On Multilevel Quasi-Monte Carlo Methods

Candidate Number

869133

University of Oxford

A thesis submitted in partial fulfilment of the MSc in

Mathematical and Computational Finance

Trinity 2015

Acknowledgements

I would like to express my gratitude to my supervisor Professor Mike Giles for his support and for the time he has devoted to me. Without his help, this paper would not have been the same.

Abstract

In this paper, we show that the use of scrambled Sobol' sequences in the context of multilevel quasi-Monte Carlo path simulation leads to a reduction of the order of complexity, that is the computational cost required to attain a specific accuracy ε. More specifically, Sobol' points lead to a cost of $\mathcal{O}(\varepsilon^{-p})$, where p ranges from $p \approx 1.12$ to $p \approx 2.01$ for the options we consider, which represents an improvement over the previous best known results obtained by Giles and Waterhouse ([GW09]) with the use of randomised rank-1 lattice rules.

Contents

List of Figures

1 Introduction

Monte Carlo methods are general sampling methods which are widely used to compute expectations arising in stochastic systems. In computational finance, they are particularly useful to evaluate the expected value of a contract whose payoff depends on the solution of a stochastic differential equation. However, a major drawback of this class of algorithms is that they can be computationally expensive, and therefore inadequate for some applications. It is later shown that to achieve a root-mean-square error of ε, the plain Monte Carlo approach with an Euler–Maruyama path discretisation incurs a cost of $\mathcal{O}(\varepsilon^{-3})$, which can be significant for small ε.

Building on multigrid ideas from the theory of iterative solutions of discretised PDEs, Giles introduced a Multilevel Monte Carlo (MLMC) algorithm ([Gil08b]). He showed that the multilevel approach reduced the computational cost to $\mathcal{O}(\varepsilon^{-2}(\log \varepsilon)^2)$ in most cases considered, almost reaching the lower bound derived by Creutzig et al. ([CDMK09]) for the Euler–Maruyama discretisation ([Gil15], §5.1). The use of a Milstein discretisation in a subsequent paper ([Gil08a]) was shown to eliminate the logarithm term by speeding up the convergence of the variance of the multilevel estimator, and therefore improving further the order of complexity.

In 2009, Giles and Waterhouse published a paper ([GW09]) in which they combined the multilevel approach with ideas borrowed from classical quasi-Monte Carlo theory. By using a set of 32 independently randomised rank-1 lattice rules, they obtained a significant variance reduction on the coarsest levels, and further reduced the computational cost to approximately $\mathcal{O}(\varepsilon^{-1.5})$ in the case of a Lipschitz European payoff.

In this paper, we show that the use of Sobol' sequences leads to significant improvements. We implement a multilevel approach with a set of 32 independently scrambled Sobol' sequences together with a Milstein path discretisation, and we show that it achieves a computational cost of $\mathcal{O}(\varepsilon^{-1.23})$ in the European option case.

The paper is structured in three main parts. We first recall basic notions related to quasi-Monte Carlo methods and its implementation in the context of financial engineering. We then give an introduction to the multilevel approach of Giles, make the link with quasi-Monte Carlo theory, and give an outline for a MLQMC algorithm. Finally, we apply the theory to price a variety of financial options, and compare our results with those in ([GW09]).

2 Background

In this section, we recall some ideas and concepts that will be recurrent throughout the paper. In particular, we briefly describe quasi-Monte Carlo methods and their implementation, and we give an introduction to the theory of low-discrepancy sequences.

2.1 Quasi-Monte Carlo methods

In this paper, we fix $d \in \mathbb{N}$, and we let $\mathcal{I}^d = [0,1]^d$ be the closed, d-dimensional unit hypercube. Consider a random variable X, uniformly distributed on $\mathcal{I}$. For an integrable function f defined on $\mathcal{I}$, the expectation of $f(X)$ is equal to its integral. In other words,

$$\mathbb{E}[f(X)] = \int_{\mathcal{I}} f(x)\,\mathrm{d}x,$$

and similarly for $\mathcal{I}^d$ when $d > 1$. Therefore, the problem of computing the expected value of a random variable can be replaced by numerical integration. To be more specific, the estimator

$$\theta_N = \frac{1}{N}\sum_{i=1}^{N} f(x_i), \quad x_i \sim \mathcal{U}(0,1),$$

defined for each N is unbiased, and by the strong law of large numbers, it converges almost surely to $\mathbb{E}[f(X)]$ as $N \to \infty$. The variance of θ_N is $N^{-1}\mathbb{V}[f]$, so that the root-mean-square error (RMSE) is $\mathcal{O}(N^{-1/2})$. To achieve a prespecified RMSE of ε therefore requires generating $N = \mathcal{O}(\varepsilon^{-2})$ samples.

This can prove to be computationally expensive, especially if evaluating each sample is costly. For example, this is the case in option pricing, where one is interested in the expected value of a quantity which is a functional of the solution to a given stochastic differential equation (SDE). To fix ideas, suppose we want to estimate $\mathbb{E}[f(S_T)]$, where f is a (scalar) payoff function which we assume to be Lipschitz continuous, and where S_t satisfies the multidimensional SDE

$$\mathrm{d}S_t = a(S_t,t)\,\mathrm{d}t + b(S_t,t)\,\mathrm{d}W_t, \quad 0 < t < T.$$

Consider a time grid $0 = t_0 < t_1 < \cdots < t_m = T$ with timestep h, and let $S_n = S(t_n) = S(nh)$. The simplest approximation to this SDE is the *Euler–*

Maruyama path discretisation, which leads to the scheme

$$\widehat{S}_{n+1} = \widehat{S}_n + a(\widehat{S}_n, t_n)h + b(\widehat{S}_n, t_n)\Delta W_n,$$

and our estimate for $\mathbb{E}[f(S_T)]$ is then the mean of the payoff values $f(\widehat{S}_{T/h})$ across N independent path simulations. Theoretical results ([BT95; KP92]) show that under certain conditions on $a(S_t, t)$ and $b(S_t, t)$, the error in the expected payoff due to using a finite timestep h (the *weak error*) is $\mathcal{O}(h)$, while the expected error in the individual paths (the *strong error*) is $\mathcal{O}(\sqrt{h})$. This implies that the RMSE is of the form $c_1 N^{-1/2} + c_2 h$ for constants $c_1, c_2 \in \mathbb{R}$. To achieve a RMSE of ε as before therefore requires $N = \mathcal{O}(\varepsilon^{-2})$ samples, and the use of a timestep $h = \mathcal{O}(\varepsilon)$ leads a total computational cost of $C = \mathcal{O}(\varepsilon^{-3})$. For small ε, this cost can be significant, outlining one of the weaknesses of Monte Carlo methods.

A common way of reducing this high cost is the use of quasi-Monte Carlo (QMC) methods. The basic idea behind QMC is to replace the d-dimensional points uniformly sampled from $\mathcal{I}^d$ by well-chosen deterministic points. Indeed, by using a more regularly distributed set of points we can achieve a better sampling of the function f, and therefore obtain faster convergence. As we will see in Section 2.3, the uniformity of a set is formally measured by its *discrepancy*. Roughly speaking, a d-dimensional sequence will be referred to as *low-discrepancy* if it fills $\mathcal{I}^d$ more uniformly than uncorrelated random points. The quasi-Monte Carlo method then estimates the integral of a function f over a d-dimensional hypercube with an N-point equal-weight quadrature rule

$$\int_{\mathcal{I}^d} f(u)\,\mathrm{d}u \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i), \tag{2.1}$$

where $(x_i)_{1\leq i\leq N}$ is a d-dimensional low-discrepancy sequence. As we have seen above, plain Monte Carlo achieves an accuracy of $\varepsilon = \mathcal{O}(N^{-1/2})$. One advantage of QMC is that in the best cases, it leads to $\mathcal{O}(N^{-1})$ convergence: in fact, it can be shown ([KS05]) that under appropriate conditions, the error in a quasi-Monte Carlo approximation is $\mathcal{O}(N^{-1+\delta})$ for any $\delta > 0$. However, the use of a deterministic sequence comes at a cost, as we lose the possibility of constructing confidence intervals, and the estimate in (2.1) is biased.

One way of regaining confidence intervals is by introducing randomised QMC. The basic idea is to repeat a QMC integration M times independently,

giving estimates $I_1, \dots, I_M$. Using these M realisations and the Central Limit Theorem then allows us to construct confidence intervals in the classical way. To be more specific, we consider M independently randomised low-discrepancy sequences $(x_i^{(1)}), \dots, (x_i^{(M)})$ and we define estimators

$$I_m = \frac{1}{N} \sum_{i=1}^{N} f(x_i^{(m)}), \quad m = 1, \dots, M.$$

We then construct the global estimator $I = M^{-1} \sum_{j=1}^{M} I_j$, which is unbiased. This allows the estimation of the variance of the estimator in the usual way, and therefore the computation of confidence intervals. Note that the choice of M is crucial: by increasing M, one can obtain a better variance estimate using the Central Limit Theorem, but this comes at the cost of a poorer error. In this paper, we have chosen to use $M = 32$.

2.2 Notes on implementation

In this section, we give more details on the implementation of quasi-Monte Carlo integration in a financial context. We first describe how to approximate SDEs that we encounter when modelling underlying assets, and we then outline the different steps in the implementation of a quasi-Monte Carlo method.

2.2.1 Path discretisation

In many financial applications, part of the pricing problem is to model the behaviour of the underlying assets. In our applications, the assets will follow SDEs of the form

$$\mathrm{d}S = a(S,t)\,\mathrm{d}t + b(S,t)\,\mathrm{d}W. \tag{2.2}$$

For example, we will consider the case of Geometric Brownian Motion (GBM), where $a(S,t) = \mu S$ and $b(S,t) = \sigma S$ for some $\mu, \sigma \in \mathbb{R}$. As mentioned in Section 2.1, an *Euler–Maruyama* path discretisation leads to a weak error of $\mathcal{O}(h)$ and a strong error of $\mathcal{O}(\sqrt{h})$. Although strong convergence is usually not important, it proves to be key for multilevel Monte Carlo methods. In particular, the best order of complexity for a given RMSE cannot be achieved with an Euler–Maruyama approximation, and instead one needs to use a Milstein discretisation (this is a direct consequence of Theorem 2.1 in [GW09]). For the

SDE (2.2), a *Milstein* path discretisation leads to the scheme

$$\widehat{S}_{n+1} = \widehat{S}_n + a_n h + b_n \Delta W_n + \frac{1}{2}\frac{\partial b_n}{\partial S} b_n((\Delta W_n)^2 - h),$$

where a_n, b_n and $\partial b_n/\partial S$ are all evaluated at $(\widehat{S}_n, t_n)$. Under some conditions on $a(S,t)$ and $b(S,t)$ in (2.2), it has been shown ([KP92]) that the Milstein scheme achieves a strong convergence of order one, which is why it is more suitable for multilevel Monte Carlo methods. For more details on the use of the Milstein discretisation in the multilevel context, see ([GW09], §3).

2.2.2 QMC implementation

As described in Section 2.1, our aim is to estimate an expected payoff $\mathbb{E}[f(S(T))]$. Consider an SDE path simulation (as one described in Section 2.2.1) with M timesteps, giving a path $\widehat{S}$. Starting with a standard M-dimensional normal random variable Z, we can express this expected value as

$$\mathbb{E}[\widehat{f}(\widehat{S}(T))] = \int \widehat{f}(\widehat{S}(T))\phi(Z)\ \mathrm{d}Z, \tag{2.3}$$

where $\phi(Z)$ is the density function of Z, and where $\mathrm{d}Z = \prod_{i=1}^{M} \mathrm{d}Z_i$, with Z_i being standard one-dimensional normals. Let Φ denote the corresponding distribution function of Z, and let $U_i \sim \mathcal{U}(0,1)$ for $i = 1, \ldots, M$. Then, putting $Z_i = \Phi^{-1}(U_i)$ for each i turns the integral in (2.3) into

$$\mathbb{E}[\widehat{f}(\widehat{S})] = \int_{\mathcal{I}^M} \widehat{f}(\widehat{S})\ \mathrm{d}U.$$

This is then approximated as in (2.1), and so the QMC procedure can be summarised as follows:

$$U \xrightarrow{(1)} Z \xrightarrow{(2)} \Delta W \xrightarrow{(3)} \widehat{S} \xrightarrow{(4)} \widehat{f}.$$

Step (1) consists of generating quasi-normals from quasi-uniforms, and is achieved as described above by applying Φ^{-1} to the quasi-uniforms. Step (3) is described in Section 2.2.1 above, and Step (4) is simply a matter of evaluating the payoff function given the discrete states $\widehat{S}_i$.

As a consequence, the two aspects of a randomised QMC implementation on which we will focus are the choice of the low-discrepancy sequence and its randomisation method (which can be seen as Step (0)) as well as the generation

of Brownian increments from normal random variables, which is Step (2). We will devote Section 2.3 to the theory of low-discrepancy sequences, and we will describe specific sequences and randomisation techniques in Section 2.4. We now focus on the latter aspect of quasi-Monte Carlo simulation, which is the generation of Brownian increments ΔW.

Let $W(t)$ denote a scalar standard Brownian motion, and a for a timestep h, define $W_n = W(nh)$ where $n \in \mathbb{N}$. Then, each W_n is normally distributed, and for any $i \geq j$ we have $\mathbb{E}[W_i W_j] = t_j$, so that the covariance matrix for W is $\Sigma = (\Sigma_{ij})$, where $\Sigma_{ij} = \min\{t_i, t_j\}$. We now need to find a matrix L such that $LL^\top = \Sigma$. Clearly, L is not unique, and while the choice of L does not matter for Monte Carlo applications, it is very important for quasi-Monte Carlo methods ([Gla03], §3.1).

A first approach is to take L to be the *Cholesky* factor of Σ, which is easily found to be

$$L = \sqrt{h} \begin{pmatrix} 1 & 0 & \dots & 0 & 0 \\ 1 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 1 & \dots & 1 & 0 \\ 1 & 1 & \dots & 1 & 1 \end{pmatrix}.$$

This gives $W_n = \sum_{m=1}^{n} \sqrt{h}\, Z_m$, so $\Delta W_n = W_n - W_{n-1} = \sqrt{h}\, Z_n$ and we obtain a vector of normal random variables with mean zero and variance h.

A second method is known as *Principal Component Analysis*. In this case, we set $L = U\Lambda^{1/2}$, where U, Λ are the matrices of eigenvectors and eigenvalues respectively, and where the eigenvalues are arranged in descending order. In other words, the nth column of L is $\sqrt{\lambda_n}\, u_n$, where λ_n is the nth largest eigenvalue, and u_n its corresponding eigenvector. In Section 4.1, we describe a hybrid PCA method which we use for our numerical results later on.

The third established method to compute L is known as the *Brownian bridge* construction. The idea behind Brownian bridge is to use the first component of Z to define the terminal value $W(M)$, the second value of Z to define $W(M/2)$ conditional on $W(M)$, and so on. More specifically, we let $W_M = \sqrt{T}\, Z_1$. Conditional on W_M, the midpoint value $W_{M/2}$ is normally distributed with mean $\frac{1}{2}W_M$ and variance $T/4$, and so

$$W_{M/2} = \frac{1}{2} W_M + \sqrt{T/4}\, Z_2.$$

Repeating this procedure, the third iteration produces $W_{M/4}$ and $W_{3M/4}$, and we can carry on recursively to generate all Brownian values, and therefore generate ΔW. Note that behind this construction, there is the implicit assumption that M is a power of two. If this is not the case, the method still works but is slightly more complex. The advantage of Brownian bridge is its flexibility, and furthermore it is well-known that this construction is suited for low-discrepancy methods ([Gla03], §3.1.1). More specifically, for European payoffs, Brownian bridge almost reduces the generation of Brownian increments to a one-dimensional problem. Under a Brownian bridge construction, and as opposed to a PCA approach for example, the main shape of the Brownian path is determined by the values of the first few normal random variables Z_i (see [Gla03], §5.5). This is particularly important in light of a result by Koksma and Hlawka (Theorem 2.1 of this paper) which gives a bound for the quasi-Monte Carlo error in terms of the dimension of the problem.

2.3 Low-discrepancy sequences

In this section, we give a formal introduction to discrepancy theory. We then make the link with Monte Carlo methods via the Koksma–Hlawka inequality.

Consider a set $P = \{x_1, \dots, x_N\} \in \mathcal{I}^d$. For an arbitrary subset $B \subseteq \mathcal{I}^d$, let $A(B; P)$ be the number of elements of P lying in B. If $\mathcal{B}$ denotes a non-empty family of Lebesgue-measurable set, then a general notion of *discrepancy* of the set P is given by

$$D_N(\mathcal{B}; P) = \sup_{B \in \mathcal{B}} \left| \frac{A(B; P)}{N} - \lambda_d(B) \right| \in [0, 1],$$

where $\lambda_d(B)$ is the d-dimensional Lebesgue measure of B (in this context, one can think of $\lambda_d(B)$ as the volume of the d-box B). By specifying $\mathcal{B}$, we obtain various definitions of discrepancy. For example, we will refer to the *discrepancy* of P as $D_N(P) = D_N(\mathcal{B}, P)$ where $\mathcal{B}$ is the family of subintervals of $[0, 1)^d$ of the form $\prod_{i=1}^d [u_i, v_i)$. If all the u_i are identically zero in $\mathcal{B}$, then we obtain the *star discrepancy* of P, denoted $D_N^*(P)$. For our purposes, it will be sufficient to consider the discrepancies defined above, but we point out that several other notions of discrepancy exist ([Nie87], §2.1). We now explain the link with Monte Carlo methods.

