for Bayesian Inference in Marketing

Version 0.2

Date 2025-01-28

Maintainer Federico Bumbaca <federico.bumbaca@colorado.edu>

Description Estimates unit-level and population-level parameters from a hierarchical model in marketing applications. The package includes: Hierarchical Linear Models with a mixture of normals prior and covariates, Hierarchical Multinomial Logits with a mixture of normals prior and covariates, Hierarchical Multinomial Logits with a Dirichlet Process prior and covariates. For more details, see Bumbaca, F. (Rico), Misra, S., & Rossi, P. E. (2020) <doi:10.1177/0022243720952410> ``Scalable Target Marketing: Distributed Markov Chain Monte Carlo for Bayesian Hierarchical Models". Journal of Marketing Research, 57(6), 999-1018.

License GPL (>= 2)

Encoding UTF-8

RoxygenNote 7.3.1

Imports Rcpp (>= 1.0.9), parallel, bayesm

LinkingTo Rcpp, RcppArmadillo, bayesm

NeedsCompilation yes

Author Federico Bumbaca [aut, cre], Jackson Novak [aut]

Repository CRAN

Date/Publication 2025-02-25 12:30:02 UTC

Contents

`combine_draws` *Combine Lists of Draws From a Posterior Predictive Distribution*

Description

`combine_draws` combines and resamples parameter draws returned from an MCMC algorithm.

Usage

```
combine_draws(draws, r)
```

Arguments

`draws`	A list of draws from a posterior predictive distribution
`r`	Number of MCMC draws

Value

A matrix or data frame containing ‘R‘ randomly sampled rows from the combined ‘betadraw‘ components.

Author(s)

Federico Bumbaca, Leeds School of Business, University of Colorado Boulder, <federico.bumbaca@colorado.edu>

References

Bumbaca, F. (Rico), Misra, S., & Rossi, P. E. (2020). Scalable Target Marketing: Distributed Markov Chain Monte Carlo for Bayesian Hierarchical Models. Journal of Marketing Research, 57(6), 999-1018.

See Also

`rhierLinearMixtureParallel`, `rhierMnlRwMixtureParallel`, `rhierLinearDPParallel`, `rhierMnlDPParallel`

drawMixture *Gibbs Sampler Inference for a Mixture of Multivariate Normals*

Description

drawMixture implements a Gibbs sampler to conduct inference on draws from a multivariate normal mixture.

Usage

```
drawMixture(out, N, Z, Prior, Mcmc, verbose)
```

Arguments

out	A list containing compdraw, probdraw, and (optionally) Deltadraw.
N	An integer specifying the number of observational units to sample
Z	An $(nreg) \times nz$ or $(nlgt) \times nz$ matrix of unit characteristics
Prior	A list with one required parameter: 'ncomp', and optional parameters: 'mubar', 'Amu', 'nu', 'V', 'Ad', 'deltaBar', and 'a'.
Mcmc	A list with one required parameter: 'R' - number of iterations, and optional parameters: 's', 'w', 'keep', 'nprint', and 'drawcomp'.
verbose	If TRUE, enumerates model parameters and timing information.

Value

A list containing the following elements:

- **nmix**: A list with the following components:
  - **probdraw**: A matrix of size (R/keep) x (ncomp), containing the probability draws at each Gibbs iteration.
  - **compdraw**: A list containing the drawn mixture components at each Gibbs iteration.
- **Deltadraw** (optional): A matrix of size (R/keep) x (nz * nvar), containing the delta draws, if Z is not NULL. If Z is NULL, this element is not included.

Author(s)

Federico Bumbaca, Leeds School of Business, University of Colorado Boulder, <federico.bumbaca@colorado.edu>

References

Bumbaca, Federico (Rico), Sanjog Misra, and Peter E. Rossi (2020), "Scalable Target Marketing: Distributed Markov Chain Monte Carlo for Bayesian Hierarchical Models", Journal of Marketing Research, 57(6), 999-1018.

Chapter 5, *Bayesian Statistics and Marketing* by Rossi, Allenby, and McCulloch.

See Also

rhierLinearDPParallel, rhierMnlDPParallel,

Examples

```
# Linear DP
## Generate single component linear data with Z
R = 1000
nreg = 1000
nobs = 5 #number of observations
nvar = 3 #columns
nz = 2

Z=matrix(runif(nreg*nz),ncol=nz)
Z=t(t(Z)-apply(Z,2,mean))
Delta=matrix(c(1,0,1,0,1,2),ncol=nz)
tau0=1
iota=c(rep(1,nobs))

## create arguments for rmixture
tcomps=NULL
a = diag(1, nrow=3)
tcomps[[1]] = list(mu=c(1,-2,0),rooti=a)
tpvec = 1
ncomp=length(tcomps)

regdata=NULL
betas=matrix(double(nreg*nvar),ncol=nvar)
tind=double(nreg)

for (reg in 1:nreg) {
 tempout=bayesm::rmixture(1,tpvec,tcomps)
 if (is.null(Z)){
   betas[reg,]= as.vector(tempout$x)
 }else{
   betas[reg,]=Delta%*%Z[reg,]+as.vector(tempout$x)}
 tind[reg]=tempout$z
 X=cbind(iota,matrix(runif(nobs*(nvar-1)),ncol=(nvar-1)))
 tau=tau0*runif(1,min=0.5,max=1)
 y=X%*%betas[reg,]+sqrt(tau)*rnorm(nobs)
 regdata[[reg]]=list(y=y,X=X,beta=betas[reg,],tau=tau)
}

Prior1=list(ncomp=ncomp)
keep=1
Mcmc1=list(R=R,keep=keep)
Data1=list(list(regdata=regdata,Z=Z))

#subsample data
N = length(Data1[[1]]$regdata)
```

```
s=1

#Partition data into s shards
Data2 = partition_data(Data = Data1, s = s)

#Run distributed first stage
timing_result1 = system.time({
 out_distributed = parallel::mclapply(Data2, FUN = rhierLinearDPParallel,
 Prior = Prior1, Mcmc = Mcmc1, mc.cores = s, mc.set.seed = FALSE)
})

Z = matrix(unlist(Z), ncol = nz, byrow = TRUE)

# Conduct inference on first-stage draws
draws = parallel::mclapply(out_distributed, FUN = drawMixture,
Prior=Prior1, Mcmc=Mcmc1, N=N, Z = Z,
                           mc.cores = s, mc.set.seed = FALSE)

#Generate single component multinomial data with Z
##parameters
R = 1000
p = 3 # number of choice alternatives
ncoef = 3
nlgt=1000
nz = 2

# Define Z matrix
Z = matrix(runif(nz*nlgt),ncol=nz)
Z = t(t(Z)-apply(Z,2,mean))          # demean Z
Delta=matrix(c(1,0,1,0,1,2),ncol=2)

tcomps=NULL
a = diag(1, nrow=3)
tcomps[[1]] = list(mu=c(-1,2,4),rooti=a)
tpvec = 1
ncomp=length(tcomps)

simmnlwX= function(n,X,beta){
 k=length(beta)
 Xbeta=X %*% beta
 j=nrow(Xbeta)/n
 Xbeta=matrix(Xbeta,byrow=TRUE,ncol=j)
 Prob=exp(Xbeta)
 iota=c(rep(1,j))
 denom=Prob %*% iota
 Prob=Prob/as.vector(denom)
 y=vector("double",n)
 ind=1:j
 for (i in 1:n) {
   yvec = rmultinom(1, 1, Prob[i,])
   y[i] = ind%*%yvec
 }
```

```
 return(list(y=y,X=X,beta=beta,prob=Prob))
}

## simulate data
simlgtdata=NULL
ni=rep(5,nlgt)
for (i in 1:nlgt)
{
 if (is.null(Z))
 {
   betai=as.vector(bayesm::rmixture(1,tpvec,tcomps)$x)
 } else {
   betai=Delta %*% Z[i,]+as.vector(bayesm::rmixture(1,tpvec,tcomps)$x)
 }
 Xa=matrix(runif(ni[i]*p,min=-1.5,max=0),ncol=p)
 X=bayesm::createX(p,na=1,nd=NULL,Xa=Xa,Xd=NULL,base=1)
 outa=simmnlwX(ni[i],X,betai)
 simlgtdata[[i]]=list(y=outa$y,X=X,beta=betai)
}

## set parms for priors and Z
Prior1=list(ncomp=ncomp)
keep=1
Mcmc1=list(R=R,keep=keep)
Data1=list(list(lgtdata=simlgtdata, p=p, Z=Z))

N = length(Data1[[1]]$lgtdata)

s=1

#Partition data into s shards
Data2 = partition_data(Data = Data1, s = s)

#Run distributed first stage
timing_result1 = system.time({
 out_distributed = parallel::mclapply(Data2, FUN = rhierMnlDPParallel,
 Prior = Prior1, Mcmc = Mcmc1, mc.cores = s, mc.set.seed = FALSE)
})

#Conduct inference on first-stage draws
draws = parallel::mclapply(out_distributed, FUN = drawMixture,
Prior=Prior1, Mcmc=Mcmc1, N=N, Z = Z, mc.cores = s, mc.set.seed = FALSE)
```

`drawPosteriorParallel` *Draw from Posterior Parallel Distribution*

Description

`drawPosteriorParallel` draws from a posterior predictive distribution.

Usage

```
drawPosteriorParallel(draws, Z, Prior, Mcmc)
```

Arguments

`draws`	(list) - a list of length s where each sublist contains compdraw,
`Z`	(matrix) - (optional) an $nreg/s \times nz$ matrix of unit characteristics
`Prior`	(list) - (optional) a list of optional parameters 'v' and 'nu'
`Mcmc`	(list) - a list containing 'R' and optionally 'keep'

Value

A list containing:

- **betadraw**: A matrix of size $R \times nvar$ containing the drawn `beta` values from the Gibbs sampling procedure.

