Unbiased Simulation of Stochastic Differential Equations: A Comprehensive Study

Nathaniel Cogneaux - Paul-Emile Galine

January 14, 2024

Abstract

This project focuses on implementing and analyzing Pierre-Henry Labordère's X. Tan, and N. Touzi method's [1] for constructing an unbiased Monte Carlo estimator for E(g(X)), where X follows a Stochastic Differential Equation (SDE). The primary objectives include replicating selected results from the original paper, providing a detailed explanation of the paper's approach, and conducting a comparative analysis with the traditional Euler Scheme. This study aims to enhance the understanding of unbiased estimators in SDEs and explore their potential advantages over classical numerical methods.

Contents

1 Introduction

The main problem we have to deal with is to compute a Monte Carlo estimator for the following value:

$$V_0 := \mathbb{E}[g(X_{t_1}, \ldots, X_{t_n})], \tag{1}$$

for some function $g : \mathbb{R}^{d\times n} \to \mathbb{R}$ and a discrete time grid $0 < t_1 < \ldots < t_n = T$. Here, X is the strong solution of the Stochastic Differential Equation (SDE):

$$X_0 = x_0, \quad dX_t = \mu(t, X_t)\, dt + \sigma(t, X_t)\, dW_t. \tag{2}$$

- $d \geq 1$ is the dimensionality of the process.
- W is a d-dimensional Brownian motion.
- $\mu : [0, T] \times \mathbb{R}^d \to \mathbb{R}^d$ and $\sigma : [0, T] \times \mathbb{R}^d \to \mathcal{M}_d$ are the drift and diffusion coefficients, respectively. We assume these functions satisfy the standard Lipschitz and growth conditions required for the existence and uniqueness of a strong solution to the Stochastic Differential Equation.
- $T > 0$ is the time horizon.
- $x_0 \in \mathbb{R}^d$ is the initial condition.

For such problems, the standard Monte Carlo approach involves running N independent and identically distributed (i.i.d.) copies of an approximation of X and then estimating V_0 as the empirical average of $g()$ applied to these simulations. Then, thanks to the Law of Large Numbers, we expect this approximation to converge to V_0. This approximation is typically achieved through a Euler Scheme method. However such method is introducing two types of errors:

- A statistical error, described by the Central Limit Theorem, which is of order $\frac{1}{\sqrt{n}}$ for the Euler Scheme.
- A bias error, which is of order $\frac{C}{m}$, where m is the number of steps in the Euler scheme and C a constant.

In contrast to the conventional approach, this paper presents an unbiased approximation of V_0 with finite variance, applicable to a broad spectrum of SDEs. In other words, the algorithm in [1] provide a computable/simulatable random variable ψ such that $V_0 = \mathbb{E}[\psi]$. Therefore, given $N \in \mathbb{N}^*$ samples and i.i.d. random variables $(\psi_n)_{n=1}^N \sim \psi$, the estimator $\hat{V}_0^N = \frac{1}{N}\sum_{n=1}^N \psi_n$ is an unbiased estimator for V_0.

2 The Markovian Case

2.1 Mathematical Framework

In the following, we focus the study on SDEs with a constant diffusion coefficient, denoted by $\sigma_0 \in \mathcal{M}_d$. The diffusion coefficient σ_0 is assumed to be nondegenerate.
To introduce the main algorithm and for the sake of simplicity we focus first on the Markovian case, characterized by a single time point $n = 1$ in the time grid $t_0, \ldots, t_n$. So, the objective is to evaluate:

$$V_0 := \mathbb{E}[g(X_T)], \tag{3}$$

where X follows the SDE defined as

$$X_0 = x_0, \quad dX_t = \mu(t, X_t)\, dt + \sigma_0\, dW_t, \tag{4}$$

With the drift $\mu : [0, T] \times \mathbb{R}^d \to \mathbb{R}^d$. The function μ must be bounded, continuous, uniformly $\frac{1}{2}$-Hölder continuous in time t, and uniformly Lipschitz in space x, thus satisfying the condition:

$$\|\mu(t, x) - \mu(s, y)\| \leq L(\sqrt{|t - s|} + \|x - y\|) \quad \forall (s, x), (t, y) \in [0, T] \times \mathbb{R}^d, \tag{5}$$

for some constant $L > 0$.

2.2 The Unbiased Simulation Algorithm for the Markovian case

To introduce the unbiased simulation algorithm, we first define a random discrete time grid on the desired interval [0, T]. Let $\beta > 0$ be a fixed positive constant and $(\tau_i)_{i>0}$ a sequence of i.i.d. random variables following an $\mathcal{E}(\beta)$ exponential distribution. Then the time grid is defined as:

$$T_k := \left(\sum_{i=1}^{k} \tau_i \right) \wedge T, \quad k \geq 0, \tag{6}$$

We then denote $T_0 := 0$ and $\Delta T_{k+1} := T_{k+1} - T_k$. We also need the maximum index k, N_T, for which $T_k < T$, thus given by

$$N_T := \max\{k : T_k < T\}. \tag{7}$$

In particular, note that $T_{N_T} < T$ and $T_{N_T+1} = T$.
Once we have defined this random time grid, the goal is to do a sort of Euler Scheme method on it:
The process $\hat{X}$ is defined as the Euler scheme solution of X on the grid $(T_k)_{k\geq 0}$, that is:

$$\hat{X}_{T_{k+1}} := \hat{X}_{T_k} + \mu(T_k, \hat{X}_{T_k})\Delta T_{k+1} + \sigma_0 \Delta W_{T_{k+1}}, \quad k = 0, 1, \ldots, N_T. \tag{8}$$