Let S be a sequence (i.e. a set) in $\mathcal{I}^d$. By classical results in the theory of uniform distribution of sequences ([KN74], §2.1), S is uniformly distributed if and only if $D_N(S) = 0$, or equivalently $D_N^*(S) = 0$. In this sense, the

discrepancy is an adequate measure of the "regularity" of a set, as mentioned in Section 2.1. In addition to this appeal, low-discrepancy sequences play a central role in bounding the error of the approximation (2.1). The central result in that direction is a bound obtained by Jurjen Koksma and generalised by Edmund Hlawka. First, let us recall that for a sufficiently differentiable function f, the *Hardy–Krause variation* of f is defined as

$$V(f) = \int_{\mathcal{I}^d} \left| \frac{\partial^d f}{\partial x_1 \dots \partial x_d} \right| \mathrm{d}x.$$

For an alternative definition, see ([KN74], §2.5). We can now state the main theorem.

Theorem 2.1 (Koksma–Hlawka inequality)**.** *If f has bounded Hardy–Krause variation $V(f)$ on $\mathcal{I}$, then for any $x_1, \dots, x_N \in \mathcal{I}$, we have*

$$\left| \frac{1}{N} \sum_{n=1}^{N} f(x_n) - \int_0^1 f(u) \, \mathrm{d}u \right| \leq V(f) D_N^*(x_1, \dots, x_N).$$

In short, the error induced by the quadrature rule (2.1) is bounded by a product of two terms: first, a measure of the variation of the integrand, and second a measure of how the sampled points deviate from being uniform. In general, the Koksma–Hlawka inequality is tight ([Nie87], Theorem 2.12) in the sense that it is the best possible result, even for C^∞ functions. A second observation is that this result is a strict bound, whereas in the case of plain Monte Carlo, the best we can achieve is a probabilistic bound (by using the Central Limit Theorem). By constructing sequences with known asymptotic discrepancy, we can use this bound to show that QMC gives potentially significant improvements over traditional Monte Carlo. To be more specific, we will give examples of sequences whose star discrepancy is $\mathcal{O}(N^{-1}(\log N)^d)$, where d is the dimension of the problem. For d sufficiently small, this leads to a much better error than the usual $\mathcal{O}(N^{-1/2})$ of plain Monte Carlo.

However, it has to be noted that there are several limitations to using the Koksma–Hlawka inequality in practice. First, both $V(f)$ and $D_N^*(x_1, \dots, x_N)$ are expensive to compute, and in some cases even more costly than evaluating the original integral. Even worse, the Hardy–Krause variation is only rarely bounded in financial applications (which happens if f is unbounded for example), and even when it is, the bound given above often grossly overestimates the true integration error.

2.4 Main examples

We describe some of the low-discrepancy sequences we use in later parts of the paper. In particular, we mention randomisation techniques and results on asymptotic discrepancy for each sequence.

2.4.1 Lattice rules

Recall that a lattice Λ is a $\mathbb{Z}$-module with a finite basis $\{w_1, \dots, w_n\}$ such that any $u \in \Lambda$ can be written as

$$u = \lambda_1 w_1 + \cdots + \lambda_n w_n, \qquad \lambda_i \in \mathbb{Z}.$$

In other words, Λ is a vector space over $\mathbb{Z}$ spanned by $\{w_1, \dots, w_n\}$[1].

Definition 2.2. *A rank-1 lattice rule is a set $P_n = \{x_1, \dots, x_n\} \in \mathcal{I}^d$ with*

$$x_k = \frac{kz}{n} \bmod 1, \quad k = 0, \dots, n-1.$$

where z is a generating vector with integer components coprime with n.

The definition above, together with ordinary addition, turns P_n into a finite cyclic group. To make the link with lattices, recall that the rank of a (free) module is the cardinality of any basis. Equivalently, the rank of a lattice is the smallest number of cyclic groups into which it may be decomposed (this follows by the Chinese Remainder Theorem). In this case, "rank-1" refers to the fact that P_n can only be decomposed as the product of one cyclic group, namely P_n itself.

The advantage of using rank-1 lattice rules is that they have low discrepancy, in the sense defined in Section 2.3. In particular, it has been shown that for any $d \geq 2$ and $N \geq 2$, there are rank-1 lattice rules who achieve a discrepancy $D_N(P) \leq CN^{-1}(\log N)^d$ for a constant C ([Nie78] ,§1). Since then, several constructions of these lattice rules have been outlined (e.g. [Nuy07]).

In the context of QMC, randomisation is achieved via the introduction of an offset vector, a technique initially introduced by Cranley and Patterson ([CP76]). More specifically, let $\Delta_1, \dots, \Delta_M$ be M independent random vectors,

[1]Formally, let R be an integral domain with fraction field Q, and let K be a finite-dimensional Q-linear vector space. An *R-lattice* in K is an R-submodule $M \subseteq K$ such that M is finitely generated of rank equal to $\deg(K/Q)$, and such that M spans K as a Q-vector space (e.g. [Sam08]). Here, taking $R = \mathbb{Z}$, we recover the definition given above.

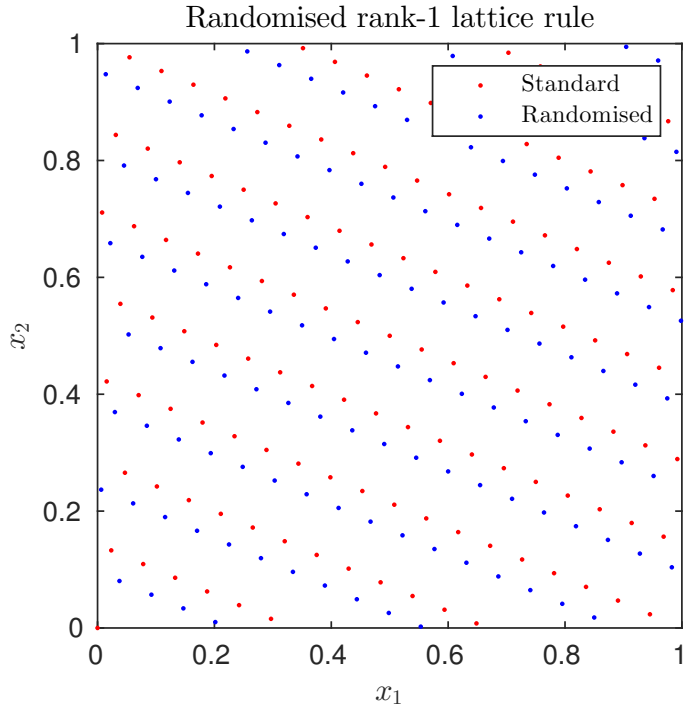

Figure 2.1: Two-dimensional rank-1 lattice rule (128 points)

uniformly distributed in $\mathcal{I}^d$, and for each $m = 1, \ldots, M$, define a sequence

$$x_k^{(m)} = \left(\frac{kz}{n} + \Delta_m \right) \bmod 1, \quad k = 0, \ldots, n-1.$$

This offset operation applied to our rank-1 lattice rule construction gives a set of M uniformly distributed sequences ([Gla03], §5.4). These M sequences give M independent, identically distributed and unbiased estimates of the form (2.1) and therefore allow the computing of confidence intervals as required.

We give an example of 128 points generated with a rank-1 lattice rule in Figure 2.1 above, both with and without a Cranley–Patterson shift.

2.4.2 Sobol' sequences

Here, we describe the generation of a Sobol' sequence (x_n) in one dimension, following the original method by Sobol' outlined in [BF88]. The paper describes a method which generalises up to dimension 40, but a remark by Joe and Kuo ([JK03]), which is used in Matlab, extends this to $d = 1111$.

To start, we need a sequence (v_n) of *direction numbers*, where $v_i = m_i/2^i$ for m_i an odd integer satisfying $0 < m_i < 2^i$. To specify the sequence (m_n), choose a *primitive* polynomial P of order d in the polynomial ring $\mathbb{Z}_2[x]$, where $\mathbb{Z}_2 = \mathbb{Z}/2\mathbb{Z}$ is the group of primitive residue classes modulo 2. Suppose that P takes the form

$$P = x^d + a_1 x^{d-1} + \cdots + a_{d-1}x + 1 \in \mathbb{Z}_2[x].$$

We then define the sequence (m_n) recursively,

$$m_i = 2a_1 m_{i-1} \oplus 2^2 a_2 m_{i-2} \oplus \ldots \oplus 2^{d-1} a_{d-1} m_{i-d+1} \oplus 2^d m_{i-d} \oplus m_{i-d},$$

where $\oplus$ denotes a bit-by-bit exclusive-or operation. As noted in [BF88], if P is of degree d, then $m_1, \ldots, m_d$ can be chosen arbitrarily so long as each m_i is odd and satisfies the condition $0 < m_i < 2^i$ given above. Subsequent values $m_{d+1}, \ldots$ are then determined by the recurrence relation. For reference, a working example is given in ([BF88], §2). Finally, the Sobol' sequence (x_n) can be specified by

$$x_i = b_1 v_1 \oplus b_2 v_2 + \ldots,$$

where $\ldots b_3 b_2 b_1$ is the binary representation of i.

The Sobol' sequences described above are examples of (t, d)-sequences (see [Gla03], §5.1.4), which were also introduced by Sobol'. These sequences are well-understood, and for example their star discrepancy is known to satisfy

$$D_N^*(P) \leq C_1 \frac{(\log N)^{d-1}}{N} + \mathcal{O}\left(C_2 \frac{(\log N)^{d-2}}{N}\right),$$

where C_1, C_2 are constants depending only on d ([Nie87], Theorem 4.10).

Sobol' sequences can be randomised in various ways. A classical technique is the use of digital scrambling, introduced by Owen ([Owe98]), which can be applied to more general families of sequences. However, in our paper, we use a different scrambling, implemented in Matlab, which was developed by Matoušek (see [Mat98] for more details).

Figure 2.2 shows that Sobol' points are more uniformly distributed on $\mathcal{I}^2$ than random points. It also demonstrates a property of Sobol' points: each square on the grid above contains exactly two Sobol' points, whereas they contain

anywhere from zero to five random points. For a complete description of this phenomenon, see ([Gla03], §5.1.4).

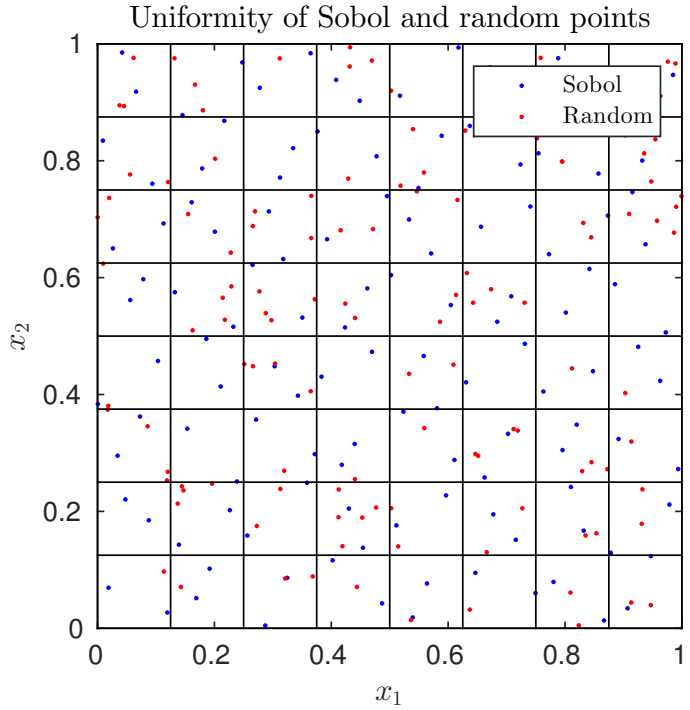

Figure 2.2: Two-dimensional Sobol' sequence (128 points)

2.4.3 Halton sequences

A third commonly cited low-discrepancy sequence is known as the *Halton sequence*. Let $p \geq 2$ be an integer. For $n \in \mathbb{N}$, let $n = n_0 + n_1 p + n_2 p^2 + \ldots$ be the base p expansion of n. This expansion is clearly finite as n is an integer, and furthermore each $n_i \in \{0, \ldots, p-1\}$. Define the *radical inverse function* $\phi_p : \mathbb{N} \to [0, 1)$ in base p as

$$\phi_p(n) = \sum_{s=0}^{\infty} n_s p^{-1-s}.$$

Now let $p_1, \ldots, p_d \geq 2$ be integers. We can define a *Halton sequence* P of dimension d in the bases $p_1, \ldots, p_d$ as the sequence (x_n) with

$$x_i = (\phi_{p_1}(i), \ldots, \phi_{p_d}(i)) \in [0, 1)^d, \quad \forall i \geq 0.$$

For $d = 1$, we obtain a particular case called the *Van der Corput sequence*.

It can be shown ([Nie87], Theorems 3.6–3.8) that the star discrepancy of a Halton sequence as above satisfies

$$D_N^*(P) \leq C_1 \frac{(\log N)^d}{N} + \mathcal{O}\left(\frac{(\log N)^{d-1}}{N}\right), \quad \forall N \geq 2,$$

where C_1 is a constant depending only on the primes $p_1, \ldots, p_d$ and which is known explicitly. It is also known that this constant can be minimised by choosing $p_1, \ldots, p_d$ to be the first d prime numbers ([Nie87], §3.1).

Figure 2.3 below shows 128 points generated from a Halton sequence, together with pseudo-random points.

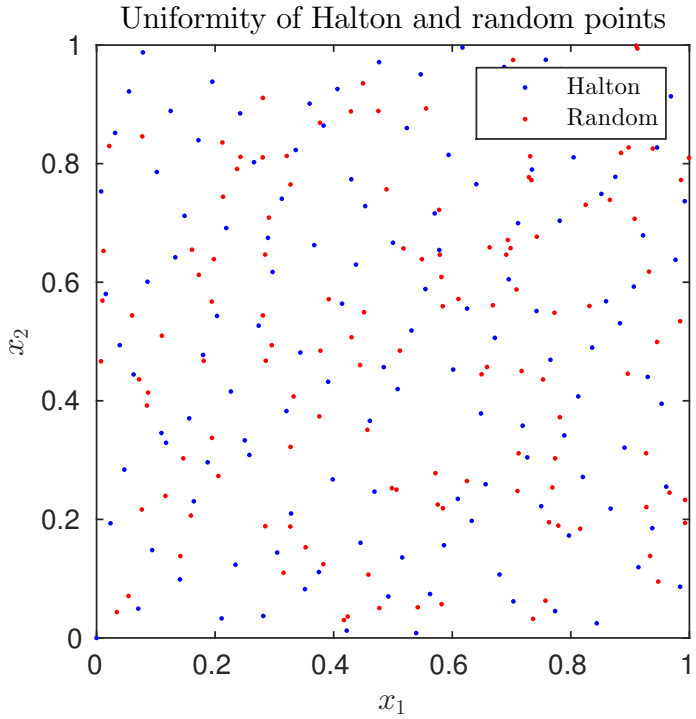

Figure 2.3: Two-dimensional Halton sequence (128 points)

For Halton sequences, randomisation can also be achieved in several ways. The technique implemented in Matlab is based on a permutation of the radical inverse coefficients, which is obtained by applying a reverse-radix operation to the possible coefficients ([KW97]). Another method, described in ([KP15], §1), is based on applying a p-adic shift using the Monna map, a generalisation of the radical inverse function defined above.

2.4.4 Higher dimensions

The constructions described above can be used to generate d-dimensional sequences for any value of $d \geq 1$. However, in practice, many low-discrepancy sequences suffer a loss of uniformity in higher dimensions ([DKS13; SAKK11]). Figure 2.4.4 below shows examples where a cross-dimensional study of 1000-dimensional lattice rules and Sobol' points reveals a decline in uniformity.

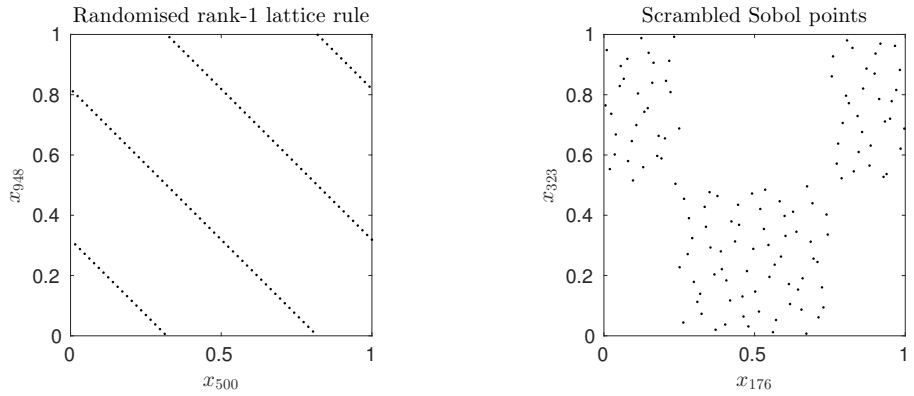

Figure 2.4: Uniformity of low-discrepancy sequences in high dimensions

Note that the behaviour described above is even more pronounced for the Halton sequence, as the decline in their uniformity is inherent to their construction ([Gla03], §5.2.1.). As an aside, we point out that an alternative to these Halton sequences was found by Faure, who developed an extension to the Van der Corput sequence who does not suffer such loss of uniformity. He showed that the Faure sequences are low-discrepancy, and furthermore that the constant term in the expression for the discrepancy goes to zero quicker than it does for the Halton sequence. Note that a complete description of his construction can be found in ([Gla03], §5.2.2).

In Section 4.6, we will describe how to analyse the effect of this curse of dimensionality in the context of a multilevel quasi-Monte Carlo integration. In particular, we will show that for high dimensions, it might be better to combine low-discrepancy and pseudo-random sequences to achieve better sampling of the unit hypercube and faster convergence of the MLQMC algorithm.

