CQF | INSTITUTE

Certificate in Quantitative Finance

Exam 2 Report

June 2022 Program

Name: Xiaodong Yang

Date: 29th September 2022

Contents

1 Introduction

1.1 Stochastic Differential Equations

Stochastic processes and especially the theory of stochastic differential equations have played a fundamental role in the theory of option pricing. Black and Scholes (1973) were the first who derived the equilibrium price of an option which gives the owner the right to buy a stock before a certain date at a fixed price. They used a stochastic process to model the price of the stock. It was Merton (1973) who, in his fundamental paper on the theory of rational option pricing, made their arguments mathematically more rigorous and made a more extensive use of stochastic theory. The theory was used to derive a partial differential equation to describe the option price as a function of the stock price and the time that remains till the date on which the option expires.

A convenient way to model the uncertain future prices S(t) is by using stochastic processes. In particular it is often assumed that the process S(t) is the solution of the following stochastic differential equation.

$$dS_t = rS_t dt + \sigma S_t dW_t \tag{1}$$

Let $Y_t = \ln S_t$, implement the $Itô$ Formula, we get

$$dY_t = \frac{\partial Y_t}{\partial t}dt + \frac{\partial Y_t}{\partial S_t}dS_t + \frac{\partial^2 Y_t}{\partial S_t^2}dS_t^2$$

So we have

$$dY_t = (r - \frac{1}{2}\sigma^2)dt + \sigma dW_t$$

Using integral rule

$$Y_t = Y_0 + \int_0^t (r - \frac{1}{2}\sigma^2)ds + \int_0^T \sigma dW_s$$

The closed form solution is

$$S_{t+\delta t} = S_t \exp\left\{\left(r - \frac{1}{2}\sigma^2\right)\delta t + \sigma\phi\sqrt{\delta t}\right\} \tag{2}$$

1.2 Numerical Methods for Stochastic Differential Equations

Methods for the computational solution of stochastic differential equations are based on similar techniques, including Euler-Maruyama scheme, Milstein scheme and closed form solution with different from the speed of convergence, accuracy, and stability, etc.

1.2.1 Euler-Maruyama Method

Stochastic differential equation is represented by below

$$dX_t = a(t, X_t)dt + b(t, X_t)dW_t, \quad X(0) = X_0 \tag{3}$$

The samplest scheme for soving Equation 3 is using the E-M method. That is

$$X_{n+1} = X_n + \int_{t_n}^{t_{n+1}} a(X_s, s)ds + \int_{t_n}^{t_{n+1}} b(X_s, s)dW_s$$

Using Taylor series expansion

$$\int_{t_n}^{t_{n+1}} a(X_s, s)ds \approx a(t_n, X_n) \int_{t_n}^{t_{n+1}} ds = a(t_n, X_n)\delta t$$

$$\int_{t_n}^{t_{n+1}} b(X_s, s)dW_s \approx b(t_n, X_n) \int_{t_n}^{t_{n+1}} dW_s = b(t_n, X_n)\Delta W_n$$

Plug $a(t, X)$ and $b(t, X)$ to solution of EM method

$$X_{n+1} = X_n + a(t_n, X_n)\delta t + b(t_n, X_n)\Delta W_n$$

The Forward Euler-Maruyama method for GBM gives

$$\frac{\delta S_t}{S_t} = \frac{S_{t+\delta t} - S_t}{S_t} \sim r\delta t + \sigma\phi\sqrt{\delta t}$$

i.e

$$S_{t+\delta t} \sim S_t(1 + r\delta t + \sigma\phi\sqrt{\delta t}) \tag{4}$$

1.2.2 Milstein Method

The Milstein scheme add an extra term $o(\Delta t)$ based on the Euler-Maruyama scheme. Fadugba et al. (2013) researched the convergence of the Euler-Maruyama method and Milstein scheme for the solution of stochastic differential equations, which included that Milstein method converges faster to the stochastic solution process and more accurate than Euler Maruyama method.

We shall now look at a method with exact solution

$$dY_t = A(Y_t, t)dt + B(Y_t, t)dW_t \tag{5}$$

can be discretised as

$$dY_{i+1} = Y_i + A\Delta t + B\Delta W_t + \frac{1}{2}B\frac{\partial B}{\partial Y_i}\left((\Delta W_t)^2 - \Delta t\right) \tag{6}$$

Applying Milstein to the above equation, we get

$$S_{t+\delta t} \sim S_t + rS_t\delta t + \sigma S_t\sqrt{\delta t}\phi + \sigma S_t\frac{\partial}{\partial S_t}\sigma S_t \cdot \frac{1}{2}\sigma^2(\phi^2 - 1)\delta t$$

So we have

$$S_{t+\delta t} \sim S_t\left(1 + r\delta t + \sigma\phi\sqrt{\delta t} + \frac{1}{2}\sigma^2\left(\phi^2 - 1\right)\delta t + \cdots\right) \tag{7}$$

1.3 Exotic Options

1.3.1 Asian Options

An Asian option is a special type of option contract, where the payoff is based on the average of prices of underlying asset over some period prior to maturity. Hence, the effect from volatility of underlying asset prices is smooth for Asian options Boyle & Potapchik (2008). There are two types of Asian options in financial markets: fixed strike, where averaging price is used in place of underlying price; and fixed price, where averaging price is used in place of strike Wikipedia (2022*a*).

The payoff of Asian Options with fixed strike price is given by

$$\text{Payoff}_{\text{Call}} = \max(A(0,T) - K, 0) \tag{8}$$

$$\text{Payoff}_{\text{Put}} = \max(K - A(0,T), 0) \tag{9}$$

The payoff of Asian Options with fixed price is given by

$$\text{Payoff}_{\text{Call}} = \max(S(T) - kA(0,T), 0) \tag{10}$$

$$\text{Payoff}_{\text{Put}} = \max(kA(0,T) - S(T), 0) \tag{11}$$

where $A(0,T)$ is the average of underlying stock prices in a specific period. There are two methods to calculate the average of prices: arithmetic and geometric average. Although the arithmetic average is used in practice, it does not have a simple closed-form solution under the standard Black–Scholes assumptions. However, geometric Asian options can be priced in closed form. Moreover, geometric Asian options can be used to improve the performance of the various numerical methods for pricing arithmetic Asian options Boyle & Potapchik (2008).

1.3.2 Binary Options

Binary option is another special contract of exotic options. The contract holder have a profit when underlying asset price is beyond the strike price, otherwise the options wouldn't be exercised. There are two categories of Binary options: asset-or-nothing and cash-or-nothing options. For the first type, the option pays off nothing if the underlying asset price ends up below the strike price, otherwise the payoff is the final asset price at expiry date. For the second type, the option pays off nothing if the underlying asset price ends up below the strike price and pays a fixed amount if it ends up above the strike price Thavaneswaran et al. (2013).

The payoff of Binary Options (Asset-or-Nothing) is given by

$$\text{Payoff}_{\text{Call}} = S_T \times I_{S(T)>k} \tag{12}$$

$$\text{Payoff}_{\text{Put}} = S_T \times I_{S(T)<k} \tag{13}$$

The payoff of Binary Options (Cash-or-Nothing) is given by

$$\text{Payoff}_{\text{Call}} = I_{S(T)>k} \tag{14}$$

$$\text{Payoff}_{\text{Put}} = I_{S(T)<k} \tag{15}$$

where $I_{S(T)>K}$ is an indicator that equals 1 if $S(T) > K$ and zero otherwise. Note that we assume the Binary option is Cash-or-Nothing in this report.

