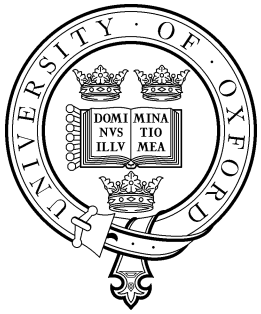
Investigation into Vibrato Monte Carlo for the Computation of Greeks of Discontinuous Payoffs



Sylvestre Burgos
Lady Margaret Hall
University of Oxford

A thesis submitted in partial fulfillment of the
MSc in Mathematical and Computational Finance

July 2009

Contents

Chapter 1

Abstract

Monte Carlo simulation is a popular method in computational finance. Its basic theory is relatively simple, it is also quite easy to implement and allows nevertheless an efficient pricing of financial options, even in high-dimensional problems (basket options, interest rates products...).

The pricing of options is just one use of Monte Carlo in finance. More important than the prices themselves are their sensitivities to input parameters (underlying asset value, interest rates, market volatility...). Indeed we need those sensitivities (also known as "Greeks") to hedge against market risk.

In this paper, we will first recall classical approaches to the computation of Greeks through Monte Carlo simulation: finite differences, Likelihood Ratio method (LRM) and Pathwise Sensitivities (PwS). Each of those approaches has particular limitations in the case of options with discontinuous payoffs. We will expound those limitations and introduce a new hybrid method proposed by Prof. Mike Giles, the Vibrato Monte Carlo, which combines both Pathwise Sensitivity and Likelihood Ratio methods to get around their shortcomings.

We will discuss the possible use of Vibrato Monte Carlo ideas for options with discontinuous payoffs. My personal contribution is an improvement to the standard Vibrato Monte Carlo yielding both computational savings and an improved accuracy. I will call it Allargando Vibrato Monte Carlo (AVMC). I then also extend the Vibrato Monte Carlo technique to discretely sampled path dependent options (digital option with discretely sampled barrier, lookback option with discretely sampled maximum).

Chapter 2

Computation of Greeks in a Monte Carlo Context

We will consider an underlying asset whose evolution is described by the following SDE.

$$\mathrm{d}S(t) = a(S,t)\mathrm{d}t + b(S,t)\mathrm{d}W_t \tag{2.1}$$

$a(S,t)$ is the drift and $b(S,t)$ the volatility.

2.1 Classical Methods

In the general case, the evolution SDE (2.1) cannot be integrated exactly. We will hence use the Euler-Maruyama discretization which is the simplest approximation of the SDE.

We split the time interval $[0,T]$ into N regular intervals of length $h = T/N$. We then use

$$\hat{S}_{n+1} = \hat{S}_n + a(\hat{S}_n, t_n)h + b(\hat{S}_n, t_n)\Delta W_{n+1} \tag{2.2}$$

$\hat{S}_n$ is the approximation of $S(nh)$. ΔW_n are brownian increments; these can be constructed as $\Delta W_n = LZ_n$ where Z_n is a vector of independent $\mathcal{N}(0,1)$ brownian increments and L is a matrix such that $LL^T = Cov(\Delta W_n)$.

2.1.1 Finite Difference

Let's consider θ an input parameter (underlying asset's value, risk-free interest rate, volatility, time to expiry...). Let's also define $V(\theta)$ to be the European option value as a function of this input parameter.

$$V = \exp(-rT)\,\mathbb{E}\left[f(S(T))\right] \tag{2.3}$$

$f(S_T)$ is the payoff of our option.

$\exp(-rT)$ is the discount factor. For the sake of readability we will not take this factor into account in the following pages. Indeed we can (as in many papers) consider without changing the theory that

$$V = \mathbb{E}\left[f(S(T))\right] \tag{2.4}$$

We must only pay attention to the fact that taking this factor into account will induce minor calculation changes when computing sensitivities to r or T.

We evaluate (2.4) through Monte Carlo.

The simplest approach to computing the first and second order derivatives with respect to θ is to use finite differences. Using central differences we get:

$$\frac{\partial V}{\partial \theta} = \frac{V(\theta + \Delta\theta) - V(\theta - \Delta\theta)}{2\Delta\theta} \tag{2.5}$$

$$\frac{\partial^2 V}{\partial \theta^2} = \frac{V(\theta + \Delta\theta) - 2V(\theta) + V(\theta - \Delta\theta)}{(\Delta\theta)^2} \tag{2.6}$$

The advantages of this somewhat naive method are its simplicity and ease of implementation. Its main disadvantage is that the choice of $\Delta\theta$ is the result of a trade-off. A too large $\Delta\theta$ can lead to a significant bias due to approximation error, whereas a too small $\Delta\theta$ will lead to a very large variance in cases where the payoff is not differentiable enough. See [1] or [6] for more details.

2.1.2 Likelihood Ratio Method (LRM)

For a one-dimensional SDE whose terminal probability distribution is known, we can write

$$V = \mathbb{E}\left[f(S(T))\right] = \int f(S)p(S)\,dS \tag{2.7}$$

As the dependence of V on θ only comes through $p(S)$, we can write (assuming some light conditions discussed in [1] in pages 393-395) that

$$\frac{\partial V}{\partial \theta} = \int f \frac{\partial p}{\partial \theta}\,dS = \int f \frac{\partial \log p}{\partial \theta} p\,dS = \mathbb{E}\left[f \frac{\partial \log p}{\partial \theta}\right] \tag{2.8}$$

We then compute this expectation through standard Monte Carlo simulations.

This method generalizes for an SDE path simulation and a payoff which is a function of the N intermediate states $\hat{S} = (\hat{S}_n)_{n\in[0,N]}$

$$\frac{\partial V}{\partial \theta} = \int f(\hat{S}) \frac{\partial p(\hat{S})}{\partial \theta}\,d\hat{S} = \int f(\hat{S}) \frac{\partial \log p(\hat{S})}{\partial \theta} p(\hat{S})\,d\hat{S} = \mathbb{E}\left[f(\hat{S}) \frac{\partial \log p(\hat{S})}{\partial \theta}\right] \tag{2.9}$$

Here we write $d\hat{S} = \prod_{k=1}^{N} d\hat{S}_k$ and $p(\hat{S}) = \prod_{k=1}^{N} p(\hat{S}_k|\hat{S}_{k-1})$ which hence yields

$$V = \mathbb{E}\left[f(\hat{S})\frac{\partial \log p(\hat{S})}{\partial \theta}\right] = \mathbb{E}\left[f(\hat{S})\sum_{k=1}^{n}\frac{\partial \log p(\hat{S}_k|\hat{S}_{k-1})}{\partial \theta}\right] \tag{2.10}$$

In the same manner, we can also compute higher orders sensitivities through the Likelihood Ratio method. Differentiating twice, we get the following expression.

$$\frac{\partial^2 V}{\partial \theta^2} = \int f\frac{\partial^2 p}{\partial \theta^2}\,dS = \mathbb{E}\left[f\left(\frac{\partial^2 \log p}{\partial \theta^2} + \left(\frac{\partial \log p}{\partial \theta}\right)^2\right)\right] \tag{2.11}$$

One of the most significant advantages of the Likelihood Ratio method is that it does not require anywhere the payoff function to be differentiable. Hence we can even use it in cases where this condition is not met.

The drawback of this method is that we need to know the final distribution of S to use its first (simple) version. Otherwise we have to go through SDE path simulations, which leads to a Monte Carlo estimator whose variance is $O(h^{-1})$ (See [1]). It is particularly ill-adapted for path-dependent options (barrier options, lookback options ...) where very fine time steps are required.

2.1.3 Pathwise Sensitivity

In the Pathwise Sensitivity method, we write (assuming some light conditions discussed in [1], pages 393-395 and [7], page 250) using the chain rule

$$\frac{\partial V}{\partial \theta} = \frac{\partial}{\partial \theta}\mathbb{E}\left[f(S_T)\right] = \mathbb{E}\left[\frac{\partial f(S_T)}{\partial \theta}\right] = \mathbb{E}\left[\frac{\partial f}{\partial S}\frac{\partial S_T}{\partial \theta}\right] \tag{2.12}$$

This expectation is then evaluated through Monte Carlo simulations:
$\frac{\partial f}{\partial S}$ is a property of the payoff function.
$\frac{\partial S_T}{\partial \theta}$ is obtained by differentiating (2.2) with respect to θ to get (2.13) and then iterating it.

In the general case (where h and ΔW_{n+1} may depend on Θ), we get:

$$\frac{\partial \hat{S}_{n+1}}{\partial \theta} = \frac{\partial \hat{S}_n}{\partial \theta} + \frac{\partial a(\hat{S}_n, t_n)h}{\partial \theta} + \frac{\partial b(\hat{S}_n, t_n)\Delta W_{n+1}}{\partial \theta} \tag{2.13}$$

In the most frequent case where h and ΔW_{n+1} do not depend on Θ (that is for all first order sensitivities but the sensitivity to time Θ), this simplifies to:

$$\frac{\partial \hat{S}_{n+1}}{\partial \theta} = \frac{\partial \hat{S}_n}{\partial \theta} + \frac{\partial a(\hat{S}_n, t_n)}{\partial \theta}h + \frac{\partial b(\hat{S}_n, t_n)}{\partial \theta}\Delta W_{n+1} \tag{2.14}$$

$$\frac{\partial \hat{S}_{n+1}}{\partial \theta} = \frac{\partial \hat{S}_n}{\partial \theta} + \frac{\partial a(\hat{S}_n, t_n)}{\partial \hat{S}}\frac{\partial \hat{S}_n}{\partial \theta}h + \frac{\partial b(\hat{S}_n, t_n)}{\partial \hat{S}}\frac{\partial \hat{S}_n}{\partial \theta}\Delta W_{n+1} \tag{2.15}$$

In the same way we can (assuming additional conditions on the payoff, as discussed in [1], page 393) compute higher order sensitivities. Differentiating twice, we get

$$\frac{\partial^2 V}{\partial \theta^2} = \mathbb{E}\left[\frac{\partial^2 f}{\partial S^2}(\frac{\partial S_T}{\partial \theta})^2 + \frac{\partial f}{\partial S}\frac{\partial^2 S_T}{\delta \theta^2}\right] \tag{2.16}$$

In the same way as for $\frac{\partial S_T}{\delta \theta}$, we will compute $\frac{\partial^2 S_T}{\delta \theta^2}$ using the general formula

$$\frac{\partial^2 \hat{S}_{n+1}}{\partial \theta^2} = \frac{\partial^2 \hat{S}_n}{\partial \theta^2} + \frac{\partial^2 a(\hat{S}_n, t_n)h}{\partial \theta^2} + \frac{\partial^2 b(\hat{S}_n, t_n)\Delta W_{n+1}}{\partial \theta^2} \tag{2.17}$$

This formula also simplifies in the same cases where h and ΔW_{n+1} do not depend on Θ.

This also easily extends to path simulations and path dependent payoffs: with the same notations as before, we get

$$\frac{\partial V}{\partial \theta} = \mathbb{E}\left[\frac{\partial f}{\partial \hat{S}}\frac{\partial \hat{S}_T}{\partial \theta}\right] \tag{2.18}$$

Unlike the Likelihood Ratio method, the Pathwise Sensitivity approach extends well to path simulations as the variance does not blow up with the number of time steps.

It clearly appears that this method also requires a differentiable payoff. Failure to meet this condition leads to an explosion of the estimator's variance.

The "obvious" way of getting around this restriction is to smooth the payoff function. Smoothing can actually become a complex problem though, especially when we have to deal with multidimensional problems and when we need differentiability with respect to several variables (or when we need higher orders of differentiability with respect to one variable).

2.2 Adjoint Implementation

Pathwise Sensitivity's path simulations can be quite costly in multidimensional cases, i.e. precisely where Monte Carlo is most useful (e.g. sensitivities of a fixed-income derivatives book with respect to points along the forward curve ...).

If we take equation (2.1) in an m-dimensional setting and consider its Euler discretization (2.2), we can write for a family of $(F_n) : \mathbb{R}^m \to \mathbb{R}^m$ that $S_{n+1} = F_n(S_n)$.

Pathwise Sensitivity calculations are done as usual; take here for example the delta with respect to the j^{th} variable.

$$\frac{\partial}{\partial S_j(0)}\mathbb{E}\left[f(\hat{S}_T)\right] = \mathbb{E}\left[\frac{\partial}{\partial S_j(0)}f(\hat{S}_T)\right] = \mathbb{E}\left[\sum_{i=1}^{m}\left(\frac{\partial f(\hat{S}_T)}{\partial \hat{S}_i(T)}\frac{\partial \hat{S}_i(T)}{\partial \hat{S}_j(0)}\right)\right] \tag{2.19}$$

Defining $\Delta_{ij}(n) = \frac{\partial \hat{S}_i(n)}{\partial \hat{S}_j(0)} \forall (i,j) \in [1,\dots,m]$, we need to compute

$$\mathbb{E}\left[\sum_{i=1}^{m}\left(\frac{\partial f(\hat{S}_T)}{\partial \hat{S}_i(T)}\Delta_{ij}(n)\right)\right] \tag{2.20}$$

We get $\Delta_{ij}(N)$ by differentiating (2.2) and using recursion.
Using $m \times m$ matrices D_n whose elements are

$$D_{ik}(n) = \delta_{ik} + \frac{\partial a_i}{\partial S_k}h + \sum_{l=1}^{d}\frac{\partial b_{il}}{\partial S_k}(\Delta W)_l(n+1) \tag{2.21}$$

We can write

$$\begin{cases} \Delta_{n+1} = D_n \Delta_n \\ \Delta_0 = I_{m\times m} \end{cases} \tag{2.22}$$

We hence have a $m \times m$ recursion which can be very costly for high dimensional problems.

We can get around this with the so-called adjoint implementation presented by Giles and Glasserman ([4], pages 88-92). Using our previous results, we get:

$$\begin{aligned} \frac{\partial f}{\partial S_0} &= \frac{\partial f}{\partial S_N}\Delta_N && (2.23)\\ &= \frac{\partial f}{\partial S_N}D_{N-1}D_{N-2}\dots D_0\Delta_0 \\ &= V_0^T\Delta_0 && (2.24) \end{aligned}$$

The idea is to avoid using (2.23) recursively computing $\Delta_N = D_{N-1}D_{N-2}\dots D_0\Delta_0$ while simulating the path, which involves an $m\times m$ recursion. We will instead proceed backwards and use (2.24): assuming we have stored the $(D_k)_{k\in[0,N-1]}$ during our path simulations, we compute V_0 with this backward recursion:

$$\begin{cases} V_N = \left(\frac{\partial g}{\partial S_N}\right)^T \Delta_n \\ V_n = D_n^T V_{n+1} \end{cases} \tag{2.25}$$

This is a vector recursion. We just have to update m values at each time step instead of m^2. This represents considerable computational savings. We finally get the exact same result as with the standard implementation with a cost diminution by a factor $O(m)$.

The computations of other sensitivities are done in a similar way and also yield a gain by a factor $O(m)$. What's more, these different sensitivities' computations use the same adjoint variables (i.e. the stored variables $D_k \dots$). This means the

simultaneous computation of multiple sensitivities can be done at a very low cost: we just have to perform additional backward recursions (2.25) for each Greek.

We have shown that an adjoint formulation to the Pathwise Sensitivity method can be used to greatly reduce computational cost in highly multidimensional settings. We will see that this approach to Pathwise Sensitivities also extends to Vibrato Monte Carlo (which greatly relies on pathwise sensitivity computations).

2.3 Greeks through Vibrato Monte Carlo

2.3.1 Conditional Expectation

The conditional expectation technique, as expounded by Glasserman in [1], is a manner of getting around payoff discontinuity issues encountered with Pathwise Sensitivity.

We simulate the $(N-1)$ first steps of the path as for a standard Pathwise Sensitivity calculation.

$$\hat{S}_{n+1} = \hat{S}_n + a(\hat{S}_n, t_n)h + b(\hat{S}_n, t_n)\Delta W_{n+1} \tag{2.26}$$

$$\frac{\partial \hat{S}_{n+1}}{\partial \theta} = \frac{\partial \hat{S}_n}{\partial \theta} + \frac{\partial a(\hat{S}_n, t_n)h}{\partial \theta} + \frac{\partial b(\hat{S}_n, t_n)\Delta W_{n+1}}{\partial \theta} \tag{2.27}$$

Instead of simulating the last step, we then consider for every path simulation, i.e. for every fixed set $W = (\Delta W_k)_{k\in[0,N-1]}$ the full distribution of $\left(\hat{S}_N|W\right)$. We can write (with Z a unit normal random variable) in the 1-dimensional case:

$$\hat{S}_N(W,Z) = \hat{S}_{N-1} + a\left(\hat{S}_{N-1}, (N-1)h\right)h + b\left(\hat{S}_{N-1}, (N-1)h\right)\sqrt{(h)}Z \tag{2.28}$$

We hence get a normal distribution for $\hat{S}_N$.

$$p(\hat{S}_N|W) = \frac{1}{\sigma_W\sqrt{2\pi}}\exp\left(-\frac{\left(\hat{S}_N - \mu_W\right)^2}{2\sigma_W^2}\right) \tag{2.29}$$

$$with \begin{cases} \mu_W = \hat{S}_{N-1} + a\left(\hat{S}_{N-1}, (N-1)h\right)h \\ \sigma_W = b\left(\hat{S}_{N-1}, (N-1)h\right)\sqrt{h} \end{cases} \tag{2.30}$$

Given a certain path simulation (m), we can compute this final conditional distribution and hence $\mathbb{E}\left[f\left(\hat{S}_N^{(m)}\right)|\hat{S}_{N-1}^{(m)}\right]$.

By the chain rule, our Monte Carlo estimator to $\mathbb{E}[f(S(T))]$ will be that of $\mathbb{E}_W[\mathbb{E}_Z[f(S_T)|W]]$, i.e. we will compute

$$\hat{V} = \frac{1}{M}\sum_{m=1}^{M} \mathbb{E}\left[f(\hat{S}_N^{(m)})|\hat{S}_{N-1}^{(m)}\right] \tag{2.31}$$

Each of the $\mathbb{E}\left[f(\hat{S}_N^{(m)})|\hat{S}_{N-1}^{(m)}\right]$ is differentiable with respect to the input parameters. Hence we can apply the Pathwise Sensitivity technique to compute the sensitivities, even with discontinuous payoffs.