Author(s)

Federico Bumbaca, Leeds School of Business, University of Colorado Boulder, <federico.bumbaca@colorado.edu>

Federico Bumbaca, <federico.bumbaca@colorado.edu>

References

Bumbaca, F. (Rico), Misra, S., & Rossi, P. E. (2020). Scalable Target Marketing: Distributed Markov Chain Monte Carlo for Bayesian Hierarchical Models. Journal of Marketing Research, 57(6), 999-1018.

Examples

```
s=1
R=2000
nreg = 2000
nobs=5 #number of observations
nvar=3 #columns
nz=2

Z=NULL
Delta=matrix(c(1,0,1,0,1,2),ncol=nz)
tau0=1
iota=c(rep(1,nobs))

## create arguments for rmixture
```

```
#Default
tcomps=NULL
a = diag(1, nrow=3)
tcomps[[1]] = list(mu=c(0,-1,-2),rooti=a)
tpvec = 1
ncomp=length(tcomps)

regdata=NULL
betas=matrix(double(nreg*nvar),ncol=nvar)
tind=double(nreg)

for (reg in 1:nreg) {
 tempout=bayesm::rmixture(1,tpvec,tcomps)
 if (is.null(Z)){
   betas[reg,]= as.vector(tempout$x)
 }else{
   betas[reg,]=Delta%*%Z[reg,]+as.vector(tempout$x)}
 tind[reg]=tempout$z
 X=cbind(iota,matrix(runif(nobs*(nvar-1)),ncol=(nvar-1)))
 tau=tau0*runif(1,min=0.5,max=1)
 y=X%*%betas[reg,]+sqrt(tau)*rnorm(nobs)
 regdata[[reg]]=list(y=y,X=X,beta=betas[reg,],tau=tau)
}

Prior1=list(ncomp=ncomp)
keep=1
Mcmc1=list(R=R,keep=keep)
Data1=list(list(regdata=regdata,Z=Z))


Data2 = partition_data(Data1, s)

draws = parallel::mclapply(Data2, FUN = rhierLinearMixtureParallel, Prior = Prior1, Mcmc = Mcmc1,
mc.cores = s, mc.set.seed = TRUE)

out = parallel::mclapply(draws,FUN=drawPosteriorParallel,
Z=Z, Prior = Prior1, Mcmc = Mcmc1, mc.cores=s,
mc.set.seed = TRUE)
```

`hello` *A placeholder function using roxygen*

Description

This function shows a standard text on the console. In a time-honored tradition, it defaults to displaying *hello, world*.

Usage

```
hello(txt = "world")
```

Arguments

txt	An optional character variable, defaults to 'world'

Value

Nothing is returned but as a side effect output is printed

Examples

```
hello()
hello("and goodbye")
```

---

partition_data *Partition Data Into Shards*

Description

A function to partition data into s shards for use in distributed estimation.

Usage

```
partition_data(Data, s)
```

Arguments

Data	A list of containing either 'regdata' or 'lgtdata' and 'Z'(optional). If 'Data' contains 'lgtdata', it should also contain 'p' number of choice alternatives.
s	The number of shards to partition the data into.

Value

A list of 's' shards where each shard contains:

p	(integer) - Number of choice alternatives (only if 'Data' contains 'lgtdata')
lgtdata or regdata	(list, length: n) - A list of n elements where each element contains 'X', 'y', 'beta', and 'tau'
Z	(Matrix) - A n x nz matrix of units chars. Null if 'Data' does not contain Z [Optional]

Author(s)

Federico Bumbaca, Leeds School of Business, University of Colorado Boulder, <federico.bumbaca@colorado.edu>

References

Bumbaca, F. (Rico), Misra, S., & Rossi, P. E. (2020). Scalable Target Marketing: Distributed Markov Chain Monte Carlo for Bayesian Hierarchical Models. Journal of Marketing Research, 57(6), 999-1018.

Examples

```
# Generate hierarchical linear data
R=1000 #number of draws
nreg=2000 #number of observational units
nobs=5 #number of observations per unit
nvar=3 #columns
nz=2

Z=matrix(runif(nreg*nz),ncol=nz)
Z=t(t(Z)-apply(Z,2,mean))
Delta=matrix(c(1,-1,2,0,1,0), ncol = nz)
tau0=.1
iota=c(rep(1,nobs))

## create arguments for rmixture
tcomps=NULL
a = diag(1, nrow=3)
tcomps[[1]] = list(mu=c(-5,0,0),rooti=a)
tcomps[[2]] = list(mu=c(5, -5, 2),rooti=a)
tcomps[[3]] = list(mu=c(5,5,-2),rooti=a)
tpvec = c(.33,.33,.34)
ncomp=length(tcomps)
regdata=NULL
betas=matrix(double(nreg*nvar),ncol=nvar)
tind=double(nreg)
for (reg in 1:nreg) {
  tempout=bayesm::rmixture(1,tpvec,tcomps)
  if (is.null(Z)){
    betas[reg,]= as.vector(tempout$x)
  }else{
    betas[reg,]=Delta%*%Z[reg,]+as.vector(tempout$x)}
  tind[reg]=tempout$z
  X=cbind(iota,matrix(runif(nobs*(nvar-1)),ncol=(nvar-1)))
  tau=tau0*runif(1,min=0.5,max=1)
  y=X%*%betas[reg,]+sqrt(tau)*rnorm(nobs)
  regdata[[reg]]=list(y=y,X=X,beta=betas[reg,],tau=tau)
}

Prior1=list(ncomp=ncomp)
keep=1
Mcmc1=list(R=R,keep=keep)
Data1=list(list(regdata=regdata,Z=Z))

length(Data1)

Data2 = partition_data(Data1, s = 3)
```

```
length(Data2)
```

`rheteroLinearIndepMetrop` *Distributed Independence Metropolis-Hastings Algorithm for Draws From Multivariate Normal Distribution*

Description

`rheteroLinearIndepMetrop` implements an Independence Metropolis-Hastings algorithm with a Gibbs sampler.

Usage

```
rheteroLinearIndepMetrop(Data, betadraws, Mcmc, Prior)
```

Arguments

`Data`	A list of data partitions where each partition includes: 'regdata' - A $nreg$ size list of multinomial regdata, and optional 'Z'- $nreg \times nz$ matrix of unit characteristics.
`betadraws`	A list of betadraws returned from either `rhierLinearMixtureParallel` or `rhierLinearDPParallel`
`Mcmc`	A list with one required parameter: 'R'-number of iterations, and optional parameters: 'keep' and 'nprint'.
`Prior`	A list with one required parameter: 'ncomp', and optional parameters: 'deltabar', 'Ad', 'mubar', 'Amu', 'nu', 'V', 'nu.e', and 'ssq'.

Details

Model and Priors: `nreg` regression equations with `nvar` as the number of X vars in each equation
$y_i = X_i\beta_i + e_i$ with $e_i \sim N(0, \tau_i)$
$\tau_i \sim nu.e * ssq_i/\chi^2_{nu.e}$ where τ_i is the variance of e_i
$B = Z\Delta + U$ or $\beta_i = \Delta' Z[i,]' + u_i$
Δ is an $nz \times nvar$ matrix

$u_i \sim N(\mu_{ind}, \Sigma_{ind})$

$delta = vec(\Delta) \sim N(deltabar, A_d^{-1})$
$\mu_j \sim N(mubar, \Sigma_j(x)Amu^{-1})$
$\Sigma_j \sim IW(nu, V)$

$\theta_i = (\mu_i, \Sigma_i) \sim DP(G_0(\lambda), alpha)$

$G_0(\lambda)$:
$\mu_i|\Sigma_i \sim N(0, \Sigma_i(x)a^{-1})$
$\Sigma_i \sim IW(nu, nu * v * I)$
$delta = vec(\Delta) \sim N(deltabar, A_d^{-1})$

$\lambda(a, nu, v)$:
$a \sim$ uniform[alim[1], alimb[2]]
$nu \sim$ dim(data)-1 + exp(z)
$z \sim$ uniform[dim(data)-1+nulim[1], nulim[2]]
$v \sim$ uniform[vlim[1], vlim[2]]

Z should *not* include an intercept and should be centered for ease of interpretation. The mean of each of the `nreg` βs is the mean of the normal mixture. Use `summary()` to compute this mean from the `compdraw` output.

The prior on Σ_i is parameterized such that $mode(\Sigma) = nu/(nu + 2)vI$. The support of nu enforces a non-degenerate IW density; $nulim[1] > 0$

The default choices of alim, nulim, and vlim determine the location and approximate size of candidate "atoms" or possible normal components. The defaults are sensible given a reasonable scaling of the X variables. You want to insure that alim is set for a wide enough range of values (remember a is a precision parameter) and the v is big enough to propose Sigma matrices wide enough to cover the data range.

A careful analyst should look at the posterior distribution of a, nu, v to make sure that the support is set correctly in alim, nulim, vlim. In other words, if we see the posterior bunched up at one end of these support ranges, we should widen the range and rerun.

If you want to force the procedure to use many small atoms, then set nulim to consider only large values and set vlim to consider only small scaling constants. Set alphamax to a large number. This will create a very "lumpy" density estimate somewhat like the classical Kernel density estimates. Of course, this is not advised if you have a prior belief that densities are relatively smooth.

Argument Details: `Data = list(regdata, Z)` *[`Z` optional]*

`regdata:`	A $nreg/s_shard$ size list of regdata
`regdata[[i]]$X:`	$n_i \times nvar$ design matrix for equation i
`regdata[[i]]$y:`	$n_i \times 1$ vector of observations for equation i
`Z:`	A list of s partitions where each partition include $(nreg/s_shard) \times nz$ matrix of unit characteristics

`betadraw:` A matrix with R rows and $nvar$ columns of beta draws.

