Package 'ranlip'

June 24, 2021

Type Package

Title Generation of Random Vectors with User-Defined Density

Version 1.0.1

Date 2021-06-18

Maintainer Gleb Beliakov <gleb@deakin.edu.au>

Author Gleb Beliakov [aut, cre],
Daniela Calderon [aut],
James Theiler [aut],
Brian Gough [aut]

Description Random vectors with arbitrary Lipschitz density are generated using acceptance/ rejection. The method is based on G. Beliakov (2005) <doi:10.1016/j.cpc.2005.03.105>.

License LGPL-3

LazyData FALSE

Imports Rcpp (>= 1.0.0)

LinkingTo Rcpp

RoxygenNote 5.0.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2021-06-24 07:40:02 UTC

R topics documented:

`ranlip` *Ranlip Package*

Description

This function shows a list of function included in this toolbox

Usage

```
ranlip()
```

Details

The method of building the hat function, and generation of random variates using acceptance/ rejection described

Value

`output`	No return value

Author(s)

Gleb Beliakov, Daniela L. Calderon, Deakin University

References

[1] G. Beliakov. Class library ranlip for multivariate nonuniform random variate generation. Computer Physics Communications, 170:93-108, 2005.

[2] G. Beliakov. Universal nonuniform random vector generator based on acceptance-rejection. ACM Transactions on Modeling and Computer Simulation, 15:205-232, 2005.

[3] L. Devroye Non-uniform Random Variate Generation. Springer Verlag New York, 1986.

[4] W.Hormann, A rejection technique for sampling from t-concave distribution. ACM Transactions on Mathematical Software, 21:182-193,1995.

[5]W.Hormann, J. Leydold and G Derflinger. Austomatic Nonuniform Random Variate Generation, Springer, Berlin, 2004.

[6] J. Leydold and W.Hormann. A sweep-plane algorithm for generatin random tuples in simple polytopes. Mathematics of Computation, 67:1617-1635, 1998.

[7] M. Luescher. A portable high-quality random number generator for lattice field theory calculation. Computer Physics Communications, 79:100-11, 1994.

Examples

```
ranlip()
```

`ranlip.FreeMem` *Free Memory Function*

Description

Freeing the memory occupied by the data structures. It destroys the hat function (Preparehatfunction) and RandomVec().

Usage

```
ranlip.FreeMem()
```

Value

`output`	No return value, called for side effects

Author(s)

Gleb Beliakov, Daniela L. Calderon, Deakin University

Examples

```
dim<-3
left<-c(0,0,0)
right<-c(5,5,5)

ranlip.Init(dim, left, right);
    ranlip.FreeMem();
```

`ranlip.Init` *Initialization of the internal variables*

Description

Function for initialing the internal variables. Int must be called only once before any other method.

Usage

```
ranlip.Init(dim, left, right)
```

Arguments

`dim`	The dimension
`left`	An array of size dim which determine the domain of p: leftli <= xli <= rightli
`right`	An array of size dim which determine the domain of p: leftli <= xli <= rightli

Value

`output`	No return value, called to initialise the internal variables.

Author(s)

Gleb Beliakov, Daniela L. Calderon, Deakin University

Examples

```
    dim<-3
left<-c(0,0,0)
right<-c(1,1,1)
ranlip.Init(dim, left, right);

ranlip.FreeMem();
```

`ranlip.LoadPartition` *Load the computed hat function*

Description

Loads previously computed hat function from file name(string)

Usage

```
ranlip.LoadPartition(string)
```

Arguments

`string`	The file name to read saved partition from

Value

`output`	The output is 0 if the function was successful, or: 2 if the file cannot be opened, 3 if file is corrupted, 4 if memory cannot be allocated

Author(s)

Gleb Beliakov, Daniela L. Calderon, Deakin University

Examples

```
Fn <- function(x,dim){
out <- sum(exp(-x))

return(out)
}
    out<-ranlip.LoadPartition("mypartition.txt")
if(out>0) {
print("Error loading hat function. ")
err<-switch(out,"Unknown","Fice cannot be opened",
 "File is corrupted","Memory not allocated")
print(err)
} else {

r<-ranlip.RandomVec( Fn)
print(r)
}
    ranlip.FreeMem();
```

---

ranlip.PrepareHatFunction

Builds the hat function for a given Lipschitz constant

Description

Function for Building the hat function using Lipschitz constant

Usage

```
ranlip.PrepareHatFunction(num, numfine, Lip, dist)
```

Arguments

`num`	The number of subdivisions in each variable to partition the Domain D into hyperrectangles D|k. On each D|k, the hat function will have a constant value h|k
`numfine`	The number of subdivisions in the finer partition in each variable. Each D|k is subdivided into (numfine-1)^dim smaller hyperrectangles, in order to improve the quality of the overstimate h|k. nunmfine should be a power of 2 for numerical efficiency reason (if not, it will be automatically changed to a power of 2 larger than the supplied value) numdine can be 2, in which case the fine partition is not used
`Lip`	Lipschitz constant supplied
`dist`	The distribution function p(x) where x is the array of size dim.

Value

`output`	No return value. Generates and stores internally the hat function.

Author(s)

Gleb Beliakov, Daniela L. Calderon, Deakin University

Examples

```
    dim<-2
left<-c(-1,-1,0)
right<-c(1,1,5)

ranlip.Init(dim, left, right)


num <- 10
numfine <- 2
Lip <- 1



Fn <- function(x,dim){
r<-x[1]*x[1]+x[2]*x[2]
r<-sqrt(r)
out <- exp(-( (x[1]+0.2)^2+(x[2]+0.1)^2)/1.1 )*exp(-sqrt(r))
return(out)
}


ranlip.PrepareHatFunction(num, numfine, Lip, Fn);
ranlip.RandomVec(Fn)
r<-ranlip.RandomVec( Fn)
print(r)
r<-ranlip.RandomVec( Fn)
print(r)

  ranlip.FreeMem()
```

`ranlip.PrepareHatFunctionAuto` *Computation function of building the hat function and an estimated Lipschitz constant*

Description

Builds the hat function and automatically computes an estimate to the Lipschitz constant.

