Package ‘mlmc’

October 13, 2022

Type Package

Title Multi-Level Monte Carlo

Version 1.0.0

Maintainer Louis Aslett <aslett@stats.ox.ac.uk>

Description An implementation of Multi-level Monte Carlo for R. This package builds on the original 'Matlab' and C++ implementations by Mike Giles to provide a full MLMC driver and example level samplers. Multi-core parallel sampling of levels is provided built-in.

Imports ggplot2, grid, parallel, Rcpp

License GPL-2

LazyData TRUE

RoxygenNote 5.0.1

LinkingTo Rcpp

NeedsCompilation yes

Author Louis Aslett [cre, aut, trl],
Mike Giles [ctb],
Tigran Nagapetyan [ctb],
Sebastian Vollmer [ctb]

Repository CRAN

Date/Publication 2016-04-24 23:19:03

R topics documented:

mcqmc06_l — *Financial options using a Milstein discretisation*

Description

Financial options based on scalar geometric Brownian motion, similar to Mike Giles' MCQMC06 paper, using a Milstein discretisation

Usage

```
mcqmc06_l(l, N, option)
```

Arguments

`l`	the level to be simulated.
`N`	the number of samples to be computed.
`option`	the option type, between 1 and 5. The options are: **1 = European call;** **2 = Asian call;** **3 = lookback call;** **4 = digital call;** **5 = barrier call.**

Details

This function is based on GPL-2 C++ code by Mike Giles.

Author(s)

Louis Aslett <aslett@stats.ox.ac.uk>

Mike Giles <Mike.Giles@maths.ox.ac.uk>

References

M.B. Giles. 'Improved multilevel Monte Carlo convergence using the Milstein scheme', p.343-358 in *Monte Carlo and Quasi-Monte Carlo Methods 2006*, Springer, 2007.

Examples

```
## Not run:
# These are similar to the MLMC tests for the MCQMC06 paper
# using a Milstein discretisation with 2^l timesteps on level l
#
# The figures are slightly different due to:
# -- change in MSE split
# -- change in cost calculation
# -- different random number generation
```

```
# -- switch to S_0=100

M    <- 2 # refinement cost factor
N0   <- 200 # initial samples on coarse levels
Lmin <- 2 # minimum refinement level
Lmax <- 10 # maximum refinement level

test.res <- list()
for(option in 1:5) {
  if(option==1) {
    cat("\n ---- Computing European call ---- \n")
    N      <- 20000 # samples for convergence tests
    L      <- 8 # levels for convergence tests
    Eps    <- c(0.005, 0.01, 0.02, 0.05, 0.1)
  } else if(option==2) {
    cat("\n ---- Computing Asian call ---- \n")
    N      <- 20000 # samples for convergence tests
    L      <- 8 # levels for convergence tests
    Eps    <- c(0.005, 0.01, 0.02, 0.05, 0.1)
  } else if(option==3) {
    cat("\n ---- Computing lookback call ---- \n")
    N      <- 20000 # samples for convergence tests
    L      <- 10 # levels for convergence tests
    Eps    <- c(0.005, 0.01, 0.02, 0.05, 0.1)
  } else if(option==4) {
    cat("\n ---- Computing digital call ---- \n")
    N      <- 200000 # samples for convergence tests
    L      <- 8 # levels for convergence tests
    Eps    <- c(0.01, 0.02, 0.05, 0.1, 0.2)
  } else if(option==5) {
    cat("\n ---- Computing barrier call ---- \n")
    N      <- 200000 # samples for convergence tests
    L      <- 8 # levels for convergence tests
    Eps    <- c(0.005, 0.01, 0.02, 0.05, 0.1)
  }

  test.res[[option]] <- mlmc.test(mcqmc06_l, M, N, L, N0, Eps, Lmin, Lmax, option=option)

  # plot results
  plot(test.res[[option]])
}

## End(Not run)

# The level sampler can be called directly to retrieve the relevant level sums:
mcqmc06_l(l=7, N=10, option=1)
```

`mlmc` *Multi-level Monte Carlo estimation*

Description

This function is the Multi-level Monte Carlo driver which will sample from the levels of user specified function.

