Monte–Carlo methods for Pricing American Style Options.

Pierre Cohort.

Premia 5

Contents

1 Introduction.

In this paper, we present some recent Monte–Carlo algorithms devoted to the pricing of american (more precisely, bermudean) style options. We also describe the implementations of these algorithms and the related Scilab interface : the amclib Scilab library.

First, we briefly recall the market model to be used (section (2)) and the minimal mathematical background of the pricing theory (section (3)). Whereas the investigated algorithms can deal with general markov stocks and exotic options, we chose to restrict our attention to a d-dimensional constant rates Black–Scholes model and to the pricing of vanilla options. This restriction allowed us to provide more efficient implementations. The extension of the Black–Scholes and the vanilla settings to more general markets and options is often straightforward.

Second, we take up the description of the following algorithms : Longstaff-Schwartz (1998), Tsitsikliss-VanRoy (2000), the quantization algorithms (Bally, Pagès, Printem 2000), Broadie–Glassermann (1997) and Barraquand–Martineau (1995). The algorithms based on the parametrization or the recursive computation of the exercise boundary and based on the Malliavin calculus will be investigated in the next amclib version. For each algorithm, we provide

- a precise description of its mathematical aspects.
- a sketch of one of its possible implementations (sections "practical design"). Apart from the language details, this sketch is very close to the true C implementation.

Third, we give the global structure of the C implementation and we depict in details (types, meaning, relevant values) the pricing routines parameters (section (5)). Finally, we decribe the Scilab interface of the C routines : Scilab function parameters, build of the library, use and examples function (section (6)).

2 The market model.

We restrict our attention to a constant rate d–dimensional Black–Scholes model $S_t := \left(S_t^1, \ldots, S_t^d\right)$ defined for $t \in [0, T]$ on a probability space $(\Omega, \mathcal{A}, P)$. More precisely, let

- $(B_t)_{t\geq 0}$ be a d–dimensional brownian motion defined on $(\Omega, \mathcal{A}, P)$.
- $r > 0$ be the riskfree interest rate of the market, assumed to be deterministic and constant.
- $\delta_1, \ldots, \delta_d > 0$ be the dividend rates of the stocks $S_t^1, \ldots, S_t^d$, assumed to be deterministic and constant.
- $\mu_1, \ldots, \mu_d > 0$ be the mean rates of the stocks $S_t^1, \ldots, S_t^d$, assumed to be deterministic and constant.
- Σ be a symetric positive definite $d \times d$ matrix and let σ be its Cholesky square root.

Then, we assume that S_0^i is deterministic and we consider the stock S_t^i satisfying

$$dS_t^i = S_t^i \left(\mu_i \, dt + \sum_{1\leq j\leq i} \sigma_{i,j} dB_t^j \right)$$

or, equivalently,

$$S_t^i = S_0^i \exp\left(-t \left(\frac{1}{2} \sum_{1\leq j\leq i} \sigma_{i,j}^2 - r + \delta_i \right) + \sum_{1\leq j\leq i} \sigma_{i,j} W_t^j \right)$$

where $(W_t)_{t\in[0,T]}$ is a brownian motion under the unique risk–neutral probability measure $\mathbb{P}$ of the market.

3 Pricing of Bermudean style options.

Let $\mathcal{O}$ be a Vanilla option with maturity $T > 0$, payoff $(\varphi(S_t))_{0\leq t\leq T}$ (satisfying $\mathbb{E}\left(\varphi(S_t)^2\right) < +\infty$ for every t) and exercise times $t_0 = 0 < t_1 < \ldots < t_N = T$.

Recall that the discounted value of a self–financed portfolio is a martingale under the risk–neutral probability measure $\mathbb{P}$. So, the hedging value Q_0^τ at time 0 of the option related to an exercise strategy τ is

$$Q_0^\tau = \mathbb{E}\left(B(0,\tau)\,\varphi(S_\tau)\right)$$

Hence, the price Q_0 of $\mathcal{O}$ at time 0 is given by

$$Q_0 = \sup_{\tau\in\mathcal{T}_{0,N}} \mathbb{E}\left(B(0,\tau)\,\varphi(S_\tau)\right) \tag{1}$$

where $\mathcal{T}_{0,N}$ is the set of the $\mathcal{F}$–stopping times taking values in $\{t_0,\ldots,t_N\}$ and where $\mathcal{F}$ is the completion of the filtration generated be the process $(S_t)_{0\leq t\leq T}$.

Equivalently, the price Q_0 can be obtained as the terminal value of the following backward dynamical programming algorithm

$$\left\{\begin{array}{l} Q_N := \varphi(S_{t_N}) \\ Q_{j-1} := \max\left(\varphi\left(S_{t_{j-1}}\right), \mathbb{E}\left(B(t_{j-1},t_j)\,Q_j|\mathcal{F}_{t_{j-1}}\right)\right), \quad 1\leq j\leq N. \end{array}\right. \tag{2}$$

At last, one can show that the stopping time

$$\tau^* := \min\left\{t_j; \varphi\left(S_{t_j}\right) = Q_j\right\} \tag{3}$$

attains the supremum in (1), *i.e.*,

$$Q_0 = \mathbb{E}\left(B(0,\tau^*)\,\varphi(S_{\tau^*})\right) \tag{4}$$

and is the smallest stopping time checking (4).

Throughout the sequel, a price Q_0 computed with an algorithm based on the formulae (4) will be called a forward price whereas a price Q_0 computed with an algorithm based on the dynamical programming (2) will be called a backward price.

4 Pricing algorithms.

4.1 The Longstaff–Schwartz algorithm.

4.1.1 Basic principle.

The algorithm consists in approximating the stopping time τ^* along M Black–Scholes paths $\left(\omega^i\right)_{1\leq i\leq M}$ by a random variable τ^M and then estimating the forward price Q_0 according to the Monte–Carlo formulae

$$Q_0^M = \frac{1}{M}\sum_{1\leq i\leq M} B\left(0,\tau^M\left(\omega^i\right)\right)\varphi\left(S_{\tau^M(\omega^i)}\right) \tag{5}$$

We set out the construction of τ^M in the following steps : first, we give the dynamical programming algorithm which gives recursively the stopping time τ^*. Then, we present the Longstaff–Schwartz approximation of this recursion. Finaly, we give the corresponding empirical version, leading to τ^M.

4.1.2 Forward price.

Dynamical programming algorithm for τ^*. Let $\tau_N^* := T$ and for $0 \leq j \leq N-1$, let τ_j^* be the optimal stopping time as defined in (3) but for the Bermudean option with maturity T, initial information $\mathcal{F}_{t_j}$ and exercise times $t_j, \ldots, t_N = T$, *i.e.*,

$$\tau_j^* := \min\left\{t_i; i \geq j; \varphi\left(S_{t_i}\right) = Q_i\right\} \tag{6}$$

Then, $\tau_0^* = \tau^*$ and one can derive that the sequence $\left(\tau_j^*\right)_{0\leq j\leq N}$ checks

$$\left\{\begin{array}{l} \tau_N^* := T \\ \tau_j^* := t_j \mathbf{1}_{A_j} + \tau_{j+1}^* \mathbf{1}_{\complement A_j} \quad 0 \leq j \leq N-1. \end{array}\right. \tag{7}$$

where

$$A_j := \left\{\varphi\left(S_{t_j}\right) \geq \mathbb{E}\left(B\left(t_j, \tau_{j+1}^*\right)\varphi\left(S_{\tau_{j+1}^*}\right)|S_{t_j}\right)\right\}$$

Since $\left\{\varphi\left(S_{t_j}\right) = 0\right\} \subset \complement A_j$, it is useless to know $\mathbb{E}\left(B\left(t_j, \tau_{j+1}^*\right)\varphi\left(S_{\tau_{j+1}^*}\right)|S_{t_j}\right)$ on the set $\left\{\varphi\left(S_{t_j}\right) = 0\right\}$. So, the authors only consider the conditional expectation $\mathbb{E}_j\left(.|S_{t_j}\right)$ defined on the measure space $(\Omega_j, \mathbb{M}_j)$ where $\Omega_j := \left\{\varphi\left(S_{t_j}\right) > 0\right\}$ and $\mathbb{M}_j = \mathbb{P}\left(\Omega_j \cap .\right)$. The set A_j then reads

$$A_j = \left\{\varphi\left(S_{t_j}\right) > \overline{\mathbb{E}}_j\left(B\left(t_j, \tau_{j+1}^*\right)\varphi\left(S_{\tau_{j+1}^*}\right)|S_{t_j}\right)\right\} \in \mathcal{F}_{t_j}$$

where $\overline{\mathbb{E}}_j\left(.|S_{t_j}\right) = \mathbb{E}_j\left(.|S_{t_j}\right)$ on Ω_j and $= 0$ elsewhere.

So, the main step to estimate τ^* from (7) is to approximate

$$\mathbb{E}_j\left(B\left(t_j, \tau_{j+1}^*\right)\varphi\left(S_{\tau_{j+1}^*}\right)|S_{t_j}\right) \tag{8}$$

or equivalently, to approximate a function $\psi_j : S_{t_j}\left(\Omega_j\right) \to \mathbb{R}_+$ satisfying

$$\psi_j\left(S_{t_j}\right) = \mathbb{E}_j\left(B\left(t_j, \tau_{j+1}^*\right)\varphi\left(S_{\tau_{j+1}^*}\right)|S_{t_j}\right) \tag{9}$$

The authors introduce a least square regression method to perform this approximation.

A regression method. For every $1 \leq j \leq N-1$, let $\mathcal{L}\left(S_{t_j}\right)$ be the law of S_{t_j} and let $\mathcal{L}_j$ be $\mathcal{L}\left(S_{t_j}\right)$ restricted to $S_{t_j}\left(\Omega_j\right)$. Let $\left(g_i^j\right)_{i\geq 1}$ be a topological basis of $L^2\left(\mathbb{R}^d, \mathcal{B}_d, \mathcal{L}_j\right)$. We assume that $g^j \equiv 0$ outside $S_{t_j}\left(\Omega_j\right)$. For $j = 0$, the conditionnal expectation with respect to S_{t_0} reduces to an expectation so that we don't need a basis at time 0.

Fixing an index $k \geq 1$, one can approximate ψ_{N-1} by its orthogonal projection $\langle \alpha^{N-1}, g^{N-1}\rangle$ on the space spanned by $g^{N-1} := \left\{g_1^{N-1}, \ldots, g_k^{N-1}\right\}$, and compute the stopping time $\widehat{\tau}_{N-1} := t_{N-1}\mathbf{1}_{\widehat{A}_{N-1}} + T\mathbf{1}_{\complement\widehat{A}_{N-1}}$ where

$$\widehat{A}_{N-1} := \left\{\varphi\left(S_{t_{N-1}}\right) > \langle \alpha^{N-1}, g^{N-1}\rangle\left(S_{t_{N-1}}\right)\right\}$$

Iterating this procedure backward in time for $j = N-2$ downto 1, one obtains the approximated dynamical programming algorithm

$$\left\{\begin{array}{l} \widehat{\tau}_N := T \\ \widehat{\tau}_j := t_j \mathbf{1}_{\widehat{A}_j} + \widehat{\tau}_{j+1}\mathbf{1}_{\complement\widehat{A}_j} \quad 1 \leq j \leq N-1. \end{array}\right. \tag{10}$$

where

$$\widehat{A}_j := \left\{\varphi\left(S_{t_j}\right) > \langle \alpha^j, g^j\rangle\left(S_{t_j}\right)\right\} \quad 1 \leq j \leq N-1$$

and where $\alpha^j := \left\{\alpha_1^j \ldots, \alpha_k^j\right\} \in \mathbb{R}^k$ is the unique solution of the least square problem

$$\min_{\alpha\in\mathbb{R}^k} \mathbb{E}_j\left(\langle \alpha^j, g^j\rangle\left(S_{t_j}\right) - B\left(t_j, \tau_{j+1}^*\right)\varphi\left(S_{\tau_{j+1}^*}\right)\right)^2 \tag{11}$$

To make (10) implementable, one needs an empirical version of (11).

Implementable regression. Let $\left\{\omega^1, \ldots, \omega^M\right\}$ be an i.i.d. sample path from $\mathbb{P}$. Let (after reordering indexes) $\left\{\omega^1, \ldots, \omega^{M_j}\right\}$ be the points of this sample belonging to Ω_j. Then, the authors consider the empirical version of (11)

$$\min_{\alpha\in\mathbb{R}^k} \sum_{1\leq m\leq M_j} \left(\langle \alpha^j, g^j\rangle\left(S_{t_j}\left(\omega^m\right)\right) - B\left(t_j, \widehat{\tau}_{j+1}\left(\omega^m\right)\right)\varphi\left(S_{\widehat{\tau}_{j+1}(\omega^m)}\right)\right)^2 \tag{12}$$

and finally get the implementable algorithm

$$\begin{cases} \tau_N^M := T \\ \tau_j^M := t_j \mathbf{1}_{A_j^M} + \tau_{j+1}^M \mathbf{1}_{\complement A_j^M} \quad 1 \leq j \leq N-1. \end{cases} \tag{13}$$

where

$$A_j^M := \left\{\varphi\left(S_{t_j}\right) > \langle \alpha^j, g^j\rangle\left(S_{t_j}\right)\right\} \quad 1 \leq j \leq N-1$$

and where α^j is the unique solution of (12) (set $\alpha^j = 0$ if $M_j = 0$).

