# Package 'SAMCpack'

October 26, 2018

**Type** Package

**Title** Stochastic Approximation Monte Carlo (SAMC) Sampler and Methods

**Version** 0.1.1

**Description** Stochastic Approximation Monte Carlo (SAMC) is one of the celebrated Markov chain Monte Carlo (MCMC) algorithms. It is known to be capable of sampling from multimodal or doubly intractable distributions. We provide generic SAMC samplers for continuous distributions. User-specified densities in R and C++ are both supported. We also provide functions for specific problems that exploit SAMC computation. See Liang et al (2010) <doi:10.1002/9780470669723> for complete introduction to the method.

**License** GPL (>= 3)

**Encoding** UTF-8

**LazyData** true

**Depends** R (>= 3.0.0)

**Imports** Rcpp, RcppXPtrUtils, utils, Rdpack

**LinkingTo** Rcpp, RcppArmadillo

**RoxygenNote** 6.1.0

**Suggests** knitr, rmarkdown, microbenchmark, pander, geoR, RandomFields

**VignetteBuilder** knitr

**RdMacros** Rdpack

**NeedsCompilation** yes

**Author** Yichen Cheng [aut],
Ick Hoon Jin [aut],
Faming Liang [aut],
Kisung You [aut, cre]

**Maintainer** Kisung You <kyou@nd.edu>

**Repository** CRAN

**Date/Publication** 2018-10-26 08:30:07 UTC

# R topics documented:



---

`SAMCpack-package` *Stochastic Approximation Monte Carlo (SAMC) Sampler and Methods*

---

## Description

Stochastic Approximation Monte Carlo (SAMC) is one of the celebrated Markov chain Monte Carlo (MCMC) algorithms. It is known to be capable of sampling from multimodal or doubly intractable distributions. We provide generic SAMC samplers for continuous distributions. User-specified densities in R and C++ are both supported. We also provide functions for specific problems that exploit SAMC computation. See Liang et al (2010) <doi:10.1002/9780470669723> for complete introduction to the method.

## Author(s)

Kisung You

---

`gdata` *Dataset for SAMCrsa*

---

## Description

Simulated data list with response variable distributed with 'powered exponential' covariance matrix, with $\phi = 25, \tau = 1, \kappa = 1, \sigma = 1$.

## Usage

```
data(gdata)
```

## Format

A sample dataset with 1000 observations. It's save as an R's native `list`, and see the table for description of variables in the `gdata` list,

| VARIABLE | SIZE | DESCRIPTION |
|---|---|---|
| `coords` | $(1000 \times 2)$ matrix | 2D location for each observation |
| `y` | length-1000 vector | response value |
| `X` | length-1000 vcetor | covariate |

This is the default example data for SAMCrsa.

## Details

Below is the code used to generate *gdata*:

```
require("geoR")
require("RandomFields")
DataNum=1000
gData=grf(DataNum,grid="irreg",DataNum,DataNum,xlims=c(0,100),ylims=c(0,100),nsim=1,mean=0,
          cov.mode="powered.exponential",cov.par=c(1,25),nugget=1,kappa=1)
gdata = list(y=gData$data+.5+rnorm(DataNum), X=rnorm(DataNum), coords=gData$coords)
```

---

SAMC *Stochastic Approximation Monte Carlo (SAMC) Sampler*

---

## Description

The function SAMC is a generic SAMC sampler for distributions on continuous domain. An $R$ function for negative log density of your choice is required, as well as some parameters in SAMC framework.

## Usage

```
SAMC(nv, energy, options = list())
```

## Arguments

| | |
|---|---|
| nv | number of variables. |
| energy | an R function for negative log density. |
| options | a list specifying parameters/options for SAMC algorithm. Below, vector($k$) means a vector of length $k$, and matrix likewise. |

| PARAMETER | SPECIFICATION | DESCRIPTION |
|---|---|---|
| domain | vector(2) or matrix($(nv \times 2)$) | domain of sample space |
| partition | vector($m$) | energy partition |
| vecpi | vector($m-1$) | desired sampling distribution |
| tau | positive number | temperature |
| niter | positive integer | number of iterations to be run |
| t0 | $(0.5, 1]$ | gain factor sequence value |
| xi | positive number | gain factor sequence exponent |
| stepsize | positive number | stepsize for random-walk sampler |
| trange | vector(2) or matrix($m \times 2$) | domain of estimates for $\log(g_i/\pi_i)$ |

## Value

a named list containing

**samples** an $(niter \times nv)$ samples generated.

**frequency** length-$m$ vector of visiting frequency for energy partition.

**theta** length-$m$ vector of estimates of $\log(g_i/\pi_i)$

## Author(s)

Kisung You

## References

Liang F, Liu C, Carroll R (2010). *Advanced Markov Chain Monte Carlo Methods: Learning from Past Samples*, series Wiley Series in Computational Statistics. Wiley. ISBN 9780470748268.