The estimator of the Unbiased Simulation Algorithm ψ is then computed in the following way:

$$\psi := e^{\beta T} \left[g(\hat{X}_T) - g(\hat{X}_{T_{N_T}}) \mathbf{1}_{\{N_T > 0\}} \right] \beta^{-N_T} \prod_{k=1}^{N_T} \mathcal{W}_k^1, \tag{9}$$

Where $\mathcal{W}_k^1$ represents a weight function defined as

$$\mathcal{W}_k^1 := \frac{(\mu(T_k, \hat{X}_{T_k}) - \mu(T_{k-1}, \hat{X}_{T_{k-1}})) \cdot (\sigma_0^T)^{-1} \Delta W_{T_{k+1}}}{\Delta T_{k+1}} \tag{10}$$

We now give the main result of the paper:

Theorem 1. *Let V_0 be defined as above, with the function g assumed to be globally Lipschitz. Then it is unbiased: $V_0 = \mathbb{E}[\hat{\psi}]$, with finite variance, i.e. $\mathbb{E}[\hat{\psi}^2] < \infty$.*

Proof. See [1] for the detailed proof. □

The theorem demonstrates that $\hat{V}_0^N = \frac{1}{N}\sum_{n=1}^{N} \psi_n$, with i.i.d. random variables $(\psi_n)_{n=1}^N \sim \psi$, is an unbiased estimator for V_0 with finite variance. Applying the Central Limit Theorem to this unbiased estimator with finite variance, we obtain:

- A 95% confidence interval IC for V_0, defined as

$$IC = \left[\hat{V}_0^N - 1.96 \times \frac{\hat{\sigma}_N}{\sqrt{N}}, \hat{V}_0^N + 1.96 \times \frac{\hat{\sigma}_N}{\sqrt{N}}\right], \tag{11}$$

  where $\hat{\sigma}_N$ is the standard deviation estimator, defined as

$$\hat{\sigma}_N = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}\left(\psi_i - \hat{V}_0^N\right)^2}, \tag{12}$$

  and 1.96 is the 0.05 quantile of the standard normal distribution $N(0,1)$.

- A statistical error, expressed as

$$\sqrt{\frac{\mathrm{Var}(\hat{\psi})}{N}}, \tag{13}$$

  where N is the number of Monte Carlo simulations.

2.3 Choosing the right β

Until now we did not mention the choice of β. Thus, we aim to justify the right choice rigorously. In practice we focus on minimizing two key aspects:

1. *The Computational Cost of the US Method:* Given that N_T is a Poisson process with parameter βT, the computational cost can be expressed as $\mathbb{E}[N_T] \times C = \beta T \times C$, where C is a constant depending on the number of simulations.

2. *The Statistical Error:* This is given by $\sqrt{\frac{\mathrm{Var}(\hat{\psi})}{N}}$.

So, optimizing β can be reduced to minimizing the product of the variance bound and the mean computational effort. From the proof of Theorem 1 , we know that $\mathbb{E}[\psi^2] \leq C' e^{2\beta T} e^{-\beta T + \frac{\gamma L^2 T}{\beta}}$ (where C', L, γ are positive constants independent of β and precisely defined in [1]). Therefore, the objective is to minimize the function $f(\beta) := \beta T \exp\left(T(\beta + \frac{\gamma L^2}{\beta})\right)$, as the other constants mentioned are independent of β. Direct computation shows that the optimal β, denoted β^*, is uniquely solved by $f'(\beta) = 0$ on $(0, \infty)$, yielding:

$$\beta^* = \sqrt{\gamma L^2 T + \frac{1}{4T^2}} - \frac{1}{2T}. \tag{14}$$

2.4 Our pseudo code for the Markovian Case

In this part, we present a clearer and more elaborate approach to applying the Unbiased method practically. Nonetheless, we do not claim this approach to be the most efficient or quickest method available for implementation. The implementation details are available on our GitHub repository, which can be accessed by clicking here.

1. Random Time Grid Generation

```
Function RandomTimeGrid(β, T):
    Initialize a list l_T with the first element as 0
    Generate exponential variable sumTau with parameter β

    While sumTau < T:
        Append sumTau to l_T
        Add exponential variable with parameter β to sumTau

    Append T to l_T
    Calculate N_T as the length of l_T minus 1

    Return l_T and N_T
```

2. Unbiased Simulation - Markovian Case (1-dimensional)

```
Function Unbiased_Simulation_Markovian_Case_1D(
                funcG, X_0, funcMu, Σ, β, T):

    Get random grid arrTimeGrid, N_T using RandomTimeGrid(β, T)
    Compute arrDeltaT as difference between consecutive elements in
    arrTimeGrid

    Initialize X_hat array of size N_T + 2, set first element to X_0
    Initialize arrDeltaW array of size N_T + 1

    For each index in arrDeltaW:
        Generate normal variable with mean 0, stddev sqrt(arrDeltaT[i])

    For k from 0 to N_T:
        Update X_hat[k + 1] using Euler scheme formula

    If N_T > 0:
        Initialize prodW1 to 1
        For k from 1 to N_T:
            Update prodW1 using W_k^1 formula

        Compute and return the estimator

    Else:
        Compute and return the estimator for N_T = 0
```

3. Unbiased Simulation - Markovian Case (Multi-dimensional)

```
Function Unbiased_Simulation_Markovian_Case_MultiD(
                funcG, X_0, funcMu, Σ, β, T, nDim):

    (Similar to 1−dimensional case with multi−dimensional adjustments)
    Initialize X_hat and arrDeltaW as two−dimensional arrays

    For each time step k:
        Update X_hat[k+1] using multi−dimensional Euler scheme

    (Remaining steps similar to 1−dimensional case with
    dimensional adjustments such as matrix multiplications instead of
    simple multiplications)
```

