Quasi-Monte Carlo Simulations for Longstaff Schwartz Pricing of American Options

Eric Couffignals
Mathematical Institute
Lady Margaret Hall
University of Oxford

MScMCF: Dissertation

Trinity Term, 2010

Acknowledgements

This thesis would not have been possible without the help of Professor Mike Giles. He supported me for this dissertation during all the term. I appreciated his guidance and his precision in all the anwers to the questions I had. I am also grateful to Jiujiu Xiong with who I worked at the beginning to get the results on the European options pricing with QMC.

Contents

List of Figures

List of Tables

1 Introduction

In this dissertation, we investigate the use of Quasi-Monte Carlo methods for the pricing of Amerian options.

An American option grants the holder the right to select the time at which to exercise the option. It makes American options more difficult to price than European as it requires the solving of an optimal stopping problem. American options can be found in all major financial markets including the equity, commodity, credit, swap and foreign exchange markets.
Binomial trees or finite differences methods are very efficient in the case of option with one underlying, but for multifactor situations, it is better to use Monte Carlo methods. In this dissertation, we will price American options using the Longstaff-Schwartz algorithm, a powerful approach developed in 2001 and based on the use of least-squares to estimate the conditional expected payoff to the option holder from continuation.

Quasi-Monte Carlo methods enable us to get the same accuracy as pure Monte Carlo methods at a much lower computational cost. With Monte Carlo methods, the convergence of the estimate is ensured by the law of large numbers. However, it is slow as the root mean square error is $\mathcal{O}(N^{-1/2})$ instead of $\mathcal{O}(N^{-1})$ in the best cases for Quasi-Monte Carlo methods.
Quasi-Monte Carlo methods are based on the generation of quasi-random uniform variables. There exist different manners to do this; we will concentrate on the generation of Sobol sequences to obtain these quasi-random points. In order to improve Quasi- Monte Carlo methods, we will also use the Brownian bridge construction and perform a reduction of effective dimension using the principal component analysis (PCA).

This dissertation begins with a presentation of Quasi-Monte Carlo theory.

We then present the Longstaff-Schwartz algorithm in section 3 and give two algorithms to get lower and upper bounds for the American put price. In the last section, we discuss the numerical results.

2 Quasi Monte Carlo Theory

2.1 General Principles

The standard Monte Carlo and the Quasi Monte Carlo both approximate high-dimensional hypercube integral

$$\int_{[0,1]^d} f(x)\, dx$$

by

$$\frac{1}{N}\sum_{i=1}^{N} f(x^{(i)}).$$

The difference between the two techniques lies in the choice of the points $x^{(1)}, x^{(2)}, ..., x^{(N)}$. As regards to classic Monte Carlo simulations, the points are chosen to be independent and identically distributed. They are uniform random vectors from the unit cube.

The Quasi Monte Carlo approach is more subtle: points are not generated randomly but pseudo-randomly. We don't use random uniform variables but deterministic variables that are distributed more uniformly, which improves the rate of convergence. There exist different kind of pseudo-random sequences. In this paper, we choose Sobol sequences.

Giles points out in [MG08] that using QMC methods can improve the rate of convergence to $\mathcal{O}(N^{-1})$ or even better sometimes while the r.m.s. error is $\mathcal{O}(N^{-1/2})$ for pure Monte Carlo methods.

2.2 Generation of Sobol Sequences and Randomised QMC

2.2.1 Low Discrepancy Sequences

As Giles (see [MG10]) underlines, ' the key is to use points which are fairly uniformly spread within the hypercube, not clustered anywhere.' The discrepancy is a measure of the "level of uniformity" or more exactly the deviation from uniformity. It is defined by (see [GA04]):

$$D_N^* = \max_{A \in \mathcal{A}} \left| \frac{\#\{x_i \in \mathcal{A}\}}{n} - vol(A) \right|$$

where $\mathcal{A}$ is a collection of subsets of $[0,1)^d$, $\#\{x_i \in \mathcal{A}\}$ is the number of points x_i contained in $\mathcal{A}$ and vol(A) is the the volume (measure) of $\mathcal{A}$. If we choose $\mathcal{A}$ to be the collection of all rectangles in $[0,1)^d$ of the form:

$$\prod_{j=1}^{d} [0, u_j), 0 \leq u_j \leq 1$$

we define the star discrepancy $D_N^*(x^1, x, ..., x^N)$. The lower the star discrepancy is, the more uniformly distributed the points are.

Niederreiter (see [NH92]) states that there are sequences for which:

$$D_N^* \leq C \frac{(logN)^d}{N} \tag{1}$$

Taking Giles' notations (see [MG10]), the Koksma-Hlawka inequality enables us to bound the error:

$$\left| \frac{1}{N} \sum_{i=1}^{N} f(x^{(i)}) - \int_a^b f(x)\,dx \right| \leq V(f) D_N^*(x^1, x, ..., x^N) \tag{2}$$

where V(f), given f is sufficiently differentiable, is the Hardy-Krause variation of f defined as:

$$V(f) = \int_{[0,1]^d} \left| \frac{\partial^d f}{\partial x_1 ... \partial x_d} \right| \mathrm{d}x$$

However, as Glasserman (see [GA04]) underlines, 'the condition V(f) be finite is restrictive. It requires for example, that f be bounded, a condition often violated in option pricing applications'. In addition, both V(f) and the star discrepancy D_N^* are difficult to compute. We can though, using (1) and (2) get some information as regards to the asymptotic behaviour:

$$Error < C\frac{(logN)^d}{N}$$

We can thus see that for small dimension d, QMC methods will be much better than the standard Monte-Carlo $N^{-1/2}$ r.m.s. error. But for large dimension d, there is no clear benefit since $(logN)^d$ can be enormous.

2.2.2 Sobol Sequences

Sobol sequences are an example of quasi-random low-discrepancy sequences and are one of the most popular approach.
These sequences have the property that for small dimensions d < 40 the subsequence $2^m \leq i \leq 2^{m+1}$ of length 2^m has precisely 2^{m-d} points in each of the little cubes of volume 2^{-d} formed by bisecting the unit hypercube in each dimension ' [MG10].
Thus for instance, if we cut any dimension into halves, each portion has 2^{m-1} points. If we cut any dimension into quarters, each portion has 2^{m-2} points.

The first step of the construction of Sobol sequences is the generation of direction numbers (see [JS08]).
Given a dimension k, we take a primitive polynomial P of degree p with coefficients chosen from $\{0, 1\}$. In addition, we choose the primitive polynomial to be modulo 2. Thus we get a polynomial of order $2^p - 1$, i.e. $2^p - 1$ is the smallest integer m for which P(x) divide $(x^m - 1)$. All the polynoms chosen for the generation of sobol sequences are sorted by increasing degree. Each of them is represented by a binary number which gives the coefficients of the polynoms. Since the first and last coefficient are equal to 1 because the polynomial is primitive, we don't take them into account. Thus, for instance, a primitive polynomial of degree 5 is: $P(x) = x^5 + x^2 + 1$ (see [GA04]). Its binary representation is 100101 which corresponds to 37 in decimal code.

Degre	Primitive polynomials
0	1
1	3 (x+1)
2	7 $(x^2 + x + 1)$
3	11 $(x^3 + x + 1), 13(x^3 + x^2 + 1)$
4	19, 25
5	37,59, 47, 61, 55, 41

Table 1: Primitive polynomials of degree 5 or less

Let's consider the primitive ploynomial P of degree p such that:

$$P(x) = \sum_{i=0}^{p} a_i.x^{k-i}$$

We also choose p integers $v_1, ..., v_p$ called the *direction numbers*: for each number v_l, the l Most Significant Bit (MSB) have to be different from zero and the l^{th} strongest bit have to be equal to 1 (they are several possibilities, see [BR88] to know which one to choose).

We then define a recurrence determining the next direction numbers $v_{p+1}, ..., v_b$:

$$v_l = \frac{v_{l-k}}{2^p} \oplus \sum_{i=1}^{p} a_i.x^{l-i} \quad \forall l > p_k$$

where $\oplus$ denotes a bit-by-bit exclusive or-operation (addition in binary code). The pseudo-random characters of Sobol sequences comes from this recurrence. The less significant bits are generated from the most significant bits. Finally, we will be able to get b direction numbers for each dimension.

Once we get the direction numbers, we can generate Sobol sequences. Taking the same notations, as Jauvion (see [JAU08]), we define v_{kl} as the l^{th} directive number for the dimension k. We then choose a strictly postive integer n and for each dimension, we define x_k as the sum of the directive numbers such that for the l^{th} directive number, the l^{th} bit of n is 1:

$$x_k = \sum_{l=1}^{b} v_{kl}.\mathbf{1}_{the\ l^{th} of\ n\ is\ 1} \quad \forall k \in [1, d]$$

By dividing each x_k by 2^b, we get a sequence of quasi-random numbers between 0 and 1. The integer n will be used to generate the n^{th} trajectory. The following table gives 10 paths of 5 points: they are the first five sobol points in dimension 10:

Sequences	1	2	3	4	5	6	7	8	9	10
1	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
2	0.25	0.75	0.25	0.75	0.25	0.75	0.25	0.75	0.75	0.25
3	0.75	0.25	0.75	0.25	0.75	0.25	0.75	0.25	0.25	0.75
4	0.125	0.625	0.875	0.875	0.625	0.125	0.375	0.375	0.875	0.625
5	.625	0.125	0.375	0.375	0.125	0.625	0.875	0.875	0.375	0.125

Table 2: The first five Sobol points in dimension 10

Then, by applying the inverse of the Normal cumulative distribution function Φ^{-1} , we can get quasi-random gaussian variables.
The following figure gives the projection of 512 points in 2 dimensions using sobol sequences and random points. We can see that using sobol sequences increases the uniformity.

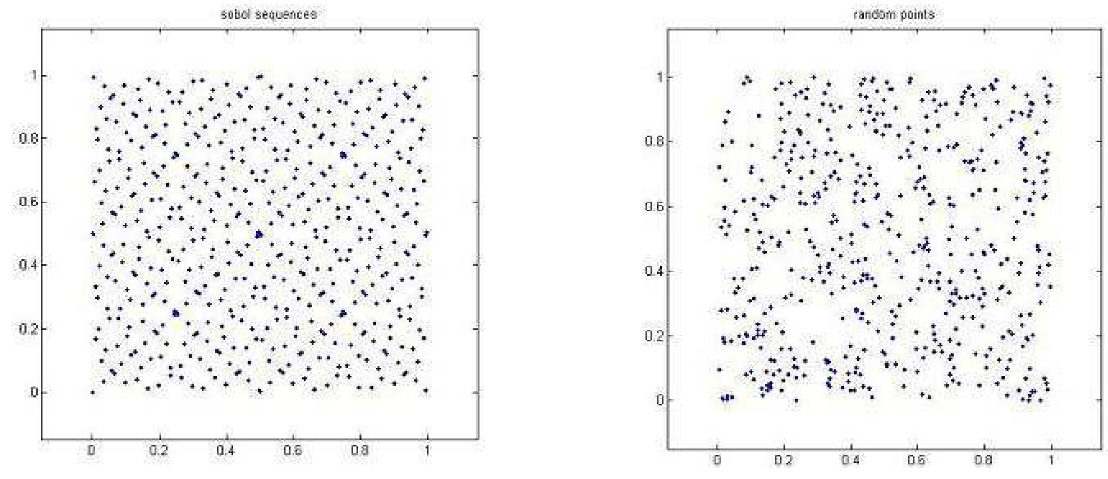


Figure 1: Comparison: Sobol sequences vs Random points

2.3 Randomised QMC

Randomisation enables to measure error through a confidence interval while preserving much accuracy of the pure QMC. As explained by Glasserman (see [GA04]), 'the tradeoff it poses, sacrificing some precision to get a better measure of error, is essentially the same one we faced with several of the variance reduction techniques'.
An other reason of using randomisation is that it also can improve accuracy. The theorem of Owen, stated in [OW97], shows that using a class of randomised nets can reduce the root mean square of integration from $O(\frac{1}{n^{1.5-\epsilon}})$ to $O(\frac{1}{n^{1-\epsilon}})$. Glasserman ([GA04]) points out that Owen's results may not be applicable for pricing derivatives since they apply to smooth integrands.