3 Multilevel quasi-Monte Carlo

In this section, we describe the multilevel quasi-Monte Carlo theory and outline some details of its implementation. Sections 3.1 and 3.2 are entirely based on the founding papers by Giles ([Gil08b]) and Giles and Waterhouse ([GW09]).

3.1 The multilevel approach

The initial setting is the same as the one in Section 2.1. We consider Monte Carlo path simulations with different timesteps $h_\ell = 2^{-\ell}T$, for $\ell = 0, \dots, L$. Let W_t denote a given Brownian path, P an option payoff, and let $\widehat{P}_\ell$ denote its approximation using a numerical path discretisation with timestep h_ℓ. The basis of the multilevel approach is the trivial observation that

$$\mathbb{E}[\widehat{P}_L] = \mathbb{E}[\widehat{P}_0] + \sum_{\ell=1}^{L} \mathbb{E}[\widehat{P}_\ell - \widehat{P}_{\ell-1}]$$

by linearity of the expectation operator. The idea is then to estimate each of the expectations in the sum in a way which minimises the overall variance of the estimator for a given computational cost. Let $\widehat{Y}_0$ be an estimator for $\mathbb{E}[\widehat{P}_0]$ using N_0 samples, and let $\widehat{Y}_\ell$ (for $\ell > 0$) be an estimator for $\mathbb{E}[\widehat{P}_\ell - \widehat{P}_{\ell-1}]$ using N_ℓ paths. The estimator on level ℓ is a mean of N_ℓ independent samples, which for $\ell > 0$ is given by

$$\widehat{Y}_\ell = \frac{1}{N_\ell} \sum_{i=1}^{N_\ell} \left(\widehat{P}_\ell^{(i)} - \widehat{P}_{\ell-1}^{(i)} \right).$$

To minimise the variance, it is important that $\widehat{P}_\ell^{(i)} - \widehat{P}_{\ell-1}^{(i)}$ comes from approximations on different levels, but with the same Brownian path. Under this assumption, the variance of the estimator is given by $\mathbb{V}[\widehat{Y}_\ell] = N_\ell^{-1} V_\ell$, where $V_\ell = \mathbb{V}[\widehat{P}_\ell^{(i)} - \widehat{P}_{\ell-1}^{(i)}]$ is the variance of a single sample. The variance of the global estimator $Y = \sum_{\ell=0}^{L} \widehat{Y}_\ell$ is therefore $\mathbb{V}[Y] = \sum_{\ell=0}^{L} N_\ell^{-1} V_\ell$, for a computational cost proportional to $\sum_{\ell=0}^{L} N_\ell h_\ell^{-1}$. Using the method of Lagrange multipliers and treating the N_ℓ as continuous variables, the variance is minimised for a fixed computational cost by choosing N_ℓ to be proportional to $\sqrt{V_\ell h_\ell}$. The analysis is further refined in the main theorem below.

Theorem 3.1 ([Gil08b], Theorem 3.1)**.** *Let P denote a functional of the solution of the SDE (2.2) for a given Brownian path $W(t)$, and let $\widehat{P}_\ell$ denote the cor-*

responding approximation using a numerical discretisation with timestep $h_\ell = M^{-\ell}T$.

If there exist independent estimators $\widehat{Y}_\ell$ *based on* N_ℓ *Monte Carlo samples, and positive constants* $\alpha \geq \frac{1}{2}$, β, c_1, c_2, c_3 *such that*

1. $\left|\mathbb{E}[\widehat{P}_\ell - P]\right| \leq c_1 h_\ell^\alpha$,
2. $\mathbb{E}[\widehat{Y}_\ell] = \begin{cases} \mathbb{E}[\widehat{P}_0], & \ell = 0, \\ \mathbb{E}[\widehat{P}_\ell - \widehat{P}_{\ell-1}], & \ell > 0, \end{cases}$
3. $\mathbb{E}[\widehat{Y}_\ell] \leq c_2 N_\ell^{-1} h_\ell^\beta$,
4. C_ℓ, *the computational complexity of* $\widehat{Y}_\ell$, *is bounded by* $C_\ell \leq c_3 N_\ell h_\ell^{-1}$,

then there exists a positive constant c_4 *such that for any* $\varepsilon < e^{-1}$, *there are values* L *and* N_ℓ *for which the multilevel estimator*

$$\widehat{Y} = \sum_{\ell=0}^{L} \widehat{Y}_\ell$$

has a mean-square-error with bound

$$MSE = \mathbb{E}\left[\left(\widehat{Y} - \mathbb{E}[P]\right)^2\right] < \varepsilon^2,$$

with a computational complexity C *with bound*

$$C \leq \begin{cases} c_4 \varepsilon^{-2}, & \beta > 1, \\ c_4 \varepsilon^{-2} (\log \varepsilon)^2, & \beta = 1, \\ c_4 \varepsilon^{-2-(1-\beta)/\alpha}, & 0 < \beta < 1. \end{cases}$$

Proof. See ([Gil08b], §3). □

In the case of an Euler path discretisation, Theorem 3.1 above implies that achieving a RMSE of ε incurs a computational cost of $\mathcal{O}(\varepsilon^{-2}(\log \varepsilon)^2)$. This is an improvement compared to the $\mathcal{O}(\varepsilon^{-3})$ of plain Monte Carlo, but still leaves scope for improvement. The first step is the use of a Milstein path discretisation, which achieves $\beta > 1$ above, and therefore reduces the complexity to $\mathcal{O}(\varepsilon^{-2})$. A further improvement, which is the central theme of this paper, is the application of a quasi-Monte Carlo element to Giles' multilevel method.

3.2 QMC component

In this section, we will follow a paper by Giles and Waterhouse [GW09] in which the authors develop a MLQMC algorithm using a randomised rank-1 lattice rule. Although we have used another kind of low-discrepancy sequence, namely Sobol' points, the theory which we describe below is similar for all such sequences.

At level ℓ, we define the number of samples N_ℓ to be the number of QMC points, and $\widehat{Y}_\ell$ is the average of $\widehat{P}_\ell$ (for $\ell = 0$) or $\widehat{P}_\ell - \widehat{P}_{\ell-1}$ (for $\ell > 0$) over the 32 sets of N_ℓ quasi-Monte Carlo points, each set being randomised appropriately. Note here that some authors use fewer than 32 sets, but doing so might hinder the confidence interval obtained via randomisation of the QMC component. We then compute an (unbiased) estimate of the variance V_ℓ of $\widehat{Y}_\ell$ from the 32 different averages.

Assuming first order weak convergence, the remaining bias at the finest level $\mathbb{E}[P - \widehat{P}_L]$ is approximately equal to $\widehat{Y}_L$. To allow for the possibility that $\widehat{Y}_\ell$ changes sign as ℓ increases before settling into first order asymptotic convergence, we estimate the magnitude of the bias with $\max\{\frac{1}{2}|\widehat{Y}_{L-1}|, |\widehat{Y}_L|\}$. As before, we know that the RMSE is the sum of the (combined) variance of the estimator and the square of the bias. To achieve a RMSE of ε, we therefore choose to make each term smaller than $\varepsilon^2/2$. The multilevel QMC algorithm can then be summarised as follows:

1. start with $L = 0$
2. compute V_L over the 32 sets of points, using $N_L = 1$
3. while the combined variance is greater than $\varepsilon^2/2$, double N_ℓ on the level with largest $V_\ell/(2^\ell N_\ell)$
4. if $L < 2$ or the bias estimate is greater than $\varepsilon/\sqrt{2}$, set $L = L + 1$ and go to step 2.

As explained in ([GW09]), on one hand, doubling N_ℓ will eliminate most of the variance V_ℓ. On the other hand, it will also incur a cost proportional to $2^\ell N_\ell$. We therefore choose ℓ to maximise $V_\ell/(2^\ell N_\ell)$, as this will lead to an optimal reduction in the variance per unit cost.

4 Numerical Results

In this section, we start by giving a summary of our findings in terms of the order of complexity required by the MLQMC algorithm to achieve a pre-specified accuracy. We then briefly describe the particular implementation of the MLQMC algorithm for a variety of financial options, and we then give our results, comparing them with rank-1 lattice rules.

Summary of results

Using independently scrambled Sobol' sequences with a Milstein path discretisation, we estimated the cost required to achieve an accuracy of ε. We then performed a regression to approximate this cost as $\mathcal{O}(\varepsilon^{-p})$, where p is the order of complexity, which is dependent on the type of the option and the choice of Brownian bridge or PCA construction, as described below.

	Option				
Method	European	Asian	Lookback	Digital	Barrier
BB	1.23	1.12	1.58	2.01	1.92
PCA	1.30	1.15	1.66	2.01	1.81

Figure 4.1: Orders of complexity for various options using MLQMC

In the simplest case of a Lipschitz European payoff, we obtain $p \approx 1.23$, while the previous best known result reported by Giles and Waterhouse was approximately $p \approx 1.5$ ([Gil15], §5.2). Furthermore, significant improvements can be observed across all option types, except the digital option. This is consistent with the results obtained in ([GW09]) where the authors note that the use of a QMC component does not add benefits over a plain multilevel method for the digital option. Overall, it seems that Sobol' sequences allow a better sampling of the payoff functions for the options we considered, and therefore a faster convergence of the MLQMC algorithm.

4.1 European call option

We first apply our multilevel quasi-Monte Carlo technique to a European call option, which has a payoff $P = \exp(-rT)(S(T) - K)^+$. We use an initial stock price $S(0) = 1$, a strike $K = 1$, and parameters $T = 1$, $r = 0.05$, $\sigma = 0.2$. We first recall the results obtained by Giles and Waterhouse with lattice rules in

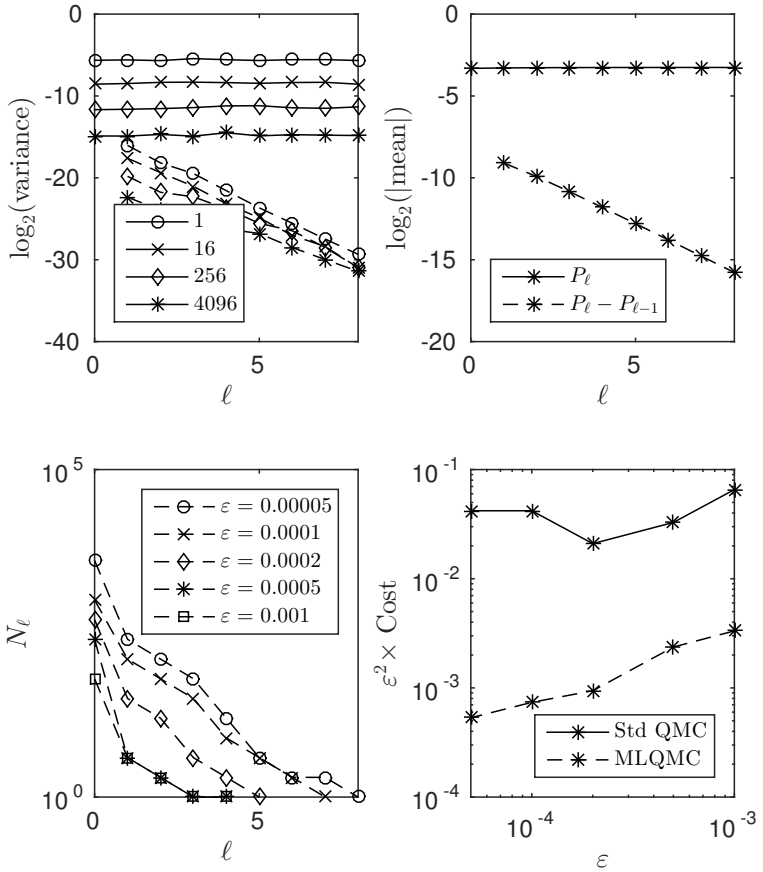

Figure 4.2: European call option with rank-1 lattice rule

Figure 4.2 and we then present our results with Sobol' points in Figure 4.3.

The solid lines in the top left plot of Figure 4.2 show how the variance of $\widehat{P}_\ell$ varies with the level when four different numbers N_ℓ of low-discrepancy points are used, and therefore illustrate the effect of using quasi-Monte Carlo methods (the case $N_\ell = 1$ corresponding to a standard Monte Carlo method). On the other hand, the dashed lines show the variance of $\widehat{P}_\ell - \widehat{P}_{\ell-1}$ on different levels, again for $N_\ell = 1, 16, 256$ and 4096 low-discrepancy points, and therefore illustrate the effect of combining the multilevel approach with QMC methods.

Recall that the quasi-Monte Carlo interpretation of the estimator $\widehat{Y}_\ell$ from Section 3.1 is an average over N_ℓ low-discrepancy points. When using a standard Monte Carlo method, the variance of the average of M points with common variance σ^2 is σ^2/M. For example, the variance of the estimator for $N_\ell = 4096$ will approximately be 1/256th of the corresponding variance when $N_\ell = 16$. Therefore, in order to obtain a fair comparison and useful plots

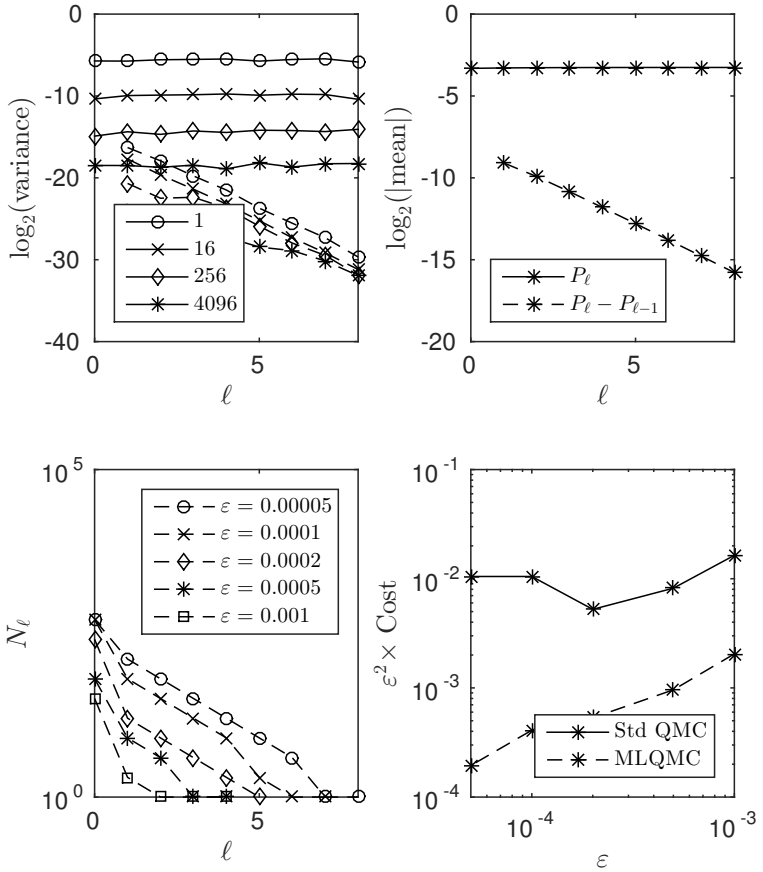

Figure 4.3: European call option with Sobol' sequence

of the variances of $\widehat{P}_\ell$ and $\widehat{P}_\ell - \widehat{P}_{\ell-1}$ for various N_ℓ, we have multiplied the variance estimates by the number of low-discrepancy points to correct for this difference in the denominator.

The solid lines results show that using quasi-Monte Carlo on its own already gives significant improvements over plain Monte Carlo. The dashed lines show that combining QMC with a multilevel approach gives additional improvements. This is particularly clear on the coarsest levels, as the benefit decreases on the finest levels. However, most of the computational cost of the multilevel method is on the coarsest levels and so overall we still obtain a major reduction in the computational cost.

The top right plot of both figures shows that $\mathbb{E}[\widehat{P}_\ell - \widehat{P}_{\ell-1}] = \mathcal{O}(h_\ell)$, which demonstrates that we obtain first order weak convergence as expected. On the bottom left plot, we can see the number of samples N_ℓ per level, which decreases as the theory predicts.

The bottom right plot of both figures shows the behaviour of $\varepsilon^2 C_m$ for various values of ε, where C_m is the computational cost of the multilevel QMC algorithm, defined as $C_m = 32 \sum_\ell 2^\ell N_\ell$, i.e. the total number of fine grid timesteps on all levels. In the standard QMC case, the cost C_s is the product of the number of samples required to achieve the desired variance and the number of timesteps. For the standard QMC, we see that $\varepsilon^2 C_s$ is roughly constant. However, it is clearly increasing for the multilevel case, both for lattice rules and Sobol' points. To have a better idea of the order of complexity, we have run a simple (logarithmic) regression on the cost to find the power of ε which best describes the cost. Figure 4.4 presents our results in the case of Sobol' points and a Brownian bridge construction.

