Package 'HI'

February 19, 2015

Title Simulation from distributions supported by nested hyperplanes

Version 0.4

Date 2013-03-26

Author Giovanni Petris <GPetris@Uark.edu> and Luca Tardella

<luca.tardella@@uniroma1.it>; original C code for ARMS by Wally Gilks.

Maintainer Giovanni Petris <GPetris@Uark.edu>

Depends R (>= 1.7.0)

Description Simulation from distributions supported by nested hyperplanes, using the algorithm described in Petris & Tardella, ``A geometric approach to transdimensional Markov chain Monte Carlo'', Canadian Journal of Statistics, v.31, n.4, (2003). Also random direction multivariate Adaptive Rejection Metropolis Sampling.

License GPL (>= 2)

NeedsCompilation yes

Repository CRAN

Date/Publication 2013-03-26 17:04:14

R topics documented:

arms *Function to perform Adaptive Rejection Metropolis Sampling*

Description

Generates a sequence of random variables using ARMS. For multivariate densities, ARMS is used along randomly selected straight lines through the current point.

Usage

```
arms(y.start, myldens, indFunc, n.sample, ...)
```

Arguments

`y.start`	Initial point
`myldens`	Univariate or multivariate log target density
`indFunc`	Indicator function of the convex support of the target density
`n.sample`	Desired sample size
`...`	Parameters passed to `myldens` and `indFunc`

Details

Strictly speaking, the support of the target density must be a bounded convex set. When this is not the case, the following tricks usually work. If the support is not bounded, restrict it to a bounded set having probability practically one. A workaround, if the support is not convex, is to consider the convex set generated by the support and define `myldens` to return `log(.Machine$double.xmin)` outside the true support (see the last example.)

The next point is generated along a randomly selected line through the current point using arms.

Make sure the value returned by `myldens` is never smaller than `log(.Machine$double.xmin)`, to avoid divisions by zero.

Value

An `n.sample` by `length(y.start)` matrix, whose rows are the sampled points.

Note

The function is based on original C code by Wally Gilks for the univariate case, http://www.mrc-bsu.cam.ac.uk/pub/methodology/adaptive_rejection/.

Author(s)

Giovanni Petris <GPetris@uark.edu>

References

Gilks, W.R., Best, N.G. and Tan, K.K.C. (1995) Adaptive rejection Metropolis sampling within Gibbs sampling (Corr: 97V46 p541-542 with Neal, R.M.), *Applied Statistics* **44**:455–472.