1.3.3 Lookback Options

Lookbacks are defined by the property that their expiry payoffs depend on the realized maximum or minimum of the asset price over a specified time window, called the lookback window Buchen & Konstandatos (2005). Lookback options, in the terminology of finance, are a type of exotic option with path dependency, among many other kind of options. The payoff depends on the optimal (maximum or minimum) underlying asset's price occurring over the life of the option. The option allows the holder to "look back" over time to determine the payoff. There exist two kinds of lookback options: with floating strike and with fixed strikeWikipedia (2022*b*).

The payoff of Lookback Options with fixed strike is given by

$$\text{Payoff}_{\text{Call}} = \max(S_{max} - K, 0) \tag{16}$$

$$\text{Payoff}_{\text{Put}} = \max(K - S_{min}, 0) \tag{17}$$

The payoff of Lookback Options with floating strike is given by

$$\text{Payoff}_{\text{Call}} = \max(S(T) - S_{min}, 0) = S(T) - S_{min} \tag{18}$$

$$\text{Payoff}_{\text{Put}} = \max(S_{max} - S(T), 0) = S_{max} - S(T) \tag{19}$$

where S_{max} is the maximum price of underlying in specific period.

1.4 Monte Carlo Simulations

Monte Carlo Simulation (MCS) is a important technique for option pricing in finance because of its flexibility and easy to implementation. MCS avoid complicated mathematics and have a straightforward implementation conceptually and practically Jabbour & Liu (2011). In practice, MCS are the procedures of sampling the path of underlying asset prices based on mean and volatility of underlying asset price. Generically, it consists of the following three steps Boyle & Potapchik (2008):

1. simulate sample stock paths under the risk-neutral measure
2. evaluate the discounted payoff on each sample path and
3. take the average of all the discounted payoffs

On the downside, Monte Carlo is typically expensive in terms of computation time.

$$V(S,t) = e^{-r(T-t)}\mathbb{E}^{\mathbb{Q}}[\textbf{Payoff}] \tag{20}$$

Python code for closed form solution is as follows

```
class Monte_Carlo_ClosedForm:
    def __init__(self, spot, strike, rate, sigma, T, Nsteps, Nsim):
        random.seed(10000)
        self.spot = spot                    # Initial stock price
        self.strike = strike                # Strike price
        self.rate = rate                    # Risk-free rate
        self.T = T                          # Maturity
        self.sigma = sigma                  # Volatility
        self.Nsteps = Nsteps                # Time steps
        self.Nsim = Nsim                    # Times of simulations
        self._dt_ = self.T / self.Nsteps    # Delta t
        self._St_ = zeros((self.Nsteps, self.Nsim))
        self._St_[0] = self.spot
        for i in range(0, self.Nsteps-1):
            w = random.standard_normal(self.Nsim)
            self._St_[i+1] = self._St_[i]*np.exp(
                (self.rate - 0.5*self.sigma**2)*self._dt_
                 + self.sigma*np.sqrt(self._dt_)*w)
        # The __dict__ attribute
        '''
        Contains all the attributes defined for the object itself.
        It maps the attribute name to its value.
        '''
        for i in ['callPrice', 'putPrice']:
            self.__dict__[i] = None
        [self.AsianCall, self.AsianPut] = self._Asian()
        [self.BinaryCall, self.BinaryPut] = self._Binary()
        [self.LookbackCall, self.LookbackPut] = self._Lookback()
    # Asian Options
    def _Asian(self):
        average = self._St_.mean(axis=0)
        Asian_call = mean(exp(-self.rate*self.T) *
                          maximum(0, average - self.strike))
        Asian_put = mean(exp(-self.rate*self.T) *
                         maximum(0, self.strike - average))
```

```
        return [Asian_call, Asian_put]
    # Binary Options
    def _Binary(self):
        Ind_Call = 1*((self._St_[-1]-self.strike) > 0)
        Binary_call = mean(exp(-self.rate*self.T) * Ind_Call)
        Ind_Put = 1*(self.strike-(self._St_[-1]) > 0)
        Binary_put = mean(exp(-self.rate*self.T) * Ind_Put)
        return [Binary_call, Binary_put]
    # Lookback Options
    def _Lookback(self):
        Opt_Max = self._St_.max(axis=0)
        Opt_Min = self._St_.min(axis=0)
        Lookback_call = mean(exp(-self.rate*self.T) *
                             maximum(0, Opt_Max - self.strike))
        Lookback_put = mean(exp(-self.rate*self.T) *
                            maximum(0, self.strike - Opt_Min))
        return [Lookback_call, Lookback_put]
```

2 Numerical Results

Table 1: Summary

Methods	Asian Call	Asian Put	Binary Call	Binary Put	Lookback Call	Lookback Put
Closed Form	5.787223	3.343797	0.532241	0.418988	18.943660	12.152153
Euler-Maruyama	5.787142	3.343850	0.532327	0.418902	18.943295	12.152118
Milstein	5.787121	3.343720	0.532241	0.418988	18.943288	12.151907

In this project, the underlying stock price (S=100), strike price (E or K=100), time to maturity (T-t=1), volatility (σ=20%), and risk-free rate (r=5%) are known. Vary values of parameters influencing the option pricing are employed to price the options, including time steps 2.1, simulation times 2.2, risk-free rate 2.3, volatility 2.4, maturity 2.5, simulation method 2.6.

2.1 Time Steps

Time Steps is employed to calculate the Δt, which influence the accuracy of option pricing. Table 3 shows that the option price are closer to actual price as time steps increasing. It can be seen as a continuous process when $\Delta t \to 0$. This issue will be discussed in Section 3.1.

Table 2: Option Prices with Vary Time Steps

Time Steps	Asian Call	Asian Put	Binary Call	Binary Put	Lookback Call	Lookback Put
1	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
2	3.3651	2.1505	0.5168	0.4344	6.7302	4.3010
4	4.5601	2.7768	0.5239	0.4273	10.9105	6.9655
12	5.3211	3.1684	0.5294	0.4218	14.7130	9.4181
48	5.6345	3.3044	0.5311	0.4201	17.0568	10.9257
72	5.6796	3.3170	0.5308	0.4204	17.4657	11.1784
180	5.7225	3.3231	0.5316	0.4197	18.1272	11.6083
252	5.7215	3.3321	0.5300	0.4212	18.2619	11.7492

2.2 Times of Simulation

Monte Carlo is utilised to produce the paths of underlying asset price as sample. The payoff of options is the expectation of all sample of final asset prices. When the paths simulated are enough abundant, the sample mean converges to the population mean based on the Law of large numbers. Moreover, the distribution of the sample mean slowly becomes a normal distribution when the sample size is large enough based on the Central limit theorem. This issue will be discussed in Section 3.1.