## Examples

```
##### Two-Dimensional Multimodal sampling
## Step 1 : Define negative log-density function as an R function
func_r = function(x){
x1 = x[1]; x2 = x[2];
val1 = (-(x1*sin(20*x2)+x2*sin(20*x1))^2)*cosh(sin(10*x1)*x1);
val2 = (-(x1*cos(10*x2)-x2*sin(10*x1))^2)*cosh(cos(20*x2)*x2);
return(val1+val2);
}

## Step 2 : Prepare a setting option
myoption = list()
myoption$partition = c(-Inf,seq(from=-8,to=0,length.out=41))
myoption$tau       = 1.0
myoption$domain    = c(-1.1,1.1)
myoption$vecpi     = as.vector(rep(1/41,41))
myoption$niter     = 20000
myoption$stepsize  = c(0.25, 10)

## Step 3 : Run The Code
res = SAMC(2,func_r,options=myoption)

## Step 4 : Visualize
select = seq(from=101,to=myoption$niter,by=100) # 100 burn-in, 1/100 thinning
par(mfrow=c(1,2))
plot(res$samples[select,1], res$samples[select,2],xlab='x',ylab='y',main='samples')
barplot(as.vector(res$frequency/sum(res$frequency)),
        main="visiting frequency by energy partition",
        names.arg=myoption$partition[-1], xlab="energy")
```

SAMCPLUS *SAMC Sampler with C++*

## Description

The function SAMCPLUS is a generic SAMC sampler for distributions on continuous domain. Instead of an R function, SAMCPLUS requires a function pointer to be provided for faster sampling, with all other values and parameters being equal to its cousin SAMC. We limited the flexibility of the function pointer to be passed. See the below for more details or the vignette.

## Usage

```
SAMCPLUS(nv, energy, data = NA, options = list())
```

## Arguments

| | |
|---|---|
| nv | number of variables. |
| energy | a CPP function pointer for negative log density. |
| data | extra data to be supplemented. It should be a vector, a matrix, or a list. |
| options | a list specifying parameters/options for SAMC algorithm, |

| PARAMETER | SPECIFICATION | DESCRIPTION |
|---|---|---|
| domain | vector(2) or matrix($(nv \times 2)$) | domain of sample space |
| partition | vector($m$) | energy partition |
| vecpi | vector($m-1$) | desired sampling distribution |
| tau | positive number | temperature |
| niter | positive integer | number of iterations to be run |
| t0 | $(0.5, 1]$ | gain factor sequence value |
| xi | positive number | gain factor sequence exponent |
| stepsize | positive number | stepsize for random-walk sampler |
| trange | vector(2) or matrix($m \times 2$) | domain of estimates for $\log(g_i/\pi_i)$ |

## Value

a named list containing

**samples** an $(niter \times nv)$ samples generated.

**frequency** length-$m$ vector of visiting frequency for energy partition.

**theta** length-$m$ vector of estimates of $\log(g_i/\pi_i)$

## Note on writing your own C++ function

First, the output should be returned as SEXP rather than double in evaluating the negative log density. Second, the variable and extra data should be provided as arma::vec and arma::mat type, with an exception for Rcpp::List for list-valued data. This means, for the data Even though we

could let data to be passed freely, we believe using **RcppArmadillo**, which is a templated C++ linear algebra library, enables easier writing of one's own C++ code in a style of R or MATLAB while providing sufficient computational power. Furthermore, limiting extra data to one of 3 types (vector, matrix, and list) reduces potential type-matching issue in encapsulating of the current environment by removing unexpected errors a user might have incurred.

## Author(s)

Kisung You

## References

Liang F, Liu C, Carroll R (2010). *Advanced Markov Chain Monte Carlo Methods: Learning from Past Samples*, series Wiley Series in Computational Statistics. Wiley. ISBN 9780470748268.

