MATH-414: STOCHASTIC SIMULATION

Group Project:

Multi-Level Monte Carlo Markov Chain methods for Option pricing

---

Emanuele SORGENTE, Raffaele ANCAROLA

11TH JANUARY 2023

EPFL

1 Introduction

This project is based on the paper "Multilevel Monte Carlo Methods" by Giles [1] which explores the field of Monte Carlo methods and their application to numerical integration and optimization. Monte Carlo methods are a class of statistical techniques that use random sampling to approximate complex computations or estimations that are otherwise difficult to solve analytically. They are widely used in a variety of fields, including physics, finance, and engineering.

In this coursework, we will first introduce the concept of multi-level Monte Carlo (*MLMC*) methods, which are a variant of traditional Monte Carlo methods that aim to reduce the computational cost of numerical integration by taking advantage of the hierarchical structure of the problem being solved. *MLMC* methods work by dividing the problem into a hierarchy of levels, each of which can be solved independently using Monte Carlo techniques. This allows for a more efficient use of computational resources, as the number of samples required at each level can be adjusted based on the relative error and variance at that level. Finally, *MLMC* methods will be compared to the traditional Monte Carlo in terms of efficiency and *mean square error* (MSE) reduction.

2 Data and Methodology

Let's consider the underlying Stochastic Differential Equation describing the time evolution of the *option pricing*:

$$\begin{cases} dS(t) = rS(t)dt + \sigma S(t)dW_t \quad t \in [0, T] \\ S(0) = 1 \end{cases} \tag{1}$$

.

Here dW_t refers to the differential of a standard *Wiener process* W_t. We further define

$$\mu := \mathbb{E}[Y] := \mathbb{E}[f(S)] \tag{2}$$

for some given real functional f applying to a stochastic process. In our case of examination, μ will be the expected payoff at the final time $T = 1$, which will be the computational goal of this homework.

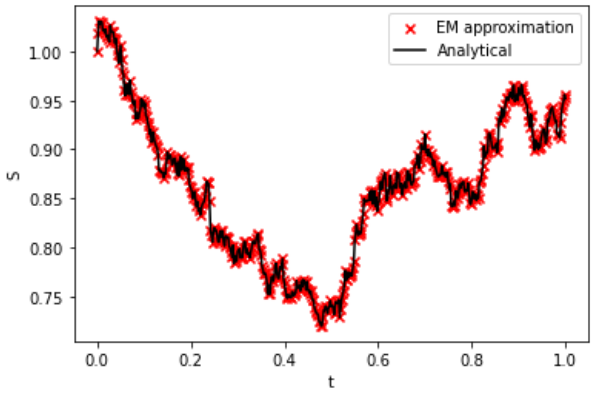


Figure 1: Realization of a EM approximated solution (eq. 4) compared to the analytical solution for the system in equation 1.

In order to simplify the experiment, we will fix the values:

- The drift $r = 0.05$
- The volatility $\sigma = 0.2$

This system admits also an analytical solution as a log-normal stochastic process, which makes it easier to verify any other numerical solutions,

$$S(t) = \exp\left((r - \frac{\sigma^2}{2})t + \sigma W_t\right) \tag{3}$$

.

2.1 Time finite difference discretization

The stochastic differential equation defined in eq. 1 can be discretized into a uniform grid $t_m = mh$ with $m = 1, \dots, M \in \mathbb{N}$, $T = Mh$, $h > 0$. The first order approximation of the time derivative gives rise to the *Euler-Maruyama* time discretization,

$$\begin{cases} S_{m+1} = (1 + rh + \sigma \Delta W_m) \cdot S_m \quad m = 1, \dots, M \\ S_0 = 1 \end{cases} \tag{4}$$

where $\Delta W_m \sim \mathcal{N}(0, h)$ is a h increment of the Wiener process W_t. The rate of convergence for this approximation is well known to be $|\mathbb{E}[S_m] - \mathbb{E}[S(mh)]| = \mathcal{O}(t_m h)$ for the weak error and $\mathbb{E}[|S_m - S(mh)|^p]^{\frac{1}{p}} = \mathcal{O}(t_m h^{\frac{1}{2}})$ for the strong error [2].

2.2 Multi-level Monte Carlo Algorithm

Now we want to build a Monte Carlo estimator of the expected payoff defined in equation 2 which efficiently computes the best approximation for any given function f. The special case we will consider in the next sections are going to be the *Asian* option (eq. 13) and the *Barrier call* option (eq. 26). Let $\ell = 1, \dots, L \in \mathbb{N}$ denote the levels of simulation and Y_ℓ, then the expectation can be decomposed in telescopic sum over different levels,

$$\mu = \mathbb{E}[Y_L] = \mathbb{E}[Y_0] + \sum_{\ell=1}^{L} \mathbb{E}[Y_\ell - Y_{\ell-1}] = \sum_{\ell=0}^{L} \mathbb{E}[Y_\ell - Y_{\ell-1}] \tag{5}$$

with $Y_{-1} = 0$ by convention. Now approximate each expectation by a Monte Carlo estimator using N_ℓ samples for each independent level. This is called the multi-level Monte Carlo estimator and expresses as,

$$\hat{\mu}_{MLMC} := \sum_{\ell=0}^{L} \hat{\mu}_\ell = \sum_{\ell=0}^{L} \frac{1}{N_\ell} \sum_{n=0}^{N_\ell} (Y_\ell^{(n,\ell)} - Y_{\ell-1}^{(n,\ell)}) \tag{6}$$

with $\hat{\mu}_\ell$ is independent from $\hat{\mu}_k$ for $k \neq \ell$. Furthermore, in order to minimize the variance of each level, $Y_\ell^{(n,\ell)}$ and $Y_{\ell-1}^{(n,\ell)}$ must be simulated within the same underlying noise. To do so, we will assume a discretization setup which decays exponentially as the level increases, i.e. fixing the first level discretization to h_0, the others will be $h_\ell = h_0 \cdot 2^{-\gamma \ell}$. For the sake of simplicity we will take $\gamma = 1$.