4. Monte Carlo Estimator

```
Function MC_estimator(funcG, X_0, funcMu, Σ, β,
                      T, nDim, nSamples):

    Initialize psi_hats array of size nSamples

    If nDim > 1:
        For each sample i:
            Call Unbiased_Simulation_Markovian_Case_MultiD
            Store result in psi_hats[i]
    Else:
        For each sample i:
            Call Unbiased_Simulation_Markovian_Case_1D
            Store result in psi_hats[i]

    Calculate mean p and stddev s of psi_hats
    Calculate confidence interval and statistical error

    Return p, confidence interval, and statistical error
```

2.5 Numerical Results and Comparison with Euler-Scheme

In the following, we aim to apply the method to a numerical example and compare it with a traditional approach: a Euler Scheme method. We focus on the one-dimensional case (i.e., $d = 1$), where W is a Brownian motion and X is a process following the SDE:

$$X_0 = 0, \quad dX_t = \left(0.1(\sqrt{e^{X_t}} - 1) - \frac{1}{8}\right) dt + \frac{1}{2} dW_t. \tag{15}$$

Our objective is to compute a call option payoff:

$$V_0 := \mathbb{E}[\left(e^{X_t} - K\right)^+], \tag{16}$$

where we set respectively the strike price and maturity date: $K = 1$ and $T = 1$

2.5.1 Estimating V_0 with an Euler-Scheme

The Euler Scheme is a standard numerical approach for approximating solutions to these kind of stochastic differential equations. Consider a discretization of the time interval $[0, T]$ into m equal parts, each with a length $\Delta = \frac{T}{m} = 0.1$, and define $t_k = k\Delta = \frac{k}{10}$. The Euler approximation of our SDE at each discrete time point t_{k+1} is given by the iteration:

$$\bar{X}_{t_{k+1}} = \bar{X}_{t_k} + \mu(\bar{X}_{t_k})\Delta + \sigma(W_{t_{k+1}} - W_{t_k}), \tag{17}$$

where $\bar{X}_{t_0} = x_0 = 0$, μ is the drift term (here, $\mu(x) = 0.1(\sqrt{e^x} - 1) - \frac{1}{8})$, and σ (here, σ=0.5) is the diffusion coefficient and W a standard Brownian motion.

We then evaluate $V_0^\Delta := \mathbb{E}[g(\bar{X}_T)]$ using a standard Monte Carlo method:
This involves simulating an independent and identically distributed (i.i.d.) sequence of random variables that follow the distribution of $\bar{X}_T$. The estimate for V_0^Δ is given by $\frac{1}{N}\sum_{i=1}^{N} g(\bar{X}_{T,i})$, where N is the number of simulations, and $\bar{X}_{T,i}$ represents the ith simulated value of $\bar{X}_T$.
Once we have our approximation of V_0^Δ, this provides a biased estimate of V_0 as $\bar{X}_T$ is an approximation of X_T.

2.5.2 Numerical Results

We implemented both Euler Scheme Method and Unbiaised Simulations (US) approaches for the computation of V_0 for the example defined above 15. Using different number of simulations and fixing the parameter $\beta = 0.1$ for US as in the paper [1] , we obtained the following results:

Method	Mean value	Statistical error	95% Confidence Interval	Computation time
US (N = 10^4)	0.20860405	0.004339791	[0.20009806, 0.21711004]	0.135535s
Euler Scheme (nSteps = 10^4)	0.20268141	0.004164094	[0.19451979, 0.21084304]	0.475454s
US (N = 10^5)	0.20422262	0.001397635	[0.20148326, 0.20696199]	1.461565s
Euler Scheme (nSteps = 10^5)	0.20521232	0.001316617	[0.20263176, 0.20779289]	4.760887s
US (N = 10^6)	0.20561571	0.000449698	[0.2047343, 0.20649712]	14.002025s
Euler Scheme (nSteps = 10^6)	0.20517708	0.000416448	[0.20436084, 0.20599332]	48.404329s
US (N = 10^7)	0.20561192	0.000142441	[0.20533273, 0.2058911]	2min 20.386787s
Euler Scheme (nSteps = 10^7)	0.20492765	0.000131292	[0.20467031, 0.20518498]	8min 4.982595s
US (N = 10^8)	0.20565379	4.5036e-05	[0.20556552, 0.20574206]	23min 30.145957s
Euler Scheme (nSteps = 10^8)	0.20478412	4.1499e-05	[0.20470278, 0.20486545]	1h 21min 39.45896s

Figure 1: Numerical results for V_0 defined in 16 with $\beta = 0.1$ and m=10 for the Euler-Scheme

Remark: For the Unbiased Simulation (US) approach, the statistical error is computed by: Statistical Error (US) = $\frac{\hat{\sigma}_N}{\sqrt{N}}$*, where* $\hat{\sigma}_N^2$ *is the estimated variance of* ψ*. In contrast, the statistical error for the Euler Scheme is Statistical Error (Euler) =* $\frac{\bar{\sigma}_N}{\sqrt{N}}$*, where* $\bar{\sigma}_N^2$ *is the estimated variance of* $g(\bar{X}_T)$*.*

Our results for US method very closely aligns with the paper [1] except for the computation time of the US method which depends of course on the programming language, the computer and other external variables.

We get almost the same statistical error (taking into account that this is an estimation that is random) and our confidence intervals contains the target value (mean value of the paper) for large N.