2.3 Risk-free Rate

Risk-free rate is a parameter associated with the currency. Figure 3 shows that the relation between the call option prices and risk-free rates is positive for three kinds of exotic options. For put options,

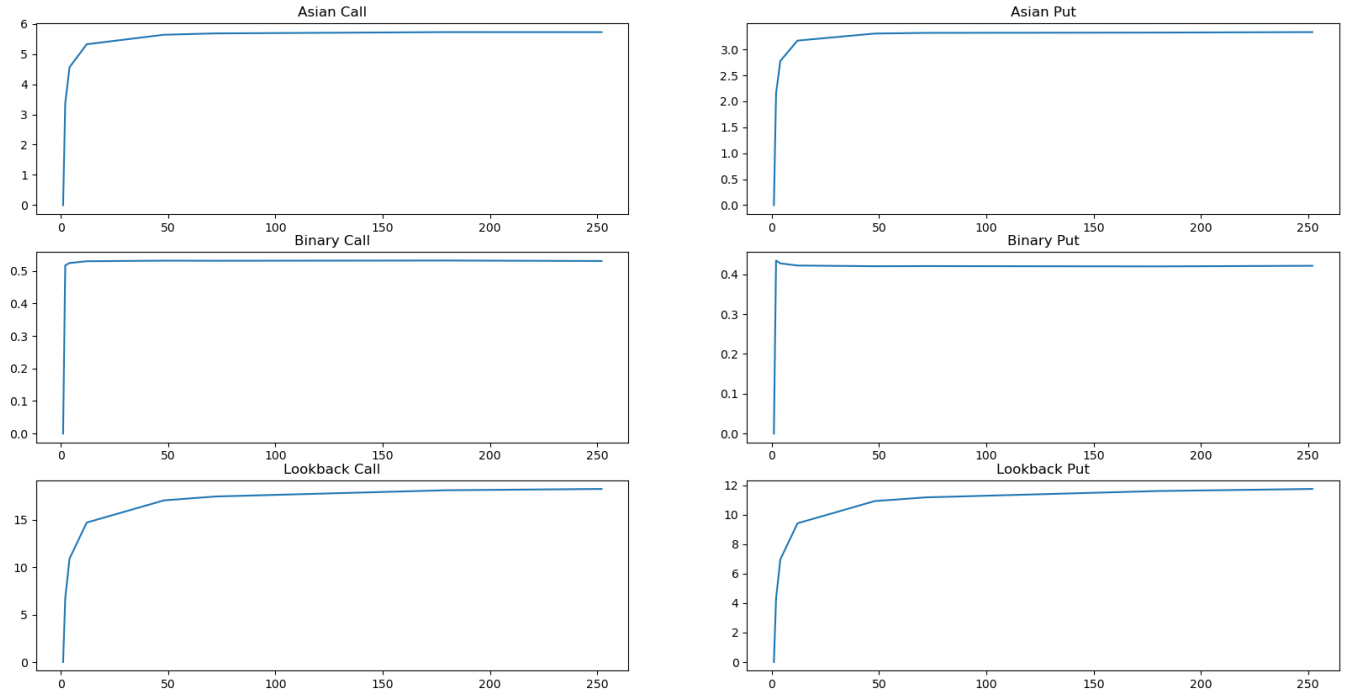


Figure 1: Visualisation of Option Prices with Vary Time Steps

Table 3: Option Prices with Vary Simulation Times

Time Steps	Asian Call	Asian Put	Binary Call	Binary Put	Lookback Call	Lookback Put
10	6.4201	5.0456	0.5707	0.3805	17.8644	13.7602
100	5.7623	3.5513	0.4281	0.5232	18.4832	12.4950
1000	5.4273	3.5452	0.5203	0.4309	18.2145	12.5459
10000	5.6884	3.3390	0.5244	0.4268	18.7221	12.1506
100000	5.7872	3.3438	0.5322	0.4190	18.9437	12.1522

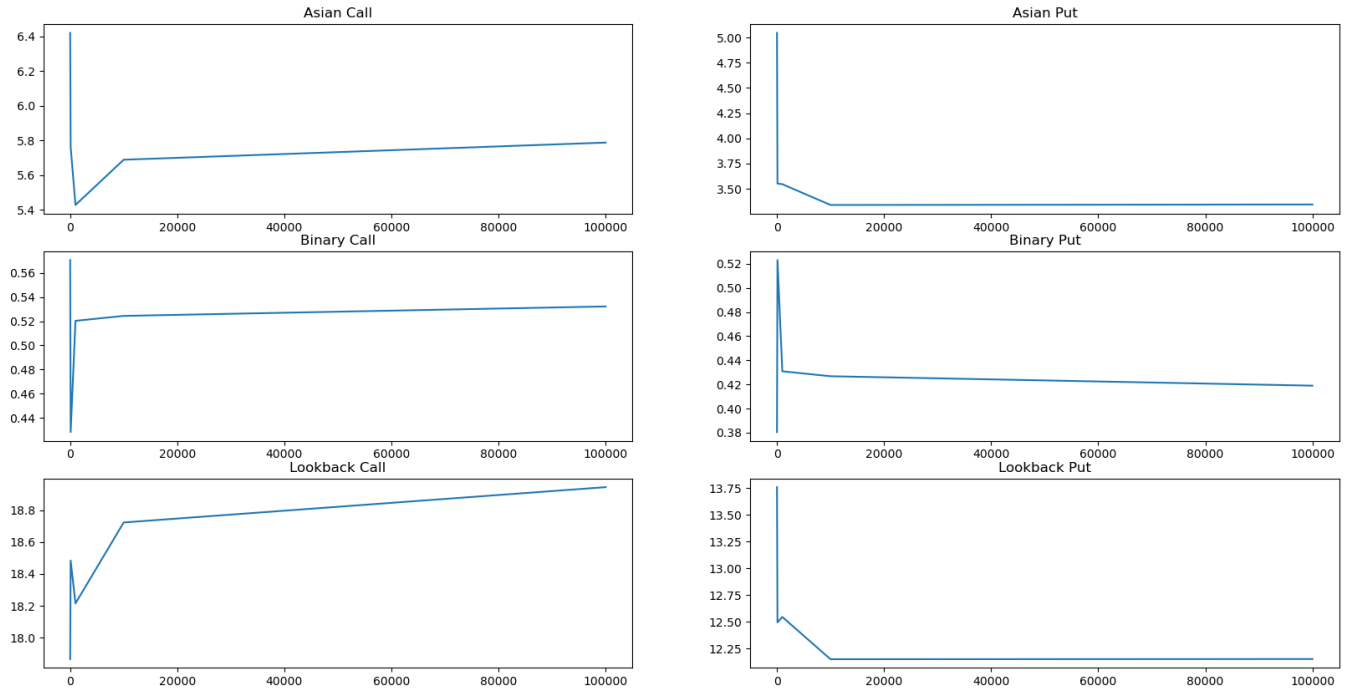


Figure 2: Visualisation of Option Prices with Vary Times of Simulation

its prices decline as the risk-free rates increase.

Table 4: Option Prices with Vary Risk-free Rates

Risk-free Rate	Asian Call	Asian Put	Binary Call	Binary Put	Lookback Call	Lookback Put
0.01	4.8509	4.3280	0.4755	0.5145	17.1794	14.2212
0.02	5.0775	4.0648	0.4902	0.4900	17.6076	13.6764
0.03	5.3092	3.8131	0.5051	0.4653	18.0444	13.1504
0.04	5.5458	3.5729	0.5193	0.4415	18.4898	12.6425
0.05	5.7872	3.3438	0.5322	0.4190	18.9437	12.1522
0.06	6.0335	3.1259	0.5453	0.3964	19.4057	11.6792
0.07	6.2842	2.9186	0.5578	0.3746	19.8756	11.2230
0.08	6.5392	2.7218	0.5697	0.3534	20.3529	10.7832
0.09	6.7984	2.5353	0.5819	0.3320	20.8370	10.3594
0.10	7.0615	2.3586	0.5929	0.3119	21.3279	9.9519