At time 0, the decision to exercise or not is deterministic (*i.e.*, one has either $\tau_0^M \equiv 0$ or $\tau_0^M \equiv \tau_1^M$. Then, the output of the algorithm is

$$Q_0^f = \max\left(\varphi\left(S_{t_0}\right), \frac{1}{M}\sum_{1\leq m\leq M} B\left(0, \tau_1^M\right)\varphi\left(S_{\tau_1^M}\right)\right).$$

4.1.3 Backward simulation.

Before to sum up the algorithm, we give an improvement of its effectiveness.

In their paper, the authors compute and store simultaneously all the Black–Scholes paths (*i.e.*, the vectors $\left(S_{t_j}\left(\omega^l\right)\right), \quad 0 \leq j \leq N \quad 1 \leq l \leq M$). This procedure can be very memory consumming since requiring $M \times N \times d \times$ (size of machine number) octets.

But, the algorithm (13) works backward in time and the regression at time t_j of the vector $\left(B\left(t_j, \widehat{\tau}_{j+1}\left(\omega^l\right)\right)\varphi\left(S_{\widehat{\tau}_{j+1}(\omega^l)}\right)\right)_{1\leq l\leq M_j}$ only requires the knowledge of the vector $\left(S_{t_j}\left(\omega^l\right)\right)_{1\leq l\leq M_j}$

So, we propose to use a backward simulation of the stock S_{t_j} : knowing that $B_0 = 0$ and $B_{t_{j+1}} = b$, the law of B_{t_j} is

$$\mathcal{N}\left(\frac{t_j}{t_{j+1}}b, \frac{t_j}{t_{j+1}}\left(t_{j+1} - t_j\right) I_d\right) \tag{14}$$

At time t_j, one can then compute $\left(B_{t_j}\left(\omega^l\right)\right)_{1\leq l\leq M}$ and $\left(S_{t_j}\left(\omega^l\right)\right)_{1\leq l\leq M}$ from $\left(B_{t_{j+1}}\left(\omega^l\right)\right)_{1\leq l\leq M}$ and forget simultaneously this last vector.

The memory consumption of the backward simulation procedure is about N times less than the Longstaff–Schwartz global simulation technique.

We now sum up the different stages of the algorithm.

4.1.4 Practical design.

Let

- $(B[m])_{1\leq l\leq M}$ be a memory vectors of size $M\times d$ for the storage of the Brownian motion samples.
- $(S[m])_{1\leq l\leq M}$ be a memory vectors of size $M\times d$ for the storage of the Black–Scholes stocks.
- $(P[m])_{1\leq l\leq M}$ be a memory vector of size M for the storage of the optimal stopped stocks.

The following steps make up our version of the Longstaff–Schwartz algorithm.

1. At time T and for every l, initialise $B[m]$ according to $\mathcal{N}(0, T\,I_d)$ and put the corresponding stock in $S[m]$. Then, put $B(t_{N-1}, T)\,\varphi(S[m])$ in $P[m]$.
2. For $j = N-1$ downto 1.
   (a) (Stock simulation) Compute $B[m]$ according to (14) with $b =$ old value of $B[m]$. Then compute $S[m]$.
   (b) (Regression) Keep the at–the–monney paths $\{S[m_1], \ldots, S[m_{M_j}]\}$ and compute the solution of (13), given by the formulae
   $$\alpha^j = D_j^{-1} \sum_{1\leq p\leq M_j} P[m_p]\,{}^t g^j\left(S[m_p]\right) \tag{15}$$
   where ^{t} is the transpose operator and where
   $$D_j := \sum_{1\leq p\leq M_j} g^j\left(S[m_p]\right)\,{}^t g^j\left(S[m_p]\right).$$
   is the (unnormalised) empirical dispersion matrix of the regressor. Use the Cholesky algorithm to compute D_j^{-1}.
   (c) (Dynamical programming) For $p = 1$ to M_j,
   $$P[m_p] := \varphi\left(S[m_p]\right) \text{ if } \varphi\left(S[m_p]\right) > \langle \alpha^j, g^j\rangle\left(S[m_p]\right)$$
   (d) (Discounting) For $m = 1$ to M, $P[m] := B(t_{j-1}, t_j)\,P[m]$.
3. (Forward Price) The output of the algorithm is
   $$\max\left(\varphi(S_0), \frac{1}{M}\sum_{1\leq m\leq M} P[m]\right).$$

4.1.5 Some other versions.

Normalised regressor. Some numerical problems may appear in the regression procedure when too larges values of the stock are send through the regression basis. To avoid this problem and to make the algorithm more insensitive to the Black–Scholes parameters values, it seems natural to perform the regression with respect to a normalised version of S_{t_j}.

For instance, one can deal at each time with the normalised stock

$$\widetilde{S}_{t_j} = \Sigma_j^{-1}\left(S_{t_j} - m_j\right)$$

where

$$m_j := \left[S_0^i \exp\left(-t_j\left(\delta_i - r\right)\right)\right]_{1\leq i\leq d}$$

and

$$\Sigma_j = \text{ Cholesky Square Root of } \left[m_j^p m_j^q\left(\exp\left(-t_j/2\sum_{1\leq l\leq d}\sigma_{pl}\sigma_{ql}\right)-1\right)\right]_{1\leq p,q\leq d}$$

are respectively the mean and the dispersion matrix of the Black–Scholes model at time t_j.

Hermite basis. In general, the basis $\left\{g_1^j\left(S_{t_j}\right),\ldots,g_k^j\left(S_{t_j}\right)\right\}$ is not orthogonal in $L^2\left(\Omega_j,\mathcal{F},\mathbb{M}_j\right)$. Hence, estimating the projection of the random variable $B\left(t_j,\widehat{\tau}_{j+1}\right)\varphi\left(S_{\widehat{\tau}_{j+1}}\right)$ requires the computation and the inversion of the (non–diagonal) regressor dispersion matrix. To avoid this numerical procedure, we suggest to perform the regression with respect to the brownian motion itself and to consider the following Hermite basis :

$$H_{n_1,\ldots,n_d}\left(x_1,\ldots,x_d\right) := \frac{H_{n_1}\left(x_1\right)\ldots H_{n_d}\left(x_d\right)}{\left(2^{n_1+\ldots+n_d}n_1!\ldots n_d!\right)^{1/2}} \quad n_1,\ldots,n_d\in\mathbb{N}$$

where H_i is the i$^{\text{th}}$ Hermite polynomial :

$$\left\{\begin{array}{l} H_0\left(x\right)=1 \\ H_1\left(x\right)=x \\ H_{i+1}\left(x\right)=xH_i\left(x\right)-iH_{i-1}\left(x\right). \end{array}\right.$$

Indeed, for every $1\leq j\leq N-1$, the family

$$\left\{H_{n_1,\ldots,n_d}\left(B_{t_j}/\left(2t_j\right)^{1/2}\right)\right\}_{n_1,\ldots,n_d\in\mathbb{N}} \tag{16}$$

is orthonormal in $L^2\left(\Omega,\mathcal{F},\mathbb{P}\right)$ and the computation of the regressor coefficient $\alpha^j_{n_1,\ldots,n_d}$ reduces to the scalar product

$$\alpha^j_{n_1,\ldots,n_d} = \mathbb{E}\left(H_{n_1,\ldots,n_d}\left(B_{t_j}/\left(2t_j\right)^{1/2}\right)B\left(t_j,\widehat{\tau}_{j+1}\right)\varphi\left(S_{\widehat{\tau}_{j+1}}\right)\right) \tag{17}$$

We point out that the expectation in formulae (17) is not $\mathbb{E}_j$ since the family (16) is not orthogonal on $L^2\left(\Omega_j,\mathcal{F},\mathbb{M}_j\right)$

This procedure is more simple and efficient than the use of formulae (15). In counterpart, the reduction complexity given by the restriction to the at–the–money paths measure $\mathbb{M}_j$ is lost.

4.2 The Tsitsiklis–VanRoy algorithm.

4.2.1 Basic principle.

This algorithm acts on a dynamical programming algorithm equivalent to (2) but involving $\widetilde{Q}_j := \mathbb{E}\left(B\left(t_j,t_{j+1}\right)Q_{j+1}|S_{t_j}\right)$ instead of Q_j. The approximation of $\widetilde{Q}_j$ is performed with a regression method.

4.2.2 Backward price.

Another form of the dynamical programming algorithm. Remember the notations of the algorithm (2) and let

$$\widetilde{Q}_j := \mathbb{E}\left(B\left(t_j, t_{j+1}\right) Q_{j+1} | S_{t_j}\right) \qquad 0 \leq j \leq N-1.$$

Then, (2) reads

$$\begin{cases} \widetilde{Q}_{N-1} := \mathbb{E}\left(B\left(t_{N-1}, t_N\right) \varphi\left(S_{t_N}\right) | S_{t_{N-1}}\right) \\ \widetilde{Q}_j := \mathbb{E}\left(B\left(t_j, t_{j+1}\right) \max\left(\varphi\left(S_{t_{j+1}}\right), \widetilde{Q}_{j+1}\right) | S_{t_j}\right), \quad 0 \leq j \leq N-2. \end{cases} \tag{18}$$

A regression method to approximate $\widetilde{Q}_j$. For every $1 \leq j \leq N-1$, let $\mathcal{L}_j$ be the law of S_{t_j}, let $\left\{g_i^j\right\}_{i \geq 1}$ be a topological basis of $L^2\left(\mathbb{R}^d, \mathcal{B}_d, \mathcal{L}_j\right)$ and let $\chi_j : \mathbb{R}^d \to \mathbb{R}_+$ such that $\widetilde{Q}_j = \chi_j\left(S_{t_j}\right)$.

At time $N-1$, the authors approximate χ_{N-1} by its projection $\langle \alpha^{N-1}, g^{N-1} \rangle$ on the subspace spanned by $\left\{g_1^{N-1}, \ldots, g_k^{N-1}\right\}$, according to the $L^2\left(\mathbb{R}^d, \mathcal{B}_d, \mathcal{L}_{N-1}\right)$ norm. They iterate this procedure backward in time, projecting at time t_j $(j \geq 1)$ the function

$$B\left(t_j, t_{j+1}\right) \max\left(\varphi, \langle \alpha^{j+1}, g^{j+1} \rangle\right)$$

on the subspace vect $\left\{g_1^j, \ldots, g_k^j\right\}$ according to the $L^2\left(\mathbb{R}^d, \mathcal{B}_d, \mathcal{L}_j\right)$ norm . The resulting approximated dynamical programming algorithm is then

$$\begin{cases} \alpha^{N-1} = D_{N-1}^{-1} \mathbb{E}\left(B\left(t_{N-1}, t_N\right) \varphi\left(S_{t_N}\right) {}^t g^{N-1}\left(\varphi\left(S_{t_{N-1}}\right)\right)\right) \\ \alpha^j = D_j^{-1} \mathbb{E}\left(B\left(t_j, t_{j+1}\right) \max\left(\varphi\left(S_{t_{j+1}}\right), \langle \alpha^{j+1}, g^{j+1} \rangle\left(S_{t_{j+1}}\right)\right) {}^t g^j\left(S_{t_j}\right)\right) \quad 1 \leq j \leq N-2. \\ \alpha = \mathbb{E}\left(B\left(0, t_1\right) \max\left(\varphi\left(S_{t_1}\right), \langle \alpha^1, g^1 \rangle\left(S_{t_1}\right)\right)\right) \end{cases} \tag{19}$$

where D_j is the dispersion matrix of the vector $g^j\left(S_{t_j}\right)$. At time 0, the needed backward price Q_0^b is given by

$$Q_0^b = \max\left(\varphi\left(S_{t_0}\right), \alpha\right)$$

Empirical version. Let $\omega^1, \ldots, \omega^M$ be an i.i.d. sample from $\mathbb{P}$ and let $S_{t_j}^m := S_{t_j}\left(\omega^m\right)$. The empirical version of (19) is

$$\begin{cases} \alpha_M^{N-1} = D_{N-1}^{-1} \sum_{1 \leq l \leq M} B\left(t_{N-1}, t_N\right) \varphi\left(S_{t_N}^m\right) {}^t g^{N-1}\left(\varphi\left(S_{t_{N-1}}^m\right)\right) \\ \alpha_M^j = D_j^{-1} \sum_{1 \leq l \leq M} B\left(t_j, t_{j+1}\right) \max\left(\varphi\left(S_{t_{j+1}}^m\right), \langle \alpha_M^{j+1}, g^{j+1} \rangle\left(S_{t_{j+1}}^m\right)\right) {}^t g^j\left(S_{t_j}^m\right) \quad 1 \leq j \leq N-2. \\ \alpha_M = \sum_{1 \leq l \leq M} B\left(0, t_1\right) \max\left(\varphi\left(S_{t_1}^m\right), \langle \alpha_M^1, g^1 \rangle\left(S_{t_1}^m\right)\right) \end{cases} \tag{20}$$

where D_j denotes the empirical dispersion matrix

$$D_j := \sum_{1 \leq l \leq M} g^j\left(S_{t_j}^m\right) {}^t g^j\left(S_{t_j}^m\right)$$

The price backward Q_0^b is then

$$Q_0^b = \max\left(\varphi\left(S_{t_0}\right), \alpha_M\right)$$

4.2.3 Forward price.

Here, we give a forward price formulae for the Tsitsiklis–VanRoy algorithm. Such a formulae is not derived in the author's paper.