2.5.3 Error comparison between Euler-Scheme and US

Concerning the comparison between the Euler-Scheme and US method, we get very similar results for both methods in terms of statistical error which is not surprising as we take the same N for both. Having large N, the statistical error are almost negligible in both methods as can be seen.
However, as opposed to the Unbiased Simulation, the error from the Euler-Scheme method is divided into two parts, a statistical error (of order $\frac{1}{\sqrt{N}}$) and a bias (of order $\frac{1}{m}$).
Here, we made the choice of taking $mSteps = 10$ so looking at the bias is crucial as it is significant compared to the statistical error.
Hence, we decided to compute the bias with a fair approximation, even if we don't know the actual value of $E[g(X_T)]$ we have an unbiased estimator $\hat{V}_0^N = \frac{1}{N}\sum_{n=1}^{N}\psi_n$. Thus, if we consider very large values of N such as $N = 10^8$, the mean value of the US estimator, being unbiased, is a good approximation of $E[g(X_T)]$ since its own statistical error becomes negligible for such high N.
Therefore, we get the following results:

Method	Mean value	Estimated Bias	Statistical error
US (N = 10^4)	0.20860405	0.002950256	0.004339791
Euler Scheme (nSteps = 10^4)	0.20268141	-0.002972379	0.004164094
US (N = 10^5)	0.20422262	-0.00143117	0.001397635
Euler Scheme (nSteps = 10^5)	0.20521232	-0.000441467	0.001316617
US (N = 10^6)	0.20561571	-3.8081e-05	0.000449698
Euler Scheme (nSteps = 10^6)	0.20517708	-0.000476712	0.000416448
US (N = 10^7)	0.20561192	-4.1876e-05	0.000142441
Euler Scheme (nSteps = 10^7)	0.20492765	-0.000726144	0.000131292
US (N = 10^8)	0.20565379	0.0	4.5036e-05
Euler Scheme (nSteps = 10^8)	0.20478412	-0.000869676	4.1499e-05

Figure 2: Numerical results with estimated bias for V_0 defined in 16 with $\beta = 0.1$ and m=10 for the Euler-Scheme

Thanks to the above results, we see that the US method produces a more accurate estimator, without bias and still with comparable statistical errors, than the Euler-Scheme. And this, while staying a faster method.

2.5.4 Computation time comparaison between Euler-Scheme and US

Note that the bias of the Euler-Scheme can be easily reduced by calibrating m, the number of steps within the scheme.
However, even if we pick m in a more efficient setting such as $m \approx \sqrt{N}$ to reduce both errors, we get a very expensive computation time compared to US without producing better estimations (in terms of statistical and bias error).

Indeed, the computation time of the US estimator is proportional to $\mathbb{E}[N_T] \times N$, where N is the number of simulations, and N_T follows a Poisson distribution with parameter βT. Therefore, the computation time for the US method is of the order $\beta T N$.
In contrast, the computation time for the Euler Scheme is of the order Nm, where m is the number of time steps. Thus giving a complexity of $N\sqrt{N}$.
Given that in our example $T = 1$ and since $\beta = 0.05$ is relatively small, we expect the US estimator to converge significantly faster in this context.
This shows the real edge of the US method, in some examples such that β and T are small we get a high accuracy for a small computation time compared to a standard Euler-Scheme.

2.6 Careful with β

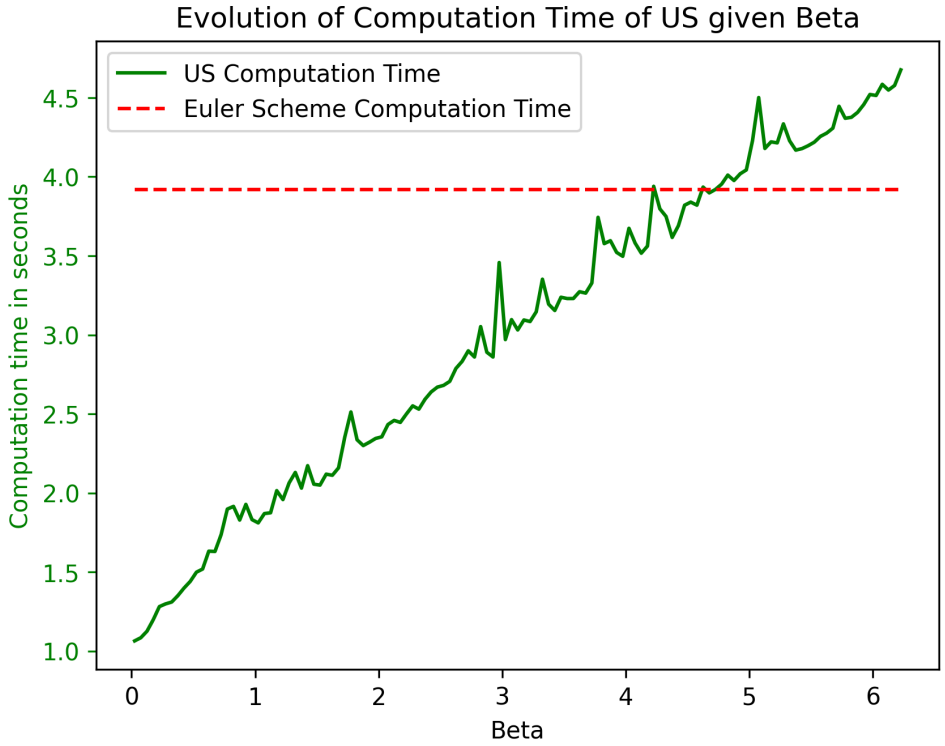
As shown in the computations 14, one should be careful when choosing β, it should not be too big to avoid having a huge computational time nor being too close to zero, otherwise the variance bound could explode. We can observe this in the example treated below where we want to compute:

$$V_0 := \mathbb{E}[\sin(X_T)], \tag{18}$$

where X is defined by the stochastic differential equation (SDE), for some constant $\mu_0 \in \mathbb{R}$,