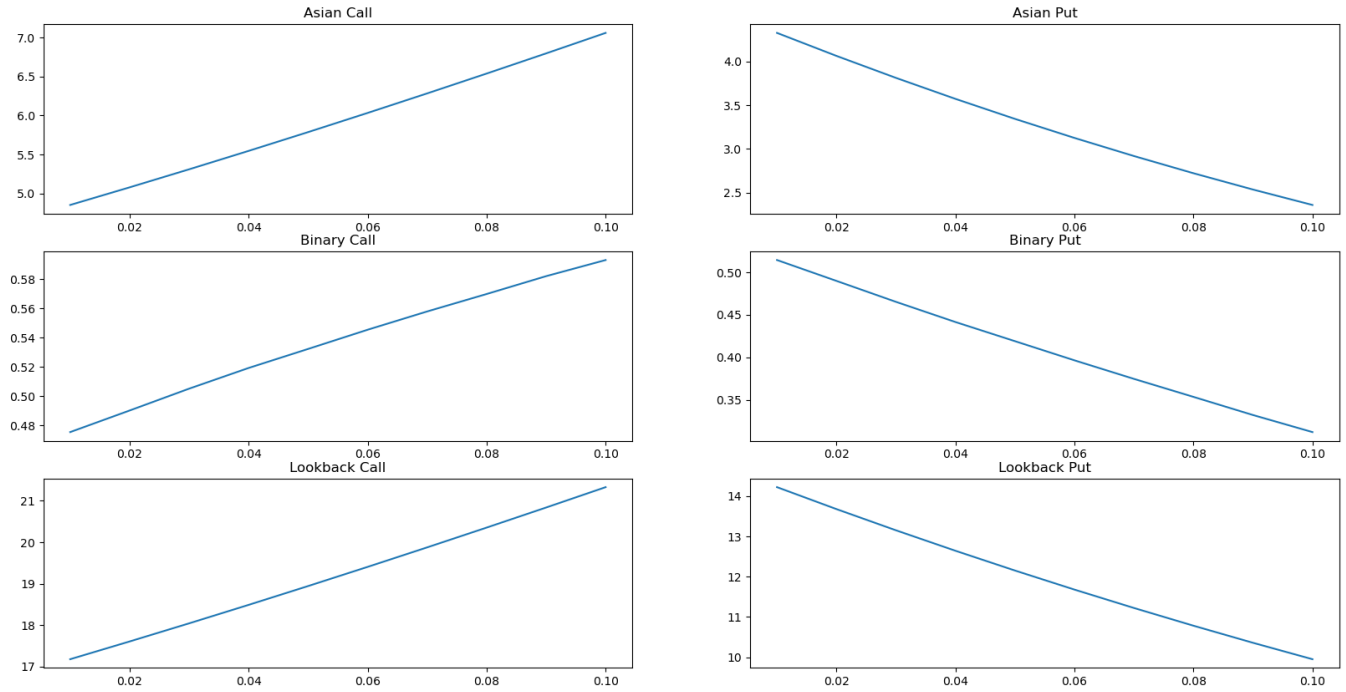


Figure 3: Visualisation of Option Prices with Vary Risk-free Rates

2.4 Volatility

Volatility is a parameter associated with the underlying prices. In general, the volatility has an positive impact on option prices. However, the Binary call option prices are roughly the same suggested by Figure 4. This issue will be discussed in Section 3.2.

2.5 Maturity

All kinds of exotic options are priced by vary time to expiry, from 1-day to 2-year. For Asian and Lookback options, Figure 5 suggested that longer time to expiry, higher option prices. Why the trends of prices of Binary options are roughly the same? This issue will be discussed in Section 3.3.

Table 5: Option Prices with Vary Volatility

Volatility	Asian Call	Asian Put	Binary Call	Binary Put	Lookback Call	Lookback Put
0.1	3.6499	1.2208	0.6406	0.3106	10.6814	5.3253
0.2	5.7872	3.3438	0.5322	0.4190	18.9437	12.1522
0.3	7.9891	5.5285	0.4821	0.4691	27.8279	18.6471
0.4	10.2017	7.7207	0.4479	0.5034	37.2754	24.7536
0.5	12.4126	9.9074	0.4198	0.5314	47.2871	30.4732
0.6	14.6154	12.0818	0.3951	0.5561	57.8767	35.8178
0.7	16.8058	14.2394	0.3718	0.5795	69.0635	40.8024
0.8	18.9800	16.3757	0.3498	0.6014	80.8663	45.4438
0.9	21.1362	18.4885	0.3298	0.6214	93.3019	49.7586
1.0	23.2708	20.5738	0.3105	0.6407	106.3934	53.7625

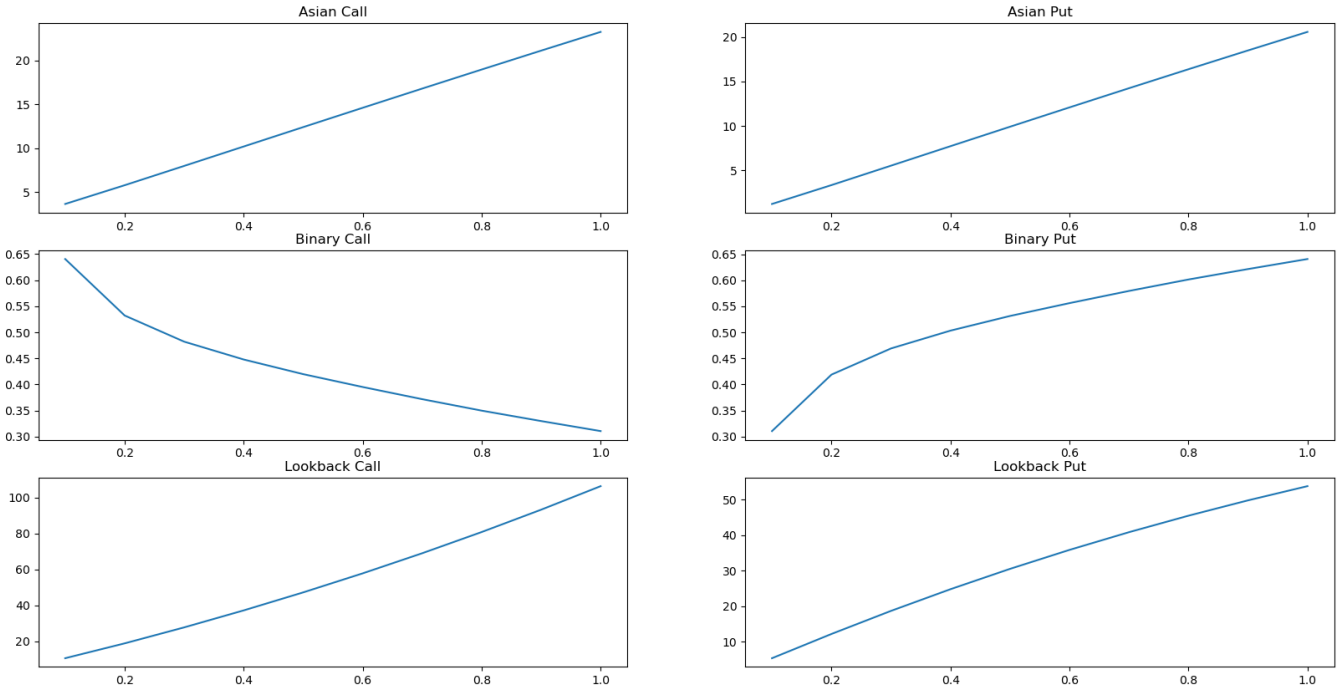


Figure 4: Visualisation of Option Prices with Vary Volatility

Table 6: Option Prices with Vary Maturities

Maturity	Asian Call	Asian Put	Binary Call	Binary Put	Lookback Call	Lookback Put
1/252	0.2965	0.2852	0.5040	0.4958	1.0062	0.9768
1/12	1.4389	1.2246	0.5157	0.4802	4.8321	4.2469
0.25	2.6181	1.9866	0.5238	0.4638	8.7152	6.9782
0.5	3.8701	2.6237	0.5294	0.4459	12.7847	9.3407
1.0	5.7872	3.3438	0.5322	0.4190	18.9437	12.1522
2.0	8.7393	4.0226	0.5281	0.3767	28.3347	15.0970

