AUTOMATIC CONTROL VARIATES FOR OPTION PRICING USING NEURAL NETWORKS

A PREPRINT

El Filali Ech-Chafiq Zineb
Univ. Grenoble Alpes, CNRS,
Grenoble INP, LJK,
38000 Grenoble, France.
Quantitative analyst at Natixis, Paris
zinebelfilaliechchafique@gmail.com

Lelong Jérôme
Univ. Grenoble Alpes, CNRS,
Grenoble INP, LJK,
38000 Grenoble, France.
jerome.lelong@univ-grenoble-alpes.fr

Reghai Adil
Head of Quantitative Research
Equity and commodity Markets,
Natixis, 47 quai d'Austerlitz
75013 Paris
adil.reghai@natixis.com

June 30, 2020

ABSTRACT

Many pricing problems boil down to the computation of a high dimensional integral, which is usually estimated using Monte Carlo. In fact, the accuracy of a Monte Carlo estimator with M simulations is given by $\frac{\sigma}{\sqrt{M}}$. Meaning that its convergence is immune to the dimension of the problem. However, this convergence can be relatively slow depending on the variance σ of the function to be integrated. To resolve such a problem, one would perform some variance reduction techniques such as importance sampling, stratification, or control variates.

In this paper, we will study two approaches for improving the convergence of Monte Carlo using Neural Networks. The first approach relies on the fact that many high dimensional financial problems are of low effective dimensions[15]. We expose a method to reduce the dimension of such problems in order to keep only the necessary variables. The integration can then be done using fast numerical integration techniques such as Gaussian quadrature. The second approach consists in building an automatic control variate using neural networks. We learn the function to be integrated (which incorporates the diffusion model plus the payoff function) in order to build a network that is highly correlated to it. As the network that we use can be integrated exactly, we can use it as a control variate.

Keywords Monte Carlo · Neural Networks · Variance reduction · Control Variate · effective dimension

1 Introduction

Classic methods for pricing and hedging can be very slow at times. With the new regulatory context and as we seek high level performances, the demand for efficient pricing and hedging methods becomes very pressing. The question is how efficient are the methods commonly used. Depending on a number of factors such as the diffusion model or the number of underlying assets, different pricers can be considered. However, we notice that Monte Carlo is the most commonly used one. In fact, it offers a big flexibility since it can be used under any diffusion model and with as many underlying assets as needed. Moreover, its variance is immune to the dimension of the problem which makes it very suitable for high dimensional integration problems which are very common in finance. So how does the Monte Carlo method work and how do we use it in financial problems?

We consider the following SDE governing the diffusion of the underlying assets S^i

$$dS_t^i = \mu_i(t, S_t^i)dt + \sigma_i(t, S_t^i)dB_t^i. \tag{1}$$

where the B^i's are correlated Brownian motions and let g be a payoff function depending on the processes $(S_t^i)_{0\leq t\leq T}$. The computation of g's price often bounds to the computation of an expectation of the form $\mathbb{E}(f(X))$ which using the strong law of large numbers may be approached with the estimator $S_M = \frac{1}{M}\sum_{i=1}^{M} f(X_i)$, where the X_i's are random samples of X and M a sufficiently large number. S_M is the Monte Carlo estimator. We pose $\sigma^2 = \mathrm{Var}(f(X))$, the variance of S_M is given by

$$\mathrm{Var}(S_M) = \mathrm{Var}\left(\frac{1}{M}\sum_{i=1}^{M} f(X_i)\right) = \frac{\sigma^2}{M}. \tag{2}$$

Note that the variance of the estimator is proportional to the variance of the integrand. Therefore, it is always interesting to find a variable Y such that

$$\begin{cases} \mathbb{E}(Y) = \mathbb{E}(f(X)), \\ \mathrm{Var}(Y) < \mathrm{Var}(f(X)). \end{cases}$$

There are several techniques to find such a variable Y. Here, we will focus on control variates.

Let $Y = f(X) - h(X) + \mathbb{E}(h(X))$ where we assume that we can compute $\mathbb{E}(h(X))$ exactly. writing that

$$\begin{cases} \mathrm{Var}(Y) = \mathrm{Var}(f(X)) + \mathrm{Var}(h(X)) - 2\,\mathrm{Cov}(f(X), h(X)), \\ \quad \mathrm{Cov}(f(X), h(X)) > \dfrac{1}{2}\,\mathrm{Var}(h(X)) \end{cases} \tag{3}$$

We deduce that $\mathrm{Var}(Y) < \mathrm{Var}(f(X))$.

In [5] and [6], the authors study control variates that depend on some parameter θ, the control variate has then the form $h(X,\theta) - \mathbb{E}\left(h(X,\theta)\right)$ and suggest two adaptive algorithms to find an adequate parameter θ^* optimising the control variate. The first one uses a stochastic approximation scheme where the parameter θ^* is chosen as to minimise the variance of the estimator and the minimisation algorithm is based on stochastic gradient descent. The second algorithm is based on a sample average algorithm where θ^* is estimated using a first random sample of X so as to minimise the variance of the estimator. Then a second sample of X is drawn independently from the first one to estimate the actual expectation. A similar work has been conducted in [9], the authors work on adaptive Monte Carlo which is a generalisation of this parametric control variate problem. Actually, one can write $\mathbb{E}(X)$ as $\mathbb{E}(H(\theta, X))$. The problem is then to find such a parametric representation of X and then solve for θ^*. Another issue that needs to be addressed is how to find the actual $h(X)$ that approximates $f(X)$ appropriately. In [10], the author suggests to use a projection of $f(X)$ on the first p terms of an orthonormal basis $(e_k)_{1\leq k\leq p}$ in $L^2(D)$ where D is the domain of integration of f. So $h(X)$ writes $\sum_{k=1}^{p} a_k e_k(x)$. The paper works on finding an optimal estimation for the coefficients a_k. In [3], Glynn and Szechtman show how to construct a control variate in the case where the simulation of X involves

simulating $V_1, V_2, \ldots, V_\tau$ iid with $\mathbb{E}(V_j)$ known for all j and τ a stochastic variable. In [11], the authors suppose that for a multivariate scope, if we replace all the variables but one with their means, the resulting function's expectation becomes easy to compute and it has a lot of information about the multivariate function. they then combine all these resulting functions to construct the control variate. In, [12], the authors study the combination of multiple control variates and the convergence limit of Monte Carlo when the number of control variates grows to infinity.

