Package ‘mlmc’

November 11, 2024

Type Package

Title Multi-Level Monte Carlo

Version 2.1.1

Maintainer Louis Aslett <louis.aslett@durham.ac.uk>

Description An implementation of MLMC (Multi-Level Monte Carlo), Giles (2008) <doi:10.1287/opre.1070.0496>, Heinrich (1998) <doi:10.1006/jcom.1998.0471>, 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.

BugReports https://github.com/louisaslett/mlmc/issues

URL https://mlmc.louisaslett.com/, https://github.com/louisaslett/mlmc

Imports ggplot2, grid, parallel, Rcpp

License GPL-2

RoxygenNote 7.3.2

Encoding UTF-8

LinkingTo Rcpp

NeedsCompilation yes

Author Louis Aslett [cre, aut, trl] (<https://orcid.org/0000-0003-2211-233X>), Mike Giles [ctb] (<https://orcid.org/0000-0002-5445-3721>), Tigran Nagapetyan [ctb] (<https://orcid.org/0000-0002-0379-1157>), Sebastian Vollmer [ctb] (<https://orcid.org/0000-0003-2831-1401>)

Repository CRAN

Date/Publication 2024-11-11 18:30:01 UTC

Contents

`mcqmc06_l` *Financial options using a Milstein discretisation*

Description

Financial options based on scalar geometric Brownian motion, similar to Mike Giles' MCQMC06 paper, Giles (2008), 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.

Value

A named list containing:

`sums` is a vector of length six $\left(\sum Y_i, \sum Y_i^2, \sum Y_i^3, \sum Y_i^4, \sum X_i, \sum X_i^2\right)$ where Y_i are iid simulations with expectation $E[P_0]$ when $l = 0$ and expectation $E[P_l - P_{l-1}]$ when $l > 0$, and X_i are iid simulations with expectation $E[P_l]$. Note that only the first two components of this are used by the main `mlmc()` driver, the full vector is used by `mlmc.test()` for convergence tests etc;

`cost` is a scalar with the total cost of the paths simulated, computed as $N \times 2^l$ for level l.

Author(s)

Louis Aslett <louis.aslett@durham.ac.uk>

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

References

Giles, M. (2008) 'Improved Multilevel Monte Carlo Convergence using the Milstein Scheme', in A. Keller, S. Heinrich, and H. Niederreiter (eds) *Monte Carlo and Quasi-Monte Carlo Methods 2006*. Berlin, Heidelberg: Springer, pp. 343–358. Available at: doi:10.1007/9783540744962_20.