Joint Wiener process generation With this assumption, we can generate the same discretization grid which fits both for a level ℓ and its ancestor $\ell - 1$. In particular, the generated noise is first sampled from a Wiener in the fine grained grid and then time differences $\Delta W_{\ell,m}^{(n,\ell)}$ are taken, with m the time discretization index of the fine level. As those time differences are zero-mean normal distributed with variance h, then the coarser level's differences $\Delta W_{\ell-1,k}^{(n,\ell)}$, indexed by k, have variance $2h$. Assuming further both share the same points, i.e. $\Delta W_{\ell-1,k}^{(n,\ell)} = \Delta W_{\ell,m}^{(n,\ell)}$ for all $k = 2m$, then one finds an expression for the coarser level conditioned on the differences of the finer.

$$\Delta W_{\ell-1,k}^{(n,\ell)} = \Delta W_{\ell,m}^{(n,\ell)} + \Delta W_{\ell,m+1}^{(n,\ell)}, \quad \forall\, m = 2k \tag{7}$$

MLMC Algorithm Finally, the algorithm for the Multi-level Monte Carlo is shown below:

Inputs: time steps $\{h_\ell\}_{\ell=0}^{L}$ and samples per level $\{N_\ell\}_{\ell=0}^{L}$

- For $\ell = 0, ..., L$:
  - For $n = 1, \ldots, N_\ell$:
    - Generate independent samples $\Delta W_{\ell,m}^{(n,\ell)} \sim \mathcal{N}(0, h_\ell)$ for $m = 0, \ldots, (M_\ell - 1)$
    - Generate $\Delta W_{\ell-1,k}^{(n,\ell)}$ for $k = 0, \ldots, (M_{\ell-1} - 1)$ as shown in equation 7.
    - Simulate $S_\ell^{(n,\ell)}$ using EM with $h = h_\ell$, $\Delta\{W_{\ell,m}^{(n,\ell)}\}_{m=0}^{M_\ell - 1}$ as noise.
    - Compute $Y_\ell^{(n,\ell)} = f(S_\ell^{(n,\ell)})$
    - If $\ell > 0$:
      - Simulate $S_{\ell-1}^{(n,\ell)}$ using EM with $h = h_{\ell-1}$, $\{\Delta W_{\ell-1,k}^{(n,\ell)}\}_{k=0}^{M_{\ell-1}-1}$ as noise.
      - Compute $Y_{\ell-1}^{(n,\ell)} = f(S_{\ell-1}^{(n,\ell)})$
    - Else, $Y_{-1}^{(n,\ell)} = 0$
  - Compute $\hat{\mu}_\ell = \frac{1}{N_\ell} \sum_{n=1}^{N_\ell} (Y_\ell^{(n,\ell)} - Y_{\ell-1}^{(n,\ell)})$
- Return $\hat{\mu}_L^{MLMC} = \sum_{\ell=0}^{L} \hat{\mu}_\ell$

2.3 Optimal sampling per level analysis (a)

Let's denote C_{MLMC} the overall cost of a run and V_{MLMC} the variance achieved of the estimator $\hat{\mu}_{MLMC}$. Furthermore, let's denote and N_ℓ the necessary number of samples which estimates $\mathbb{E}[Y_\ell - Y_{\ell-1}, C_\ell]$ the cost of the estimation procedure and V_ℓ the variance of the estimator. Considering that for each level ℓ, $\hat{\mu}_\ell$ was previously defined as a Monte Carlo estimator using N_ℓ samples (eq. 6), the overall cost and variance express as,

$$C_{MLMC} = \sum_{\ell=0}^{L} N_\ell C_\ell \qquad\qquad V_{MLMC} = \sum_{\ell=0}^{L} \frac{V_\ell}{N_\ell} \tag{8}$$

Our aim is to find the optimal number of samples per level N_ℓ such that the overall cost is fixed and the variance is minimized at the same time. Hence, in the *Lagrange multipliers* formalism this reduces to find a constant $\lambda \in \mathbb{R}$ such that the following *Lagrangian* expression is minimized,

$$\mathcal{L}(N_1, \ldots, N_L) = \sum_{\ell=0}^{L} \frac{V_\ell}{N_\ell} + \lambda \left(\sum_{\ell=0}^{L} N_\ell C_\ell - C_\varepsilon \right) \tag{9}$$

where C_ε is the fixed cost. By setting the gradient of the *Lagrangian* to zero to find the minimum, one finds that the optimal number of samples per level is related to λ by,

$$N_\ell^* = \frac{1}{\sqrt{\lambda}} \sqrt{\frac{V_\ell}{C_\ell}}, \quad \text{for } \ell = 0, \ldots, L \tag{10}$$

.

Injecting this result into the overall variance expression one finds the optimal one at fixed cost, which can be bounded by a positive factor ε^2 representing an aimed fixed precision for all levels.

$$\forall\, l = 1, \ldots, L \;:\; V^*_{MLMC} = \frac{1}{N^*_\ell}\sqrt{\frac{V_\ell}{C_\ell}}\left(\sum_{\ell=0}^{L}\sqrt{V_\ell C_\ell}\right) = \varepsilon^2 \tag{11}$$

.