Recall that an optimal stopping time for (2) is given by

$$\tau^* = \min\left\{t_i \geq 0; \varphi\left(S_{t_i}\right) = Q_i\right\}$$

and, for $1 \leq j \leq N-1$, let $\tau_j^* = \min\left\{t_i; i \geq j; \varphi\left(S_{t_i}\right) = Q_i\right\}$. The sequence $\left(\tau_j^*\right)$ checks

$$\left\{\begin{array}{l} \tau_N^* := T \\ \tau_j^* := t_j \mathbf{1}_{B_j} + \tau_{j+1}^* \mathbf{1}_{\complement B_j} \quad 1 \leq j \leq N-1. \end{array}\right. \tag{21}$$

where $B_j := \left\{\varphi\left(S_{t_j}\right) \geq \widetilde{Q}_j\right\}$. This algorithm can be approximated by

$$\left\{\begin{array}{l} \widehat{\tau}_N := T \\ \widehat{\tau}_j := t_j \mathbf{1}_{\widehat{B}_j} + \widehat{\tau}_{j+1} \mathbf{1}_{\complement \widehat{B}_j} \quad 1 \leq j \leq N-1. \end{array}\right. \tag{22}$$

where $\widehat{B}_j := \left\{\varphi\left(S_{t_j}\right) \geq \langle \alpha^j, g^j \rangle\right\}$ and where α^j is defined in (19). The practical computation of (22) is performed according to

$$\left\{\begin{array}{l} \tau_N^M := T \\ \tau_j^M := t_j \mathbf{1}_{B_j^M} + \tau_{j+1}^M \mathbf{1}_{\complement B_j^M} \quad 1 \leq j \leq N-1. \end{array}\right. \tag{23}$$

where $B_j := \left\{\varphi\left(S_{t_j}\right) \geq \langle \alpha_M^j, g^j \rangle\right\}$ and where α_M^j is defined in (20).

One finaly gets the needed forward price

$$Q_0^f = \max\left(\varphi\left(S_{t_0}\right), \frac{1}{M} \sum_{1 \leq m \leq M} B\left(0, \tau_1^M\left(\omega^m\right)\right) \varphi\left(S^m_{\tau_1^M(\omega^m)}\right)\right)$$

4.2.4 Practical design.

Let

- $(B[m])_{1 \leq l \leq M}$ be a memory vectors of size $M \times d$ for the storage of the Brownian motion samples.
- $(S[m])_{1 \leq l \leq M}$ be a memory vectors of size $M \times d$ for the storage of the Black–Scholes stocks.
- $(Q[m])_{1 \leq l \leq M}$ be a memory vector of size M for the storage of the dynamical programming prices.
- $(P[m])_{1 \leq l \leq M}$ be a memory vector of size M for the storage of the optimal stopped stocks.

The following steps make up our version of the Tsitsiklis–VanRoy algorithm.

1. At time T and for every l, initialise $B[m]$ according to $\mathcal{N}\left(0, T\, I_d\right)$ and put the corresponding stock in $S[m]$. Then, put $B\left(t_{N-1}, T\right) \varphi\left(S[m]\right)$ in $P[m]$ and $Q[m]$.

2. For $j = N-2$ downto 1.

   (a) (Stock simulation) Compute $B[m]$ according to (14) with $b =$ old value of $B[m]$. Then compute $S[m]$.

   (b) (Regression) Compute the regression coefficient α_M^j, given by the formulae

$$\alpha_M^j = D_j^{-1} \sum_{1 \le l \le M} Q[m]\, {}^t g^j\left(S[m]\right) \tag{24}$$

   where ^{t} is the transpose operator and where

$$D_j := \sum_{1 \le p \le M_j} g^j\left(S[m_p]\right)\, {}^t g^j\left(S[m_p]\right).$$

   is the (unnormalised) empirical dispersion matrix of the regressor. Use the Cholesky algorithm to compute D_j^{-1}.

   (c) (Dynamical programming)

$$Q[m] = \max\left(\varphi\left(S[m]\right), \langle \alpha_M^j, g^j\rangle\left(S[m]\right)\right).$$

   If $\varphi\left(S[m]\right) \ge \langle \alpha_M^j, g^j\rangle\left(S[m]\right)$, then $P[m] = \varphi\left(S[m]\right)$.

   (d) (Discounting) $Q[m] = B\left(t_{j-1}, t_j\right) Q[m]$ and $P[m] = B\left(t_{j-1}, t_j\right) P[m]$

3. (Bakward price)

$$Q_0^b = \max\left(\varphi\left(S_{t_0}\right), \frac{1}{M} \sum_{1 \le l \le M} Q[m]\right)$$

4. (Forward Price)

$$Q_0^f = \max\left(\varphi\left(S_{t_0}\right), \frac{1}{M} \sum_{1 \le l \le M} P[m]\right)$$

4.2.5 Some other versions

As in the Longstaff–Schwartz algorithm, one can use a normalised regressor or an Hermite basis for the Tsitsiklis–VanRoy algorithm.

4.3 The Quantization algorithms.

The quantization algorithm has been introduced by Bally and Pagès. The material presented in this section is taken from [**?**].

4.3.1 Basic principle.

The basic principle of the quantization method is, at time t_j, to discretize the stock S_{t_j} or more generally an underlying Markov process U_{t_j} by a set of "quantization levels" $y^j := \left(y_i^j\right)_{1 \le i \le n_j} \subset \mathbb{R}^d$ and then to approximate the transition kernel of $\left(U_{t_j}, U_{t_{j+1}}\right)$ by a discrete one, defined on $y^j \times y^{j+1}$.

More precisely, let $(U_t)_{0\leq t\leq T}$ be a process defined on $(\Omega, \mathcal{A}, \mathbb{P})$ such that for every t, $S_t = \psi(U_t)$ where $\psi : \mathbb{R}^d \to \mathbb{R}^d$ is injective. Let $\widehat{P}_j(x, dy)$ be the transition kernel between times t_j and t_{j+1} of the Markov chain $U_0, U_{t_1}, \ldots, U_{t_N}$.

For $1 \leq j \leq N$, let y^j be a set of discretization vectors $y^j := \left\{y_1^j, \ldots, y_{n_j}^j\right\} \subset \mathbb{R}^d$, and let $y^0 = \{y\}$ (U_0 is deterministic and then need not to be discretized).

4.3.2 Dynamical programming algorithm.

Remember the notations of (2), let $\chi_j : \mathbb{R}^d \to \mathbb{R}_+$ such that $Q_j = \chi_j\left(U_{t_j}\right)$ and let $\phi = \varphi \circ \psi$. The dynamical programming (2) reads

$$\begin{cases} \chi_N = \phi \\ \chi_j = \max\left(\phi, P_j \chi_{j+1}\right) \quad 0 \leq j \leq N-1. \end{cases} \tag{25}$$

As already said, we will approximate P_j by a discrete transition kernel $\widehat{P}_j$ and then approximate (25) by

$$\begin{cases} \widehat{\chi}_N = \phi \\ \widehat{\chi}_j = \max\left(\phi, \widehat{P}_j \widehat{\chi}_{j+1}\right) \quad 0 \leq j \leq N-1. \end{cases} \tag{26}$$

4.3.3 Quantized kernel.

We want to define $\widehat{P}_j$ as a transition kernel defined on $y^j \times y^{j+1}$ satisfying the following requirement : the number $\widehat{P}_j\left(y_k^j, y_l^{j+1}\right)$ must approximate the probability that the process U move from a neighbourhood of y_k^j at time t_j, to a neighbourhood of y_l^{j+1} at time t_{j+1}. So, we propose to pick

$$\widehat{P}_j\left(y_k^j, y_l^{j+1}\right) := \frac{\mathbb{E}\left(\mathbf{1}_{C_k(y^j)}\left(U_{t_j}\right) P_j\left(U_{t_j}, C_l\left(y^{j+1}\right)\right)\right)}{\mathbb{E}\left(\mathbf{1}_{C_k(y^j)}\left(U_{t_j}\right)\right)} \tag{27}$$

where the set $C_i\left(y^j\right)$ is the i^{th} Voronoï cell of y^j, defined by

$$C_i\left(y^j\right) := \left\{z \in \mathbb{R}^d \quad ; \quad \left\|z - y_i^j\right\| = \min_{1\leq l\leq n_j} \left\|z - y_l^j\right\|\right\}. \tag{28}$$

There would be some other way to specify $\widehat{P}_j\left(y_k^j, y_l^j\right)$ but the formulae (27) will be easily implementable through a MonteCarlo method.

Note that for every $x \in \mathbb{R}^d$, one has $P_j\left(x, \cup_{1\leq l\leq n_{j+1}} \partial C_l\left(y^{j+1}\right)\right) = 0$. Then, the formulae (27) defines a probability transition kernel.

4.3.4 Backward price.

For notational convenience, set

$$\alpha_{k,l}^j := \widehat{P}_j\left(y_k^j, y_l^{j+1}\right).$$

Knowing at time t_j the numbers $\alpha_{k,l}^j$, one can compute the function $\widehat{P}_j\widehat{\chi}_{j+1}$ on the set y^j. Hence, the dynamical programming algorithm (26) reduces to

$$\begin{cases} \widehat{\chi}_N\left(y_k^N\right) = \phi\left(y_k^N\right) & 1 \le k \le n_N \\ \widehat{\chi}_j\left(y_k^j\right) = \max\left(\phi\left(y_k^j\right), \sum\limits_{1\le l\le n_{j+1}} \alpha_{k,l}^j \widehat{\chi}_{j+1}\left(y_l^{j+1}\right)\right) & 0 \le j \le N-1,\ 1 \le k \le n_j. \end{cases} \tag{29}$$

and the needed backward price is $Q_0^b = \widehat{\chi}_0(y)$.

4.3.5 Empirical version for Q_0^b.

Let $\omega^1, \ldots, \omega^M$ be an i.i.d. sample path from $\mathbb{P}$ and let $U_{t_j}^m := U_{t_j}(\omega^m)$. To perform (29), one has to estimate the numbers $\alpha_{k,l}^j$. The formulae (27) suggests us to consider the MonteCarlo estimator

$$\widehat{\alpha}_{k,l}^j := \frac{\sum_{1\le m\le M} \mathbf{1}_{C_k(y^j)}\left(U_{t_j}^m\right)\mathbf{1}_{C_l(y^{j+1})}\left(U_{t_{j+1}}^m\right)}{\sum_{1\le m\le M} \mathbf{1}_{C_k(y^j)}\left(U_{t_j}^m\right)} \tag{30}$$

where the dependence on M of $\widehat{\alpha}_{k,l}^j$ has been omited. The implementable version of (29) is finaly obtained by replacing $\alpha_{k,l}^j$ by $\widehat{\alpha}_{k,l}^j$ in (29).

4.3.6 Forward price.

Once computed the numbers $\widehat{\chi}_j\left(y_k^j\right)$, we propose to approximate τ^* by the stopping time

$$\widehat{\tau} := \min\left\{t_j; \phi\left(\mathcal{Q}_j U_{t_j}\right) = \widehat{\chi}\left(\mathcal{Q}_j U_{t_j}\right)\right\} \tag{31}$$

where $\mathcal{Q}_j : \mathbb{R}^d \to y^j$ is the quantizing function

$$\mathcal{Q}_j := \sum_{1\le k\le n_j} \mathbf{1}_{C_k(y^j)}\, y_l^j.$$

Informally, $\widehat{\tau}$ may be viewed as an approximation of τ^* along the quantized stock paths.

The corresponding forward price is then

$$Q_0^f = \mathbb{E}\left(B\left(0,\widehat{\tau}\right)\phi\left(\mathcal{Q}_{\widehat{\tau}}U_{\widehat{\tau}}\right)\right). \tag{32}$$

4.3.7 Empirical version for Q_0^f.

The empirical version of (32) is given by the MonteCarlo formulae

$$Q_0^f = \frac{1}{M}\sum_{1\le m\le M} B\left(0,\widehat{\tau}\left(\omega^m\right)\right)\phi\left(\mathcal{Q}_{\widehat{\tau}(\omega^m)}U_{\widehat{\tau}(\omega^m)}^m\right)$$

where $\omega^1, \ldots, \omega^M$ is an i.i.d. sample path from $\mathbb{P}$.

Clearly, there are many ways to implement the quantization method, depending on the choice of the quantization sets y^j. We present three of them in the following sections.