Examples

```
# 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
#
# Note the following takes quite a while to run, for a toy example see after
# this block.

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, 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
  B   <- 0.85*K

  k   <- 0.5*sig^2/r;
  d1  <- (r+0.5*sig^2)*T / (sig*sqrt(T))
  d2  <- (r-0.5*sig^2)*T / (sig*sqrt(T))
  d3  <- (2*log(B/K) + (r+0.5*sig^2)*T) / (sig*sqrt(T))
  d4  <- (2*log(B/K) + (r-0.5*sig^2)*T) / (sig*sqrt(T))

  if(option == 1) {
    val <- K*( pnorm(d1) - exp(-r*T)*pnorm(d2) )
  } else if(option == 2) {
    val <- NA
  } else if(option == 3) {
    val <- K*( pnorm(d1) - pnorm(-d1)*k - exp(-r*T)*(pnorm(d2) - pnorm(d2)*k) )
  } else if(option == 4) {
    val <- K*exp(-r*T)*pnorm(d2)
  } else if(option == 5) {
    val <- K*(                             pnorm(d1) - exp(-r*T)*pnorm(d2) -
            ((K/B)^(1-1/k))*((B^2)/(K^2)*pnorm(d3) - exp(-r*T)*pnorm(d4)) )
  }

  if(is.na(val)) {
    cat(sprintf("\n Exact value unknown, MLMC value: %f \n", test.res[[option]]$P[1]))
  } else {
    cat(sprintf("\n Exact value: %f, MLMC value: %f \n", val, test.res[[option]]$P[1]))
  }

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


# 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 = NA,
  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 (root mean square error). Must be > 0.

`mlmc_l` a user supplied function which provides the estimate for level l. It must take at least two arguments, the first is the level number to be simulated and the second the number of paths. Additional arguments can be taken if desired: all additional `...` arguments to this function are forwarded to the user defined `mlmc_l` function.

The user supplied function should return a named list containing one element named `sums` and second named `cost`, where:

`sums` is a vector of length at least two. The first two elements should be $\left(\sum Y_i, \sum Y_i^2\right)$ where Y_i are iid simulations with expectation $E[P_0]$ when $l = 0$ and expectation $E[P_l - P_{l-1}]$ when $l > 0$. Note that typically the user supplied level sampler will actually return a vector of length six, also enabling use of the `mlmc.test()` function to perform convergence tests, kurtosis, and telescoping sum checks. See `mlmc.test()` for the definition of these remaining four elements.

`cost` is a scalar with the total cost of the paths simulated. For example, in the financial options samplers included in this package, this is calculated as NM^l, where N is the number of paths requested in the call to the user function `mlmc_l`, M is the refinement cost factor ($M = 2$ for `mcqmc06_l()` and $M = 4$ for `opre_l()`), and l is the level being sampled.

See the function (and source code of) `opre_l()` and `mcqmc06_l()` in this package for an example of user supplied level samplers.

`alpha` the weak error, $O(2^{-\alpha l})$. Must be > 0 if specified. If `NA` then `alpha` will be estimated.

`beta` the variance, $O(2^{-\beta l})$. Must be > 0 if specified. If `NA` then `beta` will be estimated.

gamma	the sample cost, $O(2^{\gamma l})$. Must be > 0 if specified. If NA then gamma will be estimated.
parallel	if an integer is supplied, R will fork parallel parallel processes and spread the simulations required at each level as evenly as possible across all cores.
...	additional arguments which are passed on when the user supplied mlmc_l function is called.

Details

The Multilevel Monte Carlo Method method originated in the 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 $E[P_L]$,

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

where N_l samples are produced at level l. The inclusion of the level l in the superscript (l, n) indicates 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 behind 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 https://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 named list containing:

P The MLMC estimate;

Nl A vector of the number of samples performed on each level;

Cl Per sample cost at each level.

Author(s)

Louis Aslett <louis.aslett@durham.ac.uk>

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

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

References

Giles, M.B. (2008) 'Multilevel Monte Carlo Path Simulation', *Operations Research*, 56(3), pp. 607–617. Available at: doi:10.1287/opre.1070.0496.

Giles, M.B. (2015) 'Multilevel Monte Carlo methods', *Acta Numerica*, 24, pp. 259–328. Available at: doi:10.1017/S096249291500001X.

Heinrich, S. (1998) 'Monte Carlo Complexity of Global Solution of Integral Equations', *Journal of Complexity*, 14(2), pp. 151–175. Available at: doi:10.1006/jcom.1998.0471.

Examples

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

mlmc(2, 10, 1000, 0.01, mcqmc06_l, 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,
  N,
  L,
  N0,
  eps.v,
  Lmin,
  Lmax,
  alpha = NA,
  beta = NA,
  gamma = NA,
  parallel = NA,
  silent = FALSE,
  ...
)
```

Arguments

`mlmc_l`	a user supplied function which provides the estimate for level l. It must take at least two arguments, the first is the level number to be simulated and the second the number of paths. Additional arguments can be taken if desired: all additional `...` arguments to this function are forwarded to the user defined `mlmc_l` function.

The user supplied function should return a named list containing one element named sums and second named cost, where:

sums is a vector of length six $\left(\sum Y_i, \sum Y_i^2, \sum Y_i^3, \sum Y_i^4, \sum X_i, \sum X_i^2\right)$ where Y_i are iid simulations with expectation $E[P_0]$ when $l = 0$ and expectation $E[P_l - P_{l-1}]$ when $l > 0$, and X_i are iid simulations with expectation $E[P_l]$. Note that this differs from the main mlmc() driver, which only requires the first two of these elements in order to calculate the estimate. The remaining elements are required by mlmc.test() since they are used for convergence tests, kurtosis, and telescoping sum checks.

cost is a scalar with the total cost of the paths simulated. For example, in the financial options samplers included in this package, this is calculated as NM^l, where N is the number of paths requested in the call to the user function mlmc_l, M is the refinement cost factor ($M = 2$ for mcqmc06_l() and $M = 4$ for opre_l()), and l is the level being sampled.