In this paper, we want to create sophisticated control variates which automatically adapt to the function f. To do so, one needs to find a function h, highly correlated to f, and such that we have an exact method for computing $\mathbb{E}(h(X))$ or at least a numerical method that is more precise than Monte Carlo. We suggest two main methods for creating automatic control variates. The first one, deals with high dimensional problems with low effective dimensions (see [15] and [14]). For these cases, the function f only relies on some components of X that we find using an adequate neural network and thus f can be integrated using a simple numerical integration method such as Gaussian quadratures. The idea of reducing the dimension of X before searching for the control variate has been exposed in [2] where the authors use Principal Component Analysis to reduce the dimension of the problem before using an adaptive integration algorithm based on quasi Monte Carlo. In the second method, we suggest to use a neural network H as the control variate. The network learns $f(X)$ so that it is naturally correlated to it (we actually learn the diffusion model, the discretization and the payoff function all at once). The architecture of the network is chosen wisely, so as $\mathbb{E}(H(X))$ can be computed easily.

2 Feed forward networks in a nutshell

A feed forward network is composed of a stack of layers each of which is a composition of a linear transformation with an activation function (a non linear function). The idea behind these networks comes from the Universal Approximation Theorem which states that a neural network with one hidden layer can approximate any continuous function for inputs within a specific range (see [8]). More formally, let $\psi : \mathbb{R}^p \longrightarrow \mathbb{R}^q$ (for our case $p = N$ and $q = 1$) be an unknown function that we wish to approximate. We will construct a feed forward network containing a stack of L layers as follows:

$$\begin{aligned} y = a_L(x) &\in \mathbb{R}^q \\ a_L(x) &= W_L a_{L-1}(x) + b_L \\ a_k(x) &= \sigma_a(W_k a_{k-1}(x) + b_k) \ \forall k \in [|1, L-1|] \\ a_0(x) &= x \in \mathbb{R}^p \end{aligned}$$

a_0 is the input layer, it simply contains the input of the network. $a_1, \ldots, a_{L-1}$ are hidden layers that apply (non linear) transformations to their inputs. Finally, a_L is the output layer which contains the output of the network. Each layer k is parametrized by some weights W_k and bias b_k. Training the network consists in finding the best parameters to approximate ψ.

We define a loss function $l_\theta(\psi(x), a_L(x))$ measuring the distance between the real function and the estimated values where θ represents the whole set of weights and bias. We run a minimisation algorithm on l (for example stochastic gradient descent, Adam or RMSprop) in order to find the optimal weights and biases. The optimisation problem writes

$$\min_\theta \ l_\theta(\psi(x), a_L(x)) \tag{4}$$

All the optimisers that we might use are first order methods, which means that we need to compute the gradient of the objective function with respect to each parameter W_k or b_k. Computing all these gradients can become very consuming. However, we use the back propagation algorithm which allows to compute only the gradients with respect to the last layer's parameters and propagate them backward to obtain all the rest of the gradients. More details on back propagation

can be found in [1]. In the following, we use the euclidean distance as a loss function, i.e $l(z, y) = ||y - z||_2$ and Adam for optimisation (more details on this algorithm can be found in [7].

3 First Approach: Network dimension reduction

Let f be a function depending on N independent identically distributed standard normal variables $(Z_1, \dots, Z_N)$. We will search for n normal directions summarising the variance of f of the form $\tilde{Z}_i = \sum_{k=1}^{N} u_{ik} Z_k \ \forall i = 1, \dots, n$ with the condition $\sum_{k=1}^{N} u_{ik}^2 = 1 \ \forall i = 1, \dots, n$. This condition allows the directions to be standardised (i.e $\mathrm{Var}(\tilde{Z}_i) = 1 \ \forall i = 1, \dots, n$). For this purpose, we will build a network that predicts the payoff function given as inputs the random variables $(Z_1, \dots, Z_N)$. It will be constituted of a number of hidden layers which we divide into two sections. The first section is the dimension reduction cell. It contains one hidden non activated layer (this means we only apply a linear transformation to its input) with exactly n units. We also omit the bias from this layer so as it takes the form WZ. The output of this layer should contain the $\tilde{Z}_i$'s. The second section may contain as many hidden layers as needed. Its main role is to reconstitute the payoff function given only the directions obtained in the output of the first part of the network. As the network will try to converge, the dimension reduction cell will find the variables that summarise best the variance, this way allowing the second part of the network to recover the payoff function easily. This idea is summarised in Figure 1

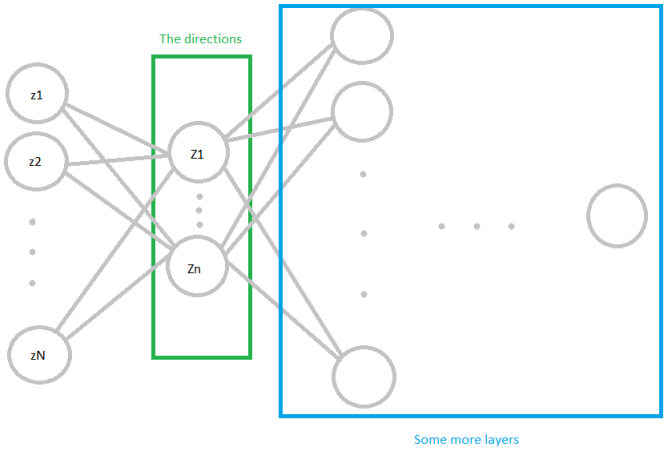


Figure 1: Architecture of the network used for dimension reduction

Once the network is trained, we have $\tilde{Z}_i = \sum_{k=1}^{N} u_{ik} Z_k \ \forall i = 1, \dots, n$ with U being the normalised weight matrix of the first layer.