Examples

```
#### ==> Warning: running the examples may take a few minutes! <== ####
### Univariate densities
## Unif(-r,r)
y <- arms(runif(1,-1,1), function(x,r) 1, function(x,r) (x>-r)*(x<r), 5000, r=2)
summary(y); hist(y,prob=TRUE,main="Unif(-r,r); r=2")
## Normal(mean,1)
norldens <- function(x,mean) -(x-mean)^2/2
y <- arms(runif(1,3,17), norldens, function(x,mean) ((x-mean)>-7)*((x-mean)<7),
          5000, mean=10)
summary(y); hist(y,prob=TRUE,main="Gaussian(m,1); m=10")
curve(dnorm(x,mean=10),3,17,add=TRUE)
## Exponential(1)
y <- arms(5, function(x) -x, function(x) (x>0)*(x<70), 5000)
summary(y); hist(y,prob=TRUE,main="Exponential(1)")
curve(exp(-x),0,8,add=TRUE)
## Gamma(4.5,1)
y <- arms(runif(1,1e-4,20), function(x) 3.5*log(x)-x,
          function(x) (x>1e-4)*(x<20), 5000)
summary(y); hist(y,prob=TRUE,main="Gamma(4.5,1)")
curve(dgamma(x,shape=4.5,scale=1),1e-4,20,add=TRUE)
## Gamma(0.5,1) (this one is not log-concave)
y <- arms(runif(1,1e-8,10), function(x) -0.5*log(x)-x,
          function(x) (x>1e-8)*(x<10), 5000)
summary(y); hist(y,prob=TRUE,main="Gamma(0.5,1)")
curve(dgamma(x,shape=0.5,scale=1),1e-8,10,add=TRUE)
## Beta(.2,.2) (this one neither)
y <- arms(runif(1), function(x) (0.2-1)*log(x)+(0.2-1)*log(1-x),
          function(x) (x>1e-5)*(x<1-1e-5), 5000)
summary(y); hist(y,prob=TRUE,main="Beta(0.2,0.2)")
curve(dbeta(x,0.2,0.2),1e-5,1-1e-5,add=TRUE)
## Triangular
y <- arms(runif(1,-1,1), function(x) log(1-abs(x)), function(x) abs(x)<1, 5000)
summary(y); hist(y,prob=TRUE,ylim=c(0,1),main="Triangular")
curve(1-abs(x),-1,1,add=TRUE)
## Multimodal examples (Mixture of normals)
lmixnorm <- function(x,weights,means,sds) {
    log(crossprod(weights, exp(-0.5*((x-means)/sds)^2 - log(sds))))
}
y <- arms(0, lmixnorm, function(x,...) (x>(-100))*(x<100), 5000, weights=c(1,3,2),
          means=c(-10,0,10), sds=c(1.5,3,1.5))
summary(y); hist(y,prob=TRUE,main="Mixture of Normals")
curve(colSums(c(1,3,2)/6*dnorm(matrix(x,3,length(x),byrow=TRUE),c(-10,0,10),c(1.5,3,1.5))),
      par("usr")[1], par("usr")[2], add=TRUE)

### Bivariate densities
## Bivariate standard normal
```

```
y <- arms(c(0,2), function(x) -crossprod(x)/2,
          function(x) (min(x)>-5)*(max(x)<5), 500)
plot(y, main="Bivariate standard normal", asp=1)
## Uniform in the unit square
y <- arms(c(0.2,.6), function(x) 1,
          function(x) (min(x)>0)*(max(x)<1), 500)
plot(y, main="Uniform in the unit square", asp=1)
polygon(c(0,1,1,0),c(0,0,1,1))
## Uniform in the circle of radius r
y <- arms(c(0.2,0), function(x,...) 1,
          function(x,r2) sum(x^2)<r2, 500, r2=2^2)
plot(y, main="Uniform in the circle of radius r; r=2", asp=1)
curve(-sqrt(4-x^2), -2, 2, add=TRUE)
curve(sqrt(4-x^2), -2, 2, add=TRUE)
## Uniform on the simplex
simp <- function(x) if ( any(x<0) || (sum(x)>1) ) 0 else 1
y <- arms(c(0.2,0.2), function(x) 1, simp, 500)
plot(y, xlim=c(0,1), ylim=c(0,1), main="Uniform in the simplex", asp=1)
polygon(c(0,1,0), c(0,0,1))
## A bimodal distribution (mixture of normals)
bimodal <- function(x) { log(prod(dnorm(x,mean=3))+prod(dnorm(x,mean=-3))) }
y <- arms(c(-2,2), bimodal, function(x) all(x>(-10))*all(x<(10)), 500)
plot(y, main="Mixture of bivariate Normals", asp=1)

## A bivariate distribution with non-convex support
support <- function(x) {
    return(as.numeric( -1 < x[2] && x[2] < 1 &&
                       -2 < x[1] &&
                       ( x[1] < 1 || crossprod(x-c(1,0)) < 1 ) ) )
}
Min.log <- log(.Machine$double.xmin) + 10
logf <- function(x) {
    if ( x[1] < 0 ) return(log(1/4))
    else
        if (crossprod(x-c(1,0)) < 1 ) return(log(1/pi))
    return(Min.log)
}
x <- as.matrix(expand.grid(seq(-2.2,2.2,length=40),seq(-1.1,1.1,length=40)))
y <- sapply(1:nrow(x), function(i) support(x[i,]))
plot(x,type='n',asp=1)
points(x[y==1,],pch=1,cex=1,col='green')
z <- arms(c(0,0), logf, support, 1000)
points(z,pch=20,cex=0.5,col='blue')
polygon(c(-2,0,0,-2),c(-1,-1,1,1))
curve(-sqrt(1-(x-1)^2),0,2,add=TRUE)
curve(sqrt(1-(x-1)^2),0,2,add=TRUE)
sum( z[,1] < 0 ) # sampled points in the square
sum( apply(t(z)-c(1,0),2,crossprod) < 1 ) # sampled points in the circle
```

---

convex.bounds *Function to find the boundaries of a convex set*

Description

Finds the boundaries of a bounded convex set along a specified straight line, using a bisection approach. It is mainly intended for use within arms.

Usage

```
convex.bounds(x, dir, indFunc, ..., tol=1e-07)
```

Arguments

x	A point within the set
dir	A vector specifying a direction
indFunc	Indicator function of the set
...	Parameters passed to indFunc
tol	Tolerance

Details

Uses a bisection algorithm along a line having parametric representation x + t * dir.

Value

A two dimensional vector ans. The boundaries of the set are x + ans[1] * dir and x + ans[2] * dir.

Author(s)

Giovanni Petris <GPetris@uark.edu>

Examples

```
## boundaries of a unit circle
convex.bounds(c(0,0), c(1,1), indFunc=function(x) crossprod(x)<1)
```

---

rballunif — *Function to generate a random vector uniformly distributed in a sphere*

Description

Generates a random vector from a uniform distribution in a sphere centered at the origin

Usage

```
rballunif(n,d)
```

Arguments

n	Dimension of the random vector to be generated
d	Radius of the sphere

Details

The function is not vectorized: it is intended to generate one random vector at a time.