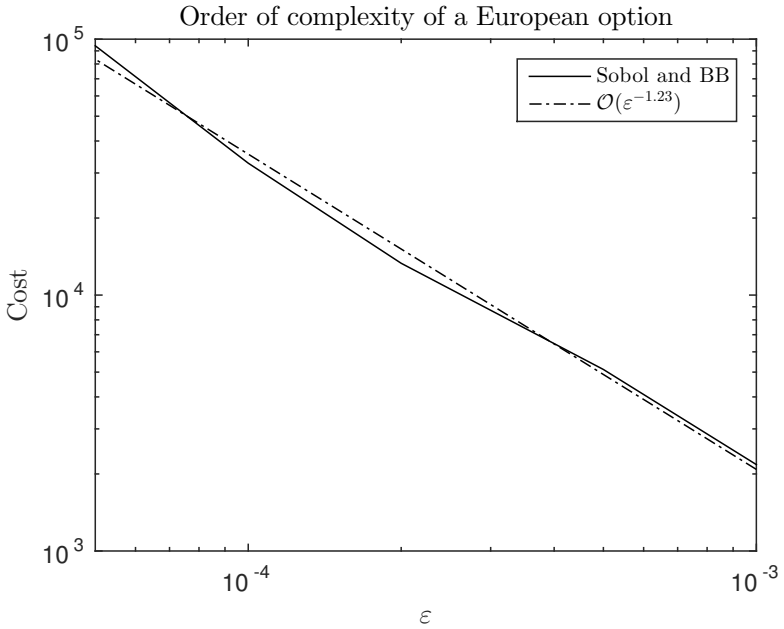

Figure 4.4: Regression on the order of complexity of a European call

Regression gives a cost which is close to $\mathcal{O}(\varepsilon^{-1.23})$. This is near-optimal, as in the best case the error is inversely proportional to the number of points, and therefore at best inversely proportional to the computational cost. Furthermore, it almost gives an improvement of two orders of complexity over the plain Monte Carlo method.

Brownian bridge vs PCA

For each option considered, we will compare the Brownian bridge construction and a hybrid PCA method based on Fast Fourier Transforms, which we now briefly describe.

Consider a scalar Brownian motion $W(t)$ and times $0 \leq t_1 < t_2$. Classical results in stochastic calculus ([Gla03], §3.1.1) imply that for any $t \in (t_1, t_2)$ and given $W(t_1)$ and $W(t_2)$, the random variable $W(t)$ is normally distributed with

$$\begin{aligned}\mathbb{E}[W(t)] &= W(t_1) + \frac{t - t_1}{t_2 - t_1}\left(W(t_2) - W(t_1)\right), \\ \mathbb{V}[W(t)] &= \frac{(t_2 - t)(t - t_1)}{t_2 - t_1}.\end{aligned}$$

Now, we further specify that W satisfies $W(0) = 0$, we set $W_N = \sqrt{T}\, Z_1$, and we consider a time grid $0 = t_0 < t_1 < t_2 < \cdots < t_{N-1} < t_N = 1$, so that $W_n = W(n/N) = W(t_n)$. By the results above, it is easy to show that the covariance matrix Ω for the discrete Brownian values W_n is given by $\Omega_{ij} = \min\{t_i, t_j\} - t_i t_j$. In particular, we can compute Ω^{-1}, and so the eigenvalues and unit eigenvectors of Ω are

$$\begin{aligned}\lambda_i &= \frac{1}{4N}\left(\sin\left(\frac{i\pi}{2N}\right)\right)^{-2}, \\ (v_i)_j &= \frac{2}{\sqrt{2N}}\sin\left(\frac{ij\pi}{N}\right),\end{aligned}$$

respectively, where $i, j = 1, \ldots, N-1$. Now in the hybrid construction, the discrete Brownian values W_n are defined for each n as

$$W_n = \frac{n}{N}W_N + \sum_{i=1}^{N-1} Z_{i+1}\sqrt{\lambda_i}(v_i)_n,$$

where λ_i is an eigenvalue of Ω, $(v_i)_n$ the nth component of the corresponding unit eigenvector, and the Z_i are independent standard normal random variables. The link with Fast Fourier Transforms is made explicit if we define $a_i = \frac{2}{\sqrt{2N}} Z_{i+1}\sqrt{\lambda_i}$ for each i. Indeed, this leads to

$$W_n = \frac{n}{N}W_N + \sum_{i=1}^{N-1} a_i \sin\left(\frac{in\pi}{N}\right), \quad n = 1, \ldots, N-1.$$

We can then apply a discrete sine transform to the coefficients a_i and produce the discrete Brownian values $W_1, \ldots, W_{N-1}$ as required. The advantage of this hybrid method is that when N is a power of 2, this results in a much faster algorithm than a standard PCA construction.

In the case of a European call option, we obtain the following results.

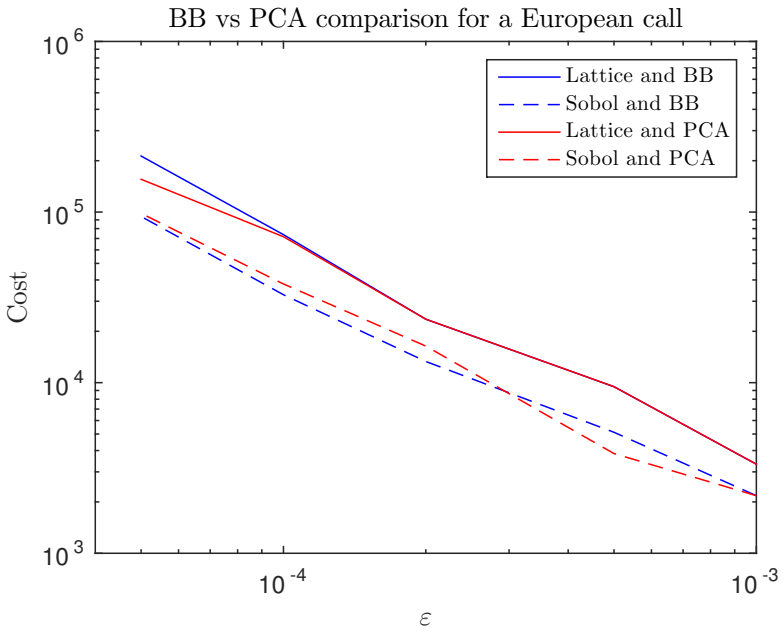

Figure 4.5: Comparison of BB vs PCA for a European call

The first observation here is that Sobol' sequences outperform rank-1 lattice rules for this type of option. The second observation is that BB and PCA produce similar results. The only difference seems to be that Brownian bridge is slightly more efficient for smaller values of ε while PCA is more performant on larger values of ε. In any case, it seems complicated to conclude anything on that behaviour from these results alone.

4.2 Asian option

We consider an Asian option with discounted payoff $P = \exp(-rT)(\overline{S} - K)^+$, where $\overline{S}$ is the (continuous) arithmetic average of S. As described by Giles and Waterhouse ([GW09], §6.2), we have for the fine path

$$\overline{\widehat{S}} = \frac{1}{T} \sum_{n=0}^{n_T-1} \left(\frac{1}{2} h(\widehat{S}_n + \widehat{S}_{n+1}) + b_n \Delta I_n \right),$$

where

$$\Delta I_n = \int_{t_n}^{t_{n+1}} (W(t) - W(t_n)) \, \mathrm{d}t - \frac{1}{2} h \Delta W$$

is a $\mathcal{N}(0, h^3/12)$ random variable, independent of ΔW. The approximation for the coarse path is similar, except that the values for ΔI_n are derived from the fine path values. Noting that

$$\begin{aligned}
&\int_{t_n}^{t_n+2h} (W(t) - W(t_n)) \, \mathrm{d}t - h(W(t_n + 2h) - W(t_n)) \\
&\quad = \int_{t_n}^{t_n+h} (W(t) - W(t_n)) \, \mathrm{d}t - \frac{1}{2} h(W(t_n + h) - W(t_n)) \\
&\quad + \int_{t_n+h}^{t_n+2h} (W(t) - W(t_n + h)) \, \mathrm{d}t - \frac{1}{2} h(W(t_n + 2h) - W(t_n + h)) \\
&\quad + \frac{1}{2} h(W(t_n + h) - W(t_n)) - \frac{1}{2} h(W(t_n + 2h) - W(t_n + h)),
\end{aligned}$$

we deduce that

$$\Delta I^{\mathrm{c}} = \Delta I^{\mathrm{f1}} + \Delta I^{\mathrm{f2}} + \frac{1}{2} h(\Delta W^{\mathrm{f1}} - \Delta W^{\mathrm{f2}}),$$

where ΔI^{c} is the value for the coarse timestep, ΔI^{f1}, ΔW^{f1} are the values for the first fine timestep, and ΔI^{f2}, ΔW^{f2} are the values for the second fine timestep.

Again, we first mention the existing computations for rank-1 lattice rules, and we then present our results for Sobol' sequences. As in the European case, we take $S(0) = 1$, $K = 1$, and parameters $T = 1$, $r = 0.05$, $\sigma = 0.2$.

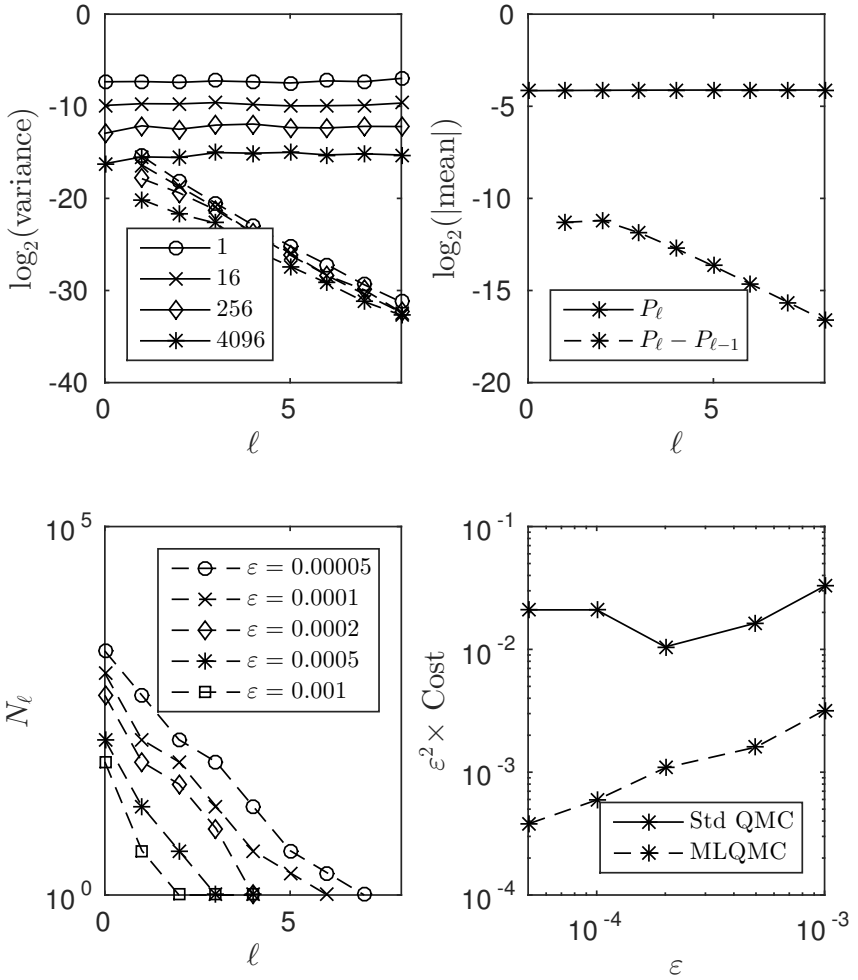

Figure 4.6: Asian call option with rank-1 lattice rule

The graphs of Figure 4.6 are similar to those in obtained for European options. More specifically, the top left plot shows that both the multilevel approach and the QMC element reduce the variance of the estimator. Furthermore, the benefits of the multilevel approach are more significant on the coarsest levels.

The top left graph of Figure 4.7 shows that on the coarsest levels, Sobol' sequences outperform rank-1 lattice rules, with the benefits decreasing across the levels. In addition, we see that $\varepsilon^2 \times \text{Cost}$ is increasing in the bottom right plots of both figures, but it is hard to quantify the exact computational cost.

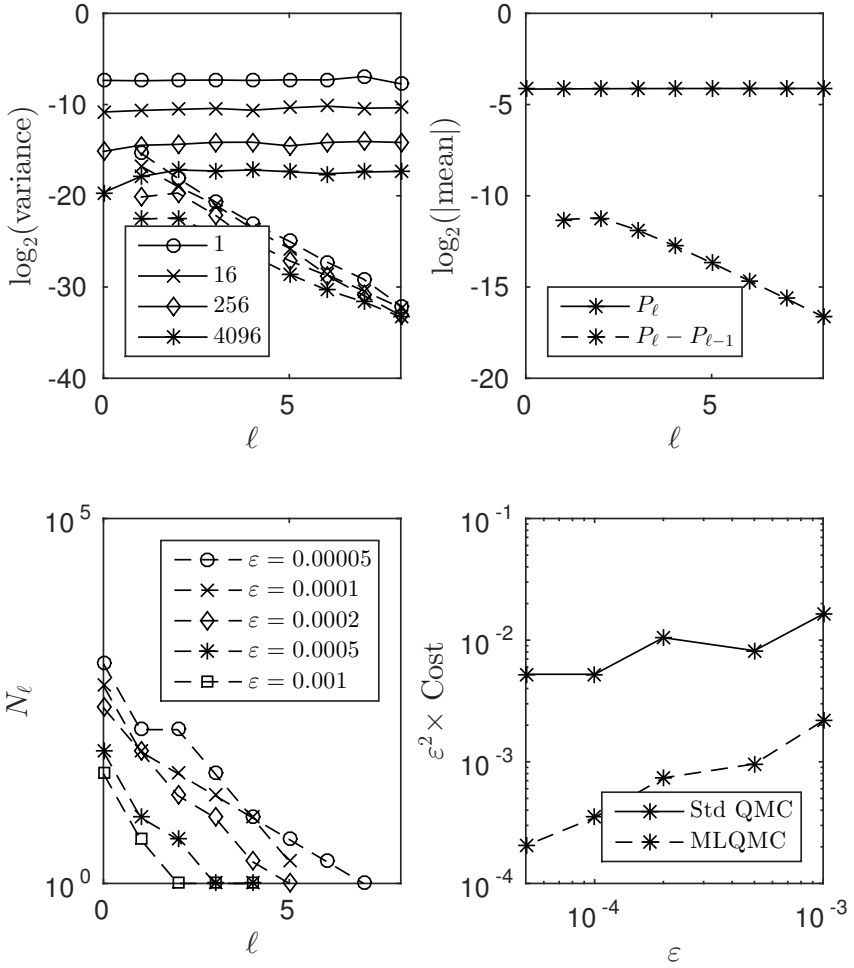

Figure 4.7: Asian call option with Sobol' sequence

Regression for the Asian option is given in Figure 4.8, and shows that the cost of our problem is approximately $\mathcal{O}(\varepsilon^{-1.12})$. Again this is near-optimal, and shows again the advantage of using QMC in the multilevel setting.

The BB vs PCA graph given in Figure 4.9 confirms our results that Sobol' sequences outperform rank-1 lattice rules. Furthermore, there is very little difference between the two methods described to generate Brownian values: indeed, Brownian bridge and PCA constructions lead to very similar results.

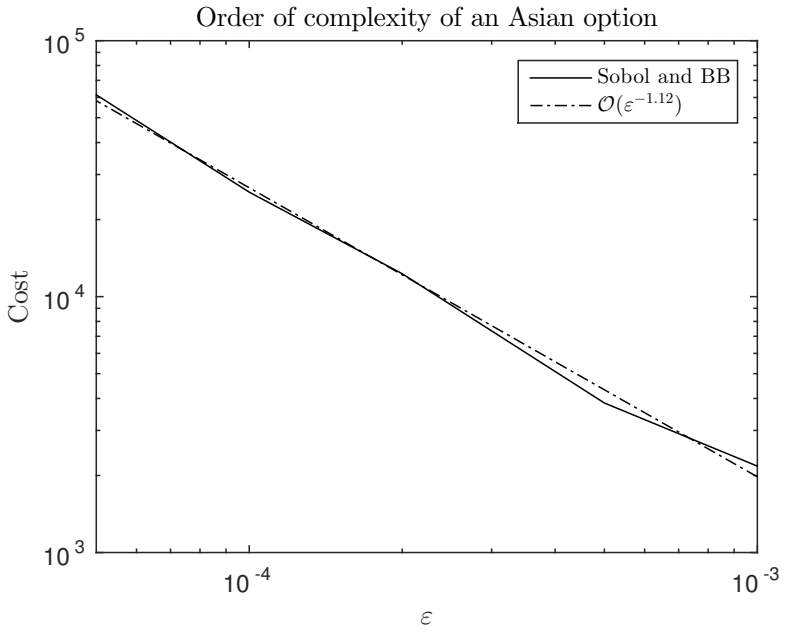

Figure 4.8: Regression on the order of complexity of an Asian option

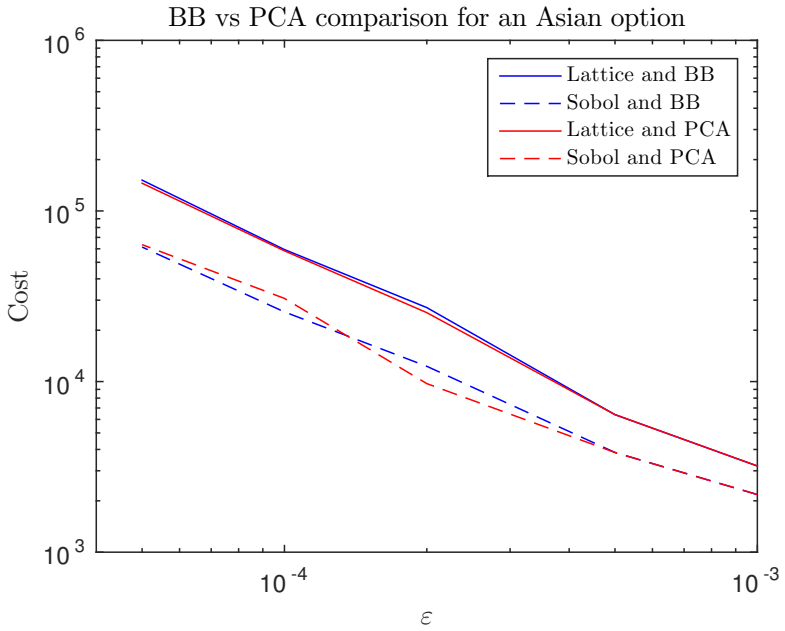

Figure 4.9: Comparison of BB vs PCA for an Asian option

4.3 Lookback option

We now consider a lookback option, with discounted payoff

$$P = \exp(-rT)\left(S(T) - \min_{0<t<T} S(t)\right).$$

For the fine path calculation, it is a standard result ([Gla03], §6.4) that the minimum value can be given as

$$\widehat{S}^{\mathrm{f}}_{n,\min} = \frac{1}{2}\left(\widehat{S}^{\mathrm{f}}_n + \widehat{S}^{\mathrm{f}}_{n+1} - \sqrt{\left(\widehat{S}^{\mathrm{f}}_{n+1} - \widehat{S}^{\mathrm{f}}_n\right)^2 - 2b_n^2 h \log U_n}\right),$$

where $U_n \sim \mathcal{U}(0,1)$ is a uniform random variable. We therefore obtain an approximation to $\min_{[0,T]} S(t)$ by taking the minimum over all timesteps, and therefore an approximation $\widehat{P}_\ell$ to the payoff.