`Prior = list(deltabar, Ad, Prioralphalist, lambda_hyper, nu, V, nu_e, mubar, Amu, ssq, ncomp)` *[all but* `ncomp` *are optional]*

`deltabar:`	$(nz \times nvar) \times 1$ vector of prior means (def: 0)
`Ad:`	prior precision matrix for vec(D) (def: 0.01*I)
`mubar:`	$nvar \times 1$ prior mean vector for normal component mean (def: 0)
`Amu:`	prior precision for normal component mean (def: 0.01)
`nu.e:`	d.f. parameter for regression error variance prior (def: 3)
`V:`	PDS location parameter for IW prior on normal component Sigma (def: nu*I)
`ssq:`	scale parameter for regression error variance prior (def: `var(y_i)`)
`ncomp:`	number of components used in normal mixture

`Mcmc = list(R, keep, nprint)` *[only* `R` *required]*

`R:`	number of MCMC draws
`keep:`	MCMC thinning parameter – keep every `keep`pth draw (def: 1)
`nprint:`	print the estimated time remaining for every `nprint`'th draw (def: 100, set to 0 for no print)

Value

A list containing:

- **betadraw**: A matrix of size $R \times nvar$ containing the drawn `beta` values from the Gibbs sampling procedure.

Author(s)

Federico Bumbaca, Leeds School of Business, University of Colorado Boulder, <federico.bumbaca@colorado.edu>

References

Bumbaca, F. (Rico), Misra, S., & Rossi, P. E. (2020). Scalable Target Marketing: Distributed Markov Chain Monte Carlo for Bayesian Hierarchical Models. Journal of Marketing Research, 57(6), 999-1018.

See Also

`rhierLinearMixtureParallel`, `rheteroMnlIndepMetrop`

Examples

```
######### Single Component with rhierLinearMixtureParallel########
R = 500

set.seed(66)
nreg=1000
nobs=5 #number of observations
nvar=3 #columns
nz=2

Z=matrix(runif(nreg*nz),ncol=nz)
Z=t(t(Z)-apply(Z,2,mean))

Delta=matrix(c(1,-1,2,0,1,0),ncol=nz)
tau0=.1
iota=c(rep(1,nobs))

#Default
tcomps=NULL
a=matrix(c(1,0,0,0.5773503,1.1547005,0,-0.4082483,0.4082483,1.2247449),ncol=3)
```

```
tcomps[[1]]=list(mu=c(0,-1,-2),rooti=a)
tpvec=c(1)

regdata=NULL
betas=matrix(double(nreg*nvar),ncol=nvar)
tind=double(nreg)

for (reg in 1:nreg) {
 tempout=bayesm::rmixture(1,tpvec,tcomps)
 if (is.null(Z)){
   betas[reg,]= as.vector(tempout$x)
 }else{
   betas[reg,]=Delta %*% Z[reg,]+as.vector(tempout$x)}
 tind[reg]=tempout$z
 X=cbind(iota,matrix(runif(nobs*(nvar-1)),ncol=(nvar-1)))
 tau=tau0*runif(1,min=0.5,max=1)
 y=X %*% betas[reg,]+sqrt(tau)*rnorm(nobs)
 regdata[[reg]]=list(y=y,X=X,beta=betas[reg,],tau=tau)
}


Data1=list(list(regdata=regdata,Z=Z))
s = 1
Data2=scalablebayesm::partition_data(Data1,s=s)

Prior1=list(ncomp=1)
Mcmc1=list(R=R,keep=1)

set.seed(1)
out2 = parallel::mclapply(Data2, FUN = rhierLinearMixtureParallel,
Prior = Prior1, Mcmc = Mcmc1,
mc.cores = s, mc.set.seed = FALSE)

betadraws = parallel::mclapply(out2,FUN=drawPosteriorParallel,Z=Z,
Prior = Prior1, Mcmc = Mcmc1, mc.cores=s,mc.set.seed = FALSE)
betadraws = combine_draws(betadraws, R)

out_indep = parallel::mclapply(Data2, FUN=rheteroLinearIndepMetrop,
betadraws = betadraws, Mcmc = Mcmc1, Prior = Prior1, mc.cores = s, mc.set.seed = FALSE)


######### Multiple Components with rhierLinearMixtureParallel########
R = 500

set.seed(66)
nreg=1000
nobs=5 #number of observations
nvar=3 #columns
nz=2

Z=matrix(runif(nreg*nz),ncol=nz)
Z=t(t(Z)-apply(Z,2,mean))
```

```
Delta=matrix(c(1,-1,2,0,1,0),ncol=nz)
tau0=.1
iota=c(rep(1,nobs))

#Default
tcomps=NULL
a=matrix(c(1,0,0,0.5773503,1.1547005,0,-0.4082483,0.4082483,1.2247449),ncol=3)
tcomps[[1]]=list(mu=c(0,-1,-2),rooti=a)
tcomps[[2]]=list(mu=c(0,-1,-2)*2,rooti=a)
tcomps[[3]]=list(mu=c(0,-1,-2)*4,rooti=a)
tpvec=c(.4,.2,.4)

regdata=NULL
betas=matrix(double(nreg*nvar),ncol=nvar)
tind=double(nreg)

for (reg in 1:nreg) {
 tempout=bayesm::rmixture(1,tpvec,tcomps)
 if (is.null(Z)){
   betas[reg,]= as.vector(tempout$x)
 }else{
   betas[reg,]=Delta %*% Z[reg,]+as.vector(tempout$x)}
 tind[reg]=tempout$z
 X=cbind(iota,matrix(runif(nobs*(nvar-1)),ncol=(nvar-1)))
 tau=tau0*runif(1,min=0.5,max=1)
 y=X %*% betas[reg,]+sqrt(tau)*rnorm(nobs)
 regdata[[reg]]=list(y=y,X=X,beta=betas[reg,],tau=tau)
}


Data1=list(list(regdata=regdata,Z=Z))
s = 1
Data2=scalablebayesm::partition_data(Data1, s=s)

Prior1=list(ncomp=3)
Mcmc1=list(R=R,keep=1)

set.seed(1)
out2 = parallel::mclapply(Data2, FUN = rhierLinearMixtureParallel, Prior = Prior1, Mcmc = Mcmc1,
mc.cores = s, mc.set.seed = FALSE)

betadraws = parallel::mclapply(out2,FUN=drawPosteriorParallel,Z=Z,
Prior = Prior1, Mcmc = Mcmc1, mc.cores=s,mc.set.seed = TRUE)
betadraws = combine_draws(betadraws, R)

out_indep = parallel::mclapply(Data2, FUN=rheteroLinearIndepMetrop,
betadraws = betadraws, Mcmc = Mcmc1, Prior = Prior1, mc.cores = s, mc.set.seed = TRUE)
```

`rheteroMnlIndepMetrop` *Independence Metropolis-Hastings Algorithm for Draws From Multinomial Distribution*

Description

`rheteroMnlIndepMetrop` implements an Independence Metropolis algorithm with a Gibbs sampler.

Usage

```
rheteroMnlIndepMetrop(Data, draws, Mcmc)
```

Arguments

`Data`	A list of s partitions where each partition includes: 'p'- number of choice alternatives, 'lgtdata' - An $nlgt$ size list of multinomial logistic data, and optional 'Z'- matrix of unit characteristics.
`draws`	A list of draws returned from either `rhierMnlRwMixtureParallel`.
`Mcmc`	A list with one required parameter: 'R'-number of iterations, and optional parameters: 'keep' and 'nprint'.

Details

Model and Priors: $y_i \sim MNL(X_i, \beta_i)$ with $i = 1, \ldots,$ length(lgtdata) and where β_i is $1 \times nvar$
β_i = $Z\Delta$[i,] + u_i
Note: ZΔ is the matrix Z $\times \Delta$ and [i,] refers to ith row of this product
Delta is an $nz \times nvar$ array
$u_i \sim N(\mu_{ind}, \Sigma_{ind})$ with $ind \sim$ multinomial(pvec)
$pvec \sim$ dirichlet(a)
$delta = vec(\Delta) \sim N(deltabar, A_d^{-1})$
$\mu_j \sim N(mubar, \Sigma_j(x) Amu^{-1})$
$\Sigma_j \sim IW(nu, V)$
Note: Z should NOT include an intercept and is centered for ease of interpretation. The mean of each of the `nlgt` βs is the mean of the normal mixture. Use `summary()` to compute this mean from the `compdraw` output.

Be careful in assessing prior parameter `Amu`: 0.01 is too small for many applications. See chapter 5 of Rossi et al for full discussion.

Argument Details: `Data = list(lgtdata, Z, p)` *[`Z` optional]*

`lgtdata:`	A $nlgt/shards$ size list of multinominal lgtdata
`lgtdata[[i]]$y:`	$n_i \times 1$ vector of multinomial outcomes (1, ..., p) for ith unit
`lgtdata[[i]]$X:`	$(n_i \times p) \times nvar$ design matrix for ith unit
`Z:`	A list of s partitions where each partition include $(nlgt/numberofshards) \times nz$ matrix of unit charact

`p:` number of choice alternatives

`draws:` A matrix with R rows and $nlgt$ columns of beta draws.

`Mcmc = list(R, keep, nprint, s, w)` *[only* `R` *required]*

`R:`	number of MCMC draws
`keep:`	MCMC thinning parameter – keep every `keep`th draw (def: 1)
`nprint:`	print the estimated time remaining for every `nprint`'th draw (def: 100, set to 0 for no print)
`s:`	scaling parameter for RW Metropolis (def: 2.93/`sqrt(nvar)`)
`w:`	fractional likelihood weighting parameter (def: 0.1)

Value

A list containing:

- **betadraw**: A matrix of size $R \times nvar$ containing the drawn `beta` values from the Gibbs sampling procedure.