The main restriction is that in many situations (especially multidimensional cases) the Conditional Expectation method leads to complicated integral computations.

2.3.2 Vibrato Monte Carlo

2.3.2.1 Principle

Building on the conditional expectation technique, Giles proposes the Vibrato Monte Carlo technique which offers a new way of computing Greeks and addresses many of the aforementioned limitations.

We simulate paths up to step $N-1$ and then consider the full distribution of $\left(\hat{S}_N|W\right)$ for fixed sets $W = (\Delta W_k)_{k\in[0,N-1]}$.

With the same notations as above, we can write for any given $W^{(m)}$

$$\hat{S}_N^{(m)} = \mu_W^{(m)} + \sigma_W^{(m)} Z \tag{2.32}$$

$$V = \mathbb{E}_W[\mathbb{E}_Z[f(S_T)|W]] \tag{2.33}$$

$$V = \int\left(\int f(\hat{S}_N) p_S\left(\hat{S}_N|W\right)\,\mathrm{d}\hat{S}_N\right) p_W(W)\,\mathrm{d}W \tag{2.34}$$

A Monte Carlo estimator of V is then

$$\hat{V} = \frac{1}{M}\sum_{m=1}^{M}\left(\mathbb{E}_Z\left[f\left(\hat{S}_N\right)|W^{(m)}\right]\right) \tag{2.35}$$

Using the splitting technique with D samples to estimate $\mathbb{E}_Z\left[f\left(\hat{S}_N\right)|W^{(m)}\right]$ gives

$$\tilde{\mathbb{E}}_Z\left[f\left(\hat{S}_N\right)|W^{(m)}\right] = \frac{1}{D}\sum_{d=1}^{D}\left(f\left(\hat{S}_N^{(d)}\right)\right) \tag{2.36}$$

with

$$\hat{S}_N^{(m,d)} = \mu_W^{(m)} + \sigma_W^{(m)} Z^{(d)} \tag{2.37}$$

We finally get

$$\hat{V} = \frac{1}{M}\sum_{m=1}^{M}\left(\frac{1}{D}\sum_{d=1}^{D}\left(f\left(\hat{S}_N^{(m,d)}\right)\right)\right) \tag{2.38}$$

To compute the sensitivities to an input parameter θ, we apply the Pathwise Sensitivity approach to the simulated paths (i.e. at fixed W) to get $\frac{\partial \mu_W}{\partial \theta}$ and $\frac{\partial \sigma_W}{\partial \theta}$ (Note that we can compute this using the adjoint implementation).

We then use the Likelihood Ratio method to write

$$\frac{\partial V}{\partial \theta} = \mathbb{E}_W\left[\frac{\partial}{\partial \theta}\mathbb{E}_Z\left[f\left(\hat{S}_N\right)|W\right]\right] = \mathbb{E}_W\left[\mathbb{E}_Z\left[f\left(\hat{S}_N\right)\frac{\partial(\log p_S)}{\partial \theta}|W\right]\right] \tag{2.39}$$

With $p_S = p_S\left(\mu_W, \sigma_W\right)$ as in (2.29). Still in a 1-dimensional case

$$\frac{\partial(\log p_S)}{\partial \theta} = \frac{\partial(\log p_S)}{\partial \mu_W}\frac{\partial \mu_W}{\partial \theta} + \frac{\partial(\log p_S)}{\partial \sigma_W}\frac{\partial \sigma_W}{\partial \theta} \tag{2.40}$$

A Monte Carlo estimator of this sensitivity is

$$\frac{\partial \hat{V}}{\partial \theta} = \frac{1}{M}\sum_{m=1}^{M}\left(\mathbb{E}_Z\left[f\left(\hat{S}_N\right)\frac{\partial(\log p_S)}{\partial \theta}|W^{(m)}\right]\right) \tag{2.41}$$

$$\begin{aligned}\frac{\partial \hat{V}}{\partial \theta} = \frac{1}{M}\sum_{m=1}^{M}\Big(&\frac{\partial \mu_W^{(m)}}{\partial \theta}\mathbb{E}_Z\left[f\left(\hat{S}_N\right)\frac{\partial(\log p_S)}{\partial \mu_W}|W^{(m)}\right] \\ &+ \frac{\partial \sigma_W^{(m)}}{\partial \theta}\mathbb{E}_Z\left[f\left(\hat{S}_N\right)\frac{\partial(\log p_S)}{\partial \sigma_W}|W^{(m)}\right]\Big)\end{aligned} \tag{2.42}$$

We have

$$\begin{cases} \log p_S = -\log \sigma_W - \frac{1}{2}\log(2\pi) - \frac{\left(\hat{S}_N - \mu_W\right)^2}{2\sigma_W^2} \\ \frac{\partial(\log p_S)}{\partial \mu_W} = \frac{\hat{S}_N - \mu_W}{\sigma_W^2} \\ \frac{\partial(\log p_S)}{\partial \sigma_W} = \left(-\frac{1}{\sigma_W} + \frac{\left(\hat{S}_N - \mu_W\right)^2}{\sigma_W^3}\right) \end{cases} \tag{2.43}$$

So $\mathbb{E}_Z\left[f\left(\hat{S}_N\right)\frac{\partial(\log p_S)}{\partial \mu_W}|W^{(m)}\right]$ and $\mathbb{E}_Z\left[f\left(\hat{S}_N\right)\frac{\partial(\log p_S)}{\partial \sigma_W}|W^{(m)}\right]$ themselves can be estimated through

$$\begin{cases} \tilde{\mathbb{E}}_Z\left[f\left(\hat{S}_N\right)\frac{\partial(\log p_S)}{\partial \mu_W}|W^{(m)}\right] = \frac{1}{D}\sum_{d=1}^{D}\left(f\left(\hat{S}_N\right)\frac{\hat{S}_N - \mu_W}{\sigma_W^2}|W^{(m)}\right) \\ \tilde{\mathbb{E}}_Z\left[f\left(\hat{S}_N\right)\frac{\partial(\log p_S)}{\partial \sigma_W}|W^{(m)}\right] = \frac{1}{D}\sum_{d=1}^{D}\left(f\left(\hat{S}_N\right)\left(-\frac{1}{\sigma_W} + \frac{\left(\hat{S}_N - \mu_W\right)^2}{\sigma_W^3}\right)|W^{(m)}\right) \end{cases} \tag{2.44}$$

Finally we get a first Monte Carlo estimator of the sensitivity to θ by substituting these estimators (2.44) in (2.42).

2.3.2.2 Variance Reduction

We can also get better estimators for $\mathbb{E}_Z[\cdots]$ through the use of antithetic variables. Due to the symmetry of the distribution of Z, we have

$$\mathbb{E}_Z\left[f\left(\hat{S}_N\right)|\hat{S}_{N-1}\right] = \mathbb{E}_Z\left[f\left(\mu_W + \sigma_W Z\right)\right] = \mathbb{E}_Z\left[f\left(\mu_W - \sigma_W Z\right)\right] \tag{2.45}$$

Hence we can use

$$\mathbb{E}_Z\left[f\left(\hat{S}_N\right)|W^{(m)}\right] = f\left(\mu_W\right) + \mathbb{E}_Z\left[f\left(\mu_W + \sigma_W Z\right) - f\left(\mu_W\right)\right] \tag{2.46}$$

$$= f\left(\mu_W\right) + \mathbb{E}_Z\left[\frac{1}{2}\left(f\left(\mu_W + \sigma_W Z\right) - 2f\left(\mu_W\right) + f\left(\mu_W - \sigma_W Z\right)\right)\right] \tag{2.47}$$

It is a better estimator than the "naive" estimator we presented first. It is especially good when f is smooth: the estimator contains the expectation of a quantity with small mean and variance, a single sample would be sufficient in that case.

In the same way

$$\begin{aligned}\mathbb{E}_Z\left[f\left(\hat{S}_N\right)\frac{\partial(\log p_S)}{\partial\mu_W}|W^{(m)}\right] &= \mathbb{E}_Z\left[\frac{\hat{S}_N - \mu_W}{\sigma_W^2}f\left(\hat{S}_N\right)\right] \\ &= \mathbb{E}_Z\left[\frac{Z}{\sigma_W}f\left(\mu_W + \sigma_W Z\right)\right] \\ &= \mathbb{E}_Z\left[\frac{Z}{2\sigma_W}\left(f\left(\mu_W + \sigma_W Z\right) - f\left(\mu_W - \sigma_W Z\right)\right)\right]\end{aligned}$$

Similarly, using $\mathbb{E}_Z\left[Z^2 - 1\right] = 0$.

$$\begin{aligned}\mathbb{E}_Z\left[f\left(\hat{S}_N\right)\frac{\partial(\log p_S)}{\partial\sigma_W}\right] &= \mathbb{E}_Z\left[\left(-\frac{1}{\sigma_W} + \frac{\left(\hat{S}_N - \mu_W\right)^2}{\sigma_W^3}\right)f\left(\hat{S}_N\right)\right] \\ &= \mathbb{E}_Z\left[\frac{Z^2-1}{\sigma_W}f\left(\mu_W + \sigma_W Z\right)\right] \\ &= \mathbb{E}_Z\left[\frac{Z^2-1}{\sigma_W}\left(f\left(\mu_W + \sigma_W Z\right) - f\left(\mu_W\right)\right)\right] \\ &= \mathbb{E}_Z\left[\frac{Z^2-1}{2\sigma_W}\left(f\left(\mu_W + \sigma_W Z\right) - 2f\left(\mu_W\right) + f\left(\mu_W - \sigma_W Z\right)\right)\right]\end{aligned}$$

With a differentiable f, the above expectation has magnitude $O(1)$ and a single sample could be sufficient. Nevertheless, the computational cost of this part of the

computation being small compared to path simulations, we can use several samples and improve the precision at a very low cost.

When f is not differentiable the above expectation is of a quantity whose magnitude is $O(\sigma_W^{-1}) = O(h^{-1/2})$ for paths arriving close to the strike ($\mu_W + \sigma_W$, μ_W and $\mu_W - \sigma_W$ can arrive on different sides of the strike): We should then use several samples to compute it more efficiently.

2.3.2.3 Optimal number of samples

Let us estimate the optimal number of samples for a given computational cost.

Consider W and Z independent random variables. Define then

$$g(W^{(m)}, Z^{(m,d)}) = \left(f\left(\hat{S}_N^{(m,d)}\right)\right) \tag{2.48}$$

As before we consider the following unbiased estimator for $\mathbb{E}_W\left[\mathbb{E}_Z\left[g(W,Z)|W\right]\right]$

$$\hat{V}(M,D) = \frac{1}{M}\sum_{m=1}^{M}\left(\frac{1}{D}\sum_{d=1}^{D} g(W^{(m)}, Z^{(m,d)})\right) \tag{2.49}$$

Its variance is as explained in [8] and used in [2]:

$$\mathbb{V}\left[\hat{V}(M,D)\right] = \frac{1}{M}\mathbb{V}_W\left[\mathbb{E}_Z\left[g(W,Z)|W\right]\right] + \frac{1}{MD}\mathbb{E}_W\left[\mathbb{V}_Z\left[g(W,Z)|W\right]\right] \tag{2.50}$$

Hence the variance of our estimator is of the form

$$\frac{\nu_1}{M} + \frac{\nu_2}{MD} \tag{2.51}$$

Its computational cost is of the form

$$c_1 M + c_2 M D \tag{2.52}$$

where c_1 corresponds to the cost of simulating the paths and c_2 to the evaluation of the payoff.

For a fixed computational cost, the variance is minimized by minimizing

$$(\nu_1 + \nu_2/D)(c_1 + c_2 D) = \nu_1 c_2 D + \nu_1 c_1 + \nu_2 c_2 + \nu_2 c_1/D \tag{2.53}$$

That is

$$D_{opt} = \sqrt{\frac{\nu_2 c_1}{\nu_1 c_2}} \tag{2.54}$$

This formula confirms our intuitive idea that when the computational cost of payoff evaluation is small in comparison to the cost of simulating the paths, we should take several samples for Z.

More precisely we can get an estimate of the behaviour of D_{opt} as we increase the number N of time steps.

c_1 is the cost of computing a path, this is hence clearly a $O(N)$.

c_2 is the cost of computing a final step and evaluating the final value, this is hence of complexity $O(1)$.

$\nu_2 = \mathbb{E}_W(\mathbb{V}_Z(g)) = \mathbb{E}_W(\mathbb{E}_Z(g^2) - (\mathbb{E}_Z(g))^2)$. Let us consider the canonical case of a digital call. When W leads us "far away" from the strike, $\mathbb{E}_Z(g^2)$ and $(\mathbb{E}_Z(g))^2$ will be roughly either 1 or 0 simulatneously. Hence the only part of W's distribution that matters for the computation of ν_2 is that around the strike, the zone where $\mu_W + \sigma_W Z$ has a reasonable chance of reaching both the in-the-money and out-of-the-money zones. The size of that zone is proportional to the final step's standard deviation (a large final variance means more change to move a lot and cross the strike). This means the terms in the expectation will undergo a form of homothety with the final step's standard deviation. Hence ν_2 is $0(N^{-1/2})$.

$\nu_1 = \mathbb{V}_W(\mathbb{E}_Z(g))$ can be evaluated with a similar reasoning ([2] p9) as $O(N^{-1/2})$ as well.

We finally get that $D_{opt} = O(\sqrt{N})$ which totally confirms our hypothesis that the cost of the final step evaluation is small when compared to that of simulating the paths: typically we will take $N = 100$ which means that taking a number $d \approx 10$ of samples for the final timestep is a reasonable choice.

NB: We could note that in the cas of Lipschitz payoffs, we find $\nu_1 = O(1)$ and $\nu_2 = O(1)$ which means that in any case $D_{opt} = O(\sqrt{N})$.

2.3.2.4 Generalization to the Multidimensional Case

This method generalizes well in the case of a payoff depending on several variables following a multivariate Gaussian distribution conditional on previous path simulations. For example, we can use it with basket options whose value depends on several assets' values at expiry.

Let μ_W be the vector of means, Σ_W be the covariance matrix, C a matrix such that $\Sigma_W = CC^T$ and Z be a vector of uncorrelated unit Normal variables. We then write the vector of final values

$$\hat{S}_N(W, Z) = \mu_W + CZ \tag{2.55}$$

Provided that Σ_W is not singular, we have

$$\begin{cases} \log p_S = -\frac{1}{2}\log|\Sigma_W| - \frac{d}{2}\log(2\pi) - \frac{1}{2}\left(\hat{S}_N - \mu_W\right)^T \Sigma_W^{-1}\left(\hat{S}_N - \mu_W\right) \\ \frac{\partial(\log p_S)}{\partial \mu_W} = \Sigma_W^{-1}\left(\hat{S}_N - \mu_W\right) = C^{-T}Z \\ \frac{\partial(\log p_S)}{\partial \Sigma_W} = -\frac{1}{2}\Sigma_W^{-1} + \frac{1}{2}\Sigma_W^{-1}\left(\hat{S}_N - \mu_W\right)\left(\hat{S}_N - \mu_W\right)^T \Sigma_W^{-1} \end{cases} \tag{2.56}$$

Hence we get for any given W

$$\begin{aligned} \mathbb{E}_Z\left[f\left(\hat{S}_N\right)\frac{\partial(\log p_S)}{\partial\theta}\right] = \left(\frac{\partial\mu_W}{\partial\theta}\right)^T \mathbb{E}_Z\left[f\left(\hat{S}_N\right)\frac{\partial(\log p_S)}{\partial\mu_W}\right] \\ + Trace\left(\frac{\partial\Sigma_W}{\partial\theta}\mathbb{E}_Z\left[f\left(\hat{S}_N\right)\frac{\partial(\log p_S)}{\partial\Sigma_W}\right]\right) \end{aligned} \tag{2.57}$$

And once again we use the following efficient estimators

$$\mathbb{E}_Z\left[f\left(\hat{S}_N\right)\frac{\partial(\log p_S)}{\partial\mu_W}\right] = \mathbb{E}_Z\left[\frac{1}{2}\left(f\left(\mu_W + CZ\right) - f\left(\mu_W - CZ\right)\right)C^{-T}Z\right] \tag{2.58}$$

$$\begin{aligned} &\mathbb{E}_Z\left[f\left(\hat{S}_N\right)\frac{\partial(\log p_S)}{\partial\Sigma_W}\right] = \\ &\quad \mathbb{E}_Z\left[\frac{1}{2}\left(f\left(\mu_W + CZ\right) - 2f\left(\mu_W\right) + f\left(\mu_W - CZ\right)\right)C^{-T}(ZZ^T - I)C^{-1}\right] \end{aligned} \tag{2.59}$$

2.3.2.5 Use with path-dependent options

I also experimented with the use of this multidimensional Vibrato Monte Carlo to price discretely sampled path-dependent options (barrier, lookback...).

We still consider (2.1) as the model for the asset evolution. Let us explain the principle by considering the case of a single intermediate sample (treatment of multiple intermediate samples is totally similar).

The payoff depends on the values of a single asset at the final time T (corresponding to time step N) and at a certain intermediate time T_1 (most closely approximated by time step N_1).

We simulate the paths as before and use Pathwise Sensitivity till time $T_1 - h$. We then "skip" time T_1 using a time step of size $2h$ and continue a path simulation till time $T - h$. $\hat{S}_T$ is then simulated as before using $\hat{S}_N = \mu_W + \sigma_W Z$ with Z a unit normal variable.

Simulation of the value at intermediate time $\hat{S}_{T_1}$ is done in a slightly different way: $\hat{S}_{T_1-h}$ and $\hat{S}_{T_1+h}$ are given by the previous path simulation (W). We cannot simulate

$\hat{S}_{T_1}$ just the same way as the final value. $\hat{S}_{T_1+h}$ adds a constraint. We have to use a brownian bridge.

According to that Brownian Bridge construction, the distribution of $\hat{S}_T$ will be Normal, centered on $\frac{1}{2}\left(\hat{S}_{T_1-h} + \hat{S}_{T_1+h}\right)$ and its variance will be $hb(S,t)^2/2$. This means we can actually simulate $\hat{S}_{T_1}$ in the same way as before, using an independent gaussian variable whose variance and mean is determined by W.

Vibrato Monte Carlo technique can hence be used in about the same manner for options depending on $n-1$ intermediate samples and for options depending on n independent underlying assets.