## Examples

```
##### Two-Dimensional Multimodal sampling
## Step 1 : Define negative log-density function as a CPP function
cppscript = "SEXP funcH(arma::vec x){
double x1 = x(0);
double x2 = x(1);
double val1 = (-std::pow((x1*sin(20*x2)+x2*sin(20*x1)),2))*cosh(sin(10*x1)*x1);
double val2 = (-std::pow((x1*cos(10*x2)-x2*sin(10*x1)),2))*cosh(cos(20*x2)*x2);
return Rcpp::wrap(val1+val2);
}"
func_ptr = RcppXPtrUtils::cppXPtr(cppscript,depends="RcppArmadillo") # as a pointer

## Step 2 : Prepare a setting option
myoption = list()
myoption$partition = c(-Inf,seq(from=-8,to=0,length.out=41))
myoption$tau       = 1.0
myoption$domain    = c(-1.1,1.1)
myoption$vecpi     = as.vector(rep(1/41,41))
myoption$niter     = 200001
myoption$stepsize  = 0.25

## Step 3 : Run The Code
res = SAMCPLUS(2,func_ptr,options=myoption)

## Step 4 : Visualize
select = seq(from=101,to=myoption$niter,by=100) # 100 burn-in, 1/100 thinning
par(mfrow=c(1,2))
plot(res$samples[select,1], res$samples[select,2],xlab='x',ylab='y',main='samples')
barplot(as.vector(res$frequency/sum(res$frequency)),
        main="visiting frequency by energy partition",
        names.arg=myoption$partition[-1], xlab="energy")

##### (2) Use Extra Data
## Define negative log-density function as CPP function
cppscript2 = "SEXP funcH2(arma::vec x, arma::vec data){
```

```
double x1 = x(0);
double x2 = x(1);
double par1 = data(0);
double par2 = data(1);

double val1 = (-std::pow((x1*sin(par2*x2)+x2*sin(par2*x1)),2))*cosh(sin(par1*x1)*x1);
double val2 = (-std::pow((x1*cos(par1*x2)-x2*sin(par1*x1)),2))*cosh(cos(par2*x2)*x2);
return Rcpp::wrap(val1+val2);
}"
func_ptr2 = RcppXPtrUtils::cppXPtr(cppscript2,depends="RcppArmadillo") # as a pointer

## Run The Code
vecdata = as.vector(c(10,20))
res2 = SAMCPLUS(2,func_ptr2,data=vecdata, options=myoption)
select = seq(from=101,to=ex_niter,by=100) # 100 burn-in, 1/100 thinning
par(mfrow=c(1,2))
plot(res2$samples[select,1], res2$samples[select,2],xlab='x',ylab='y',main='samples')
barplot(as.vector(res2$frequency/sum(res2$frequency)),
        main="visiting frequency by energy partition",
        names.arg=ex_part[2:(m+1)], xlab="energy")
```

---

SAMCrsa *A Resampling-based Stochastic Approximation Method for Analysis of Large Geostatitical data*

---

## Description

Performs parameter estimation using a resampling-based Stochastic Approximation (RSA) method. It is a stochatic approximation method. At every iteration, only a subset of the data is drawn and used to update the estimation of the parameters. The data are assumed to have a powered exponential correlation structure.

## Usage

```
SAMCrsa(coords, y, X = NULL, nsubset = max(ceiling(length(y)/5), 10),
  stepscale = 200, niter = 2500, warm = 100)
```

## Arguments

| | |
|---|---|
| coords | an $(n \times 2)$ matrix. 2D location coordinates. |
| y | a length-$n$ vector of response value. |
| X | an $(n \times k)$ matrix of extra covariates. |
| nsubset | the size of the subset drawn from the data. It is recommended to be set to 300 or higher. |

| | |
|---|---|
| `stepscale` | gain factor control. It specifies the number of iterations when the gain factor begins to shrink. For example, one can be set it equal to 2 times the burn-in steps. |
| `niter` | the total number of iterations for stochastic approximation. In practice, it is recommended to be set to 2500 or higher. |
| `warm` | the number of burn-in iterations |

## Value

a named list containing

**beta** the coefficient estimates of the mean effect. It is a vector of length equal to the number of coefficients plus 1.

**phi** the shape estimate in the powered exponential correlation matrix.

**sigmasq** the estimate of error variance.

**tausq** the estimate of nugget variance.

## Author(s)

Yichen Cheng, Faming Liang, Kisung You

## References

Liang F, Cheng Y, Song Q, Park J, Yang P (2013). "A Resampling-Based Stochastic Approximation Method for Analysis of Large Geostatistical Data." *Journal of the American Statistical Association*, **108**(501), 325-339. doi: 10.1080/01621459.2012.746061.

## Examples

```
##### load example data pre-loaded
data(gdata)

##### run RSA
output = SAMCrsa(gdata$coords, gdata$y, gdata$X, nsubset=50, stepscale=40, niter=100, warm=20)
```

# Index