For the coarse path calculation, $\widehat{P}_{\ell-1}$ is defined in a similar way, except that we used an interpolated midpoint in the Milstein discretisation ([GW09], §3). This leads to

$$\widehat{S}^{\mathrm{c}}_{m,\min} = \min\left\{\frac{1}{2}\left(\widehat{S}^{\mathrm{c}}_m + \widehat{S}^{\mathrm{c}}_{m+\frac{1}{2}} - \sqrt{\left(\widehat{S}^{\mathrm{c}}_{m+\frac{1}{2}} - \widehat{S}^{\mathrm{c}}_m\right)^2 - b_m^2 h \log U_{2m-1}}\right),\right.$$
$$\left.\frac{1}{2}\left(\widehat{S}^{\mathrm{c}}_{m+\frac{1}{2}} + \widehat{S}^{\mathrm{c}}_{m+1} - \sqrt{\left(\widehat{S}^{\mathrm{c}}_{m+1} - \widehat{S}^{\mathrm{c}}_{m+\frac{1}{2}}\right)^2 - b_m^2 h \log U_{2m}}\right)\right\}.$$

Note that the uniform random variables U_{2m-1} and U_{2m} in the coarse path calculation are re-used from the fine path calculation. This ensures that the minimum from the coarse path is close to the minimum from the fine path, giving a low variance for $\widehat{P}_\ell - \widehat{P}_{\ell-1}$.

Figures 4.10 and 4.11 give the numerical results for $S(0) = 1$, $T = 1$, $r = 0.05$, and $\sigma = 0.2$.

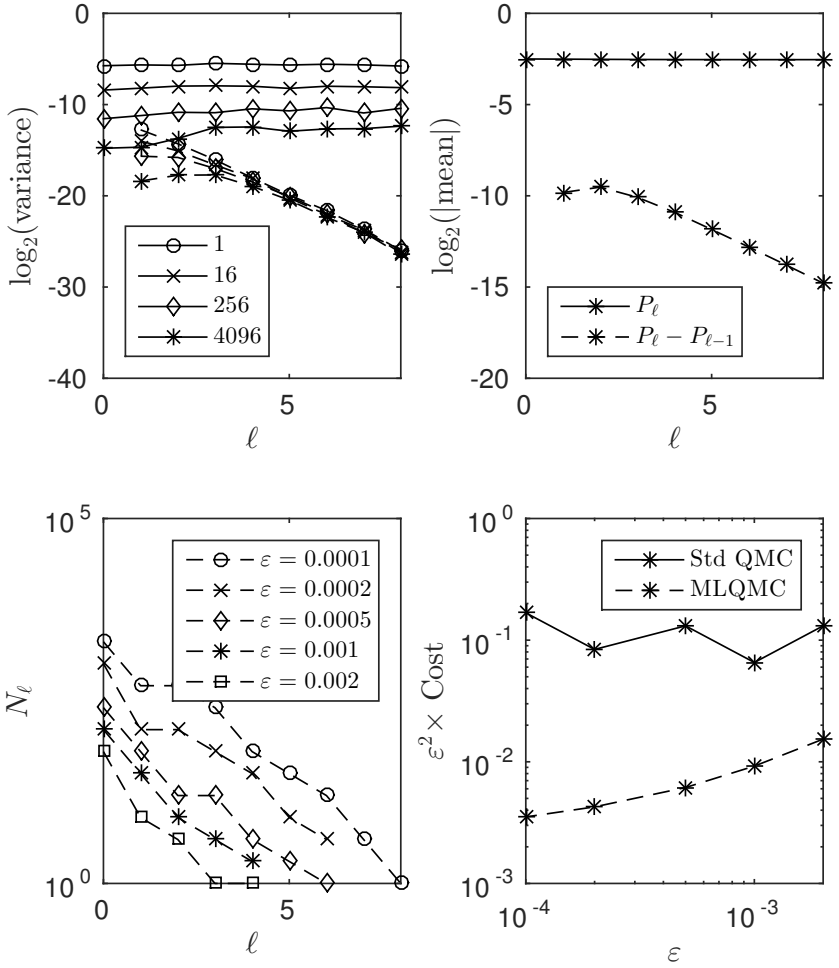

Figure 4.10: Lookback option with rank-1 lattice rule

An interesting observation from the top left graph in Figure 4.10 above is that the benefit of combining QMC with the multilevel method only appears on the two or three coarsest levels (see dashed lines), as compared to the first five or six levels in the previous two cases. In addition, when looking at QMC on its own (solid lines), the benefit also decreases across the levels, whereas it was approximately constant for European and Asian options. However, rank-1 lattice rules still provide an improvement over a plain MLMC algorithm, as will be clear with Figures 4.12 and 4.13.

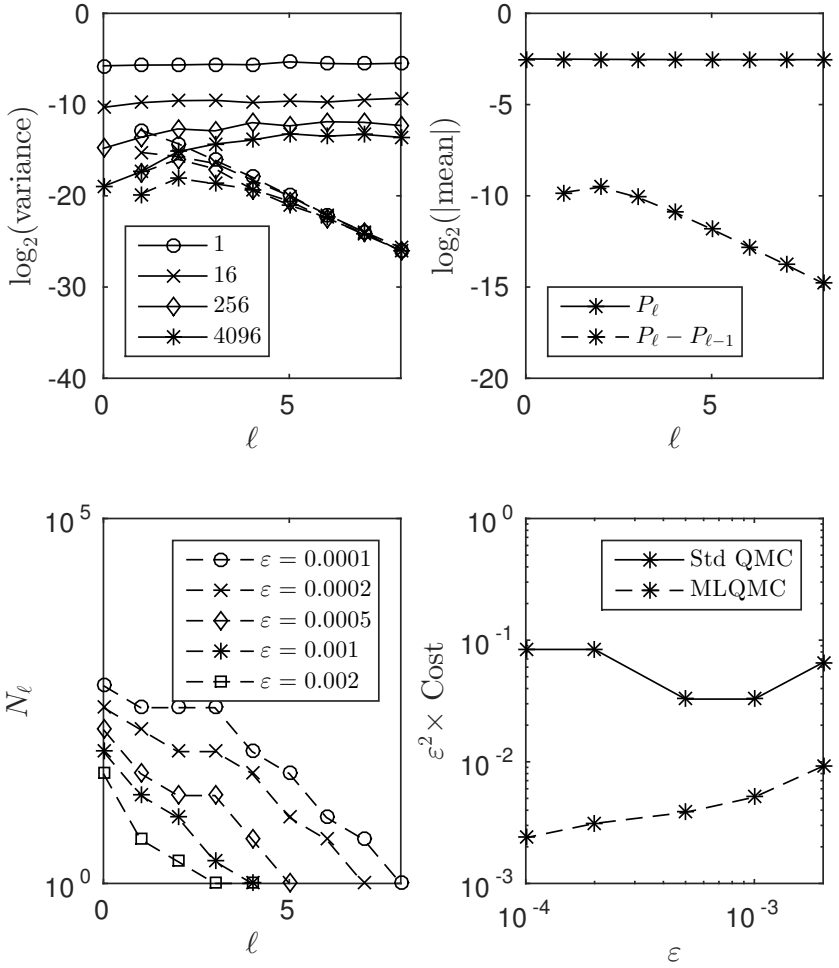

Figure 4.11: Lookback option with Sobol' sequence

The results in Figure 4.11 are qualitatively similar to those in Figure 4.10. As in the previous cases, we perform a regression on the order of complexity, and we compare the BB and PCA constructions.

Regression for the lookback option is given in Figure 4.12 below, and shows that the cost of our problem is approximately $\mathcal{O}(\varepsilon^{-1.58})$ for the lookback option. Although this is not as low as the European case, it is still much lower than the $\mathcal{O}(\varepsilon^{-2})$ lower bound achieved without QMC component.

The BB vs PCA graph given in Figure 4.13 confirms our results that Sobol' sequences outperform rank-1 lattice rules. Furthermore, Brownian bridge and PCA constructions once again lead to similar results.

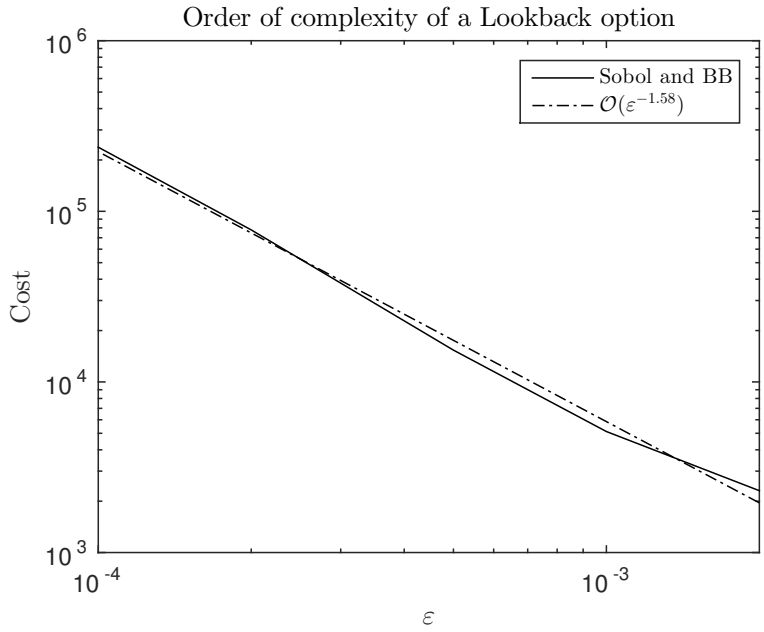

Figure 4.12: Regression on the order of complexity of a lookback option

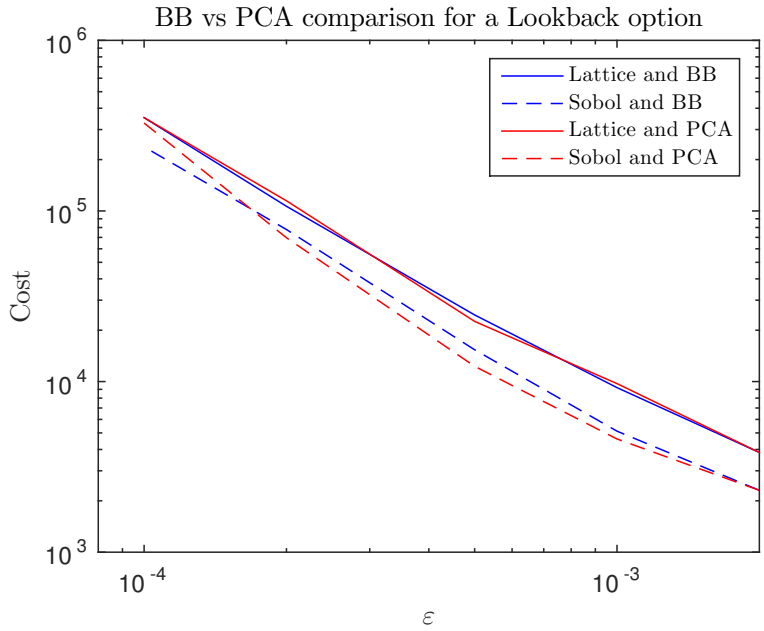

Figure 4.13: Comparison of BB vs PCA for a lookback option

4.4 Barrier option

For the barrier option case, we consider a down-and-out call with discounted payoff

$$P = \exp(-rT)(S(T) - K)^+ \mathbf{1}_{\tau > T},$$

where $\mathbf{1}_{\tau>T}$ denotes an indicator function which is 1 if $\tau > T$ and 0 otherwise, and where τ is the barrier crossing time, i.e. $\tau = \inf_{t>0} S(t) < B$.

For the fine path simulation, the conditional expectation of the payoff ([Gla03], §6.4) can be expressed as

$$\exp(-rT)(\widehat{S}^{\mathrm{f}}_{n_T} - K)^+ \prod_{n=0}^{n_T-1} \widehat{p}_n,$$

where $\widehat{p}_n$ denotes the probability that the interpolated path did not cross the barrier at timestep n, and which equals

$$\widehat{p}_n = 1 - \exp\left(\frac{-2(\widehat{S}^{\mathrm{f}}_n - B)^+(\widehat{S}^{\mathrm{f}}_{n+1} - B)^+}{b_n^2 h}\right).$$

For the coarse path calculation, we use the midpoint trick as for the lookback option case. Given the value $\widehat{S}^{\mathrm{c}}_{m+\frac{1}{2}}$ at each timestep, the probability that the Brownian interpolation path does not cross the barrier during the mth (coarse) timestep is

$$\widehat{p}^{\,\mathrm{c}}_m = \left\{1 - \exp\left(\frac{-2(\widehat{S}^{\mathrm{c}}_m - B)^+(\widehat{S}^{\mathrm{c}}_{m+\frac{1}{2}} - B)^+}{b_m^2 h}\right)\right\} \times \left\{1 - \exp\left(\frac{-2(\widehat{S}^{\mathrm{c}}_{m+\frac{1}{2}} - B)^+(\widehat{S}^{\mathrm{c}}_{m+1} - B)^+}{b_m^2 h}\right)\right\}.$$

We give the numerical results with $S(0) = 1$, $K = 1$, $B = 0.85$ and the usual parameters $T = 1$, $r = 0.05$, $\sigma = 0.2$ in Figures 4.14 and 4.15.

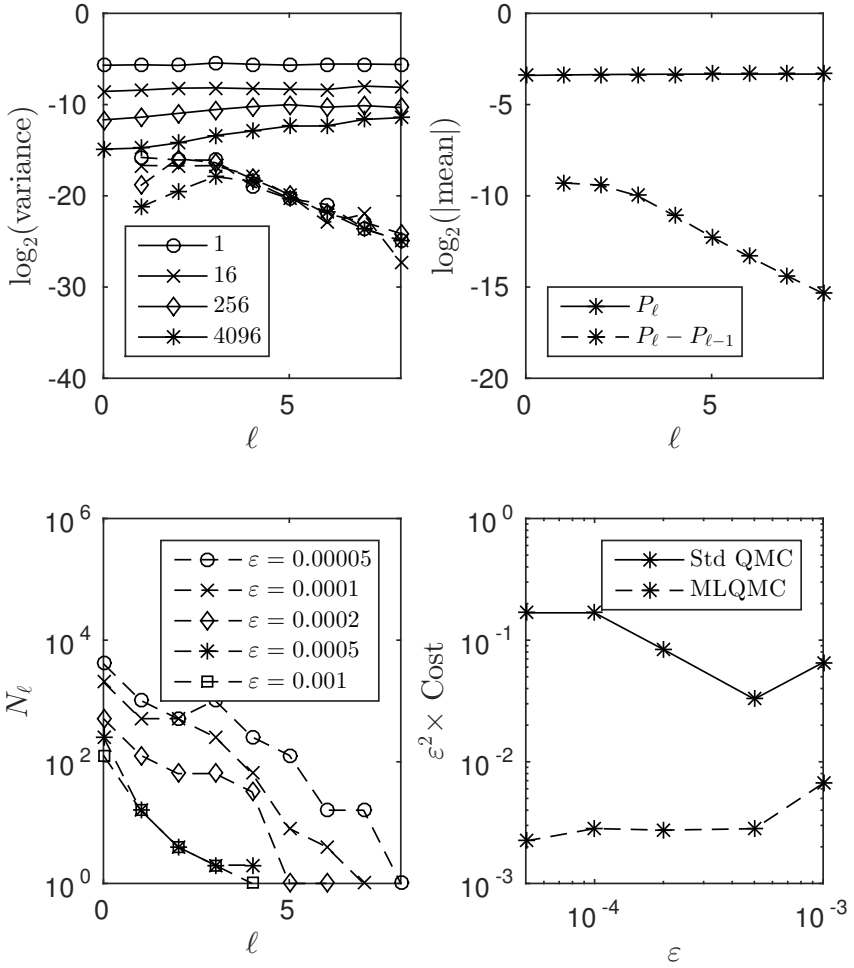

Figure 4.14: Barrier option with rank-1 lattice rule

In the top left graph of Figure 4.14, the solid lines show that the variance of $\widehat{P}_\ell$ (solid lines) decreases at a slower rate than before, displaying a similar behaviour than for lookback options. Worse, the dashed lines show that on some levels, there is at least one occurrence where increasing the number of points increases the variance of $\widehat{P}_\ell - \widehat{P}_{\ell-1}$ (for example at level $\ell = 7$). The same qualitative results can be observed in Figure 4.15 for Sobol' sequences.