Usage

```
ranlip.PrepareHatFunctionAuto(num, numfine, minLip, dist)
```

Arguments

num	The number of subdivisions in each variable to partition the Domain D into hyperrectangles D|k. On each D|k, the hat function will have a constant value h|k
numfine	The number of subdivisions in the finer partition in each variable. Each D|k is subdivided into (numfine-1)^dim smaller hyperrectangles, in order to improve the quality of the overstimate h|k. nunmfine should be a power of 2 for numerical efficiency reason (if not, it will be automatically changed to a power of 2 larger than the supplied value) numdine can be 2, in which case the fine partition is not used
minLip	the lower bound on the value of the computed Lipschitz constant, the default value is 0
dist	The distribution function p(x) where x is the array of size dim.

Value

output	The estimated Lipschitz constant. Stores the hat function internlly.

Author(s)

Gleb Beliakov, Daniela L. Calderon, Deakin University

Examples

```
    dim<-3
left<-c(0,0,0)
right<-c(5,5,5)

ranlip.Init(dim, left, right);


num <- 10
numfine <- 2
MinLip <- 1



Fn <- function(x,dim){
r<-x[1]*x[1]+x[2]*x[2]
out <- exp(-( (x[1]+0.2)^2+(x[2]+0.1)^2)/1.1 )*(1-exp(-sqrt(r)))
return(out)
}
```

```
Lip<-ranlip.PrepareHatFunctionAuto(num, numfine, MinLip, Fn)

print(Lip)
ranlip.RandomVec( Fn)
  ranlip.FreeMem()
```

`ranlip.RandomVec` *Generated variate function with density p*

Description

Function for generating a random variate with density p. It should be called after ranlip.PrepareHatFunctionAuto() and/or ranlip.PrepareHatFunction().

Usage

```
ranlip.RandomVec(dist)
```

Arguments

`dist`	The distribution function p(x) where x is the array of size dim..

Value

`output`	The output is a random variate with the density p.

Author(s)

Gleb Beliakov, Daniela L. Calderon, Deakin University

Examples

```
    dim<-3
left<-c(0,0,0)
right<-c(5,5,5)

ranlip.Init(dim, left, right);

num <- 10
numfine <- 4
MinLip <- 1

Fn <- function(x,dim){
out <- exp(-sum(x*x))

return(out)
}
```

```
ranlip.PrepareHatFunctionAuto(num, numfine, MinLip, Fn)

r<-ranlip.RandomVec( Fn)

print(r)
r<-ranlip.RandomVec( Fn)

print(r)

   ranlip.FreeMem()
```

`ranlip.RandomVecN` *Generates variate function with density p*

Description

Function for generating n random variates with density p. It should be called after ranlip.PrepareHatFunctionAuto() and ranlip.PrepareHatFunction().

Usage

```
ranlip.RandomVecN(n,dist)
```

Arguments

`n`	The number of random vectors desired
`dist`	The distribution function p(x) where x is the array of size dim..

Value

`output`	The output is n random variates with the density p, in a matrix arranged by rows.

Author(s)

Gleb Beliakov, Daniela L. Calderon, Deakin University

Examples

```
    dim<-2
left<-c(-2,-2)
right<-c(2,2)

ranlip.Init(dim, left, right);

num <- 10
numfine <- 4
MinLip <- 1

Fn <- function(x,dim){
r<-x[1]*x[1]+x[2]*x[2]
out <- exp(-( (x[1]+0.2)^2+(x[2]+0.1)^2)/1.1 )*(1-exp(-sqrt(r)))
return(out)
}


ranlip.PrepareHatFunctionAuto(num, numfine, MinLip, Fn)

rv<-ranlip.RandomVecN(100, Fn)


plot(rv[,1],rv[,2],cex=0.5)

    ranlip.FreeMem()
```

---

ranlip.SavePartition *Saves the computed hat function into a file*

Description

Function for saving previously computed hat function to file name(string)

Usage

```
ranlip.SavePartition(string)
```

Arguments

string	The file name to save the partition

Value

output	The output is 0 if the function was successful, or 1 if no hat function was computed, or 2 if the file cannot be opened.

Author(s)

Gleb Beliakov, Daniela L. Calderon, Deakin University

Examples

```
    dim<-3
left<-c(0,0,0)
right<-c(5,5,5)

ranlip.Init(dim, left, right);


num <- 10
numfine <- 2
MinLip <- 1

Fn <- function(x,dim){
out <- exp(-sum(x*x))

return(out)
}

ranlip.PrepareHatFunctionAuto(num, numfine, MinLip, Fn);

# we don't want to create this file unnecessarily
 out<-ranlip.SavePartition("mypartition.txt")
if(out>0) {print("File cannot be opened.")}


  ranlip.FreeMem()
```

---

ranlip.Seed — *Function of setting the seed*

Description

Function for setting the seed of the default uniform random number generator ranlux.

Usage

```
ranlip.Seed(seed)
```

Arguments

`seed`	Integer value to seed the random generator

Value

`output`	No return value, called to set the seed of the random generator.

Author(s)

Gleb Beliakov, Daniela L. Calderon, Deakin University

Examples

```
ranlip.Seed(17);
```

Index