Acknowledgements Special thanks to my advisor Dr. Riaz Ahmad, for his tutorship and undying patience with me, it took a lot of time researching and writing this paper. It has been a great joy to study under his guidance and encouragement. This is true for all the teachers of the CQF who lectured and advised this year. I would like to express my gratitude for their enlightening instruction and warm-hearted assistance. I'd also like to thank the references cited in the bibliography section for their research and insight. It was these documents, studies, and readings from which I based the model's design and without the help of the guide from their research; it would not be possible to complete the task.

Abstract

This is a Computational Finance task on the use of the Monte Carlo scheme to price Asian options. For the appropriate form of payoff, to consider both Arithmetic Sampling and Geometric Sampling methods using the Milstein scheme for simulating the underlying stock price. The antithetic variance reduction technique will be used in the simulation. Both examples will include various results of the experiment of a same data set. Conclusions about the outcomes will be described.

Project Objectives

1 Define and observe how options are priced. Methods.

2 Understand the Black-Scholes equation and adapt it to model price European options, in order to model an Asian option pricing model.

3 Implement a basic Monte Carlo simulation of the option to approximate expected payoff in Matlab.

4 Explore different time stepping methods, employing Milstein schemes, to improve the accuracy of the approximation.

5 Implement Antithetic Variance Reduction Methods to illustrate within results any significant change; i.e. a variance reduction or otherwise.

6 Evaluate the accuracy of the implemented models and apply them to Asian Call options.

1 Introduction to Monte Carlo methods

Behaviors of Asset Prices The asset price model from () states that asset prices must move randomly; therefore, the past history is reflected fully in the current price. The markets immediately respond to new information on an asset. Consequently, unanticipated changes on the asset price are classified as a Markov Process.

Monte-Carlo Simulation In option pricing, 'Monte-Carlo' refers to a set of techniques used to generate underlying values of *typically* stock prices or interest rates so we can implement a pricing scheme to value *options* contracts *E payoff*. Typically, the dynamics of these underlying assets are assumed to be driven by a *continuous-time* stochastic process. Despite being continuous, but we'll need to sample over *discrete* time steps in order to simulate an options price that follows our underlying asset.

1.1 Random Nature of Assets

First, we'll assume a random walk defined by the Stochastic Differential Equation:

$$dS_t = S_t(\mu dt + \sigma dW_t) \tag{1}$$

Let S_t be a stock over time. W_t is a Wiener process or Brownian motion, with μ ('the percentage drift') and σ('the percentage volatility') as constants.

Where:
μ is used to model deterministic trends, while the σ term is often used to define a set of unpredictable events occurring during this motion or dispersion of variation.

1.2 Modeling Brownian Simulations

The log of the value of the underlying obeys Brownian motion. Let: $X = lnS$

The dynamics of the underlying asset S is modeled through a geometric Brownian motion (GBM):

$$dS_t = \mu S_t dt + \sigma S_t dX_t \qquad S(0) = S_0 \tag{2}$$

where $X(S)$ is a Brownian motion over $[0, T]$. Written as an integral, the value of the underlying asset at time t is given by:

$$S(t) = S_0 e^{\mu t - \frac{1}{2}\sigma^2 t + \sigma X_t} \tag{3}$$

The Black-Scholes stock price dynamics under the risk neutral measure are:

$$dS_t = rS_t dt + \sigma S_t dW_t \tag{4}$$

If δt represent a very small time step that is theoretically continuous. Where dt is defined discrete time. In order to satisfy Black-Scholes Equation we must assume $\mu \equiv r$. Theory behind the transformation extends the scope of this paper.

1.3 Discrete-time from Continuous

We simulate S_t over the time interval $[0, T]$ which we assume to be discretized as $0 = t_1 < t_2 < ... < t_m = T$. Since time is discretized by equally fixed-width spaces of dt defined by $t_i - t_{i-1}$; we can refer to this discrete time step sampled at equal width, simply as dt from here on.

Integrating dS_t from t to $t + dt$ yields:

$$S_{t+dt} = S_t + \int_t^{t+dt} \mu(S_u, u)du + \int_t^{t+dt} \sigma(S_u, u)dW_u. \tag{5}$$

1.4 MATLAB: Define Time step algorithm

The equation above is the starting point for any discretization scheme. At time t, the value of S_t is known, and we wish to obtain the next value S_{t+dt}.

1.5 The Milstein Scheme

The scheme works for SDEs for which the coefficients $\mu(S_t)$ and $\sigma(S_t)$ depend only on S, and do not depend on t directly. Hence, we assume that the stock price S_t is driven by the SDE:

$$\begin{aligned} dS_t &= \mu(S_t)dt + \sigma(S_t)dW_t \\ &= \mu_t dt + \sigma_t dW_t \end{aligned} \tag{6}$$

[0] all equations, studies, and references for this section are credited to F. Rouah, Kloeden & Platen, and Glasserman. *check Bibligrophy*

If we put into integral form

$$S_{t+dt} = S_t + \int_t^{t+dt} \mu_s ds + \int_t^{t+dt} \sigma_s dW_s \tag{7}$$

We then do expansions, derive at a general solution after applying Ito's Lemma & differentiation defined as:

$$S_{t+dt} = S_t + \mu_t dt + \sigma_t \sqrt{dtZ} + \frac{1}{2}\ \sigma_t' \sigma_t dt(Z^2 - 1) \tag{8}$$