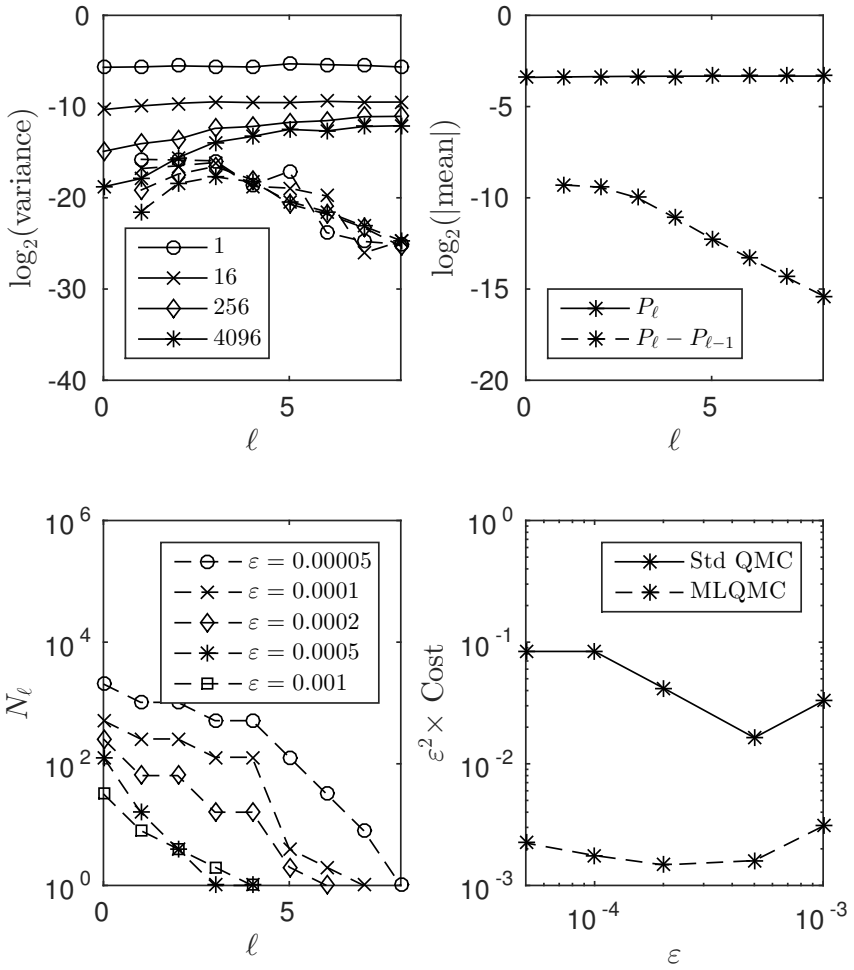

Figure 4.15: Barrier option with Sobol' sequence

The graph of the regression for the barrier option is given in Figure 4.16. To summarise, it shows that the cost of our problem is approximately $\mathcal{O}(\varepsilon^{-1.92})$ for the barrier option. This is worse than the previous cases, but still represents an improvement over the plain techniques considering that it is (relatively) more complicated to price a barrier option than a simple European option. Furthermore, this order of complexity is consistent with the qualitative results from Figure 4.15. Indeed, we see on the bottom right plot that for the MLQMC algorithm, the product of ε^2 with the computational cost is almost constant, indicating that we should not expect a cost much lower than $\mathcal{O}(\varepsilon^{-2})$.

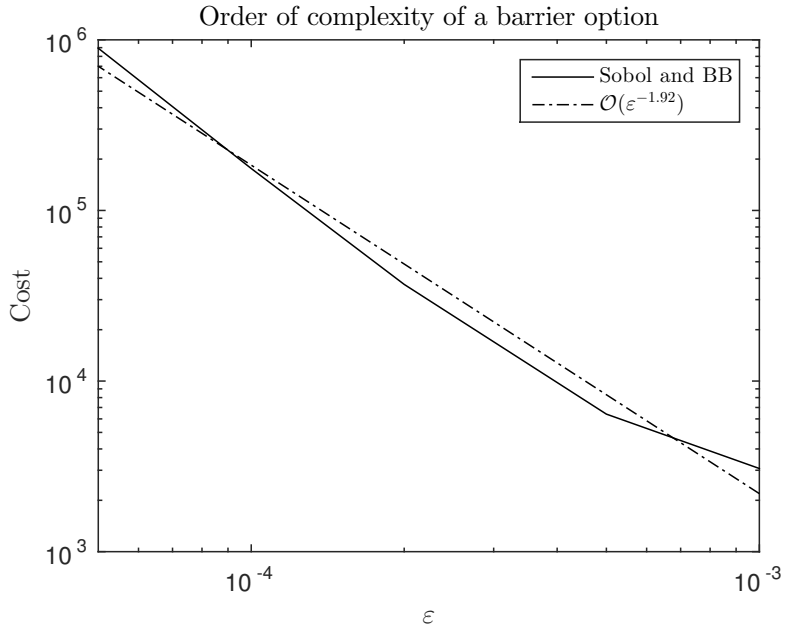

Figure 4.16: Regression on the order of complexity of a barrier option

The BB vs PCA graph given in Figure 4.17 is similar to what we have seen so far. On one hand it shows that Sobol' sequences also outperform rank-1 lattice rules for barrier options. This is particularly clear for larger values of ε, with the difference in the results obtained from the two low-discrepancy sequences diminishing as ε decreased. On the other hand, it confirms that Brownian bridge and PCA methods lead to similar results.

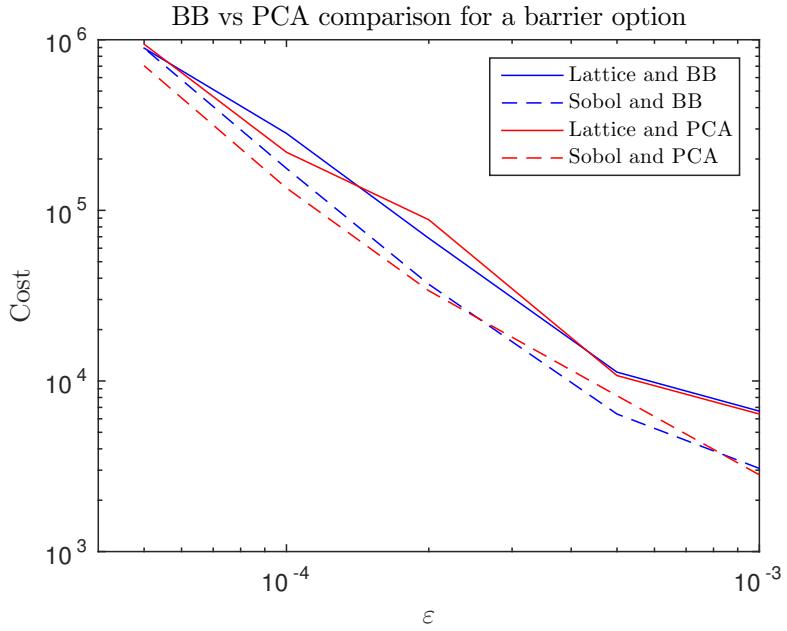

Figure 4.17: Comparison of BB vs PCA for a barrier option

4.5 Digital option

We consider a digital option with discounted payoff

$$P = \exp(-rT)\mathbf{1}_{S(T)>K},$$

where again $\mathbf{1}_{S(T)>K}$ denotes an indicator function which is 1 if the terminal value $S(T)$ is greater than the strike K, and 0 otherwise.

We follow the same procedure as Giles in ([Gil08a]), smoothing the payoff using the technique of conditional expectation ([Gla03], §7.2.3) in which the path calculations are terminated one timestep before the final time T. The method is described in more details in ([GW09], §6.5).

The numerical results with $S(0) = 1$, $K = 1$, $T = 1$, $r = 0.05$, and $\sigma = 0.2$ are given in Figures 4.18 and 4.19.

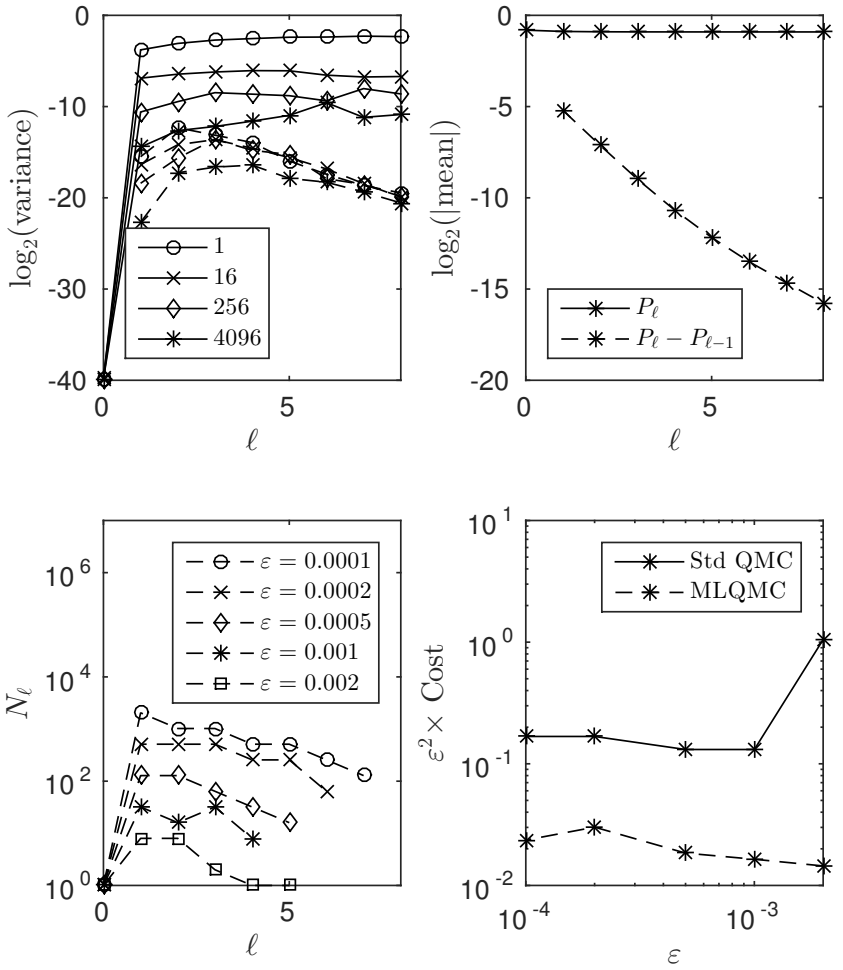

Figure 4.18: Digital option with rank-1 lattice rule

On the top left plot of Figure 4.18, we see that combining the multilevel method with quasi-Monte Carlo techniques significantly reduces the variance of $\widehat{P}_\ell - \widehat{P}_{\ell-1}$ on the coarsest levels. However, the solid lines results suggest that this might be due to the multilevel approach rather than the QMC component. This can be seen for example at level $\ell = 6$, where increasing the number of quasi-uniform points from $N_\ell = 256$ to $N_\ell = 4096$ has no effect on the variance of $\widehat{P}_\ell$. This qualitative observation, which attempts to distinguish the benefits provided by the multilevel and QMC components, will be discussed further when analysing Figure 4.21.

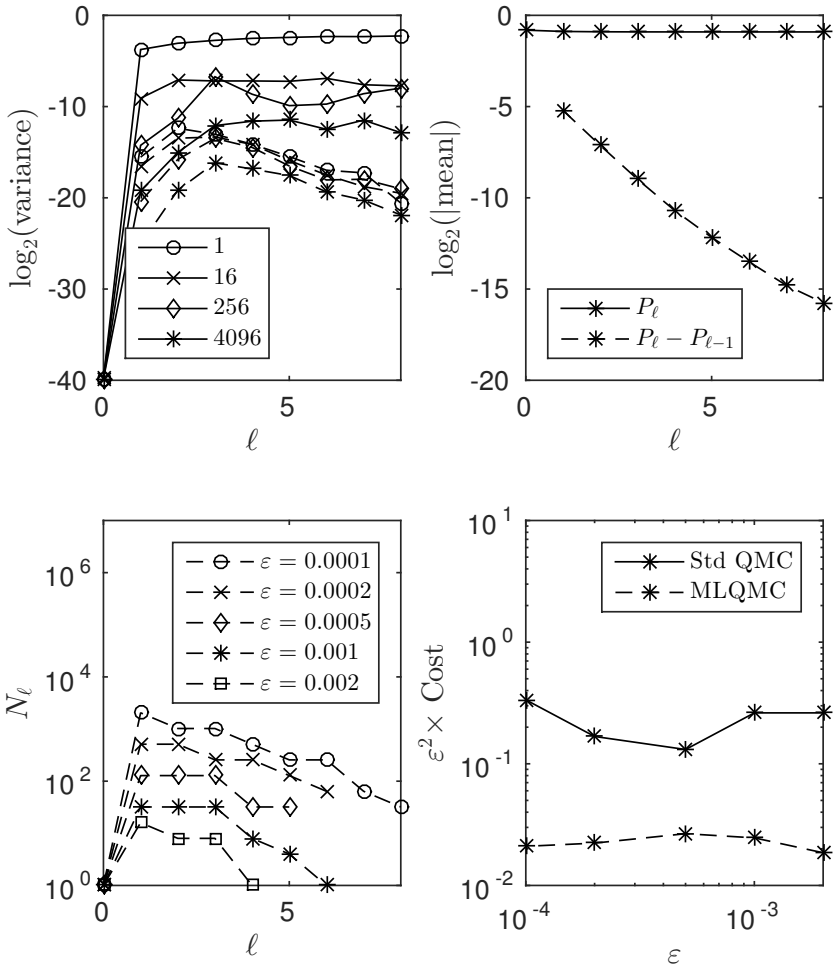

Figure 4.19: Digital option with Sobol' sequence

The graph of the regression for the digital option is presented in Figure 4.20. To summarise, it shows that the cost of our problem is approximately $\mathcal{O}(\varepsilon^{-2.01})$, which is our worst cost out of all applications considered. As mentioned previously, this is consistent with what Giles and Waterhouse report in ([GW09], §6.5) where they mention that the QMC element does not add significant benefits for this option.

The BB vs PCA graph given in Figure 4.21 is perhaps even more surprising. Indeed, it shows that lattice rules outperform Sobol' sequences for some values of ε. This might be a consequence of the complexity of the payoff, but we cannot draw conclusions from this graph alone.

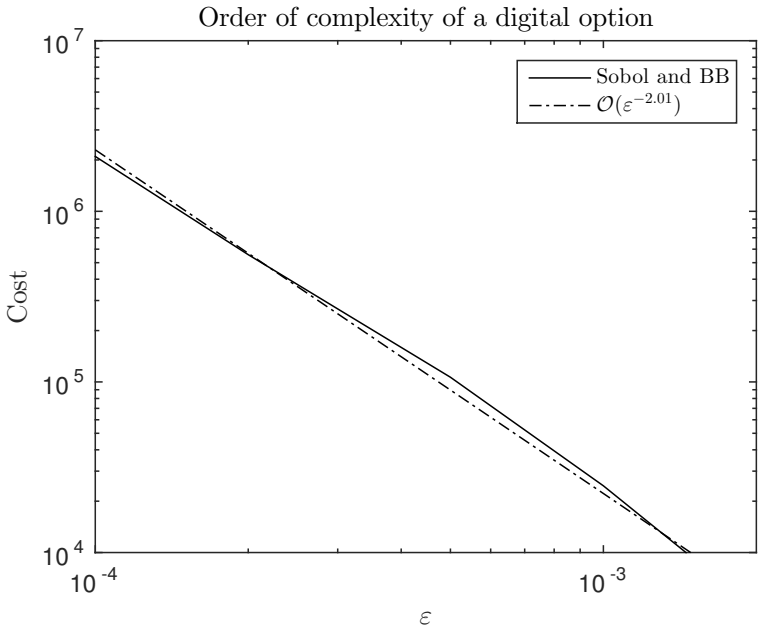

Figure 4.20: Regression on the order of complexity of a digital option

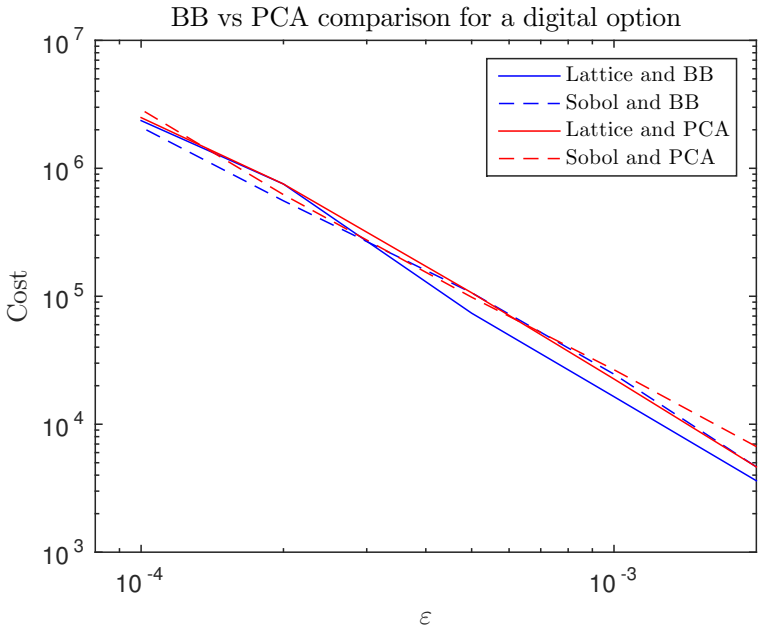

Figure 4.21: Comparison of BB vs PCA for a digital option

4.6 Dimension reduction

In this section, we analyse the importance of dimension reduction in the MLQMC algorithm in light of the observations from Section 2.4.4.