When the network outputs the directions of interest, it becomes "useless". We extract the weights and get rid of the rest of the network. We will look for a matrix V such as $\mathrm{Im}(U) \oplus^{\perp} \mathrm{Im}(V) = \mathbb{R}^N$. To do so, we write, $Z = \begin{pmatrix} Z_1 \\ \vdots \\ Z_N \end{pmatrix}$ and $\tilde{Z} = \begin{pmatrix} \tilde{Z}_1 \\ \vdots \\ \tilde{Z}_n \end{pmatrix}$ such as $\begin{pmatrix} \tilde{Z} \\ \tilde{Z}^{\perp} \end{pmatrix} = \begin{pmatrix} U \\ V \end{pmatrix} Z = MZ.$
as $\tilde{Z}^{\perp} = VZ$ and $\tilde{Z} = UZ$ are independent, i.e.

$$0 = \mathrm{Cov}(\tilde{Z}, \tilde{Z}^{\perp}) = \mathrm{Cov}(UZ, VZ) = U\ \mathrm{Cov}(Z, Z)\ V^T = UV^T$$

which means that

$$\mathrm{Im}(V^T) \subseteq \ker(U). \tag{5}$$

In addition we have the following results

1. $\dim(\mathrm{Im}(U)) = n$ because the network will output independent directions, otherwise the effective dimension is smaller than the number of directions we impose on the network. This implies that $\dim(\ker(U)) = N - n$
2. On one hand, by adding the $\tilde{Z}^{\perp}$ variable, we wish to complete the original space, meaning that $\dim(\mathrm{Im}(V)) = N - n$ and $\dim(\ker(V)) = n$. On the other hand $\mathrm{Im}(V^T) = \ker(V)^T$ which means that $\dim(\mathrm{Im}(V^T)) = \dim(\ker(V)^T) = N - n$

$$\dim(\mathrm{Im}(V^T)) = \dim(\ker(U)) = N - n. \tag{6}$$

Finally, we conclude from Equations 5 and 6 that $\mathrm{Im}(V^T) = \ker(U)$. To determine V, it suffices to find a basis of $\ker(U)$. The determination of such a basis can be done using Gaussian elimination as follows
We consider the matrix U of size $n \times N$. We construct the row augmented matrix $\begin{pmatrix} U \\ I_N \end{pmatrix}$ where I_N is the $N \times N$ identity matrix. We perform Gaussian elimination on this new matrix (column wise) and we obtain a matrix $\begin{pmatrix} A \\ B \end{pmatrix}$. A basis of $\ker(U)$ consists in the non-zero columns of B for which the corresponding column in A is a zero column. If the method is correct, this matrix should be of rank $N - n$. If not, this means that the columns of U were not independent and the number n of directions is bigger than the effective dimension of the problem.
Once this is done, we simulate $\tilde{Z} \sim \mathcal{N}_n(0, UU^T)$ and $\tilde{Z}^{\perp} \sim \mathcal{N}_{N-n}(0, VV^T)$ since $\mathrm{Cov}(\tilde{Z}) = \mathrm{Cov}(UZ) = UU^T$ and $\mathrm{Cov}(\tilde{Z}^{\perp}) = \mathrm{Cov}(VZ) = VV^T$. The price of the option is given by $\mathbb{E}(f(Z))$ and since

$$\begin{aligned}
\mathbb{E}\left(f(Z)\right) &= \mathbb{E}\left(\mathbb{E}\left(f(Z)/\tilde{Z}\right)\right) \\
&= \mathbb{E}\left(f\left(\mathbb{E}\left(Z/\tilde{Z}\right)\right)\right) + \mathbb{E}\left(\mathbb{E}\left(f(Z)/\tilde{Z}\right) - f\left(\mathbb{E}\left(Z/\tilde{Z}\right)\right)\right) \\
&= \mathbb{E}\left(f\left(M^{-1}\begin{pmatrix}\tilde{Z}\\0\end{pmatrix}\right)\right) + \mathbb{E}\left(f\left(M^{-1}\begin{pmatrix}\tilde{Z}\\\tilde{Z}^{\perp}\end{pmatrix}\right) - f\left(M^{-1}\begin{pmatrix}\tilde{Z}\\0\end{pmatrix}\right)\right)
\end{aligned} \tag{7}$$

We will use the variable $f(\mathbb{E}(Z/\tilde{Z}))$ as a control variate. Since it depends only on the new variables $\tilde{Z}$ of small dimension, we can use a numerical integrator much faster and more precise than Monte Carlo. Also we can use Sobol Sequences for $\tilde{Z}$ simulation to have better convergence results (for more details on Sobol Sequences please refer to [13]). $\tilde{Z}^{\perp}$ is simulated using standard normal random variables.

4 Second Approach: Automatic control variate

Let $H : \mathbb{R}^N \longrightarrow \mathbb{R}$ be a neural network that approximates the payoff function f given the normal variable $Z = (Z_1, \ldots, Z_N)$. We write

$$Y = f(Z) - H(Z) + \mathbb{E}(H(Z))$$

If the network is well trained, $H(Z)$ is highly correlated to $f(Z)$ and we just need to know how to compute $\mathbb{E}(H(Z))$ analytically in order to use it as a control variate. We suggest two ideas for doing so.