1.6 Milstein Scheme for the Black-Scholes Model

First, we'll define The Black-Scholes stock price dynamics under the risk neutral measure as:

$$dS_t = rS_t dt + \sigma S_t dW_t \tag{9}$$

we have $\mu(S_t) = rSt$ and $\sigma(S_t) = \sigma S_t$. The general solution to Milstein above when differentiated; we get the following Milstein scheme adjusted for a driftless Martingale with risk neutral measure without the μ we get Milstein's BSE:

$$S_{t+dt} = S_t + rS_t dt + \sigma_{S_t} \sqrt{dtZ} + \frac{1}{2}\ \sigma^2 dt(Z^2 - 1) \tag{10}$$

which adds the correction term $\frac{1}{2}\ \sigma^2 dt(Z^2 - 1)$, to the Euler scheme in (5). In the Black-Scholes model for the log-stock price, for the Euler Scheme using BS Frameworks

$$dlnS_t = (r\frac{1}{2}\sigma^2)dt + dW_t \tag{11}$$

, we have $\mu S_t = (r - \frac{1}{2}\ \sigma^2)$ and $\sigma S_t = \sigma$ so that $\mu_t' = \sigma_t' = 0$ The Milstein scheme is therefore:

$$lnS_{t+dt} = lnS_t + (r - \frac{1}{2}\ \sigma^2)dt + \sigma\sqrt{dtZ} \tag{12}$$

which is identical to the Euler discretization scheme for the BS stock price dynamic with risk neutral measure:

1.7 Euler Method

We discussed the Milstein discretization 12 matches the Euler discretization under the Black-Scholes risk neutral measure defined by: 4

Let's assume that $W_{t+dt} - W_t$ and $\sqrt{dt}Z$ are identical in distribution, where **Z** is a standard normal variable. Hence the Euler discretization of asset is defined by the following SDE

dW = randn(1)*sqrt(dt)

$$lnS_{t+dt} = S_t + (r - \frac{1}{2}\sigma^2)dt + \sigma\sqrt{dt}Z \qquad \text{so that.}$$

$$S_{t+dt} = S_t \exp((r - \frac{1}{2}\sigma^2)dt + \sigma\sqrt{dt}Z) \tag{13}$$

$$\text{Where } dt = t_i - t_{i-1}$$

Therefore, while the Milstein scheme improves the discretization of S_t in the Black-Scholes model, it does not improve the discretization of ln S_t.

[0]Sm(n,t)= Sm(n,t-1)+ (r-q)*Sm(n,t-1)*dt + v*Sm(n,t-1)*W...+ 0.5*v^2*Sm(n,t-1)*(Z^2-1)*dt;

1.8 Milstein BSE scheme Matlab

[1]Defines the time step & discretization scheme

```
% Monte Carlo Simulation settings.
N = 50000;          % Number of MC simulations to run.
T = 100;            % Number of t_steps.
dt = mat/T;         % Time increment or dt (t_step)

% Initialization for the terminal stock price matrices
% for the Milstein discretization schemes.
S = zeros(N,T);
S(:,1) = S0;

% Simulate the stock price under the Milstein schemes.
for n=1:N
    for t=2:T
        dW = randn(1)*sqrt(dt);
        S(n,t) = S(n,t-1) + (r-q-v^2/2)*S(n,t-1)*dt + v*S(n,t-1)*dW...
                + 0.5*v^2*S(n,t-1)*dW^2;
    end
```

2 Introduction to Black-Scholes

Before we begin pricing our Asian call options, we must observe the basic intuition behind options contracts and their pricing model. To classify what a financial option is, we look at the simplest European call option. It is a contract with the following conditions and description from $Wilmott$ et al. (1995): At a predetermined time T in the future, or Expiration, the owner of the option may purchase the underlying asset for a predetermined price, also known as the Strike price K. The other party of the option is the writer; they have an obligation to sell the asset to the holder if they want to buy it.

2.1 Basic Notation

C = Call option price

S = Current stock price

K = Strike price of the option

r = risk-free interest rate (a number between 0 and 1)

σ = volatility of the stocks return (a number between 0 and 1)

t = time to option maturity (in years)

N = normal cumulative distribution function

[1]Fabrice Douglas Rouah

2.2 Vanilla options

When $S > K$ at expiry, it would make financial sense to exercise the call option, hence handing over the amount K, obtaining an asset worth S. The profit would be $S - K$. If $S < K$ at expiry, a loss of $K - S$ would be made and the option is worthless. We can write a call option as: $C(S, T) = max(S - K, 0)$.

2.3 Different types of options

Plain Vanillas are typically *European* options which can be exercised only at expiration and *American* options which can be exercised at any time before expiry.

Path-Dependent options are typically referred to as 'Exotics' and unlike Vanilla's they are not often traded on exchanges and must be made over-the-counter. In our study, we'll be looking at one specific type of Path-Dependent option called an *Asian* option that's pay-off is calculated from the average price of the asset over a period of time. There are other path dependent options, such as Lookback options, and other non-path-dependent exotic options, but we'll primarily be focusing on pricing Asian options in this experiment.

2.4 Black-Scholes Equation

The Black-Scholes equation seeks to describe price V of an option over time. The derivation of this equation is excluded, we'll provide a generalized equation.

$$\frac{\partial \mathrm{C}}{\partial \mathrm{t}} + \frac{1}{2}\sigma^2 \mathrm{S}^2 \frac{\partial^2 \mathrm{C}}{\partial \mathrm{C}^2} + \mathrm{rS}\frac{\partial \mathrm{C}}{\partial \mathrm{S}} = \mathrm{rC} \tag{14}$$

Notice that that equation (14) is a partial differential equation. The solution of this equation gives us the Black-Scholes formula.