4.3.8 The Random Quantization algorithm.

Quantization sets. We assume that $n_j = n$ for every j and that the process U is the Brownian motion. We use a self–quantization procedure to generate the quantization sets. More precisely, let $\omega^1, \ldots, \omega^n$ be an i.i.d. sample from $\mathbb{P}$. Then , we set

$$y_k^j := U_{t_j}\left(\omega^k\right) \quad 1 \leq j \leq n \quad 1 \leq k \leq n.$$

One can show that this choice is in some sense somewhat good but is not the best among all the random choices (see [5]).

Now, we can give an implementation of the random quantization method.

Practical design. Recall that $n_j = n$ for every $1 \leq j \leq N$. Let

- $(Y[i][j])_{1\leq i\leq n;\, 1\leq j\leq N}$ be a memory array of size $(n \times d) \times N$ for the storage of the quantization sets.
- $(S[i][j])_{1\leq i\leq n;\, 1\leq j\leq N}$ be a memory array of size $(n \times d) \times N$ for the storage of the Black–Scholes transformation of the quantization sets.
- $(B[m])_{1\leq m\leq M}$ a memory vector of size $M \times d$ for the storage of the Brownian motion paths at a given time.
- $(C[m])_{1\leq m\leq M}$ be an integer memory vector of size M for the storage of the cell numbers of B.
- $(Q[i][j])_{1\leq i\leq n;\, 1\leq j\leq N}$ be a memory array of size $n \times N$ for the storage of the dynamical programming prices.
- $(\Lambda[k][l])_{1\leq k\leq n;\, 1\leq l\leq n}$ be a memory array of size $n \times n$ for the storage of the transition probabilities at a given time.
- $(W[k])_{1\leq k\leq n}$ be a memory vector of size n for the storage, at a given time j, of the numbers $|\{1 \leq m \leq M; B[m] \in C_k\left(Y[][j]\right)\}|$.

The following steps make up the needed implementation.

1. (Quantization sets) For $1 \leq k \leq n$: initialise $Y[k][1], \ldots, Y[k][N]$ with a Brownian path $B_{t_1}\left(\omega^k\right), \ldots, B_{t_N}\left(\omega^k\right)$; compute the corresponding Black–Scholes stock path $S[k][1], \ldots, S[k][N]$.
2. (Brownian bridge initialisation) For $1 \leq m \leq M$: initialize $B[m]$ according to $\mathcal{N}\left(0, T\, I_d\right)$. Then,

   $$C[m] := \text{NearestIndex}(B[m])$$

   where the function "NearestIndex" gives the index k_0 of the Voronoï cell $C_{k_0}\left(Y[.][N]\right)$ in which $B[m]$ belongs.
3. (prices initialisation) For $1 \leq k \leq n$, $Q[k][N] = \varphi\left(S[k][N]\right)$.
4. For $j = N - 1$ downto 1 :
   (a) (Brownian bridge simulation) For $1 \leq m \leq M$, Compute $B[m]$ according to (14) with $b =$ old value of $B[m]$.

(b) (Transitions computation) For $1 \leq m \leq M$:

i. $AuxC :=$ NearestIndex$(B[m])$,

ii. $\Lambda[AuxN][C[m]]+ = 1$,

iii. $W[AuxN]+ = 1$,

iv. $C[m] = AuxC$.

(c) (Transitions normalisation) For $1 \leq k \leq n$: if $C[k] > 0$, for $1 \leq l \leq n$ $\Lambda[k][l] = \Lambda[k][l]/W[k]$.

(d) (Dynamical programming) For $1 \leq k \leq n$: compute

$$\mathcal{S} := \sum_{1 \leq l \leq n} Q[l][j+1]\Lambda[k][l]$$

and set

$$Q[k][j] := \max\left(\varphi\left(S[k][j]\right), B(t_j, t_{j+1})\mathcal{S}\right).$$

5. (Backward price) The backward price is

$$Q_0^b := \max\left(\varphi\left(Spot\right), B(0,t_1) \sum_{1 \leq l \leq n} Q[l][1]W[l]/M\right).$$

6. (Forward price)

(a) If $Q_0^b = \varphi\left(Spot\right)$ then $Q_0^f := Q_0^b$ else :

(b) Let P be a memory vector of size d. Let $\xi := 0$. For $1 \leq m \leq M$:

i. Set $P = 0$, $j = 0$

ii. $\begin{cases} \text{Do} \\ \quad P := P + \sqrt{t_{j+1} - t_j}\, \mathcal{N}\left(0, I_d\right), j = j+1 \\ \text{while } Q[j][\text{NearestIndex}(P)] > \varphi\left(S[\text{NearestIndex}(P)][j]\right). \end{cases}$

iii. Compute $\xi = \xi + B\left(0, t_j\right)\varphi\left(S[\text{NearestIndex}(P)][j]\right)$.

(c) (Output) The needed forward price is

$$Q_0^f := \xi/M.$$

4.3.9 Optimal quantization and random weights.

Quantization sets. As in the preceding section, we set $n_j = n$ for $1 \leq j \leq n$ and we assume that the process U is the Brownian motion. One can show (see [**?**]) that an error bound for the backward price Q_0^b may be derived in term of the distortion

$$\mathcal{D}_{t_j}\left(y^j\right) := \int_{\mathbb{R}^d} \min_{1 \leq k \leq n} \left\|u - y_k^j\right\|^2 \mathcal{N}\left(0, t_j\, I_d\right)(du).$$

The modulus $\mathcal{D}_{t_j}$ is in some sense a measurement of the quality of y^j. So, it seems useful to choose a quantization set inducing a low distortion.

In fact, one can show that there exists an optimal set y^{*j} checking

$$\mathcal{D}_{t_j}\left(y^{*j}\right) = \min_{y \in (\mathbb{R}^d)^n} \mathcal{D}_{t_j}\left(y\right)$$

From a scaling invariance property, one can choose $y^{*j} = \sqrt{t_j}\, y^*$ where y^* is an optimal quantization set of the distortion $\mathcal{D} := \mathcal{D}_1$. So, we set

$$y_k^j := \sqrt{t_j}\, y_k^* \quad 1 \leq k \leq n;\, 1 \leq j \leq N. \tag{33}$$

In the following section, we briefly recall an algorithm to numerically compute y^*.

Approximation of y^*. Here, we chose the Lloyd I algorithm to perform the optimization of $\mathcal{D}$.

Let $Y \sim \mathcal{N}(0, I_d)$ and let $y^{(0)} \in \left(\mathbb{R}^d\right)^n$ with no component agregates. Then, using a MonteCarlo method, compute the sequence $y^{(n)}$ defined by

$$y_k^{(n+1)} := \mathbb{E}\left(Y | Y \in C_k\left(y^{(n)}\right)\right).$$

One can show that $\mathcal{D}\left(y^{(n)}\right)$ is decreasing and that the sequence $y^{(n)}$ converges toward a local minimum $y^{(\infty)}$ of $\mathcal{D}$. In the $\mathcal{N}(0, I_d)$ setting, the local minimum $y^{(\infty)}$ is in practice a global minimum.

Once computed, the set y^* is stored on a file, to be re–used through (33) without new computation.

For more details on the Lloyd I algorithm and its convergence, see [9]. There is another classical optimization procedure of $\mathcal{D}$: the Kohonen algorithm; See[4].

Faster nearest index. The computation of the nearest quantization point of a sample $z \in \mathbb{R}^d$ with respect to the unstructured quantization set y^* is usualy performed as follow : For $k = 1$ to n,

$$\left\{\begin{array}{l} n = 0;\, i = 0. \\ \text{ do } aux = aux + \left(z[i] - y_k^*[i]\right)^2;\, i = i + 1 \text{ while } aux < min \\ \text{if } aux < min \text{ then } output = k. \end{array}\right. \tag{34}$$

Since the quantization set y^* is computed once for all, one can compute and store some additional information on y^*. This may be useful to make the nearest index search faster :

Once computed y^*, use a MonteCarlo method to estimate the radius $r_1, \ldots, r_n$ of the Voronoï cells $C_1(y^*), \ldots, C_n(y^*)$, defined by

$$r_k := \max\left\{d\left(z, y_k^*\right);\, z \in C_k\left(y^*\right)\right\},$$

and store these numbers with y^*.

Let $\delta_1, \ldots, \delta_d \in \mathbb{R}^d$. Then, by the triangle inequality,

$$z \in C_k\left(y^*\right) \Longrightarrow \bigwedge_{1 \leq i \leq d} d\left(\delta_i, z\right) - r_i \leq d\left(y_k^*, z\right) \leq d\left(\delta_i, z\right) + r_i.$$

So, if n is large and if $M \gg n$ searches has to be done, the algorithm (34) can be improved in the following way :

1. Before to start the searches, compute and store the $2 \times n \times d$ numbers $\beta_{i,k} := d\left(\delta_i, y_k^*\right)^2$.
2. To search the nearest index of a sample z :

(a) Compute and store the $2 \times d$ numbers $\lambda_i^{\pm} := \left(d\left(\delta_i, z\right) \pm r_i\right)^2$.

(b) for $k = 1$ to n, run the loop (34) only if the tests

$$\bigwedge_{1 \leq i \leq d} \left(\lambda_i^+ \geq \beta_{i,k}\right) \wedge \left(\lambda_i^- \leq \beta_{i,k}\right) \tag{35}$$

are check.

If n is large and if $M \gg n$, the preceding procedure is faster than (34). Indeed, the quantization points that checks the two first tests in (35) must belong to a neighbourhood of a sphere. So, one replace the loop (34) by only two tests for a large subset of indexes. Moreover, since d "landmarks" are used, the subset of the indexes that check all the tests in (35) is very small. The loop is then runned only a few times.

The preceding landmarks algorithm would not be used when n is not sufficiently large regarding d. In that case, the radius r_i are too large with respect to the diameter of the tesselation y^*. Consequently, the landmarks procedure become inefficient and even longer than (34).

Practical design. There is only two differences between the design of the optimal quantization algorithm and the random quantization algorithm's one.

First, replace the step 1. by the procedure (33). Second, to avoid the storage of $d \times N$ landmarks, perform the nearest index search straight on the set y^* with the normalised sample $B[m]/\sqrt{t_j}$.

The other steps are similar.

4.3.10 Payoff vectorization.

A large part of the computations done in the quantization algorithm does not depend on the payoff function and on the Black–Scholes parameters (*e.g.* quantized kernel of the brownian motion and brownian path quantization for the forward price). Hence, one can "vectorize" the payoff parameter by computing simultaneously numerous option prices, the computations needed for each additionnal payoff beeing simply the dynamical programming.

4.4 The Broadie–Glassermann algorithm.

4.4.1 Basic principle.

The Broadie–Glassermann algorithm is a stock discretization method. As in the quantization setting, a "mesh" $Y^j = \left\{Y_1^j, \ldots, Y_n^j\right\} \subset \left(\mathbb{R}^d\right)^n$ is generated at time t_j $(j > 0)$ and the dynamical programming algorithm is approximated on $Y^1, \ldots, Y^N$ by mean of some weights $\beta_{k,l}^j$ between Y_k^j and Y_l^{j+1}. The difference with the quantization method is that the $\beta_{k,l}^j$s are not computed according to the stock transition kernel from time t_j to t_{j+1} but according to an importance sampling principle.

4.4.2 Mesh generation.

Let $f_j(x,u)$ be the kernel function of the Black–Scholes stock between times t_j and t_{j+1},

$$f_j(x,u) := \left(\frac{1}{2\pi(t_{j+1}-t_j)}\right)^{d/2} \left|\sigma^{-1}\right| \frac{1}{\prod_{1\le i\le d} u_i} \exp\left(-\frac{\left\|\sigma^{-1}\Theta(x,u)\right\|^2}{2(t_{j+1}-t_j)}\right)$$

where

$$\Theta(x,u) := \left[\log(u_i/x_i) + (t_{j+1}-t_j)\left(\frac{1}{2}\sum_{1\le l\le d}\sigma_{i,l}^2 - r + \delta_i\right)\right]_{1\le i\le d}.$$

Assume that all the samples $Y_1^j,\dots,Y_n^j$ considered in the sequel are defined on some probability space $\left(\widetilde{\Omega},\widetilde{\mathcal{A}},\widetilde{\mathbb{P}}\right)$.

Let $Y_1^1,\dots,Y_n^1$ be an i.i.d. sample from the law $f_0(S_{t_0},u)\,du$. The set $\left\{Y_1^1,\dots,Y_n^1\right\}$ will be used to discretize S_{t_1}.

Next, consider an i.i.d. sample $Y_1^2,\dots,Y_n^2$ from the law

$$\sum_{1\le l\le n} f_1\left(Y_l^1,u\right)\,du \tag{36}$$

Iterating this procedure forward in time, one obtains N "meshes" $\left(Y_k^j\right)_{1\le k\le n}$, $1\le j\le N$ such that $Y_1^1,\dots,Y_n^1$ is an i.i.d. sample from the law $f_0(S_{t_0},u)\,du$ and for $2\le j\le N$, $Y_1^j,\dots,Y_n^j$ is an i.i.d. sample from the law $\sum_{1\le l\le n} f_{j-1}\left(Y_l^{j-1},u\right)\,du$.