Hence, by taking N^*_ℓ as natural number, one finds its optimum,

$$N^*_\ell = \left\lceil \varepsilon^{-2}\sqrt{\frac{V_\ell}{C_\ell}}\left(\sum_{\ell=0}^{L}\sqrt{V_\ell C_\ell}\right)\right\rceil \tag{12}$$

.

3 Simulations and specific studies

3.1 Classical Monte Carlo simulation (b)

In this section we aim to estimate $\mathbb{E}[Y]$ with a classical Monte Carlo simulator combined with *Euler Maruyama (EM)* (equation 4) approximation. We will take f as the *Asian Option*, defined as follow:

$$Y = \exp(-r)\cdot\max(0, \bar{S} - K), \quad , \bar{S} = \int_0^T S(t)dt, \quad K > 0 \tag{13}$$

.

In order to simplify the experiment, we will fix $K = 1$. Because *EM* involves a discretization $\{S_m\}_{m=1}^M$, where $T = M \cdot h$ and h the discretization step, one needs to approximate the integral in eq. 13 by the aid of the trapezoidal rule for numerical integration , already implemented as numpy function [3].

Bias and Variance convergence study The variance $\mathsf{Var}[\hat{\mu}_{MC}]$ is known to scale with a factor $\mathcal{O}(1/N)$ w.r.t the number of samples because it's a *Monte Carlo* estimator and, using the strong error argument about *EM*, we expect that exists $\beta \geq 1$ s.t. the variance is bounded by

$$\mathsf{Var}[\hat{\mu}_{MC}] = \frac{\mathsf{Var}[Y_h]}{N} = \mathcal{O}\left(\frac{h^\beta}{N}\right) \tag{14}$$

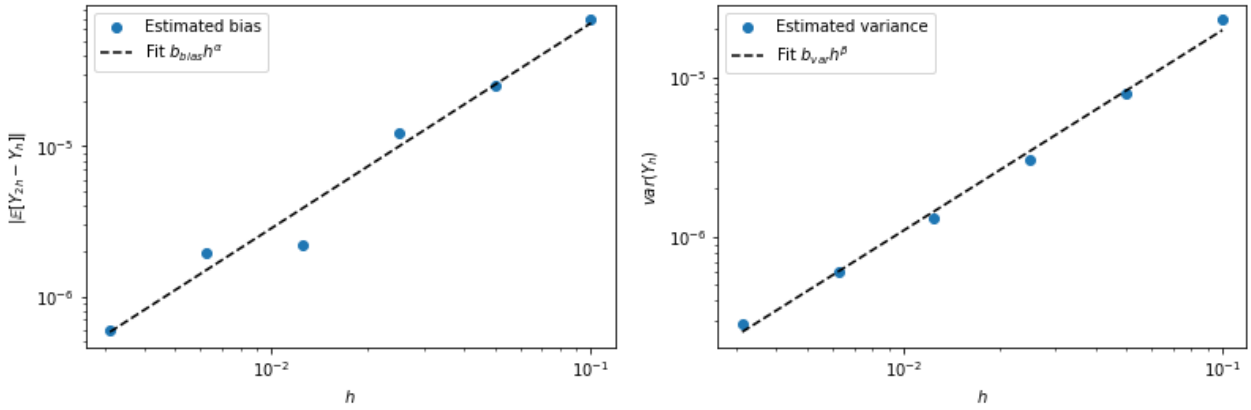
.

The β factor is introduced in order to correct convergence rate drift given by the integration of stochastic process' realizations S_m, each of different accuracies. Assume the variance of S_m increases with m and the integral operation in the Asian option average all of those values, then the resulting accuracy will be bounded between $\mathcal{O}(h^2)$ of S_1 and $\mathcal{O}(h)$ of S_M.

Analogously, the bias is defined as the distance in expectation from the analytical solution, which is only due to the *EM* approximation as a Monte Carlo expectation estimator is unbiased. We call its distinct polynomial coefficient α and its defined as,

$$|\mathbb{E}[\hat{\mu}_{MC}] - \mathbb{E}[Y]| = \mathcal{O}(h^\alpha) \tag{15}$$

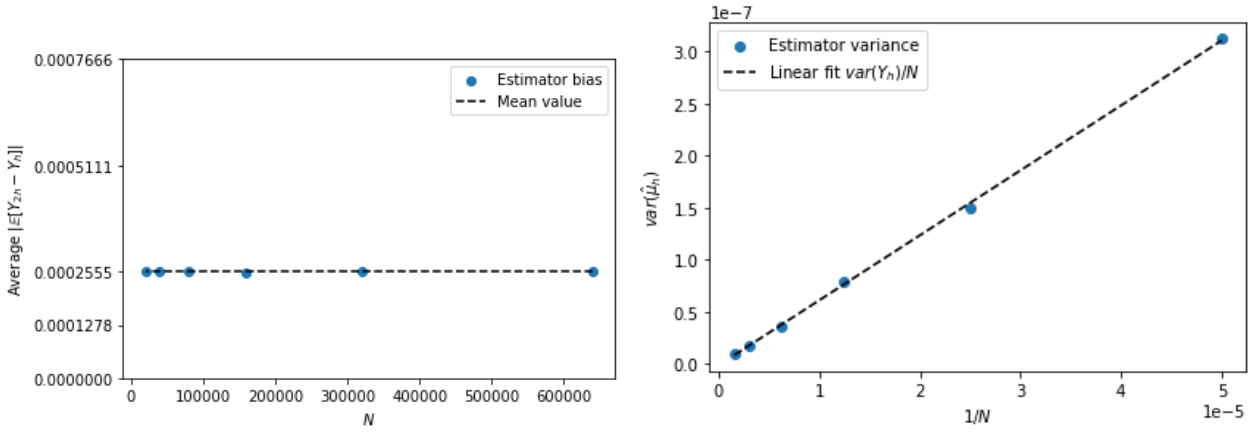
Although we would expect analytical values $\beta = 1$ and $\alpha = 1$ with all S_m at the same accuracy, we observe in figure 2 that the strong convergence rate for the Asian option is attended with $\beta \approx 1.25$ and the weak rate