2.5 Black-Scholes formula for European option price

The Black-Scholes formula allows us the calculate the price of European call and put options.

$$\mathrm{C(S,t)} = \mathrm{N(d_1)S} - \mathrm{N(d_2)Ke}^{-rt} \tag{15}$$

$$\mathrm{d}_1 = \frac{1}{\sigma\sqrt{\mathrm{t}}}\left[\ln\left(\frac{S}{K}\right) + t\left(r + \frac{\sigma^2}{2}\right)\right] \tag{16}$$

$$\mathrm{d}_2 = \frac{1}{\sigma\sqrt{\mathrm{t}}}\left[\ln\left(\frac{S}{K}\right) + t\left(r - \frac{\sigma^2}{2}\right)\right] \tag{17}$$

$$N(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} \mathrm{e}^{-\frac{1}{2}z^2} dz \tag{18}$$

3 Asian Options Overview

We'll be pricing an Exotic Path-Dependent Asian option in our model. Some early uses of these options were used for hedging crude oil futures and $'Asian'$ references it's birthplace of Tokyo. It is $'exotic'$ in the sense that it's pay-off is a function of the underlying asset at multiple points throughout the contract's lifetime, rather than just the value at expiry. This specific exotic uses the *mean* of the underlying asset, we'll call Stock, S_t to price A_t. To do this, we'll observe S_t's price sampled at appropriate equal intervals and use as the basis for our call option's pay-off. This is the intuition behind the term "path-dependent", because S_t is continuous; but the option value at expiration aggregates discrete samplings of S_t and calculates a mean price, this will be our Asian call spot value A_t at expiration E we'll define as A_T.

We'll be focused on the two main types of Asians; the *arithmetic* Asian call and the *geometric* Asian call. We won't bother discussing theoretical continuous pricing schemes since we're focusing on discrete time modeling. The main difference between the *arithmetic* and *geometric* option is how we actually calculate the mean of the underlying values; our method will affect our expected discounted payoff at expiration. We'll focus on a call option only in this example.

Unlike pricing a vanilla European option with a Monte Carlo method, where we only need to generate values of contracts at expiry, with this model we will need to generate multiple spot paths, and then those spot paths must be sampled at correct points to correctly price our *Asian* Call. We will still be modeling our asset price path via a standard Geometric Brownian Motion; Later on, we will implement and discuss *Antithetical variance reduction* methods in our model.

3.1 Arithmetic Pricing

In order to calculate the *Arithmetic* mean A of the spot price we'll implement the following formula:

$$A(0,T) = \frac{1}{N}\sum_{t=1}^{N} S(t_i) \tag{19}$$

Our objective is to determine the value of the Asian option

Black-Scholes method of Asian Options We will begin by defining the Black Scholes model where the dynamics of the underlying asset are:

$$dS = S(\mu dt + \sigma dW_t)$$

3.2 Asian Payoff

The Asian option we consider has the discounted payoff

$$\hat{P} = \exp(-rT), \qquad \max(0, \overline{S} - K) \qquad f = ma;$$

Where

$$\overline{S} = T^{-1}\int_0^T S(t)dt;$$

The simplest approximation of $\overline{S}$, cited by following:

$$\overline{S} = T^{-1} \sum_{0}^{nT-1} \frac{1}{2} h(\hat{S}_n + \hat{S}_{n+1});$$

where $n_T = T/h$ is the number of timesteps. This corresponds to a linear approximation to $S(t)$ but improved accuracy can be achieved by approximating the behavior within a timestep as simple Brownian motion, with constant drift and volatility, conditional on the computed values $\hat{S}_n$. Taking b_n to be the constant volatility within the time interval $[t_n, t_{n+1}]$, standard Brownian Bridge results (see section 3.1 in [Gla04]) give

Arithmetic Payoff

```
function ret = Payoff_ArithmeticAsian(S, K, C)
    mS = mean(S,1);           % mean with respect to one path
    if C==1
        ret = max(mS-K,0);
    else
        ret = max(K-mS,0);
```

3.3 Geometric Pricing

In the same fashion, for *Geometric* mean A of the spot we'll use:

$$A(0,T) = \exp\left(\frac{1}{N}\sum_{i=1}^{N} \log(S(t_i))\right) \tag{20}$$

The geometric is going to be similar to the arithmetic price, but is easier to find once we know the arithmetic average. if $S(t)$ a geometric Brownian motion, then $\hat{A}_T$ is the sum of lognormally distributed random variables. In contrast to the arithmetic average, the distribution of the geometric average can easily be obtained. The geometric mean of n values log. The exponent is a weighted average of the independent normal increments of the process and therefore normally distributed.

Geometric Payoff

```
function ret = GeometricAsian(S, K, C)
% S = NSim x Nt matrix of simulated prices
% K = Strike price
% C = 1 -> Call; C = 0 -> Put
    mS = prod(S(:,2:end).^(1/size(S,2)),2);
    if C==1
        ret = max(mS-K,0);
    else
        ret = max(K-mS,0);
    end
    ret = mean(ret);
```

3.4 Control Variate on Asian

we can estimate $V_1 - \beta(V_2 - E(V_2))$ where $\beta = \frac{\text{cov}(V_1,V_2)}{\text{var}(V_2)}$

We simulate many values of the random variables and replace theV_1, V_2 by their averages which we'll denote as an $\overline{V}_1, \overline{V}_2$ So, with our random negative cumulative distribution function we can correlate two random variates Z & $-Z$ and sum the two. Let's suppose our confidence interval was $\phi = 0.95$, $\mu = 1$, and $\epsilon_t \sim N(0,4)$.