4.4.3 Weights.

One has to approximate the dynamical programming algorithm

$$\begin{cases} \chi_N = \varphi, \\ \chi_j\left(S_{t_j}\right) = \max\left(\varphi\left(S_{t_j}\right), \mathbb{E}\left(\chi_{j+1}\left(S_{t_{j+1}}\right)|S_{t_j}\right)\right) \quad 0\le j\le N-1. \end{cases} \tag{37}$$

on the meshes $\left(Y_k^j\right)_{1\le k\le n}$. For $j>0$, the authors propose to approximate the conditional expectations

$$\mathbb{E}\left(\chi_{j+1}\left(S_{t_{j+1}}\right)|S_{t_j}=Y_k^j\right)$$

by the sum

$$\frac{1}{n}\sum_{1\le l\le n}\beta_{k,l}^j\,\chi_{j+1}\left(Y_l^{j+1}\right). \tag{38}$$

They choose the weights β_{kl}^j by considering (38) as a MonteCarlo sum. Indeed, one has

$$\mathbb{E}\left(\chi_{j+1}\left(S_{t_{j+1}}\right)|S_{t_j}=Y_k^j\right) \quad = \quad \int_{\mathbb{R}^d}\chi_{j+1}(u)\,f_j\left(Y_k^j,u\right)\,du$$

$$= \int_{\mathbb{R}^d} \chi_{j+1}(u) \frac{f_j\left(Y_k^j, u\right)}{\sum_{1\le l\le n} f_j\left(Y_l^j, u\right)} \sum_{1\le l\le n} f_j\left(Y_l^j, u\right) du$$

$$= \widetilde{\mathbb{E}}\left(\chi_{j+1}(Y) \frac{f_j\left(Y_k^j, Y\right)}{\sum_{1\le l\le n} f_j\left(Y_l^j, Y\right)}\right)$$

where $Y : \left(\widetilde{\Omega}, \widetilde{\mathcal{A}}, \widetilde{\mathbb{P}}\right) \to \mathbb{R}^d$ has the density $\sum_{1\le l\le n} f_j\left(Y_l^j, u\right)$. But, the mesh $\left(Y_l^{j+1}\right)_{1\le l\le n}$ is an i.i.d. sample from the law $\sum_{1\le l\le n} f_j\left(Y_l^j, u\right) du$. So one can approximate (39) by the MonteCarlo estimator

$$\frac{1}{n} \sum_{1\le l\le n} \chi_{j+1}\left(Y_l^{j+1}\right) \frac{f_j\left(Y_k^j, Y_l^{j+1}\right)}{\sum_{1\le m\le n} f_j\left(Y_m^j, Y_l^{j+1}\right)} \tag{39}$$

Identifying (39) and (38), one obtains

$$\beta_{k,l}^j := \frac{f_j\left(Y_k^j, Y_l^{j+1}\right)}{\sum_{1\le m\le n} f_j\left(Y_m^j, Y_l^{j+1}\right)}. \tag{40}$$

Since $Y_1^1, \ldots, Y_n^1$ is drawn from $f_0(S_{t_0}, u)\, du$, the weights between S_{t_0} and Y^1 are equal to 1.

4.4.4 Backward price.

Once computed the weights $\beta_{k,l}^j$ according to the formulae (40), one obtains the approximated dynamical programming algorithm

$$\begin{cases} \chi_N = \varphi, \\ \chi_j\left(Y_k^j\right) = \max\left(\varphi\left(Y_k^j\right), \sum_{1\le l\le n} \beta_{k,l}^j \chi_{j+1}\left(Y_l^{j+1}\right)\right) \end{cases} \quad 1 \le k \le n,\ 1 \le j \le N.$$

The needed backward price is finaly

$$Q_0^b := \max\left(\varphi(S_{t_0}), \sum_{1\le l\le n} \beta_{0,l}^0 \chi_1\left(Y_l^1\right)\right).$$

4.4.5 Forward price.

Using arguments of the section (4.4.3), one can approximate the conditionnal expectation $\mathbb{E}\left(\chi_{j+1}\left(S_{t_{j+1}}\right) | S_{t_j} = x\right)$ by the sum

$$\widehat{Q}_j(x) := \sum_{1\le l\le n} \beta_l^j(x) \chi_{j+1}\left(Y_l^{j+1}\right)$$

where

$$\beta_l^j(x) := \frac{f\left(x, Y_l^{j+1}\right)}{\sum_{1\le m\le n} f_j\left(Y_m^j, Y_l^{j+1}\right)}$$

So, one can approximate pathwise the optimal stopping time τ^* by

$$\widehat{\tau} := \min\left\{t_j; \varphi\left(S_{t_j}\right) \geq \widehat{Q}_j\left(S_{t_j}\right)\right\}$$

The related forward price is as usual

$$Q_0^f := \mathbb{E}\left(B\left(0,\widehat{\tau}\right)\varphi\left(S_{\widehat{\tau}}\right)\right).$$

4.4.6 Practical design.

Let

1. $(Y[k][j])_{1\leq k\leq n,\, 1\leq j\leq N}$ be a memory array of size $n \times N$ for the storage of the mesh.
2. $(Q[k][j])_{1\leq k\leq n,\, 1\leq j\leq N}$ be a memory array of size $n \times N$ for the storage of the prices over the mesh.
3. $(W[k][j])_{1\leq k\leq n,\, 1\leq j\leq (N-1)}$ be a memory array of size $n \times N-1$ for the storage of some partial computation of the weights.

The following steps make up our implementation of the Broadie–Glassermann algorithm :

1. (Mesh initialization)
   (a) For $k = 1$ to n, put in $Y[k][1]$ a draw from the law $f_0\left(S_{t_1}, u\right) du$.
   (b) For $j = 2$ to N, for $k = 1$ to n :
      i. put in U a draw from the uniform law on $\{1, \ldots, n\}$.
      ii. then, put in $Y[k][j]$ a draw from the law $f_{j-1}\left(Y_U^{j-1}, u\right) du$.
2. (Partial computation of the weights) For $j = 1$ to $N-1$, for $k = 1$ to n,
$$W[k][j] = 1/\sum_{1\leq m\leq n} f_j\left(Y[m][j], Y[k][j+1]\right).$$
3. (Prices initialisation) For $k = 1$ to n, $Q[k][N] = \varphi\left(Y[k][N]\right)$.
4. (Dynamical programming) For $j = N-1$ downto 1, for $k = 1$ to n, compute
$$\mathcal{S} := \frac{1}{n}\sum_{1\leq l\leq n} Q[l][j+1] f_j\left(Y_k^j, Y_l^{j+1}\right) W[l][j]$$
and set
$$Q[k][j] := \max\left(\varphi\left(Y[k][j]\right), B(t_j, t_{j+1})\mathcal{S}\right).$$
5. (Backward price) The backward price is
$$Q_0^b := \max\left(\varphi\left(Spot\right), B\left(0,t_1\right)\frac{1}{n}\sum_{1\leq l\leq n} Q[l][1] f_0\left(S_{t_0}, Y[l][1]\right)\right).$$
6. (Forward price)

(a) If $Q_0^b = \varphi(Spot)$ then $Q_0^f := Q_0^b$ else :

(b) Let S be a memory vector of size d. Let $\xi = 0$. For $1 \leq m \leq M$:

i. Set $S = S_{t_0}$ and $j = 0$.

ii. $\begin{cases} \text{Do} \\ \quad j = j + 1 \\ \quad S = S + \text{Black–Scholes increment between times } t_{j-1} \text{ and } t_j \text{ knowing } S_{t_{j-1}} = S. \\ \quad \mathcal{S} := \sum_{1 \leq l \leq n} Q[l][j+1] f_j\left(S, Y_l^{j+1}\right) W[l][j] \\ \text{while } B(t_j, t_{j+1}) \frac{1}{n} \mathcal{S} > \varphi(S). \end{cases}$

iii. Compute $\xi = \xi + B(0, t_j)\varphi(S)$.

(c) (Output) The needed forward price is

$$Q_0^f := \xi / M$$

4.5 The Barraquand–Martineau algorithm.

4.5.1 Basic principle.

To avoid the high dimensionality problem, the authors propose to approximate the optimal stopping strategy by the following sub–optimal one : assume that the option holder knows at time t the payoff values $\{\varphi(S_r); r \geq t\}$ but not the stock values $\{S_r; r \leq t\}$. Then, the option holder can only exercise according to a strategy optimizing

$$\sup_{\tau \in \mathcal{G}_{0,N}} \mathbb{E}(B(0,\tau)\varphi(S_\tau)) \tag{41}$$

where $\mathcal{G}_{0,N}$ is the set of the $\mathcal{G}$–stopping times taking values in $\{t_0, \ldots, t_N\}$ and where $\mathcal{G}$ is the filtration generated by the payoff process.

To compute (41), the authors propose a quantization method acting on a one–dimensional dynamical programming algorithm.

4.5.2 Dynamical programming algorithm.

Let $\mathcal{G}_t := \sigma(\varphi(S_r); r \leq t), t \geq 0$ be the filtration generated by the PayOff process $(\varphi(S_t))_{t \geq 0}$. The authors approximate the dynamical programming (2) by

$$\begin{cases} Q_N := \varphi(S_{t_N}) \\ Q_{j-1} := \max\left(\varphi\left(S_{t_{j-1}}\right), \mathbb{E}\left(B(t_{j-1}, t_j) Q_j | \mathcal{G}_{t_{j-1}}\right)\right), \quad 1 \leq j \leq N. \end{cases} \tag{42}$$

Since the process $(\varphi(S_t))_{t \geq 0}$ is (in general) not Markov with respect to $\mathcal{G}$, a second approximation is done and (42) is replaced by

$$\begin{cases} Q_N := \varphi(S_{t_N}) \\ Q_{j-1} := \max\left(\varphi\left(S_{t_{j-1}}\right), \mathbb{E}\left(B(t_{j-1}, t_j) Q_j | \varphi\left(S_{t_{j-1}}\right)\right)\right), \quad 1 \leq j \leq N. \end{cases} \tag{43}$$

The algorithm (43) can now be handled by a one–dimensional quantization technique.

4.5.3 Quantized payoff space and transitions.

For $1 \leq j \leq N$, let $\left\{z_2^j < \ldots < z_{n_j}^j\right\} \subset \mathbb{R}$ and let $z_1^j := -\infty$, $z_{n_j+1}^j := +\infty$. Since S_{t_0} is deterministic, the payoff $\varphi(S_{t_0})$ need not to be discretized and we set $y^0 := \{\varphi(S_{t_0})\}$. Then, define the quantization sets $y^j := \left\{y_1^j, \ldots, y_{n_j}^j\right\}$ by

$$y_k^j := \mathbb{E}\left(\varphi\left(S_{t_j}\right) | \varphi\left(S_{t_j}\right) \in \left[z_k^j, z_{k+1}^j\right]\right). \tag{44}$$

Let $P_j(x, du)$ be the law of $\varphi\left(S_{t_{j+1}}\right)$ knowing that $\varphi\left(S_{t_j}\right) = x$. As in the quantization algorithm, the kernel P_j is discretized on $y^j \times y^{j+1}$ by the formulae

$$\widehat{P}_j\left(y_k^j, y_l^{j+1}\right) := \frac{\mathbb{E}\left(\mathbf{1}_{\left[z_k^j, z_{k+1}^j\right]}\left(\varphi\left(S_{t_j}\right)\right) P_j\left(\varphi\left(S_{t_j}\right), \left[z_l^{j+1}, z_{l+1}^{j+1}\right]\right)\right)}{\mathbb{E}\left(\mathbf{1}_{\left[z_k^j, z_{k+1}^j\right]}\left(\varphi\left(S_{t_j}\right)\right)\right)} \tag{45}$$

The next step is to approximate (43) with (44) and (45).

4.5.4 Backward price.

Let $\chi_j : \mathbb{R} \to \mathbb{R}$ such that $\mathbb{E}\left(B(t_j, t_{j+1}) Q_j | \varphi\left(S_{t_j}\right)\right) = \chi_j\left(\varphi\left(S_{t_j}\right)\right)$. For notational convenience, let $\alpha_{k,l}^j := \widehat{P}_j\left(y_k^j, y_l^{j+1}\right)$. Then, the quantized version of (43) is

$$\begin{cases} \chi_N\left(y_k^N\right) := y_k^N \quad 1 \leq k \leq n_N \\ \chi_j\left(y_k^j\right) := \max\left(y_k^j, \sum_{1 \leq l \leq n_{j+1}} \alpha_{k,l}^j \chi_{j+1}\left(y_l^{j+1}\right)\right), \quad 1 \leq k \leq n_j,\ 0 \leq j \leq N. \end{cases} \tag{46}$$

The needed backward price is finaly

$$Q_0^b = \chi_0\left(S_{t_0}\right).$$

Empirical version. The empirical version of (46) is simply obtained by taking the MonteCarlo estimators of (44) and (43),

$$\widetilde{y}_k^j = \frac{1}{\left|A_k^j\right|} \sum_{m \in A_k^j} \varphi\left(S_{t_j}\left(\omega^m\right)\right)$$

and

$$\widehat{\alpha}_{k,l}^j := \frac{\left|A_k^j \cap A_l^{j+1}\right|}{\left|A_k^j\right|}$$

where $\omega^1, \ldots, \omega^M$ is an i.i.d. sample from $\mathbb{P}$ and where

$$A_k^j := \left\{l; \varphi\left(S_{t_j}\left(\omega^l\right)\right) \in \left[z_k^j, z_{k+1}^j\right]\right\}$$

4.5.5 Practical design.

Quantization sets. Here, we don't use (44) to design the quantization sets y^j but we consider some mean order statistics. More precisely, for $1 \leq j \leq N$, assume that $n_j = n$, let $\varphi_1^j, \ldots, \varphi_n^j$ be n independent replications of the random variable $\varphi\left(S_{t_j}\right)$ and let $\phi_1^j, \ldots, \phi_n^j$ be the corresponding order statistics. Then, define

$$y_k^j := \mathbb{E}\left(\phi_k^j\right).$$

and replace the set $\left[z_k^j, z_{k+1}^j\right]$ by the Voronoï cell $C_k\left(y^j\right)$ (see (28)).