(a) Estimated bias, $\alpha = 1.36 \pm 0.24$ (b) Estimated variance, $\beta = 1.25 \pm 0.01$

Figure 2: Behaviour of bias and variance varying time step h at fixed number of samples $N = 500000$.

is $\alpha \approx 1.36$. The number of samples N was chosen large enough to sufficiently reduce the variance of those estimations, which ensures that experimental values are most likely to be greater than 1 and this confirms the hypothesis of the error drift and that a correction is to be taken into consideration for the next points.



(a) Estimated bias, not varying (b) Estimated variance, proportional to $1/N$

Figure 3: Behaviour of bias and variance varying number of samples N at fixed time step $h = 0.1$.

Furthermore, one sees from figure 3 that only the variance have a clear dependency on the number of samples. This was predicted from equations 14 and 15. In this the values found are $(2.55 \pm 0.01) \cdot 10^{-4}$ and $(6.2 \pm 0.4) \cdot 10^{-3}$ for the bias and the variance of Y_h respectively.

MSE analysis The *MSE* of $\hat{\mu}_{MC}^{(h)}$ can be expressed in terms of variance and bias of the estimator,

$$MSE[\hat{\mu}_{MC}^{(h)}] = \mathrm{Var}[\hat{\mu}_{MC}^{(h)}] + |\mathbb{E}[Y_h] - \mathbb{E}[Y]|^2 \leq \frac{\mathrm{Var}[Y_h]}{N} + (\hat{b}_{MC}^{(h)})^2, \quad \hat{b}_{MC}^{(h)} = |\mathbb{E}[Y_h] - \mathbb{E}[Y_{2h}]| \tag{16}$$

where $\hat{b}_{MC}^{(h)}$ is a bias surrogate introduced with the purpose of upper bound the real bias. A simple adaptive

algorithm bounding the *MSE* up to a factor $2\varepsilon^2$ can be:

Inputs: initial guess time step h_0, target accuracy ε, pilot number of samples N_{pilot}.

1. Set $h = h_0$.
2. Execute a pilot run of $\hat{\mu}_{MC}^{(h)}$ and $\hat{\mu}_{MC}^{(2h)}$ with a given number of samples N_{pilot}, both runs with the same samples.
3. Estimate variance Var$[Y_h]$ and bias $\hat{b}_{MC}^{(h)}$ using the pilot run samples.
4. If $\hat{b}_{MC}^{(h)} \geq \varepsilon$, then halve h and repeat from (2.), otherwise continue.
5. Set $N = \lceil \text{Var}[Y_h]/\varepsilon^2 \rceil$
6. Run actual simulation of $\hat{\mu}_{MC}^{(h)}$ with N samples.

Cost evaluation Since we choose h s.t. $\hat{b}_{MC}^{(h)} < \varepsilon$ and figure 2 shows that the bias varies in h with an exponent factor α, then we find the optimal number of halving operations to be $L \propto \frac{1}{\alpha}\log_2(\frac{1}{\varepsilon})$. The cost of simulation is proportional to the inverse of h, which means $C_L \propto 2^L$. Finally, lower bounding N to Var$[Y_h]/\varepsilon^2$ one finds the total cost,

$$C_{MC} = N \cdot C_L = \mathcal{O}\left(\frac{1}{\varepsilon^{2+\frac{1}{\alpha}}}\right) \tag{17}$$

3.2 Two level MLMC (c)

Focusing on a 2-level system, we aim to check how it performs compared to a crude Monte Carlo simulation. For instance, we take a setup where $N_0 = 10^5$, $h_0 = 0.2$ and $h_1 = h_0/2 = 0.1$. We can further suppose that $C_1 = 2C_0$ because the cost increases linearly in the number of discretization steps, i.e. $C_0 = \mathcal{O}(T/h_0)$.

Variance reduction at equivalent cost From equation 12 and using the ratio between the costs, it follows directly that the optimal ratio N_1/N_0 is given by,

$$\frac{N_1}{N_0} = \sqrt{\frac{V_1 C_0}{V_0 C_1}} = \sqrt{\frac{V_1}{2V_0}} \tag{18}$$

where $V_0 = \text{Var}[Y_0]$ and $V_1 = \text{Var}[Y_1 - Y_0]$.

We aim to estimate the variance reduction obtained by the 2-level *MLMC* compared to a crude Monte Carlo simulation, both taking an equivalent cost. The total cost of a 2-level *MLMC* is $C_{MLML}^{2-level} = N_0 C_0 + N_1 C_1$ and a crude Monte Carlo approach running N_{MC} samples with $h = h_1$ would have a cost of $C_{MC}^{(h_1)} = N_{MC} C_1$. It follows by taking the equivalent over the two cost expression that,

$$N_{MC} = \left\lceil \frac{N_0}{2} + N_1 \right\rceil = \left\lceil \frac{N_0}{2}\left(1 + \sqrt{\frac{2V_1}{V_0}}\right) \right\rceil \tag{19}$$

.