As regards to randomisation for sobol points $x^{(i)}$, we use digital scrambling which preserves the properties of the sequences. Actually, we add a random process X^m so that we can obtain a confidence interval:

$$x^{(i,m)} = x^{(i)} \veebar X^m$$

where we apply bitwise the exclusive-or operation $\veebar$ (see [MG10]) so that:

$$\begin{array}{l} \;\;0.1010011 \\ \veebar 0.0110110 \\ = 0.1100101 \end{array}$$

2.4 Dimension reduction through Principal Component Analysis (PCA)

2.4.1 Motivation

American options being path-dependent, we have to generate paths. We use the Euler-Maruyama approximation:

$$\widehat{S_{n+1}} = \widehat{S_n} + a(\widehat{S_n}, t_n)h + b(\widehat{S_n}, t_n)\Delta W_n$$

where $\Delta W_n = W_n - W_{n-1}$ are i.i.d. brownian $N(0, h)$ increments.
The generation of Sobol sequences has enabled us, through the inversion of the Normal cumulative distribution function, to get Normal variables. These Normal variables will be used to generate the Brownian increments. It doesn't matter how ΔW is generated when we use standard Monte Carlo: ΔW has just to have the correct distribution. However, it does matter when we perform QMC simulations.

For classic Monte Carlo simulations, the correlated Normals are generated through $Y = LX$ where X is a vector of i.i.d. standard normal variables. We have the same distribution for any L such that $LL^T = \Sigma$.
But Giles remarks in [MG10], that for Quasi-Monte Carlo simulations, the points are more uniformly distributed through the lowest dimensions. Therefore, we have to think about how the dimensions are allocated to the problem. For QMC, a change of L corresponds to a change of coordinate which can have a lot of influence. According to Giles, an efficient technique is to use the PCA construction: $L = U\Lambda^{1/2}$ where the eigenvalues are arranged in descending order. We can also the Brownian Bridge method.

2.4.2 The Brownian Bridge method

The Brownian Bridge method imposes the last value W_N to be equal to $\sqrt{T}Z_1$ where we define Z_1 as the first normal variable generated (we have W(0)=0). The principle of the Brownian Bridge construction is that we the, continue recursively, trying everytime to find the distribution of the midpoint. We assume here that N is a power of 2. Conditionally on the initial and final value of the Brownian path values, the midpoint value $W_{N/2}$ is distributed with mean $\frac{1}{2}W_N$ and variance T/4. Consequently, we can construct $W_{N/2}$ as follows:

$$W_{N/2} = \frac{1}{2}W_N + \sqrt{T/4}\ Z_2$$

By continuing recursively, we get the quarter and three-quarters points:

$$W_{N/4} = \frac{1}{2}W_{N/2} + \sqrt{T/8}\ Z_3$$

$$W_{3N/4} = \frac{1}{2}(W_N + W_{N/2}) + \sqrt{T/8}\ Z_4$$

etc...

2.4.3 Standard Brownian path PCA

In this section, we follow Giles (see [MG07]). We define a scalar Brownian path $W_n = W(t_n), n = 1, 2, ...N$, a scalar standard Brownian motion, with t_n=n/N. Each value is normally distributed with zero mean. The covariance matrix for W is Ω with elements:

$$\Omega_{j,k} = min(t_j, t_j)$$

. The Principal Component Analysis (PCA) defines the W_n as:

$$W_n = \sum_{m=1}^{N} Z_m \sqrt{\lambda_m} (V_m)_n$$

where:

- λ_m is an eigenvalue of Ω
- $(V_m)_n$ is the n^{th} component corresponding unit eigenvector
- Z_m are independent $N(0,1)$ random variables that we can obtain by using Sobol sequences

The eigenvalues and unit eigenvectors of Ω are:

$$\lambda_m = \frac{1}{4N}\left(sin\left(\frac{(2m-1)\pi}{2(2N+1)}\right)\right)^{-2}$$

$$(V_m)_n = \frac{1}{\sqrt{2N+1}} sin\left(\frac{(2m-1)n\pi}{2N+1}\right)$$

Remark:
Giles suggests (see [MG07]) that since the eigenmodes V_m are Fourier modes, the required PCA summation can be performed using an FFT of the appropriate length, at a cost which is $O(NlogN)$.' It is thus much better than the

cost of the standard implementation which is $O(N^2)$.
If we define:

$$a_m = Z_m \sqrt{\lambda_m} \frac{2}{\sqrt{2N+1}}$$

$$b_m = \begin{cases} a_{(m+1)/2}, & \text{for odd } m \\ 0, & \text{for odd } m \end{cases}$$

we get:

$$W_n = \sum_{m=1}^{2N} b_m sin\left(\frac{mn\pi}{2N+1}\right)$$

By applying a type-I discrete sine transform to the coefficients b_m, we produce $W_n, n = 1, ..., 2N$ (we will need only the first N values).

2.4.4 The Hybrid PCA method

By generalising section 2.4.2 and taking $W_N \equiv W(1)$ (see [MG07]), we get for any integers j,k with $0 < j < k < N$:

$$W_k = t_k W_N + \sqrt{t_k(1-t_k)}\; Z_1$$

and

$$\begin{aligned} W_j &= \tfrac{t_j}{t_k} W_k + \sqrt{t_j(t_k - t_j)}\; Z_2 \\ &= t_j W_N + \tfrac{t_j}{t_k}\sqrt{t_k(1-t_k)}\; Z_1 + \sqrt{t_j(t_k-t_j)}\; Z_2 \end{aligned}$$

where Z_1 and Z_2 are independent standard normal random variables. Hence, the covariance between W_j and W_k is:

$$Cov(W_j, W_k) = \frac{t_j}{t_k} t_k(1-t_k) = t_j - t_j t_k$$

Therefore, the discrete Brownian path values W_n at intermediate times are normally distributed with mean $t_n W(1)$ and covariance matrix Ω^* with elements:

$$\Omega^* = min(t_j, t_k) - t_j t_k$$

. The eigenvalues and unit eigenvectors of $\Omega*$ are:

$$\lambda_m^* = \frac{1}{4N}\left(sin\left(\frac{m\pi}{2N}\right)\right)^{-2}$$

$$(V_m^*)_n = \frac{2}{\sqrt{2N}} sin\left(\frac{mn\pi}{N}\right)$$

Remark:
Giles outlines in [MG07] the hybrid Brownian Bridge -PCA construction where the W_n are defined as:

$$W_n = \frac{n}{N} W_N + \sum_{m=1}^{N-1} Z_m \sqrt{\lambda_m^*} (V_m^*)_n,$$

where:

- λ_m^* is an eigenvalue of Ω
- $(V_m^*)_n$ is the n^{th} component corresponding unit eigenvector
- Z_m are independent $N(0,1)$ random variables

If we define:

$$a_m = Z_m \sqrt{\lambda_m} \frac{2}{\sqrt{2N}},$$

we get:

$$W_n = \frac{n}{N} W_N + \sum_{m=1}^{N-1} a_m sin\left(\frac{mn\pi}{N}\right)$$

By applying a type-I discrete sine transform to the coefficients a_m, we can produce directly $W_n, n = 1, ..., N$.

The advantage of the hybrid method is that the FFT dimension is a simple power of 2 and approximately half the size in comparison to the standard PCA construction. Thus, it is particularly efficient from a computational point of view.

2.5 Pricing of European Put

As a check point, we decided to check the efficiency of Quasi-Monte Carlo methods for the pricing of European options. All calculations use 128 timesteps and 64 'sets' of points. QMC and MC error bounds show the 3 standard deviations bounds which means that the exact price should be at 99.7 % in the confidence interval. We tested four different techniques:

- Pure Monte Carlo (MC)
- Quasi-Monte Carlo with a PCA construction (QMC-PCA)
- Quasi-Monte Carlo with a Brownian Bridge (QMC-BB)
- Quasi-Monte Carlo with a hybrid Brownian Bridge and a PCA summation performed using a FFT (QMC-HYB)