Author(s)

Federico Bumbaca, Leeds School of Business, University of Colorado Boulder, <federico.bumbaca@colorado.edu>

References

Bumbaca, F. (Rico), Misra, S., & Rossi, P. E. (2020). Scalable Target Marketing: Distributed Markov Chain Monte Carlo for Bayesian Hierarchical Models. Journal of Marketing Research, 57(6), 999-1018.

See Also

`rhierMnlRwMixtureParallel`, `rheteroLinearIndepMetrop`

Examples

```
R = 500

######### Single Component with rhierMnlRwMixtureParallel########
##parameters
p=3 # num of choice alterns
ncoef=3
nlgt=1000
nz=2
Z=matrix(runif(nz*nlgt),ncol=nz)
Z=t(t(Z)-apply(Z,2,mean)) # demean Z

ncomp=1 # no of mixture components
Delta=matrix(c(1,0,1,0,1,2),ncol=2)
comps=NULL
comps[[1]]=list(mu=c(0,2,1),rooti=diag(rep(1,3)))
pvec=c(1)
```

```
simmnlwX= function(n,X,beta){
  k=length(beta)
  Xbeta=X %*% beta
  j=nrow(Xbeta)/n
  Xbeta=matrix(Xbeta,byrow=TRUE,ncol=j)
  Prob=exp(Xbeta)
  iota=c(rep(1,j))
  denom=Prob %*% iota
  Prob=Prob/as.vector(denom)
  y=vector("double",n)
  ind=1:j
  for (i in 1:n) {
    yvec = rmultinom(1, 1, Prob[i,])
    y[i] = ind%*%yvec
  }
  return(list(y=y,X=X,beta=beta,prob=Prob))
}

## simulate data
simlgtdata=NULL
ni=rep(5,nlgt)
for (i in 1:nlgt)
{
  if (is.null(Z))
  {
    betai=as.vector(bayesm::rmixture(1,pvec,comps)$x)
  } else {
    betai=Delta %*% Z[i,]+as.vector(bayesm::rmixture(1,pvec,comps)$x)
  }
  Xa=matrix(runif(ni[i]*p,min=-1.5,max=0),ncol=p)
  X=bayesm::createX(p,na=1,nd=NULL,Xa=Xa,Xd=NULL,base=1)
  outa=simmnlwX(ni[i],X,betai)
  simlgtdata[[i]]=list(y=outa$y,X=X,beta=betai)
}

## set MCMC parameters
Prior1=list(ncomp=ncomp)
keep=1
Mcmc1=list(R=R,keep=keep)
Data1=list(list(p=p,lgtdata=simlgtdata,Z=Z))
s = 1
Data2 = partition_data(Data1, s=s)

out2 = parallel::mclapply(Data2, FUN = rhierMnlRwMixtureParallel, Prior = Prior1,
Mcmc = Mcmc1,mc.cores = s, mc.set.seed = FALSE)

betadraws = parallel::mclapply(out2,FUN=drawPosteriorParallel,Z=Z,
Prior = Prior1, Mcmc = Mcmc1, mc.cores=s,mc.set.seed = FALSE)
betadraws = combine_draws(betadraws, R)

out_indep = parallel::mclapply(Data2, FUN=rheteroMnlIndepMetrop, draws = betadraws,
Mcmc = Mcmc1, mc.cores = s, mc.set.seed = FALSE)
```

```
######### Multiple Components with rhierMnlRwMixtureParallel########
##parameters
R=500
p=3 # num of choice alterns
ncoef=3
nlgt=1000  # num of cross sectional units
nz=2
Z=matrix(runif(nz*nlgt),ncol=nz)
Z=t(t(Z)-apply(Z,2,mean)) # demean Z

ncomp=3
Delta=matrix(c(1,0,1,0,1,2),ncol=2)

comps=NULL
comps[[1]]=list(mu=c(0,2,1),rooti=diag(rep(1,3)))
comps[[2]]=list(mu=c(1,0,2),rooti=diag(rep(1,3)))
comps[[3]]=list(mu=c(2,1,0),rooti=diag(rep(1,3)))
pvec=c(.4,.2,.4)

simmnlwX= function(n,X,beta) {
  k=length(beta)
  Xbeta=X %*% beta
  j=nrow(Xbeta)/n
  Xbeta=matrix(Xbeta,byrow=TRUE,ncol=j)
  Prob=exp(Xbeta)
  iota=c(rep(1,j))
  denom=Prob %*% iota
  Prob=Prob/as.vector(denom)
  y=vector("double",n)
  ind=1:j
  for (i in 1:n)
  {yvec=rmultinom(1,1,Prob[i,]); y[i]=ind %*% yvec}
  return(list(y=y,X=X,beta=beta,prob=Prob))
}

## simulate data
simlgtdata=NULL
ni=rep(5,nlgt)
for (i in 1:nlgt)
{
  if (is.null(Z))
  {
    betai=as.vector(bayesm::rmixture(1,pvec,comps)$x)
  } else {
    betai=Delta %*% Z[i,]+as.vector(bayesm::rmixture(1,pvec,comps)$x)
  }
  Xa=matrix(runif(ni[i]*p,min=-1.5,max=0),ncol=p)
  X=bayesm::createX(p,na=1,nd=NULL,Xa=Xa,Xd=NULL,base=1)
  outa=simmnlwX(ni[i],X,betai)
  simlgtdata[[i]]=list(y=outa$y,X=X,beta=betai)
}
```

```
## set parameters for priors and Z
Prior1=list(ncomp=ncomp)
keep = 1
Mcmc1=list(R=R,keep=keep)
Data1=list(list(p=p,lgtdata=simlgtdata,Z=Z))
s = 1
Data2 = partition_data(Data1,s)

out2 = parallel::mclapply(Data2, FUN = rhierMnlRwMixtureParallel, Prior = Prior1,
Mcmc = Mcmc1, mc.cores = s, mc.set.seed = FALSE)

betadraws = parallel::mclapply(out2,FUN=drawPosteriorParallel,Z=Z,
Prior = Prior1, Mcmc = Mcmc1, mc.cores=s,mc.set.seed = FALSE)
betadraws = combine_draws(betadraws, R)

out_indep = parallel::mclapply(Data2, FUN=rheteroMnlIndepMetrop, draws = betadraws,
Mcmc = Mcmc1, mc.cores = s, mc.set.seed = FALSE)
```

`rhierLinearDPParallel` *MCMC Algorithm for Hierarchical Linear Model with Dirichlet Process Prior Heterogeneity*

Description

`rhierLinearDPParallel` is an MCMC algorithm for a hierarchical linear model with a Dirichlet Process prior for the distribution of heterogeneity. A base normal model is used so that the DP can be interpreted as allowing for a mixture of normals with as many components as panel units. The function implements a Gibbs sampler on the coefficients for each panel unit. This procedure can be interpreted as a Bayesian semi-parametric method since the DP prior accommodates heterogeneity of unknown form.

Usage

```
rhierLinearDPParallel(Data, Prior, Mcmc, verbose = FALSE)
```

Arguments

`Data`	A list of: 'regdata' - A $nreg$ size list of regdata, and optional 'Z'- $nreg \times nz$ matrix of unit characteristics.
`Prior`	A list with one required parameter: 'ncomp', and optional parameters: 'deltabar', 'Ad', 'mubar', 'Amu', 'nu', 'V', 'nu.e', and 'ssq'.
`Mcmc`	A list with one required parameter: 'R'-number of iterations, and optional parameters: 'keep' and 'nprint'.
`verbose`	If `TRUE`, enumerates model parameters and timing information.

Details

Model and Priors: `nreg` regression equations with `nvar` as the number of X vars in each equation
$y_i = X_i\beta_i + e_i$ with $e_i \sim N(0, \tau_i)$
$\tau_i \sim nu.e * ssq_i/\chi^2_{nu.e}$ where τ_i is the variance of e_i
$B = Z\Delta + U$ or $\beta_i = \Delta' Z[i,]' + u_i$
Δ is an $nz \times nvar$ matrix

$u_i \sim N(\mu_{ind}, \Sigma_{ind})$

$delta = vec(\Delta) \sim N(deltabar, A_d^{-1})$
$\mu_j \sim N(mubar, \Sigma_j(x)Amu^{-1})$
$\Sigma_j \sim IW(nu, V)$

$\theta_i = (\mu_i, \Sigma_i) \sim DP(G_0(\lambda), alpha)$

$G_0(\lambda)$:
$\mu_i|\Sigma_i \sim N(0, \Sigma_i(x)a^{-1})$
$\Sigma_i \sim IW(nu, nu * v * I)$
$delta = vec(\Delta) \sim N(deltabar, A_d^{-1})$

$\lambda(a, nu, v)$:
$a \sim$ uniform[alim[1], alimb[2]]
$nu \sim$ dim(data)-1 + exp(z)
$z \sim$ uniform[dim(data)-1+nulim[1], nulim[2]]
$v \sim$ uniform[vlim[1], vlim[2]]

Z should *not* include an intercept and should be centered for ease of interpretation. The mean of each of the `nreg` βs is the mean of the normal mixture. Use `summary()` to compute this mean from the `compdraw` output.

The prior on Σ_i is parameterized such that $mode(\Sigma) = nu/(nu + 2)vI$. The support of nu enforces a non-degenerate IW density; $nulim[1] > 0$

The default choices of alim, nulim, and vlim determine the location and approximate size of candidate "atoms" or possible normal components. The defaults are sensible given a reasonable scaling of the X variables. You want to insure that alim is set for a wide enough range of values (remember a is a precision parameter) and the v is big enough to propose Sigma matrices wide enough to cover the data range.