Hence, given $\tilde{V}_1 := \text{Var}[Y_1]$, it's straight forward to estimate the variances of each estimator and, hence, an estimator for the estimators variance ratio quantifying the variance reduction:

$$\begin{cases} V_{MC}^{(h_1)} = \frac{\tilde{V}_1}{N_{MC}} \approx \frac{2\tilde{V}_1}{N_0\left(1+\sqrt{\frac{2V_1}{V_0}}\right)} \\ V_{MLMC}^{2-level} = \frac{V_0}{N_0} + \frac{V_1}{N_1} \approx \frac{V_0}{N_0}\left(1+\sqrt{\frac{2V_1}{V_0}}\right) \end{cases} \implies \frac{V_{MLMC}^{2-level}}{V_{MC}^{(h_1)}} = \frac{V_0}{2\tilde{V}_1}\left(1+\sqrt{\frac{2V_1}{V_0}}\right)^2 \tag{20}$$

.

Bias comparison Followed by the expectation expression of *MLMC* (eq. 5), we obtain the same bias for both estimators:

$$|\mathbb{E}[\hat{\mu}_{MLMC}^{2-level}] - \mathbb{E}[Y]| = |\mathbb{E}[Y_1] - \mathbb{E}[Y]| = |\mathbb{E}[Y_{h_1}] - E[Y]| = |\mathbb{E}[\hat{\mu}_{MC}^{(h_1)}] - \mathbb{E}[Y]| \tag{21}$$

.

Hence, we don't expect different biases in practical terms.

Computational procedure and results A pilot run with $N_1 = N_0$ is executed first in order to estimate the variances V_0 and V_1 using an unbiased Monte Carlo estimation given a set of independent samples for each level. Then using eq. 18 we re-calibrated the value of N_1 in order to optimise the cost. Then, constructing a crude Monte Carlo estimator at the same cost as optimized *MLMC* (eq. 19), we evaluated the ratio of variances. All results are shown in table 1. The most interesting results are mainly of ratios, where we have a huge gap between the variance V_0 and V_1. This is expected because the order of magnitude of Y_0 is much larger than $Y_1 - Y_0$. Thus, the computational cost necessary to attend any target accuracy is much lower on the second step. In the overall variance reduction, consider that $V_0 \approx \tilde{V}_1$ and $V_1 \ll V_0$, then eq. 20 suggests that the variance ratio is approximately a half. We find in fact, that the ratio is slightly larger than $1/2$. So, in order get a real benefit, one needs to increase the number of levels.

Pilot run, N_1/N_0 ratio estimation		
V_0 [10^{-3}]	V_1 [10^{-3}]	Optimal N_1/N_0 ratio
5.89	0.0728	0.0785
Optimal run, variance reduction		
$V_{MLMC}^{2-levels}$ [10^{-3}]	$V_{MC}^{2-levels}$ [10^{-3}]	$V_{MLMC}^{2-levels}/V_M C^{2-levels}$
3.95	6.02	0.656

Table 1: Optimal number of samples per level estimation and variance reduction; the system setup is the same as the previous point.

3.3 Multi Level simulation with MSE bound (d)

In this section the purpose is to find the best configuration of a *MLMC* setup based on a pilot run. In particular, we will bound the MSE (eq. 22) to a factor $2\varepsilon^2$. Differently from the previous section, the comparison criterion with a crude *MC* simulation will be the equivalent MSE bound, rather than an equivalent cost.

Simulation setup Here the Asian option (eq. 13) is still used as functional of the solution, for each level $\ell = 0, \ldots, L$ the time step is fixed to an exponential decay $h_\ell = h_0 \cdot 2^{-\ell}$ so that the cost follows an exponential growth $C_\ell = C_0 \cdot 2^\ell$. While h_0 is user defined, we aim to find the optimal configuration of L and $\{N_\ell\}_{\ell=1}^L$.

MSE bound for optimal In order to bound MSE, it suffices to bound both variance and squared bias up to a given threshold ε^2 because MSE is defined as the sum of the two parts:

$$MSE[\hat{\mu}^{(L)}_{MLMC}] = \mathbb{E}[(\hat{\mu}^{(L)}_{MLMC} - \mathbb{E}[Y])^2] := V^{(L)}_{MLMC} + (b^{(L)}_{MLMC})^2 \tag{22}$$

where $V^{(L)}_{MLMC}$ and $b^{(L)}_{MLMC}$ are respectively the variance and the bias of the *MLMC* estimator.

Variance bound Using the identity in equation 12 for the optimal $\{N_\ell\}_{\ell=1}^{L}$ minimising the variance at fixed cost, one finds that for any $L \geq 1$ and given threshold ε^2 we have bounded variance:

$$V^{(L)}_{MLMC} = \sum_{\ell=1}^{L} \frac{V_\ell}{N_\ell} = \varepsilon^2 \sum_{\ell=1}^{L} \frac{\sqrt{V_\ell C_\ell}}{\sum_{\ell'=1}^{L} \sqrt{V_{\ell'} C_{\ell'}}} = \varepsilon^2 \tag{23}$$

.

