Package: cPseudoMaRg (via r-universe)

November 20, 2024

Type Package

Title Constructs a Correlated Pseudo-Marginal Sampler

Version 1.0.1

Description The primary function makeCPMSampler() generates a sampler function which performs the correlated pseudo-marginal method of Deligiannidis, Doucet and Pitt (2017) <arXiv:1511.04992>. If the 'rho=' argument of makeCPMSampler() is set to 0, then the generated sampler function performs the original pseudo-marginal method of Andrieu and Roberts (2009) <DOI:10.1214/07-AOS574>. The sampler function is constructed with the user's choice of prior, parameter proposal distribution, and the likelihood approximation scheme. Note that this algorithm is not automatically tuned--each one of these arguments must be carefully chosen.

License MIT + file LICENSE

RoxygenNote 7.1.1

Suggests knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Author Taylor Brown [aut, cre]

Maintainer Taylor Brown <trb5me@virginia.edu>

Date/Publication 2021-09-05 00:30:12 UTC

Repository https://tbrown122387.r-universe.dev

RemoteUrl https://github.com/cran/cPseudoMaRg

RemoteRef HEAD

RemoteSha 20c9c34f2986e3cde9f7ef6409c9957ed2f7140f

Contents

`isBadNum` *checks if a log-density evaluation is not a valid number*

Description

checks if a log-density evaluation is not a valid number

Usage

```
isBadNum(num)
```

Arguments

`num`	evaluation of a log-density

Value

TRUE or FALSE

Examples

```
isBadNum(NaN)
```

`makeCPMSampler` *correlated pseudo-marginal: generates functions that output a big vector*

Description

correlated pseudo-marginal: generates functions that output a big vector

Usage

```
makeCPMSampler(
  paramKernSamp,
  logParamKernEval,
  logPriorEval,
  logLikeApproxEval,
  yData,
  numU,
  numIters,
```

```
  rho = 0.99,
  storeEvery = 1,
  nansInLLFatal = TRUE
)
```

Arguments

paramKernSamp	function(theta) -> theta proposal
logParamKernEval	function(oldTheta, newTheta) -> logDensity.
logPriorEval	function(theta) -> logDensity.
logLikeApproxEval	function(y, thetaProposal, uProposal) -> logApproxDensity.
yData	the observed data
numU	integer number of u samples
numIters	integer number of MCMC iterations
rho	correlation tuning parameter (-1,1)
storeEvery	increase this integer if you want to use thinning
nansInLLFatal	terminate the entire chain on NaNs, or simply disregard sample

Value

vector of theta samples

Examples

```
# sim data
realTheta1 <- .2 + .3
realTheta2 <- .2
realParams <- c(realTheta1, realTheta2)
numObs <- 10
realX <- rnorm(numObs, mean = 0, sd = sqrt(realTheta2))
realY <- rnorm(numObs, mean = realX, sd = sqrt(realTheta1 - realTheta2))
# tuning params
numImportanceSamps <- 1000
numMCMCIters <- 1000
randomWalkScale <- 1.5
recordEveryTh <- 1
sampler <- makeCPMSampler(
 paramKernSamp = function(params){
   return(params + rnorm(2)*randomWalkScale)
 },
 logParamKernEval = function(oldTheta, newTheta){
   dnorm(newTheta[1], oldTheta[1], sd = randomWalkScale, log = TRUE)
   + dnorm(newTheta[2], oldTheta[2], sd = randomWalkScale, log = TRUE)
 },
 logPriorEval = function(theta){
   if( (theta[1] > theta[2]) & all(theta > 0)){
```

```
    0
  }else{
    -Inf
  }
 },
 logLikeApproxEval = function(y, thetaProposal, uProposal){
  if( (thetaProposal[1] > thetaProposal[2]) & (all(thetaProposal > 0))){
    xSamps <- uProposal*sqrt(thetaProposal[2])
    logCondLikes <- sapply(xSamps,
                           function(xsamp) {
                             sum(dnorm(y,
                                       xsamp,
                                       sqrt(thetaProposal[1] - thetaProposal[2]),
                                       log = TRUE)) })
    m <- max(logCondLikes)
    log(sum(exp(logCondLikes - m))) + m - log(length(y))
  }else{
    -Inf
  }
 },
 realY, numImportanceSamps, numMCMCIters, .99, recordEveryTh
)
res <- sampler(realParams)
```

`mean.cpmResults` *calculates the posterior mean point estimate*

Description

calculates the posterior mean point estimate

Usage

```
## S3 method for class 'cpmResults'
mean(x, ...)
```

Arguments

`x`	a cpmResults object
`...`	arguments to be passed to or from methods.

Value

a vector of parameter estimates (posterior mean)

plot.cpmResults *plots a cpmResults object*

Description

plots a cpmResults object

Usage

```
## S3 method for class 'cpmResults'
plot(x, ...)
```

Arguments

x	a cpmResults object
...	arguments to be passed to or from methods.

print.cpmResults *prints a cpmResults object*

Description

prints a cpmResults object

Usage

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

Arguments

x	a cpmResults object
...	arguments to be passed to or from methods.

Value

the same cpmResults object

Index