In Matlab, we want to simulate data for this process and then estimate ϕ Then, we want to repeat the simulation and estimation process several times, each time saving our estimate of ϕ. Where the distribution of ϕ's relative to a normal distribution. The last line of code tells you the percentage of estimated ϕ's that are smaller than the actual value. Running this code will show you that there is a lot more probability mass to the left of the actual value, exposing a bias in our estimation

```matlab
  Z = randn(N,1) ;            % We generate a standard Gaussian vectors
  nZ = -Z ;                   % We generate the antithetic random vector

    call = zeros(N,1) ;
    conf = zeros(N,1) ;
    for i=1:N
        % We generate final asset prices from both random vectors
        pS_fin = S * exp((r-0.5*sigma^2)*T + sigma*sqrt(T)*Z(1:i,1)) ;
        nS_fin = S * exp((r-0.5*sigma^2)*T + sigma*sqrt(T)*nZ(1:i,1)) ;

        % We compute the payoff vector for the call for both random vectors
        resCp = max(pS_fin - K, 0) ;
        resCn = max(nS_fin - K, 0) ;

        % We compute the average between normal and antithetic payoffs
        resC = 0.5 * (resCp + resCn) ;

        % We finally discount the average of the payoff
        call(i) = exp(-r*T)*mean(resC) ;

        % We compute the variance and the confidence interval 95%
        var = sum((resC-mean(resC)).^2)/(i) ;
        conf(i) = 1.96*var/sqrt(i) ;
    end
    plot(resCp,'r')
    hold on
    plot(resCn,'b')
    axis([0,50,-1,1])
    ylim auto
    hold off;

%     fprintf ('payoff vector for the call positive Z f%.6\n', resCp);
%     fprintf ('payoff vector for the call negative Z %f.6\n', resCn);
%     fprintf ('average b/twn normal and antithetic payoffs %f.6\n', resC);
    fprintf ( 'number of simulation %g \n', N)
    fprintf ('average mean variance %f\n', var)
    fprintf ( 'confidence interval value %f\n', conf(i))
```

And find the eigenvalues of the matrix:

$$\begin{bmatrix} \phi_1 & \phi_2 \\ 1 & 0 \end{bmatrix} \tag{21}$$

```
quant = 0.95;
xi = norminv(quant);
lb = MCmean -xi* MCstd/2;
ub = MCmean +xi* MCstd/2;
lbCV = CVmean -xi* CVstd/2;
ubCV = CVmean +xi* CVstd/2;

Price_AM = MCmean;
CI_AM = [lb ub];
Price = CVmean;
CI = [lbCV ubCV];
```

3.5 The Milstein scheme discretization & convergence

For a scalar SDE, the Milstein discretization of equation is:

$$\overline{S}_{n+1} = \overline{S}_n + ah + b\Delta W_n + \frac{1}{2}\frac{\partial b}{\partial S} b(\Delta W_n)^2 \tag{22}$$

In the above equation, the subscript n is used to denote the timestep index, and a, b and $\partial b/\partial S$ are evaluated at $\overline{S}_n$,t_n.

All of the numerical results to be presented are for the case of geometric Brownian motion for which the SDE is

$$dS_t = rSdt + \sigma S_t dW_t \qquad 0 < t < T. \tag{23}$$

By switching to the new variable $X = \log S$, it is possible to construct numerical approximations which are exact, but here we directly simulate the geometric Brownian motion using the Milstein method as an indication of the behavior with more complicated models, for example those with a local volatility function $\sigma(S, t)$.

Let S_m represent the Milstein scheme as a function of $(n, t-1)$ where n represents the discrete time step defined by $_{t-1}$ defined by the 2nd time step minus the 1st time step in a looping function $t = 1, 2, ..., n$. In the limit $n \to \infty$ the discrete sampled averages become the continuous sampled averages. The continuous arithmetic average is given by the averaging state variables $A - T$ and G_T

$$A_T = \frac{1}{n}\int_0^T S_m dt \tag{24}$$

while the continuous geometric average is defined as

$$G_T = exp(\frac{1}{T}\int\limits_0^T \ln S_m dt) \tag{25}$$

In Matlab we defined the Milstein discretization of theoretical continuous averaging

$$Sm_{n,t} = Sm_{n,t-1} + (r-q) \times Sm_{n,t-1}dt + v \times Sm_{n,t-1}W \ldots + 0.5v^2 \times Sm_{n,t-1} \times (Z^2-1) \times dt;$$

3.6 Estimator Construction

In all of the cases to be presented, we simulate the paths using the Milstein method. The refinement factor is $M = 2$, so each level has twice as many timesteps as the previous level. The difference between the applications is in how we use the computed discrete path data to estimate $E[\overline{P}_t - \overline{P}_{t-1}]$

In each case, the estimator for $E[\overline{P}_t - \overline{P}_{t-1}]]$ is an average of values from N_t independent path simulations. For each Brownian input, the value which is computed is of the form P f − P c . Here P f is a fine-path estimate using l l−1 l timestep h=2−lT, and P c is the corresponding coarse-path estimate using l−1 timestep h=2−(l−1)T. To ensure that the identity **EQUATION** is correctly respected, to avoid the introduction of an undesired bias, we require that

4 Experiment & Results of MC SIM

Here we will implement a set of fixed variables to run our Monte Carlo Simulation

```
% Inline function for the Black Scholes call
C = inline('s*exp(-q*T)*normcdf((log(s/K) + (r+v^2/2)*T)/v/sqrt(T)) - ...
    K*exp(-r*T)*normcdf((log(s/K) + (r+v^2/2)*T)/v/sqrt(T) - v*sqrt(T))',...
    's','K','r','v','T');
```