4.1 Numerical integration

For this part, we stick to the network architecture given in Figure 1. We may write

$$H(Z) = (\tilde{H} \; o \; a_1)(Z)$$

where $a_1(Z) = WZ$ is the first hidden layer of the network (the green block in Figure 1) and $\tilde{H}$ represents the rest of the layers (the blue block in Figure 1).

$$\begin{aligned}\mathbb{E}(H(Z)) &= \mathbb{E}((\tilde{H} \; o \; a_1)(Z)) \\ &= \mathbb{E}(\tilde{H}(\tilde{Z}))\end{aligned}$$

where $\tilde{Z} = a_1(Z) = WZ \sim \mathcal{N}(0, WW^T)$. Since $\tilde{Z}$ has a small dimension, the integral can be estimated numerically using Gauss Hermite polynomials for example. Of course, this should only be used when the effective dimension is small enough.

4.2 Analytic integration

Here, we suggest a network having a simpler form, which will allow us to integrate it with an exact formula. This will make the computation much faster since the integration of the whole network can be numerically costly. Moreover, the network we suggest is capable of dealing with any payoff function with no restriction on the number of neurons of the first layer. The architecture of this network is described in Figure 2

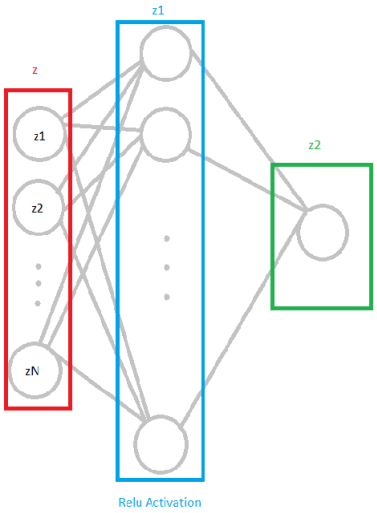


Figure 2: Architecture for automatic control variates

We consider the Relu activation function

$$Relu(x) = \max(0, x) = x_+$$

Then, the network H writes

$$\begin{aligned} H(Z) &= Z_2 \\ &= W_2 Z_1 + b_2 \\ &= W_2 (W_1 Z + b_1)^+ + b_2 \end{aligned}$$

Thus,

$$\mathbb{E}(H(Z)) = W_2 \mathbb{E}((W_1 Z + b_1)^+) + b_2$$

But

$$W_1 Z + b_1 = \begin{pmatrix} \sum_{j=1}^N (W_1)_{1j} Z_j + (b_1)_1 \\ \vdots \\ \sum_{j=1}^N (W_1)_{nj} Z_j + (b_1)_n \end{pmatrix} = \begin{pmatrix} \sigma_1 Y_1 + \mu_1 \\ \vdots \\ \sigma_n Y_n + \mu_n \end{pmatrix}$$

with $\mu_i = (b_1)_i$ and $\sigma_i^2 = \sum_{j=1}^N (W_1)_{ij}$ and $Y_i \sim \mathcal{N}(0,1)$. Then, we just have to compute $\mathbb{E}((\sigma Y + \mu)^+)$. In fact

$$\begin{aligned} \mathbb{E}((\sigma Y + \mu)^+) &= \sigma \int_{-\frac{\mu}{\sigma}}^{+\infty} y f_{\mathcal{N}(0,1)}(y) dy + \mu \mathbb{P}\left(Y \geq -\frac{\mu}{\sigma}\right) \\ &= \sigma \int_{-\frac{\mu}{\sigma}}^{+\infty} \frac{1}{\sqrt{2\pi}} y \exp\left(-\frac{1}{2} y^2\right) dy + \mu \left(1 - \mathcal{N}\left(-\frac{\mu}{\sigma}\right)\right) \\ &= \frac{\sigma}{\sqrt{2\pi}} \left[-\exp(-\frac{1}{2} y^2)\right]_{-\frac{\mu}{\sigma}}^{+\infty} + \mu \left(1 - \mathcal{N}\left(-\frac{\mu}{\sigma}\right)\right) \\ &= \frac{\sigma}{2\pi} \exp\left(-\frac{1}{2} \frac{\mu^2}{\sigma^2}\right) + \mu \left(1 - \mathcal{N}\left(-\frac{\mu}{\sigma}\right)\right), \end{aligned}$$

where $\mathcal{N}$ is the cumulative distribution function of the standard normal distribution. Finally,

$$\mathbb{E}(H(Z)) = W_2 \begin{pmatrix} \cdots \\ \frac{\sigma_i}{2\pi} \exp\left(-\frac{1}{2} \frac{\mu_i^2}{\sigma_i^2}\right) + \mu_i \left(1 - \mathcal{N}\left(-\frac{\mu_i}{\sigma_i}\right)\right) \\ \cdots \end{pmatrix} + b_2. \tag{8}$$

with $\mu_i = (b_1)_i$ and $\sigma_i^2 = \sum_{j=1}^N (W_1)_{ij}$

5 Main numerical results

5.1 Black & Scholes

Let us consider the Black and Scholes model.

$$\begin{aligned} dS_t^i &= r S_t^i dt + \sigma_i S_t^i dB_t^i \\ d(B_t^i B_t^j) &= \rho_{ij} dt \end{aligned}$$

This is a relatively simple model, where each underlying volatility is constant. We will first test our control variates on this model for different payoffs.