Bias bound Noticing that the expectation of the whole estimator $\hat{\mu}^{(L)}_{MLMC}$ is equivalent to expectation of the last level Y_L, then we can find a tight bound by evaluating the telescopic sum of higher levels,

$$b^{(L)}_{MLMC} = |\mathbb{E}[Y_L] - \mathbb{E}[Y]| \leq \left(\sum_{k=0}^{M} |\mathbb{E}[Y_{L+k+1}] - \mathbb{E}[Y_{L+k}]| \right) + |\mathbb{E}[Y_{L+M}] - E[Y]| \tag{24}$$

for any integer $M \geq 0$. Assuming now that weak errors have an exponential decay trend $E_l := |\mathbb{E}[Y_\ell] - \mathbb{E}[Y_{\ell-1}]| = \tilde{E}_0 \cdot 2^{-\alpha\ell}$ for all $\ell \geq 1$ (remark: first level is excluded because $Y_{-1} = 0$). Injecting this expression in eq. 24, taking the limit of $M \to \infty$ and bounding the expression to a chosen threshold ε one finds an analytical expression for the optimal max level L^*,

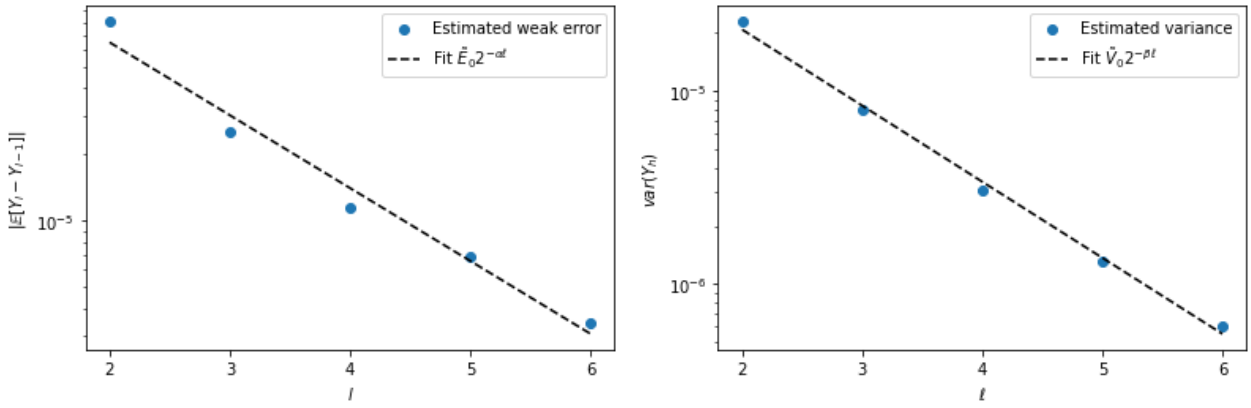
$$b^{(L^*)}_{MLMC} \leq \tilde{E}_0 \cdot \frac{2^{-\alpha(L^*+1)}}{1 - 2^{-\alpha}} < \varepsilon \quad \implies \quad L^* = \left\lceil \frac{1}{\alpha} \log_2 \left(\frac{\tilde{E}_0}{\varepsilon(1 - 2^{-\alpha})} \right) \right\rceil \tag{25}$$

.

Bias and variances results As expected we find that the bias and variances decay exponentially, factors are shown in table 2. The graphs in figure 4 show the behaviour of the errors as the level increases and, by fitting that curve, it's possible to retrieve the coefficients $\tilde{V_0}$ and $\tilde{E_0}$, as well as the exponents β, α. Furthermore, note that $V_0 \neq \tilde{V_0}$, because $Y_{-1} = 0$.

Pilot run			
α	$\tilde{E_0}$ [10^{-4}]	β	$\tilde{V_0}$ [10^{-4}]
1.10 ± 0.1	2.98 ± 0.05	1.310 ± 0.005	1.28 ± 0.01
Cost comparison			
MC decay exponent		*MLMC* decay exponent	
3.16 ± 0.05		2.02 ± 0.05	

Table 2: Results for the log-log linear fitted values in figures 4 and 5.



(a) Estimated weak error

(b) Estimated variance

Figure 4: Plots of weak errors and variance varying level ell at fixed number of samples $N_\ell = 40000$. First level was discarded.

We can also see that the bias measurement is much more unstable compared to the variance. In order to simulate a lower number of total samples, we used the median trick after the variance reduction given by the mean, which ensures a reliable result with a bounded cost [4]. Similarly to the results of the previous section (see figures 2) we have that α and β is slightly larger than 1. This has also a non indifferent influence on the cost evaluation.

CPU cost evaluation In figure 5 *MLMC* and crude *MC* are compared using the python function `time.process_time()`, which gives the time relative to the number of operations performed by a processing unit. *MLMC* simulation was performed with the optimal values of N_ℓ and L given by the identities (22, 25) and the values of $\tilde{E}_0$ and $\tilde{V}_0$ found with the aid of a pilot simulation. Crude *MC* was performed in the same fashion of the algorithm described in the previous section 3.1. It turns out that for a target accuracy around $\varepsilon = 3.5 \cdot 10^{-4}$, we find the same computational cost and crude *MC* performs even better for larger ε. However, the computational cost increases with a factor around 3 for *MC* and a factor 2 for *MLMC*, then for any smaller target accuracies *MLMC* would perform better in practice. This result is straight forward explained by eq. 17 for crude *MC* and theorem 2.1 of Giles's work [1] for *MLMC*. Briefly, the direct consequence of $\beta > 1$ is that the total cost of *MLMC* is bounded by a factor $\mathcal{O}(\varepsilon^{-2})$, which is exactly what we find. Furthermore, the conditions to apply the theorem are met because of the setup of the system. Which is surprising instead, is the amount of required samples compared to the level depth. A target accuracy of $\varepsilon = 10^{-5}$ requires only $L = 6$ levels and $N_0 \approx 10^8$, which could seem very close to the unfeasibility. On the other hand, N_0/h_0 still remains small compared to the crude Monte Carlo cost N_{MC}/h_L. The results in table 3 show that, below the accuracy $3.42 \cdot 10^{-4}$, *MLMC*'s cost scales much slower then crude *MC*. We remark that this results is not only limited to the time complexity, but also the