A careful analyst should look at the posterior distribution of a, nu, v to make sure that the support is set correctly in alim, nulim, vlim. In other words, if we see the posterior bunched up at one end of these support ranges, we should widen the range and rerun.

If you want to force the procedure to use many small atoms, then set nulim to consider only large values and set vlim to consider only small scaling constants. Set alphamax to a large number. This will create a very "lumpy" density estimate somewhat like the classical Kernel density estimates. Of course, this is not advised if you have a prior belief that densities are relatively smooth.

Argument Details: `Data = list(regdata, Z)` *[`Z` optional]*

`regdata:`	A $nreg/s_shard$ size list of regdata
`regdata[[i]]$X:`	$n_i \times nvar$ design matrix for equation i
`regdata[[i]]$y:`	$n_i \times 1$ vector of observations for equation i
`Z:`	A list of s partitions where each partition include $(nreg/s_shard) \times nz$ matrix of unit characteristics

`Prior = list(deltabar, Ad, Prioralphalist, lambda_hyper, nu, V, nu_e, mubar, Amu, ssq, ncomp)` *[all but* `ncomp` *are optional]*

`deltabar:`	$(nz \times nvar) \times 1$ vector of prior means (def: 0)
`Ad:`	prior precision matrix for vec(D) (def: 0.01*I)
`mubar:`	$nvar \times 1$ prior mean vector for normal component mean (def: 0)
`Amu:`	prior precision for normal component mean (def: 0.01)
`nu.e:`	d.f. parameter for regression error variance prior (def: 3)
`V:`	PDS location parameter for IW prior on normal component Sigma (def: nu*I)
`ssq:`	scale parameter for regression error variance prior (def: `var(y_i)`)
`ncomp:`	number of components used in normal mixture

`Mcmc = list(R, keep, nprint)` *[only* `R` *required]*

`R:`	number of MCMC draws
`keep:`	MCMC thinning parameter – keep every `keep`th draw (def: 1)
`nprint:`	print the estimated time remaining for every `nprint`'th draw (def: 100, set to 0 for no print)

Value

A list containing:

`compdraw`	A list (length: R/keep) where each list contains 'mu' (vector, length: 'ncomp') and 'rooti' (matrix, shape: ncomp $\times$ ncomp)
`probdraw`	A $(R/keep) \times (ncomp)$ matrix that reports the probability that each draw came from a particular component
`Deltadraw`	A $(R/keep) \times (nz \times nvar)$ matrix of draws of Delta, first row is initial value. Deltadraw is NULL if Z is NULL

Author(s)

Federico Bumbaca, Leeds School of Business, University of Colorado Boulder, <federico.bumbaca@colorado.edu>

References

Bumbaca, Federico (Rico), Sanjog Misra, and Peter E. Rossi (2020), "Scalable Target Marketing: Distributed Markov Chain Monte Carlo for Bayesian Hierarchical Models", Journal of Marketing Research, 57(6), 999-1018.

Chapter 5, *Bayesian Statistics and Marketing* by Rossi, Allenby, and McCulloch.

See Also

partition_data, drawPosteriorParallel, combine_draws, rheteroLinearIndepMetrop

Examples

```
######### Single Component with rhierLinearDPParallel########
R = 1000

nreg = 2000
nobs = 5 #number of observations
nvar = 3 #columns
nz = 2

Z=matrix(runif(nreg*nz),ncol=nz)
Z=t(t(Z)-apply(Z,2,mean))

Delta=matrix(c(1,-1,2,0,1,0),ncol=nz)
tau0=.1
iota=c(rep(1,nobs))

#Default
tcomps=NULL
a=matrix(c(1,0,0,0.5773503,1.1547005,0,-0.4082483,0.4082483,1.2247449),ncol=3)
tcomps[[1]]=list(mu=c(0,-1,-2),rooti=a)
tpvec=c(1)

regdata=NULL
betas=matrix(double(nreg*nvar),ncol=nvar)
tind=double(nreg)

for (reg in 1:nreg) {
 tempout=bayesm::rmixture(1,tpvec,tcomps)
 if (is.null(Z)){
   betas[reg,]= as.vector(tempout$x)
 }else{
   betas[reg,]=Delta %*% Z[reg,]+as.vector(tempout$x)}
 tind[reg]=tempout$z
 X=cbind(iota,matrix(runif(nobs*(nvar-1)),ncol=(nvar-1)))
 tau=tau0*runif(1,min=0.5,max=1)
 y=X %*% betas[reg,]+sqrt(tau)*rnorm(nobs)
 regdata[[reg]]=list(y=y,X=X,beta=betas[reg,],tau=tau)
}


Data1=list(list(regdata=regdata,Z=Z))
s = 1
Data2=scalablebayesm::partition_data(Data1,s=s)

Prior1=list(ncomp=1)
Mcmc1=list(R=R,keep=1)
```

```
out2 = parallel::mclapply(Data2, FUN = rhierLinearDPParallel, Prior = Prior1,
Mcmc = Mcmc1, mc.cores = s, mc.set.seed = FALSE)

######### Multiple Components with rhierLinearDPParallel########
R = 1000

set.seed(66)
nreg=2000
nobs=5 #number of observations
nvar=3 #columns
nz=2

Z=matrix(runif(nreg*nz),ncol=nz)
Z=t(t(Z)-apply(Z,2,mean))

Delta=matrix(c(1,-1,2,0,1,0),ncol=nz)
tau0=.1
iota=c(rep(1,nobs))

#Default
tcomps=NULL
a=matrix(c(1,0,0,0.5773503,1.1547005,0,-0.4082483,0.4082483,1.2247449),ncol=3)
tcomps[[1]]=list(mu=c(0,-1,-2),rooti=a)
tcomps[[2]]=list(mu=c(0,-1,-2)*2,rooti=a)
tcomps[[3]]=list(mu=c(0,-1,-2)*4,rooti=a)
tpvec=c(.4,.2,.4)

regdata=NULL
betas=matrix(double(nreg*nvar),ncol=nvar)
tind=double(nreg)

for (reg in 1:nreg) {
 tempout=bayesm::rmixture(1,tpvec,tcomps)
 if (is.null(Z)){
   betas[reg,]= as.vector(tempout$x)
 }else{
   betas[reg,]=Delta %*% Z[reg,]+as.vector(tempout$x)}
 tind[reg]=tempout$z
 X=cbind(iota,matrix(runif(nobs*(nvar-1)),ncol=(nvar-1)))
 tau=tau0*runif(1,min=0.5,max=1)
 y=X %*% betas[reg,]+sqrt(tau)*rnorm(nobs)
 regdata[[reg]]=list(y=y,X=X,beta=betas[reg,],tau=tau)
}


Data1=list(list(regdata=regdata,Z=Z))
s = 1
Data2=scalablebayesm::partition_data(Data1, s=s)

Prior1=list(ncomp=3)
Mcmc1=list(R=R,keep=1)

out2 = parallel::mclapply(Data2, FUN = rhierLinearDPParallel, Prior = Prior1,
```

```
Mcmc = Mcmc1, mc.cores = s, mc.set.seed = FALSE)
```

`rhierLinearMixtureParallel` *MCMC Algorithm for Hierarchical Multinomial Linear Model with Mixture-of-Normals Heterogeneity*

Description

`rhierLinearMixtureParallel` implements a MCMC algorithm for hierarchical linear model with a mixture of normals heterogeneity distribution.

Usage

```
rhierLinearMixtureParallel(Data, Prior, Mcmc, verbose = FALSE)
```

Arguments

`Data`	A list containing: 'regdata' - A $nreg$ size list of multinomial regdata, and optional 'Z'- $nreg \times nz$ matrix of unit characteristics.
`Prior`	A list with one required parameter: 'ncomp', and optional parameters: 'deltabar', 'Ad', 'mubar', 'Amu', 'nu', 'V', 'nu.e', and 'ssq'.
`Mcmc`	A list with one required parameter: 'R'-number of iterations, and optional parameters: 'keep' and 'nprint'.
`verbose`	If `TRUE`, enumerates model parameters and timing information.

Details

Model and Priors: `nreg` regression equations with `nvar` as the number of X vars in each equation
$y_i = X_i\beta_i + e_i$ with $e_i \sim N(0, \tau_i)$
$\tau_i \sim nu.e * ssq_i/\chi^2_{nu.e}$ where τ_i is the variance of e_i
$B = Z\Delta + U$ or $\beta_i = \Delta' Z[i,]' + u_i$
Δ is an $nz \times nvar$ matrix

Z should *not* include an intercept and should be centered for ease of interpretation. The mean of each of the `nreg` βs is the mean of the normal mixture. Use `summary()` to compute this mean from the `compdraw` output.
$u_i \sim N(\mu_{ind}, \Sigma_{ind})$
$ind \sim multinomial(pvec)$

$pvec \sim dirichlet(a)$
$delta = vec(\Delta) \sim N(deltabar, A_d^{-1})$
$\mu_j \sim N(mubar, \Sigma_j(x) Amu^{-1})$

$\Sigma_j \sim IW(nu, V)$

Be careful in assessing the prior parameter Amu: 0.01 can be too small for some applications. See chapter 5 of Rossi et al for full discussion.