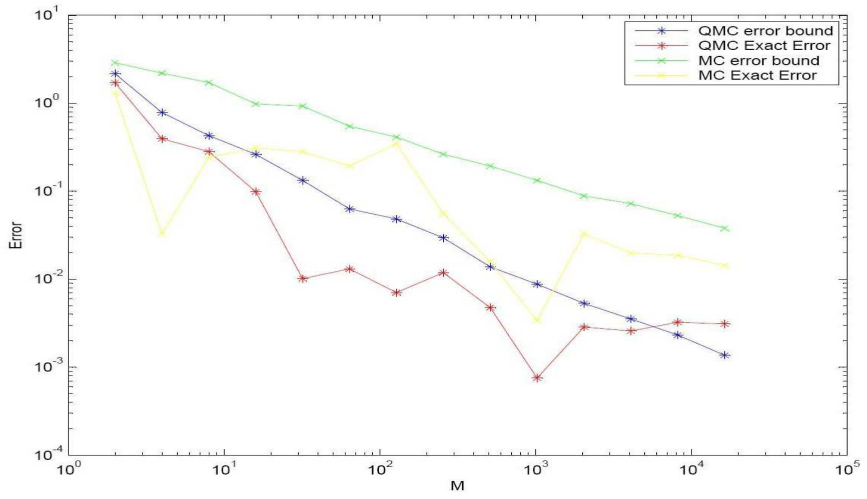


Figure 2: QMC and MC convergence in the case of standard PCA method

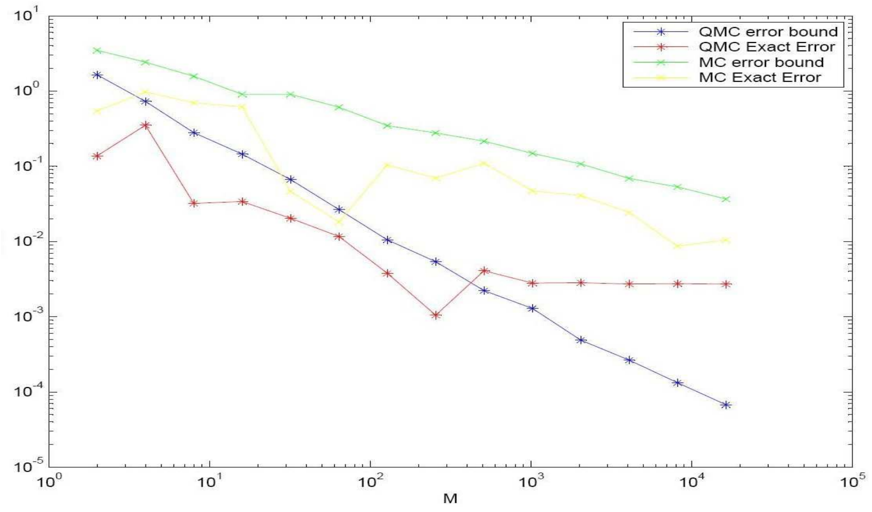


Figure 3: QMC and MC convergence in the case of Brownian Bridge

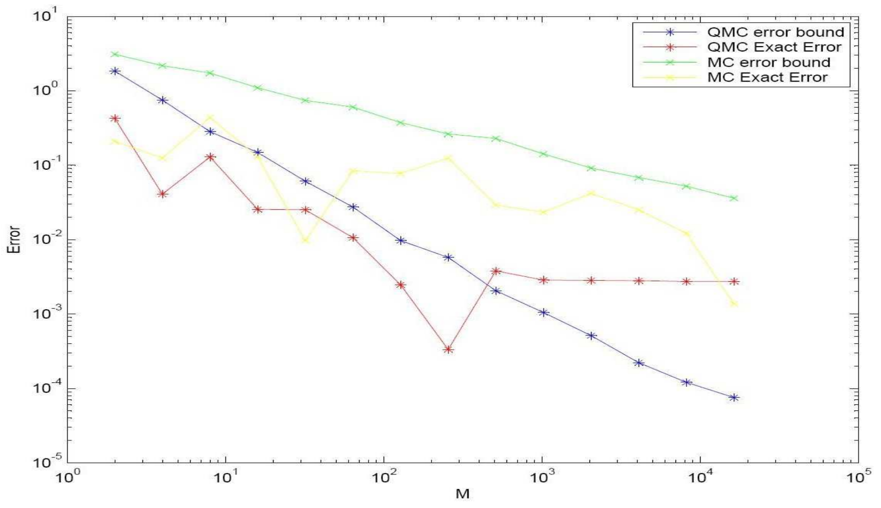


Figure 4: QMC and MC convergence in the case of Hybrid Brownian Bridge PCA

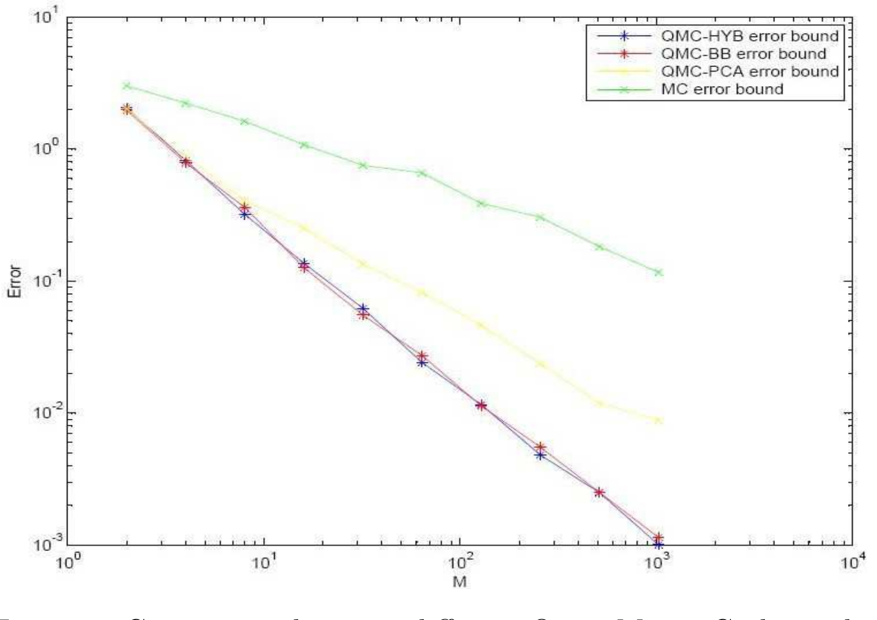


Figure 5: Comparison between different Quasi-Monte Carlo method

The figure 2, 3 and 4 show us that QMC is more effective than pure Monte Carlo simulations. When the number of paths is very large, the fact that the exact error is greater than the bound is because the sampling error becomes so small that the exact error is dominated by the weak convergence, which isn't bounded by the sampling error bound.

The figure 5 shows that the hybrid versions of QMC are more efficient than pure QMC. For the same number of paths (N=1000), the errror is divided by a factor superior to 100 between a pure Monte Carlo simulation and a hybrid QMC. As expected, the slope is -1 in the case of hybrid QMC and approximatively -1/2 for Monte Carlo simulations.

We can conclude that we obtained for a same computational cost, a much better accuracy with QMC techniques.

3 Pricing of American Options

3.1 Introduction to American options

In this section, we follow section 5.14 of [KO10] and chapter 8 of [GA04].

3.1.1 General framework

We formulate the problem where we consider the markets to be complete, with Σ an underlying probability space and $\mathcal{F}$=$\{\mathcal{F}_t : t \in [0,T]\}$ the associated sigma algebra which represents the amount of available information at each date t. We denote $\mathbb{Q}$ the unique risk-neutral probability measure. Since this is unique, we will be able to price all contingent claims by determining the expectation of the discounted payments.
We consider that S, the underlying of the American option, followed a geometric Brownian motion such that:

$$\begin{cases} dS(t) = rS(t)dt + \sigma S(t)dW(t), & \text{for t} > 0 \\ S(0) = S_0 \end{cases}$$

where r is the interest rate and σ the volatility. They will be constants in all this paper.

3.1.2 Definition of American options

An American option can be exerciced at any time up to its expiration. Consequently, the seller of the option doesn't know a priori the exact time of the payment: the value of an American option is the value by exercing optimallly.
We can represent American option pricing problems by defining the payment at each time t to be a continous-time process $B(t), 0 \leq t \leq T$ and by specifying a class of admissible stopping times S, adapted to the filtration corresponding to the chosen market model, with values in [0,T] almost surely.

For each exercice strategy and corresponding stopping time τ, we define B_τ as the payment of the American put with exercice strategy τ:

$$B_\tau = (K - S_\tau)^+$$

For this exercice strategy, the price of the option will be the discounted payment under the risk-neutral measure $\mathbb{Q}$:

$$p_{B_\tau} = \mathbb{E}^{\mathbb{Q}}(e^{-r\tau} B_\tau) \tag{3}$$

Remark:
The buyer of the option can decide on a strategy which is suboptimal. In this case, the seller can't perfectly set up a portfolio replicating exactly this strategy. However, Karatzas and Schreve have shown in [SC10] that there exists an admissible trading strategy $X^*(t)$ with initial wealth equalling (3) but we could then go above B_t:

$$X^*(t) \geq B_t \ \ \forall t \in [0, T] \ a.s.$$

3.1.3 Hedging strategy and fair price of an American option

According to Korn (see [KO10] p.242), a hedging strategy for an American contingent claim, is ' a portfolio process $\pi \in \mathcal{A}(x)$ with corresponding wealth process $X^\pi(t) \geq B_t$ for all $t \in [0, T]$ with price $x > 0$ for the American contingent claim B.'

The seller should be prepared against the worst strategy which is the one that will maximise the value of the option for the buyer. Thus, the fair price of the American option is given by:

$$\hat{p} = \sup_{\tau \in S[0,T]} \mathbb{E}^{\mathbb{Q}}(e^{-r\tau} B_\tau)$$

THEOREM:
There exists a stopping time τ^* such that the supremum $\hat{p}$ will be attained for the hedging strategy π^* corresponding to τ^*.

Glasserman (see [GA04]) points out that the the supremum $\hat{p}$ is attained for a stopping time τ^* which satisfied:

$$\tau^* = inf\{t \geq 0 : S_t \leq b_t^*\}$$

where b^* is an optimal exercice boundary. The following figure illustrates the exercice boundary for the American put: as the underlying asset reaches the boundary for the first time, the option is exerciced:

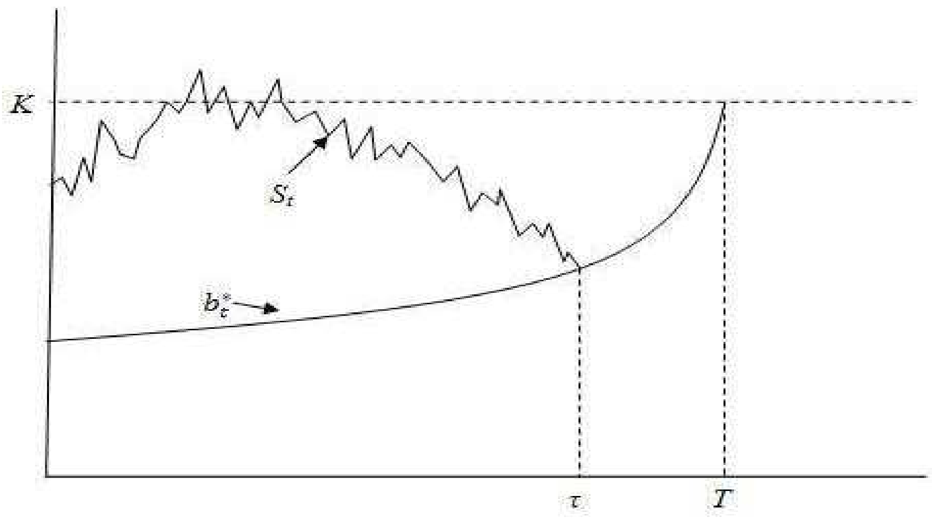


Figure 6: Exercice Boundary for American put

In this dissertation, we use Monte Carlo simulations and consequently have to consider options that can only be exerciced at a fixed set of dates $t_1, t_2 ..., t_M$. We thus approximate American options by Bermudan options with a large number of exercice opportunities.

3.2 The Longstaff-Schwartz algorithm

We choose here the same notation as Korn (see [KO10], section 5.14.1) and follow him in his presentation of the Longstaff-Schwartz algorithm.

3.2.1 Dynamic programming formulation

The Longstaff-Swartz algorithm is one of the most popular among the practioners, particularly for the pricing of American options on more than one underlying asset. In the latter case, binomial trees methods are usually easier and more efficient.
The algorithm is based on backwards induction (or dynamic programming principle). We price the American option by approximating a Bermudan option with exercise dates among $\{t_1, t_2, ..., t_m = T\}$. We also define:

- $S(i)$ the class of admissible stopping times with values in $\{i, ..., m\}$
- $V(i) = e^{-rt_i} B_{t_i}$

Since we don't know the optimal exercice strategy, we work backwards and start at time T. Supposing that the option has not been exerciced until then, its value at time T is merely equal to its intrinsic value B_T. Its net present value is $\mathbb{E}^{\mathbb{Q}}(V(m))$ and the optimal stopping time is $t_m = T$.
We then go one step backward to the exercice date t_{m-1}. For each possible value of the underlying $S_{t_{m-1}}$, we have to decide if we exercice the option or not. We do that by comparing the intrinsic value B_{m-1} of the option at time t_{m-1} and the value of keeping the option until time T which is equal to: $\mathbb{E}^{\mathbb{Q}}(e^{-r(T-t_{m-1})} B_T | S_{t_{m-1}})$. The optimal stopping time $t_{\tau^*(m-1)}$ (conditioned on not having exerciced before t_{m-1} and on the actual price $S_{t_{m-1}}$) is then either m or m-1.
We continue backwards to the exercice date t_{m-2} and proceed as above which

gives us the optimal stratgey (exercice or hold the option) and the optimal stopping time.
Continuing backwards leads to the following algorithm:

- Start at i=m, $\tau(i) = m$.
- Work backwards from $i = m - 1$ *to* $i = 0$ and determine the optimal exercice strategy $\tau(i) \in S(i)$ at each time t_i using:

$$\tau(i) = \begin{cases} i, & \text{if } V(i) \geq \mathbb{E}^{\mathbb{Q}}(V(\tau(i+1))|S(t_i)) \\ \tau(i+1), & \text{else} \end{cases}$$

- At time t=0, the price of the American option (approximated by a Bermudan) is $\mathbb{E}^{\mathbb{Q}}(V(\tau(0))$.

Given that we manage to determine all the conditional expectations, we could directly give the price of the option. However, this is not possible and we carry on perfoming calculations. In their original paper (see [LS01]), Longstaff and Schwartz exposes a way to determine the conditional expectations by using a least-squares approach.