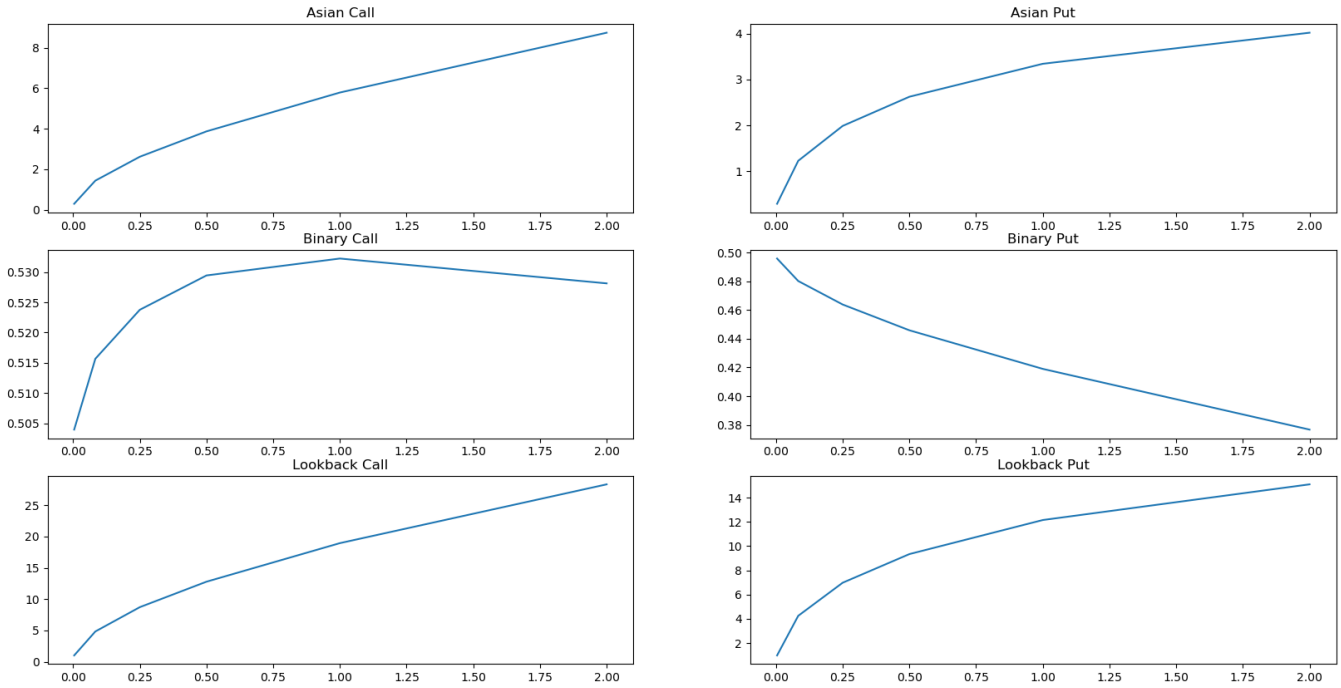


Figure 5: Visualisation of Option Prices with Vary Maturities

2.6 Simulation Methods

Table 7 shows the comparison among three kinds of simulation methods. The prices from closed form solution are considered as a benchmark. It concludes that the prices from Milstein schemes are more accurate than EulerMaruyama schemes for Binary options. However, the EulerMaruyama schemes' performance is better for Asian and Lookback options suggested by Table 7. This issue will be discussed in Section 3.4.

Table 7: Comparison of Euler-Maruyama & Milstein

Exotic Option	More Accurate
Asian Call	Euler-Maruyama
Asian Put	Euler-Maruyama
Binary Call	Milstein
Binary Put	Milstein
Lookback Call	Euler-Maruyama
Lookback Put	Euler-Maruyama

Table 8: Option Prices with Vary Simulation Methods

Methods	Asian Call	Asian Put	Binary Call	Binary Put	Lookback Call	Lookback Put
Closed Form	5.787223	3.343797	0.532241	0.418988	18.943660	12.152153
Euler-Maruyama	5.787142	3.343850	0.532327	0.418902	18.943295	12.152118
Milstein	5.787121	3.343720	0.532241	0.418988	18.943288	12.151907

Table 9: Difference(%) of Euler-Maruyama & Milstein

Methods	Asian Call	Asian Put	Binary Call	Binary Put	Lookback Call	Lookback Put
Euler-Maruyama	0.0014	0.0016	0.0161	0.0204	0.0019	0.0003
Milstein	0.0018	0.0023	0	0	0.002	0.002

Python code for Asian Call Option is as follows

```
ClosedForm = Monte_Carlo_ClosedForm(S0,K,r,sigma,T,Nsteps,Nsim)
ClosedForm_df = pd.DataFrame([ClosedForm.AsianCall, ClosedForm.AsianPut,
                    ClosedForm.BinaryCall, ClosedForm.BinaryPut,
                    ClosedForm.LookbackCall, ClosedForm.LookbackPut]).T

EulerMaruyama = Monte_Carlo_EulerMaruyama(S0,K,r,sigma,T,Nsteps,Nsim)
EulerMaruyama_df = pd.DataFrame([EulerMaruyama.AsianCall, EulerMaruyama.AsianPut,
                    EulerMaruyama.BinaryCall, EulerMaruyama.BinaryPut,
                    EulerMaruyama.LookbackCall, EulerMaruyama.LookbackPut]).T

Milstein = Monte_Carlo_Milstein(S0,K,r,sigma,T,Nsteps,Nsim)
Milstein_df = pd.DataFrame([Milstein.AsianCall, Milstein.AsianPut,
                    Milstein.BinaryCall, Milstein.BinaryPut,
                    Milstein.LookbackCall, Milstein.LookbackPut]).T

results_df1 = pd.concat([ClosedForm_df, EulerMaruyama_df])
results_df2 = pd.concat([results_df1, Milstein_df])
results_df2.index = ['Closed Form', 'Euler-Maruyama', 'Milstein']
results_df2.index.name = 'Methods'
results_df2.columns = ['Asian Call', 'Asian Put',
                       'Binary Call', 'Binary Put',
                       'Lookback Call', 'Lookback Put']

result_np = results_df2.values
EM_CF = abs(result_np[1] - result_np[0])
M_CF = abs(result_np[2] - result_np[0])
comparison = []
for i in range(len(result_np[0])):
    if EM_CF[i] < M_CF[i]:
        results = 'Euler-Maruyama'
    else:
        results = 'Milstein'
    comparison.append(results)
```

```
header = ['Asian Call', 'Asian Put',
          'Binary Call', 'Binary Put',
          'Lookback Call', 'Lookback Put']
table = [[comparison[0],comparison[1],comparison[2],
          comparison[3],comparison[4],comparison[5]]]
```

3 Interesting Observations and Problems Encountered

3.1 Time Steps and Simulation Times

The underlying asset price paths are produced by Monte Carlo Simulation with Euler-Maruyama or Milstein. The $\mathbb{E}[S(T)]$ is the expectation of final prices of underlying asset prices. Using M Monte Carlo samples we may from the sample average

$$a_M = \frac{1}{M}\sum_{i=1}^{M} S_i$$

The overall error has from

$$\begin{aligned} a_M - \mathbb{E}[S(T)] &= a_M - \mathbb{E}[S(T) - S_T + S_T] \\ &= a_M - \mathbb{E}[S_T] + \mathbb{E}[S_T - S(T)] \end{aligned}$$

Note that the total error includes two terms. The statistical error, $a_M - \mathbb{E}[S_T]$ is about how well we can calculate an expected value from a finite number of samples; it does not depend on how accurately the solution of the SDE from numerical methods, especially it does not depend on Δt) and it will decline if we simulate more sample paths. $\mathbb{E}[S_T - S(T)]$ increases because we have approximated the SDE with a difference equation; this is the difference that would remain if we had access to the exact expected value of the numerical solution and it will decline if we reduce the stepsize Higham (2015*a*).