- a call on a basket $\max(0.0, \sum_i \omega_i S_T^i - K)$
- a put on a worst of $\max(0.0, K - \min_i(S_T^i))$

- an arithmetic Asian option $\max(0.0, \frac{1}{d}\sum_i \sum_j \omega_j S_{t_i}^j - K)$
- a binary option (a digit) on the basket $G\mathbb{1}_{\sum_i \omega_i S_T^i \geq K}$

We use the following parameters
$S_0^i = 100.0\ \forall i, K = 100.0, \omega_i = \frac{1}{nbsj}\ \forall i, T = 1$ year, $d = 10, G = 100.0, \sigma_i = 0.4\ \forall i, \rho_{ij} = 0.75\ \forall i, j, nbsj = 10.$

5.1.1 Network dimension reduction

In this first experiment we will compare the two methods that use dimension reduction presented in Section 3 (method 1) and Section 4.1 (method 2). Table 1 shows the price given by Monte Carlo as well as the ones given by the networks for the different payoffs. We define Cost as the sum of the training time and the price computation time using the network divided by the Monte Carlo computation time. The speed up represents the ratio between the variances of the standard Monte Carlo and the one with the control variate divided by Cost

Payoff	Basket	Asian	Digit	Worst Of
Monte Carlo Price	15.95	9.75	46.00	26.48
Method 1 price	16.30	9.86	49.51	26.62
Method 1 Var ratio	336.12	166.79	15.28	4.08
Method 1 Cost	24.23	10.22	19.34	23.76
Method 1 Speed up	13.87	16.31	0.79	0.17
Method 2 price	16.19	9.86	47.50	26.55
Method 2 Speed Up	266.93	379.32	35.98	11.44
Method 2 Cost	14.32	3.34	14.97	14.23
Method 2 Speed up	18.64	113.46	2.43	0.80

Table 1: Dimension reduction control variates performances under Black and Scholes

We can see that the control variate is effective for most of the payoffs. Nevertheless, the first method is much more efficient when the function to be integrated is close to being linear (the first term in 7 converges to the actual expectation and we do not even need the bias correction represented by the second term of 7). This is the case for the Basket or the Asian option. In the other cases, the methods still give good control variates and we can see that we have a significant variance reduction in most cases. Note that the Worst Of and the digit are better approximated with the second method since the first one will only capture the linear part of the function whilst the second method will learn the whole payoff function.

5.1.2 Automatic control variates

In this section we study the same payoffs described earlier, we test the automatic control variates described in Section 4.2.

Payoff	Basket	Asian	Digit	Worst Of
Monte Carlo Price	15.86	9.83	46.57	26.68
Control variate	16.22	9.81	46.95	26.64
Var ratio	673.90	119.54	16.07	21.74
Cost	13.82	4.82	12.98	9.75
Speed up	48.76	24.80	1.23	2.23

Table 2: Automatic control variates performances under Black and Scholes

Again, the control variate allows significant speed ups for the different payoffs. We also notice that for the Digit and Worst Of payoffs the speeds are better than when we use the method of dimension reduction and we learn only the linear part of the functions.

5.2 Local Volatility model

We consider a local volatility model where we consider the following form for the volatility function

$$\sigma(t,x) = 0.6(1.2 - e^{-0.1t}e^{-0.001(xe^{rt}-s)^2})e^{-0.05\sqrt{t}}$$

This parametric volatility function is taken from [4], where the authors advise to take s equal to the spot price of the underlying asset in order for the formula to make sense. This will allow the bottom of the smile to be located at the forward money. There is no exact simulation method for this model as in the Black Scholes model, thus we need a discretization scheme. We use an Euler discretization with 100 steps per year. The results are displayed in Table 3.

5.2.1 Network dimension reduction

Payoff	Basket	Asian	Digit	Worst Of
Monte Carlo Price	7.91	4.33	60.41	8.56
Method 1 Price	7.92	4.37	61.75	8.56
Method 1 Var ratio	36.58	76.48	2.81	2.68
Method 1 Cost	8.83	3.02	4.89	4.71
Method 1 Speed up	4.14	25.32	0.57	0.56
Method 2 price	7.84	4.36	61.77	8.50
Method 2 Var ratio	48.59	78.95	3.33	8.18
Method 2 Cost	1.21	1.24	1.50	1.21
Method 2 Speed up	40.15	63.67	2.22	6.76

Table 3: Dimension reduction control variates performances under a local volatility model

Here, we have introduced two levels of complexity. First, the model is more complicated than the Black and Scholes model. Second, the introduction of the time grid increases the inputs for the networks. The first method's speed ups decrease a little but are still acceptable. The second method deals better with this complexity and we even have some improvements in the speed ups.

5.2.2 Automatic control variates

Payoff	Basket	Asian	Digit	Worst Of
Monte Carlo Price	7.90	4.36	60.49	8.54
Control variate	7.85	4.36	60.38	8.49
Var ratio	8.29	9.77	2.06	5.59
Cost	1.85	1.75	2.62	1.85
Speed up	4.48	5.58	0.78	3.02

Table 4: Automatic control variates performances under a local volatility model

The automatic control variate struggles a little bit more with this model than the control variate with numerical integration. This might be due to the fact that we only use one hidden layer in the network and since this model is more complicated we think that one hidden layer is not really sufficient to capture its complexity. However, we still have some interesting speed ups which proves that no matter how complicated the payoff, the model or the time grid can get, the network will still make the best of what you feed to it to give interesting results.