3.2.2 The Longstaff-Schwartz's least-squares approach

Since S_t is a Markov process, we have the following relations:

$$B_t = f(S(t))$$

$$V(i) = g(i, S(t_i)) = e^{-rt_i} f(S(t_i))$$

$$\mathbb{E}^{\mathbb{Q}}(V(j)|S(t_i)) = u(S(t_i)) \text{ for } i < j$$

where u is a suitable mesurable function belonging to a parametric family U. It is then possible to approximate the conditional expectation in the least-squares sense by setting up a regression model and solving:

$$\min_{u \in U} \mathbb{E}^{\mathbb{Q}} \left[\mathbb{E}^{\mathbb{Q}}(g(i+1, S(t_{i+1})) | S(t_i)) - u(S(t_i)) \right]^2$$

Longstaff and Schwartz justify that we can follow this approach as they point out that the conditional expectation being an element of L^2 and L^2 being a Hilbert space having a countable orthonormal basis, the conditional expectation can be represented as a linear function of the element of the basis.
It is consequently necessary to specify the function space U:

$$U := \left\{ u : \mathbb{R}^d \longrightarrow \mathbb{R} \mid u(x) = \sum_{i=1}^{k} a_i H_i(x), a_i \in \mathbb{R} \right\}$$

where $H_i : \mathbb{R}^d \longrightarrow \mathbb{R}$ are basis functions and $k \in \mathbb{N}$.
Longstaff and Swartz chose the basis functions to be weighted Laguerre polynomials, defined as follows:

$$\begin{gathered} L_0(x) = exp(X/2), \\ L_1(x) = exp(X/2)(1 - X), \\ L_2(x) = exp(X/2)(1 - 2X + X^2/2), \\ L_n(x) = exp(X/2) \tfrac{e^x}{n!} \tfrac{\partial^n}{\partial X^n}(X^n e^{-X}). \end{gathered}$$

The least-squares approach is indeed a regression problem since the parametrisation of U is linear in the coefficients a_i.

With m timsteps and N paths, it is possible to solve explicitely the regression problem:

$$\min_{u \in \mathbb{R}^k} \frac{1}{N} \sum_{j=1}^{N} \left(g(i+1, S(t_{i+1}^j)) - \sum_{l=1}^{k} a_l H_l S(t_i^j) \right)^2$$

The solution of this linear regression problem is the optimal coefficient vector $a_1^*, ..., a_m^*$. Korn (see [K010]) gives an explicit representation of the a_i^* by computing the inverse of the following matrix $H(i)$ at every timestep:

$$H(i,j) = (H_1(S(t_i)^{)}, ..., H_k(S(t_i)^j)),$$
$$H(i) = (H(i,1)', ..., H(i,N)')',$$
$$H^+ = (H(i)'H(i))^{-1}H(i)',$$

and we finally get:

$$a_i^* = H_i^+ g(i+1)$$

where $g(i+1) = g(i+1, S(t_{i+1}^j)$ is the data vector.

Thanks to the coefficients a_i and the basis functions we can now give an estimate of the conditional expectation: we define $\hat{C}(S;i)$ the functional value of the continuation value (that is the value of not exercicing the option and holding it) of the American option at time t_i:

$$\hat{C}(S(t_i)^j; i) = \sum_{l=1}^{k} a_l^*(i) H_l(S(t_i)^j))$$

3.2.3 The algorithm

We can now give the Longstaff-Schwartz algorithm as Korn presents it in [KO10]:

- Choose a number k of basis functions $H_1, ..., H_k$. In this dissertation, we choose Laguerre polynomials like Longstaff and Schwartz
- Generate the N underlying geometric brownian paths of the stock prices at each timestep: $S(t_1)^j, ..., S(t_m)^j$
- Set the terminal value for each path:

  $\hat{V}(m,j) := e^{-rt} f(S(T)^j),\ j = 1, ..., N$
- Continue backward in time and at each time t_i, $i = m-1, ..., 1$:

- Solve the regression problem and compute the $\hat{a}_i^*$:
$a_i^* = H_i^+ g(i+1)$

- Compute the estimates of the continuation values:
$\hat{C}(S(t_i)^j; i) = \sum_{l=1}^k \hat{a}_l^*(i) H_l(S(t_i)^j)),\ j = 1..., N$

- Decide to exercice or hold the option further for each path. For $j = 1, ..., N$, set:

$$\hat{V}(i,j) := \begin{cases} e^{-rt_i} f(S(t_i)^j), & \text{if } e^{-rt_i} f(S(t_i)^j) \geq \hat{C}(S(t_i)^j; i) \\ \hat{V}(i+1, j), & \text{else} \end{cases}$$

- The option price is finally:
  $\hat{V}(0) := \frac{1}{N} \sum_{j=1}^N \hat{V}(1, j)$

3.2.4 Convergence of the Longstaff-Schwartz algorithm

Clement et al. studied the convergence of the Longstaff-Schwartz algorithm in [CL02] in relation to two approximations involved in the algorithm:

- The first approximation is due to the fact that we replace the conditional expectation in the dynamic programming by projections on a finite set of basis functions

- The second approximation comes from the Monte Carlo error as the option price (expected value) is estimated by an arithmetic mean.

Korn (see [KO10]) points out that Clement et al. introduced:

$$V^k(0) = \sup_{\tau \in S(H_1, ..., H_k)} \mathbb{E}^{\mathbb{Q}}(e^{-r\tau} B_\tau)$$

'where the set $S(H_1, ..., H_k)$ only contains exercice strategies based on the solution of the regression problems with the basis functions $H_1, ..., H_k$.'
We also define $V^{k,N}(0) = \hat{V}(0)$.

As regards to the first approximation, Clement et al. shown that, if the sequence of basis functions is total in a suitable L^2-function space, the option price $V^k(0)$ converges towards the real option price with a growing number k of basis functions:

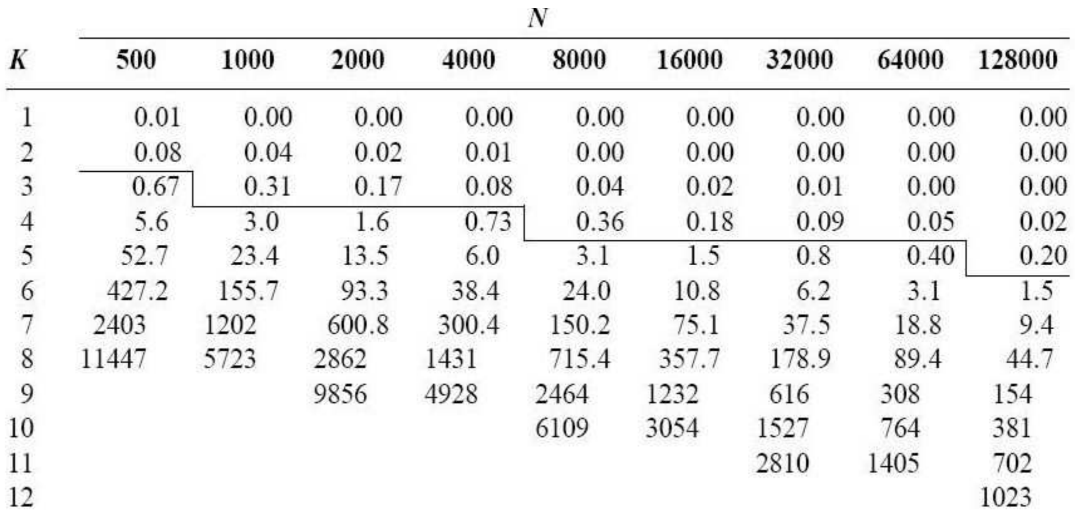
$$\lim_{k \to +\infty} V^k(0) = V(0)$$

As regards to the second approximation, $V^{k,N}(0)$ converges almost surely towards the approximating option price $V^k(0)$:

$$\lim_{N \to +\infty} V^{k,N}(0) = V^k(0)$$

Clement et al. shown that the rate of convergence towards $V^k(0)$ is $\mathcal{O}(N^{-1/2})$. The problem is that we don't have a similar result for the number of basis functions.

Glasserman and Yu investigated in [GY04] the relationship between the number of paths N and the number of basis functions k. They computed the mean square error (MSE) of the coefficients $\hat{a}$ of the regression problem. They found that the number of paths required to achieve a convergence of the $MSE(\hat{a})$ grows exponentailly with the number of basis functions used. The following figure ([GY04] p.2099) gives the estimates of $MSE(\hat{a})$ for different number of paths N and different number of basis functionsk. They used Hermite polynomials instead of Laguerre polynomials but we can reasonably think that it gives a good indication. Glasserman and Yu displayed a horizontal line indicating a limit above which the $MSE(\hat{a})$ don't converge towards zero anymore:



	N								
K	**500**	**1000**	**2000**	**4000**	**8000**	**16000**	**32000**	**64000**	**128000**
1	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2	0.08	0.04	0.02	0.01	0.00	0.00	0.00	0.00	0.00
3	0.67	0.31	0.17	0.08	0.04	0.02	0.01	0.00	0.00
4	5.6	3.0	1.6	0.73	0.36	0.18	0.09	0.05	0.02
5	52.7	23.4	13.5	6.0	3.1	1.5	0.8	0.40	0.20
6	427.2	155.7	93.3	38.4	24.0	10.8	6.2	3.1	1.5
7	2403	1202	600.8	300.4	150.2	75.1	37.5	18.8	9.4
8	11447	5723	2862	1431	715.4	357.7	178.9	89.4	44.7
9			9856	4928	2464	1232	616	308	154
10					6109	3054	1527	764	381
11							2810	1405	702
12									1023

Figure 7: Number of paths vs number of basis functions

3.3 Getting a low-biased estimator

As Giles mentions in [MG10"], the estimate obtained with the Longstaff-Schwartz algorithm tend to be low-biased because of the sub-optimal exercice boundary. However, since the optimisation of the exercice strategy is done on a finite set of paths and that we are using the same set of paths for regression and the evaluation, the estimate might be high-biased.
To be sure that the estimate be low-biased, we use two sets of paths: one of N1 paths to determine the coefficients a and an other of N2 paths for the valuation.
We thus apply the following algorithm to get a lower bound of the American option:

- Generate N2 underlying geometric brownian paths of the stock prices at each timestep: $S(t_1)^j, ..., S(t_m)^j$

- Set the terminal value for each path:

  $\hat{V}(m,j) := e^{-rt} f(S(T)^j),\ j = 1, ..., N2$

- Continue backward in time and at each time t_i, $i = m-1, ..., 1$:
  - Compute the estimates of the continuation values with the coefficients a that we got from the resolution of the regression problem with N1 paths:

  $$\hat{C}(S(t_i)^j; i) = \textstyle\sum_{l=1}^{k} \hat{a}_l^*(i) H_l(S(t_i)^j)), \ j = 1..., N$$

  - Decide to exercice or hold the option further for each path. For $j = 1, ..., N2$, set:

  $$\hat{V}(i,j) := \begin{cases} e^{-rt_i} f(S(t_i)^j), & \text{if } e^{-rt_i} f(S(t_i)^j) \geq \hat{C}(S(t_i)^j; i) \\ \hat{V}(i+1,j), & \text{else} \end{cases}$$

- The option price is finally:

  $$\hat{V}(0) := \frac{1}{N2} \sum_{j=1}^{N2} \hat{V}(1,j)$$

However , as Howard Thom underlines in [TH09], a backward method is not as efficient as a forward method since the backward method will do all the calculations rather than just the earliset. Consequently, he suggests a foward method which makes possible to stop after the first exercice date and then save compuation time. To get the lower bound, we implemented this method presented below and saved significant computation time:

- Generate N2 underlying geometric brownian paths of the stock prices at each timestep: $S(t_1)^j, ..., S(t_m)^j$

- For each path j=1, ..., N2, go forward from i=1, ..., m by performing the following steps:
  - Compute the estimates of the continuation values with the coefficients a that we got from the resolution of the regression problem with N1 paths for each timestep t_i:

$$\hat{C}(S(t_i)^j; i) = \textstyle\sum_{l=1}^{k} \hat{a}_l^*(i) H_l(S(t_i)^j)), \ j = 1..., N$$

- Decide to exercice or hold the option further for each path. If

$$\hat{C}(S(t_i)^j; i) < e^{-rt_i} f(S(t_i)^j) \ or \ i = m$$

, set

$$\hat{V}_{i,j}(0) = e^{-rt_i} f(S(t_i)^j)$$

- The option price is finally:

$$\hat{V}(0) := \frac{1}{N2} \sum_{j=1}^{N2} \hat{V}(1, j)$$

3.4 Alternative high-biased estimator

3.4.1 The theory

In the previous section, we managed to obtain a lower bound for the American option price. It remains now to compute an upper bound to judge the quality of the estimate. Here we follow Giles who gives a clear summary of the upper bound computation.
Given a martingale M with $M_0 = 0$, we have for any stopping rule τ:

$$\mathbb{E}^{\mathbb{Q}}[e^{-r\tau} B_\tau] = \mathbb{E}^{\mathbb{Q}}[e^{-r\tau} B_\tau - M_\tau] \leq \mathbb{E}^{\mathbb{Q}}[\max_k (e^{-rk} B_k - M_k)]$$

Since the above statement is true for all stopping rules τ and all martingales M, we get:

$$V_0(X_0) = \sup_\tau \mathbb{E}^{\mathbb{Q}}[e^{-r\tau} B_\tau] \leq \inf_M \mathbb{E}^{\mathbb{Q}}[\max_k (e^{-rk} B_k - M_k)]$$

This is the key duality approach. Rogers has proven equality in the continous cas (see [RO02]):

$$\sup_{\tau} \mathbb{E}^{\mathbb{Q}}[e^{-r\tau} B_{\tau}] = \inf_{M} \mathbb{E}^{\mathbb{Q}}[\max_{k}(e^{-rk} B_k - M_k)]$$

where an arbitrary τ will give a lower bound, an arbitrary M will give an upper bound. As Giles states in [MG10"], 'making both of them better shrinks the gap between them to zero'.
In order to find the upper bound, we have thus to find the optimal martingale. M. Glasserman proved in (see [GA04]) that it is equal to:

$$M_k^j = \sum_{i=1}^{k} (V(i,j) - \mathbb{E}^{\mathbb{Q}}[V(i,j) | S(t_i)^j])$$

We approximate this optimal martingale by simulating P mini-paths from each point $S(t_i)^j$ and averaging:

$$\hat{M}_k^j = \sum_{j=1}^{k} \left(V(i,j) - \frac{1}{P} \sum_{j=1}^{P} V(i,p) \right)$$

$\hat{C}(S(t_i)^j; i)$ being the approximate continuation value given by the Longstaff-Schwartz algorithm. We take:

$$V(i,j) = \max \ \left(B_{i,j}, \hat{C}(S(t_i)^j; i) \right)$$

Finally, with N simated paths, the price of the option V_{upper} will satisfy:

$$V_{upper} \approx \frac{1}{N} \sum_{j=1}^{N} \max_{i=1,\dots,m} (e^{-ri} B_{i,j} - \hat{M}_i^j))$$

3.4.2 The algorithm

Here we follow Thom H. (see [TO09]):

- Generate N underlying geometric brownian paths of the stock prices at each timestep: $S(t_1)^j, ..., S(t_m)^j$

- For each path j=1, ..., N, go forward from i=1, ..., m by performing the following steps:
  - Generate P mini-paths for each timestep t_i
  - For each mini-path p, find $\hat{V}(i,p) = \max \Big(B_{i,p}, \hat{C}(S(t_i)^p; i)\Big)$
  - Find $\hat{M}_i^j$
  - Calculate $\max\limits_{i=1,...,m} (e^{-ri} B_i - \hat{M}_i^j))$

- Compute the average of these estimates over the N paths.

4 Numerical results and discussion

4.1 Numerical results

S0	BT	FD	LSchwartz: MC \ QMC \ QMC Hybrid
10	0.611	0.6089	0.5882 \ 0.5897 \ 0.5883
25	1.5274	1.5221	1.4793 \ 1.4757 \ 1.4785
50	3.055	3.0443	2.9543 \ 2.9454 \ 2.9498

Table 3: Longstaff-Schwartz algorithm for American put options pricing

S0	LSchwartz: MC \ QMC \ QMC Hybrid
10	0.0041 \ 0.0014 \ 0.0014
25	0.011 \ 0.0043 \ 0.0039
50	0.0190 \ 0.0086 \ 0.0077

Table 4: Monte-Carlo error bounds for Longstaff-Schwartz algorithm

S0	BT	FD	Lower Bound: MC \ QMC \ QMC Hybrid
10	0.611	0.6089	0.5833 \ 0.5842 \ 0.5844
25	1.5274	1.5221	1.4653 \ 1.4649\ 1.4592
50	3.055	3.0443	2.9199 \ 2.9243 \ 2.9154

Table 5: Modified Longstaff-Schwartz algorithm: Obtention of a lower bound for American put options prices

S0	Lower Bound: MC \ QMC \ QMC Hybrid
10	0.0033 \ 0.0010 \ 0.0010
25	0.0087 \ 0.0033 \ 0.0022
50	0.022 \ 0.0053 \ 0.0050

Table 6: Monte Carlo error bounds for modified Longstaff-Schwartz algorithm (obtention of lower bounds)

S0	BT	FD	Upper Bound: MC \ QMC \ QMC Hybrid
10	0.611	0.6089	0.7532 \0.7534
25	1.5274	1.5221	1.8785 \1.8798
50	3.055	3.0443	3.7661 \3.7642

Table 7: Upper bounds for American put: simulation with 10 mini-paths

S0	Upper Bound: MC \ QMC \ QMC Hybrid
10	0.0014 \ 0.0011
25	0.0033 \ 0.0032
50	0.056 \ 0.0054

Table 8: Monte Carlo error bounds for Upper Bounds (10 mini-paths)

S0	BT	FD	Upper Bound: MC \ QMC \ QMC Hybrid
10	0.611	0.6089	0.6791 \0.6784
25	1.5274	1.5221	1.6981 \1.6952
50	3.055	3.0443	3.3922 \3.3932

Table 9: Upper bounds for American put: simulation with 100 mini-paths

S0	Upper Bound: MC \ QMC \ QMC Hybrid
10	0.00006 \ 0.00005
25	0.0015 \ 0.0015
50	0.0025 \ 0.0024

Table 10: Monte Carlo error bounds for Upper Bounds (100 mini-paths)

Algorithm	Price: MC \ QMC Hybrid
Longstaff-Schwartz	1.4641 \ 1.4646
Lower Bound	1.4646 \ 1.4643

Table 11: American put prices for N=32 000 paths, S_0=K=25

Algorithm	Price: MC \ QMC Hybrid
Longstaff-Schwartz	0.0029 \ 0.0005
Lower Bound	0.0015\ 0.007

Table 12: MC Error bounds for N=32 000 paths, S_0=K=25

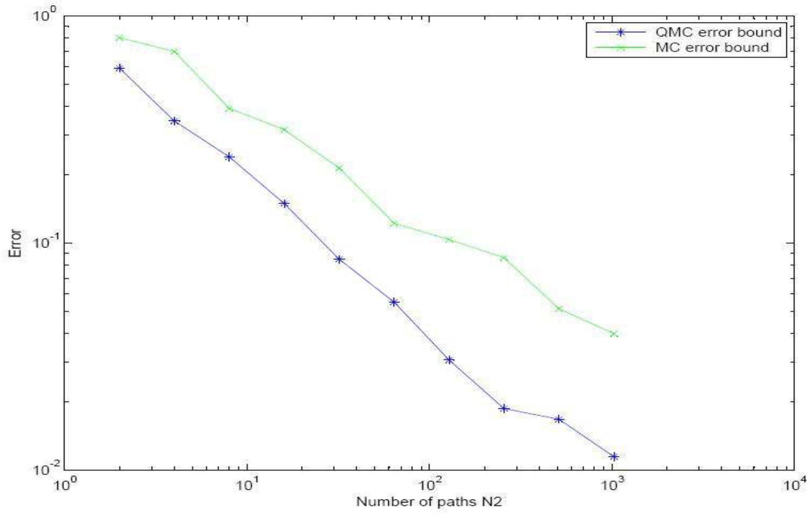


Figure 8: QMC-HYB and MC convergence for Lower Bounds of American put

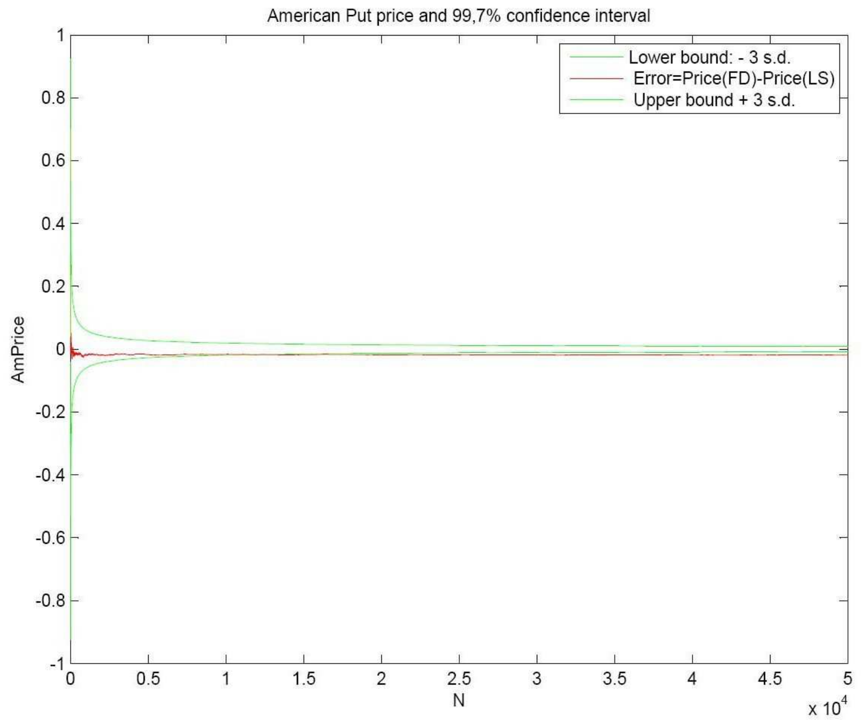


Figure 9: Confidence interval for American put price obtained with the Longstaff-Schwartz algorithm

4.2 Discussion

In tables 3 to 10, we give the values of at-the-money American put options. As regards to control methods, we use binomial tree (BT) pricing and Finite Difference methods (FD).
For binomial tree pricing, we use the online simulator of Columbia University and take a number of nodes equal to 100 (http://www.math.columbia.edu).
For Finite Differences, we use an algorithm designed by Dr Christoph Reisinger of Oxford Univeristy based on a Crank-Nicolson method using a projected Gauss-Seidel algorithm to solve the linear complementarity problem in each timestep (see [RE10]).

We choose an interest rate r equal to 0.05, a volatility σ equal to 0.2, a maturity of 1 year ($T = 1$) and we take 4 basis functions. All the options are valued when they are at-the-money. Consequently the strike is always equal to S_0.
We also always take 128 timesteps in order to reduce as much as possible the weak error. The simulations are based on 1000 paths for tables 3 to 10. The Monte Carlo error bounds (plus or minus three standard deviations) are determined thanks to a set of 16 sets points.