Usage

```
mlmc(Lmin, Lmax, N0, eps, mlmc_l, alpha = NA, beta = NA, gamma,
  parallel = NA, ...)
```

Arguments

`Lmin`	the minimum level of refinement. Must be ≥ 2.
`Lmax`	the maximum level of refinement. Must be $\geq$ Lmin.
`N0`	initial number of samples which are used for the first 3 levels and for any subsequent levels which are automatically added. Must be > 0.
`eps`	the target accuracy of the estimate. Must be > 0.
`mlmc_l`	a user supplied function which provides the estimate for level l
`alpha`	the weak error, $O(2^{-alpha*l})$. If `NA` then `alpha` will be estimated.
`beta`	the variance, $O(2^{-beta*l})$. If `NA` then `beta` will be estimated.
`gamma`	the sample cost, $O(2^{gamma*l})$. Must be > 0.
`parallel`	if an integer is supplied, R will fork `parallel` parallel processes an compute each level estimate in parallel.
`...`	additional arguments which are passed on when the user supplied `mlmc_l` function is called

Details

Multilevel Monte Carlo Method method originated in works Giles (2008) and Heinrich (1998).

Consider a sequence $P_0, P_1, \ldots,$ which approximates P_L with increasing accuracy, but also increasing cost, we have the simple identity

$$E[P_L] = E[P_0] + \sum_{l=1}^{L} E[P_l - P_{l-1}],$$

and therefore we can use the following unbiased estimator for $E[P_L]$,

$$N_0^{-1} \sum_{n=1}^{N_0} P_0^{(0,n)} + \sum_{l=1}^{L} \{N_l^{-1} \sum_{n=1}^{N_l} (P_l^{(l,n)} - P_{l-1}^{(l,n)})\}$$

with the inclusion of the level l in the superscript (l, n) indicating that the samples used at each level of correction are independent.

Set C_0, and V_0 to be the cost and variance of one sample of P_0, and C_l, V_l to be the cost and variance of one sample of $P_l - P_{l-1}$, then the overall cost and variance of the multilevel estimator is $\sum_{l=0}^{L} N_l C_l\}$ and $\sum_{l=0}^{L} N_l^{-1} V_l$, respectively.

The idea begind the method, is that provided that the product $V_l C_l$ decreases with l, i.e. the cost increases with level slower than the variance decreases, then one can achieve significant computational savings, which can be formalised as in Theorem 1 of Giles (2015).

For further information on multilevel Monte Carlo methods, see the webpage http://people.maths.ox.ac.uk/gilesm/mlmc_community.html which lists the research groups working in the area, and their main publications.

This function is based on GPL-2 'Matlab' code by Mike Giles.

Value

A list containing:

`P` The MLMC estimate;

`Nl` A vector of the number of samples performed on each level.

Author(s)

Louis Aslett <aslett@stats.ox.ac.uk>

Mike Giles <Mike.Giles@maths.ox.ac.uk>

Tigran Nagapetyan <nagapetyan@stats.ox.ac.uk>

References

M.B. Giles. Multilevel Monte Carlo path simulation. *Operations Research*, 56(3):607-617, 2008.

M.B. Giles. Multilevel Monte Carlo methods. *Acta Numerica*, 24:259-328, 2015.

S. Heinrich. Monte Carlo complexity of global solution of integral equations. *Journal of Complexity*, 14(2):151-175, 1998.

Examples

```
mlmc(2, 6, 1000, 0.01, opre_l, gamma=1, option=1)

mlmc(2, 10, 1000, 0.01, mcqmc06_l, gamma=1, option=1)
```

`mlmc.test` *Multi-level Monte Carlo estimation test suite*

Description

Computes a suite of diagnostic values for an MLMC estimation problem.