Argument Details: Data = list(regdata, Z) *[Z optional]*

regdata:	A $nreg$ size list of regdata
regdata[[i]]$X:	$n_i \times nvar$ design matrix for equation i
regdata[[i]]$y:	$n_i \times 1$ vector of observations for equation i
Z:	An $(nreg) \times nz$ matrix of unit characteristics

Prior = list(deltabar, Ad, mubar, Amu, nu.e, V, ssq, ncomp) *[all but* ncomp *are optional]*

deltabar:	$(nz \times nvar) \times 1$ vector of prior means (def: 0)
Ad:	prior precision matrix for vec(D) (def: 0.01*I)
mubar:	$nvar \times 1$ prior mean vector for normal component mean (def: 0)
Amu:	prior precision for normal component mean (def: 0.01)
nu.e:	d.f. parameter for regression error variance prior (def: 3)
V:	PDS location parameter for IW prior on normal component Sigma (def: nu*I)
ssq:	scale parameter for regression error variance prior (def: var(y_i))
ncomp:	number of components used in normal mixture

Mcmc = list(R, keep, nprint) *[only* R *required]*

R:	number of MCMC draws
keep:	MCMC thinning parameter – keep every keepth draw (def: 1)
nprint:	print the estimated time remaining for every nprint'th draw (def: 100, set to 0 for no print)

Value

A list of sharded partitions where each partition contains the following:

compdraw	A list (length: R/keep) where each list contains 'mu' (vector, length: 'ncomp') and 'rooti' (matrix, shape: ncomp $\times$ ncomp)
probdraw	A $(R/keep) \times (ncomp)$ matrix that reports the probability that each draw came from a particular component
Deltadraw	A $(R/keep) \times (nz \times nvar)$ matrix of draws of Delta, first row is initial value. Deltadraw is NULL if Z is NULL

Author(s)

Federico Bumbaca, Leeds School of Business, University of Colorado Boulder, <federico.bumbaca@colorado.edu>

References

Bumbaca, Federico (Rico), Sanjog Misra, and Peter E. Rossi (2020), "Scalable Target Marketing: Distributed Markov Chain Monte Carlo for Bayesian Hierarchical Models", Journal of Marketing Research, 57(6), 999-1018.

Chapter 5, *Bayesian Statistics and Marketing* by Rossi, Allenby, and McCulloch.

See Also

partition_data, drawPosteriorParallel, combine_draws, rheteroLinearIndepMetrop

Examples

```
######### Single Component with rhierLinearMixtureParallel########
R = 500

nreg=1000
nobs=5 #number of observations
nvar=3 #columns
nz=2

Z=matrix(runif(nreg*nz),ncol=nz)
Z=t(t(Z)-apply(Z,2,mean))

Delta=matrix(c(1,-1,2,0,1,0),ncol=nz)
tau0=.1
iota=c(rep(1,nobs))

#Default
tcomps=NULL
a=matrix(c(1,0,0,0.5773503,1.1547005,0,-0.4082483,0.4082483,1.2247449),ncol=3)
tcomps[[1]]=list(mu=c(0,-1,-2),rooti=a)
tpvec=c(1)

regdata=NULL
betas=matrix(double(nreg*nvar),ncol=nvar)
tind=double(nreg)

for (reg in 1:nreg) {
 tempout=bayesm::rmixture(1,tpvec,tcomps)
 if (is.null(Z)){
   betas[reg,]= as.vector(tempout$x)
 }else{
   betas[reg,]=Delta %*% Z[reg,]+as.vector(tempout$x)}
 tind[reg]=tempout$z
 X=cbind(iota,matrix(runif(nobs*(nvar-1)),ncol=(nvar-1)))
 tau=tau0*runif(1,min=0.5,max=1)
 y=X %*% betas[reg,]+sqrt(tau)*rnorm(nobs)
 regdata[[reg]]=list(y=y,X=X,beta=betas[reg,],tau=tau)
}
```

```
Data1=list(list(regdata=regdata,Z=Z))
s = 1
Data2=scalablebayesm::partition_data(Data1,s=s)

Prior1=list(ncomp=1)
Mcmc1=list(R=R,keep=1)

out2 = parallel::mclapply(Data2, FUN = rhierLinearMixtureParallel, Prior = Prior1,
Mcmc = Mcmc1, mc.cores = s, mc.set.seed = FALSE)

######### Multiple Components with rhierLinearMixtureParallel########
R = 500

set.seed(66)
nreg=1000
nobs=5 #number of observations
nvar=3 #columns
nz=2

Z=matrix(runif(nreg*nz),ncol=nz)
Z=t(t(Z)-apply(Z,2,mean))

Delta=matrix(c(1,-1,2,0,1,0),ncol=nz)
tau0=.1
iota=c(rep(1,nobs))

#Default
tcomps=NULL
a=matrix(c(1,0,0,0.5773503,1.1547005,0,-0.4082483,0.4082483,1.2247449),ncol=3)
tcomps[[1]]=list(mu=c(0,-1,-2),rooti=a)
tcomps[[2]]=list(mu=c(0,-1,-2)*2,rooti=a)
tcomps[[3]]=list(mu=c(0,-1,-2)*4,rooti=a)
tpvec=c(.4,.2,.4)

regdata=NULL
betas=matrix(double(nreg*nvar),ncol=nvar)
tind=double(nreg)

for (reg in 1:nreg) {
 tempout=bayesm::rmixture(1,tpvec,tcomps)
 if (is.null(Z)){
   betas[reg,]= as.vector(tempout$x)
 }else{
   betas[reg,]=Delta %*% Z[reg,]+as.vector(tempout$x)}
 tind[reg]=tempout$z
 X=cbind(iota,matrix(runif(nobs*(nvar-1)),ncol=(nvar-1)))
 tau=tau0*runif(1,min=0.5,max=1)
 y=X %*% betas[reg,]+sqrt(tau)*rnorm(nobs)
 regdata[[reg]]=list(y=y,X=X,beta=betas[reg,],tau=tau)
}
```

```
Data1=list(list(regdata=regdata,Z=Z))
s = 1
Data2=scalablebayesm::partition_data(Data1, s=s)

Prior1=list(ncomp=3)
Mcmc1=list(R=R,keep=1)

set.seed(1)
out2 = parallel::mclapply(Data2, FUN = rhierLinearMixtureParallel, Prior = Prior1,
Mcmc = Mcmc1, mc.cores = s, mc.set.seed = FALSE)
```

`rhierMnlDPParallel` *MCMC Algorithm for Hierarchical Multinomial Logit with Dirichlet Process Prior Heterogeneity*

Description

`rhierMnlDPParallel` is an MCMC algorithm for a hierarchical multinomial logit with a Dirichlet Process prior describing the distribution of heteorogeneity. A base normal model is used so that the DP can be interpreted as allowing for a mixture of normals with as many components as panel units. This is a hybrid Gibbs Sampler with a RW Metropolis step for the MNL coefficients for each panel unit. This procedure can be interpreted as a Bayesian semi-parametric method in the sense that the DP prior can accommodate heterogeneity of an unknown form.

Usage

```
rhierMnlDPParallel(Data, Prior, Mcmc, verbose = FALSE)
```

Arguments

`Data`	A list of: 'p'- number of choice alternatives, 'lgtdata' - An $nlgt$ size list of multinomial logistic data, and optional 'Z'- matrix of unit characteristics.
`Prior`	A list with one required parameter: 'ncomp', and optional parameters: 'mubar', 'Amu', 'nu', 'V', 'Ad', 'deltaBar', and 'a'.
`Mcmc`	A list with one required parameter: 'R' - number of iterations, and optional parameters: 's', 'w', 'keep', 'nprint', and 'drawcomp'.
`verbose`	If `TRUE`, enumerates model parameters and timing information.

Details

Model and Priors: $y_i \sim MNL(X_i, \beta_i)$ with $i = 1, \ldots, length(lgtdata)$ and where θ_i is $nvar x 1$
$\beta_i = Z\Delta$[i,] + u_i
Note: ZΔ is the matrix $Z * \Delta$; [i,] refers to ith row of this product
Delta is an $nz x nvar$ matrix
$\beta_i \sim N(\mu_i, \Sigma_i)$

$\theta_i = (\mu_i, \Sigma_i) \sim DP(G_0(\lambda), alpha)$

$G_0(\lambda)$:
$\mu_i|\Sigma_i \sim N(0, \Sigma_i(x)a^{-1})$
$\Sigma_i \sim IW(nu, nu * v * I)$
$delta = vec(\Delta) \sim N(deltabar, A_d^{-1})$

$\lambda(a, nu, v)$:
$a \sim$ uniform[alim[1], alimb[2]]
$nu \sim$ dim(data)-1 + exp(z)
$z \sim$ uniform[dim(data)-1+nulim[1], nulim[2]]
$v \sim$ uniform[vlim[1], vlim[2]]
$alpha \sim (1 - (alpha - alphamin)/(alphamax - alphamin))^{power}$
alpha = alphamin then expected number of components = `Istarmin`
alpha = alphamax then expected number of components = `Istarmax`

Z should NOT include an intercept and is centered for ease of interpretation. The mean of each of the `nlgt` βs is the mean of the normal mixture. Use `summary()` to compute this mean from the `compdraw` output.

We parameterize the prior on Σ_i such that $mode(\Sigma) = nu/(nu + 2)vI$. The support of nu enforces a non-degenerate IW density; $nulim[1] > 0$.

The default choices of alim, nulim, and vlim determine the location and approximate size of candidate "atoms" or possible normal components. The defaults are sensible given a reasonable scaling of the X variables. You want to insure that alim is set for a wide enough range of values (remember a is a precision parameter) and the v is big enough to propose Sigma matrices wide enough to cover the data range.

A careful analyst should look at the posterior distribution of a, nu, v to make sure that the support is set correctly in alim, nulim, vlim. In other words, if we see the posterior bunched up at one end of these support ranges, we should widen the range and rerun.

If you want to force the procedure to use many small atoms, then set nulim to consider only large values and set vlim to consider only small scaling constants. Set alphamax to a large number. This will create a very "lumpy" density estimate somewhat like the classical Kernel density estimates. Of course, this is not advised if you have a prior belief that densities are relatively smooth.

Value

A list containing the following elements:

- **nmix**: A list with the following components:
  - **probdraw**: A matrix of size `(R/keep)` x `(ncomp`, containing the probability draws at each Gibbs iteration.
  - **compdraw**: A list containing the drawn mixture components at each Gibbs iteration.
- **Deltadraw** (optional): A matrix of size `(R/keep)` x `(nz * nvar)`, containing the delta draws, if `Z` is not `NULL`. If `Z` is `NULL`, this element is not included.