2.3.3 Allargando Vibrato Monte Carlo

I will now describe a new version of Vibrato Monte Carlo I started to explore. Instead of splitting the time interval into equal time steps, I suggest the use of a wider final time step. I call it Allargando Vibrato Monte Carlo, an allusion to the musical term meaning "getting slower", that is making time subdivisions larger.

The fundamental idea behind Vibrato Monte Carlo is to consider $\mathbb{E}\left[f(\hat{S}_N^{(m)})|\hat{S}_{N-1}^{(m)}\right]$ instead of $f(\hat{S}_N^{(m)})$. This expectation smooths the final value function that we consider in the Pathwise Sensitivity part of the calculations. Payoff is smoothed through diffusion during $[(N-1)h, Nh]$.

We could also consider the expectation of the payoff, conditional on the value at time $(N-e)h$ (for some $e > 1$). This would add more diffusion and smooth the final value function further, hence reducing the variance of the Vibrato Monte Carlo estimator.

As will be shown in the numerical experiments (Chapter 3), increasing the size of the final time step helps get "smoother" estimators and reduce their variance. However this improvement comes at the cost of accuracy. Increasing the size of the final time step introduces an additional discretization error.

A standard constant-timestepping Euler discretization already suffers from a weak error

$$\mathbb{E}(f(S_T)) - \mathbb{E}(f(\hat{S}_N)) = O(h) \tag{2.60}$$

Using a larger final Euler time step $[(N-e)h, Nh]$ $(e > 1)$ alters the evolution of the asset during that interval. Instead of evolving according to (2.1) (e.g. geometric brownian motion...), it will evolve as a simple brownian motion. This introduces an additional error.

Hence choosing an optimal final step size (i.e. choosing the value of e) will depend on a trade-off between variance reduction and bias increase. To quantify this, we will look in Chapter 3 for a step size that minimizes the mean square error defined as

$$MSE = \mathbb{E}((\hat{Y}_V)^2) \tag{2.61}$$

$$= \frac{1}{M}\mathbb{V}(f(\hat{S}_N)) + (\mathbb{E}(f(S_T)) - \mathbb{E}(f(\hat{S}_N)))^2 \tag{2.62}$$

Chapter 3

Simulations

3.1 Theoretical Calculations

In all our simulations, we will consider an underlying asset whose evolution is modelled by a geometric brownian motion.

$$\frac{\mathrm{d}S_t}{S_t} = r\mathrm{d}t + \sigma\mathrm{d}W_t \tag{3.1}$$

This equation can be integrated and we can calculate $S(T)$ explicitly. We hence get:

$$S(T) = S(0)\exp\left((r - \frac{\sigma^2}{2})T + \sigma W(T)\right) \tag{3.2}$$

Then we can also compute the desired sensitivities explicitly.
Let's define:

$$d1 = \frac{\log(S) - \log(K) + (r + 1/2\sigma^2)T}{\sigma\sqrt{T}} \tag{3.3}$$

$$d2 = \frac{\log(S) - \log(K) + (r - 1/2\sigma^2)T}{\sigma\sqrt{T}} \tag{3.4}$$

We are interested in computing the sensitivities of the price to the different input parameters. These are the so-called "Greeks":

- Δ: sensitivity to the underlying's initial price.
- $Vega$: Sensitivity to volatility.
- ρ: Sensitivity to interest rate.
- Θ: Sensitivity to time-to-expiry.

- Γ: Second order sensitivity to the underlying's initial price.

Using $\mathcal{N}$ the normal cumulative distribution function, we get:

$$\begin{cases} V = \exp(-rT)\mathcal{N}(d_2) \\ \Delta = \exp(-rT)\frac{1}{\sigma S\sqrt{2\pi T}}\exp(-\frac{1}{2}d_2^2) \\ Vega = -\exp(-rT)\frac{1}{\sqrt{2\pi}}\exp(-\frac{1}{2}d_2^2)(\frac{d_2}{\sigma}+\sqrt{T}) \\ \rho = \exp(-rT)\left(-T\mathcal{N}(d2)+\frac{\sqrt{T}}{\sigma\sqrt{2\pi}}exp(-\frac{1}{2}d_2^2)\right) \\ \Theta = -rV + rS\Delta + \frac{1}{2}\sigma^2S^2\Gamma \\ \Gamma = -\exp(-rT)\frac{d_1}{\sqrt{2\pi}\sigma^2TS^2}\exp(-\frac{1}{2}d_2^2) \end{cases} \tag{3.5}$$

In all the following numerical experiments, we will take the following values:

$$\begin{cases} S_0 = 50 \\ K = 55 \\ r = 5\% \\ \sigma = 10\% \\ T = 1 \end{cases} \tag{3.6}$$

Function `digital_call.m` computes the option's value and its Greeks analytically. We get:

$$\begin{cases} V = 0.2925 \\ \Delta = 0.0669 \\ Vega = 1.3479 \\ \rho = 3.0513 \\ \Theta = 0.2200 \\ \Gamma = 0.0054 \end{cases} \tag{3.7}$$

M will be the number of path simulations, N will be the number of time steps, d will be the number of final step samples in Vibrato Monte Carlo.

3.2 Objectives - Dealing with Discontinuity

As explained in Chapter 2, classical methods fail with discontinuous payoffs for various reasons: Pathwise Sensitivity cannot deal with those payoffs "as is". Likelyhood Ratio method estimator's variance explodes as we reduce the time steps of our path simulations. These limitations were a motivation for the development of new techniques.

We will first deal with the case of a European digital call in 3.3 and 3.4.

We will see how classical techniques can be used with such a discontinuous payoff. We will then study the use of Vibrato Monte Carlo and of Allargando Vibrato Monte Carlo.

We will then move on to the possible extension of Vibrato Monte Carlo to some types of path-dependent options.

3.3 Classical Ways

3.3.1 Likelihood Ratio Method

As previously explained, the Likelihood Ratio method is not *per se* incompatible with discontinuous payoffs.

In our geometric brownian motion setting, we can compute the value of the digital option and its Greeks using the final probability density function of the underlying asset. In the case of a geometric brownian motion, this density function is lognormal:

$$S(T) = S(0)\exp((r - \frac{\sigma^2}{2})T + \sigma W_T) \tag{3.8}$$

$$p(S_T) = \frac{1}{S\sigma\sqrt{2\pi T}}\exp\left[-\frac{1}{2}\left(\frac{\log(S/S_0) - (r - \frac{\sigma^2}{2})T}{\sigma\sqrt{T}}\right)^2\right] \tag{3.9}$$

$$\log p(S_T) = -\log S - \log\sigma - \frac{1}{2}\log(2\pi) - \frac{1}{2}\log T - \frac{1}{2}\left(\frac{log(S/S_0) - (r - \frac{\sigma^2}{2})T}{\sigma\sqrt{T}}\right)^2 \tag{3.10}$$

We then compute the following score functions

$$\frac{\partial \log p}{\partial S_0} = \frac{1}{S_0\sigma\sqrt{T}}\frac{log(S/S_0) - (r - \frac{\sigma^2}{2})T}{\sigma\sqrt{T}} = \frac{W_T}{S_0\sigma T} \tag{3.11}$$

$$\begin{aligned}\frac{\partial \log p}{\partial \sigma} &= -\frac{1}{\sigma} - \frac{log(S/S_0) - (r - \frac{\sigma^2}{2})T}{\sigma} + \frac{(log(S/S_0) - (r - \frac{\sigma^2}{2})T)^2}{\sigma^3 T} \\ &= \frac{W_T^2 - T}{\sigma T} - W_T\end{aligned} \tag{3.12}$$

$$\frac{\partial \log p}{\partial r} = \frac{log(S/S_0) - (r - \frac{\sigma^2}{2})T}{\sigma\sqrt{T}}\frac{\sqrt{T}}{\sigma} = \frac{W_T}{\sigma} \tag{3.13}$$

$$\begin{aligned}\frac{\partial \log p}{\partial T} &= -\frac{1}{2T} + \frac{log(S/S_0) - (r - \frac{\sigma^2}{2})T}{\sigma\sqrt{T}}\left(\frac{2(r - \frac{\sigma^2}{2})T + log(S/S_0) - (r - \frac{\sigma^2}{2})T}{2\sigma T^{3/2}}\right) \\ &= -\frac{1}{2T} + W_T\frac{r - \frac{\sigma^2}{2}}{T\sigma} + \frac{W_T^2}{2T^2}\end{aligned} \tag{3.14}$$

$$\begin{aligned}\frac{\partial^2 \log p}{\partial T^2} &= \left(\frac{1}{S_0\sigma\sqrt{T}}\frac{log(S/S_0) - (r - \frac{\sigma^2}{2})T}{\sigma\sqrt{T}}\right)^2 - \frac{1 + \log(S/S_0) - (r - \frac{\sigma^2}{2})T}{S_0^2\sigma^2 T} \\ &= \left(\frac{W_T}{S_0\sigma T}\right)^2 - \frac{1}{S_0^2\sigma^2 T} - \frac{W_T}{S_0^2\sigma T}\end{aligned} \tag{3.15}$$

Function `lrm.m` computes the value of the option and its Greeks through the Likelihood Ratio method. It can do so both by considering the final distribution of S_T and by simulating paths.

Using this with $M = 10^5$ simulation and 100 time steps we get

$$\begin{cases} V = 0.2905 \\ \Delta = 0.0650 \\ Vega = 1.2689 \\ \rho = 3.0305 \\ \theta = 0.1984 \\ \Gamma = 0.0096 \end{cases} \tag{3.16}$$

That means the absolute biases are respectively

$$\begin{cases} \delta V = 1.90 \cdot 10^{-3} \\ \delta \Delta = 1.90 \cdot 10^{-3} \\ \delta Vega = 7.90 \cdot 10^{-2} \\ \delta \rho = 2.07 \cdot 10^{-2} \\ \delta \theta = 2.15 \cdot 10^{-2} \\ \delta \Gamma = 4.2 \cdot 10^{-3} \end{cases} \tag{3.17}$$

Standard deviations for each of these values are respectively

$$\begin{cases} \sigma(V) = 1.40 \cdot 10^{-3} \\ \sigma(\Delta) = 3.30 \cdot 10^{-3} \\ \sigma(Vega) = 2.35 \cdot 10^{-1} \\ \sigma(\rho) = 1.68 \cdot 10^{-2} \\ \sigma(\theta) = 2.07 \cdot 10^{-2} \\ \sigma(\Gamma) = 9.4 \cdot 10^{-3} \end{cases} \tag{3.18}$$

This means the true (theoretical) values of the option and its sensitivities lie well within the confidence interval $\left[\hat{V} - 1.96 \cdot \hat{\sigma}, \hat{V} - 1.96 \cdot \hat{\sigma}\right]$ we get through Likelihood Ratio method.

Experimenting with different values of M in figure 3.1 page 23 (using 100 time steps) confirms that variance of individual samples is not reduced through the use of more simulations. The variance of the estimators will hence decrease with the number of samples as $\frac{1}{M}$.

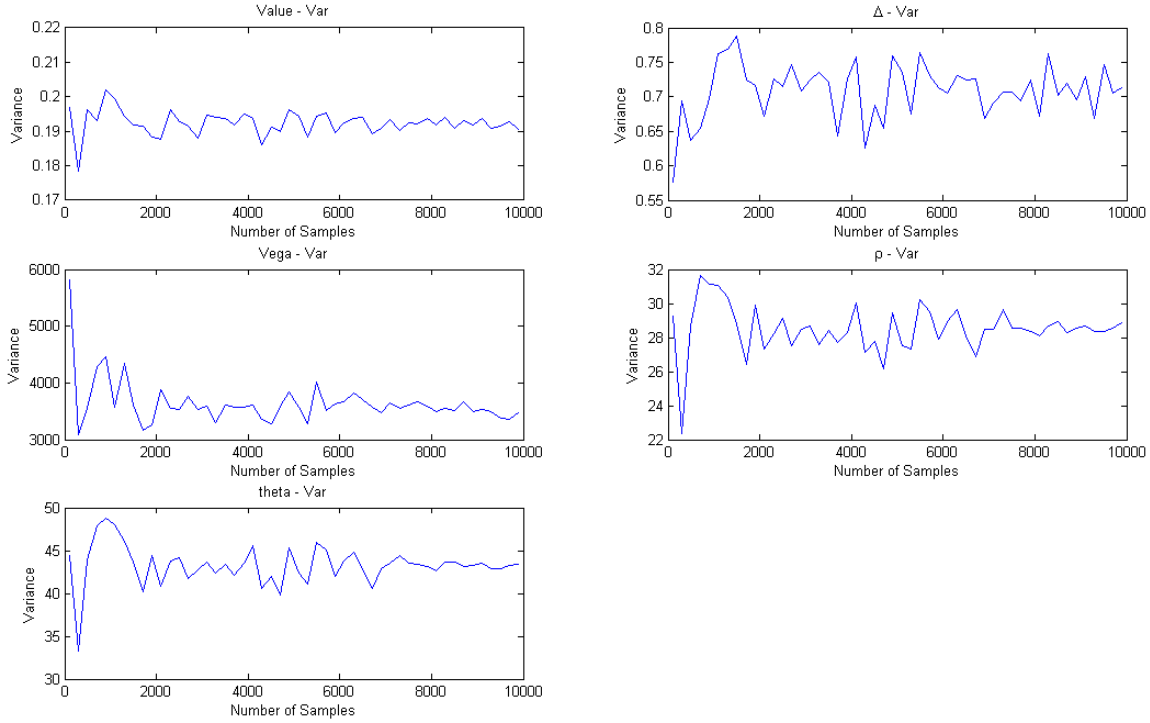


Figure 3.1: LRM - Samples' Variance Evolution with the Number of Simulated Paths

Considering the final distribution of S_T as we did here is only possible in a few particular cases. In general we have to simulate SDE paths using Euler discretization as described in Chapter 2. The problem that arises is that if we try to reduce the time step $h = T/N$ of our simulations, the variance of estimators of Greeks like $Vega$ or Δ will vary as $O(\frac{1}{h})$ and explode.

Figure 3.2 page 24 illustrates this phenomenon: it plots the variance of the individual estimators of $Vega$ for 100000 samples as a function of the number of time steps. Variance of $Vega$'s estimator is directly proportional to that variance (by a factor $\frac{1}{M}$) and hence experiences the same explosion.

The superimposed linear interpolation helps show the linear nature of $\mathbb{V}(N)$.

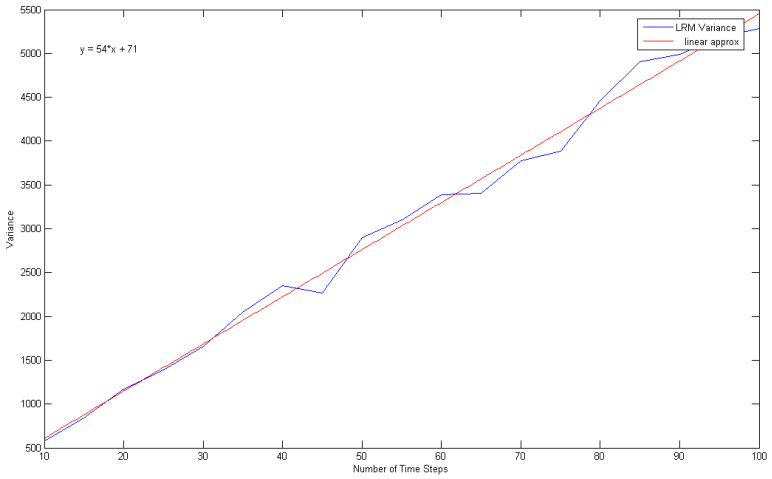


Figure 3.2: LRM - Individual Sample Variance Evolution with the Number of Time Steps

This translates into very high Mean Square Errors for some estimators, as shown in figure 3.3 (this figure uses simulations with $M = 100000$ paths) . These high Mean Square Errors mean poor performances of the estimators as the number of time steps increases: this can be seen through the relative error of the estimators $\frac{|Estimate-Value|}{Value}$ plotted in figure 3.4.

The confidence interval we can get from this method (of width $3.92 \cdot \hat{\sigma}$) gets worse and worse as we refine the path simulations.

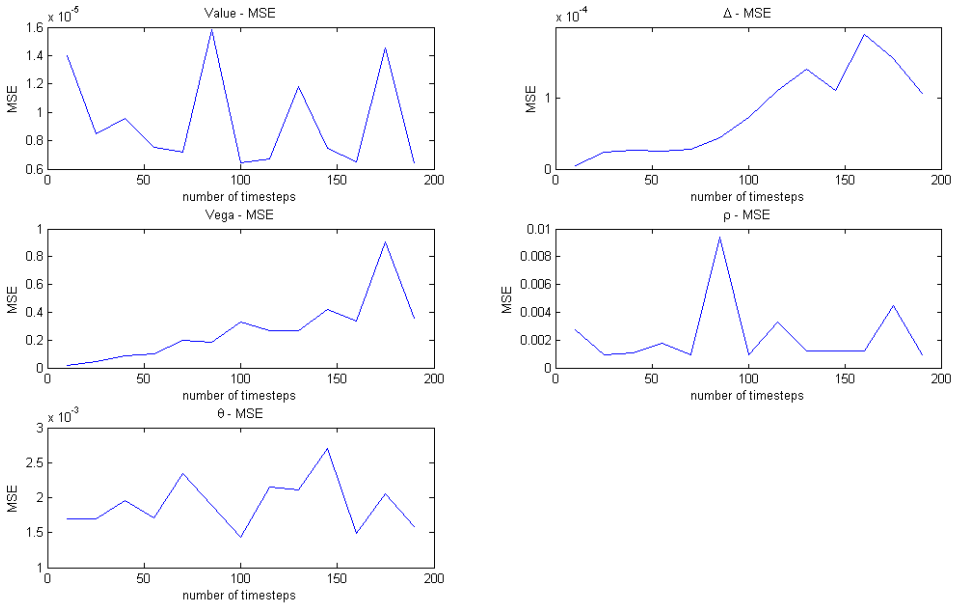


Figure 3.3: LRM - Evolution of MSE with the number of time steps

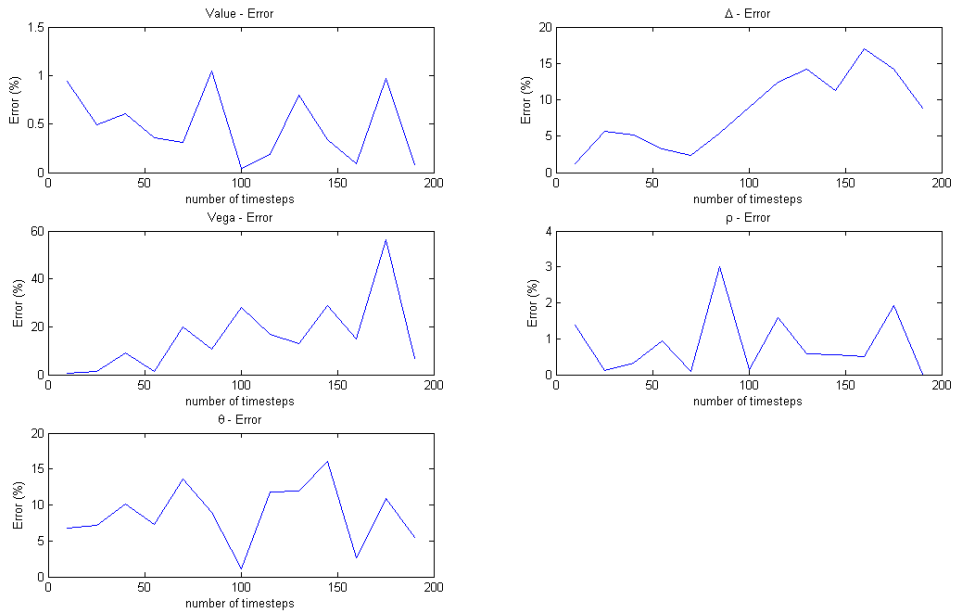


Figure 3.4: LRM - Evolution of the relative error with the number of time steps

3.3.2 Pathwise Sensitivity on Smoothed Payoffs

An "easy way" around the problem of discontinuity with Pathwise Sensitivity is simply to smooth the payoff function. To get first order sensitivities, a "linear by part"-smoothing will be sufficient.