Value

An pseudo-random vector from a uniform distribution in the n-dimensional sphere centered at the origin and having radius d

Author(s)

Giovanni Petris <GPetris@uark.edu>

---

trans.dens *Functions for transdimensional MCMC*

Description

Computes the value of the 'g' density at a given point and, optionally, returns the backtransformed point and the model to which the point belongs.

Usage

```
trans.dens(y, ldens.list, which.models, ..., back.transform=F)
trans.up(x, ldens.list, which.models, ...)
trans2(y, ldens.list, k, ...)
transUp2(y, ldens.list, k, ...)
transBack2(y, ldens.list, k, ...)
```

Arguments

y	Vector or matrix of points (by row) at which the density of the absolutely continuous auxiliary distribution has to be evaluated
x	Vector or matrix of points corresponding to y (see examples)
ldens.list	List of densities (of submodels)
which.models	List of integers, in increasing order, giving the number of components to be dropped when evaluating the density in which.models in the corresponding position. A first element equal 0 (full model) is added if not already present
back.transform	Logical that determines the output
k	Difference between the dimension of the larger model and the dimension of the smaller model
...	Other arguments passed to the functions in ldens.list

Details

See the reference for details. The functions with the 2 in the name operate on pairs of models only.

Value

If back.transform=F, trans.dens returns the density of the absolutely continuous auxiliary distribution evaluated at the point(s) y.\ If back.transform=T, trans.dens returns in addition the point x corresponding to y in the original space and the index of the subspace to which x belongs.\ trans.up is a (stochastic) right inverse of the correspondence between y and x

Author(s)

Giovanni Petris <GPetris@uark.edu>, Luca Tardella

References

Petris \& Tardella, A geometric approach to transdimensional Markov chain Monte Carlo. *The Canadian Journal of Statistics*, vol.31, n.4, (2003).

Examples

```
#### ==> Warning: running the examples may take a few minutes! <== ####
### Generate a sample from a mixture of 0,1,2-dim standard normals
ldens.list <- list(f0 = function(x) sum(dnorm(x,log=TRUE)),
                   f1 = function(x) dnorm(x,log=TRUE),
                   f2 = function() 0)
trans.mix <- function(y) {
    trans.dens(y, ldens.list=ldens.list, which.models=0:2)
}

trans.rmix <- arms(c(0,0), trans.mix, function(x) crossprod(x)<1e4, 500)
rmix <- trans.dens(y=trans.rmix, ldens.list=ldens.list,
                      which.models=0:2, back.transform = TRUE)
table(rmix[,2])/nrow(rmix) # should be about equally distributed
plot(trans.rmix,col=rmix[,2]+3,asp=1, xlab="y.1", ylab="y.2",
     main="A sample from the auxiliary continuous distribution")
x <- rmix[,-(1:2)]
plot(x, col=rmix[,2]+3, asp=1,
     main="The sample transformed back to the original space")
### trans.up as a right inverse of trans.dens
set.seed(6324)
y <- trans.up(x, ldens.list, 0:2)
stopifnot(all.equal(x, trans.dens(y, ldens.list, 0:2, back.transform=TRUE)[,-(1:2)]))

### More trans.up
z <- trans.up(matrix(0,1000,2), ldens.list, 0:2)
plot(z,asp=1,col=5) # should look uniform in a circle corresponding to model 2
z <- trans.up(cbind(runif(1000,-3,3),0), ldens.list, 0:2)
plot(z,asp=1,col=4) # should look uniform in a region corresponding to model 1

### trans2, transBack2
```

```
ldens.list <- list(f0 = function(x) sum(dnorm(x,log=TRUE)),
                   f1 = function(x) dnorm(x,log=TRUE))
trans.mix <- function(y) {
    trans2(y, ldens.list=ldens.list, k=1)[-2]
}
trans.rmix <- arms(c(0,0), trans.mix, function(x) crossprod(x)<1e2, 1000)
rmix <- transBack2(y=trans.rmix, ldens.list=ldens.list, k=1)
table(rmix[,2]==0)/nrow(rmix) # should be about equally distributed
plot(trans.rmix,col=(rmix[,2]==0)+3,asp=1, xlab="y.1", ylab="y.2",
     main="A sample from the auxiliary continuous distribution")
plot(rmix, col=(rmix[,2]==0)+3, asp=1,
     main="The sample transformed back to the original space")

### trunsUp2
z <- t(sapply(1:1000, function(i) transUp2(c(-2+0.004*i,0), ldens.list, 1)))
plot(z,asp=1,col=2)
```

Index