3.2 How does the Volatility Affect the Binary Call Option Price?

The Binary call option prices decline as the volatility increases, which is different from Asian options and Lookback options. In this part, It can be explained by mathematical prove and empirical evidence. The risk-free rate is assumed as 0 in later discussion. The expectation of Binary call option price under Q-measure is defined as $E^{\mathbb{Q}}[1_{S_T>K}]$. After taking logarithm for both sides, we have

$$Q(S_T > K) = Q(\log S_T > \log K)$$

Under Q-measure, we have

$$S_T = S_0 exp(-\frac{1}{2}\sigma^2 T + \sigma W_T)$$

So, the distribution of $\log S_T$ is below

$$\log S_T \sim N(\log S_0 - \frac{1}{2}\sigma^2 T, \sigma^2 T)$$

Plug S_T to above the equation, we get

$$Q\left(\log S_0 - \frac{1}{2}\sigma^2 T + \sigma\sqrt{T}N > \log K\right)$$

which equals to

$$Q\left(N > \frac{\log(\frac{K}{S_0}) + \frac{1}{2}\sigma^2 T}{\sigma\sqrt{T}}\right)$$

A new function is defined as below:

$$f(\mathrm{y}) = Q(N > \mathrm{y})$$

It can be written as

$$\mathrm{y} = \mathrm{y}(\sigma) = \frac{\log(\frac{K}{S_0}) + \frac{1}{2}\sigma^2 T}{\sigma\sqrt{T}}$$

If $K > S_0$, called Out-the-Money, we get

$$\mathrm{y}(\sigma) \to \begin{cases} +\infty, & \sigma \to 0 \\ +\infty, & \sigma \to +\infty \end{cases}$$

The $\mathrm{y}(\sigma)$ has a minimum value when

$$\sigma^* = \sqrt{\log(\frac{K}{S_0})}$$

We conclude that

$$f(\mathrm{y}) = \begin{cases} 0, & \mathrm{y} = \mathrm{y}(0) \\ f(\mathrm{y})_{Max}, & \mathrm{y} = \mathrm{y}(\sigma^*) \\ 0, & \mathrm{y} = \mathrm{y}(+\infty) \end{cases}$$

which is showed by Figure 6. The option prices go up to the peak and then decline as increasing volatility.

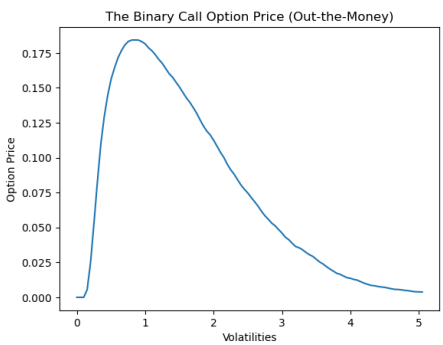


Figure 6: Binary Call Option Price (OTM)

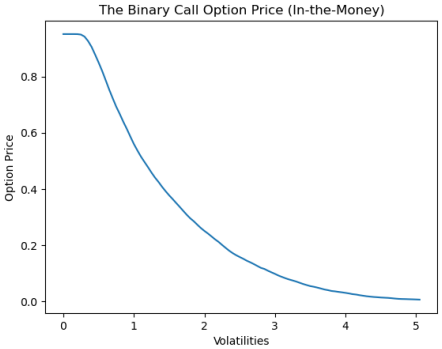


Figure 7: Binary Call Option Price (ITM)

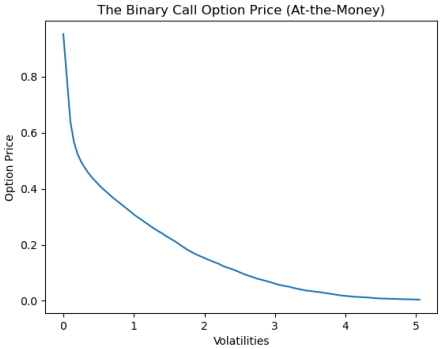


Figure 8: Binary Call Option Price (ATM)

If $K < S_0$, called In-the-Money, we get

$$\mathrm{y}(\sigma) \to \begin{cases} -\infty, & \sigma \to 0 \\ +\infty, & \sigma \to +\infty \end{cases}$$

which is increasing, and there is no maximum and minimum. We conclude that

$$f(\mathrm{y}) = \begin{cases} 1, & \mathrm{y} = \mathrm{y}(0) \\ 0, & \mathrm{y} = \mathrm{y}(+\infty) \end{cases}$$

which is strictly decreasing showed by Figure 7. If $K = S_0$, called At-the-Money, we have

$$f(\mathrm{y}) = Q(N > \frac{1}{2}\sigma\sqrt{T})$$

3.3 The Sensitivity of Maturity in Binary Option Pricing

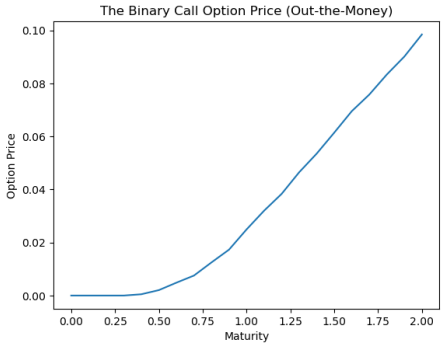


Figure 9: Binary Call Option Price (OTM)

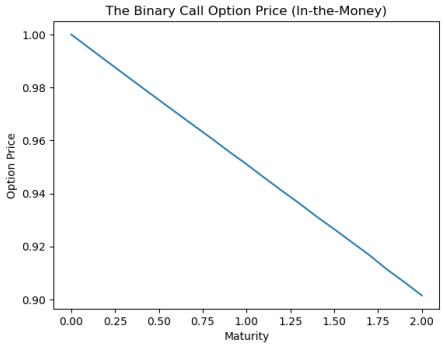


Figure 10: Binary Call Option Price (ITM)

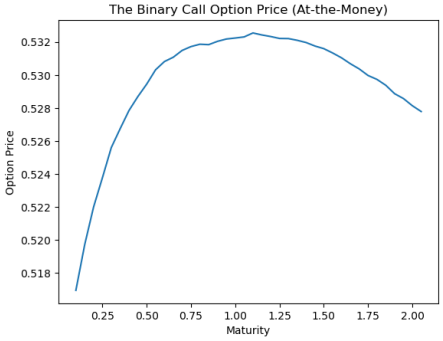


Figure 11: Binary Call Option Price (ATM)

3.4 Convergence

The concept of convergence formalizes what it means for one stochastic process to get closer to another as the discrete time steps Δt are reduced. There are two types: "weak convergence" and "strong convergence". For weak convergence, the error term is defined by

$$e(\Delta t) = \sup_{t_n} |E(X(t_n)) - E(Y(t_n))| \tag{21}$$

and for strong convergence, the error term is defined by

$$e(\Delta t) = \sup_{t_n} E[|X(t_n) - Y(t_n)|] \tag{22}$$

$X(t)$ exhibits strong and weak convergence to Y(t) if relevant error trends to zero with Δt

$$\lim_{\Delta t \to 0} e(\Delta t) = 0$$

The weak error term simply computes the error between the expected values of the two stochastic processes at a given point. So weak convergence captures the average of the simulated approximations. The strong error term is the mean of errors, which is the difference between the approximation and the exact solution for every individual sample path before the average is taken. Strong convergence is

therefore more demanding.