Function `smooth.m` gives such a linearly smoothed version of the digital option payoff $\chi(S_T > K)$. It takes as an input argument a value *range* which indicates the width of the area where the payoff is smoothed. The payoff is 0 on $[0, K - range]$, 1 on $[K + range, \infty]$ and links these two parts by an affine function on $[K - range, K + range]$.

Applying Pathwise Sensitivity to that continuous smoothed payoff is possible. This is done in function `PwS.m`. The wider the smoothing, the smaller the variance of the estimators. However this smoothing makes our estimators biased. A trade-off has to be made between variance reduction and bias. To quantify this, we will plot in figure 3.5 the mean square error of our estimators as a function of *range*. Each simulation is done with 20000 paths.

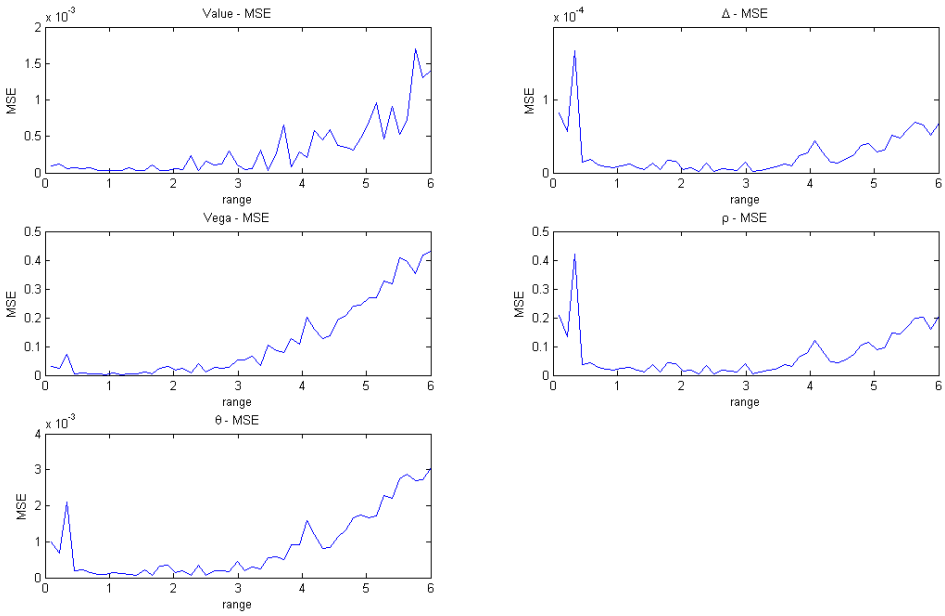


Figure 3.5: PwS - Evolution of MSE with the range of the payoff smoothing

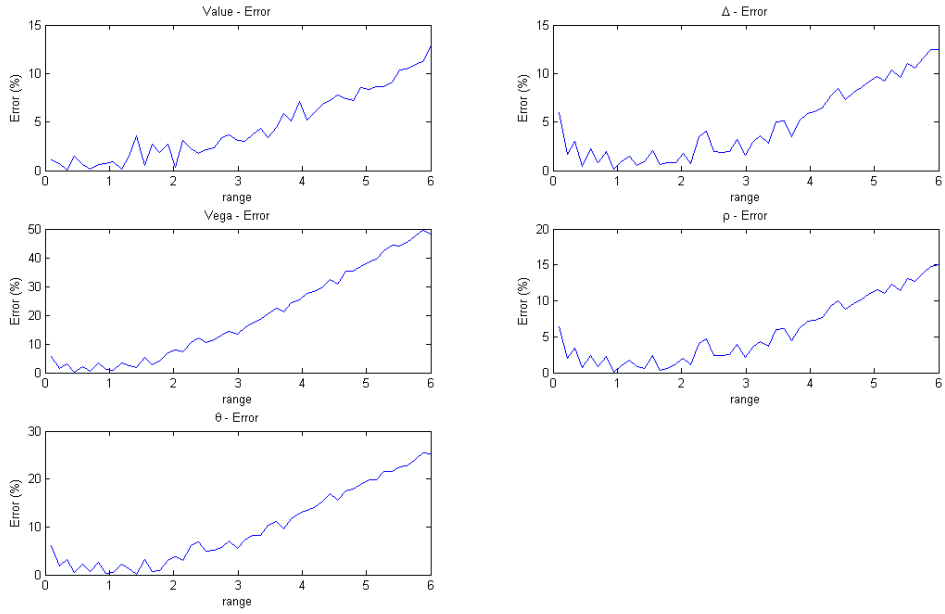


Figure 3.6: PwS - Evolution of the relative error with the range of the payoff smoothing

As expected, the Mean Square Error is reduced by the smoothing of the payoff (Variance tends to ∞ as $range$ tends to 0). We find that the Mean Square Error for Δ, $Vega$, ρ and θ is minimized for $range \in [1, 2]$.

Plotting the bias induced by the smoothing in figure 3.6, we see that for $range \approx 1$, the relative error $\frac{|Estimate - Value|}{Value}$ is acceptable ($\approx 2\%$ for the value, for ρ and δ. It is a bit larger ($\lesssim 5\%$) for Θ or $Vega$, which is not ideal. The idea of "simply" smoothing the payoff fails to give totally satisfactory results.

3.3.3 Adjoint Implementation of Pathwise Sensitivity

I provide a practical implementation of Pathwise Sensitivity in `adjointPwS.m`. This function yields the same results as a "normal" implementation of Pathwise Sensitivity and also suffers from the same drawbacks.

In our particular case, `adjointPwS.m` is actually slower than `PwS.m`. This is because we are simply providing `adjointPwS.m` as an illustration of adjoint implementation: for comprehensibility, we consider a 1-dimensional problem. In this situation, no benefits are to be expected from this implementation (computational savings are of a factor $O(m)$ where m is the dimension). Execution is made slower by the additional storage requirements.

In practice we would only use adjoint implementation in multidimensional cases.

3.3.4 Pathwise-Likelihood Ratio Decomposition

Fournie et al. propose in [5] an improvement of the previous method. They split up the discontinuous payoff as the sum of a smoothed payoff (as above) and a corrective term (a discontinuous function whose support is narrow and centered around the discontinuities of the payoff).

The idea is to apply the Pathwise Sensitivity method to the continuous part of the decomposition and to apply the Likelihood Ratio method to the corrective term.

Summing the estimators of the two parts will give mixed unbiased estimators for the original payoff. This way, a stronger/wider smoothing will not imply an increased bias.

We will simplify a bit their original idea: we evaluate the estimators of the smoothed payoff and that of the corrective term independently. The global variance will then be the sum of their respective variances. We implement this method in `PwS_LRM_hybrid.m` using the same linear smoothing as above. We will then evaluate the variance of the resulting estimator as a function of *range*, that is the width of the smoothing. Figure 3.7 shows the evolution of the Mean Square Error. We plot those figures taking 20000 samples for each simulation.

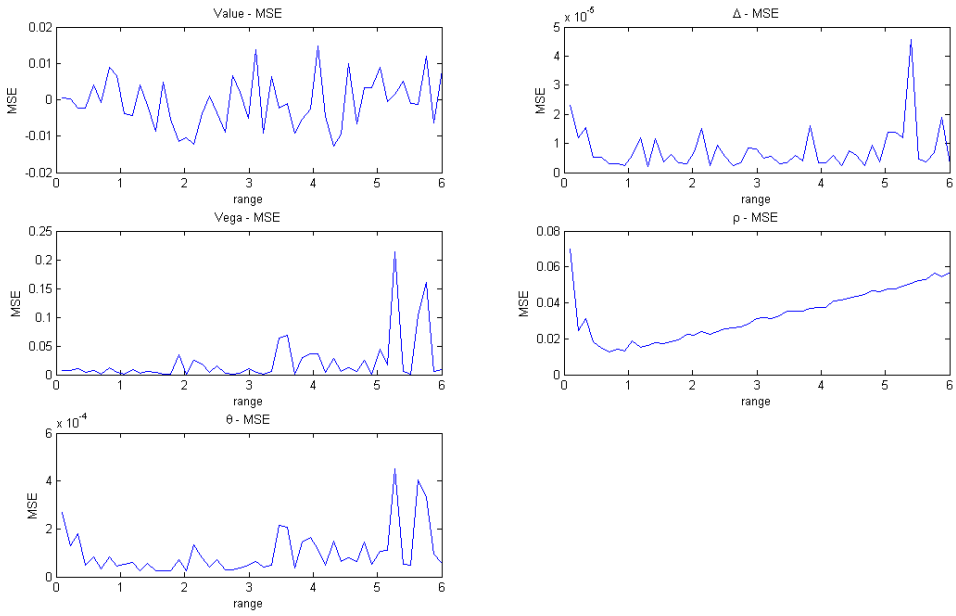


Figure 3.7: PwS-LRM Hybrid - Evolution of MSE with the range of the payoff smoothing

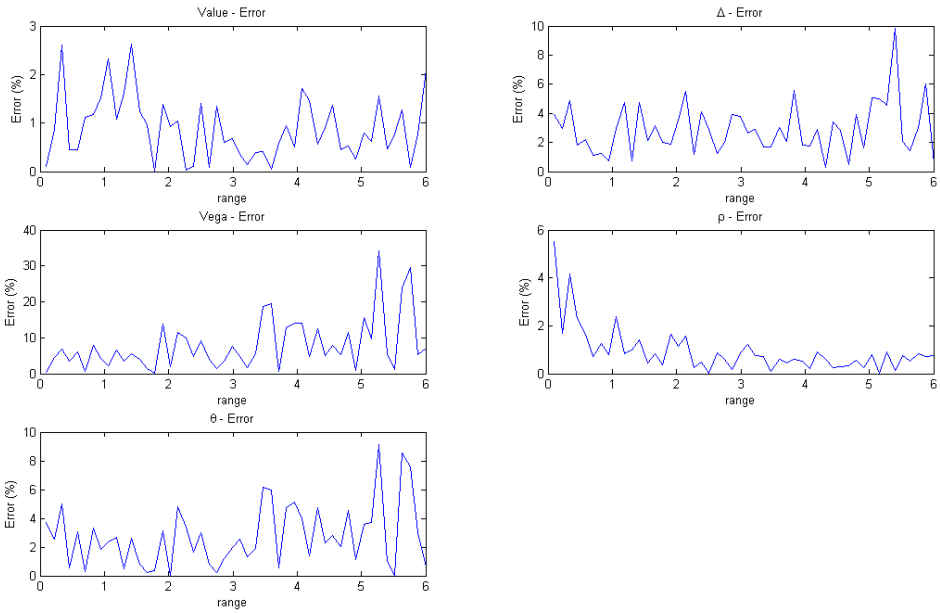


Figure 3.8: PwS-LRM Hybrid - Evolution of the relative error with the range of the payoff smoothing

This method does not bring significant gains compared to a simple Pathwise Sensitivity on a smoothed payoff. This is because the bias introduced by a slightly

smoothed payoff ($range \approx 2$) is quite small. Its removal by the additional Likelihood Ratio term does not balance the additional variance term that appears in the MSE.

As intended this method fixes the problem of bias induced by payoff smoothing: this would be really useful if we used an important smoothing (large values of $range$). Here it is not the case. We could possibly try to improve the implementation and not evaluate the Pathwise Sensitivity and Likelihood Ratio parts independently. This would possibly reduce the variance added by the corrective term; this gain would probably be small and done at the price of an unnecessary complexity.

3.4 Vibrato Monte Carlo

3.4.1 Simple Vibrato Monte Carlo

We use our implementation of Vibrato Monte Carlo found in `VMC.m`.

First we check that Vibrato Monte Carlo does not inherit the drawbacks from the Likelihood Ratio and Pathwise Sensitivity methods.

We already proved in 2.3.2 that Vibrato Monte Carlo could deal with discontinuous payoffs. This is confirmed by the numerical experiments. Taking $M = 10^5$ samples, $N = 100$ time steps and $d = 10$ samples for the final "Likelihood Ratio step", we get the following values:

$$\begin{cases} V = 0.294 \\ \Delta = 0.0665 \\ Vega = 1.341 \\ \rho = 3.026 \\ \theta = 0.218 \end{cases} \tag{3.19}$$

That means the bias is respectively

$$\begin{cases} \delta V = 1.88 \cdot 10^{-3} \\ \delta \Delta = -3.93 \cdot 10^{-4} \\ \delta Vega = -6.63 \cdot 10^{-3} \\ \delta \rho = -2.45 \cdot 10^{-2} \\ \delta \theta = -1.55 \cdot 10^{-3} \end{cases} \tag{3.20}$$

Standard deviation for each of these values is respectively

$$\begin{cases} \sigma(V) = 1.33 \cdot 10^{-3} \\ \sigma(\Delta) = 1.55 \cdot 10^{-3} \\ \sigma(Vega) = 8.71 \cdot 10^{-3} \\ \sigma(\rho) = 7.72 \cdot 10^{-2} \\ \sigma(\theta) = 7.95 \cdot 10^{-3} \end{cases} \tag{3.21}$$

The true theoretical values lie in the (relatively tight) confidence interval

$$\left[\hat{V} - 1.96 \cdot \hat{\sigma}, \hat{V} - 1.96 \cdot \hat{\sigma}\right]$$

.

We will then check that the variances of the Vibrato Monte Carlo estimators do not explode as badly as in the Likelihood Ratio method as we increase the number of time steps. Figure 3.9 page 31 shows the evolution of the variance of the estimator of $Vega$ with the number of time steps N in the case of Vibrato Monte Carlo and in the case of Likelihood Ratio method. We take $M = 100000$.

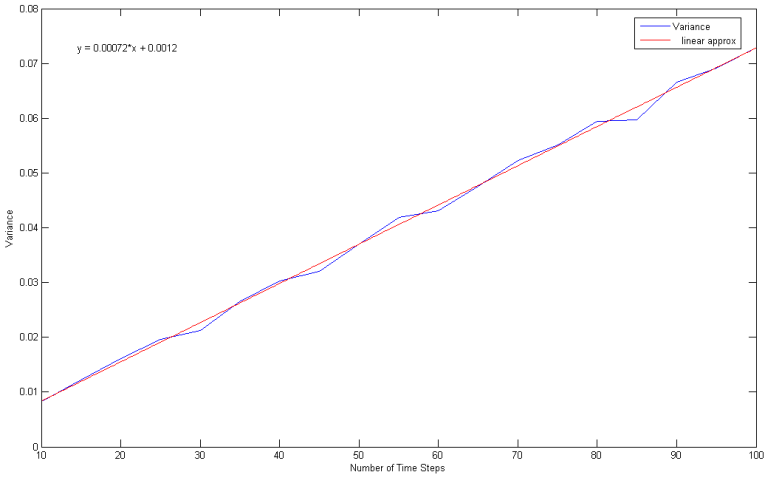


Figure 3.9: VMC - Vega Estimator's Evolution with the Number of Time Steps

We notice that even though the variance of the estimator increases with the number of time steps, this does not increase sufficiently to be as concerning as in the Likelihood Ratio case.

As shown in figure 3.10 page 32 the variance can be reduced through the increase of the number of path simulations. We will take $N = 100$ time steps for our simulations.

We can also study the importance of the number of final samples for Z. Figure 3.11 page 33 plots the evolution of the variance of our estimators as a function of d, number of samples taken for the final Likelihood Ratio method step while keeping the number of time steps and samples constant.