The implementation of such a procedure is straightforward (no distortion optimization algorithm is required) and one can show that the distortion induced by the set y^j is somewhat good.

Backward price. Once computed the quantization sets, the end of the algorithm is similar to a one–dimensional quantization method (see section (4.3)). The details are then omited.

5 Implementation.

5.1 Structure of the implementation.

The implementation of the pricing algorithm have been done in the C language. Since the Black–Scholes model, the payoff functions and some mathematical tools are used over all the algorithms, the corresponding routines have been grouped into specific files. The structure of the implementation then reads as follow :

1. Black–Scholes routines : the file *black.c* contains the routines related to the Black–Scholes model (essentially : backward brownian bridge generation, backward Black–Scholes stock path generation, forward Black–Scholes stock path generation, Black–Scholes stock path normalisation, density function of the Black–Scholes transition kernel).

2. Option routines : the files *option.c* contains the routines related to the pre–existing payoff functions. See section (5.2.3) for more informations.

3. Mathematical tools : the mathematical tools are grouped in the following files :

   (a) *basis.c* : contains the canonical basis for dimensions= $1 \ldots 10$ and the Hermite basis for dimensions= $1 \ldots 10$. These basis are used in the regression algorithms. See section (5.2.4) for more informations.

   (b) *random.c* : contains some pseudo–random numbers generators and the Box–Muller gaussian generator. See section (5.4) for more informations.

   (c) *lds.c* : contains the Sobol' low discrepancy generator. Used in the *loscbld* routine.

   (d) *sort.c* : contains a quick sort algorithm. Used in the Barraquand–Martineau algorithm to initialise the payoff quantizers.

(e) *cholesky.c* : contains the Cholesky decomposition algorithm and some other linear tools.

(f) *fmath.c* : contains the number π and the Max, Min functions.

4. Other tools :

   (a) *message.c* : contains the warning/error messages.

   (b) *memory.c* : memory allocation tools.

5. Payoff formulae compilation : the file *compile.c* contains a syntactic analysis–formulae tree constructor function and a tree evaluation function. Used in *option.c* for the non pre–existing payoff formulaes.

6. Pricing routines : the pricing routines are contained in the following files :

   (a) *loscb.c* : Longstaff–Schwartz algorithm, backward simulation (function *loscb*).

   (b) *loscbn.c* : Longstaff–Schwartz algorithm, backward simulation, normalised regressor (function *loscbn*).

   (c) *loscbh.c* : Longstaff–Schwartz algorithm, backward simulation, Hermite basis (function *loscbh*).

   (d) *loscbld.c* : Longstaff–Schwartz algorithm, backward simulation, for quasi-random generator.

   (e) *tsrob.c* : Tsitsiklis–VanRoy algorithm, backward simulation (function *tsrob*).

   (f) *raq.c* : Random quantization algorithm, backward simulation (function *raq*).

   (g) *qopt.c* : Optimal quantization algorithm, backward simulation (function *qopt*) .

   (h) *brgl.c* : Broadie–Glassermann algorithm (function *brgl*).

   (i) *bama.c* : Barraquand–Martineau algorithm (function *bama*).

7. Scilab interface : the file *intamc.c* contains the scilab interfaces of the pricing routines. See section (6) for more informations.

5.2 Pricing routines parameters.

5.2.1 Parameter families.

The head of each pricing function contains three parameter families. First, the algorithm family (section (5.2.4)); the related parameters have the prefix "*AL_*". Second, the option family (section (5.2.3)); the related parameters have the prefix "*OP_*". Third, the Black–Scholes family (section (5.2.2)); the related parameters have the prefix "*BS_*". The option and the Black–Scholes families do not depend on the head routines. The parameters of the Black–Scholes and the option families are all input parameters.

5.2.2 Black–Scholes family parameters.

Let δ be the size in octets of a double variable.

1. *BS_Spot* :

   Type : double pointer. Must point toward a memory vector of size $BS\_Dimension \times \delta$ octets.

   Meaning : initial values of the Black–Scholes stocks.

   Relevant values : coordinates > 0.

2. *BS_Interest_Rate* :

   Type : double.

   Meaning : riskfree interest rate of the Black–Scholes model.

   Relevant values : > 0.

3. *BS_Dividend_Rate* :

   Type : double pointer. Must point toward a memory vector of size $BS\_Dimension \times \delta$ octets.

   Meaning : the dividend rates of the Black–Scholes stocks.

   Relevant values : coordinates > 0.

4. *BS_Dimension* :

   Type : integer.

   Meaning : dimension of the Black–Scholes model

   Relevant values : > 0. CAUTION : the regression basis used in the loscb, loscbn, loscbh, tsrob routines are implemented only for the dimension $1 \ldots 10$.

5. *BS_Volatility* :

   Type : double pointer. Must point toward a memory vector of size $BS\_Dimension \times \delta$ octets.

   Meaning : volatilities of the Black–Scholes stocks.

   Relevant values : coordinates > 0.

6. *BS_Correlation* :

   Type : double pointer. Must point toward a memory vector of size $BS\_Dimension \times BS\_Dimension \times \delta$ octets.

   Meaning : correlations between the (correlated) standard brownian motions underlying the Black–Scholes stocks. Only the strict upper triangle Δ of the related matrix is taken into account.

   Relevant values : meaningful coordinates in $]-1, 1[$ and matrix with diagonal one and upper–lower triangles Δ positive definite.

5.2.3 Option family parameters.

Let δ be the size in octets of a double variable.

1. *OP_Option_Name* :

Type : char pointer.

Meaning : name of the payoff function to be used in the algorithm.

Relevant values : the pre–existing vanilla payoff functions are :

"PutMin" : $(OP\_Strike[0] - \min(s_1, \ldots, s_{BS\_Dimension}))_+$.

"CallMax" : $(\max(s_1, \ldots, s_{BS\_Dimension}) - OP\_Strike[0])_+$.

"PutBasket" : $\left(OP\_Strike[0] - \sum_{1 \le i \le BS\_Dimension} OP\_Basket[i]\, s_i\right)_+$.

"CallBasket" : $\left(\sum_{1 \le i \le BS\_Dimension} OP\_Basket[i]\, s_i - OP\_Strike[0]\right)_+$.

"PutGeom" : $\left(OP\_Strike[0] - (s_1 \ldots s_{BS\_Dimension})^{1/BS\_Dimension}\right)_+$.

"CallGeom" : $\left((s_1 \ldots s_{BS\_Dimension})^{1/BS\_Dimension} - OP\_Strike[0]\right)_+$.

"MinOfPut" : $\min_{1 \le i \le BS\_Dimension} \left((OP\_Strike[i] - s_i)_+\right)$.

"BestOfCall" : $\max_{1 \le i \le BS\_Dimension} \left((s_i - OP\_Strike[i])_+\right)$.

If the parameter *OP_Option_Name* does not belong to the preceding items, it is sent through a compiler formulae; see section (5.3).

2. *OP_Basket* :

Type : double pointer. When meaningful, must point toward a memory vector of size $BS\_Dimension \times \delta$ octets.

Meaning : Meaningful if the parameter *OP_Option_Name* belongs to {"*PutBasket*", "*CallBasket*"}. In that case, coefficients of the basket in the payoff functions "PutBasket" and "CallBasket".

Relevant values : no restriction.

3. *OP_Strike* :

Type : double pointer. If the parameter *OP_Option_Name* belongs to {"*MinOfPut*", "*BestOfCall*"}, must point toward a memory vector of size $BS\_Dimension \times \delta$ octets. If the parameter *OP_Option_Name* belongs to {"*PutMin*", "*CallMax*", "*PutBasket*", "*CallBasket*", "*PutGeom*", "*CallGeom*"}, must point toward a memory vector of size δ octets.

Meaning : Meaningful if the parameter *OP_Option_Name* is one of the pre–existing payoff functions. In that case, strike of the payoff functions {"*PutMin*", "*CallMax*", "*PutBasket*", "*CallBasket*", "*PutGeom*", "*CallGeom*"}, or strike vector of the payoff functions {"*MinOfPut*", "*BestOfCall*"}.

Relevant values : no restriction.

4. *OP_Maturity* :

Type : double.

Meaning : maturity of the option.

Relevant values : > 0.

5. $OP\_Exercice\_Dates$:

Type : integer.

Meaning : number of possible exercise dates of the bermudean option. The exercise dates are $t_j = OP\_Maturity.j/(OP\_Exercice\_Dates - 1)$, $j = 0, \ldots, OP\_Exercice\_Dates - 1$.

Relevant values : > 0.

5.2.4 Algorithms parameters.

The *loscb*, *loscbn* routines.

1. AL_FPrice (Output parameter)

Type : double pointer.

Meaning : adress of the output of the algorithm (forward price).

2. $AL\_MonteCarlo\_Iterations$

Type : long integer.

Meaning : MonteCarlo parameter of the algorithm. See section (4.1) for more informations.

Relevant values : no restriction. If negative, no computation is done.

3. AL_Basis_Name

Type : char pointer.

Meaning : name of the basis to be used for the regression procedures.

Relevant values : the implemented basis are : (canonical) "CanD1",...,"CanD10" and (Hermite) "HerD1",...,"HerD10".

4. $AL\_Basis\_Dimension$

Type : integer.

Meaning : dimension of the regression basis AL_Basis_Name.

Relevant values : > 0. The implemented maximal values are : "CanD1":20, "CanD2":21, "CanD3":20, "CanD4":21, "CanD5":20, "CanD6":19, "CanD7":20, "CanD8":23, "CanD9":26, "CanD10":29 , "HerD1":6, "HerD2":21, "HerD3":20, "HerD4":21, "HerD5":20, "HerD6":19, "HerD7":20, "HerD8":23, "HerD9":26, "HerD10":29.

5. $AL\_PayOff\_As\_Regressor$

Type : integer.

Meaning : the payoff function is introduced in the basis regression at every time greater than or equal to $AL\_PayOff\_As\_Regressor$.

Relevant values : no restriction.

6. *AL_Antithetic*

Type : integer.

Meaning : if $AL\_Antithetic! = 0$, antithetic Black–Scholes paths are used ($AL\_MonteCarlo\_Iterations/2$ plus $AL\_MonteCarlo\_Iterations/2$ antithetic).

Relevant values : no restriction.

7. *AL_ErrorMessage* (Output parameter)

Type : char pointer. Must point toward a sufficiently large memory vector of char (default size in the Scilab interface : 10000).

Meaning : contains the error/warning messages sent by the algorithm during its run.

8. *AL_ShuttingDown*

Type : integer.

Meaning : if $AL\_ShuttingDown = 0$, the function free is not used on the allocated variable. The routine can then be re–used with the same parameter $BS\_Dimension$ without new call to the function malloc.

Relevant value : no restriction.

The *loscbh* routine. The algorithm parameters family of the *loscbh* routine equals the *loscb* routine's one less the parameter AL_Basis_Name.

The *loscbld* routine. The algorithm parameters family of the *loscbld* routine equals the *loscb* routine's one less the parameter $AL_Antithetic$.

The *tsrob* routine. The algorithm parameters family of the *tsrob* routine equals the *loscb* routine's one less the parameter $AL\_Antithetic$ and plus the output parameter AL_BPrice :

Type : double pointer.

Meaning : adress of the first output of the algorithm (backward price).

The *raq* routine.

1. *AL_BPrice* (Output parameter)

Type : double pointer.

Meaning : adress of the first output of the algorithm (backward price).

2. *AL_FPrice* (Output parameter)

Type : double pointer.

Meaning : adress of the second output of the algorithm (forward price).

3. *AL_MonteCarlo_Iterations*

Type : long integer.

Meaning : MonteCarlo parameter of the algorithm. See section (4.3.8) for more informations.

Relevant values : no restriction. If negative, no computation is done.

4. *AL_T_Size*

Type : integer.

Meaning : size of the random quantizers.

Relevant values : > 0.