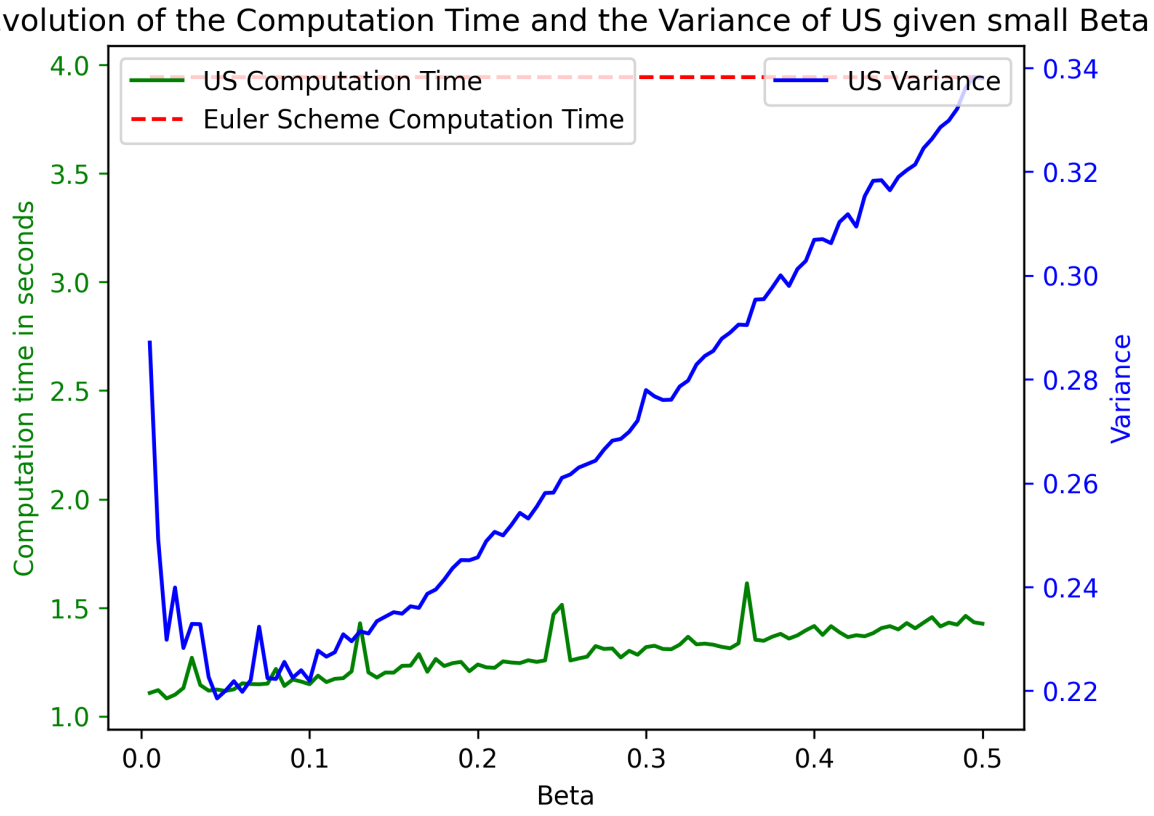
$$X_0 = 0, \quad dX_t = \mu_0 \cos(X_t)\, dt + \frac{1}{2}\, dW_t. \tag{19}$$

Here we set $T = 1$, $\mu_0 = 0.2$, $m = 10$, $N = 10^5$
We can see with the graph below that the computation time can be way worse than the Euler-Scheme for some bad choices of β:



This result is rather logical since we basically use a Euler Scheme method on our random time grid, as this one is proportional to β, so there is necessarily a β that leads to slower computation time than the Euler Scheme whatever the m picked.

In the graph below, we also insist on choosing a β that is not too small as it can increase the variance (therefore the error) of the US estimator:



3 The Path Dependent Case

Now we interest ourselves to a more general problem. We want to compute a path dependent version:

$$V_0 := \mathbb{E}[g(X_{t_1}, \dots, X_{t_n})], \tag{20}$$

For some lipschitz function $g : \mathbb{R}^{d\times n} \to \mathbb{R}$ and a discrete time grid $0 < t_1 < \dots < t_n = T$, where X follows the SDE defined in the Markovian case (4).
More general SDEs are considered in [1] especially with path dependent drifts $\mu(t, X_{t_1\wedge t}, \dots, X_{t_n\wedge t})$ instead of a simple drift $\mu(t, X_t)$. In this paper we will only consider the case of simple drifts for the sake of simplicity but you can find on our Github Repository an adapted version for path dependent drifts using the `funcMu_k` function.

3.1 The Unbiased Simulation Algorithm for the Path Dependent Case

In the path-dependent case, the algorithm given in the paper [1] is simply a recursive iteration of the Markovian case over each time subintervals $[t_k, t_{k+1}]$.
Using the same notations, W is a standard d-dimensional Brownian motion, and $(\tau_i)_{i>0}$ is a sequence of i.i.d. $\mathcal{E}(\beta)$ random variables independent of W. Then $N = (N_s)_{0\le s\le t}$ where $N_t := \max\{k : T_k < t\}$ and $(T_i)_{i>0}$ are defined as previously in 6.
Define further for every subintervals $k = 1, \dots, n$:

- $\tilde{N}_k := N_{t_k} - N_{t_{k-1}}$ the number of jump arrivals on $[t_{k-1}, t_k)$,
- $\tilde{T}_0^k := t_{k-1}$ and $\tilde{T}_j^k := t_k \wedge T_{N_{t_{k-1}}+j}$,
- $\Delta\tilde{T}_j^k := \tilde{T}_j^k - \tilde{T}_{j-1}^k$,
- $\Delta\tilde{W}_j^k := W_{\tilde{T}_j^k} - W_{\tilde{T}_{j-1}^k}$ for all $j = 1, \dots, \tilde{N}_k + 1$.