One of the first steps in a quasi-Monte Carlo method is to convert a d-dimensional vector of QMC points to a vector Z of quasi-normals. Suppose for simplicity that d is a power of 2, as it is the case in the multilevel algorithm. Following an idea of Giles, we propose to generate Z as follows:

$$Z = \left(\Phi^{-1}(u_1), \ldots, \Phi^{-1}(u_{d/2}), \Phi^{-1}(v_1), \ldots, \Phi^{-1}(v_{d/2})\right),$$

where (u_i) is a low-discrepancy sequence and (v_i) a sequence of pseudo-random numbers. We then apply a Brownian bridge construction to Z and follow the MLQMC algorithm described in Section 3.2. Based on existing empirical results ([DKS13; SAKK11]) we have chosen to implement this idea when the dimension exceeds a threshold α, where $\alpha \in \{8, 16, 32, 64\}$. Figure 4.6 shows the behaviour of the computational cost when the threshold α varies, using Sobol' points and with $\varepsilon = 0.0001$. We chose a relatively small ε to ensure that the multilevel algorithm generates sufficiently high-dimensional Sobol' sequences. Note that $\alpha = 0$ in the plot corresponds to results obtained without this strategy.

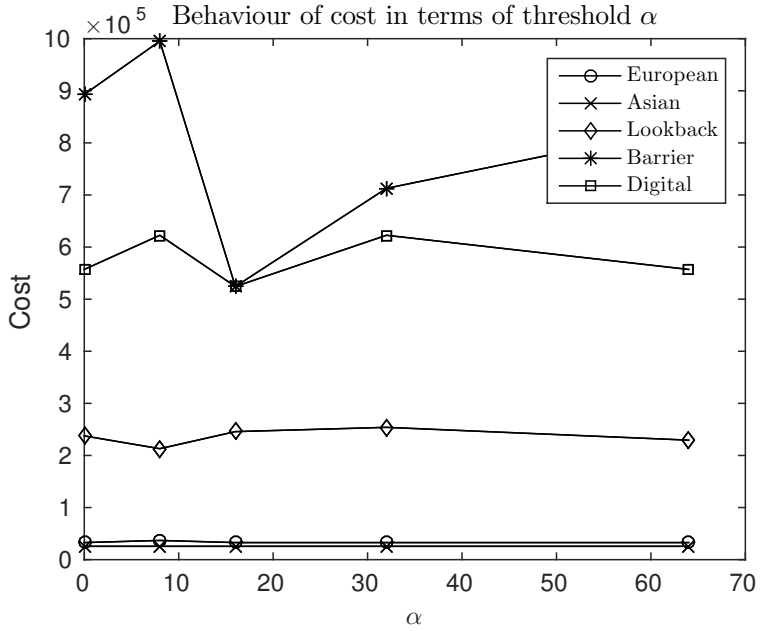

Figure 4.22: Cost in terms of threshold α

We observe that for the European and Asian options, the strategy has little to no impact. This is probably because the multilevel algorithm converges faster, and therefore does not need to generate as many high-dimensional sequences as with the other options.

A second observation is that for the three remaining cases, there exists a threshold α which is either 8 or 16 such that the cost of the MLQMC computation is minimised. Furthermore, the cost displays a shape similar to a smile. A possible explanation for this is that there exists a tradeoff between the loss of uniformity caused by the use of pseudo-random numbers and the benefit they incur by reducing the effective dimension of the problem.

Further work needs to be carried to determine how robust the smile shape is, and if there is an optimal threshold for each option. If confirmed however, these results would potentially simplify the implementation of Brownian bridge constructions on Graphics Processing Units (GPUs).

5 Conclusions and future work

In this paper, we have shown that Sobol' sequences often outperform rank-1 lattice rules in the context of option pricing. This is the case with all options that we considered, except the digital option, for which the use of quasi-Monte Carlo methods does not seem to add any benefit. In the case of a Lipschitz European payoff, we found that with Sobol' points, the MLQMC algorithm led to a computational cost of approximately $\mathcal{O}(\varepsilon^{-1.25})$, an improvement over the previous best known results of $\mathcal{O}(\varepsilon^{-1.5})$.

A first extension to our work would be to develop and test a MLQMC application of basket options using Sobol' sequences. In this case, a challenge might be the limitation on the number of dimensions imposed by Matlab ($d = 1111$). For a basket of 10 options, this would indeed restrict the number of timesteps h in the path simulation to 111, and actually to 64 as h is a power of two in the multilevel formulation.

Perhaps a more challenging extension would be to generalise the MLQMC algorithm for multidimensional SDEs. A main issue in this case is that the Milstein path discretisation requires the simulation of Lévy areas ([Gla03; KP92]), which is not yet fully understood. Initial results have already been obtained in the MLMC case by Giles and Szpruch ([GS14]) who developed an antithetic multilevel correction estimator to avoid the simulation of Lévy areas for European and Asian options. Further work is required to achieve computational costs of $\mathcal{O}(\varepsilon^{-2})$ for more complex payoffs such as digital, barrier and lookback options, and also to analyse whether a quasi-Monte Carlo treatment would add further benefits as it does in the scalar case.

In terms of practical applications, a main direction for future research would be to further investigate the ideas of dimension reduction mentioned in Section 4.6. In particular, it might be possible to identify an optimal threshold for the dimension at which the vector of uniforms is split into low-discrepancy and pseudo-random points. The initial results we obtained in that direction suggest that additional work could simplify the implementation of Brownian bridge constructions on GPUs.