MC SIMULATION INPUT VARIABLES

S_0 100 is our Spot price at 0_t

K 100 is our Strike price

r 0.05 is our annualized risk free rate

σ 0.20 is our volatility (sigma)

E 1 year is the contract T_{mat} maturity

MATLAB Implementation of the above in our model we have something like this

```
% Option features.
S0        = 100;          % Spot price
K         = 100;          % Strike price
r         = 0.05;         % Risk free rate
v         = 0.20;         % Volatility
mat       = 1.0;          % Time to maturity
Averaging = 'A';          % 'A'rithmetic or 'G'eometric
```

4.1 Monte Carlo Simulation Settings MATLAB EX

```
%==============================================
% Monte Carlo Simulation settings.
%==============================================

N = 1000000;            % Number of simulations.
T = 252;                % Number of time steps 252 for trading days year.
dt = mat/T;             % Discrete Time increment  (dt = 1/252)
```

4.2 Milstein Scheme MATLAB ex

```
% Initialize the terminal stock price matrices
% for the Milstein discretization scheme.
Sm = zeros(N,T);            % Milstein
Sm(:,1) = S0;

% Simulate the stock price under the Milstein schemes
for n=1:N
    for t=2:T
        Z = randn(1);
        W = sqrt(dt)*Z;
                Sm(n,t) = Sm(n,t-1) + (r-q)*Sm(n,t-1)*dt + v*Sm(n,t-1)*W...
                 + 0.5*v^2*Sm(n,t-1)*(Z^2-1)*dt;
        end
```

4.3 Arithmetic & Geometric Pricing MATLAB ex

```
        if strcmp(Averaging,'A')
            A(n) = mean(Sm(n,:));
        elseif strcmp(Averaging,'G')
            A(n) = exp(mean(log(Sm(n,:))));
        end
```

4.4 Milstein Sampling of the Stock

```
STm = Sm(:,end);          % Milstein stock price
```

4.5 Calculating the Asian Payoff MATLAB ex

```
% Calculate the price of the Asian option.
 AsianPrice = exp(-r*mat)*mean(AsianPayoff)       % Asian Payoff Output
 Numberofsims = N                                 % Number of Sims Output
 NumberofTsteps = T                               % Number of Steps Output
```

5 Improved multilevel Monte Carlo convergence using the Milstein scheme

In this section we'll show the Milstein scheme can be used to improve the convergence of the multilevel Monte Carlo method for scalar stochastic differential equations. Numerical results for Asian, lookback, barrier and digital options demonstrate that the computational cost to achieve a root-mean-square error of ϵ is reduced to $O\left(\epsilon^{-2}\right)$. This is achieved through a careful construction of the multilevel estimator which computes the difference in expected payoff when using different numbers of timesteps.(Giles, 2008)

5.1 Error correction in Milstein scheme

with given initial data S_0. In the case of European and digital options, we are interested in the expected value of a function of the terminal state, $f(S(T))$, but in the case of path-dependent options like Asian options and other similar exotics, the valuation depends on the entire path $S(t)$, where $0 < t < T$.

Using a simple Monte Carlo method with a numerical discretization with first order weak convergence, to achieve a root-mean-square error of ϵ would require $O\left(\epsilon^{-2}\right)$ independent paths, each with $O\left(\epsilon^{-1}\right)$ timesteps, giving a computational complexity which is $O\left(\epsilon^{-3}\right)$. We have recently introduced a new multilevel approach [Gil06] which reduces the cost to $O\left(\epsilon^{-2}(log\epsilon)^{-2}\right)$ when using an Euler path discretization for a European option with a payoff with a uniform Lipschitz bound. This multilevel approach is related to the two-level method of Kebaier [Keb05], and is similar to the multilevel method proposed.

The objective of this paper is to demonstrate that this improved complexity is attainable for scalar SDEs with a variety of exotic options through using the Milstein path discretization. it can be proved that this an immediate consequence of the improved strong order of convergence of the Milstein discretization compared to the simpler Euler discretization. However, for Asian, lookback, barrier and digital options, special numerical treatments have to be introduced, and that is the focus of the paper. Furthermore, no *a priori* convergence proofs have yet been constructed for these cases and so the paper relies on numerical demonstration of the effectiveness of the algorithms that have been developed.

The paper begins by reviewing the multilevel approach, and the theorem which describes its computational cost given certain properties of the numerical discretization. The next section discusses the **Milstein discretization** and the challenges of achieving higher order variance convergence within the multilevel method. Asian, lookback, barrier and digital options are all considered, and $O\left(\epsilon^{-2}\right)$. computational cost is demonstrated for each through the use of Brownian interpolation to approximate the behavior of paths within each timestep.

The final section indicates the direction of future research, including the need for *a priori* convergence analysis, the challenges of extending this work to multidimensional SDEs, and the use of quasi-Monte Carlo methods for further reduction of the computational complexity.

6 Multilevel Monte Carlo method

Consider Monte Carlo path simulations with different timesteps$h_l = 2^{-l}T$, $l = 0, 1, ..., L$. Thus on the coarsest level, $l = 0$, the simulations use just 1 timestep, while on the finest level, $l = L$, the simulations use 2^L timesteps. For a given Brownian path $W(t)$, let P denote the payoff, and let $\hat{P}_t$ denote its approximation using a numerical discretization with timestep h_l. Because of the linearity of the expectation operator, it is clearly true that

$$E[\hat{P}_l] = E[\hat{P}_0] + \sum_{l=1}^{L} E[\hat{P}_l - \hat{P}_{l-1}] \tag{26}$$

This expresses the expectation on the finest level as being equal to the expectation on the coarsest level plus a sum of corrections which give the difference in expectation between simulations using different numbers of timesteps. The idea behind the multilevel method is to independently estimate each of the expectations on the right-hand side in a way which minimizes the overall variance for a given computational cost.