5.3 Stochastic volatility model

We consider the Heston model for the diffusion of the assets:

$$\begin{aligned}
dS_t^i &= rS_t^i dt + \sqrt{\sigma_t^i} S_t^i dB_t^i \\
d\sigma_t^i &= \kappa_i(a_i - \sigma_t^i)dt + \nu_i\sqrt{\sigma_t^i}(\gamma_i dB_t^i + \sqrt{1-\gamma_i^2}d\tilde{B}_t^i) \\
d\langle B\rangle_t &= \Gamma dt,\ d\langle \tilde{B}\rangle_t = I_N dt
\end{aligned}$$

Γ is the correlation matrix between the different assets, I_N is the $N \times N$ identity matrix, κ is the reversion rate of the volatility process, a is the mean level of the volatility process, ν is the volatility of the volatility process and γ_i is the correlation between an asset and its volatility process which we consider constant for the sake of simplicity. the model can be equivalently written

$$\begin{aligned}
dS_t^i &= rS_t^i dt + \sqrt{\sigma_t^i} S_t^i dB_t^i \\
d\sigma_t^i &= \kappa_i(a_i - \sigma_t^i)dt + \nu_i\sqrt{\sigma_t^i}d\hat{B}_t^i
\end{aligned}$$

Where B and $\hat{B}$ are Wiener processes satisfying

$$d\langle B\rangle_t = \Gamma dt,\ d\langle B, \hat{B}\rangle_t = \gamma\Gamma dt,\ d\langle \hat{B}\rangle_t = \left(\gamma^2\Gamma + (1-\gamma^2)I_N\right)dt$$

The process $(B, \hat{B})$ with values in $\mathbb{R}^{2N}$ is a Wiener process with covariance

$$\tilde{\Gamma} = \begin{pmatrix} \Gamma & \gamma\Gamma \\ \gamma\Gamma & \gamma^2\Gamma + (1-\gamma^2)I_N \end{pmatrix}$$

Hence, the pair of processes $(B, \hat{B})$ can be easily simulated by applying the Cholesky factorization of $\tilde{\Gamma}$ to a standard Brownian motion with values in $\mathbb{R}^{2N}$. We use the following model parameters
$\forall i\ \kappa_i = 2, \sigma_i(0) = 0.04, a_i = 0.04, \nu_i = 0.01, \gamma_i = -0.2$

5.3.1 Network dimension reduction

We test the Heston model with the payoffs described earlier. The results are shown in Table 5

Payoff	Basket	Asian	Digit	Worst Of
Monte Carlo Price	9.69	5.76	56.92	12.40
Method 1	9.49	5.58	56.37	12.43
Method 1 Var ratio	20.91	32.83	1.00	2.54
Method 1 Cost	13.85	13.54	11.70	13.82
Method 1 Speed up	1.51	2.42	0.08	0.18
Method 2 price	9.57	5.92	55.19	12.58
Method 2 Var ratio	31.03	502.49	1.63	5.44
Method 2 Cost	1.28	1.27	1.36	1.35
Method 2 Speed up	24.24	395.66	1.19	4.03

Table 5: Dimension reduction control variates performances under a stochastic volatility model

We added again more complexity to the model and also we doubled the size of the networks inputs. However, the results haven't been degraded in comparison to the local volatility model. This means that the performance of the networks isn't really a decreasing function of the model's complexity. Surprisingly enough we notice that for the Asian payoff the performance of the second method is even much better with this more complicated model.

5.3.2 Automatic control variates

Payoff	Basket	Asian	Digit	Worst Of
Monte Carlo Price	9.48	5.96	55.78	12.48
Control variate with analytic integration	9.57	5.71	56.24	12.53
Var ratio	4.02	4.40	1.73	3.98
C_2	2.37	2.27	4.20	2.73
Speed up	1.69	1.93	0.41	1.46

Table 6: Automatic control variates performances under a stochastic volatility model

This is an even more complicated model and the dimensions are again bigger than before. However, we still have some important speed ups and the computation times are still very acceptable.

6 Robustness of the control variates

In this part, we test whether the networks presented in Section 3 are capable of resisting changes in the variables of the payoff and the model. Since this method only learns the most important directions for the payoff and the model and not the actual combination of the two, we hope it will be resisting to some parameters changes.

6.1 Asian price sensitivity

To study this interesting aspect, we first consider an arithmetic Asian option $(\frac{1}{d}\sum_{i=1}^{d} S_{t_i} - K)^+$. We train The network with the following set of parameters
$K = 100.0, \sigma = 0.2, S_0 = 100.0, T = 1$ year and $d = 20$.
Once the network is trained, we test it for a range of parameters larger than the ones used for the training. The prices given by the network in the following figures are the raw prices without using them as control variates. Since this example is relatively simple, those prices are quite accurate without the need to correct the bias through the Monte Carlo.

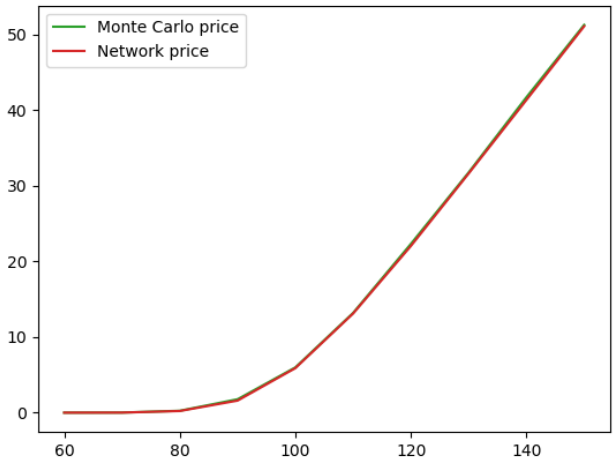


Figure 3: Asian price as a function of the spot

Trained with a spot value 100. The network is capable of approaching the model for a range of spots going from 60 to 150. This is very interesting, because it means we will not have to retrain the networks for every change in the market spot.

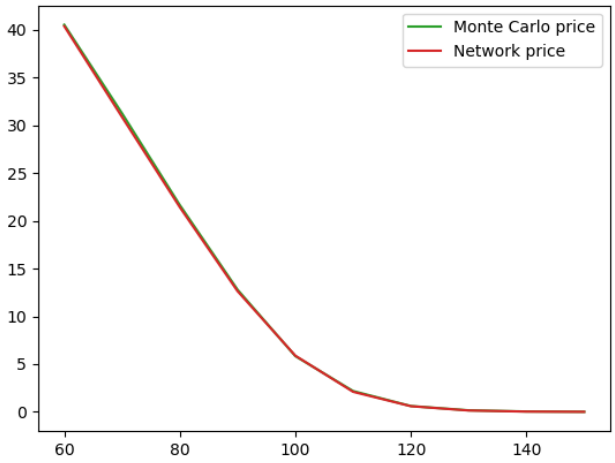


Figure 4: Asian price as a function of the strike

The network is also robust to the change of the strike parameter. Meaning that with one network we can cover a huge range of products having different strikes.