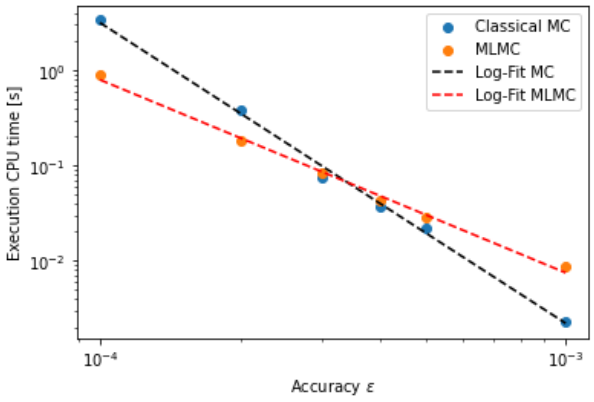


Figure 5: Cost of optimal set-up *MLMC* in comparison with classical *MC* at the same target MSE $2\varepsilon^2$.

space complexity is affected because all time steps of the simulation must be stored in order to compute the options. Hence, *MLMC* in a vectorized implementation like ours allows to gain enough space to lower the accuracy and fit into a small amount of memory.

Accuracy ε	$\hat{\mu}_{MLMC}$	Levels	$\hat{C}_{MLMC}$ [s]	$\hat{\mu}_{MC}$	$\hat{C}_{MC}$ [s]
5.7×10^{-5}	0.057677	8	3.69	0.057674	14.60
1.14×10^{-4}	0.05771	8	1.16	0.05745	3.75
2.28×10^{-4}	0.05723	7	0.17	0.05743	0.41
3.42×10^{-4}	0.05749	6	0.07	0.05747	0.05

Table 3: Results of simulations obtained with different target accuracies ε. Levels were augmented of 5 far from the necessary given in eq. 25 and the cost was computed using python function `time.process_time()` differences.

3.4 Barrier call option (e)

We define now the *barrier call* option as

$$Y = \exp(-r)\max(0, S(T) - K)\mathbb{1}_{\max_{t\in[0,T]} S(t)<S_{max}} \tag{26}$$

where $S_{max} = 1.5$ and $K = 1$ like before. We aim to check to same properties of bias and variance convergence as it's been done with the asian option. We apply the same procedure of last section executing a pilot run with fixed number of samples $N_{pilot} = 50000$ and get the coefficients α and β by linear fitting the log-log curve of

Pilot run *barrier call*			
α	$\tilde{E_0}$ $[10^{-3}]$	β	$\tilde{V_0}$ $[10^{-3}]$
0.451 ± 0.05	4.13 ± 0.05	0.425 ± 0.005	1.72 ± 0.01

Table 4: Fit results for bias and variances using *barrier call* option. α and β are the interested decay factors.

Table 4 shows that α and β coefficients are similar like in the case of asian option. However, their value is halved. This result is probably due to the strong discontinuity characterizing the Barrier Call option. In fact, contrary to the Asian option case, the lipschitz continuity can't be verified and thus, the convergence bounds holding for the simulated S_m are not guaranteed to be be transmitted to the option Y. This assumption could explain thee discrepancy of results between the Asian option and the Barrier Call option.

3.5 Crude Monte Carlo for the two payoffs (f)

We now compute a Crude Monte Carlo estimator $\mathbb{E}[Y]$ for both Asian option and Barrier Call option.
Where $K = 2$ and $S_{max} = 2.5$.

Variance Reduction Technique (VRT) We are interested in looking how the results would appear after the implementation of a Variance Reduction Technique for the the estimator mentioned above. We choosed

to use the VRT's algorithm for Antithetic variables, since its simplicity in implementation. The idea is to generate N/2 iid pairs of random variable, negatively correlated to the initial N iid r.v. It follows then that $\hat{\mu}_{AV} = \hat{\mu}_{MC}$ but we'll have now that $\mathbb{V}(\hat{\mu}_{AV}) < \mathbb{V}(\hat{\mu}_{MC})$
The algorithm for Antithetic variables is shown below:
Input: N number of crude samples

- Generate $N/2$ iid replicas $S^{(1)}, ..., S^{N/2}$ of S
- For each $S^{(i)}$ compute $Y^{(i)} = \psi(S^{(i)})$ and $Y_a^{(i)} = \psi(2\mathbb{E}[S] - S^{(i)})$
- Compute $\hat{\mu}_{AV} = \frac{1}{N} \sum_{i=1}^{N/2} (Y^{(i)} + Y_a^{(i)})$
- Compute $\hat{\sigma}_{AV} = \frac{1}{N/2-1} \sum_{i=1}^{N/2} (\frac{Y^{(i)}+Y_a^{(i)}}{2} - \hat{\mu}_{AV})^2$

	Estimation $\hat{\mu}$		Variance Var$[\hat{\mu}]$	
	Asian	Barrier Call	Asian	Barrier Call
MC	0.0575	0.1044	$6.143 \cdot 10^{-8}$	$2.171 \cdot 10^{-7}$
AV	0.0575	0.1044	$2.881 \cdot 10^{-8}$	$1.068 \cdot 10^{-7}$

Table 5: Results of Antithetic variable estimation compared to crude Monte Carlo using both Asian Option and Barrier Call Option with $N = 100,000$ in an interval of 1000 evaluations.

As we can see from table 5, μ is the same for both MC and AV as previously mentioned in the assumptions. And σ^2 in AV is even smaller than a half than in MC. Therefore we can say that the variance reduction completed with success.

Extension on MLMC Regarding the possible use of AV's VRT in the context of *MLMC*, it's feasible because *MLMC* combines a sequence of crude Monte Carlo estimators $\hat{\mu}_\ell$ for each level. Thus, the same algorithm can be applied to any level estimator. However, it's not strictly necessary because *MLMC* was designed to target a specific accuracy in an efficient way and further reducing the variance wouldn't gain much on the total *MSE*.