Let $\hat{Y}_0$ be an estimator for $E[\hat{P}_0]$ using N_0 samples, and let $\hat{Y}_l$ for $l > 0$ be an estimator for $E[\hat{P}_l - \hat{P}_{l-1}]$ using N_l paths. The simplest estimator is mean of N_l independent samples, which $l > 0$ is:

$$\hat{Y}_l = N_l{}^{-}1 = \sum_{i=1}^{N_l} (\hat{P}_l^{(i)} - \hat{P}_{l-1}^{(i)}). \tag{27}$$

The key point here is the quantity $\hat{P}_l^{(i)} - \hat{P}_{l-1}^{(i)}$ comes from two discrete approximations with different timesteps but the same Brownian path. The variance of this simple estimator is $V|\hat{Y}_l| = \hat{N}_l^{(-1)} V_l$ where V_l is the variance of a single sample. Combining this with independent estimators for each of the other levels, the variance of the combined estimator $\hat{Y} = \sum_{l=0}^{L} \hat{Y}$ is $V|\hat{Y}| = \sum_{l=0}^{L} \hat{Y} N_l^{-1} V_l$, while it's computational cost is proportional to $\sum_{l=0}^{L} N_l h_l^{-1}$. Treating the N_l as continuous variables the variance is minimized for a fixed computational cost by choosing N_l to be proportional to $\sqrt{V_l h_l}$.

In the particular case of an Euler discretization, provided $a(S,t)$ and $b(S,t)$ satisfy certain conditions there is $O(h_{1/2})$ strong convergence. From this it follows that $V[\hat{P}_l - P] = O(h_l)$ for a European option with a continuous payoff. Hence for the simple estimator 27, the sample variance is V_l is $O(h_l)$, and the optimal choice for N_l is asymptotically proportional to h_l. Setting$N_l = O(\epsilon^{-2} L h_l)$, the variance of the combined estimator $\hat{Y}$ is $O(\epsilon^2)$. If L is chosen such that $L = log\epsilon^{-1}/log2 + O(1)$, as $\epsilon \to 0$, then $h_L = 2^{-L} = O(\epsilon)$, and so the bias error $E[\hat{P}_L - P]$ is $O(\epsilon)$ due to standard results on weak convergence. Consequently, we obtain a Mean Square Error which is $O(\epsilon^2)$, with a computational complexity which is $O(\epsilon{-}2L^2) = O(\epsilon^{-2}(log\epsilon)^2)$,

This analysis is generalized in the following theorem:

Theorem 1. 1 *Let P denote a functional of the solution of stochastic differential equation 23 for a given Brownian path $W(t)$, and let $\hat{P}$ denote the corresponding approximation using a numerical discretization with timestep $h_l = M^{-1} - T$*

If there exist independent estimators $\hat{Y}_l$ based on N_l Monte Carlo samples, and positive constants $\alpha \geq \frac{1}{2}, \beta, c_1, c_2, c_3$ such that

1.
$$E|\hat{P}_l - P| \leq c_1 h_l^{\alpha}$$

2.
$$E|\hat{Y}_l| = \begin{cases} E|\hat{P}_0|, & l = 0 \\ E|\hat{P}_l - \hat{P}_{l-1}|, & l > 1 \end{cases}$$

3.
$$V|\hat{Y}_l| \geq c_2 N_l^{-1} h_l^{\beta}$$

4. C_l, the computational complexity of $\hat{Y}_l$, is bounded by
$$C_l \leq c_3 N_l h_l^{-1}$$

then there exists a positive constant c_4 such that for any $\epsilon < e^{-1}$, there are values L and N_l for which the multilevel estimator
$$\hat{Y}_l = \sum_{l=0}^{L} \hat{Y}_l,$$

has a mean-square-error with bound
$$MSE = E[(\hat{Y} - E|P|)^2] < \epsilon^2$$

with a computational complexity C with bound
$$C \leq \begin{cases} c_4 \epsilon^{-2}, & \beta \geq 1 \\ c_4 \epsilon^{-2} (\log \epsilon)^2, & \beta = 1 \\ c_4 \epsilon^{-2-(1-\beta)/\alpha}, & 0 < \beta < 1 \end{cases}$$

Proof. See M.B. Giles. Multilevel Monte Carlo path simulation. Technical Report NA06/03, Oxford University Computing Laboratory, 2006 (to appear in Operations Research).

The remainder of this paper addresses the use of **the Milstein scheme** [Gla04, KP92] to construct estimators with variance convergence rates $\beta > 1$, resulting in an $O(\epsilon^{-2})$ complexity bound. Provided certain conditions are satisfied [KP92], the Milstein scheme gives $O(h)$ strong convergence. Numerical results which are not presented here demonstrate this convergence rate, and the associated $O(\epsilon^{-2})$ complexity

7 The antithetic Variates technique

Variance reduction is the search for alternative and more accurate estimators of a given quantity. The possibility of variance reduction is what separates Monte Carlo from direct simulation. Simple variance reduction methods often are remarkably effective and easy to implement.