Usage

```
mlmc.test(mlmc_l, M, N, L, N0, eps.v, Lmin, Lmax, parallel = NA,
  silent = FALSE, ...)
```

Arguments

mlmc_l	a user supplied function which provides the estimate for level l
M	refinement cost factor (2^γ in the general MLMC Throrem)
N	number of samples to use in the tests
L	number of levels to use in the tests
N0	initial number of samples which are used for the first 3 levels and for any subsequent levels which are automatically added. Must be > 0.
eps.v	a vector of all the target accuracies in the tests. Must all be > 0.
Lmin	the minimum level of refinement. Must be ≥ 2.
Lmax	the maximum level of refinement. Must be $\geq$ Lmin.
parallel	if an integer is supplied, R will fork `parallel` parallel processes an compute each level estimate in parallel.
silent	set to TRUE to supress running output (identical output can still be printed by printing the return result)
...	additional arguments which are passed on when the user supplied `mlmc_l` function is called

Details

See one of the example level sampler functions (e.g. `opre_l`) for example usage.

This function is based on GPL-2 'Matlab' code by Mike Giles.

Value

An `mlmc.test` object which contains all the computed diagnostic values. This object can be printed or plotted (see `plot.mlmc.test`).

Author(s)

Louis Aslett <aslett@stats.ox.ac.uk>

Mike Giles <Mike.Giles@maths.ox.ac.uk>

Tigran Nagapetyan <nagapetyan@stats.ox.ac.uk>

Examples

```
## Not run:
# Example calls with realistic arguments
tst <- mlmc.test(opre_l, M=4, N=2000000,
                 L=5, N0=1000,
                 eps.v=c(0.005, 0.01, 0.02, 0.05, 0.1),
                 Lmin=2, Lmax=6, option=1)
tst
plot(tst)

tst <- mlmc.test(mcqmc06_l, M=2, N=20000,
                 L=8, N0=200,
```

```
                 eps.v=c(0.005, 0.01, 0.02, 0.05, 0.1),
                 Lmin=2, Lmax=10, option=1)
tst
plot(tst)

## End(Not run)

# Toy versions for CRAN tests
tst <- mlmc.test(opre_l, M=4, N=10000,
                 L=5, N0=1000,
                 eps.v=c(0.025, 0.1),
                 Lmin=2, Lmax=6, option=1)

tst <- mlmc.test(mcqmc06_l, M=2, N=10000,
                 L=8, N0=1000,
                 eps.v=c(0.025, 0.1),
                 Lmin=2, Lmax=10, option=1)
```

`opre_l` *Financial options using an Euler-Maruyama discretisation*

Description

Financial options based on scalar geometric Brownian motion and Heston models, similar to Mike Giles' original 2008 Operations Research paper, using an Euler-Maruyama discretisation

Usage

```
opre_l(l, N, option)
```

Arguments

`l`	the level to be simulated.
`N`	the number of samples to be computed.
`option`	the option type, between 1 and 5. The options are: **1 = European call;** **2 = Asian call;** **3 = lookback call;** **4 = digital call;** **5 = Heston model.**

Details

This function is based on GPL-2 'Matlab' code by Mike Giles.

Author(s)

Louis Aslett <aslett@stats.ox.ac.uk>

Mike Giles <Mike.Giles@maths.ox.ac.uk>

Tigran Nagapetyan <nagapetyan@stats.ox.ac.uk>

References

M.B. Giles. Multilevel Monte Carlo path simulation. *Operations Research*, 56(3):607-617, 2008.