4 Conclusion

In conclusion, the use of MLMC for option pricing has been shown to be a powerful and efficient tool for solving high-dimensional and complex problems in comparison with classical Monte Carlo methods. By utilizing a multi-level approach, MLMC is able to significantly reduce the number of samples required to achieve a desired level of accuracy (MSE bound), compared to traditional MCMC methods. However, the use of this method also comes with some limitations and challenges. The method assumes the underlying asset prices follow a geometric brownian motion, which might not be suitable in cases where the underlying asset prices exhibit different behavior. Furthermore, the method assumes the perfect knowledge of parameters, such as drift r and volatility σ, which in this homework are known and it might not always be the case in the real world. Overall, the use of MLMC method for option pricing can provide a significant advantage in terms of computational and memory efficiency, but it is important to consider the assumptions and limitations of the method before applying it to real-world necessities.

Bibliography

[1] Michael B. Giles. “Multilevel Monte Carlo methods”. In: *Acta Numerica* 24 (2015), pp. 259–328. DOI: 10.1017/S096249291500001X.

[2] Yanting Ji. “Convergence rate of Euler–Maruyama scheme for SDDEs of neutral type”. In: *Journal of Inequalities and Applications* 2021 (Jan. 2021). DOI: 10.1186/s13660-020-02533-3.

[3] Numpy doc. *Numpy trapz*. 2022. URL: https://numpy.org/doc/stable/reference/generated/numpy.trapz.html#numpy.trapz.

[4] Yen-Chi Chen. *A short note on the median-of-means estimator*. 2020. URL: https://faculty.washington.edu/yenchic/short_note/note_MoM.pdf (visited on 2020).

```python
import numpy as np

"""
    It simulates with Euler-Maruyama approximation the stochastic process
       given by the equation:

    dS(t) = r * S(t) * dt + sigma * S(t) * dW(t)

    It returs the collection of S(t) over a uniformly discretizatized time
        domain (0, T].


Parameters:

    S_init = initial condition of S
    r = coefficient of deterministic part of the option pricing equation
    sigma = coefficient of the stochastic part of the option pricing
       equation
    dW = collection of standard normal realizations, \sim N(0, 1)
    T = final time
"""
def EM_option_pricing(S_init, r, sigma, h, dW, T):

    M = dW.shape[-1]
    S = np.ones(shape = (*dW.shape[:-1], M+1)) * S_init

    sqrt_h = np.sqrt(h)

    for m in range(1, M+1):
        S[...,m] = (1. + r * h + sigma * sqrt_h * dW[...,m-1])  * S[...,m
           -1]

    return S



"""
    S_init = initial condition of S
    r = coefficient of deterministic part of the option pricing equation
    sigma = coefficient of the stochastic part of the option pricing
       equation
    f = function to estimate the mean f(S(T))
    T = final time of the simulation
    h0: initial time step
    nb_samples(np.array): number of samples for each level

    return_samples: return Y_l - Y_{l-1}
    return_biases: return weak errors abs(mean(Y_l - Y_{l-1}))
    return_variances: return strong errors np.var(Y_l - Y_{l-1})
"""
def MLMC_option_pricing(S_init, r, sigma, T, f, h0, nb_samples,
```

```
    return_samples = False, return_biases = False, return_variances = False
    ):

47
48      # number of levels
49      L = len(nb_samples) - 1
50
51      # determine steps
52      steps = np.array([h0 * 2**(-l) for l in range(L+1)])
53
54      # number of time steps for each level
55      M = np.round(T / steps).astype(np.int64)
56
57      # estimated means
58      level_means = np.zeros(L+1)
59
60      # intermediate samples of Y
61      dY = []
62
63      #  loop on levels
64      for l in range(L+1):
65
66          # exponential decay browian motion sampling
67          dW_actual = np.random.standard_normal(size = M[l] * nb_samples[l])
                .reshape(( nb_samples[l], M[l] ))
68
69          Y_prev = np.zeros(nb_samples[l])
70
71          if l > 0:
72              # twice the time step brownian motion
73              dW_prev = (dW_actual[:,0::2] + dW_actual[:,1::2]) / np.sqrt
                    (2.)
74
75              # previous step computation
76              S_prev = EM_option_pricing(S_init, r, sigma, steps[l-1],
                    dW_prev, T)
77              Y_prev = f(S_prev, h = steps[l-1])
78
79          # actual step computation
80          S_actual = EM_option_pricing(S_init, r, sigma, steps[l], dW_actual
                , T)
81          Y_actual = f(S_actual, h = steps[l])
82
83          # store intermediate samples
84          dY.append(Y_actual - Y_prev)
85
86          # average over samples
87          level_means[l] = np.mean(Y_actual - Y_prev)
88
89      # collect outputs
90      outputs = [np.sum(level_means)]
91
92      # return forward model samples
```

```
    if return_samples:
        outputs.append(dY)

    # return estimated biases
    if return_biases:
        biases = [np.abs(np.mean(dy)) for dy in dY]
        outputs.append(biases)

    # return estimated variances
    if return_variances:
        variances = [np.var(dy, ddof=1) for dy in dY]
        outputs.append(variances)

    return tuple(outputs) if len(outputs) > 1 else outputs[0]
```

Script list

- Implementations: `MLMC_option_pricing.py`
- Euler maruyama test: `test_EM.py`
- Point (b): `test_crude_mc.py`
- Point (c): `test_two_levels.py`
- Point (d): `test_multi_level.py`
- Point (e): `test_multi_level_barrier.py`
- Point (f): `test_var_redux.py`