See the function (and source code of) opre_l() and mcqmc06_l() in this package for an example of user supplied level samplers.

N — number of samples to use in convergence tests, kurtosis, telescoping sum check.

L — number of levels to use in convergence tests, kurtosis, telescoping sum check.

N0 — initial number of samples which are used for the first 3 levels and for any subsequent levels which are automatically added in the complexity tests. Must be > 0.

eps.v — a vector of one or more target accuracies for the complexity tests. Must all be > 0.

Lmin — the minimum level of refinement for complexity tests. Must be ≥ 2.

Lmax — the maximum level of refinement for complexity tests. Must be $\geq$ Lmin.

alpha — the weak error, $O(2^{-\alpha l})$. Must be > 0 if specified. If NA then alpha will be estimated.

beta — the variance, $O(2^{-\beta l})$. Must be > 0 if specified. If NA then beta will be estimated.

gamma — the sample cost, $O(2^{\gamma l})$. Must be > 0 if specified. If NA then gamma will be estimated.

parallel — if an integer is supplied, R will fork parallel parallel processes. This is done for the convergence tests section by splitting the N samples as evenly as possible across cores when sampling each level. This is also done for the MLMC complexity tests by passing the parallel argument on to the mlmc() driver when targeting each accuracy level in eps.

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 <louis.aslett@durham.ac.uk>

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

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

Examples

```
# Example calls with realistic arguments
# Financial options using an Euler-Maruyama discretisation
tst <- mlmc.test(opre_l, 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)

# Financial options using a Milstein discretisation
tst <- mlmc.test(mcqmc06_l, 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)


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

tst <- mlmc.test(mcqmc06_l, 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, Giles (2008), 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.

Value

A named list containing:

`sums` is a vector of length six $\left(\sum Y_i, \sum Y_i^2, \sum Y_i^3, \sum Y_i^4, \sum X_i, \sum X_i^2\right)$ where Y_i are iid simulations with expectation $E[P_0]$ when $l = 0$ and expectation $E[P_l - P_{l-1}]$ when $l > 0$, and X_i are iid simulations with expectation $E[P_l]$. Note that only the first two components of this are used by the main `mlmc()` driver, the full vector is used by `mlmc.test()` for convergence tests etc;

`cost` is a scalar with the total cost of the paths simulated, computed as $N \times 4^l$ for level l.

Author(s)

Louis Aslett <louis.aslett@durham.ac.uk>

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

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

References

Giles, M.B. (2008) 'Multilevel Monte Carlo Path Simulation', *Operations Research*, 56(3), pp. 607–617. Available at: doi:10.1287/opre.1070.0496.

Examples

```
# 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
#
# Note the following takes quite a while to run, for a toy example see after
# this block.

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      <- 1000000 # 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      <- 1000000 # 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      <- 1000000 # 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      <- 4000000 # 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, 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

  k   <- 0.5*sig^2/r;
  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) )
  } else if(option == 2) {
    val <- NA
  } else if(option == 3) {
    val <- K*( pnorm(d1) - pnorm(-d1)*k - exp(-r*T)*(pnorm(d2) - pnorm(d2)*k) )
  } else if(option == 4) {
    val <- K*exp(-r*T)*pnorm(d2)
  } else if(option == 5) {
    val <- NA
  }

  if(is.na(val)) {
    cat(sprintf("\n Exact value unknown, MLMC value: %f \n", test.res[[option]]$P[1]))
  } else {
    cat(sprintf("\n Exact value: %f, MLMC value: %f \n", val, test.res[[option]]$P[1]))
  }

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


# 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 the absolute value of the 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.

Details

Most of the plots produced are relatively self-explanatory. However, the consistency and kurtosis plots in particular may require some background. It is highly recommended to refer to Section 3.3 of Giles (2015), where the rationale for these diagnostic plots is addressed in full detail.

Value

No return value, called for side effects.

Author(s)

Louis Aslett <louis.aslett@durham.ac.uk>

References

Giles, M.B. (2015) 'Multilevel Monte Carlo methods', *Acta Numerica*, 24, pp. 259–328. Available at: doi:10.1017/S096249291500001X.

Examples

```
tst <- mlmc.test(opre_l, 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)
```

Index