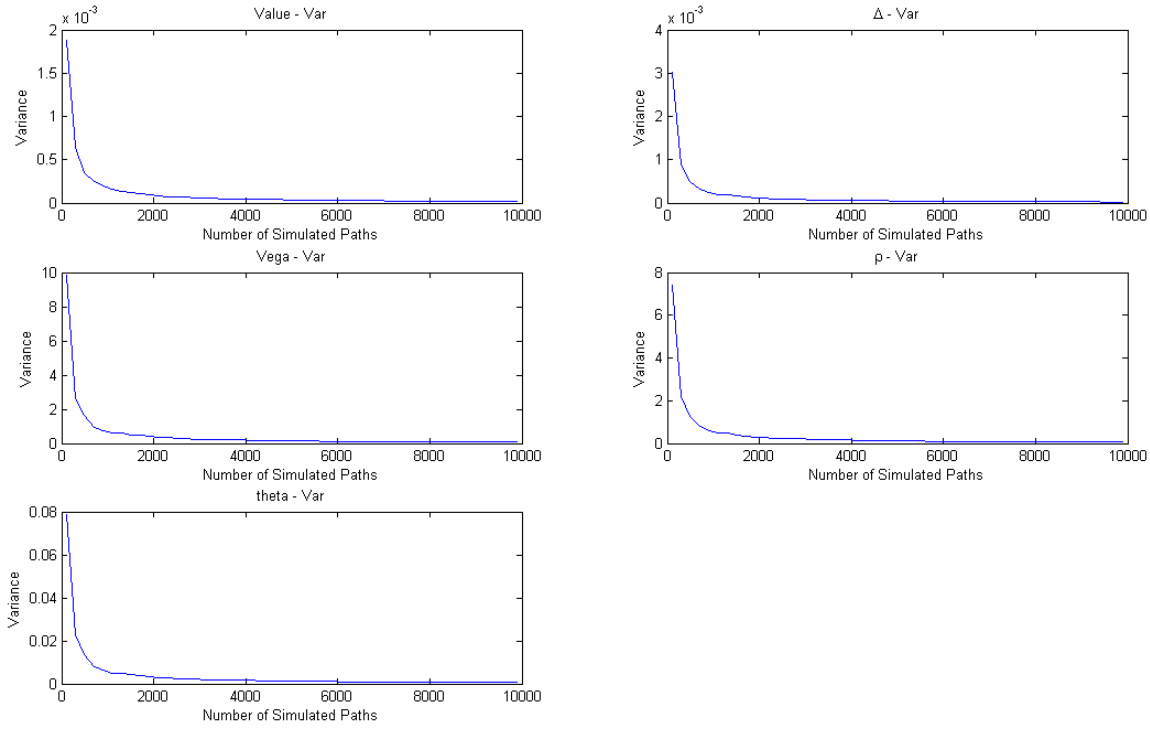


Figure 3.10: VMC - Variance Evolution with the Number of Simulated Paths

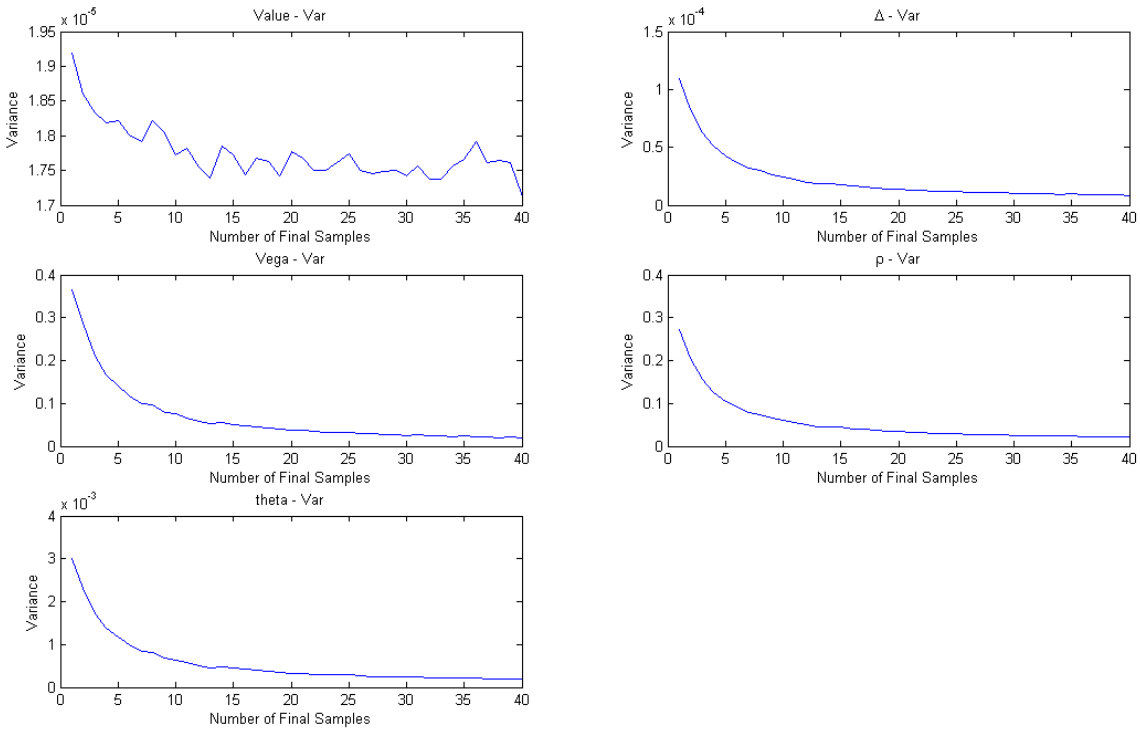


Figure 3.11: VMC - Estimators' Variance Evolution with the number of final step samples

We see that we get significant variance reduction through the use of several final samples. This can be done at a low computational cost ($O(M \cdot d)$ i.e. $O(M \cdot \sqrt{(}N)$ as explained in 2.3.2.3) when compared to the path simulation part of the calculations ($O(M \cdot N)$).

Even relatively small values of d yield significant improvements: taking e.g. $d \approx 10$ reduces the variance of the Greeks' estimators by a factor roughly equal to 5 (See also 2.3.2.3 for more details about the choice of the optimal d).

3.4.2 Use of Antithetic Variables

We can also prove the efficiency of antithetic variables in the Vibrato Monte Carlo estimators. Function `VMC.m` allows computations using both "normal" and "antithetic" estimators. Figure 3.12 page 34 compares the variance of those for different values of d.

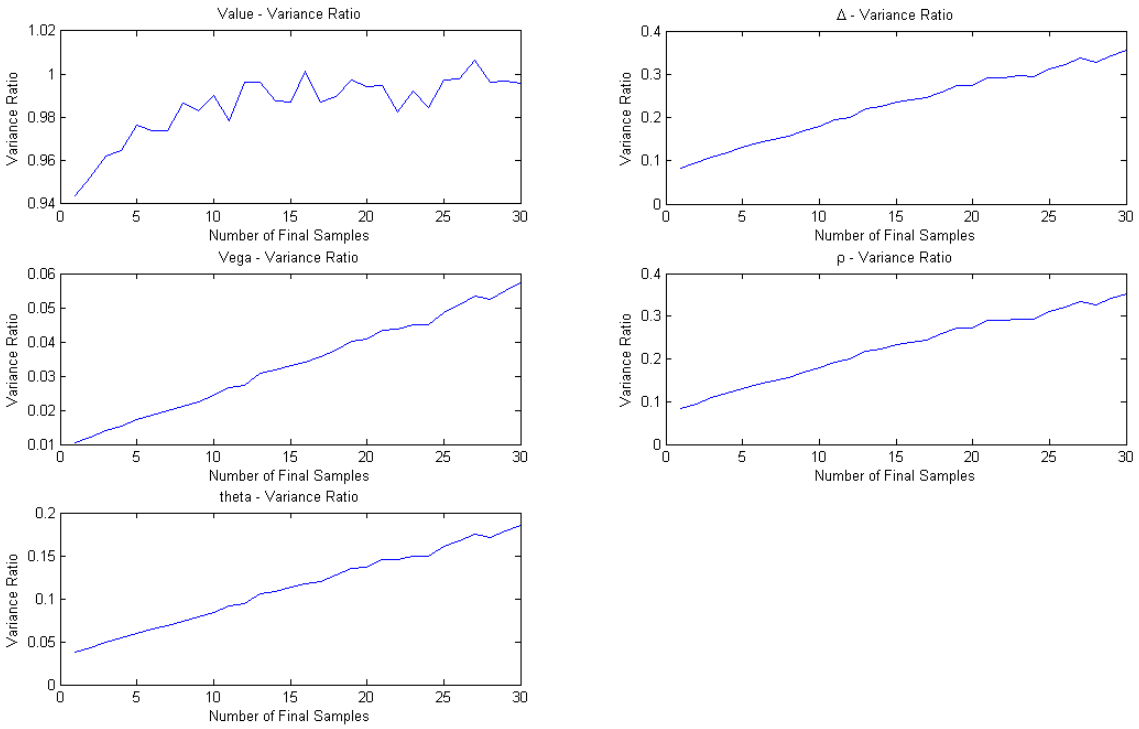


Figure 3.12: VMC - Antithetic vs Normal Estimator - Ratio of the Variances, Evolution with the number of final step samples

This plot shows the superiority of antithetic estimators (we always get a variance ratio smaller than 1). This improvement being made at virtually no computational cost, we will from now on only use the antithetic version of the Vibrato Monte Carlo.

3.5 Allargando Vibrato Monte Carlo

3.5.1 Allargando Vibrato Monte Carlo

As explained in 2.3.3, I will modify the classical Vibrato Monte Carlo technique by taking an irregular time splitting. I expect benefits from taking a wider final time step: taking a wider final step will ensure more diffusion, which means a better smoothing of the payoff, hence better estimators.

We use our implementation found in `VMC_mult.m`.

In figure 3.13 page 35 we plot the variance of the estimators as a function of the size of the final time step $h^* = e \cdot h$.

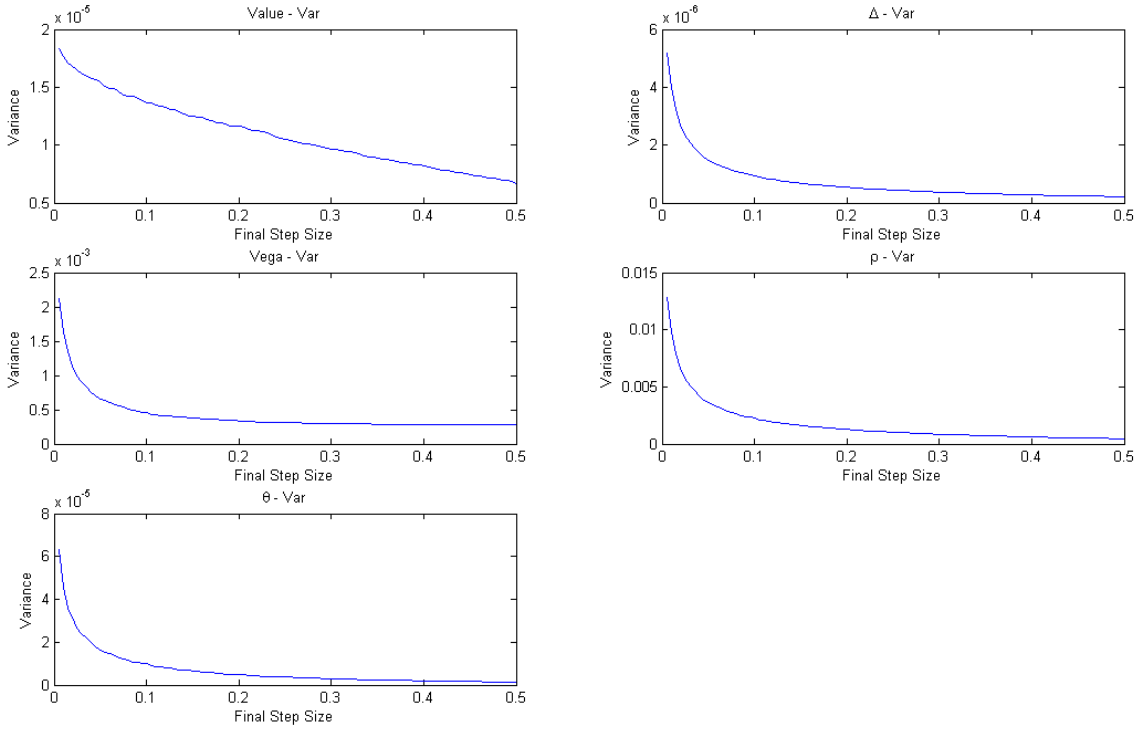


Figure 3.13: AVMC - Evolution of Variance with Final Step's Width

As expected we notice a dramatic variance reduction as h^* increases. This is the result of the additional diffusion on the wider final steps.

Then in figure 3.14 page 36 we take 200 time steps and 10 000 samples to plot the evolution of the bias with the size of the final time step $h^* = e \cdot h$.

We can actually show that variance evolves as $\frac{1}{\sqrt{h^*}}$. Indeed the final step $\hat{S}_N^{(m,d)} = \mu_W^{(m)} + \sigma_W^{(m)} Z^{(d)}$ (with $Z^{(d)}$ a unit normal random variable) is a brownian motion over a time h^*, hence we have $\sigma_W^{(m)} = 0(\sqrt{h^*})$. As in 2.3.2.2 this means that $\mathbb{V} = O((\sigma_W^{(m)})^{-1}) = O((h^*)^{-1/2})$.

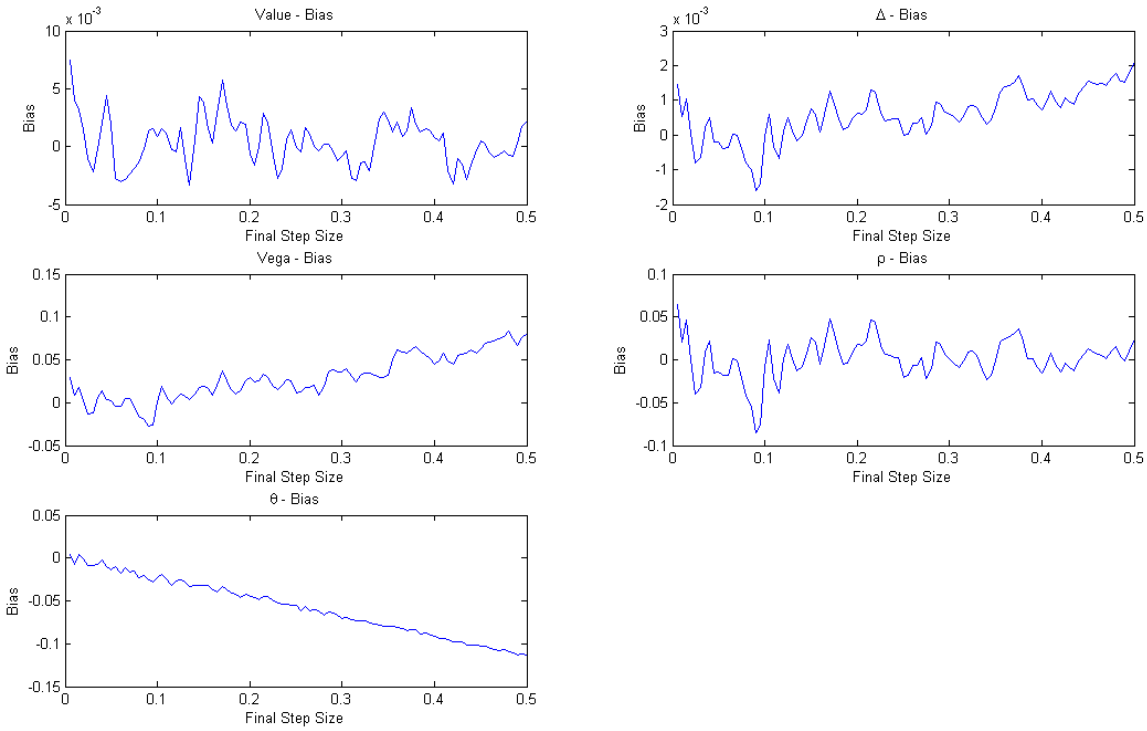


Figure 3.14: AVMC - Evolution of Bias with Final Step's Width

To evaluate the optimal size of the final time step, we will take an h^* that minimizes the Mean Square Error of the estimator. We plot in figure 3.15 the evolution of $MSE(h^*)$.

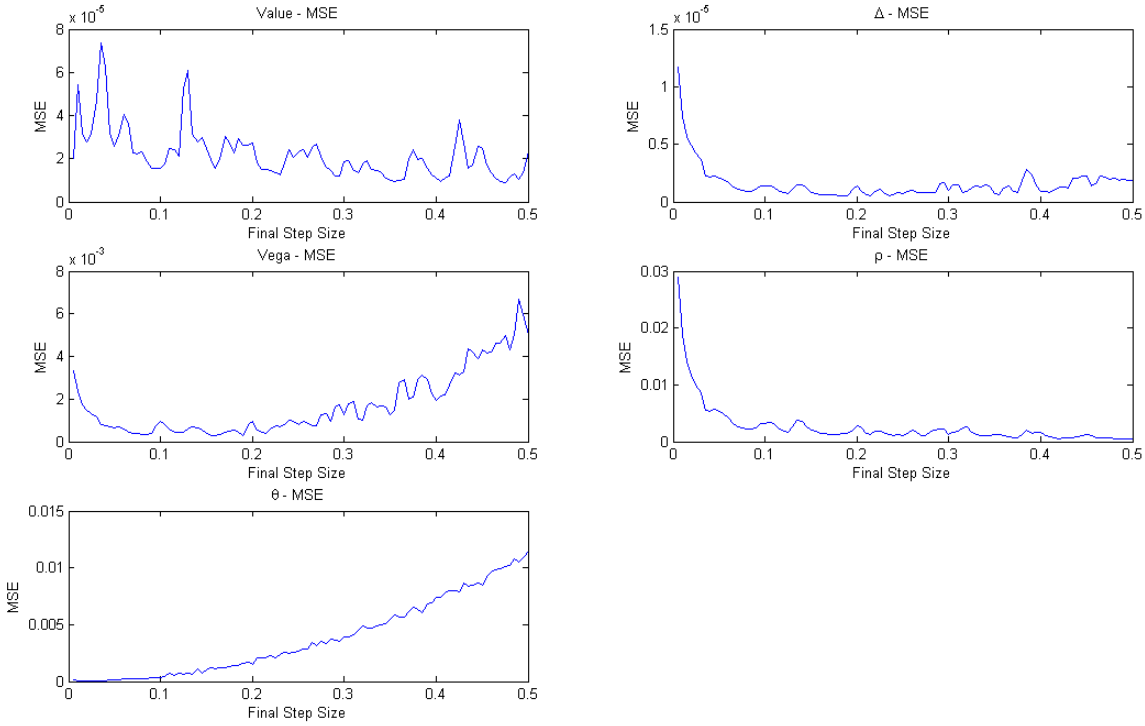


Figure 3.15: AVMC - Evolution of MSE with Final Step's Width

The experiments show that we get significant benefits from this modification of Vibrato Monte Carlo.

Mean Square Error of Δ, $Vega$ and ρ are reduced by about a factor 10 for $h^* \approx 0.2$.

As shown by our experiments, not all estimators behave the same way: whereas Δ, $Vega$, ρ estimators get significant improvements, $Value$ is not really affected by the choice of h^* and the quality of θ estimator is constantly weakened as h^* increases.