So the appropriate sense of convergence here depends on whether we are interested in the "closeness" of the whole trajectory of the solution to the SDE (in which case we care about strong convergence) or in the expected value of some function of the process (which relates to weak convergence).

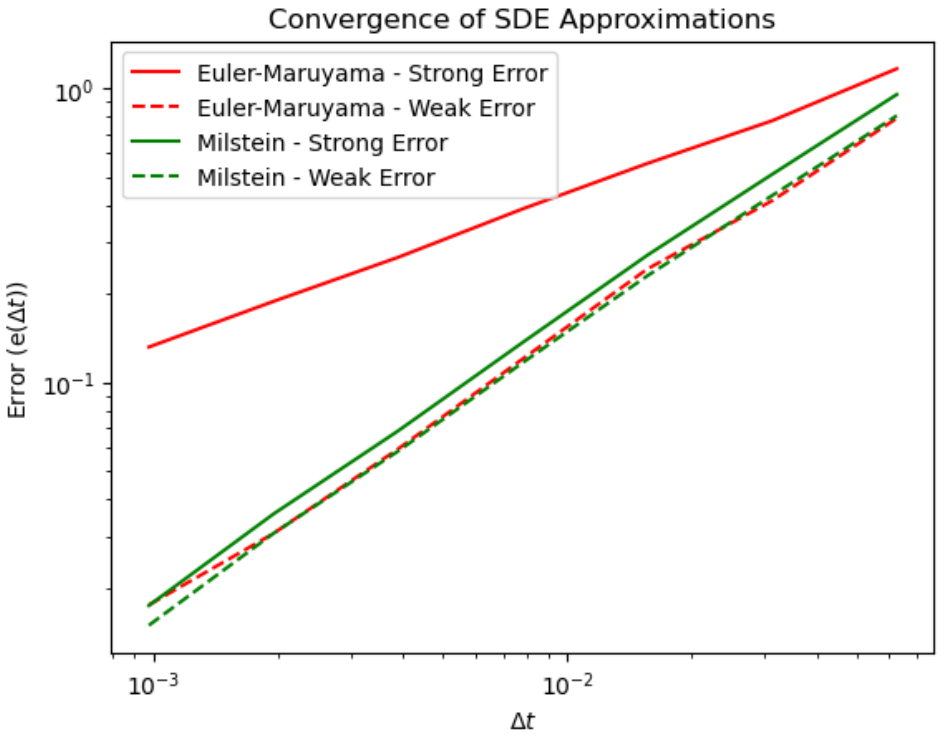


Figure 12: Comparison of Convergence

The general relationship in Figure 12 is that the strong and weak error terms decrease with Δt, which means that both approximations are strongly and weakly convergent. Besides, the weak errors (dashed lines) are always lower than the strong errors. We also see that the strong Milstein errors are lower than the E-M errors. However, the weak errors are roughly the same. Because although the errors in every sample path may be larger for the E-M approximation, on average, the negative and the positive errors presumably cancel out so that the average paths of the E-M and Milstein approximations are roughly similar, which can explain that the E-M is more accurate than Miltein method in some situations.

In Section 2.6, we found Milstein is not better than Euler-Maruyama in Asian and Lookback options, which can be explained by Figure 12. For a Binary option, for example, we are only interested in the the price of a stock at a expiry date, so we would mostly be interested in weak convergence. But for an Asian or Lookback option, we are not just interested in how the approximations might behave on final price of underlying asset, but we are also specifically interested in individual paths, because too large a divergence on any path can affect the payoff. In this case, of course, we'd be interested in strong convergence.

Python codes for this part are as follows

```
# SDE model parameters
mu, sigma, X0 = 2, 1, 1
# Initiate dt grid and lists to store errors
str_err_em, str_err_mil, weak_err_em, weak_err_mil = [], [], [], []
dt_grid = [2 ** (R-10) for R in range(7)]
mc = 10000
# Loop over values of dt
for Dt in dt_grid:
    # Setup discretized grid
    t = np.arange(Dt, 1 + Dt, Dt)
    n = len(t)
    # Initiate vectors to store errors and time series
    err_em, err_mil = np.zeros(n), np.zeros(n)
    Y_sum, Xem_sum, Xmil_sum = np.zeros(n), np.zeros(n), np.zeros(n)
    # Generate many sample paths
    for i in range(mc):
        # Create Brownian Motion
        np.random.seed(i)
        dB = np.sqrt(Dt) * np.random.randn(n)
        B  = np.cumsum(dB)
        # Exact solution
        Y = X0 * np.exp((mu - 0.5*sigma**2)*t + sigma * B)
        # Simulate stochastic processes
        Xemt, Xmilt, Xem, Xmil = X0, X0, [], []
        for j in range(n):
            # Euler-Maruyama
            Xemt += mu*Xemt* Dt + sigma * Xemt * dB[j]
            Xem.append(Xemt)
            # Milstein
            Xmilt += mu*Xmilt*Dt + sigma*Xmilt*dB[j]
                     + 0.5*sigma**2*Xmilt*(dB[j]**2 - Dt)
            Xmil.append(Xmilt)
        # Compute strong errors and add to those across from other sample paths
        err_em  += abs(Y - Xem)
        err_mil += abs(Y - Xmil)
        # Add Y and X values to previous sample paths
        Y_sum += Y
        Xem_sum += Xem
        Xmil_sum += Xmil
    # Compute mean of absolute errors and find maximum (strong error)
```

```
    str_err_em.append(max(err_em / mc))
    str_err_mil.append(max(err_mil / mc))
    # Compute error of means and find maximum (weak error)
    weak_err_em.append(max(abs(Y_sum - Xem_sum)/mc))
    weak_err_mil.append(max(abs(Y_sum - Xmil_sum)/mc))
```

3.5 Runge–Kutta Method

To get the more accurate solution of stochastic differential equation, Runge–Kutta methods were developed for ODEs, which trade these extra partial derivatives in the Taylor expansion for extra function evaluations from the readily-available differential equation.