Tables 3 to 10 show clearly that prices given by the binomial tree and Finite Differences methods lie everytime between the lower bounds and the upper bounds determined by our algorithms.
The upper bounds clearly decrease when we pass from 10 to 100 mini-paths. Using more mini-paths should enable us to come closer to the price given by the other methods: the accuracy of high-biased estimates benefits clearly from the increase of mini-paths. However, using more mini-paths increases a lot the computational cost.

Using Quasi-Monte Carlo always reduces standard deviations. This is clear for prices obtained with the Longstaff-Schwartz algorithm and the algorithm giving lower bounds. However, the benefits of Quasi-Monte Carlo for upper bounds are not obvious. . Increasing the number of paths increases the difference between MC and QMC errors. This is clear by comparing the table 4 and the table 12. The table 12 shows that using QMC-Hybrid divided the standard deviation by 6. Tables 3 to 6 (simulations with 1000 paths) show that the standard deviation is divided by 3 in average when we use hybrid QMC methods. The figure 8 also show that for a same number of paths, the standard deviation is lower for QMC. It converges towards zero quicker. It is interesting to see that the slope of the MC error is approximatively equal to -1/2 as expected wheras the slope for the QMC error is a bit lower. In the case of European options, the slope was about equal to -1. Actually, we could reasonably think that the slope of QMC error is between -1 and -1/2.

The figure 9 shows the confidence interval for the American put price obtained with the Longstaff-Schwartz algorithm. The error corresponds to the difference between the price computed with the Longstaff-Schwartz algorithm and the price computed with the Finite Differences scheme. The bounds are plus and minus three standard deviations. Thus, if we consider the price given by the Finite Differences method as the fair price, the error should lie between those bounds. We can notice that the error is very small and that it tends to approach the lower bound, which points out that the price calculated with the Longstaff-Schwartz algorithm is rather low-biased.

In addition, we can notice that the QMC-Hybrid is more efficient than a simple QMC simulation.

5 Conlusion

This paper has presented three algorithms. We managed to obtain lower and upper bounds for the American put prices. But the main goal of this dissertation was to know if using Quasi-Monte Carlo methods is more efficient than pure Monte Carlo methods.
We can conlude that QMC techniques improve the efficiency in most of the cases: standard deviations were always lower (or almost equal in the case of upper bounds) for QMC than for MC methods, it was never higher. The fact that the Sobol sequences, even after randomisation, are more evenly dispersed makes the rate of convergence faster.
Consequently, we would recommend to use Quasi-Monte Carlo simulations for American options pricing, all the more that generating Sobol sequences doesn't have a big computational cost. For a small number of paths, we have seen that the error can be divided by 6. For a large number of paths (more than 100,000), the error is divided by more than 10. The figure 5 showed that the error is divided by more than 100 for Europen options, even with a small number of paths.

There were three sources of errors: the discretisation of the exercise dates, the fact that we had a finite number of basis functions to estimate continuation value and the classic Monte Carlo error. As regards to basis functions, we used Laguerre polynomials for all the algorithms, as suggested in the original paper written by Longstaff and Schwartz (see [LS01]). Despite the number of paths grows exponentially with the number of basis functions, it would be interesting to measure properly the influence of the number of basis functions as well as the choice of the basis functions.

It could be interesting to extend the work done in this paper to multiple dimensions problems and thus to compare the efficiency of Finite Differences and Quasi-Monte Carlo methods. Determining the greeks could also be an area of future research. The possible improvements include the use of jump diffusion models and the use of multilevel QMC or adaptive QMC.

Appendices

A Matlab Code for the implementation of Brownian Bridge and PCA: Provided by Mike Giles

```
%
% calculation of multi-dimensional Brownian Bridge
%
% function dW = bb(Z,T)
% function dW = bb(Z,T,C)
%
% T        -- time interval
% Z(:,:)   -- input vectors of unit Normal variables
% C(:,:)   -- square PCA matrix for multi-dimensional Brownian motion
% (optional)...
% dW(:,:) -- output vectors of Brownian path increments
%
% first dimension of Z/dW corresponds to #timesteps * Brownian dimension
% second dimension of Z/dW corresponds to #paths
%

function W = bb(W,T,C)

%if nargin==0
%  bb_test;
%  return
%end
```

```
[N M] = size(W);

K = 1;

L = round(log(N)/log(2));

if (N~=2^L)
  error('error: size not a power of 2')
end

K = 1;

%
% for multi-dimensional Brownian motion, reorder input
% to get contiguous time
%

if nargin > 2
  K = size(C,1);
  N = N/K;
  M = M*K;

  p = reshape((reshape((1:N*K)',K,N))',N*K,[]);
  W = reshape(W(p,:),N,[]);
end

%
% perform Brownian Bridge interpolation
%
```

```
for m = 1:L
  W([1:2:2^m 2:2:2^m],:) = ...
   [ 0.5*W(1:2^(m-1),:)+W(2^(m-1)+1:2^m,:)/sqrt(2)^(m+1) ; ...
     0.5*W(1:2^(m-1),:)-W(2^(m-1)+1:2^m,:)/sqrt(2)^(m+1) ];
end

%
% for multi-dimensional Brownian motion, reorder output
% to get contiguous multiple dimensions and apply PCA
%

if nargin > 2
  W = reshape(W,N*K,[]);
  p = reshape((reshape((1:N*K)',N,K))',N*K,[]);
  W = reshape(W(p,:),K,[]);
  W = reshape(C*W,N*K,[]);
end

%
% finally, adjust for non-unit time interval
%

W = sqrt(T)*W;

return

% calculation of multi-dimensional PCA
%
% dW = pca(Z,T,option)
% dW = pca(Z,T,option,C)
```

```
%
% T       -- time interval
% Z(:,:)  -- input vectors of unit Normal variables
% C(:,:)  -- square PCA matrix for multi-dimensional Brownian motion
% (optional)...
% dW(:,:) -- output vectors of Brownian path increments
%
% first dimension of Z/dW corresponds to #timesteps * Brownian dimension
% second dimension of Z/dW corresponds to #paths
%
% option: 'mat'    -- matrix-vector version
%         'fft'    -- FFT version
%         'bb_mat' -- matrix-vector version of Brownian Bridge hybrid
%         'bb_fft' -- FFT version of Brownian Bridge hybrid
%

function W = pca(Z,T,option,varargin)

if nargin==0
  pca_test;
  return
end

[N M] = size(Z);

K = 1;

%
% for multi-dimensional Brownian motion, reorder input
% to get contiguous time
```

```
%

if nargin > 3
  C = varargin{1};
  K = size(C,1);
  N = N/K;
  M = M*K;

  p = reshape((reshape((1:N*K)',K,N))',N*K,[]);
  Z = reshape(Z(p,:),N,[]);
end

%
% perform time PCA
%

switch option
  case 'bb_fft'
    Z1 = Z(1,:);

    D = sqrt(T/(2*N^2)) ./ sin(0.5*pi*(1:N-1)/N);
    Z = diag(D) * Z(2:end,:);

    Wh = zeros(2*N,M);
    Wh(2:N,:) = complex(0,0.5)*Z;
    Wh(2*N:-1:N+2,:) = conj(Wh(2:N,:));

    W = fft(Wh);
    W = real(W(2:N+1,:));
    W = W + sqrt(T)*((1:N)/N)'*Z1;

  case 'bb_mat'
    D = sqrt(T/(2*N^2)) ./ sin(0.5*pi*(1:N-1)/N);

    U = zeros(N,N-1);
    for n = 1:N-1
      U(:,n) = sin((1:N)*pi*n/N);
    end

    W = U*diag(D)*Z(2:end,:) + sqrt(T)*((1:N)/N)'*Z(1,:);

  case 'fft'
    D = sqrt(T/(N*(2*N+1))) ./ sin(0.5*pi*(2*(1:N)-1)/(2*N+1));
    Z = diag(D) * Z;

    Wh = zeros(4*N+2,M);
    Wh(2:2:2*N,:) = complex(0,0.5)*Z;
    Wh(4*N+2:-1:2*N+4,:) = conj(Wh(2:2*N,:));

    W = fft(Wh);
    W = real(W(2:N+1,:));

  case 'mat'
    D = sqrt(T/(N*(2*N+1))) ./ sin(0.5*pi*(2*(1:N)-1)/(2*N+1));

    U = zeros(N,N);
    for n = 1:N
      U(:,n) = sin((1:N)*pi*(2*n-1)/(2*N+1));
    end
```

```
    W = U*diag(D)*Z;
end

%
% for multi-dimensional Brownian motion, reorder output
% to get contiguous multiple dimensions and apply PCA
%

if nargin > 3
  W = reshape(W,N*K,[]);
  p = reshape((reshape((1:N*K)',N,K))',N*K,[]);
  W = reshape(W(p,:),K,[]);
  W = reshape(C*W,N*K,[]);
end

%
% finally, compute increments if more than one timestep
%

if N>1
  W(K+1:end,:) = W(K+1:end,:) - W(1:end-K,:);
end

return

%
%-----------------------------------------------------------
%
```

```
function pca_test

N = 128;
M = 10;
T = 1;

K = 1;

randn('state',0)
Z = randn(N*K,M);

%
% check 'bb_fft" and 'bb_mat' equivalence
%

W1 = pca(Z,T,'bb_fft');
W2 = pca(Z,T,'bb_mat');
disp(sprintf('difference between bb_fft and bb_mat = %g',...
             norm(W1-W2)/sqrt(M*N)))

%
% check 'fft" and 'mat' equivalence
%

W1 = pca(Z,T,'fft');
W2 = pca(Z,T,'mat');
disp(sprintf('difference between   fft  and   mat  = %g',...
             norm(W1-W2)/sqrt(M*N)))

return
```