If we want to evaluate the option's value and all first order sensitivities, we have to make a trade-off.

NB: Increasing the final time step's size also constantly debases the $Vega$ estimator if the volatility gets above a certain level (around 17%).

We have to define what is an acceptable "loss" on estimators like θ (and $Vega$ in high-volatility settings). Indeed we must decide which h^* gives best improvements for Δ, $Vega$, ρ when compared to "standard" Vibrato Monte Carlo while retaining a reasonable deterioration of other estimators.

I will thus compare the values of the bias induced by the introduction of a larger h^* and the value of the estimator itself.

For example, we get significant benefits for a final time step h^* of size 0.05 (that

is $h^* = 10 \cdot h$. The Mean Square Errors of Δ, $Vega$, ρ are divided by a factor approximatively equal to 5. For that same value of h^*, the relative bias of the θ estimator is about 5%.

We may also get an additional "bonus" as to the computational cost of the method if we are not interested in computing all sensitivities (more precisely if we do not care about ill-behaved sensitivities). If we just consider well-behaved sensitivities (like Δ, $Vega$, ρ) we can take a large final time step of size $h^* = 0.3$ without losing accuracy. This means we can use shorter path simulations. These will be in our case 30% shorter, which approximately translates (with the reasonable assumption that the path calculations are the most costly part of the calculations) into a direct computational cost gain by the same amount.

3.5.2 Variable Final Step Size

I propose the following way of bypassing the issues related to "well-behaved"/ "ill-behaved" sensitivities:

We could perform path simulations from time 0 to time $(N-1)h$ as in a regular Vibrato Monte Carlo. We would then use Allargando Vibrato Monte Carlo with different values of h^* for different sensitivities (e.g. $h^*_\theta = h$ and larger values for well-behaved Greeks: h^*_δ, h^*_ρ etc.). We would need to store the pathwise sensitivities at the different times $T - h^*_{sensitivity}$ and perform independent Likelihood Ratio computations for each of these times. This would result in increased computational costs related to final Likelihood Ratio computations and would need further investigation to evaluate the practical benefits of this method.

3.6 Path Dependent Discontinuous Payoffs

3.6.1 Adapting the Vibrato Monte Carlo Idea

As explained in 2.3.2.5, we can adapt the idea of multidimensional Vibrato Monte Carlo to the case of some path dependent options. In the case of payoffs depending on values at discrete intermediate times τ_i, we perform a multidimensional Vibrato Monte Carlo using Brownian Bridges at the intermediate times.

We will subsequently consider discontinuous payoffs depending on discrete intermediate values. On this type of payoffs the same benefits as before (i.e. variance reduction as in 1-dimensional case etc.) can be expected from Vibrato Monte Carlo.

3.6.2 Barrier Option

To illustrate the use of Vibrato Monte Carlo with such options, we will consider a discretely sampled barrier option: it is a digital European call at strike K (still $K = 55$ in our numerical experiments) and maturity T ($T = 1$ year), we add an up-and-out barrier B ($B = 60$) sampled at final time T and at an intermediate time $T1$ ($T1 = 0.7$ years). The payoff function is implemented in `payoff_barrier.m`.

Implementation of Vibrato Monte Carlo method with this option is found in `VMC_barr.m`. Results of this extension of Vibrato Monte Carlo to the discretely sampled barrier option are shown in the following figures (we kept $N = 100$ time steps).

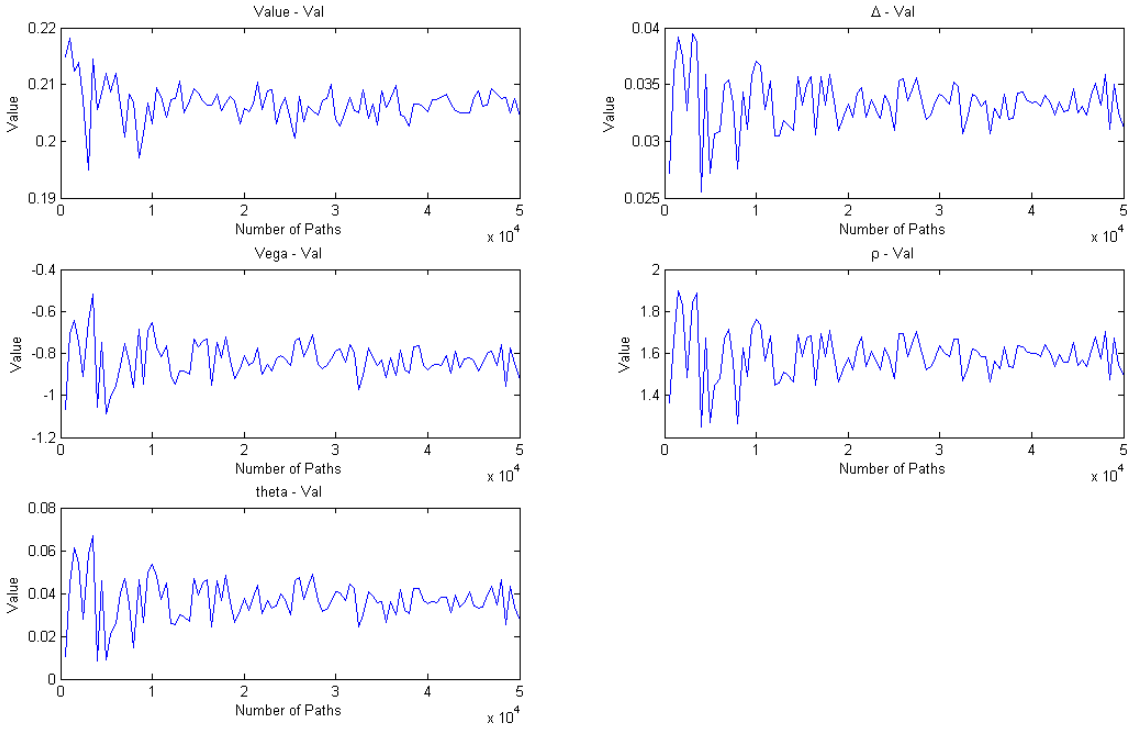


Figure 3.16: AVMC - Evolution of the value's and Greeks' estimators' values with the number of simulated paths

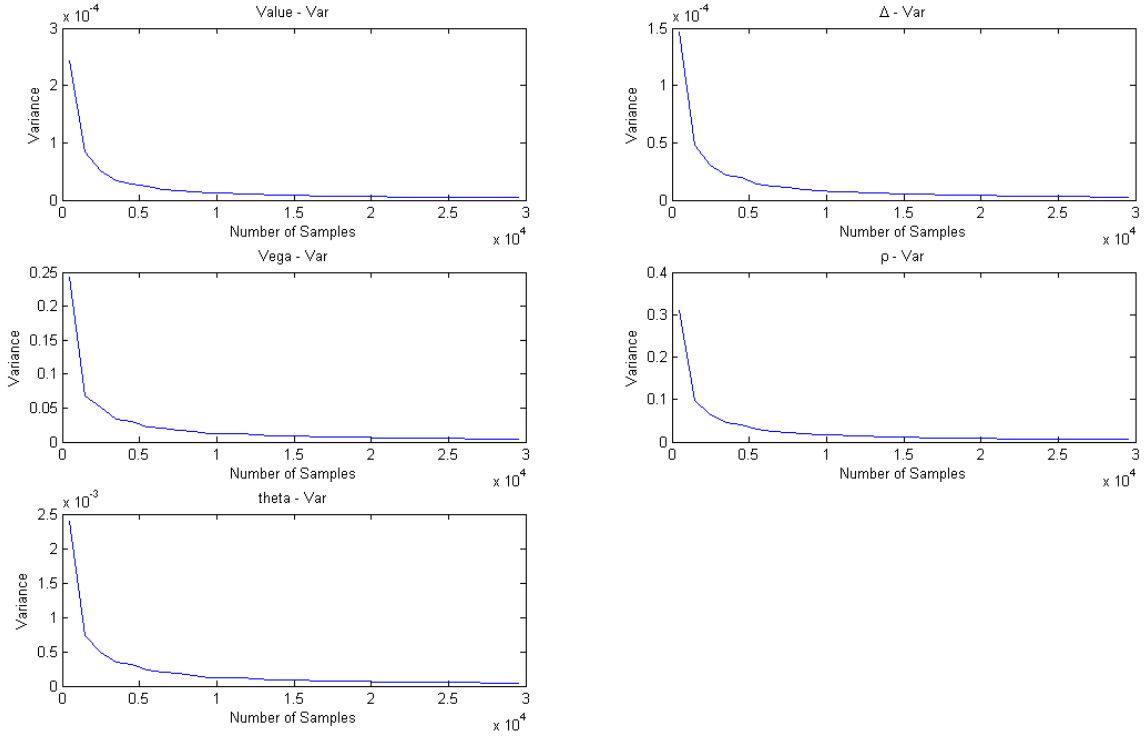


Figure 3.17: AVMC - Evolution of the estimators' variance with the number of simulated paths

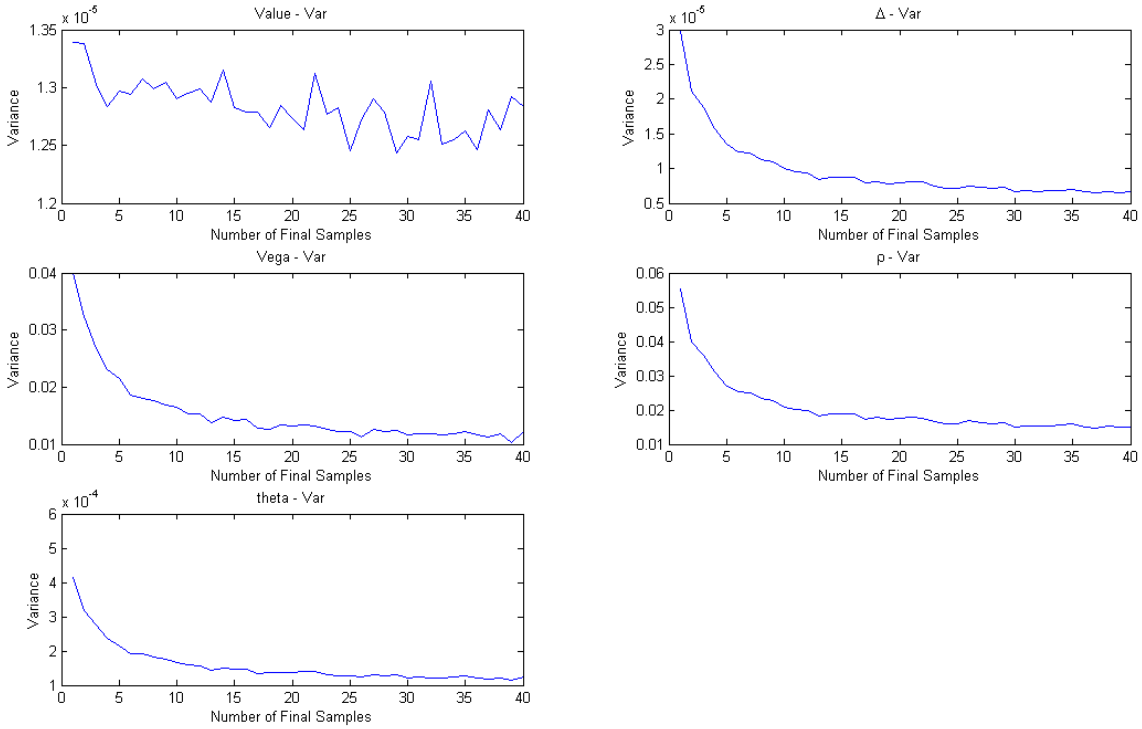


Figure 3.18: AVMC - Reduction of the estimators' variance with the number of samples for the last step

Allargando Vibrato Monte Carlo technique could also be used in this setting. We could take larger jumps of width $2h_i^*$ around the intermediate times. No significant theoretical difference would occur. We would have to pay attention to an additional constraint on the width of those jumps: we should avoid "overlapping jumps", i.e. we should ensure that we have $\forall(i,j), |\tau_j - \tau_i| > (h_i^* + h_j^*)$ to maintain independence for Z_i and Z_j.

Chapter 4

Concluding Remarks

In this paper we have looked at different ways to compute options' prices and their sensitivities in the case of discontinuous payoffs.

We expounded the limits of traditional methods and their spin-offs. Then, building on Glasserman's idea of conditional expectation, we introduced Giles' idea of Vibrato Monte Carlo calculations. This technique is a hybrid that uses pathwise computations for path simulations and then uses Likelihood Ratio method for final payoff evaluation. It combines the advantages of those two techniques: efficiency of pathwise calculations and generality of Likelihood Ratio method. In particular it deals efficiently with discontinuous payoffs.

We then developed an evolution of Vibrato Monte Carlo, which we christened Allargando Vibrato Monte Carlo: instead of constant time subdivisions for all of the calculations, we take small constant time steps for the pathwise part and then take a larger value for the final Likelihood Ratio-based step to improve payoff smoothing. This variant of the Vibrato Monte Carlo idea yields very encouraging results in terms of estimators' variance reduction.

Finally we also showed how we could extend the Vibrato Monte Carlo (and hence its spin-offs like our Allargando Vibrato Monte Carlo) and implement it in the multivariate cases, which is useful both for multi-asset options and certain classes of path-dependent options. What's more the use of adjoint implementation gives huge computational savings in high-dimensional cases.

Although we have only dealt with first-order Greeks here, we could also extend our methods to higher orders sensitivities.

Future research could also be done on getting a better theoretical understanding of Allargando Vibrato Monte Carlo estimators' behaviour; especially we could examine the relation between the optimal final step size and important parameters like the volatility, time-to-expiry etc.

Another point of interest with Allargando Vibrato Monte Carlo would be to investigate the practical benefits we could get from using different final step sizes for every computed sensitivity.

Ultimately, as suggested by Prof. Giles, we plan to study the use of Vibrato Monte Carlo techniques in Multilevel Monte Carlo analysis with the Milstein scheme, which could lead to improved convergence rates with discontinuous payoffs.

Bibliography

[1] P. Glasserman, *Monte Carlo Methods in Financial Engineering*, Springer, 2004

[2] M. Giles, *Vibrato Monte Carlo Sensitivities*, 2009

[3] M. Giles, *Monte Carlo evaluation of sensitivities in computational finance*, 2007

[4] M. Giles, P. Glasserman, *Smoking adjoints: fast Monte Carlo Greeks*, RISK, January 2006

[5] E. Fournie, J.-M. Lasry, J. Lebuchoux, P.-L. Lions and N. Touzi, *Applications of Malliavin calculus to Monte Carlo methods in finance*, Finance and Stochastics 3, p391-412, 1999

[6] P. Jackel, *Monte Carlo Methods in Finance*, John Wiley & Sons, 2002

[7] P. Protter, *Stochastic integration and differential equations*, Springer, 1990

[8] A. Asmussen, P. Glynn *Stochastic Simulation*, Springer, 2007

Code

digital_call.m

```
%
% function V = digital(r,sigma,T,S,K,opt)
%
% Black-Scholes digital european call option
%
% r     - interest rate
% sigma - volatility
% T     - time interval
% S     - asset value(s)
% K     - strike price(s)
% opt   - 'value', 'delta', 'gamma' or 'vega'
% V     - option value(s)
%

function V = digital_call(r,sigma,T,S,K,opt)

if nargin ~= 6
  error('wrong number of arguments');
end

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));

value = exp(-r*T).*N(d2);
delta = exp(-r*T)*(exp(-0.5*d2.^2)./sqrt(2*pi))./(sigma*sqrt(T)*S);
gamma = -exp(-r*T)*d1.*(exp(-0.5*d2.^2)./sqrt(2*pi))./(sigma.^2*T*S.^2);
```

```
vega =  -exp(-r*T)*(exp(-0.5*d2.^2)./sqrt(2*pi)).*(d2/sigma+sqrt(T));
theta= -r*value+r*S*delta+1/2*sigma^2*S^2*gamma;
rho=exp(-r*T).*(-T*N(d2)+(exp(-.5*d2.^2)./sqrt(2*pi))*sqrt(T)/sigma);

switch opt
  case 'value'
    V=value;
  case 'delta'
    V=delta;
  case 'gamma'
    V=gamma;
  case 'vega'
    V=vega;
  case 'theta'
    V=theta;
  case 'rho'
    V=rho;
  otherwise
    error('opt must be ''theta'',''value'', ''delta'', ''gamma'', ''vega''')
end
%
% Normal cumulative distribution function
%
function ncf = N(x)
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);
}
```