Financial theory implies that the fair price A_C of the Asian option on the arithmetic mean is

$$\hat{A}_C \equiv \text{Asian Call payoff} = E^Q[e^{-rT}(A-K)^+]$$

where the expected value $E^Q[]$ is computed with respect to the risk-neutral probability measure Q, and r is the **risk rate**

Note that we know the model in this example C_A:

- The state variables have been identified
- The goal is to price the option
- The mathematical relationship between the inputs and outputs have been identified

Antithetic variables: Imagine instead of the random number generator you actually used, you generated a $U(0,1)$ variate, call it u, and ran it through the covariance matrix, thereby generating a antithetic variate. For the next random number, use $1{-}u$ instead of generating a new u. For subsequent random numbers, alternate generating a new u and using $1{-}u$

This helps to "balance" high and low values from the random number stream, thus reducing variability of the final estimates. An the process is similar to the the second except there are now two persistence terms associated with the two lagged values of Y. We can write an AR(2) as

A Guassian random walk can be thought of as the sum of a series of random variables that are $iidN(0,\sigma^2)$

in Matlab we generate a random walk path, first choose how many time steps you want the path to be. Next, create a vector for storing the path, choose an initial value and a value for both Z & -Z

$$\begin{bmatrix} y_t \\ y_{t-1} \end{bmatrix} = \begin{bmatrix} \phi_1 & \phi_2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} y_{t-1} \\ y_{t-2} \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \epsilon_t \tag{28}$$

```
function [Y CovY] = Correlation_elimination( X )
Y=[];
mux=mean(X');
Covx=cov(X');
for i=1:size(X,2)
Y(:,i)=(X(:,i)-mux')'*(Covx)^(-0.5);
end
CovY=cov(Y');
```

where the covariance and variance terms are estimated from the data. This helps correct the estimate for random number streams that are not, in some relevant way, totally representative of the underlying distribution.

Both methods, especially control variates, are more general than these two examples might lead you to believe.

$U \sim \mathcal{U}(0,1)$

In statistics, the antithetic variates method is a variance reduction technique used in Monte Carlo methods. Considering that the error reduction in the simulated signal (using Monte Carlo methods) has a square root convergence, a very large number of sample paths is required to obtain an accurate result. The antithetic variates method reduces the variance of the simulation results.

The antithetic variates technique consists, for every sample path obtained, in taking its antithetic path — that is given a path $\{\varepsilon_1, \cdot, \varepsilon_M\}$ to also take it's inverse $\{-\varepsilon_1, \cdot, -\varepsilon_M\}$.

The advantage of this technique is twofold: it reduces the number of normal samples to be taken to generate N paths, and it reduces the variance of the sample paths, improving the accuracy.

Suppose that we would like to estimate

$$\theta = \mathrm{E}(h(X)) = \mathrm{E}(Y)$$

For that we have generated two samples

$$Y_1 \mathrm{and} Y_2$$

An unbiased estimate of θ is given by

$$\hat{\theta} = \frac{\hat{\theta}_1 + \hat{\theta}_2}{2}$$

And

$$\mathrm{Var}(\hat{\theta}) = \frac{\mathrm{Var}(Y_1) + \mathrm{Var}(Y_2) + 2\mathrm{Cov}(Y_1, Y_2)}{4}$$

So variance is reduced if $\mathrm{Cov}(Y_1, Y_2)$ is negative

If the law of the variable X follows a uniform distribution along $[0, 1]$, the first sample will be $U_1, \ldots, U_n$, where, for any given i, U_i is obtained from $\mathcal{U}(0, 1)$. The second sample is built from it's opposite $U'_1, \ldots, U'_n$, where, for any given i: $U'_i = 1 - U_i$.

If the set U_i is is uniform along $[0, 1]$, then the set U'_i is also uniform along $[0, 1]$. Furthermore, covariance is negative, allowing for initial variance reduction.

Therefore, you can implement the **antithetic method** by using a sample of uniforms, $U_1, \ldots, U_n$ and it's compliments $1 - U'_1, \ldots, 1 - U'_n$.

An example compares the classical Monte Carlo estimate (sample size: $2n$, where $n = 1500$) to the antithetic variates estimate (sample size: n, completed with the transformed sample $1{-}U_i$).

8 APPENDIX: Matlab code

8.1 Milstein Code

```matlab
% Milstein discretization for Black-Scholes.
C = inline('s*exp(-q*T)*normcdf((log(s/K) + (r-q+v^2/2)*T)/v/sqrt(T)) - ...
    K*exp(-r*T)*normcdf((log(s/K) + (r-q+v^2/2)*T)/v/sqrt(T) - v*sqrt(T))',...
    's','K','r','q','v','T');
% Option features.
S0        = 100;              % Spot price
K         = 100;              % Strike price
r         = 0.05;             % Risk free rate
q         = 0.0;              % Dividend yield
v         = 0.20;             % Volatility
mat        = 1.0;                % Time to maturity in years
Averaging = 'A';              %  'A'rithmetic or 'G'eometric
%==========================================
% Monte Carlo Simulation settings.
%==========================================
N = 1000;             % Number of simulations.
T = 365;               % Number of time steps.
dt = mat/T;            % Discrete Time increment
Sm = zeros(N,T);             % Milstein terminal
Sm(:,1) = S0;
% Simulate the stock price under the Milstein schemes
for n=1:N
    for t=2:T
        Z = randn(1);
        W= sqrt(dt)*Z;
        Sm(n,t) = Sm(n,t-1) + (r-q)*Sm(n,t-1)*dt + v*Sm(n,t-1)*W...
                 + 0.5*v^2*Sm(n,t-1)*(Z^2-1)*dt;
        end
        if strcmp(Averaging,'A')
            A(n) = mean(Sm(n,:));
        elseif strcmp(Averaging,'G')
            A(n) = exp(mean(log(Sm(n,:))));
        end
end
% Terminal stock prices.
STm = Sm(:,end);          % Milstein stock price
% Obtain the simulated option prices.
    BSPrice   = C(S0,K,r,q,v,mat);
    Milstein  = exp(-r*mat)*mean(max(STm-K,0));
    AsianPayoff = max(A - K,  0);
% Calculate the price of the Asian option.
AsianPrice = exp(-r*mat)*mean(AsianPayoff);
        fprintf('Number of simulations: %f\n',N)
% Calculate the errors.
MilsteinError   = abs(Milstein - BSPrice)/BSPrice * 100;
        fprintf('The Milstein Error is: %f\n',MilsteinError)
    if strcmp(Averaging,'A')
        fprintf('Arithmetic call price is: %f\n',AsianPrice)
    elseif strcmp(Averaging,'G')
        fprintf('Geometric call price is: %f\n',AsianPrice)
    end
```