Appendix

We attach the main functions used to build the multilevel quasi-Monte Carlo algorithm. The implementation of the test cases and the plots can be found in `multilevelqmc`, while the implementation of the QMC component is in the function `mlmc_l` which is the level ℓ estimator.

```
%
% This code implements a quasi-Monte Carlo version
% of the multilevel method of Giles (2008).
% It uses geometric Brownian motion to price
% options with various payoffs.
%
% opt = integer from 1 to 5, with
%  1 = European option
% 2 = Asian option
% 3 = Lookback option
% 4 = Digital option
%  5 = Barrier option
%
% randomisation = integer from 1 to 3, with
% 1 = rank-1 lattice rule
% 2 = Sobol' sequence
% 3 = mixture sequence (see paper)
%


function [mlmc_cost] = multilevelqmc(opt,randomisation)

clearvars -EXCEPT opt randomisation

global option z_cbc method

nvert = 2;
method = randomisation;

load z_d4096_m15_gf1
z_cbc = z;

if method ==1
    rtitle = 'Lattice_rule';
elseif method == 2
    rtitle = 'Sobol_sequence';
elseif method ==3
    rtitle = 'Halton_sequence';
```

```
else
    rtitle = 'Other';
end

for option = opt

    if option==1
        stitle = 'European';
    elseif option==2
        stitle = 'Asian';
    elseif option==3
        stitle = 'Lookback';
    elseif option==4
        stitle = 'Digital';
    elseif option==5
        stitle = 'Barrier';
    end
    disp(stitle)

    rand('state',0);
    randn('state',0);

    %
    % first, convergence tests
    %

    if option==4
        L = 0:8;
    else
        L = 0:8;
    end

    for m = 1:4
        disp(sprintf('m = %d',m))
        N = 16^(m-1);

        for l = L
            Ns = max(128,min(4096,2^(18-3*m-l))); %number of QMC families
            sums = mlmc_l(l,Ns,N);
            del1( l+1) = sums(3)/Ns;
            del2( l+1) = sums(1)/Ns;
            var1(m,l+1) = max(1e-12, (sums(4)/Ns-(sums(3)/Ns)^2)*N);
            var2(m,l+1) = max(1e-12, (sums(2)/Ns-(sums(1)/Ns)^2)*N);
        end
    end
```

```
84
85      %
86      % second, mlmc complexity tests
87      %
88
89      rand('state',0);
90      randn('state',0);
91
92      if option==3 | option==4
93          Eps = [ 0.002 0.001 0.0005 0.0002 0.0001 ];
94      else
95          Eps = [ 0.001 0.0005 0.0002 0.0001 0.00005 ];
96      end
97
98      % calculate cost and savings
99      savings = zeros(length(Eps),1);
100     mlmc_cost = zeros(length(Eps),1);
101     std_cost= zeros(length(Eps),1);
102
103     for i = 1:length(Eps)
104         eps = Eps(i);
105         Ns = 32; % number of sets/families for QMC
106         [P, Nl] = mlmc(eps,Ns,@mlmc_l);
107         l = length(Nl)-1;
108         mlmc_cost(i) = sum(Nl.*2.^(0:l));
109         Nls{i} = Nl/Ns;
110
111         [P, Nl] = mc(eps,Ns,l,@mlmc_l);
112         std_cost(i) = Nl*2^l;
113
114         disp(sprintf('mlmc_cost = %d, std_cost = %d, savings = %f\n',...
115             mlmc_cost(i),std_cost(i),std_cost(i)/ mlmc_cost(i)))
116     end
117
118     %l = max(l,4);
119     l = 8;
120
121     %
122     % plot figures
123     %
124
125     set(0,'DefaultAxesFontSize',12)
126     figure('name',rtitle); pos=get(gcf,'pos'); pos(4)=pos(4)*0.75*nvert; set(gcf,'pos',pos);
127     set(groot, 'DefaultTextInterpreter', 'LaTex');
128     set(groot, 'DefaultLegendInterpreter', 'LaTex');
```

```
129
130     set(0,'DefaultAxesColorOrder',[0 0 0]);
131     set(0,'DefaultAxesLineStyleOrder','-o|-x|-d|-*|--o|--x|--d|--*');
132
133     subplot(nvert,2,1)
134     plot(L,log(var1)'/log(2),L(2:end),log(var2(:,2:end))'/log(2));
135     xlabel('$\ell$','Interpreter','LaTex'); ylabel('$\log_2$(variance)','Interpreter','LaTex');
            %title(stitle)
136     legend('$1$','$16$','$256$','$4096$','Location','SouthWest')
137     axis([0 l -40 0])
138
139     set(0,'DefaultAxesLineStyleOrder','-*|--*')
140
141     subplot(nvert,2,2)
142     plot(L,log(abs(del1))/log(2),L(2:end),log(abs(del2(2:end)))/log(2))
143     xlabel('$\ell$','Interpreter','LaTex'); ylabel('$\log_2 (\left| \textrm{mean} \right|)$','
            Interpreter','LaTex'); %title(stitle)
144     legend('$P_\ell$','$P_\ell- P_{\ell-1}$','Location','SouthWest','Interpreter','LaTex')
145     axis([0 l -20 0])
146
147     if nvert==1
148         print('-deps2c',sprintf('mlmc_gbm%da.eps',option))
149         figure; pos=get(gcf,'pos'); pos(4)=pos(4)*0.75; set(gcf,'pos',pos);
150     end
151
152     set(0,'DefaultAxesLineStyleOrder','--o|--x|--d|--*|--s');
153
154
155     if(nvert==3)
156         subplot(nvert,2,3)
157         set(0,'DefaultAxesLineStyleOrder','--o|--x|--d|--*');
158         plot(log(16.^(0:3))/log(2),log(var1(:,2))/log(2),log(16.^(0:3))/log(2),log(var1(:,3))/
                log(2),log(16.^(0:3))/log(2),log(var1(:,4))/log(2),...
159              log(16.^(0:3))/log(2),log(var1(:,5))/log(2));
160         legend('l=1', 'l=2', 'l=3', 'l=4');
161         xlabel('log_2(m)');
162         ylabel('log_2 var (P_\ell)');
163         %{
164     set(0,'DefaultAxesLineStyleOrder','-o|-x|-d|-*');
165     plot(log(16.^(0:3))/log(2),log(var2(:,6))/log(2),log(16.^(0:3))/log(2),log(var2(:,7))/log(2)
            ,...
166          log(16.^(0:3))/log(2),log(var2(:,8))/log(2),log(16.^(0:3))/log(2),log(var1(:,9))/log
                (2));
167     legend('l=5', 'l=6', 'l=7', 'l=8');
168     xlabel('log_2 (m)');
```

```
169     ylabel('log_2 var(P_l-P_{l-1})');
170         %}
171         subplot(nvert,2,4)
172         set(0,'DefaultAxesLineStyleOrder','--o|--x|--d|--*');
173         % for a given level, variance in terms of number of points
174         plot(log(16.^(0:3))/log(2),log(var1(:,6))/log(2),log(16.^(0:3))/log(2),log(var1(:,7))/
                log(2),log(16.^(0:3))/log(2),log(var1(:,8))/log(2),...
175             log(16.^(0:3))/log(2),log(var1(:,9))/log(2));
176         legend('l=5', 'l=6', 'l=7', 'l=8');
177         xlabel('log_2(m)');
178         ylabel('log_2 var (P_\ell)');
179     end
180
181     subplot(nvert,2,2*nvert-1)
182     semilogy(0:length(Nls{5})-1,Nls{5}, ...
183         0:length(Nls{4})-1,Nls{4}, ...
184         0:length(Nls{3})-1,Nls{3}, ...
185         0:length(Nls{2})-1,Nls{2}, ...
186         0:length(Nls{1})-1,Nls{1});
187     xlabel('$\ell$','Interpreter','LaTex'); ylabel('$N_\ell$','Interpreter','LaTex'); %title(
            stitle)
188     if option==3 | option==4
189         legend('$\varepsilon=0.0001$','$\varepsilon=0.0002$','$\varepsilon=0.0005$',...
190             '$\varepsilon=0.001$','$\varepsilon=0.002$',1,'Interpreter','LaTex')
191     else
192         legend('$\varepsilon=0.00005$','$\varepsilon=0.0001$','$\varepsilon=0.0002$',...
193             '$\varepsilon=0.0005$','$\varepsilon=0.001$',1,'Interpreter','LaTex')
194     end
195     if option==4
196         axis([0 1 1 1e7])
197     elseif option==5
198         axis([0 1 1 1e6])
199     else
200         axis([0 1 1 1e5])
201     end
202
203     set(0,'DefaultAxesLineStyleOrder','-*|--*')
204
205     subplot(nvert,2,2*nvert)
206     loglog(Eps',Eps'.^2.*std_cost,Eps',Eps'.^2.*mlmc_cost)
207     xlabel('$\varepsilon$','Interpreter','LaTex'); ylabel('$\varepsilon^2 \times$ Cost','
            Interpreter','LaTex'); %title(stitle)
208     if option ==1 || option == 2
209         legend('Std QMC','MLQMC','Location','SouthEast','Interpreter','LaTex')
210     else
```

```
211         legend('Std QMC','MLQMC','Location','SouthEast','Interpreter','LaTex')
212     end
213     if option==3
214         axis([1e-4 2e-3 0.001 1])
215     elseif option==4
216         axis([1e-4 2e-3 0.01 10])
217     elseif option==5
218         axis([5e-5 1e-3 0.001 1])
219     else
220         axis([5e-5 1e-3 0.0001 0.1])
221     end
222
223     if nvert==1
224         print('-deps2c',sprintf('mlmc_gbm%db.eps',option))
225     else
226         print('-deps2c',sprintf('%s_%s.eps',stitle,rtitle))
227     end
228
229
230 end
231
232 %
    ------------------------------------------------------------------------
233
234 % function [P, Nl] = mlmc(eps,Ns,mlmc_l)
235 %
236 % multi-level Monte Carlo path estimation
237 % P = value
238 % Nl = number of lattice points at each level
239 % eps = accuracy (rms error)
240 % Ns = number of random shifts
241 % mlmc_l = function for level l estimator
242 %
243 % mlmc_l(l,Ns,N)
244 % l = level
245 % Ns = number of random shifts
246 % N = number of lattice points
247
248 function [P, Nl] = mlmc(eps,Ns,mlmc_l)
249
250 L = -1;
251 converged = 0;
252
253 while ~converged
```

```
254
255     %
256     % initial variance estimate
257     %
258
259     L = L+1;
260
261     Nl(L+1) = 1;
262     sums = feval(mlmc_l,L,Ns,Nl(L+1));
263     Pl(L+1) = sums(1)/Ns;
264     Vl(L+1) = max(sums(2)/Ns - (sums(1)/Ns).^2, 0) / (Ns-1);
265
266     %
267     % increase lattice size as needed
268     %
269
270     while sum(Vl) > 0.5*eps^2
271         err = Vl ./ (Nl.*2.^(0:L));
272         [foo,l] = max(err);
273         Nl(l) = 2*Nl(l);
274         sums = feval(mlmc_l,L,Ns,Nl(l));
275         Pl(l) = sums(1)/Ns;
276         Vl(l) = max(sums(2)/Ns - (sums(1)/Ns).^2, 0) / (Ns-1);
277         testsum = sum(Vl);
278     end
279
280     disp(sprintf(' %d ',Nl))
281
282     %
283     % test for convergence
284     %
285
286     range = -1:0;
287     if L>1 & 2^L>=16
288         con = 2.^range.*Pl(L+1+range);
289         converged = (max(abs(con)) < eps/sqrt(2)) | (2^L>=1024) ;
290     end
291 end
292
293 %
294 % evaluate multi-timestep estimator
295 %
296
297 P = sum(Pl);
298 Nl = Ns*Nl;
```

```
299
300 %  ----------------------------------------------------------
301
302 %
303 % standard Monte Carlo estimation
304 %
305
306 function [P, N] = mc(eps,Ns,L,mlmc_l)
307
308 %
309 % initial variance estimate
310 %
311
312 N = 1;
313 sums = feval(mlmc_l,L,Ns,N);
314 P = sums(3)/Ns;
315 V = max(sums(4)/Ns - (sums(3)/Ns).^2, 0) / (Ns-1);
316
317 %
318 % increase lattice size as needed
319 %
320
321 while V > 0.5*eps^2
322     N = 2*N;
323     sums = feval(mlmc_l,L,Ns,N);
324     P = sums(3)/Ns;
325     V = max(sums(4)/Ns - (sums(3)/Ns).^2, 0) / (Ns-1);
326 end
327
328 disp(sprintf(' %d ',N))
329 N = Ns*N;
330
331 %  ----------------------------------------------------------
332
333 %
334 % level l estimator
335 %
336
337 function sums = mlmc_l(l,Ns,N)
338
339 global option z_cbc method
```

```
340
341 T = 1;
342 r = 0.05;
343 sig = 0.2;
344 B = 0.85;
345
346 nf = 2^l;
347 nc = nf/2;
348
349 hf = T/nf;
350 hc = T/nc;
351
352 sums(1:4) = 0;
353
354 % define random shifts
355 delta = rand(2*nf,Ns);
356 % loop over lattice points
357 Ninc = max(1,floor(min(2^20/(Ns*nf),2^10/Ns)));
358
359
360 if (method == 1) %rank-1 lattice rule
361
362     Pf_sum = zeros(1,Ns);
363     Pc_sum = zeros(1,Ns);
364
365     for N1 = 1:Ninc:N
366         N2 = min(Ninc,N-N1+1);
367
368         %
369         % GBM model
370         %
371
372         X0 = 1;
373
374         Xf = X0*ones(1,N2*Ns);
375         Xc = Xf;
376
377         Af = zeros(1,N2*Ns);
378         Ac = Af;
379
380         Mf = Xf;
381         Mc = Xc;
382
383         Bf = 1;
384         Bc = 1;
```

```
385
386         ind1 = reshape((1:N2)'*ones(1,Ns),1,N2*Ns);
387         ind2 = reshape(ones(1,N2)'*(1:Ns),1,N2*Ns);
388
389         Uf = (z_cbc(1:2*nf)/N)*(N1:N1+N2-1);
390         Uf = rem(Uf(:,ind1)+delta(:,ind2), 1);
391
392         Zf = ncfinv(Uf(1:nf,:));
393         Lf = log(Uf(nf+1:end,:));
394         If = sqrt(hf/12)*hf*ncfinv(Uf(nf+1:end,:));
395
396         % Choice of Brownian Bridge or PCA
397
398         % Wf = pca(Zf,T,'bb_fft');
399          Wf = bb(Zf,T);
400
401         if l==0
402             Xf0 = Xf;
403             Xf = Xf + r*Xf*hf + sig*Xf.*Wf + 0.5*sig^2*Xf.*(Wf.^2-hf);
404             vf = sig*Xf0;
405             if option==2
406                 Af = Af + 0.5*hf*(Xf0+Xf) + vf.*If;
407             elseif option==3
408                 Mf = min(Mf,0.5*(Xf0+Xf-sqrt((Xf-Xf0).^2-2*hf*vf.^2.*Lf)));
409             elseif option==5
410                 Bf = Bf.*(1-exp(-2*max(0,(Xf0-B).*(Xf-B)./(hf*vf.^2))));
411             end
412
413         else
414
415             Wc = Wf(1:2:nf,:) + Wf(2:2:nf,:);
416             Ic = If(1:2:nf,:) + If(2:2:nf,:);
417
418             for n = 1:nc
419                 for m = 1:2
420                     dWf = Wf(2*n+m-2,:);
421                     Xf0 = Xf;
422                     Xf = Xf + r*Xf*hf + sig*Xf.*dWf + 0.5*sig^2*Xf.*(dWf.^2-hf);
423                     vf = sig*Xf0;
424
425                     if option==2
426                         Af = Af + 0.5*hf*(Xf0+Xf) + vf.*If(2*n-2+m,:);
427                     elseif option==3
428                         Mf = min(Mf,0.5*(Xf0+Xf-sqrt((Xf-Xf0).^2-2*hf*vf.^2.*Lf(2*n
                                -2+m,:))));
```

```
429                     elseif option==5
430                         Bf = Bf.*(1-exp(-2*max(0,(Xf0-B).*(Xf-B)./(hf*vf.^2))));
431                     end
432                 end
433
434                 dWc = Wc(n,:);
435                 Xc0 = Xc;
436                 Xc = Xc + r*Xc*hc + sig*Xc.*dWc + 0.5*sig^2*Xc.*(dWc.^2-hc);
437                 vc = sig*Xc0;
438
439                 if option==2
440                     Ac = Ac + 0.5*hc*(Xc0+Xc) + vc.*(Ic(n,:)+0.25*hc*(Wf(2*n-1,:)-
                            Wf(2*n,:)) );
441                 elseif option==3 || option==5
442                     Xc1 = 0.5*(Xc0 + Xc + vc.*(Wf(2*n-1,:)- Wf(2*n,:)));
443                     if option==3
444                         Mc = min(Mc, 0.5*(Xc0+Xc1-sqrt((Xc1-Xc0).^2-2*hf*vc.^2.*
                                Lf(2*n-1,:))));
445                         Mc = min(Mc, 0.5*(Xc1+Xc -sqrt((Xc -Xc1).^2-2*hf*vc.^2.*Lf
                                (2*n,:))));
446                     else
447                         Bc = Bc .*(1-exp(-2*max(0,(Xc0-B).*(Xc1-B)./(hf*vc.^2))));
448                         Bc = Bc .*(1-exp(-2*max(0,(Xc1-B).*(Xc -B)./(hf*vc.^2))));
449                     end
450                 end
451             end
452         end
453
454         if option==1
455             Pf = max(0,Xf-1);
456             Pc = max(0,Xc-1);
457         elseif option==2
458             Pf = max(0,Af-1);
459             % Pf = 0.5*(Pf(1:N2*Ns)+Pf(end-N2*Ns+1:end)); no longer using antithetic
                    pair
460             Pc = max(0,Ac-1);
461         elseif option==3
462             Pf = Xf - Mf;
463             Pc = Xc - Mc;
464         elseif option==4
465             if l==0
466                 Pf = ncf((Xf0+r*Xf0*hf-1)./(sig*Xf0*sqrt(hf)));
467                 Pc = Pf;
468             else
469                 dWf = Wf(nf-1,:);
```

```
                Pf = ncf((Xf0+r*Xf0*hf-1)./(sig*Xf0*sqrt(hf)));
                Pc = ncf((Xc0+r*Xc0*hc+sig*Xc0.*dWf-1)./(sig*Xc0*sqrt(hf)));
            end
        elseif option==5
            Pf = Bf.*max(0,Xf-1);
            Pc = Bc.*max(0,Xc-1);
        end

        Pf = exp(-r*T)*Pf;
        Pc = exp(-r*T)*Pc;

        if l==0
            Pc = zeros(1,N2*Ns);
        end

        Pf_sum = Pf_sum + sum(reshape(Pf,N2,Ns),1);
        Pc_sum = Pc_sum + sum(reshape(Pc,N2,Ns),1);
    end

    Pf = Pf_sum/N;
    Pc = Pc_sum/N;

else %Sobol or mixture


    Pftot = 0;
    Pctot = 0;

    for scr = 1:Ns
        Pf_sum = 0;
        Pc_sum = 0;
        if (method ==2 || method == 3);
            if(option == 2 || option == 3)
            P = sobolset(2*nf);
            else
            P = sobolset(nf);
            end
            P = scramble(P,'MatousekAffineOwen');
        end

        N2 = N;

        %
        % GBM model
        %
```

```
515
516         X0 = 1;
517
518         Xf = X0*ones(1,N2);
519         Xc = Xf;
520
521         Af = zeros(1,N2);
522         Ac = Af;
523
524         Mf = Xf;
525         Mc = Xc;
526
527         Bf = 1;
528         Bc = 1;
529
530         % if (method == 2)
531         %Uf = net(P,(scr*N2)+1);
532         %Uf = Uf((scr-1)*N2+2:scr*N2+1,:)';
533         Uf = net(P,N2)';
534         % end
535
536        if(method ==3 && nf>8)
537         Zf = [ncfinv(Uf(1:nc,:));randn(nc,N2)];
538         if(option ==2)
539         If = sqrt(hf/12)*hf*[ncfinv(Uf(nf+1:nf+nc,:));randn(nc,N2)];
540         end
541        else
542         Zf = ncfinv(Uf(1:nf,:));
543         if(option == 2)
544         If = sqrt(hf/12)*hf*ncfinv(Uf(nf+1:end,:));
545         end
546        end
547
548         Lf = log(Uf(nf+1:end,:));
549
550         % Choice of Brownian Bridge or PCA
551
552         % Wf = pca(Zf,T,'bb_fft');
553         Wf = bb(Zf,T);
554
555         if l==0
556             Xf0 = Xf;
557             Xf = Xf + r*Xf*hf + sig*Xf.*Wf + 0.5*sig^2*Xf.*(Wf.^2-hf);
558             vf = sig*Xf0;
559             if option==2
```

```
560             Af = Af + 0.5*hf*(Xf0+Xf) + vf.*If;
561         elseif option==3
562             Mf = min(Mf,0.5*(Xf0+Xf-sqrt((Xf-Xf0).^2-2*hf*vf.^2.*Lf)));
563         elseif option==5
564             Bf = Bf.*(1-exp(-2*max(0,(Xf0-B).*(Xf-B)./(hf*vf.^2))));
565         end
566
567     else
568
569         Wc = Wf(1:2:nf,:) + Wf(2:2:nf,:);
570         if(option ==2)
571         Ic = If(1:2:nf,:) + If(2:2:nf,:);
572         end
573
574         for n = 1:nc
575             for m = 1:2
576                 dWf = Wf(2*n+m-2,:);
577                 Xf0 = Xf;
578                 Xf = Xf + r*Xf*hf + sig*Xf.*dWf + 0.5*sig^2*Xf.*(dWf.^2-hf);
579                 vf = sig*Xf0;
580
581                 if option==2
582                     Af = Af + 0.5*hf*(Xf0+Xf) + vf.*If(2*n-2+m,:);
583                 elseif option==3
584                     Mf = min(Mf,0.5*(Xf0+Xf-sqrt((Xf-Xf0).^2-2*hf*vf.^2.*Lf(2*n
                            -2+m,:))));
585                 elseif option==5
586                     Bf = Bf.*(1-exp(-2*max(0,(Xf0-B).*(Xf-B)./(hf*vf.^2))));
587                 end
588             end
589
590             dWc = Wc(n,:);
591             Xc0 = Xc;
592             Xc = Xc + r*Xc*hc + sig*Xc.*dWc + 0.5*sig^2*Xc.*(dWc.^2-hc);
593             vc = sig*Xc0;
594
595             if option==2
596                 Ac = Ac + 0.5*hc*(Xc0+Xc) + vc.*(Ic(n,:)+0.25*hc*(Wf(2*n-1,:)-
                        Wf(2*n,:)) );
597             elseif option==3 || option==5
598                 Xc1 = 0.5*(Xc0 + Xc + vc.*(Wf(2*n-1,:)- Wf(2*n,:)));
599                 if option==3
600                     Mc = min(Mc, 0.5*(Xc0+Xc1-sqrt((Xc1-Xc0).^2-2*hf*vc.^2.*
                            Lf(2*n-1,:))));
```

```
                    Mc = min(Mc, 0.5*(Xc1+Xc -sqrt((Xc -Xc1).^2-2*hf*vc.^2.*Lf(2*n,:))));
                else
                    Bc = Bc .*(1-exp(-2*max(0,(Xc0-B).*(Xc1-B)./(hf*vc.^2))));
                    Bc = Bc .*(1-exp(-2*max(0,(Xc1-B).*(Xc -B)./(hf*vc.^2))));
                end
            end
        end
    end

    if option==1
        Pf = max(0,Xf-1);
        Pc = max(0,Xc-1);
    elseif option==2
        Pf = max(0,Af-1);
        % Pf = 0.5*(Pf(1:N2*Ns)+Pf(end-N2*Ns+1:end)); no longer using antithetic pair
        Pc = max(0,Ac-1);
    elseif option==3
        Pf = Xf - Mf;
        Pc = Xc - Mc;
    elseif option==4
        if l==0
            Pf = ncf((Xf0+r*Xf0*hf-1)./(sig*Xf0*sqrt(hf)));
            Pc = Pf;
        else
            dWf = Wf(nf-1,:);
            Pf = ncf((Xf0+r*Xf0*hf-1)./(sig*Xf0*sqrt(hf)));
            Pc = ncf((Xc0+r*Xc0*hc+sig*Xc0.*dWf-1)./(sig*Xc0*sqrt(hf)));
        end
    elseif option==5
        Pf = Bf.*max(0,Xf-1);
        Pc = Bc.*max(0,Xc-1);
    end

    Pf = exp(-r*T)*Pf;
    Pc = exp(-r*T)*Pc;

    if l==0
        Pc = zeros(1,N2);
    end

    Pf_sum = Pf_sum + sum(reshape(Pf,N2,1));
    Pc_sum = Pc_sum + sum(reshape(Pc,N2,1));

```

```
644             Pftot(scr) = Pf_sum/N;
645             Pctot(scr) = Pc_sum/N;
646         end
647
648         Pf = Pftot;
649         Pc = Pctot;
650     end
651
652     sums(1) = sums(1) + sum(Pf-Pc);
653     sums(2) = sums(2) + sum((Pf-Pc).^2);
654     sums(3) = sums(3) + sum(Pf);
655     sums(4) = sums(4) + sum(Pf.^2);
```

Bibliography

[BT95] V. Bally, D. Talay, "The law of the Euler scheme for stochastic differential equations, I: convergence rate of the distribution function," *Probability Theory and Related Fields*, vol. 104, no. 1, pp. 43–60, 1995.

[BF88] P. Bratley, B. Fox, "Algorithm 659: implementing Sobol's quasirandom sequences generator," *ACM Trans. Math. Software*, vol. 14, pp. 88–100, 1988.

[CP76] R. Cranley, T. Patterson, "Randomization of number theoretic methods for multiple integration," *SIAM J. Numer. Anal.*, vol. 13, pp. 904–914, 1976.

[CDMK09] J. Creutzig, S. Dereich, T. Müller-Gronbach, K. Ritter, "Infinite-dimensional quadrature and approximation of distributions," *Foundations of Computational Mathematics*, vol. 9, no. 4, pp. 391–429, 2009.

[DKS13] J. Dick, F. Kuo, I. Sloan, "High-dimensional integration: the quasi-Monte Carlo way," *Acta Numerica*, vol. 22, pp. 133–288, 2013.

[Gil08a] M. Giles, "Improved multilevel Monte Carlo convergence using the Milstein scheme," *Monte Carlo and Quasi-Monte Carlo Methods 2006, Springer*, pp. 343–358, 2008.

[Gil08b] ——, "Multi-level Monte Carlo path simulation," *Operations Research*, vol. 56, no. 3, pp. 607–617, 2008.

[Gil15] ——, "Multilevel Monte Carlo methods," *Acta Numerica*, vol. 24, pp. 259–328, 2015.

[GS14] M. Giles, L. Szpruch, "Antithetic multilevel Monte Carlo estimation for multidimensional SDEs without Lévy area simulation," *Annals of Applied Probability*, vol. 24, no. 4, pp. 1585–1620, 2014.

[GW09] M. Giles, B. Waterhouse, "Multilevel quasi-Monte Carlo path simulation," in *Advanced Financial Modelling, Radon Series on Computational and Applied Mathematics*, 2009, pp. 41–50.

[Gla03] P. Glasserman, *Monte Carlo methods in financial engineering.* Springer–Verlag, New-York, 2003.

[JK03] S. Joe, F. Kuo, "Remark on Algorithm 659: implementing Sobol's quasirandom sequence generator," *ACM Trans. Math. Software*, vol. 29, no. 1, pp. 49–57, 2003.

[KP92] P. Kloeden, E. Platen, *Numerical Solution of Stochastic Differential Equations*. Springer–Verlag, Berlin, 1992.

[KW97] L. Kocis, W. Whiten, "Computational investigations of low-discrepancy sequences," *ACM Trans. Math. Software*, vol. 23, no. 2, pp. 266–294, 1997.

[KP15] P. Kritzer, F. Pillichshammer, *Component-by-component construction of shifted Halton sequences*, http://arxiv.org/pdf/1501.07377v1.pdf, Uniform Distribution Theory (to appear), 2015.

[KN74] L. Kuipers, H. Niederreiter, *Multivariate models and multivariate dependence concepts*. Wiley, 1974.

[KS05] F. Kuo, I. Sloan, "Lifting the curse of dimensionality," *Notices of the AMS*, vol. 52, pp. 1320–1328, 2005.

[Lem09] C. Lemieux, *Monte Carlo and Quasi-Monte Carlo Sampling*. Springer–Verlag, New-York, 2009.

[Mat98] J. Matoušek, "On the l_2-discrepancy for anchored boxes," *Journal of Complexity*, vol. 14, pp. 527–556, 1998.

[Nie78] H. Niederreiter, "Existence of good lattice points in the sense of Hlawka," *Monatsh. Math.*, vol. 86, pp. 203–219, 1978.

[Nie87] ——, *Random number generation and Quasi-Monte Carlo Methods*. Society for Industrial and Applied Mathematics, 1987.

[Nuy07] D. Nuyens, "Fast construction of good lattice rules," PhD thesis, Catholic University of Leuven, 2007.

[Owe98] A. Owen, "Scrambling Sobol' and Niederreiter–Xing points," *Journal of Complexity*, vol. 14, pp. 466–489, 1998.

[Sam08] P. Samuel, *Algebraic Theory of Numbers*. Dover Publications, 2008.

[SAKK11] I. Sobol', D. Asotsky, A. Kreinin, S. Kucherenko, "Construction and comparison of high-dimensional Sobol' generators," *Wilmott Journal*, pp. 64–79, 2011.