5. *AL_ErrorMessage* (Output parameter) see the preceding routines.

6. *AL_ShuttingDown* see the preceding routines.

The *qopt* routine. The algorithm parameters family of the *qopt* routine equals the *raq* routine's one plus the two following parameters :

1. *AL_Tesselation_Path*

Type : char pointer.

Meaning : name of the path containing the tesselation *AL_Tesselation_Name*.

2. *AL_Tesselation_Name*

Type : char pointer.

Meaning : name of the tesselation to be used by the algorithm. To use the default tesselation, set *AL_Tesselation_Name* =″ *d*″; the files "d[BS_Dimension]n[AL_T_Size].tes will be loaded. If you want to design your own tesselation, adopt the following format:

```
fprintf(YourFile,"%s\n",String1);
fprintf(YourFile,"%s\n",String2);
fprintf(YourFile,"%s\n",String3);
fprintf(YourFile,"%s\n",String4);
for (i=0;i<AL_T_Size;i++){
fprintf(YourFile,"%f %f ... %f ",YourTesselation[i][1],...,YourTesselation[i][BS_Dimension]);
fprintf(YourFile,"%f\n",Radius[i]);
}
```

where String1,..,String4 are some strings without space containing some informations or comments and where *Radius*[*i*] is the radius of the Voronoï cell of the ith point of your tesselation. Then, set *AL_Tesselation_Name* to be the name of your file. See section (4.3.9) for more information.

The *brgl* routine. The algorithm parameters family of the *brgl* routine equals the *raq* routine's one but the parameters *AL_T_Size* is replaced by the parameters *AL_Mesh_Size* (integer > 0, size of the meshes; see section (4.4) for more informations).

The *bama* routine.

1. *AL_BPrice* (Output parameter)

Type : double pointer.

Meaning : adress of the first output of the algorithm (backward price).

2. *AL_MonteCarlo_Iterations*

Type : long integer.

Meaning : MonteCarlo parameter of the algorithm. See section (4.5) for more informations.

Relevant values : no restriction. If negative, no computation is done.

3. *AL_PO_Size*

Type : integer.

Meaning : size of the payoff quantizers.

Relevant values : > 0.

4. *AL_PO_Init*

Type : integer.

Meaning : number of order statistic used to initialise the payoff quantizers.

Relevant values : > 0.

5. *AL_ErrorMessage* (Output parameter) see the preceding routines.

6. *AL_ShuttingDown* see the preceding routines.

5.3 Payoff formulae.

When the parameter *OP_Option_Name* is not the name of a pre–existing payoff function, it is sent through a syntactic analyser. The generic syntax of the string *OP_Option_Name* must be

$$\text{“}Idt_1 = Num_1; \ldots; Idt_N = Num_N @ f\,(s1, \ldots, sd, Idt_1, \ldots, Idt_N, Cst_1, \ldots, Cst_M)\text{”}$$

where d is the Black–Scholes dimension, $s1, \ldots, sd$ are the reserved word for the Black–Scholes stocks, $Idt_1, \ldots, Idt_N$ are some alphabetic identifiers, and where $Num_1, \ldots, Num_N, Cst_1, \ldots, Cst_M$ are some flotting point numbers or some pre–existing constants.

First, the identifiers declared before “@” are analysed. If the related syntax is correct without duplicate identifiers, the formulae

$$\text{“}f\,(s1, \ldots, sd, Idt_1, \ldots, Idt_N, Cst_1, \ldots, Cst_M)\text{”} \tag{47}$$

is analysed. If the syntax of (47) is correct according to the grammar depicted in the file *compile.c*, the output of the compilation process is a son–brothers formulae tree. This tree is evaluated during the run of the pricing algorithm without new syntactic analysis.

The available functions are (a function can have several names):

1. Arity one : sin, cos, tan, tg, asin, arcsin, arccos, acos, atan, arctan, exp, log, ln, cosh, ch, acosh, argch, sinh, sh, asinh, argsh, tanh, th, atanh, argth, rac (=square root), sqrt, Abs (absolute value), E (integer part), trc (integer part), dec (decimal part).

2. Undefined arity : max, min, ind (ind($a1, \ldots, aN$)=1 if $\{a1 < \ldots < aN\}$, =0 otherwise).

The available pre-existing constants are π and e.

Some example of formulaes are given in the file *lists.sci*.

5.4 Pseudo–random and quasi–random numbers generators.

The available pseudo–random numbers generators are : KNUTH, LECUYER, MRGK3, MRGK5, SHUFL, TAUS. The single available quasi–random generator is SOBOL (Antonov&Saleev version of the Sobol sequence, random direction numbers). It is used only in the *loscbld* routine. Note that the generator to be used by the pricing routines does not belong to the routines parameters but is a parameter of the Scilab interface. To use the generator and the pricing routines straight from a C–source (without the Scilab interface), see section (7).

6 Scilab interface.

The Scilab interface *intamc.c* is a C–file using some translation routines from the Scilab variables to the C variables and containing the pricing routines calls with the translated parameters. It allows to create a Scilab library containing the Scilab functions *loscb*, *loscbn*, *loscbh*, *tsrob*, *raq*, *qopt*, *bama* and *brgl*.

In the next section, we describe in details the input/output Scilab parameters of these functions. Some examples of these parameters are given in the file *lists.sci*. See also the section (6.3).

6.1 Functions parameters

The functions contained in the library *amclib* have three input parameters :

1. An algorithm T–list containing the Scilab versions of the C parameter algorithm family (see section (6.1.3)).

2. An option T–list containing the Scilab versions of the C parameter option family (see section (6.1.2)).

3. A Black–Scholes T–list containing the Scilab versions of the C parameter Black–Scholes family (see section (6.1.1)).

and one output T–list. The generic call of the pricing functions is

$$--> z = Function\_Name\,(ALL, BSL, OPL)$$

where ALL is an algorithm T–list, BSL is a Black–Scholes T–list and where OPL is an option T–list

6.1.1 The Black–Scholes T–list.

1. type : Scilab T–list.
2. fields : (the name of the fields have no meaning for the interface)
   (a) (types field) ["BS","BS_Dimension","BS_Spot", "BS_Interest_Rate","BS_Dividend_Rate","BS_Volatility","BS_Correlation"]
   (b) ("BS_Dimension" field) integer > 0, dimension of the Black-Scholes Model.
   (c) ("BS_Spot" field) vector of size BS_Dimension , initial values of the Black-Scholes stocks.
   (d) ("BS_Interest_Rate" field) real number > 0, riskfree interest rate of the Black-Scholes model.
   (e) ("BS_Volatility" field) vector of size BS_Dimension, volatilities of the Black-Scholes stocks.
   (f) ("BS_Correlation" field) several possible formats :
      i. real number between -1 and 1 : constant correlation between the Black-Scholes stocks.
      ii. matrix of size (BS_Dimension,BS_Dimension) : matrix correlation of the Black-Scholes stocks; the scilab interface only takes into account the strict upper triangle of the matrix.
      iii. vector of size BS_Dimension*(BS_Dimension-1)/2 : concatenation of the rows of the strict upper triangle of the Black-Scholes stocks correlations matrix.

6.1.2 The option T–list.

1. type : Scilab T–list.
2. fields : (the name of the fields have no meaning for the interface)
   (a) (types field) ["OP","OP_Option_Name","OP_Exercise_Dates", "OP_Basket","OP_Strike"]
   (b) ("OP_Option_Name" field) string; two possible formats :
      i. one of the following pre-existing payoff functions : "PutBasket", "CallBasket", "PutMin", "CallMax", "PutGeom", "CallGeom", "BestOfCall", "MinOfPut". (see section (5.2.3) for the definition of these functions).
      ii. payoff formulae. See section (5.3) for the syntactic requirements and the available functions.
   (c) ("OP_Exercise_Dates" field) integer > 0; number of available exercise dates for the bermudean option. The exercise dates are $t_j = OP\_Maturity.j/(OP\_Exercice\_Dates - 1)$, $j = 0, \ldots, OP\_Exercice\_Dates - 1$.
   (d) ("OP_Basket" field) if OP_Option_Name="PutBasket" or ="CallBasket", vector of size BS_Dimension. OP_Basket[i] is the coefficient of the ith BlackScholes stock in the basket. This parameter has no significance if OP_Option_Name<>"PutBasket" and <>"CallBasket".two possible formats :

(e) ("OP_Strike" field) two possible formats :

i. if OP_Option_Name is in "PutBasket","CallBasket","PutMin", "CallMax","PutGeom","CallGeom", real number, strike of the corresponding option.

ii. if OP_Option_Name is in "BestOfCall", "MinOfPut", vector of size BS_Dimension, strikes of the calls (resp. put) related to the option "BestOfCall" (resp. "MinOfPut").

6.1.3 The algorithm T–list.

The *loscb, loscbn, tsrob* **T–list.**

1. type : Scilab T–list.
2. fields : (the name of the fields have no meaning for the interface)
   (a) (types field) ["LOSCB","AL_Computations_Number", "AL_MonteCarlo_Iterations","AL_Generator_Name", "AL_Basis_Name","AL_Basis_Dimension","AL_PayOff_As_Regressor", "AL_Antithetic"]
   (b) ("AL_Computations_Number" field) integer > 0, number of computations to be done.
   (c) ("AL_MonteCarlo_Iterations" field) (long) integer > 0. MonteCarlo parameter; Its significance depends on the used algorithm.
   (d) ("AL_Generator_Name" field) string. Name of the pseudo–random numbers generator to be used. The possible values of this parameter are : "KNUTH", "LECUYER", "MRGK3", "MRGK5", "SHUFL", "TAUS".
   (e) ("AL_Basis_Name" field) string, name of the regression basis to be used. The implemented basis are : canonical : "CanD1", "CanD2", ... , "CanD10". Hermite : "HerD1", "HerD2", ... "HerD10". For higher Black–Scholes dimension, implement the needed basis in the file basis.c .
   (f) ("AL_Basis_Dimension" field) integer > 0, dimension of the regression basis AL_Basis_Name. The implemented maximal values are : CanD1:20, CanD2:21, CanD3:20, CanD4:21, CanD5:20, CanD6:19, CanD7:20, CanD8:23, CanD9:26, CanD10:29 , HerD1:6, HerD2:21, HerD3:20, HerD4:21, HerD5:20, HerD6:19, HerD7:20, HerD8:23, HerD9:26, HerD10:29.
   (g) ("AL_PayOff_As_Regressor" field) integer. The payoff function is introduced in the regression basis at every time greater than or equal to AL_PayOff_As_Regressor.
   (h) ("AL_Antithetic" field) integer, if AL_Antithetic! $= 0$, use of antithetic Black–Scholes paths.

The *loscbh* **T–list.** T–list of type "LOSCB" without the field "AL_Basis_Name".

The *loscbld* **T–list.** T–list of type "LOSCB" without the field "AL_Antithetic". Note that the relevant value for the field "AL_Generator_Name" is 'SOBOL'.

The *raq* T–list

1. type : Scilab T–list.
2. fields : (the name of the fields have no meaning for the interface)
   (a) (types field) ["RAQ","AL_Computations_Number", "AL_MonteCarlo_Iterations","AL_Generator_Name","AL_T_Size"]
   (b) ("AL_Computations_Number" field) already defined.
   (c) ("AL_MonteCarlo_Iterations" field) already defined.
   (d) ("AL_Generator_Name" field) already defined.
   (e) ("AL_T_Size" field) integer > 0, size of the random tesselations.

The *qopt* T–list.

1. type : Scilab T–list.
2. fields : (the name of the fields have no meaning for the interface)
   (a) (types field) ["QOPT","AL_Computations_Number", "AL_MonteCarlo_Iterations","AL_Generator_Name","AL_T_Size", "AL_Tesselation_Path","AL_Tesselation_Name"]
   (b) ("AL_Computations_Number" field) already defined.
   (c) ("AL_MonteCarlo_Iterations" field) already defined.
   (d) ("AL_Generator_Name" field) already defined.
   (e) ("AL_T_Size" field) integer > 0, size of the random tesselations.
   (f) ("AL_Tesselation_Path" field) string, path of the optimal tesselation to be used. See section (5.2.4), paragraph "The *qopt* routine".
   (g) ("AL_Tesselation_Name" field) string, name of the optimal tesselation to be used. If AL_Tesselation_Name="", the default tesselation d[BS_Dimension]n[AL_TSize].tes is called.

The *brgl* T–list.

1. type : Scilab T–list.
2. fields : (the name of the fields have no meaning for the interface)
   (a) (types field) ["BRGL","AL_Computations_Number", "AL_MonteCarlo_Iterations","AL_Generator_Name","AL_Mesh_Size"]
   (b) ("AL_Computations_Number" field) already defined.
   (c) ("AL_MonteCarlo_Iterations" field) already defined.
   (d) ("AL_Generator_Name" field) already defined.
   (e) ("AL_Mesh_Size" field) integer > 0, size of the meshes.

The *bama* T–list.