B Matlab Code for QMC European Put options pricing

```
function V = european_put(r,sigma,T,S,K)

% r     - interest rate
% sigma - volatility
% T     - time interval
% S     - asset value(s)
% K     - strike price(s)
% V     - option value(s)

S  = max(1e-40*K,S);     % avoids problems with S=0

d1 = ( log(S) - log(K) + (r+0.5*sigma^2)*T ) / (sigma*sqrt(T));
d2 = ( log(S) - log(K) + (r-0.5*sigma^2)*T ) / (sigma*sqrt(T));


    V =K*exp(-r*T)*N(-d2)-S.*N(-d1);

% Normal cumulative distribution function

function ncf = N(x)

%ncf = 0.5*(1+erf(x/sqrt(2)));

xr = real(x);
xi = imag(x);

if abs(xi)>1e-10
```

```
  error 'imag(x) too large in N(x)'
end

ncf = 0.5*(1+erf(xr/sqrt(2))) ...
    + i*xi.*exp(-0.5*xr.^2)/sqrt(2*pi);

function v=europutmc(L,M)

r   = 0.05;
sig = 0.2;
T   = 1;
S0  = 100;
K   = 110;

N=2^L;

h=T/N;

w=randn(N,M)*sqrt(h);

S = S0*ones(1,M);

for i=1:N
  S=S+r*S*h+sig*S.*w(i,:);
end

v=exp(-r*T)*max(S-K,0);

v=sum(v)/M;
```

```
function v=europutpcascrambled(L,M,option)

r   = 0.05;
sig = 0.2;
T   = 1;
S0  = 100;
K   = 110;

N=2^L;
h=T/N;

p=sobolset(N);
p = scramble(p,'MatousekAffineOwen');
Q=net(p,M)';
Q=max(Q,eps(1));
Q=min(Q,1-eps(1));
Z=norminv(Q);

switch option
    case 'bb_fft'
       w=pca(Z,T,'bb_fft');
    case 'bb_mat'
       w=pca(Z,T,'bb_mat');
    case 'fft'
       w=pca(Z,T,'fft');
    case 'mat'
       w=pca(Z,T,'mat');
end
```

```
S = S0*ones(1,M);

for i=1:N
  S=S+r*S*h+sig*S.*w(i,:);
end

v=exp(-r*T)*max(K-S,0);
v=sum(v)/M;

function []=european(L,M)

r   = 0.05;
sig = 0.2;
T   = 1;
S0  = 100;
K   = 110;


QMC1=zeros(1,64);
QMC2=zeros(1,64);
QMC3=zeros(1,64);

MC=zeros(1,64);

NbPath=2.^(1:M);
vecterrorQMC1=zeros(1,length(NbPath));
vecterrorQMC2=zeros(1,length(NbPath));
vecterrorQMC3=zeros(1,length(NbPath));
vecterrorMC=zeros(1,length(NbPath));
```

```
for k=1:M
    k
    j = 2^k;
    for i=1:64
        QMC1(1,i)=europutpcascrambled(L,j,'bb_fft');
        QMC2(1,i)=europutpcascrambled(L,j,'bb_mat');
        QMC3(1,i)=europutpcascrambled(L,j,'mat');
        MC(1,i)=europutmc(L,j);
    end

errorQMC1=3*sqrt(sum((mean(QMC1)*ones(1,64)-QMC1).^2)/(64*(64-1)));
errorQMC2=3*sqrt(sum((mean(QMC2)*ones(1,64)-QMC2).^2)/(64*(64-1)));
errorQMC3=3*sqrt(sum((mean(QMC3)*ones(1,64)-QMC3).^2)/(64*(64-1)));
errorMC=3*sqrt(sum((mean(MC)*ones(1,64)-MC).^2)/(64*(64-1)));

vecterrorQMC1(k)=errorQMC1;
vecterrorQMC2(k)=errorQMC2;
vecterrorQMC3(k)=errorQMC3;
vecterrorMC(k)=errorMC;


end

loglog(NbPath,vecterrorQMC1,'b-*',NbPath,vecterrorQMC2,'r-*',NbPath,...
    vecterrorQMC3,'y-x',NbPath,vecterrorMC,'g-x')
 xlabel('M'); ylabel('Error');
 legend(' QMC-BBFFT error bound',' QMC-BBPCA error bound',...
     ' QMC-PCA error bound',' MC error bound',1)

function []=comparaison3(L,M,option)
```

```
r   = 0.05;
sig = 0.2;
T   = 1;
S0  = 100;
K   = 110;


QMC=zeros(1,64);
MC=zeros(1,64);

NbPath=2.^(1:M);
vecterrorQMC=zeros(1,length(NbPath));
vecterrorMC=zeros(1,length(NbPath));
vectexacterrorQMC=zeros(1,length(NbPath));
vectexacterrorMC=zeros(1,length(NbPath));

Ve  = european_call(r,sig,T,S0,K,'value');

for k=1:M
    k
    j = 2^k;
    for i=1:64
        QMC(1,i)=europutpcascrambled(L,j,option);
        MC(1,i)=europutmc(L,j);
    end

errorQMC=3*sqrt(sum((mean(QMC)*ones(1,64)-QMC).^2)/(64*(64-1)));
errorMC=3*sqrt(sum((mean(MC)*ones(1,64)-MC).^2)/(64*(64-1)));
```

```
vecterrorQMC(k)=errorQMC;
vecterrorMC(k)=errorMC;

vectexacterrorQMC(k)=abs(mean(QMC)-Ve);
vectexacterrorMC(k)=abs(mean(MC)-Ve);

end

loglog(NbPath,vecterrorQMC,'b-*',NbPath,vectexacterrorQMC,'r-*',NbPath,...
    vecterrorMC,'g-x',NbPath,vectexacterrorMC,'y-x')
 xlabel('M'); ylabel('Error');
 legend(' QMC error bound',' QMC Exact Error',' MC error bound',...
     ' MC Exact Error',1)
```