The above can be seen as the definitions we employed for the Markovian case, where instead of considering $[0, T]$ we consider the subinterval $[t_{k-1}, t_k]$. We next introduce for each subintervals again a process $(\tilde{X}_j^{k,x})$, for all $j = 0, 1, \dots, \tilde{N}_k + 1$, for each $k = 1, \dots, n$ and initial condition $x = (x_0, x_1, \dots, x_{k-1}) \in \mathbb{R}^{d\times k}$ by $\tilde{X}_0^{k,x} := x_{k-1}$ and

$$\tilde{X}_{j+1}^{k,x} := \tilde{X}_j^{k,x} + \mu(\tilde{T}_j^k, \tilde{X}_j^{k,x})\Delta\tilde{T}_{j+1}^k + \sigma_0\Delta\tilde{W}_{j+1}^k. \tag{21}$$

Similarly, for every $j = 1, \dots, \tilde{N}_k$, we define an automatic differentiation weight:

$$\tilde{W}_j^{k,x} := \frac{(\mu(\tilde{T}_j^k, \tilde{X}_j^{k,x}) - \mu(\tilde{T}_{j-1}^k, \tilde{X}_{j-1}^{k,x})) \cdot (\sigma_0^T)^{-1}\Delta\tilde{W}_j^k}{\Delta\tilde{T}_{j+1}^k}. \tag{22}$$

We can now introduce the algorithm for the path-dependent case, in a recursive way with a terminal base case. First, for $x = (x_0, x_1, \dots, x_n) \in \mathbb{R}^{d\times(n+1)}$, set the terminal base case that ends the recursion $\psi_x^{n+1} := g(x_1, \dots, x_n)$.
Next, for $k = 1, \dots, n$, denote $X^{k,x} := (x_0, x_1, \dots, x_{k-1}, \tilde{X}_{\tilde{N}_k+1}^{k,x})$ and
$X_0^{k,x} := (x_0, x_1, \dots, x_{k-1}, \tilde{X}_{\tilde{N}_k}^{k,x}\mathbf{1}_{\{\tilde{N}_k>0\}})$.
Then given ψ_{k+1}, we define recursively

$$\tilde{\psi}_k^x := e^{\beta(t_k - t_{k-1})}(\psi_{k+1}^{X^{k,x}} - \tilde{\psi}_{k+1}^{X_0^{k,x}}\mathbf{1}_{\{\tilde{N}_k>0\}})\beta^{-\tilde{N}_k}\prod_{j=1}^{\tilde{N}_k}\tilde{W}_j^{k,x} \tag{23}$$

We finally obtain the numerical algorithm of the path-dependent case with:

$$\tilde{\psi} := \tilde{\psi}_1^{x_0}. \tag{24}$$

We also mention that the main theorem (1) seen previously still holds true in this context as a recursive version of the latter on each subintervals, with some added assumptions:

Theorem 2. *Suppose that $\mu : [0, T]\times\mathbb{R}^d \to \mathbb{R}^d$ and $g : \mathbb{R}^{d\times n} \to \mathbb{R}$ are differentiable up to the order n, with bounded derivatives. Then we have $V_0 = \mathbb{E}[\tilde{\psi}]$, and the variance of $\tilde{\psi}$ is finite, i.e., $Var(\tilde{\psi}) < \infty$.*

Proof. See [1] for the detailed proof. □

3.2 Our pseudo code for the Path-dependant case

In this part, similar to the markovian scenario, we offer a more detailed implementation for the practical use of the Unbiased technique. However, once again we do not assert that this piece of code is the fastest nor most efficient one for execution. Details on the implementation can be found in our GitHub repository, which is open for access by clicking here.

1. Random Time Grid Generation for Intervals

Function RandomTimeGrid_Interval(β, $t1$, $t2$):

    Initialize a list arr_{t1t2} with the first element as $t1$
    Generate exponential variable $sumTau$ with parameter β starting at $t1$

    While $sumTau < t2$:
        Append $sumTau$ to arr_{t1t2}
        Add exponential variable with parameter β to $sumTau$

    Append $t2$ to arr_{t1t2}
    Calculate N_{t1t2} as the length of $arr_{t1t2} - 1$

    Return arr_{t1t2} and N_{t1t2}

2. Brownian Motion Simulation on Interval

Function BrownianMotionSimulation_Interval(β, $t1$, $t2$):

    Get random grid arr_{t1t2}, N_{t1t2} using RandomTimeGrid_Interval(β, $t1$, $t2$)
    Calculate $arrDelta_{t1t2}$ as difference between consecutive elements in arr_{t1t2}

    Initialize $arrDeltaW_{t1t2}$ array of size $N_{t1t2} + 1$

    For each index in $arrDeltaW_{t1t2}$:
        Generate normal variable with mean 0, stddev $\sqrt{arrDelta_{t1t2}[i]}$

    Return N_{t1t2}, arr_{t1t2}, $arrDelta_{t1t2}$, $arrDeltaW_{t1t2}$

3. Recursive Function for Path-Dependent Case

Function Psi_US_Recursive(k, X_k, X_0, $funcG$, $funcMu$, Σ, β, $lTimeIntervals$):