1. type : Scilab T–list.
2. fields : (the name of the fields have no meaning for the interface)
   (a) (types field) ["BAMA","AL_Computations_Number", "AL_MonteCarlo_Iterations","AL_Generator_Name","AL_Mesh_Size"]
   (b) ("AL_Computations_Number" field) already defined.
   (c) ("AL_MonteCarlo_Iterations" field) already defined.
   (d) ("AL_Generator_Name" field) already defined.
   (e) ("AL_PO_Size" field) integer > 0, size of the quantizers of the payoff space to be used.
   (f) ("AL_PO_Init" field) integer > 0, number of order statistics used to compute the quantizers.

6.1.4 The output parameter.

The *loscb, loscbn, loscbh, loscbld, tsrob* output T–list.

1. type : Scilab T–list.
2. fields : (the name of the fields have no meaning for the interface)
   (a) (types field) ["Alg_Name_Price","AL_FPrice","AL_ErrorMessage"]
   (b) ("AL_FPrice" field) vector of size "AL_Computations_Number", for prices.
   (c) ("AL_ErrorMessage" field) string. Error/warning messages sent by the algorithm during the computations.

The *raq, qopt, brgl* output T–list.

1. type : Scilab T–list.
2. fields : (the name of the fields have no meaning for the interface)
   (a) (types field) ["Alg_Name_Price","AL_BPrice","AL_FPrice","AL_ErrorMessage"]
   (b) ("AL_BPrice" field) vector of size "AL_Computations_Number", backward prices.
   (c) ("AL_FPrice" field) already defined.
   (d) ("AL_ErrorMessage" field) already defined.

The *bama* output T–list.

1. type : Scilab T–list.
2. fields : (the name of the fields have no meaning for the interface)
   (a) (types field) ["Alg_Name_Price","AL_BPrice","AL_ErrorMessage"]
   (b) ("AL_BPrice" field) already defined.
   (c) ("AL_ErrorMessage" field) already defined.

6.2 Build and use of the Scilab interface.

6.2.1 Building the *amclib* Scilab library.

The working directory must contain the files *intamc.c*, *bama.c (.h)*, *brgl.c (.h)*, *loscb.c (.h)*, *loscbn.c (.h)*, *loscbh.c (.h)*, *loscbld.c (.h)*, *raq.c (.h)*, *qopt.c (.h)*, *tsrob.c (.h)*, *basis.c (.h)*, *cholesky.c (.h)*, *random.c (.h)*, *lds.c (.h)*, *message.c (.h)*, *memory.c (.h)*, *black.c (.h)*, *option.c (.h)*, *fmath.c (.h)*, *compile.c (.h)*, *sort.c (.h)* and the file *builder.sce*. To build the library *amclib*, simply type

−− > exec builder.sce

The Scilab object files *.lo* and the Scilab library *amclib.so* are created.

6.2.2 Using the *amclib* Scilab library.

When begining a Scilab session, get the working directory to be the one containing the library and type

−− > exec loader.sce

The functions *loscb*, *loscbn*, *loscbh*, *tsrob*, *raq*, *qopt*, *bama* and *brgl* are loaded. See the section (6.1) to design the parameters of the functions. Numerous examples of parameters are given in the file *lists.sci*; many of them can be tested with the function *examc.sci*; see the next section.

6.3 The function *examc.sci*.

The file *lists.sci* contains about 100 examples of T–lists parameters. 80 of them can be runned with the function *examc* (contained in the same file). To load this function, type

−− > getf('lists.sci')
Then, call the function with the two following parameters :

1. the Black–Scholes dimension, between 1 and 10,
2. the name of a Scilab pricing function,

(for instance, −− > z=examc(4,'loscb')).
The output parameter of the function *examc.sci* is the following T–list

1. (types field) ['EX','b','o','a','p'] :
2. ('b' field) the Black–Scholes T–list used in the example.
3. ('o' field) the option T–list used in the example.
4. ('a' field) the algorithm T–list used in the example.
5. ('p' field) the output T–list of the pricing function called with the T–lists 'a','b' and 'o'.

7 Using the pricing routines from a C–source.

If you want to use the pricing routines straight from a C–source (*i.e.*, without the Scilab interface), get the following includes in your source file :

#include "loscb.h"
#include "loscbn.h"
#include "loscbh.h"
#include "loscbld.h"
#include "raq.h"
#include "qopt.h"
#include "brgl.h"
#include "bama.h"
#include "tsrob.h"
#include "random.h"
#include "lds.h"

and proceed your source compilation with the files *bama.c*, *brgl.c*, *loscb.c*, *loscbn.c* , *loscbh.c*, *raq.c*, *qopt.c*, *tsrob.c*, *basis.c*, *cholesky.c*, *random.c*, *lds.c*, *message.c*, *memory.c*, *black.c*, *option.c*, *fmath.c*, *compile.c*, *sort.c*.

Before to use the routines (except *loscbld*), initialise (or re–initialise) the generator with the function

*void InitGenerator(char *ErrorMessage, char *Generator_Name)*

The parameter $ErrorMessage$ must point toward a memory vector of minimal size 42 octets (length of the error message if the generator is unknown). An example of call is

InitGenerator(ErrorMessage,"LECUYER");

See section (5.2) to design the parameters needed by the pricing routines.

Before to use the routine *loscbld*, initialise (or re–initialise) the quasi–random generator with the function

*void InitLDGenerator(char *ErrorMessage, char *LDGenerator_Name, long Dimension, long Working_Dimension)*

The parameter $ErrorMessage$ must point toward a memory vector of minimal size 41 octets (length of the error message if the generator is unknown). The parameter *Dimension* must be $2*BS\_Dimension*OP\_Exercise\_Dates$ the parameter *Working_Dimension* must be $2 * BS\_Dimension$. For the time being, the single available quasi–random generator is "SOBOL" so the parameter *LDGenerator_Name* must be "SOBOL".

To remove the memory used by the generator, use the function *void KillLDGenerator()*.

8 Rogers Methods

This section is due to J.F.Bergez.
Rogers proposes in [10] a new Monte Carlo method based on Lagrangian martingales to price the American put. The expectation of the pathwise maximum of the option payoff less any martingale starting from 0 at $t = 0$ gives an upper bound for the price of the option. A good choice of the martingale makes this bound accurate.

Notations

We consider a single primitive asset whose price $(S_t)_{0\leq t}$ satisfies the Black and Scholes equation under the risk-neutral probability measure :

$$\begin{array}{l} S_0 > 0 \\ dS_t = S_t((r-d)dt + \sigma dW_t) \quad 0 \leq t \leq T \end{array} \tag{48}$$

where

- T is the maturity date
- r is the risk free rate
- d is the dividend rate
- σ is the volatility parameter
- $(W_t)_{0\leq t\leq T}$ is a standart brownian motion
- $\mu = r - d - \sigma^2/2$

The main result

Let $Z_t = e^{-rt}(K - S_t)^+$ and Y_t denote respectively the discounted payoff and price of the American put at time $t \leq T$. This method is based on a theorical result given and demonstrated in [**?**] :

$$Y_0 = \inf_{M\in H_0^1} \mathrm{E}\left[\sup_{0\leq t\leq T}(Z_t - M_t)\right] \tag{49}$$

where H_0^1 is the space of martingales M for which $\sup_{0\leq t\leq T}|M_t| \in L^1$, and such that $M_0 = 0$.
Proof : For $(M_t)_{t\leq T} \in H_0^1$ and τ denoting any stopping time of the Brownian filtration smaller than T, $\mathrm{E}(M_\tau) = 0$. Therefore

$$Y_0 = \sup_\tau \mathrm{E}(Z_\tau) = \sup_\tau \mathrm{E}(Z_\tau - M_\tau) \leq \mathrm{E}\left(\sup_{t\leq T}(Z_t - M_t)\right)$$

Hence the left hand side of (49) is smaller than the right hand side. According to well known results concerning the American put, there is a predictable non decreasing process $(A_t)_{t\leq T}$ (which comes from the Doob Meyer decomposition of the super-martingale $(Y_t)_{t\leq T}$) such that $A_t = 0$ for $t \leq \tau = \inf\{s \leq T : Y_s = Z_s\}$ and $M_t = Y_t - Y_0 + A_t$ belongs to H_0^1.
Now since $\forall t \leq T, \quad Y_t \geq Z_t$ and $A_t \geq 0$

$$\begin{array}{rcl} \sup_{t\leq T}(Z_t - M_t) & = & Y_0 + \sup_{t\leq T}(Z_t - Y_t - A_t) \\ & = & Y_0 + Z_\tau - Y_\tau - A_\tau \\ & = & Y_0 \end{array}$$

which gives the reverse inequality.

The method uses this result by choosing a good martingale which gives an accurrate upper bound. Then the expectation $\mathrm{E}\left[\sup_{0\leq t\leq T}(Z_t - M_t)\right]$ is evaluated by Monte Carlo.

The choice of the martingale

In [**?**] Rogers suggests to take M_t equal to the martingale part of the European put starting where the option goes first in the money :

$$dM_t = \mathrm{I}_{\{t^* \leq t\}} d\tilde{P}(t, S_t) \tag{50}$$

where $t^* = \inf\{0 \leq t : S_t \leq K\}$ and $\tilde{P}(t, S_t)$ is the discounted price of the European put ie $\tilde{P}(t, S_t) = e^{-rt} P(t, S_t)$. By the Black and Scholes formula $P(t, S_t) = Ke^{-r(T-t)} N(-d_2) - S_t e^{-d(T-t)} N(-d_1)$ when

- $d_1 = \frac{\ln(S_t/K) + (r - d + \sigma^2/2)(T-t)}{\sigma\sqrt{T-t}}$
- $d_2 = d_1 - \sigma\sqrt{T-t}$

N is the standart normal cumulative distribution function.

Implementation

For the previous choice of $(M_t)_{0 \leq t \leq T}$, $\inf_{\lambda \in \mathbb{R}} \mathrm{E}\left[\sup_{0 \leq t \leq T} (Z_t - \lambda M_t)\right]$ is approximated by Monte Carlo.

The first step is devoted to the approximation of λ^* wich realizes the minimum of $\mathrm{E}\left[\sup_{0 \leq t \leq T} (Z_t - \lambda M_t)\right]$. N_p paths are simulated with n equal time steps. The function $\lambda \mapsto \frac{1}{N_p} \sum_{i=1}^{N_p} \sup_{0 \leq k \leq n} \left(Z^i_{\frac{kT}{n}} - \lambda M^i_{\frac{kT}{n}} \right)$ being convex, we proceed by dichotomy to find $\hat{\lambda}$ a zero of a finite difference approximation of its derivative.

The second step uses the above estimated $\hat{\lambda}$ and a larger number N of simulated paths to return the result

$$\hat{Y}_0(S_0) = \frac{1}{N} \sum_{i=1}^{N} \sup_{0 \leq k \leq n} \left(Z^i_{\frac{kT}{n}} - \hat{\lambda} M^i_{\frac{kT}{n}} \right). \tag{51}$$

The delta is approximated by finite differences : $\frac{\hat{Y}_0(S_0+h) - \hat{Y}(S_0)}{h}$ where $\hat{Y}_0(S_0+h)$ is computed concurrently with $\hat{Y}_0(S_0)$ using the same random numbers and the same $\hat{\lambda}$.

References

[1] J.BARRAQUAND D.MARTINEAU. Numerical valuation of high dimensional multivariate american securieties. *J.Of Finance and Quantitative Analysis*, (30):383–405, 1995.

[2] F.A.LONGSTAFF E.S.SCHWARTZ. Valuing american options by simulations:a simple least-squares approach. *Working Paper Anderson Graduate School of Management University of California*, 25, 1998.

[3] G.PAGES. A space vector quantization for numerical integration. *Journal of Applied and Computational Mathematics*, 89:1–38, 1997.

[4] J.C.FORT G.PAGES. About the a.s. convergence of the kohonen algorithm with a general neighborhood function. *The Annals of Applied Probability*, 5(4), 1995. 16

[5] P.COHORT. Weak and strong law of large numbers for the random normalised distortion. *Submited for publication*, 2000. 14

[6] M.BROADIE P.GLASSERMANN. Pricing american-style securities using simulation. *J.of Economic Dynamics and Control*, 21:1323–1352, 1997.

[7] M.BROADIE P.GLASSERMANN. A stochastic mesh method for pricing high-dimensional american options. *Working Paper*, Columbia University:1–37, 1997.

[8] A.GERSHO R.M.GRAY. *Vector Quantization and Signal Compression.* Kluwer, 7th edition, 1992.

[9] M.J.SABIN R.M.GRAY. Global convergence and empirical consistency of the generalised lloyd algorithm. *IEEE Transactions on Information Theory*, 32:148–155, March 1986. 16

[10] L.C.G. Rogers. Montecarlo valuation of american option. *Preprint*, 2000. 37

[11] J.N.TSITSIKLIS B.VAN ROY. Optimal stopping of markov processes: Hilbert spaces theory, approximations algorithms and an application to pricing high-dimensional financial derivatives. *IEEE Transactions on Automatic Control*, 44(10):1840–1851, October 1999.

[12] J.N.TSITSIKLIS B.VAN ROY. Regression methods for pricing complex american-style options. *Working Paper*, MIT:1–22, 2000.