Examples

```
## Not run:
# These are similar to the MLMC tests for the original
# 2008 Operations Research paper, using an Euler-Maruyama
# discretisation with 4^l timesteps on level l.
#
# The differences are:
# -- the plots do not have the extrapolation results
# -- two plots are log_2 rather than log_4
# -- the new MLMC driver is a little different
# -- switch to X_0=100 instead of X_0=1

M    <- 4 # refinement cost factor
N0   <- 1000 # initial samples on coarse levels
Lmin <- 2 # minimum refinement level
Lmax <- 6 # maximum refinement level

test.res <- list()
for(option in 1:5) {
  if(option==1) {
    cat("\n ---- Computing European call ---- \n")
    N      <- 2000000 # samples for convergence tests
    L      <- 5 # levels for convergence tests
    Eps    <- c(0.005, 0.01, 0.02, 0.05, 0.1)
  } else if(option==2) {
    cat("\n ---- Computing Asian call ---- \n")
    N      <- 2000000 # samples for convergence tests
    L      <- 5 # levels for convergence tests
    Eps    <- c(0.005, 0.01, 0.02, 0.05, 0.1)
  } else if(option==3) {
    cat("\n ---- Computing lookback call ---- \n")
    N      <- 2000000 # samples for convergence tests
    L      <- 5 # levels for convergence tests
    Eps    <- c(0.01, 0.02, 0.05, 0.1, 0.2)
  } else if(option==4) {
    cat("\n ---- Computing digital call ---- \n")
    N      <- 3000000 # samples for convergence tests
    L      <- 5 # levels for convergence tests
    Eps    <- c(0.02, 0.05, 0.1, 0.2, 0.5)
  } else if(option==5) {
    cat("\n ---- Computing Heston model ---- \n")
```

```
    N      <- 2000000 # samples for convergence tests
    L      <- 5 # levels for convergence tests
    Eps    <- c(0.005, 0.01, 0.02, 0.05, 0.1)
  }

  test.res[[option]] <- mlmc.test(opre_l, M, N, L, N0, Eps, Lmin, Lmax, option=option)

  # print exact analytic value, based on S0=K
  T   <- 1
  r   <- 0.05
  sig <- 0.2
  K   <- 100

  d1  <- (r+0.5*sig^2)*T / (sig*sqrt(T))
  d2  <- (r-0.5*sig^2)*T / (sig*sqrt(T))

  if(option==1) {
    val <- K*( pnorm(d1) - exp(-r*T)*pnorm(d2) )
    cat(sprintf("\n Exact value: %f, MLMC value: %f \n", val, test.res[[option]]$P[1]))
  } else if(option==3) {
    k   <- 0.5*sig^2/r
    val <- K*( pnorm(d1) - pnorm(-d1)*k - exp(-r*T)*(pnorm(d2) - pnorm(d2)*k) )
    cat(sprintf("\n Exact value: %f, MLMC value: %f \n", val, test.res[[option]]$P[1]))
  } else if(option==4) {
    val <- K*exp(-r*T)*pnorm(d2)
    cat(sprintf("\n Exact value: %f, MLMC value: %f \n", val, test.res[[option]]$P[1]))
  }

  # plot results
  plot(test.res[[option]])
}

## End(Not run)

# The level sampler can be called directly to retrieve the relevant level sums:
opre_l(l=7, N=10, option=1)
```

`plot.mlmc.test` *Plot an* `mlmc.test` *object*

Description

Produces diagnostic plots on the result of an `mlmc.test` function call.

Usage

```
## S3 method for class 'mlmc.test'
plot(x, which = "all", cols = NA, ...)
```

Arguments

x	an mlmc.test object as produced by a call to the mlmc.test function.
which	a vector of strings specifying which plots to produce, or "all" to do all diagnostic plots. The options are: "var" = log_2 **of variance against level;** "mean" = log_2 **of mean against level;** "consis" = **consistency against level;** "kurt" = **kurtosis against level;** "Nl" = log_2 **of number of samples against level;** "cost" = log_10 **of cost against** log_10 **of epsilon (accuracy).**
cols	the number of columns across to plot to override the default value.
...	additional arguments which are passed on to plotting functions.

Author(s)

Louis Aslett <aslett@stats.ox.ac.uk>

Examples

```
## Not run:
tst <- mlmc.test(opre_l, M=4, N=2000000,
                 L=5, N0=1000,
                 eps.v=c(0.005, 0.01, 0.02, 0.05, 0.1),
                 Lmin=2, Lmax=6, option=1)
tst
plot(tst)

## End(Not run)
```

Index