Author(s)

Federico Bumbaca, Leeds School of Business, University of Colorado Boulder, <federico.bumbaca@colorado.edu>

References

Bumbaca, F. (Rico), Misra, S., & Rossi, P. E. (2020). Scalable Target Marketing: Distributed Markov Chain Monte Carlo for Bayesian Hierarchical Models. Journal of Marketing Research, 57(6), 999-1018.

Examples

```
set.seed(66)
R = 500

p = 3                                # num of choice alterns
ncoef = 3
nlgt = 1000                           # num of cross sectional units
nz = 2
nvar = 3
Z = matrix(runif(nz*nlgt), ncol=nz)
Z = t(t(Z)-apply(Z,2,mean))          # demean Z
ncomp = 3                            # no of mixture components
Delta=matrix(c(1,0,1,0,1,2), ncol=2)

comps = NULL
comps[[1]] = list(mu=c(0,-1,-2),   rooti=diag(rep(2,3)))
comps[[2]] = list(mu=c(0,-1,-2)*2, rooti=diag(rep(2,3)))
comps[[3]] = list(mu=c(0,-1,-2)*4, rooti=diag(rep(2,3)))
pvec=c(0.4, 0.2, 0.4)

##  simulate from MNL model conditional on X matrix
simmnlwX = function(n,X,beta) {
  k = length(beta)
  Xbeta = X%*%beta
  j = nrow(Xbeta) / n
  Xbeta = matrix(Xbeta, byrow=TRUE, ncol=j)
  Prob = exp(Xbeta)
  iota = c(rep(1,j))
  denom = Prob%*%iota
  Prob = Prob / as.vector(denom)
  y = vector("double", n)
  ind = 1:j
  for (i in 1:n) {
    yvec = rmultinom(1, 1, Prob[i,])
    y[i] = ind%*%yvec}
  return(list(y=y, X=X, beta=beta, prob=Prob))
}

## simulate data with a mixture of 3 normals
simlgtdata = NULL
ni = rep(50,nlgt)
for (i in 1:nlgt) {
  betai = Delta%*%Z[i,] + as.vector(bayesm::rmixture(1,pvec,comps)$x)
  Xa = matrix(runif(ni[i]*p,min=-1.5,max=0), ncol=p)
  X = bayesm::createX(p, na=1, nd=NULL, Xa=Xa, Xd=NULL, base=1)
```

```
  outa = simmnlwX(ni[i], X, betai)
  simlgtdata[[i]] = list(y=outa$y, X=X, beta=betai)
}

## plot betas

  old.par = par(no.readonly = TRUE)
  bmat = matrix(0, nlgt, ncoef)
  for(i in 1:nlgt) { bmat[i,] = simlgtdata[[i]]$beta }
  par(mfrow = c(ncoef,1))
  for(i in 1:ncoef) { hist(bmat[,i], breaks=30, col="magenta") }
  par(old.par)

## set Data and Mcmc lists
keep = 5
Mcmc1 = list(R=R, keep=keep)
Prior1=list(ncomp=1)
Data1 = list(p=p, lgtdata=simlgtdata, Z=Z)
Data2 = partition_data(Data = list(Data1), s = 1)

out = parallel::mclapply(Data2, FUN = rhierMnlDPParallel,
Prior = Prior1, Mcmc = Mcmc1, mc.cores = 1, mc.set.seed = FALSE)
```

`rhierMnlRwMixtureParallel` *MCMC Algorithm for Hierarchical Multinomial Logit with Mixture-of-Normals Heterogeneity*

Description

`rhierMnlRwMixtureParallel` implements a MCMC algorithm for a hierarchical multinomial logit model with a mixture of normals heterogeneity distribution.

Usage

```
rhierMnlRwMixtureParallel(Data, Prior, Mcmc, verbose = FALSE)
```

Arguments

`Data`	A list containing: 'p'- number of choice alternatives, 'lgtdata' - A $nlgt$ size list of multinomial logistic data, and optional 'Z'- matrix of unit characteristics.
`Prior`	A list with one required parameter: 'ncomp', and optional parameters: 'mubar', 'Amu', 'nu', 'V', 'Ad', 'deltaBar', and 'a'.
`Mcmc`	A list with one required parameter: 'R' - number of iterations, and optional parameters: 's', 'w', 'keep', 'nprint', and 'drawcomp'.
`verbose`	If `TRUE`, enumerates model parameters and timing information.

Details

Model and Priors: $y_i \sim MNL(X_i, \beta_i)$ with $i = 1, \ldots,$ length(lgtdata) and where β_i is $1 \times nvar$
$\beta_i = Z\Delta[i,] + u_i$
Note: $Z\Delta$ is the matrix Z $\times \Delta$ and [i,] refers to ith row of this product
Delta is an $nz \times nvar$ array
$u_i \sim N(\mu_{ind}, \Sigma_{ind})$ with $ind \sim$ multinomial(pvec)
$pvec \sim$ dirichlet(a)
$delta = vec(\Delta) \sim N(deltabar, A_d^{-1})$
$\mu_j \sim N(mubar, \Sigma_j(x) Amu^{-1})$
$\Sigma_j \sim IW(nu, V)$
Note: Z should NOT include an intercept and is centered for ease of interpretation. The mean of each of the `nlgt` βs is the mean of the normal mixture. Use `summary()` to compute this mean from the `compdraw` output.

Be careful in assessing prior parameter `Amu`: 0.01 is too small for many applications. See chapter 5 of Rossi et al for full discussion.

Argument Details: `Data = list(lgtdata, Z, p)` *[`Z` optional]*

`lgtdata:`	A $nlgt$ size list of multinominal lgtdata
`lgtdata[[i]]$y:`	$n_i \times 1$ vector of multinomial outcomes (1, ..., p) for ith unit
`lgtdata[[i]]$X:`	$(n_i \times p) \times nvar$ design matrix for ith unit
`Z:`	An $(nlgt) \times nz$ matrix of unit characteristics
`p:`	number of choice alternatives

`Prior = list(a, deltabar, Ad, mubar, Amu, nu, V, a, ncomp)` *[all but* `ncomp` *are optional]*

`a:`	$ncomp \times 1$ vector of Dirichlet prior parameters (def: `rep(5,ncomp)`)
`deltabar:`	$(nz \times nvar) \times 1$ vector of prior means (def: 0)
`Ad:`	prior precision matrix for vec(D) (def: 0.01*I)
`mubar:`	$nvar \times 1$ prior mean vector for normal component mean (def: 0 if unrestricted; 2 if restricted)
`Amu:`	prior precision for normal component mean (def: 0.01 if unrestricted; 0.1 if restricted)
`nu:`	d.f. parameter for IW prior on normal component Sigma (def: nvar+3 if unrestricted; nvar+15 if restricted)
`V:`	PDS location parameter for IW prior on normal component Sigma (def: nu*I if unrestricted; nu*D if restricted
`ncomp:`	number of components used in normal mixture

`Mcmc = list(R, keep, nprint, s, w)` *[only* `R` *required]*

`R:`	number of MCMC draws
`keep:`	MCMC thinning parameter – keep every `keep`th draw (def: 1)
`nprint:`	print the estimated time remaining for every `nprint`'th draw (def: 100, set to 0 for no print)
`s:`	scaling parameter for RW Metropolis (def: 2.93/`sqrt(nvar)`)
`w:`	fractional likelihood weighting parameter (def: 0.1)

Value

A list of sharded partitions where each partition contains the following:

compdraw	A list (length: R/keep) where each list contains ‘mu‘ (vector, length: ‘ncomp‘) and ‘rooti‘ (matrix, shape: ncomp $\times$ ncomp)
probdraw	A $(R/keep) \times (ncomp)$ matrix that reports the probability that each draw came from a particular component
Deltadraw	A $(R/keep) \times (nz \times nvar)$ matrix of draws of Delta, first row is initial value. This could be null if Z is NULL

Author(s)

Federico Bumbaca, Leeds School of Business, University of Colorado Boulder, <federico.bumbaca@colorado.edu>

References

Bumbaca, Federico (Rico), Sanjog Misra, and Peter E. Rossi (2020), "Scalable Target Marketing: Distributed Markov Chain Monte Carlo for Bayesian Hierarchical Models", Journal of Marketing Research, 57(6), 999-1018.

Chapter 5, *Bayesian Statistics and Marketing* by Rossi, Allenby, and McCulloch.