lrm.m

```
function [val,valD,valV,valR,valT,valG,var,varD,varV,varR,varT,varG]...
 = lrm(r,sigma,T,S0,K,M,N,opt)
```

```
%opt='finalsim','pathsim'
%finalsim uses the known lognormal distro of a GBM
%pathsim simulates the paths using Euler Maruyama
switch opt
    case 'finalsim'
        W=sqrt(T)*randn(1,M);
        S=S0.*exp((r-sigma^2/2)*T+sigma*W);
        f=payoff(S,K,r,T,'digcall');

        val=sum(f)/M; %value of option
        var=sum(f.^2)/M-val^2;
        var=var/M;

        Ds=f.*W/(S0*sigma*T);
        valD=sum(Ds)/M; %value of delta
        varD=sum(Ds.^2)/M-valD^2;
        varD=varD/M;

        Vs=f.*((W.^2-T)/(sigma*T)-W);
        valV=sum(Vs)/M; %value of vega
        varV=sum(Vs.^2)/M-valV^2;
        varV=varV/M;

        Rs=f.*(W/(sigma));
        valR=sum(Rs-T*exp(-r*T)*f)/M; %value of rho
        varR=sum((Rs-T*exp(-r*T)*f).^2)/M-valR^2;
        varR=varR/M;

        Ts=f.*(-1/(2*T)*ones(1,M)+W*((r-sigma^2/2)/(sigma*T))+W.^2/(2*T^2));
        valT=sum(Ts-r*exp(-r*T)*f)/M; %value of theta
        varT=sum((Rs-r*exp(-r*T)*f).^2)/M-valT^2;
        varT=varT/M;

        Gs=f.*((W/(S0*sigma*T)).^2+(-1/(S0^2*sigma^2*T)-W./(S0^2*sigma*T)));
        valG=sum(Gs)/M; %value of gamma
        varG=sum(Gs.^2)/M-valG^2;
```

```
    varG=varG/M;

case 'pathsim'
    h=T/N; %timestep
    dWs= [];
    S=S0*ones(1,M);
    for n=1:1:N % till final step, is normal euler simulation
        dW=sqrt(h)*randn(1,M);
        dWs=[dWs;dW]; % n*M matrix containing in each column ...
                      %the random sqrt(h)*Z of 1 of the M paths
        S=S.*(1+r*h+sigma*dW); % value evol
    end

    f=payoff(S,K,r,T,'digcall');

    val=sum(f)/M; %value of option
    var=sum(f.^2)/M-val^2;
    var=var/M;

    Ds=f.*dWs(1,:)/(S0*sigma*h);
    valD=sum(Ds)/M; %value of delta
    varD=sum(Ds.^2)/M-valD^2;
    varD=varD/M;

    Vs=f.*sum((dWs.^2-h)/(sigma*h));
    valV=sum(Vs)/M; %value of vega
    varV=sum(Vs.^2)/M-valV^2;
    varV=varV/M;

    Rs=f.*sum((dWs/(sigma)));
    valR=sum(Rs-T*exp(-r*T)*f)/M; %value of rho
    varR=sum((Rs-T*exp(-r*T)*f).^2)/M-valR^2;
    varR=varR/M;

    Ts=f.*sum((-1/(2*h)*ones(N,M)+...
     dWs.*((r-sigma^2/2)/(sigma*h))+dWs.^2./(2*h^2)))/N;
```

```
        valT=sum(Ts-r*exp(-r*T)*f)/M; %value of theta
        varT=sum((Rs-r*exp(-r*T)*f).^2)/M-valT^2;
        varT=varT/M;

        Gs=f.*( (dWs(1,:)/(S0*sigma*h)).^2+...
              (-1/(S0^2*sigma^2*h)-dWs(1,:)./(S0^2*sigma*h)));
        valG=sum(Gs)/M; %value of gamma
        varG=sum(Gs.^2)/M-valG^2;
        varG=varG/M;

end
```

smooth.m

```
function [V,W] = smooth( S,K,r,T,range)
% takes as argt value S (which may be an array) of underlying
% outputs discounted smoothened payoff of digital
%call with smoothening on length 2*range
% around strike K (with linear smoothening)
% also outputs dV/dS
% DEPRECATED and X=dfdr=-T*V=term added by differentiation of r in rho

if nargin ~= 5
    error('wrong number of args')
end

%value 1 with all values which are above K+range
Chi_up=cast(S>K+range,'double');
%value 1 with all values which are below K-range
Chi_lo=cast(S<K-range,'double');
%value 1 on smoothening zone
Chi_mid=1-(Chi_lo+Chi_up);

%above K+range
V=exp(-r*T)*(Chi_up+Chi_mid.*((S-(K-range))/(2*range)));
W=exp(-r*T)*(1/(2*range)*Chi_mid);
%derivative is constant on smoothening zone and zero elsewhere
```

PwS.m

```
function [val,valD,valV,valR,valT,var,varD,varV,varR,varT] ...
= PwS(r,sigma,T,S0,K,M,N,range)

% Implementation of "dumb" PwS
% Adapted to digital call case through smoothing fx
% M - number of samples
% N - number of timesteps
% range=range of smoothening of payoff

if nargin ~= 8
```

```
    error('wrong number of args')
end

h=T/N; %timestep
S=S0*ones(1,M); % array of underlying value
dS=ones(1,M); %array of deltas
vS=zeros(1,M); %array of vegas
rS=zeros(1,M); %array of rhos
tS=zeros(1,M); %array of thetas

for n=1:1:N % till final step, is normal euler simulation
    dW=sqrt(h)*randn(1,M);
    dS=dS.*(1+r*h+sigma*dW); % delta evol
    vS=vS.*(1+r*h+sigma*dW)+S.*dW; % vega evol
    rS=rS.*(1+r*h+sigma*dW)+h*S; % rho evol
    tS=tS.*(1+r*h+sigma*dW)+S.*(r/N+sigma/(2*N*h)*dW); %theta evol
    S=S.*(1+r*h+sigma*dW); % value evol
end

%df/dS
[f,dfdS]=smooth(S,K,r,T,range);

%value
val=1/M*sum(f);
var=1/M*sum(f.^2)-val^2;
var=var/M;

%delta
valD=1/M*sum(dfdS.*dS);
varD=1/M*sum((dfdS.*dS).^2)-valD^2;
varD=varD/M;

%vega
valV=1/M*sum(dfdS.*vS);
varV=1/M*sum((dfdS.*vS).^2)-valV^2;
varV=varV/M;
```

```
%rho
valR=1/M*sum(dfdS.*rS-T*f);
varR=1/M*sum((dfdS.*rS-T*f).^2)-valR^2;
varR=varR/M;

%theta
valT=1/M*sum(dfdS.*tS-r*f);
varT=1/M*sum((dfdS.*tS-r*f).^2)-valT^2;
varT=varT/M;
```

adjointPwS.m

```
function [val,valD,valV,valR,valT,var,varD,varV,varR,varT] ...
= adjointPwS(r,sigma,T,S0,K,M,N,range)

%[val,valD,valV,valR,valT,var,varD,varV,varR,varT]
% Implementation of "adjoint" PwS in 1 dimensional case (useless a priori)
% Adapted to digital call case through "smooth"-ening fx
% M - number of samples
% N - number of timesteps
% range=range of smoothening of payoff
S=S0*ones(1,M);
D=[];
Bv=[];
Br=[];
Bt=[];
h=T/N;
for n=1:1:N % euler simulation
    dW=sqrt(h)*randn(1,M);
%   dS=dS.*(1+r*h+sigma*dW); % delta evol
    Dn=(1+r*h+sigma*dW);
    D=[D;Dn]; %size n*M
%   vS=vS.*(1+r*h+sigma*dW)+S.*dW; % vega evol
    Bvn=S.*dW;
    Bv=[Bv;Bvn];
```

```
%   rS=rS.*(1+r*h+sigma*dW)+h*S; % rho evol
    Brn=h*S;
    Br=[Br;Brn];
%   tS=tS.*(1+r*h+sigma*dW)+S.*(r/N+sigma/(2*N*h)*dW); %theta evol
    Btn=S.*(r/N+sigma/(2*N*h)*dW);
    Bt=[Bt;Btn];
%   S=S.*(1+r*h+sigma*dW);
    S=S.*Dn;
end

%S=S0*prod(D);
[f,dfdS]=smooth(S,K,r,T,range);
val=sum(f)/M;
var=1/M*sum(f.^2)-val^2;
var=var/M;

dfdr=-T*f;
dfdT=-r*f;

%delta
V=dfdS; %initial V=df/dS
for n=N:-1:1
    V=V.*D(n,:);%going back to compute the sensitivity
end
valD=1/M*sum(V);
varD=1/M*sum(V.^2)-valD^2;
varD=varD/M;

%vega
V=dfdS;
valsV=zeros(1,M); %will be cumsum of V(n+1)*B(n) - size 1*M
for n=N:-1:1
    valsV=valsV+V.*Bv(n,:);
    V=V.*D(n,:);%going back to compute the sensitivity
end
valV=1/M*sum(valsV);
```

```
varV=1/M*sum(valsV.^2)-valV^2;
varV=varV/M;

%rho
V=dfdS;
valsR=zeros(1,M); %will be cumsum of V(n+1)*B(n) - size 1*M
for n=N:-1:1
    valsR=valsR+V.*Br(n,:);
    V=V.*D(n,:);%going back to compute the sensitivity
end
valR=1/M*sum(valsR+dfdr);
varR=1/M*sum((valsR+dfdr).^2)-valR^2;
varR=varR/M;

%theta
V=dfdS;
valsT=zeros(1,M); %will be cumsum of V(n+1)*B(n) - size 1*M
for n=N:-1:1
    valsT=valsT+V.*Bt(n,:);
    V=V.*D(n,:);%going back to compute the sensitivity
end
valT=1/M*sum((valsT+dfdT));
varT=1/M*sum((valsT+dfdT).^2)-valT^2;
varT=varT/M;
```

PwS_LRM_hybrid.m

```
function [val,valD,valV,valR,valT,var,varD,varV,varR,varT] = ...
PwS_LRM_hybrid(r,sigma,T,S0,K,M,N,range)
% Tries to achieve greater accuracy than smoothed
% Pws in digital call case
% Computes Smoothed PwS values
% then sums them to
% VMC values of "complementary payoff", which compensates smoothening

if nargin ~= 8
```

```
    error('wrong number of args')
end

% PwS on smoothened payoff
[val0,valD0,valV0,valR0,valT0,var0,varD0,varV0,varR0,varT0]...
=PwS(r,sigma,T,S0,K,M,N,range);

% Computes "complementary payoff", which compensates smoothening
%[V,W]=anti_smooth(S,K,r,T,range);
[val1,valD1,valV1,valR1,valT1,var1,varD1,varV1,varR1,varT1]...
=lrm_As(r,sigma,T,S0,K,M,N,range,'finalsim');

val=val1+val0;
valD=valD1+valD0;
valV=valV1+valV0;
valR=valR1+valR0;
valT=valT1+valT0;
var=var1+var0;
varD=varD1+varD0;
varV=varV1+varV0;
varR=varR1+varR0;
varT=varT1+varT0;


[val1,valD1,valV1,valR1,valT1,var1,varD1,varV1,varR1,varT1]...
=lrm_As(r,sigma,T,S0,K,M,N,range,'pathsim');

val=val1+val0;
valD=valD1+valD0;
valV=valV1+valV0;
valR=valR1+valR0;
valT=valT1+valT0;
var=var1+var0;
varD=varD1+varD0;
varV=varV1+varV0;
varR=varR1+varR0;
```

```
varT=varT1+varT0;
```

lrm_As.m

```
function [val,valD,valV,valR,valT,valG,var,varD,varV,varR,varT,varG]...
= lrm_As(r,sigma,T,S0,K,M,N,range,opt)
%opt='finalsim','pathsim'
%range=range of smoothening to counter
%same as lrm with Anstismoothing triangular payoff as final payoff
switch opt
    case 'finalsim'
        W=sqrt(T)*randn(1,M);
        S=S0.*exp((r-sigma^2/2)*T+sigma*W);
        f=anti_smooth(S,K,r,T,range);

        val=sum(f)/M; %value of option
        var=sum(f.^2)/M-val^2;
        var=var/M;

        Ds=f.*W/(S0*sigma*T);
        valD=sum(Ds)/M; %value of delta
        varD=sum(Ds.^2)/M-valD^2;
        varD=varD/M;

        Vs=f.*((W.^2-T)/(sigma*T)-W);
        valV=sum(Vs)/M; %value of vega
        varV=sum(Vs.^2)/M-valV^2;
        varV=varV/M;

        Rs=f.*(W/(sigma));
        valR=sum(Rs)/M; %value of rho
        varR=sum(Rs.^2)/M-valR^2;
        varR=varR/M;

        Ts=f.*(-1/(2*T)*ones(1,M)+W*...
```

```
     ((r-sigma^2/2)/(sigma*T))+W.^2/(2*T^2));
    valT=sum(Ts)/M; %value of theta
    varT=sum(Rs.^2)/M-valT^2;
    varT=varT/M;

    Gs=f.*((W/(S0*sigma*T)).^2+...
     (-1/(S0^2*sigma^2*T)-W./(S0^2*sigma*T)));
    valG=sum(Gs)/M; %value of theta
    varG=sum(Gs.^2)/M-valG^2;
    varG=varG/M;

case 'pathsim'
     h=T/N; %timestep
    dWs= [];
    S=S0*ones(1,M);
    for n=1:1:N % till final step, is normal euler simulation
        dW=sqrt(h)*randn(1,M);
        % n*M matrix containing in each column the random
        %sqrt(h)*Z of 1 of the M paths
        dWs=[dWs;dW];
        S=S.*(1+r*h+sigma*dW); % value evol
    end

    f=anti_smooth(S,K,r,T,range);

    val=sum(f)/M; %value of option
    var=sum(f.^2)/M-val^2;
    var=var/M;

    Ds=f.*dWs(1,:)/(S0*sigma*h);
    valD=sum(Ds)/M; %value of delta
    varD=sum(Ds.^2)/M-valD^2;
    varD=varD/M;

    Vs=f.*sum((dWs.^2-h)/(sigma*h));
    valV=sum(Vs)/M; %value of vega
```

```
        varV=sum(Vs.^2)/M-valV^2;
        varV=varV/M;

        Rs=f.*sum((dWs/(sigma)));
        valR=sum(Rs-T*exp(-r*T)*f)/M; %value of rho
        varR=sum((Rs-T*exp(-r*T)*f).^2)/M-valR^2;
        varR=varR/M;

        Ts=f.*sum((-1/(2*h)*ones(N,M)+dWs.*...
         ((r-sigma^2/2)/(sigma*h))+dWs.^2./(2*h^2)))/N;
        valT=sum(Ts-r*exp(-r*T)*f)/M; %value of theta
        varT=sum((Rs-r*exp(-r*T)*f).^2)/M-valT^2;
        varT=varT/M;

        Gs=f.*( (dWs(1,:)/(S0*sigma*h)).^2+...
         (-1/(S0^2*sigma^2*h)-dWs(1,:)./(S0^2*sigma*h)));
        valG=sum(Gs)/M; %value of gamma
        varG=sum(Gs.^2)/M-valG^2;
        varG=varG/M;

end
```

anti_smooth.m

```
function [V,W] = anti_smooth(S,K,r,T,range)
% Computes "complementary payoff", which compensates smoothening
% created by smooth function in digital call case
% outputs value of compensation and its derivative

if nargin ~= 5
    error('wrong number of args')
end

%would be "normal digital payoff"
Chi_up_K=cast(S>K,'double');
%value 1 with all values which are above K+range
Chi_up=cast(S>K+range,'double');
%value 1 with all values which are below K-range
Chi_lo=cast(S<K-range,'double');
%value 1 on smoothening zone
Chi_mid=1-(Chi_lo+Chi_up);

% triangle shape to compensate
V=exp(-r*T)*(Chi_up_K-(Chi_up+Chi_mid.*((S-(K-range))/(2*range))));
%derivative is constant on smoothening zone and zero elsewhere
W=exp(-r*T)*(-1/(2*range)*Chi_mid);
%NB: jump at K is not taken into account into our "derivative",
%because P(K)=0
```

VMC.m

```
function [val,valD,valV,valR,valT,var,varD,varV,varR,varT]...
 = VMC(r,sigma,T,S0,K,M,N,d,opt1)
% vibrato computes VMC values of 'value','delta','vega'
%in digital call case
%
% Black-Scholes model, digital call case
% Euler-Maruyama Approximation
%
% r     - interest rate
```

```
% sigma - volatility
% T     - time to expiry
% S0    - asset initial value(s)
% K     - strike price(s)
% M     - number of samples
% N     - number of timesteps
% d     - number of final samples for Z
% opt1  - 'none','antithetic' is chose variance reduction method

if nargin ~= 9
    error('wrong number of args')
end

h=T/N; %timestep

S=S0*ones(1,M); % array of underlying value
dS=ones(1,M); %array of deltas
vS=zeros(1,M); %array of vegas
rS=zeros(1,M); %array of rhos
tS=zeros(1,M); %array of thetas

for n=1:1:N-1 % till step N-1, is normal euler simulation
    dW=sqrt(h)*randn(1,M);
    dS=dS.*(1+r*h+sigma*dW); % delta evol
    vS=vS.*(1+r*h+sigma*dW)+S.*dW; % vega evol
    rS=rS.*(1+r*h+sigma*dW)+h*S; % rho evol
    tS=tS.*(1+r*h+sigma*dW)+S.*(r/N+sigma/(2*N*h)*dW); %theta evol
    S=S.*(1+r*h+sigma*dW); % value evol
end

Z=randn(d,M); % final vibrato step, N-1 to N

muw=S*(1+r*h); %array of drift of final S (dim M)
sigmaw=S*(sigma*sqrt(h)); %array of vols of final timestep
dmuw=dS*(1+r*h); % final delta-drifts
dsigmaw=dS*(sigma*sqrt(h)); % final delta-vols
```

```
vmuw=vS.*(1+r*h); % final vega-drifts
vsigmaw=(vS*(sigma*sqrt(h))+S*sqrt(h)); % final vega-vols
rmuw=rS.*(1+r*h)+h*S; % final rho-drifts
rsigmaw=rS*(sigma*sqrt(h)); % final rho-vols
tmuw=tS*(1+r*h)+r*S/N; % final theta-drifts
tsigmaw=tS*sigma*sqrt(h)+sigma/(2*N*sqrt(h))*S; %final theta-vols

%value
switch opt1
    case 'none'
     % matrix d*M of muw+sigmaw*Z
        SNmat=ones(d,1)*muw+(ones(d,1)*sigmaw).*Z;
        % matrix d*M, f(SN) discounted payoff
        VNmat=payoff(SNmat,K,r,T,'digcall');
        dfdr_mat=-T*VNmat;
        dfdT_mat=-r*VNmat;
        vals=sum(VNmat)/d; %vector of values averaged wrt Z
        dfdr=-T*vals;
        dfdT=-r*vals;
    case 'antithetic'
        SNmatP=ones(d,1)*muw+(ones(d,1)*sigmaw).*Z;
        SNmatM=ones(d,1)*muw-(ones(d,1)*sigmaw).*Z;
        SNmat=ones(d,1)*muw;
        VNmatP=payoff(SNmatP,K,r,T,'digcall');
        VNmatM=payoff(SNmatM,K,r,T,'digcall');
        VNmatC=payoff(SNmat,K,r,T,'digcall');
        % equivalent of previous VNmat but with antith vars
        VNmat=1/2*(VNmatP+VNmatM);
        dfdr_mat=-T*VNmat;
        dfdT_mat=-r*VNmat;
        vals=sum(VNmat)/d; %vector of values averaged wrt Z
        dfdr=-T*vals;
        dfdT=-r*vals;
end

% average of (average over Z) over all trajectories
```

```
val=sum(vals)/M;

% vector of variances wrt Z
varz=1/d*sum(VNmat.^2)-(1/d*sum(VNmat)).^2;
% vector of variance wrt W of avgd value (over Z)
varw=1/M*sum(vals.^2)-(val)^2;

var=1/M*varw+1/(M*d)*1/M*sum(varz); %variance of value


%delta
switch opt1
    case 'none'
     % matrix d*M, Z/sig*f(mu+sig*Z)
        matmu=Z.*(ones(d,1)*(1./sigmaw)).*VNmat;
        % matrix d*M, (Z^2-1)/sig*...
        matsi=(Z.^2-1).*(ones(d,1)*(1./sigmaw)).*VNmat;
    case 'antithetic'
    % matrix d*M, Z/sig*1/2(fp-fm)
        matmu=Z.*(ones(d,1)*(1./sigmaw))*1/2.*(VNmatP-VNmatM);
        % matrix d*M, (Z^2-1)/sig*1/2*(...
        matsi=(Z.^2-1).*(ones(d,1)*(1./sigmaw))*1/2.*(VNmatP-2*VNmatC+VNmatM);
end
%matrix d*M to be avgd over Z then W to get estimator
mat=(ones(d,1)*dmuw).*matmu+(ones(d,1)*dsigmaw).*matsi;
%valsD=1/d*(dmuw.*sum(matmu)+dsigmaw.*sum(matsi)); cannot be used, needs
%mat to compute the variance later on
valsD=1/d*sum(mat); % array Ez()

valD=sum(valsD)/M; %delta value =Ew(Ez())
%variance of delta
varD=(1/M)*(1/M*sum(valsD.^2)-(valD)^2)+1/(M*d)*1/M*sum(sum(mat.^2)/d-valsD.^2);

%vega
%we keep same values for matmu and matsi - not dependent on sensitivity type
%matrix d*M to be avgd over Z then W to get estimator
```

```
mat=(ones(d,1)*vmuw).*matmu+(ones(d,1)*vsigmaw).*matsi;
valsV=1/d*sum(mat); % array Ez()
valV=1/M*sum(valsV); % vega value  =Ew(Ez())
%variance of vega
varV=(1/M)*(1/M*sum(valsV.^2)-(valV)^2)+1/(M*d)*1/M*sum(sum(mat.^2)/d-valsV.^2);

%rho
%we keep same values for matmu...
%matrix d*M to be avgd over Z then W to get estimator
mat=(ones(d,1)*rmuw).*matmu+(ones(d,1)*rsigmaw).*matsi;
% array Ez() - includes differentiation of discount term
valsR=1/d*sum(mat+dfdr_mat);
valR=1/M*sum(valsR); % vega value  =Ew(Ez())
varR=(1/M)*(1/M*sum((valsR).^2)-(valR)^2)...
+1/(M*d)*1/M*sum(sum((mat+dfdr_mat).^2)/d-valsR.^2); %variance of rho

%theta
%we keep same values for matmu...
%matrix d*M to be avgd over Z then W to get estimator
mat=(ones(d,1)*tmuw).*matmu+(ones(d,1)*tsigmaw).*matsi;
% array Ez() - includes differentiation of discount term
valsT=1/d*sum(mat+dfdT_mat);
valT=sum(valsT)/M; %delta value =Ew(Ez())
varT=(1/M)*(1/M*sum((valsT).^2)-(valT)^2)...
+1/(M*d)*1/M*sum(sum((mat+dfdT_mat).^2)/d-valsT.^2); %variance of theta
```