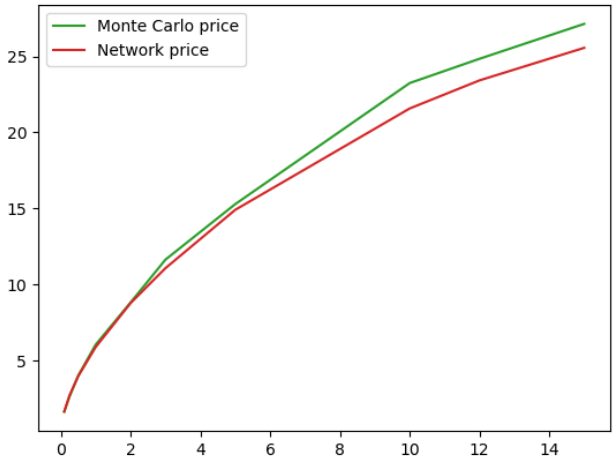


Figure 5: Asian price as a function of the maturity (Monte Carlo vs Network)

The price as a function of the maturity time has some bias, using the network as a control variate corrects this error as shown in the following figure.

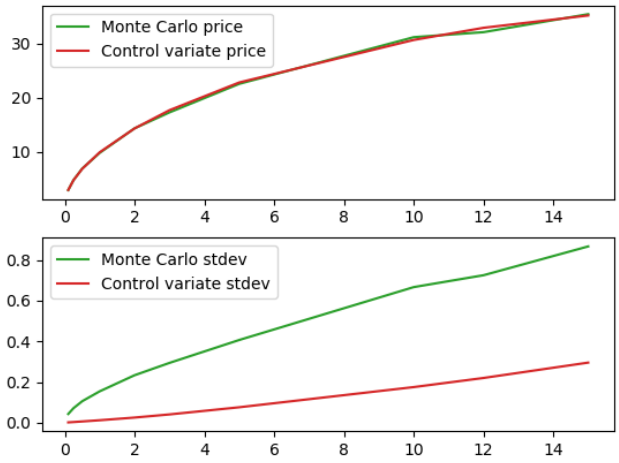


Figure 6: Asian price and stdev as a function of the maturity (Monte Carlo vs Control Variate)

Clearly, the control variate which is trained with a maturity 1 year performs some interesting variance reduction for a much larger and smaller maturities.

6.2 AutoCall price sensitivity

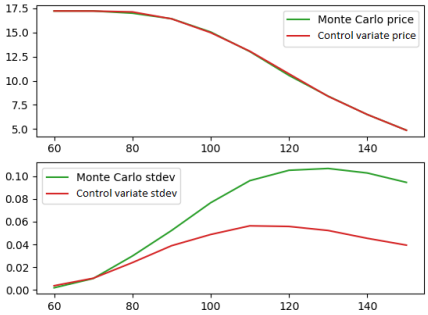
We now consider a more industrial payoff : The AutoCall. This payoff pays the following:
At each recall date t_i the option pays the following conditional coupon

$$\Pi_i = C_i \mathbb{1}_{P_{t_i} \geq AB_i} \prod_{j=0}^{i-1} \mathbb{1}_{P_{t_j} < AB_j}.$$

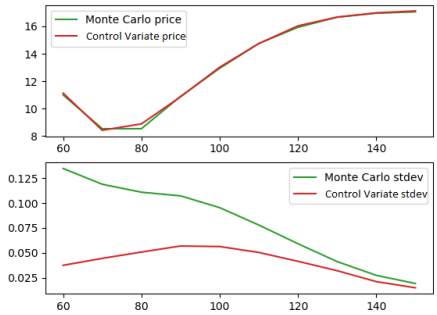
At maturity, if no AutoCall event has occurred, the option pays the following

$$\Pi_T = -(K - P_T)^+ \mathbb{1}_{P_T < DB} \prod_{i=0}^{d} \mathbb{1}_{P_{t_i} < AB_i}.$$

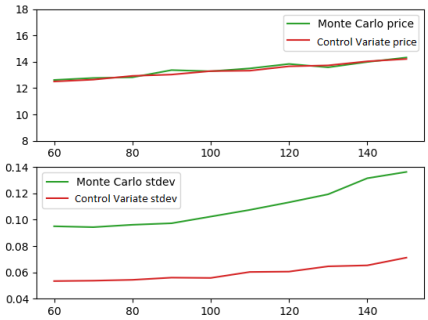
Where C_i is the value of the cash coupon at the date t_i, P_{t_i} is the performance of the underlyings at date t_i (in this context, we consider a basket performance type), AB_i is the AutoCall barrier at date t_i. If at some recall date, the performance goes above the AutoCall barrier, the client receives the corresponding coupon, and the product life ends immediately. At the maturity if the performance goes very low (lower than the Down And In Barrier DB), the client pays a put on the basket with strike K.
We train the network with the following set of parameters: $T = 3$ years, recall dates every year. $C_1 = 20.0, C_2 = 25.0, C_3 = 30.0$, $AB_i = 100.0 \ \forall i$, $K = 80.0$, and $DB = 40.0$. However this network will be tested on parameters that it has never seen. The results are shown in Figure 5