See Also

partition_data, drawPosteriorParallel, combine_draws, rheteroMnlIndepMetrop

Examples

```
R=500

######### Single Component with rhierMnlRwMixtureParallel########
##parameters
p = 3 # num of choice alterns
ncoef = 3
nlgt = 1000
nz = 2
Z = matrix(runif(nz*nlgt),ncol=nz)
Z = t(t(Z)-apply(Z,2,mean)) # demean Z

ncomp = 1 # no of mixture components
Delta = matrix(c(1,0,1,0,1,2),ncol=2)
comps = NULL
comps[[1]] = list(mu=c(0,2,1),rooti=diag(rep(1,3)))
pvec = c(1)

simmnlwX = function(n,X,beta){
  k=length(beta)
  Xbeta=X %*% beta
  j=nrow(Xbeta)/n
  Xbeta=matrix(Xbeta,byrow=TRUE,ncol=j)
  Prob=exp(Xbeta)
  iota=c(rep(1,j))
  denom=Prob %*% iota
```

```
  Prob=Prob/as.vector(denom)
  y=vector("double",n)
  ind=1:j
  for (i in 1:n) {
    yvec = rmultinom(1, 1, Prob[i,])
    y[i] = ind%*%yvec
  }
  return(list(y=y,X=X,beta=beta,prob=Prob))
}

## simulate data
simlgtdata=NULL
ni=rep(5,nlgt)
for (i in 1:nlgt)
{
  if (is.null(Z))
  {
    betai=as.vector(bayesm::rmixture(1,pvec,comps)$x)
  } else {
    betai=Delta %*% Z[i,]+as.vector(bayesm::rmixture(1,pvec,comps)$x)
  }
  Xa=matrix(runif(ni[i]*p,min=-1.5,max=0),ncol=p)
  X=bayesm::createX(p,na=1,nd=NULL,Xa=Xa,Xd=NULL,base=1)
  outa=simmnlwX(ni[i],X,betai)
  simlgtdata[[i]]=list(y=outa$y,X=X,beta=betai)
}

## set MCMC parameters
Prior1=list(ncomp=ncomp)
keep=1
Mcmc1=list(R=R,keep=keep)
Data1=list(list(p=p,lgtdata=simlgtdata,Z=Z))
s = 1
Data2 = partition_data(Data1, s=s)

out2 = parallel::mclapply(Data2, FUN = rhierMnlRwMixtureParallel, Prior = Prior1,
Mcmc = Mcmc1, mc.cores = s, mc.set.seed = FALSE)

######### Multiple Components with rhierMnlRwMixtureParallel########
##parameters
R = 500
p=3 # num of choice alterns
ncoef=3
nlgt=1000  # num of cross sectional units
nz=2
Z=matrix(runif(nz*nlgt),ncol=nz)
Z=t(t(Z)-apply(Z,2,mean)) # demean Z

ncomp=3
Delta=matrix(c(1,0,1,0,1,2),ncol=2)

comps=NULL
comps[[1]]=list(mu=c(0,2,1),rooti=diag(rep(1,3)))
```

```
comps[[2]]=list(mu=c(1,0,2),rooti=diag(rep(1,3)))
comps[[3]]=list(mu=c(2,1,0),rooti=diag(rep(1,3)))
pvec=c(.4,.2,.4)

simmnlwX= function(n,X,beta) {
  k=length(beta)
  Xbeta=X %*% beta
  j=nrow(Xbeta)/n
  Xbeta=matrix(Xbeta,byrow=TRUE,ncol=j)
  Prob=exp(Xbeta)
  iota=c(rep(1,j))
  denom=Prob %*% iota
  Prob=Prob/as.vector(denom)
  y=vector("double",n)
  ind=1:j
  for (i in 1:n)
  {yvec=rmultinom(1,1,Prob[i,]); y[i]=ind %*% yvec}
  return(list(y=y,X=X,beta=beta,prob=Prob))
}

## simulate data
simlgtdata=NULL
ni=rep(5,nlgt)
for (i in 1:nlgt)
{
  if (is.null(Z))
  {
    betai=as.vector(bayesm::rmixture(1,pvec,comps)$x)
  } else {
    betai=Delta %*% Z[i,]+as.vector(bayesm::rmixture(1,pvec,comps)$x)
  }
  Xa=matrix(runif(ni[i]*p,min=-1.5,max=0),ncol=p)
  X=bayesm::createX(p,na=1,nd=NULL,Xa=Xa,Xd=NULL,base=1)
  outa=simmnlwX(ni[i],X,betai)
  simlgtdata[[i]]=list(y=outa$y,X=X,beta=betai)
}

## set parameters for priors and Z
Prior1=list(ncomp=ncomp)
keep=1
Mcmc1=list(R=R,keep=keep)
Data1=list(list(p=p,lgtdata=simlgtdata,Z=Z))
s = 1
Data2 = partition_data(Data1,s)

out2 = parallel::mclapply(Data2, FUN = rhierMnlRwMixtureParallel, Prior = Prior1,
Mcmc = Mcmc1, mc.cores = s, mc.set.seed = FALSE)
```

sample_data *Sample Data*

Description

A function to subset data for use in distributed hierarchical bayesian algorithm for scalable target marketing.

Usage

```
sample_data(Data, Rate = 1)
```

Arguments

Data	(list) - A list of lists where each sublist contains either 'regdata' or 'lgtdata'.
Rate	(numeric) - Proportion of the data to be sampled

Value

Returns a list of the same structure as Data, but with length scaled by Rate.

Author(s)

Federico Bumbaca, <federico.bumbaca@colorado.edu>

Examples

```
# Generate hierarchical linear data
R=1000
nreg=10000
nobs=5 #number of observations
nvar=3 #columns
nz=2

Z=matrix(runif(nreg*nz),ncol=nz)
Z=t(t(Z)-apply(Z,2,mean))
Delta=matrix(c(1,-1,2,0,1,0), ncol = nz)
tau0=.1
iota=c(rep(1,nobs))

## create arguments for rmixture
tcomps=NULL
a = diag(1, nrow=3)
tcomps[[1]] = list(mu=c(-5,0,0),rooti=a)
tcomps[[2]] = list(mu=c(5, -5, 2),rooti=a)
tcomps[[3]] = list(mu=c(5,5,-2),rooti=a)
tpvec = c(.33,.33,.34)
ncomp=length(tcomps)
regdata=NULL
```

```
betas=matrix(double(nreg*nvar),ncol=nvar)
tind=double(nreg)
for (reg in 1:nreg) {
  tempout=bayesm::rmixture(1,tpvec,tcomps)
  if (is.null(Z)){
    betas[reg,]= as.vector(tempout$x)
  }else{
    betas[reg,]=Delta%*%Z[reg,]+as.vector(tempout$x)}
  tind[reg]=tempout$z
  X=cbind(iota,matrix(runif(nobs*(nvar-1)),ncol=(nvar-1)))
  tau=tau0*runif(1,min=0.5,max=1)
  y=X%*%betas[reg,]+sqrt(tau)*rnorm(nobs)
  regdata[[reg]]=list(y=y,X=X,beta=betas[reg,],tau=tau)
}

Prior1=list(ncomp=ncomp)
keep=1
Mcmc1=list(R=R,keep=keep)
Data1=list(list(regdata=regdata,Z=Z))

length(Data1[[1]]$regdata)

data_s = sample_data(Data = Data1, Rate = 0.1)
length(data_s[[1]]$regdata)
```

---

s_max *Calculate Maximum Number of Shards*

Description

A function to calculate the maximum number of shards to be used in distributed hierarchical Bayesian algorithm for scalable target marketing.

Usage

```
s_max(
  R,
  N,
  Data,
  s = 3,
  ep_squaremax = 0.001,
  ncomp = 1,
  Bpercent = 0.5,
  iterations = 10,
  keep = 1,
  npoints = 1000
)
```

Arguments

R	(integer) - Number of MCMC draws.
N	(integer) - The number of observational units in the full dataset.
Data	(list) - A list of lists where each sublist contains either 'regdata' or 'lgtdata'.
s	(integer) - A small number of shards used to evaluate the distributed algorithm.
ep_squaremax	(numeric) - A value indicating the user's maximum expected error tolerance.
ncomp	(integer) - The number of components in the mixture.
Bpercent	(numeric) - A decimal value representing the proportion of draws to burn-in
iterations	(numeric) - The number of times to estimate the maximum number of shards
keep	(numeric) - MCMC thinning parameter – keep every keepth draw (default: 1)
npoints	(integer) - The number of points at which to evaluate the difference in posterior distributions

Value

The function returns a list of: (1) A vector of s_max estimated for each iteration, (2) s_max_min calculated using C_0_min, (3) epsilon_square, (4) ep_squaremax, (5) R, (6) N, (7) Np, (8) C_0, and (9) C_0_min

Author(s)

Federico Bumbaca, Leeds School of Business, University of Colorado Boulder, <federico.bumbaca@colorado.edu>

References

Bumbaca, F. (Rico), Misra, S., & Rossi, P. E. (2020). Scalable Target Marketing: Distributed Markov Chain Monte Carlo for Bayesian Hierarchical Models. Journal of Marketing Research, 57(6), 999-1018.

Examples

```
# Generate hierarchical linear data
R = 1000
N = 2000
nobs = 5 #number of observations
nvar = 3 #columns
nz = 2

Z = matrix(runif(N*nz),ncol=nz)
Z = t(t(Z)-apply(Z,2,mean))
Delta = matrix(c(1,-1,2,0,1,0), ncol = nz)
tau0 = 0.1
iota = c(rep(1,nobs))
tcomps=NULL
a = diag(1, nrow=3)
tcomps[[1]] = list(mu=c(-5,0,0),rooti=a)
```

```
tcomps[[2]] = list(mu=c(5, -5, 2),rooti=a)
tcomps[[3]] = list(mu=c(5,5,-2),rooti=a)
tpvec = c(.33,.33,.34)
ncomp=length(tcomps)
regdata=NULL
betas=matrix(double(N*nvar),ncol=nvar)
tind=double(N)
for (reg in 1:N) {
 tempout=bayesm::rmixture(1,tpvec,tcomps)
 if (is.null(Z)){
   betas[reg,]= as.vector(tempout$x)
 }else{
   betas[reg,]=Delta%*%Z[reg,]+as.vector(tempout$x)}
 tind[reg]=tempout$z
 X=cbind(iota,matrix(runif(nobs*(nvar-1)),ncol=(nvar-1)))
 tau=tau0*runif(1,min=0.5,max=1)
 y=X%*%betas[reg,]+sqrt(tau)*rnorm(nobs)
 regdata[[reg]]=list(y=y,X=X,beta=betas[reg,],tau=tau)
}

Prior1=list(ncomp=ncomp)
keep=1
Mcmc1=list(R=R,keep=keep)
Data1=list(list(regdata=regdata,Z=Z))
returns = s_max(R = R, N = N, Data = Data1, s = 1, iterations = 2)
returns
```

Index