C Matlab Code for QMC American Put options pricing

```
function LSLaguerreValue=LSLaguerre(k,x)
switch k
    case 0
       LSLaguerreValue=exp(-0.5*x).*1;
    case 1
       LSLaguerreValue=exp(-0.5*x).*(-x+1);
    case 2
       LSLaguerreValue=exp(-0.5*x).*[1/2*(x.^2-4*x+2)];
    case 3
       LSLaguerreValue=exp(-0.5*x).*[1/6*(-x.^3+9*x.^2-18*x+6)];
    case 4
```

```
        LSLaguerreValue=exp(-0.5*x).*[1/24*(x.^4-16*x.^3+72*x.^2-96*x+24)];
    case 5
        LSLaguerreValue=exp(-0.5*x).*[1/120*(-x.^5+25*x.^4-200*x.^3+...
            600*x.^2-600*x+120)];
    case 6
        LSLaguerreValue=exp(-0.5*x).*[1/720*(x.^6-36*x.^5+450*x.^4-...
            2400*x.^3+5400*x.^2-4320*x+720)];
    case 7
        LSLaguerreValue=exp(-0.5*x).*[1/5040*(-x.^7+49*x.^6-882*x.^5+...
            7350*x.^4-29400*x.^3+52920*x.^2-35280*x+5040)];
    case 8
        LSLaguerreValue=exp(-0.5*x).*[1/40320*(x.^8-64*x.^7+1568*x.^6-...
            18816*x.^5+117600*x.^4-376320*x.^3+564480*x.^2-322560*x+40320)];
    otherwise
         disp('Error: You should choose k between 0 and 8');
end

function S=GenerateGBMPaths(N,M,T,S0,r,sigma,option)
% 2^L=M: number of timesteps
% N: number of paths




deltaT=T/M;

p=sobolset(M);
p = scramble(p,'MatousekAffineOwen');
Q=net(p,N)';
Q=max(Q,eps(1));
Q=min(Q,1-eps(1));
```

```
Z=norminv(Q);

switch option
    case 'bb_fft'
       w=pca(Z,T,'bb_fft');
    case 'bb_mat'
       w=pca(Z,T,'bb_mat');
    case 'fft'
       w=pca(Z,T,'fft');
    case 'mat'
       w=pca(Z,T,'mat');
    case 'MC'
       w=sqrt(deltaT)*randn(M,N);
end

S=zeros(M+1,N);
S(1,:)=S0*ones(1,N);

    for m=1:M
        S(m+1,:)=S(m,:).*exp((r-0.5*sigma*sigma)*deltaT+sigma*w(m,:));
    end

function coeff=regression(k,N1,L,T,S0,K,r,sigma,option)
%GIVE THE COEFF OF THE REGRESSION

% tic;
M=2^L;
deltaT=T/M;
```

```
payoff = @(x) max(K-x,0) ; % Put Payoff

%Generate N1 GBM independant paths of the stock price at the M possible
%exercice times of the options
X=zeros(M+1,N1);
X=GenerateGBMPaths(N1,M,T,S0,r,sigma,option)/S0;

%Fix the terminal values of the Bermudan option for each path
V=zeros(M+1,N1);
V(M+1,:)=exp(-r*T)*payoff(S0*X(M+1,:));


%Solve the regression problem and determine the matrix hplus
h=zeros(N1,k,M+1);
hplus=zeros(k,N1,M+1);

         for  l=1:k

         h(:,l,:)=LSLaguerre(l,X(:,:))'  ;
         end

for i=1:M+1

 hplus(:,:,i)=(h(:,:,i)'*h(:,:,i))\ h(:,:,i)';
end

%Continue backward in time
a=zeros(k,M+1); %optimal weights
C=zeros(M+1,N1); %continuation value
bound=zeros(1,N1);% 3 times the standard deviation
```

```
for i=M:-1:1

        a(:,i)=hplus(:,:,i)*V(i+1,:)'; % calculate optimal weights

        for j=1:N1
        C(i,j)=a(:,i)'*h(j,:,i)';   % estimation of continuation value
            if exp(-r*(i-1)*deltaT)*payoff(S0*X(i,j))> C(i,j)
                V(i,j)=exp(-r*i*deltaT)*payoff(S0*X(i,j));
            else
                V(i,j)=V(i+1,j);
            end
%             switch i
%                case 1
%                    price(j)=sum(V(1,1:j))/j;
%                    bound(j)=3*sqrt(sum((mean(V(1,1:j))*...
%                    ones(1,j)-V(1,1:j)).^2)/(j*(j-1)));
%             end
        end

end

coeff=a;

function AmPrice=LS(k,N,L,T,S0,K,r,sigma,option);
% GIVE THE LSCHWARTZ PRICE OF THE AMERICAN PUT
tic;
M=2^L;
deltaT=T/M;
```

```
payoff = @(x) max(K-x,0) ; % Put Payoff

%Generate N GBM independant paths of the stock price at the M possible
%exercice times of the options
X=zeros(M+1,N);
X=GenerateGBMPaths(N,M,T,S0,r,sigma,option)/S0;

%Fix the terminal values of the Bermudan option for each path
V=zeros(M+1,N);
V(M+1,:)=exp(-r*T)*payoff(S0*X(M+1,:));


%Solve the regression problem and determine the matrix hplus
h=zeros(N,k,M+1);
hplus=zeros(k,N,M+1);

         for  l=1:k
         h(:,l,:)=LSLaguerre(l,X(:,:))'  ;
         end

for i=1:M+1
 hplus(:,:,i)=(h(:,:,i)'*h(:,:,i))\ h(:,:,i)';
end

%Continue backward in time
a=zeros(k,M+1); %optimal weights
C=zeros(M+1,N); %continuation value
bound=zeros(1,N);% 3 times the standard deviation
```

```
for i=M:-1:1

        a(:,i)=hplus(:,:,i)*V(i+1,:)'; % calculate optimal weights

        for j=1:N
        C(i,j)=a(:,i)'*h(j,:,i)';   % estimation of continuation value
            if exp(-r*(i-1)*deltaT)*payoff(S0*X(i,j))> C(i,j)
                V(i,j)=exp(-r*i*deltaT)*payoff(S0*X(i,j));
            else
                V(i,j)=V(i+1,j);
            end
            switch i
               case 1
                   price(j)=sum(V(1,1:j))/j;
                   bound(j)=3*sqrt(sum((mean(V(1,1:j))*...
                       ones(1,j)-V(1,1:j)).^2)/(j*(j-1)));
            end
        end

end

% Option price
AmPrice=sum(V(1,:))/N;
% toc;

%Comparison to Finite Differences method for ATM american put
disp('Finite Differences give for Am Put ATM: ');
AmericanThetaScheme_ProjGaussSeidel(100, 200, 1/2,K,T,r,sigma, 0)

%Graph with confidence interval
```

```
axis([1 N -inf inf]);
plot(1:N,price-bound,'g',1:N,price,'r',1:N,price+ bound,'g')
xlabel('N'); ylabel('AmPrice');
legend('Lower bound',' AmPrice',' Upper bound',1);
title('American Put price and 99,7% confidence interval');



disp('American Put price with LS method: ');


% %Verification for put value
% EuroPutValue(S0, K, r, sigma,T)
% axis([1 N -inf inf]);
% plot(1:N,-bound,'b',1:N,EuroPutValue(S0, K, r, sigma,T)-price,'r',...
1:N,bound,'b')


toc;

function AmPrice=LowerBound2(k,a,N2,L,T,S0,K,r,sigma,option)


%N2: Number of paths in the new set
%M: Number of timesteps

% tic;
M=2^L;
deltaT=T/M;
```

```
payoff = @(x) max(K-x,0) ; % Put Payoff

%LOWER BOUND BOUND

%Generate N2 GBM independant paths of the stock price at the M possible
%exercice times of the options
Xl=zeros(M+1,N2);
Xl=GenerateGBMPaths(N2,M,T,S0,r,sigma,option)/S0;
Vl=zeros(M+1,N2);
% Continue forward in time
        hl=zeros(N2,k,M+1);
        for  l=1:k
        hl(:,l,:)=LSLaguerre(l,Xl(:,:))'  ;
        end

Cl=zeros(M+1,N2); %continuation value
bound=zeros(1,N2-1);
price=zeros(1,N2-1);

for j=1:N2

       for i=2:M+1

           Cl(i,j)=a(:,i)'*hl(j,:,i)';   % estimation of continuation value
            if exp(-r*(i-1)*deltaT)*payoff(S0*Xl(i,j))> Cl(i,j) || i==M+1
                Vl(1,j)=exp(-r*i*deltaT)*payoff(S0*Xl(i,j));
                break;
            else
                Vl(i,j)=Vl(i-1,j);
```

```
          end

     if j~=1
     price(j-1)=sum(Vl(1,1:j))/j;
     bound(1,j-1)=3*sqrt(sum((mean(Vl(1,1:j))*ones(1,j)-...
     Vl(1,1:j)).^2)/(j*(j-1)));
     end

     end

end

%Graph with confidence interval
axis([1 N2 -inf inf]);
plot(2:N2,price-bound,'g',2:N2,price,'r',2:N2,price+ bound,'g')
xlabel('N2'); ylabel('AmPrice Low-Biased');
legend('Lower bound',' AmPrice_Low-Biased',' Upper bound',1);
title('American Put Low-Biased price and 99,7% confidence interval');

bound(1,N2-1)/3
disp('American Put Low-Biased price with LS method: ');

AmPrice=sum(Vl(1,:))/N2;



% toc;

function AmPrice=UpperBound2(k,a,N2,N3,L,T,S0,K,r,sigma,option)
%GIVE AN UPPER BOUND OF THE AMERICAN PUT
```

```
%a: coeff from the regression
%N2: Number of Mini-paths
%M: Number of timesteps


M=2^L;
deltaT=T/M;


payoff = @(x) max(K-x,0) ; % Put Payoff

%Generate N2 GBM independant paths of the stock price at the M possible
%exercice times of the options
X=zeros(M+1,N2);
X=GenerateGBMPaths(N2,M,T,S0,r,sigma,option)/S0;

%UPPER BOUND

Xmini=zeros(N3,N2);
HighPrice=zeros(1,N2);
martingale=zeros(1,N2);
Delta1=zeros(1,N2);
CVmini=zeros(N3,N2);


    for i=1:M

        Xf=X(i+1,:);
        Xi=X(i,:);
        Dpayoff=exp(-r*deltaT*i)*payoff(S0*Xf);

        %We calcul the approximate value function, first term of delta,...
        %defined
        %as delta1
        BasisFunction=zeros(k,N2);
        for  l=1:k
                 BasisFunction(l,:)=LSLaguerre(l,Xf)  ;
        end

        Delta1=max(Dpayoff,a(:,i)'*BasisFunction);

        %We calcul the conditional expectation term of delta, defined by...
        %delta2
                % Simulation of Mini-Paths
W=zeros(N3,N2);
switch option
    case 'bb_fft'
for l=1:N2
        p=sobolset(1);
        p = scramble(p,'MatousekAffineOwen');
        Q=net(p,N3)';
        Q=max(Q,eps(1));
        Q=min(Q,1-eps(1));
        Z=norminv(Q);
        W(:,l)=pca(Z,deltaT,'bb_fft');
end
    case 'bb_mat'
  for l=1:N2
        p=sobolset(1);
        p = scramble(p,'MatousekAffineOwen');
        Q=net(p,N3)';
        Q=max(Q,eps(1));
        Q=min(Q,1-eps(1));
        Z=norminv(Q);
        W(:,l)=pca(Z,deltaT,'bb_mat');
  end
    case 'fft'
     for l=1:N2
        p=sobolset(1);
        p = scramble(p,'MatousekAffineOwen');
        Q=net(p,N3)';
        Q=max(Q,eps(1));
        Q=min(Q,1-eps(1));
        Z=norminv(Q);
        W(:,l)=pca(Z,deltaT,'fft');
    end
    case 'mat'
      for l=1:N2
        p=sobolset(1);
        p = scramble(p,'MatousekAffineOwen');
        Q=net(p,N3)';
        Q=max(Q,eps(1));
        Q=min(Q,1-eps(1));
        Z=norminv(Q);
        W(:,l)=pca(Z,deltaT,'mat');
      end
    case 'MC'
       W=sqrt(deltaT)*randn(N3,N2);
end
```

```
%          Xmini= exp((r-0.5*sigma*sigma)*deltaT+sigma*W).*repmat(Xi,N3,1);
       Xmini= (1+r*deltaT+ sigma*W).*repmat(Xi,N3,1);

                % Determine the continuation value for the sub-paths

        for p=1:N3
            for  l=1:k
            BasisMini(l,:)=LSLaguerre(l,Xmini(p,:));
            end
        CVmini(p,:)=a(:,i)'*BasisMini;
        end

        Vminipath=max(exp(-r*deltaT*i)*payoff(S0*Xmini),CVmini);
        Delta2=mean(Vminipath,1);

        %Update of martingale
        martingale=martingale+(Delta1-Delta2);

        %Update option price along paths
        HighPrice=max(HighPrice, Dpayoff-martingale);

    end

%Final time step
Dpayoff=exp(-r*deltaT*(M+1))*payoff(S0*X(end,:));
Xi=X(M,:);
W=randn(N3,N2);
```

```
Xmini=exp((r-0.5*sigma*sigma)*deltaT+sigma*W*...
     sqrt(deltaT)).*repmat(Xi,N3,1);
Vminipath=exp(-r*deltaT*(M+1))*payoff(S0*Xmini);
martingale=martingale+(Dpayoff-mean(Vminipath,1));
HighPrice=max(HighPrice,Dpayoff-martingale);

AmPrice=sum(HighPrice)/N2;

%Graph with confidence interval
axis([1 N2 -inf inf]);
iterations=2:1:N2;
% bound=3*sqrt(sum((mean(HighPrice)*ones(1,N2)-HighPrice).^2)./...
%(iterations*(iterations-1)));
bound=3*sqrt(sum((mean(HighPrice(2:N2))*ones(1,N2-1)-...
      HighPrice(2:N2)).^2)./(iterations.*(iterations-1)));
plot(2:N2,HighPrice(2:N2)-bound,'g',2:N2,HighPrice(2:N2),'r',...
     2:N1,HighPrice(2:N2)+ bound,'g')
xlabel('N2'); ylabel('AmPrice-High Biaised');
legend('Lower bound',' AmPrice-High Biaised',' Upper bound',1);
title('American Put price and 99,7% confidence interval');

bound(1,N2-1)/3


function error=ErrorLS(S0,K,L,N,option)
tic;

r   = 0.05;
sigma = 0.2;
T   = 1;
```

```
k=4;

QMC=zeros(1,32);
MC=zeros(1,32);


    for i=1:32
        i
        QMC(1,i)=LS(k,N,L,T,S0,K,r,sigma,option);
        MC(1,i)=LS(k,N,L,T,S0,K,r,sigma,'MC');

    end

error(2,2)=sqrt(sum((mean(QMC)*ones(1,32)-QMC).^2)/(32*(32-1)));
error(1,2)=sqrt(sum((mean(MC)*ones(1,32)-MC).^2)/(32*(32-1)));
error(1,1)=mean(MC);
error(2,1)=mean(QMC);
toc;

function error=ErrorLower(a,S0,K,L,N2,option)
tic;

r   = 0.05;
sigma = 0.2;
T   = 1;
k=4;

QMC=zeros(1,16);
MC=zeros(1,16);
```

```
    for i=1:16
        i
        QMC(1,i)=LowerBound2(k,a,N2,L,T,S0,K,r,sigma,option);
        MC(1,i)=LowerBound2(k,a,N2,L,T,S0,K,r,sigma,'MC');

    end

error(2,2)=sqrt(sum((mean(QMC)*ones(1,16)-QMC).^2)/(16*(16-1)));
error(1,2)=sqrt(sum((mean(MC)*ones(1,16)-MC).^2)/(16*(16-1)));
error(1,1)=mean(MC);
error(2,1)=mean(QMC);
toc;

function error=ErrorUpper(a,b,S0,K,L,N2,N3,option)
tic;

r   = 0.05;
sigma = 0.2;
T   = 1;
k=4;

QMC=zeros(1,16);
MC=zeros(1,16);


    for i=1:16
        i
        QMC(1,i)=UpperBound2(k,b,N2,N3,L,T,S0,K,r,sigma,option);
        MC(1,i)=UpperBound2(k,a,N2,N3,L,T,S0,K,r,sigma,'MC');
    end

error(2,2)=sqrt(sum((mean(QMC)*ones(1,16)-QMC).^2)/(16*(16-1)));
error(1,2)=sqrt(sum((mean(MC)*ones(1,16)-MC).^2)/(16*(16-1)));
error(1,1)=mean(MC);
error(2,1)=mean(QMC);
toc;
```

References

[AK98] Åkesson et al.,Discrete eigenfunction expansion of multi-dimensional Brownian motion and the Ornstein-Uhlenbeck process, *Carnegie Mellon University*, 1998

[BR88] Bratley et al., Implementing Sobol's Quasirandom Sequence Generator, *University of Montreal*, 1988

[CL02] Clement et al., An analysis of a least squares regression method for American option pricing, *Finance and Stochastics*, 2002

[JS08] Joe et al., Constructing Sobol's sequences with better two-dimensional projections, *SIAM J. Sci. Comput.*, 2008

[JAU08] Jauvion G., Pricing d'options en Quasi Monte Carlo, *Ecole Central Paris*, 2008

[GA04] P. Glasserman, *Monte-carlo Methods in Financial Engineering*, Springer, 2004

[GY04] Glasserman P. and Yu B., Number of paths versus number of basis functions in American option pricing, *Columbia Universityl*, 2004

[KO10] Korn et al., *Monte-carlo Methods and Models in Finance and Insurance*, CRC Press, 2010

[LS01] Longstaff F. and Schwartz E., Valuing american options by simulation: a simple least-square approach, *The Review of Financial Studies*, 2001

[MG07] Giles M., Use of FFT for fast PCA construction, 2007

[MG08] Giles M. et al., Quasi-Monte Carlo for finance applications, 2008

[MG10] Giles M., Lecture Notes: MScMCF: Numerical Methods II, Lecture 6, 2010

[MG10'] Giles M., Lecture Notes: MScMCF: Numerical Methods II, Lecture 13, 2010

[MG10"] Giles M., Lecture Notes: MScMCF: Numerical Methods II, Lecture 15, 2010

[NH92] Niederreiter H.,*Random number generation and quasi-Monte Carlo methods*, SIAM, Philadelphia, 1992

[OW97] Owen, A.B.,Scrambled net variance for integrals of smooth functions, *Annals of statistics*, 1997

[RE10] Reisinger C., Finite Difference Methods lectures, Problem Sheet 6, *University of Oxford*, 2010

[RO02] Rogers L.C.G., Monte Carlo valuation of American options, *Mathematical Finance*, Springer, 2002

[SC10] Schreve S., *Stochastic Calculus For Finance II, Continous-Time Models*, Springer, 1998

[TH09] Thom H., Longstaff-Schwartz Pricing of Bermudan Options and their Greeks, *University of Oxford*, Master's thesis, 2009