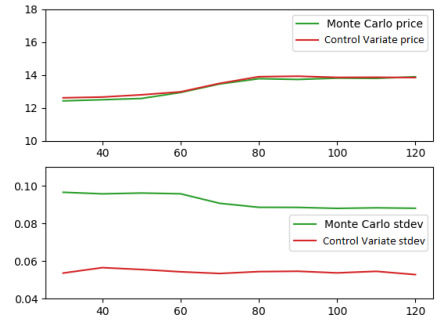
(a) AutoCall price and stdev in function of the AutoCall Barrier



(b) AutoCall price and stdev in function of the spot



(c) AutoCall price and stdev in function of the PDI strike



(d) AutoCall price and stdev in function of the PDI barrier

Figure 7: Control Variate for pricing Autocall

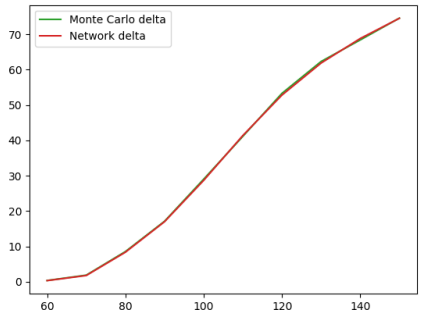
The interest of the dimension reduction method lies in its resistance to parameters change. In fact, this is due to the fact that we learn the directions where the variance of the payoff is concentrated rather than the actual payoff/model. The previous examples show that by learning a certain payoff given fixed parameters, the obtained network can be used to price the same payoff with different parameters.

6.3 Some Greeks profiles

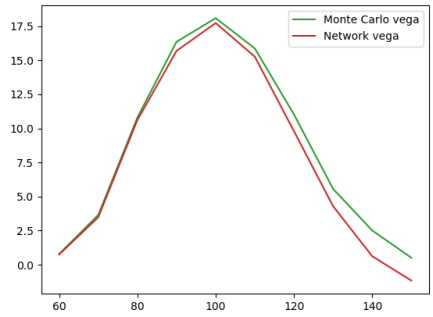
Since the dimension reduction control variate is resistent to some parameters changes, we will use it to compute some greeks which are quantities representing the sensitivity of the payoff to some parameters. We consider for this section the basket option described earlier. We compute by finite difference some of the option's Greeks using the network. Notice that these Greeks are computed with the raw network without using it as a control variate. We compare them with the Greeks given by Monte Carlo.
We define the following Greeks:

- Delta $\delta_i = \frac{\partial \Pi}{\partial S^i}$ where Π is the option price and S^i the i'th underlying.
- Vega $\nu_i = \frac{\partial \Pi}{\partial \sigma^i}$ where σ^i the i'th underlying volatility.

These Greeks profiles are shown in Figure 6



(a) Delta as a function of the spot



(b) Vega as a function of the spot

Figure 8: Basket Greeks using dimension reduction control variate

The Greeks profiles are very smooth and fit nicely to the curves given by Monte Carlo. This shows that the method can not only be used to reduce the variance of Monte Carlo price, but also in Greeks computation.

7 Conclusion

We have presented three new techniques for creating control variates using neural networks. We have tested our methods under different models and showed through numerical examples that the control variates allow significant speed ups without being time consuming. The first method we suggested represents a major quality of resistance to parameters change. So even if in some tests the method had us believing that it works better with payoffs that are close to being linear, we still highly recommend using it for this aspect of robustness to parameters change that it has. We have proven through numerical examples that even for highly complicated payoffs (such as the Autocall example) the method is still efficient. The other two methods are much more intuitive since we use a given network to approximate a payoff function. they work better with non linear functions and give very important speed ups. The last method which uses only one hidden layer is less efficient with complicated models but it has the advantage of being very fast because the expectation of the control can be computed analytically.

References

[1] Cilimkovic, M. (2010). Neural Networks and Back Propagation Algorithm. *Fett.Tu-Sofia.Bg*, pages 3–7.

[2] De Luigi, C., Lelong, J., and Maire, S. (2016). Robust adaptive numerical integration of irregular functions with applications to basket and other multi-dimensional exotic options. *Applied Numerical Mathematics*, 100.

[3] Glynn, P. W. and Szechtman, R. (2002). Some New Perspectives on the Method of Control Variates. In *Monte Carlo and Quasi-Monte Carlo Methods 2000*.

[4] Kebaier, A. and Lelong, J. (2018). Coupling Importance Sampling and Multilevel Monte Carlo using Sample Average Approximation. *Methodology and Computing in Applied Probability*, 20.

[5] Kim, S. and Henderson, S. G. (2004). Adaptive control variates. In *Proceedings - Winter Simulation Conference*.

[6] Kim, S. and Henderson, S. G. (2007). Adaptive control variates for finite-horizon simulation. *Mathematics of Operations Research*, 32.

[7] Kingma, D. P. and Ba, J. L. (2015). Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR.

[8] Kůrková, V. (1992). Kolmogorov's theorem and multilayer neural networks. *Neural Networks*, 5(3):501–506.

[9] Lapeyre, B. and Lelong, J. (2011). A framework for adaptive Monte Carlo procedures. *Monte Carlo Methods and Applications*, 17.

[10] Maire, S. (2003). Reducing variance using iterated control variates. *Journal of Statistical Computation and Simulation*, 73.

[11] Pellizzari, P. (2001). Efficient Monte Carlo pricing of European options using mean value control variates. *Decisions in Economics and Finance*, 24.

[12] Portier, F. and Segers, J. (2019). Monte Carlo integration with a growing number of control variates. *Journal of Applied Probability*, 56.

[13] Sobol, I. M. (2001). Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates. *Mathematics and Computers in Simulation*, 55.

[14] Wang, X. and Fang, K.-T. (2003). The effective dimension and quasi-Monte Carlo integration $. *Journal of Complexity*, 19:101–124.

[15] Wang, X. and Sloan, I. H. (2005). Why are high-dimensional finance problems often of low effective dimension? *SIAM Journal on Scientific Computing*, 27(1).