8.2 INSERT VARIANCE REDUCTION RESULTS

```
function [Price_AM,CI_AM, Price ,CI] = AsianCall_mc_cv(S0,K,r,T,vol,Simu)
S0 =   100;
K =    100;
r = 0.05;
T = 1;
vol = 0.20;
Simu = 1000000  ;

Total_time =round(T*365)+1;
dt= T/Total_time;
yield =0;
R = exp(-r*T);
     for j = 1:Simu,   %Simulations
                m = (r - yield - vol^2/2)*dt;
                s = vol*sqrt(dt);
                Z= m+s*randn(1,Total_time);
                S = cumsum([log(S0), Z],2);
                arithmetic_mean = mean(exp(S));
                geometric_mean=exp(mean(S));

     Vcall= max([arithmetic_mean-K;zeros(1)]);
     VGcall= max([geometric_mean-K;zeros(1)]);

     MCmean_simulation(1,j)=  Vcall;
     MCVGmean_simulation(1,j)=  VGcall;
     end
     MCmean = mean(MCmean_simulation);
     MCstd = std(MCmean_simulation)/sqrt(Simu);
     muG = 1/2*(r-vol^2/2)*(1+1/Total_time);
     sigmaG = sqrt((vol^2)/3*(1+1/Total_time)*(1+1/(2*Total_time)));
     S0GM = S0*exp(T*((sigmaG^2)/2+muG-r));
     GA_calloption = BS_European_Call(S0GM,K,sigmaG,r,T);
     C = cov(MCmean_simulation,MCVGmean_simulation);
     b = C(1,2)/C(1,1);
     CV = (R*MCmean_simulation) -b*((R*MCVGmean_simulation) - GA_calloption);
     CVmean = mean(CV);
     CVstd = std(CV)/sqrt(Simu);

quant = 0.95;
xi = norminv(quant);
lb = MCmean -xi* MCstd/2;
ub = MCmean +xi* MCstd/2;
lbCV = CVmean -xi* CVstd/2;
ubCV = CVmean +xi* CVstd/2;

Price_AM = MCmean;
CI_AM = [lb ub];
Price = CVmean;
CI = [lbCV ubCV];

fprintf('Number of Simulations %f\n', Simu)
fprintf('Price of Arithmetic Asian Call %f\n', MCmean)
fprintf('Confidence interval of the Arithmetic asian option %f\n', [lb ub])
fprintf('Price of Geometric asian option %f\n', CVmean)
fprintf('Confidence interval of the Geometric asian  %f\n', [lbCV ubCV]);

function output = BS_European_Call(S, K, sigma, r, T);
d1 = (log(S/K)+(r+sigma^2/2)*T)/(sigma*sqrt(T));
d2 = (log(S/K)+(r-sigma^2/2)*T)/(sigma*sqrt(T));
output = S*normcdf(d1)-K*exp(-r*T)*normcdf(d2);
```

(Glasserman, 2004) (Kloeden, Platen, & Schurz, 1997) (L'Ecuyer, 2009) (Black & Scholes, 1973) (Björk, 2004)

References

Björk, T. (2004). *Arbitrage theory in continuous time* (2nd ed ed.). Oxford ; New York: Oxford University Press.

Black, F., & Scholes, M. (1973). The Pricing of Options and Corporate Liabilities. *Journal of Political Economy*, *81*(3), 637–54. Retrieved from `http://EconPapers.repec.org/RePEc:ucp:jpolec:v:81:y:1973:i:3:p:637-54`

Giles, M. (2008). Improved multilevel Monte Carlo convergence using the Milstein scheme. In *Monte Carlo and quasi-Monte Carlo methods 2006. Selected papers based on the presentations at the 7th international conference 'Monte Carlo and quasi-Monte Carlo methods in scientific computing', Ulm, Germany, August 14–18, 2006.* (pp. 343–358). Berlin: Springer.

Glasserman, P. (2004). *Monte Carlo methods in financial engineering* (No. 53). New York: Springer.

Kloeden, P. E., Platen, E., & Schurz, H. (1997). *Numerical solution of SDE through computer experiments* (Corr. 2nd print ed.). Berlin ; New York: Springer.

L'Ecuyer, P. (2009, September). Quasi-Monte Carlo methods with applications in finance. *Finance and Stochastics*, *13*(3), 307–349. Retrieved 2016-10-22TZ, from `http://link.springer.com/10.1007/s00780-009-0095-y` doi: 10.1007/s00780-009-0095-y