    If $k = 0$: Raise Error
    ElseIf $k = len(lTimeIntervals)$: Return $funcG(X_k)$

    Get time grid, Brownian motion simulation on interval $[tk_{minus1}, tk]$
    Initialize Xk_{tilde} array, set initial value from $X_k[-1]$

```
For each time step in interval:
    Update Xk_tilde using Euler scheme

If Nk_tilde > 0:
    Initialize prodWk_tilde to 1
    Update prodWk_tilde using W_j^k formula

    Set last values for next recursive call

    Return weighted Psi_US_Recursive for k+1 with updated X_k

Else:
    Append last value of Xk_tilde to X_k

    Return Psi_US_Recursive for k+1 with updated X_k
```

4. Monte Carlo Estimator

```
Function MC_estimator(funcG, X_0, funcMu, Σ, β, lTimeIntervals, nSamples):

    Initialize psi_hats array of size nSamples

    For each sample i:
        Calculate psi_hats[i] using Psi_US_Recursive starting from k=1

    Calculate mean p and stddev s of psi_hats
    Calculate confidence interval and statistical error

    Return p, confidence interval, and statistical error
```

3.3 Numerical Results

In the following, we apply the method to a numerical example and compare it with the traditional Euler Scheme method. We focus our study on the same SDE of the numerical example from the Markovian Case (15).
Our objective is to compute an Asian Call Option payoff:

$$V_0 := \mathbb{E}\left[\left(\frac{1}{n}\sum_{k=1}^{n} e^{X_{t_k}} - K\right)^+\right], \tag{25}$$

where we choose $K = 1$, $T = 1$, $n = 10$ and $t_k := \frac{k}{n}T$.
Once again, we implement both methods Euler-Scheme and US method using the algorithm of the path-dependent case. For the path-dependent case, computations are slower as we have to construct a grid for each subintervals $[t_{k-1}, t_k]$ this time.
Using different number of simulations and fix the parameter $\beta = 0.05$ for US as in the paper [1] , we have obtained the following results :

In the above results, we were able to reproduce the small results as in the paper [1] with once again longer computations time for the same reasons mentioned previously.

Method	Mean value	Statistical error	Estimated Bias	Computation time
US (N = 10^4)	0.12868482	0.002538211	0.002069219	1.175523s
Euler Scheme (N = 10^4)	0.12435946	0.002278843	-0.002256134	0.513582s
US (N = 10^5)	0.12531138	0.000791301	-0.00130422	11.712267s
Euler Scheme (N = 10^5)	0.12623818	0.000733031	-0.000377415	5.057049s
US (N = 10^6)	0.12674004	0.000276576	0.000124446	1min 56.720723s
Euler Scheme (N = 10^6)	0.12613495	0.000231842	-0.000480647	51.38242s
US (N = 10^7)	0.1266156	8.7102e-05	0.0	20min 0.337274s
Euler Scheme (N = 10^7)	0.12605182	7.3353e-05	-0.000563781	8min 50.836699s

Figure 3: Numerical results with estimated bias for V_0 defined in 25 with $\beta = 0.05$ and m=10 for the Euler-Scheme

3.4 Comparison with Euler-Scheme

In our analysis, we set $m = 10$, which inevitably introduced a relatively high bias error. However, for large N, the statistical errors in both methods were negligible. Notably, despite the presence of bias, the error remained acceptably small, highlighting an observation in our study: the computation time. We can observe in this example, that in the path-dependent case, the Unbiased Simulation (US) method is less efficient in terms of computation time compared to a straightforward Euler Scheme. This observation underscores certain limitations of the Unbiased Method, particularly in scenarios involving path dependency. These findings suggest areas for potential optimization of our code and further research into the efficiency of US methods in path-dependent contexts.

4 Generalisation: Unbiased simulation of general SDEs

Until now we only considered SDEs with constant diffusion coefficients. From now on we extend our analysis to more general forms of diffusion coefficients. Consider the SDE given by:

$$X_0 = x_0, \quad \text{and} \quad dX_t = \mu(t, X_t)\, dt + \sigma(t, X_t)\, dW_t. \tag{26}$$

Here, $\mu : [0, T] \times \mathbb{R}^d \to \mathbb{R}^d$ and $\sigma : [0, T] \times \mathbb{R}^d \to M_d$ represent the drift and diffusion coefficients respectively.
The primary focus in this section is to evaluate

$$V_0 = \mathbb{E}[g(X_T)], \tag{27}$$

where $g : \mathbb{R}^d \to \mathbb{R}$ is a given function. We seek a representation of V_0 that aligns with the methodologies described in Section 2 2

4.1 Lamperti's Transformation

Under some assumptions, using the so called Lamperti's transformation, we can simplify the form of a general SDE like (26) to a SDE with constant diffusion coefficient, as in 4. This is particularly applicable in the following scenarios:

- When $d = 1$ and σ is positive and C^1, we define the function $h(t, x)$ as

$$h(t, x) := \int_0^x \frac{1}{\sigma(t, y)}\, dy, \tag{28}$$

with h being $C^{1,2}$ and strictly increasing in x. Its inverse function is denoted by $h^{-1}(t,\cdot)$. Applying Itô's formula, we find that $Y_t := h(t, X_t)$ satisfies an SDE with a constant diffusion coefficient, similar to the one in SDE

Precisely, using Itô's formulae it follows that $Y_t := h(t, X_t)$ satisfies the SDE

$$dY_t = \left(\frac{\partial h}{\partial t}(t, h^{-1}(t, Y_t)) + \frac{\mu(t, h^{-1}(t, Y_t))}{\sigma(t, h^{-1}(t, Y_t))} - \frac{1}{2}\frac{\partial \sigma}{\partial x}(t, h^{-1}(t, Y_t)) \right) dt + dW_t, \tag{29}$$

Where the diffusion coefficient is a constant as in the SDE 4. For $d > 1$ a similar transformation is also possible if sigma is a positive definite matrix.