payoff_barrier.m

```
function V = payoff_barrier(S1,S,B,K,r,T1,T)
% returns discounted payoff of digital call @(K,T)
% with discrete barrier @(B,T1)
% takes as two first argument values @ T1 and T

SM=max(S1,S);
V=exp(-r*T)*cast(S>K,'double').*cast(SM<B,'double');
```

VMC_barr.m

```
function [val,valD,valV,valR,valT,var,varD,varV,varR,varT]...
 = VMC_barr(r,sigma,T,S0,B,K,M,N,d,T1)
% vibrato computes VMC values of 'value','delta','vega' in digital call at
% strike K and expiry T case with discrete up and out barrier at time T1
% Uses antithetic vars
%
% Black-Scholes model, digital call case
% Euler-Maruyama Approximation
%
% r     - interest rate
% sigma - volatility
% T     - time to expiry
% S0    - asset initial value(s)
% K     - strike price(s)
% M     - number of samples
% N     - number of timesteps
% d     - number of final samples for Z
% T1    - intermediate time where barrier constraint is checked

if nargin ~= 10
    error('wrong number of args')
end

h=T/N; %timestep
N1=floor(T1/h);%index of barrier

S=S0*ones(1,M); % array of underlying value
dS=ones(1,M); %array of deltas
vS=zeros(1,M); %array of vegas
rS=zeros(1,M); %array of rhos
tS=zeros(1,M); %array of thetas

for n=1:1:N1-1 % till step N1-1, is normal euler simulation
    dW=sqrt(h)*randn(1,M);
    dS=dS.*(1+r*h+sigma*dW); % delta evol
```

```
    vS=vS.*(1+r*h+sigma*dW)+S.*dW; % vega evol
    rS=rS.*(1+r*h+sigma*dW)+h*S; % rho evol
    tS=tS.*(1+r*h+sigma*dW)+S.*(r/N+sigma/(2*N*h)*dW); %theta evol
    S=S.*(1+r*h+sigma*dW); % value evol
end

%storing values @ N1-1 for final evaluation
S1=S;dS1=dS;vS1=vS;rS1=rS;tS1=tS;

%skipping barrier time T1 - size 2*h
dW=sqrt(2*h)*randn(1,M);
dS=dS.*(1+r*2*h+sigma*dW); % delta evol
vS=vS.*(1+r*2*h+sigma*dW)+S.*dW; % vega evol
rS=rS.*(1+r*2*h+sigma*dW)+2*h*S; % rho evol
tS=tS.*(1+r*2*h+sigma*dW)+S.*(r/N+sigma/(2*N*2*h)*dW); %theta evol
S=S.*(1+r*2*h+sigma*dW); % value evol

% Brownian Bridge - drifts, vols @ T1
muw1=(S1+S)/2; %array of S-drifts @T1 (dim M)
sigmaw1=S1*(sigma*sqrt(h/2)); %array of vols - includes sqrt(h/2) term
dmuw1=(dS1+dS)/2; % delta-drifts
dsigmaw1=dS1*(sigma*sqrt(h/2)); % delta-vols
vmuw1=(vS1+vS)/2; % vega-drifts
vsigmaw1=(vS1*(sigma*sqrt(h/2))+S1*sqrt(h/2)); % vega-vols
rmuw1=(rS1+rS)/2; % vega-drifts
rsigmaw1=rS1*(sigma*sqrt(h/2)); % rho-vols
tmuw1=(tS1+tS)/2; % theta-drifts
tsigmaw1=tS1*sigma*sqrt(h/2)+sigma/(2*N*sqrt(h*2))*S1; % theta-vols


for n=N1+1:1:N-1 % again normal euler simulation
    dW=sqrt(h)*randn(1,M);
    dS=dS.*(1+r*h+sigma*dW); % delta evol
    vS=vS.*(1+r*h+sigma*dW)+S.*dW; % vega evol
    rS=rS.*(1+r*h+sigma*dW)+h*S; % rho evol
    tS=tS.*(1+r*h+sigma*dW)+S.*(r/N+sigma/(2*N*h)*dW); % theta evol
```

```
    S=S.*(1+r*h+sigma*dW); % value evol
end

%S=S;dS=dS;vS=vS;rS=rS;tS=tS; % stored values at final euler-time N-1

% final vibrato step, N-1 to N
Z1=randn(d,M);%d=d sims, M= M paths
Z=randn(d,M);

% drifts, vol @T
muw=S*(1+r*h); % array of S-drifts @T (dim M)
sigmaw=S*(sigma*sqrt(h)); % array of vols  - includes sqrt(h) term
dmuw=dS*(1+r*h); % delta-drifts
dsigmaw=dS*(sigma*sqrt(h)); % delta-vols
vmuw=vS*(1+r*h); % vega-drifts
vsigmaw=(vS*(sigma*sqrt(h))+S*sqrt(h)); % vega-vols
rmuw=rS*(1+r*h)+h*S; % vega-drifts
rsigmaw=rS*(sigma*sqrt(h)); % rho-vols
tmuw=tS*(1+r*h)+r*S/N; % theta-drifts
tsigmaw=tS*sigma*sqrt(h)+sigma/(2*N*sqrt(h))*S; % theta-vols

%value

SNmatP1=ones(d,1)*muw1+(ones(d,1)*sigmaw1).*Z1; %size d*M
SNmatM1=ones(d,1)*muw1-(ones(d,1)*sigmaw1).*Z1;
SNmat1=ones(d,1)*muw1;

SNmatP=ones(d,1)*muw+(ones(d,1)*sigmaw).*Z; %size d*M
SNmatM=ones(d,1)*muw-(ones(d,1)*sigmaw).*Z;
SNmat=ones(d,1)*muw;

VNmatP=payoff_barrier(SNmatP1,SNmatP,B,K,r,T1,T); % size d*M, f(mu+Cz)
VNmatM=payoff_barrier(SNmatM1,SNmatM,B,K,r,T1,T);
VNmatC=payoff_barrier(SNmat1,SNmat,B,K,r,T1,T);
VNmat=1/2*(VNmatP+VNmatM);
%used in rho calculation when differentiating discount term
```

```
dfdr_mat=-T*VNmat;
%same for theta calculation
dfdT_mat=-r*VNmat;

vals=sum(VNmat)/d; %vector of values averaged wrt Z
val=sum(vals)/M; % average of (average over Z) over all trajectories
varz=1/d*sum(VNmat.^2)-(1/d*sum(VNmat)).^2; % vector of variances wrt Z
% vector of variance wrt W of avgd value (over Z)
varw=1/M*sum(vals.^2)-(val)^2;

var=1/M*varw+1/(M*d)*1/M*sum(varz); %variance of value


%delta
% matrix d*M, Z/sig*1/2(fp-fm)
matmu1=Z1.*(ones(d,1)*(1./sigmaw1))*1/2.*(VNmatP-VNmatM);
% matrix d*M, Z/sig*1/2(fp-fm)
matmu=Z.*(ones(d,1)*(1./sigmaw))*1/2.*(VNmatP-VNmatM);

% matrix d*M, (Z1^2-1)/sig1*1/4*(...
matsi1=(Z1.^2-1).*(ones(d,1)*(1./sigmaw1))*1/4.*(VNmatP-2*VNmatC+VNmatM);
% matrix d*M, (Z^2-1)/sig*1/4*(...
matsi=(Z.^2-1).*(ones(d,1)*(1./sigmaw))*1/4.*(VNmatP-2*VNmatC+VNmatM);


%valsD=1/d*(dmuw1.*sum(matmu1)+dmuw.*sum(matmu))+1/d*...
%(2*sigmaw1.*dsigmaw1.*sum(matsi1)+2*sigmaw.*dsigmaw.*sum(matsi))%array Ez()
%is not possible to compute variance later !

%matrix d*M to be avgd over Z then W to get estimator
mat=(ones(d,1)*dmuw1).*matmu1+(ones(d,1)*dmuw).*matmu+...
    (ones(d,1)*(2*sigmaw1.*dsigmaw1)).*matsi1+(ones(d,1)*...
    (2*sigmaw.*dsigmaw)).*matsi;
valsD=1/d*sum(mat); % array Ez()
valD=sum(valsD)/M; %delta value =Ew(Ez())
varD=(1/M)*(1/M*sum(valsD.^2)-(valD)^2)...
```

```
    +1/(M*d)*1/M*sum(sum(mat.^2)/d-valsD.^2); %variance of delta

%vega
%we keep same matmu and matsi - not dependent on sensitivity type
mat=(ones(d,1)*vmuw1).*matmu1+(ones(d,1)*vmuw).*matmu...
    +(ones(d,1)*(2*sigmaw1.*vsigmaw1)).*matsi1...
    +(ones(d,1)*(2*sigmaw.*vsigmaw)).*matsi; %matrix d*M
valsV=1/d*sum(mat); % array Ez()
valV=1/M*sum(valsV); % vega value  =Ew(Ez())
varV=(1/M)*(1/M*sum(valsV.^2)-(valV)^2)...
    +1/(M*d)*1/M*sum(sum(mat.^2)/d-valsV.^2); %variance of vega

%rho
mat=(ones(d,1)*rmuw1).*matmu1+(ones(d,1)*rmuw).*matmu...
    +(ones(d,1)*(2*sigmaw1.*rsigmaw1)).*matsi1...
    +(ones(d,1)*(2*sigmaw.*rsigmaw)).*matsi; %matrix d*M
valsR=1/d*sum(mat+dfdr_mat); % array Ez()
valR=1/M*sum(valsR); % vega value  =Ew(Ez())
varR=(1/M)*(1/M*sum(valsR.^2)-(valR)^2)...
    +1/(M*d)*1/M*sum(sum((mat+dfdr_mat).^2)/d-valsR.^2); %variance of rho

%theta
mat=(ones(d,1)*tmuw1).*matmu1+(ones(d,1)*tmuw).*matmu...
    +(ones(d,1)*(2*sigmaw1.*tsigmaw1)).*matsi1...
    +(ones(d,1)*(2*sigmaw.*tsigmaw)).*matsi; %matrix d*M
valsT=1/d*sum(mat+dfdT_mat); % array Ez()
valT=sum(valsT)/M; %delta value =Ew(Ez())
varT=(1/M)*(1/M*sum(valsT.^2)-(valT)^2)...
    +1/(M*d)*1/M*sum(sum((mat+dfdT_mat).^2)/d-valsT.^2); %variance of theta
```

VMC_mult.m

```
function [val,valD,valV,valR,valT,var,varD,varV,varR,varT]...
 = VMC_mult(r,sigma,T,S0,K,M,N,d,e)

%Same as VMC function but explores a wider smoothing, ie. estimating the
%final value as average of distribution over several steps instead of just one.
```

```
%Corresponds to VMC if e=1
%Uses antithetic vars
%d=number of final samples
%e=number of skipped steps

h=T/N; %timestep

S=S0*ones(1,M); % array of underlying value
dS=ones(1,M); %array of deltas
vS=zeros(1,M); %array of vegas
rS=zeros(1,M); %array of rhos
tS=zeros(1,M); %array of thetas

for n=1:1:N-e % till step N-e, is normal euler simulation
    dW=sqrt(h)*randn(1,M);
    dS=dS.*(1+r*h+sigma*dW); % delta evol
    vS=vS.*(1+r*h+sigma*dW)+S.*dW; % vega evol
    rS=rS.*(1+r*h+sigma*dW)+h*S; % rho evol
    tS=tS.*(1+r*h+sigma*dW)+S.*(r/N+sigma/(2*N*h)*dW); %theta evol
    S=S.*(1+r*h+sigma*dW); % value evol
end

Z=randn(d,M); % final vibrato step, N-e to N

muw=S*(1+r*e*h); %array of means of final S
sigmaw=S*(sigma*sqrt(e*h)); %array of vols of final timestep
dmuw=dS*(1+r*e*h); %array of final delta mu
dsigmaw=dS*(sigma*sqrt(e*h)); % array of final delta vol
vmuw=vS.*(1+r*e*h); %array of final vega drift
vsigmaw=(vS*(sigma*sqrt(e*h))+S*sqrt(e*h)); % array of final vega vol
rmuw=rS.*(1+r*e*h)+e*h*S; %array of final vega drift
rsigmaw=rS*(sigma*sqrt(e*h)); % array of final vega vol
tmuw=tS*(1+r*e*h)+r*S/N;
tsigmaw=tS*sigma*sqrt(e*h)+sigma/(2*N*sqrt(e*h))*S;

%value
```

```
SNmatP=ones(d,1)*muw+(ones(d,1)*sigmaw).*Z;
SNmatM=ones(d,1)*muw-(ones(d,1)*sigmaw).*Z;
SNmat=ones(d,1)*muw;
%anti_smooth(S,K,r,T,range)
VNmatP=payoff(SNmatP,K,r,T,'digcall');
VNmatM=payoff(SNmatM,K,r,T,'digcall');
VNmatC=payoff(SNmat,K,r,T,'digcall');
VNmat=1/2*(VNmatP+VNmatM); % equivalent of previous VNmat with antith vars
dfdr_mat=-T*VNmat;%used in rho calc
dfdT_mat=-r*VNmat;
vals=sum(VNmat)/d; %vector of values averaged wrt Z

val=sum(vals)/M; % average of (average over Z) over all trajectories
varz=1/d*sum(VNmat.^2)-(1/d*sum(VNmat)).^2; % vector of variances wrt Z
varw=1/M*sum(vals.^2)-(val)^2; % vector of variance wrt W of avgd value (over Z)
var=1/M*varw+1/(M*d)*1/M*sum(varz); %variance of value



%delta
%antithetic vars
matmu=Z.*(ones(d,1)*(1./sigmaw))*1/2.*(VNmatP-VNmatM);
matsi=(Z.^2-1).*(ones(d,1)*(1./sigmaw))*1/2.*(VNmatP-2*VNmatC+VNmatM);
mat=(ones(d,1)*dmuw).*matmu+(ones(d,1)*dsigmaw).*matsi;
valsD=1/d*sum(mat);
valD=sum(valsD)/M;
varD=(1/M)*(1/M*sum(valsD.^2)-(valD)^2)+...
    1/(M*d)*1/M*sum(sum(mat.^2)/d-valsD.^2);

%vega
mat=(ones(d,1)*vmuw).*matmu+(ones(d,1)*vsigmaw).*matsi;
valsV=1/d*sum(mat);
valV=1/M*sum(valsV);
varV=(1/M)*(1/M*sum(valsV.^2)-(valV)^2)+...
    1/(M*d)*1/M*sum(sum(mat.^2)/d-valsV.^2);

%rho
```

```
mat=(ones(d,1)*rmuw).*matmu+(ones(d,1)*rsigmaw).*matsi;
valsR=1/d*sum(mat+dfdr_mat);
valR=1/M*sum(valsR);
varR=(1/M)*(1/M*sum(valsR.^2)-(valR)^2)+...
    1/(M*d)*1/M*sum(sum((mat+dfdr_mat).^2)/d-valsR.^2);

%theta
mat=(ones(d,1)*tmuw).*matmu+(ones(d,1)*tsigmaw).*matsi;
valsT=1/d*sum(mat+dfdT_mat);
valT=sum(valsT)/M;
varT=(1/M)*(1/M*sum(valsT.^2)-(valT)^2)+...
    1/(M*d)*1/M*sum(sum((mat+dfdT_mat).^2)/d-valsT.^2);
```