4.2 Unbiased Estimator Algorithm for general SDEs

It is possible to provide an unbiased simulations algorithm in the same fashion as in the Markovian Case explored in section 2 applying some transformations over the weights in the algorithm.
We make the same kind of assumptions over the drift and sigma as in the Markovian Framework (4) and we use the notations of the section 2.
Let us finally introduce the representation formula for $\hat{\psi}$ as follows:

$$\hat{\psi} := e^{\beta T} \left[g(\hat{X}_T) - g(\hat{X}_{T_{N_T}}) \mathbf{1}_{\{N_T > 0\}} \right] \beta^{-N_T} \prod_{k=1}^{N_T} (\mathcal{W}_k^1 + \mathcal{W}_k^2) \tag{30}$$

Where for each $k = 1, \ldots, N_T$, the weights $\mathcal{W}_k^1$ and W_k^2 are defined by:

$$\mathcal{W}_k^1 := (\mu(T_k, \hat{X}_{T_k}) - \mu(T_{k-1}, \hat{X}_{T_{k-1}})) \cdot (\sigma^T(T_k, \hat{X}_{T_k}))^{-1} \frac{\Delta W_{T_{k+1}}}{\Delta T_{k+1}}, \tag{31}$$

$$\mathcal{W}_k^2 := (a(T_k, \hat{X}_{T_k}) - a(T_{k-1}, \hat{X}_{T_{k-1}})) : \left[(\sigma^T(T_k, \hat{X}_{T_k}))^{-1} \frac{(\Delta W_{T_{k+1}} \Delta W_{T_{k+1}}^T - \Delta T_{k+1} I_d)}{\Delta T_{k+1}^2} \sigma^{-1}(T_k, \hat{X}_{T_k}) \right] \tag{32}$$

Where $A : B = Tr(AB^T)$ for any A, B d-dimensional matrices.

We do not provide in this paper any numerical results, examples of the application or pseudocode to this particular algorithm. However, our detailed implementation is also available on our GitHub repository just by, clicking here.

We then get the following theorem:

Theorem 3. *Suppose we are in the same framework as in the Markovian case, and the function g is Lipschitz continuous. Then the expectation of the absolute value of $\hat{\psi}$ is finite, and the value V_0 is equal to the expectation of $\hat{\psi}$, i.e.,*

$$\mathbb{E}[|\hat{\psi}|] < \infty \quad \textit{and} \quad V_0 = \mathbb{E}[\hat{\psi}]. \tag{33}$$

Proof. See [1] for the detailed proof. □

Despite the integrability of $\hat{\psi}$, the estimator might exhibit infinite variance, which poses significant challenges in numerical computations. This issue is particularly taken care of in [1] for one-dimensional driftless SDEs. In this case the paper introduces an antithetic variable technique to ensure finite variance in these cases. However, this approach depends on strong regularity conditions for μ and σ and is limited to the one-dimensional scenario. Therefore, the generalization to higher dimensions and path-dependent SDEs remains an open area for future research.

5 Conclusion and future work

Through this project, we studied and computed the unbiased estimator method for general SDEs. We decomposed this work step by step from the Markovian Case to Path-Dependant and finally an overview of the method for general SDEs.
Compared to a naive method such as an Euler-Scheme, the US estimator presents significant advantage, being unbiased while having the same statistical error and an equivalent computation time (faster in the Markovian case and a bit slower in the Path Dependent case).

However, in a completely general framework for SDEs, the method provided relies on strong and restrictive assumptions on the SDEs coefficients and on the dimension. As mentioned in [1], a PDE based approximations method could be more competitive and this shows that there is still room for improvements.

5.1 Aspects to explore

- Another aspect we could have explored is a comparison with the Multi Level Monte Carlo method (MLMC) as in the paper [1] which showcases the US method can be as competitive as more sophisticated methods than Euler Scheme.

- The task of developing an unbiased simulation estimator that works effectively for general SDEs with a path dependent payoff remains a significant challenge. Even in the markovian setting, it often results in infinite variance. In an ideal scenario, we should modify the Unbiased Simulation (US) method to ensure finite variance in all cases (easier said than done).

- One can argue that the unbiased simulation for path dependent problems can lead to high computation times when given a very high number of time steps (here, high number of subintervals). In such case deep recursive calls consume a lot of stack memory, that may lead to stack overflow errors. Furthermore, recursive calls involve overhead due to repeated function calls, consuming more time. In contrast, iterative solutions use less memory and often execute faster. Implementing a non-recursive version is doable since it is a standard recursion with a terminal base case. To do so, one could use a simple loop to replace the recursive calls that iterates over the time subintervals. And in order to keep track of the evolution of the $\tilde{\psi}_k^x$ one can for instance save the results in a stack.

- An interesting work to add to this paper would be to compare this method with classical PDEs approximation methods such as the finite-element methods. As mentioned in the paper, we believe that a PDEs approximations method could be more competitive than our methods especially in a one dimensional framework.

References

[1] P. Henry-Labordère, X. Tan, and N. Touzi. *Unbiased simulation of stochastic differential equations.* Ann. Appl. Probab., 27(6):3305–3341, 2017.

[2] Julien Claisse. *Monte Carlo and Finite Difference Methods with Applications in Finance.* December 8, 2021.