In the SDE context, the same trade can be made with the Milstein method, resulting in a strong order 1 method that requires evaluation of $B(Y)$ at two places on each step. A heuristic derivation can be carried out by making the replacement

$$B_Y(Y_i) \approx \frac{B(Y_i + B(Y_i)\sqrt{\Delta t_i}) - B(Y_i)}{B(Y_i)\sqrt{\Delta t_i}}$$

in the Equation 6, which leads to the Runge–Kutta method with strong convergence

$$dY_{i+1} = Y_i + A(Y_i)\Delta t + B(Y_i)\Delta W_t + \frac{1}{2}[B(Y_i + B(Y_i)\sqrt{\Delta t_i}) - B(Y_i)](\Delta W_i^2 - \Delta t_i)/\sqrt{\Delta t_i} \tag{23}$$

and for he Runge–Kutta method with weak convergence

$$\begin{aligned} dY_{i+1} = Y_i &+ \frac{1}{2}[A(\mathrm{u}) + A(Y_i)]\Delta t_i \\ &+ \frac{1}{4}[B(\mathrm{u}_+) + B(\mathrm{u}_-) - 2B(Y_i)]\Delta W_i \\ &+ \frac{1}{4}[B(\mathrm{u}_+) + B(\mathrm{u}_-)](\Delta W_i^2 - \Delta t)/\sqrt{\Delta t_i} \end{aligned}$$

where

$$\begin{aligned} \mathrm{u} &= Y_i + A\Delta t_i + B\Delta W_i \\ \mathrm{u}_+ &= Y_i + A\Delta t_i + B\sqrt{\Delta t_i} \\ \mathrm{u}_- &= Y_i + A\Delta t_i - B\sqrt{\Delta t_i} \end{aligned}$$

3.6 Monte Carlo Simulation with Antithetic Variables

The antithetic variates method is a variance reduction technique used in Monte Carlo methods. Considering that the error in the simulated signal (using Monte Carlo methods) has a one-over 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. Figure 13, Figure 14, Figure 15

show the standard error of Asian call option prices by three simulation methods.

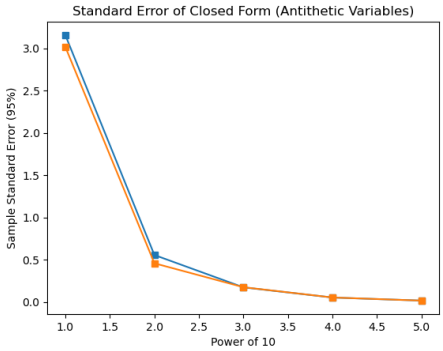


Figure 13: Asian Call Option Error (Closed Form)

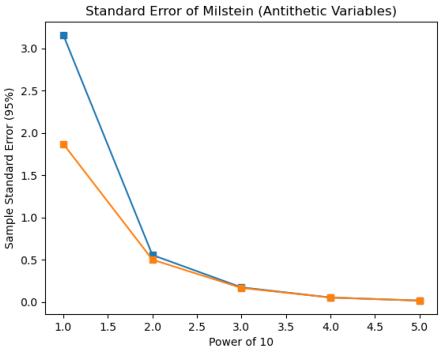


Figure 14: Asian Call Option Error (Euler-Maruyama)

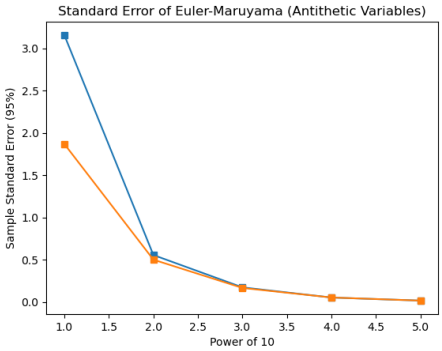


Figure 15: Asian Call Option Error (Milstein)

3.7 Multilevel Monte Carlo Simulation

For Standard Monte Carlo Simulation, a main weakness is mentioned in previous section, which is large cost and low efficiency for abundant simulation paths. Recently, Multilevel Monte Carlo Simulation is a hot topic used in option pricing. Higham (2015*b*) researched that most of samples are produced by low frequency quickly and produced small samples to cover high frequency information. It is more efficient than Standard Monte Carlo Simulation. In Higham (2015*b*), the computational experiments indicates that Multilevel Monte Carlo Simulation offers a dramatic improvement in computational complexity.

4 Conclusion

This report researches the Asian, Binary, and Lookback options by Monte Carlo Simulation with Closed Form Solution, Euler-Maruyama, and Milstein schemes. Milstein is more accurate than Euler-Maruyama for exotic options with weak convergence, such as Binary and Barrier options. The Euler-Maruyama perform better in pricing Asian and Lookback options, which are strong in convergence.

Although standard Monte Carlo Simulation is popular in finance applications, the weakness still needs to be considered, including the computation cost and efficiency. The Multilevel Monte Carlo Simulation can improve the efficiency in computation by simulating different sizes of samples to cover various paths with extra information. However, the complexity of the solution of SDE increases.

Observing all the exotic option prices, we found the sensitivity of parameters in option pricing. Moreover, there are some interesting observations. For instance, the Binary option differs from Asian and Lookback options in sensitivity of volatility and maturity.

Appendix

I Python Codes

Table of Contents

In [1]:
```
# Importing libraries
import pandas as pd
from numpy import *
import scipy.stats as si
import matplotlib
import matplotlib.pyplot as plt

from tabulate import tabulate
```

executed in 1.64s, finished 09:16:40 2022-09-29

In [2]:
```
def normfit(data, confidence=0.95):
    a = 1.0 * np.array(data)
    n = len(a)
    m, se = np.mean(a), si.sem(a)
    h = se * si.t.ppf((1 + confidence) / 2., n - 1)
    var = np.var(data, ddof=1) # Divide by n-1
    sigma = np.sqrt(var)
    return m, sigma, np.hstack((m-h,m+h))
```

executed in 4ms, finished 09:16:41 2022-09-29

1 Define Functions

In [3]:
```
S0, K, T, sigma, r, Nsteps, Nsim = 100, 100, 1, 0.2, 0.05, 2520, 100000
```

executed in 3ms, finished 09:16:42 2022-09-29

Closed Form Solution:

$$S_{t+\delta t} = S_t \exp\left\{\left(r - \frac{1}{2}\sigma^2\right)\delta t + \sigma\phi\sqrt{\delta t}\right\}$$

Euler-Maruyama Scheme:

$$S_{t+\delta t} \sim S_t(1 + r\delta t + \sigma\phi\sqrt{\delta t})$$

Milstein Scheme:

$$S_{t+\delta t} \sim S_t\left(1 + r\delta t + \sigma\phi\sqrt{\delta t} + \frac{1}{2}\left(\phi^2 - 1\right)\delta t\right)$$

1.1 Closed Form Solutions

In [299]:

```python
class Monte_Carlo_ClosedForm:

    def __init__(self, spot, strike, rate, sigma, T, Nsteps, Nsim):

        random.seed(10000)
        self.spot = spot                    # Initial stock price
        self.strike = strike                # Strike price
        self.rate = rate                    # Risk-free rate
        self.T = T                          # Maturity
        self.sigma = sigma                  # Volatility
        self.Nsteps = Nsteps                # Time steps
        self.Nsim = Nsim                    # Times of simulations
        self._dt_ = self.T / self.Nsteps    # Delta t
        self._St_ = zeros((self.Nsteps, self.Nsim))
        self._St_[0] = self.spot

        for i in range(0, self.Nsteps-1):
            w = random.standard_normal(self.Nsim)
            self._St_[i+1] = self._St_[i]*np.exp(
                (self.rate - 0.5*self.sigma**2)*self._dt_
                 + self.sigma*np.sqrt(self._dt_)*w)

        # The __dict__ attribute
        '''
        Contains all the attributes defined for the object itself.
        It maps the attribute name to its value.
        '''
        for i in ['callPrice', 'putPrice']:
            self.__dict__[i] = None

        [self.AsianCall, self.AsianPut] = self._Asian()
        [self.BinaryCall, self.BinaryPut] = self._Binary()
        [self.LookbackCall, self.LookbackPut] = self._Lookback()

    # Asian Options
    def _Asian(self):
        average = self._St_.mean(axis=0)
        Asian_call = mean(exp(-self.rate*self.T) *
                          maximum(0, average - self.strike))
        Asian_put = mean(exp(-self.rate*self.T) *
                         maximum(0, self.strike - average))
        return [Asian_call, Asian_put]

    # Binary Options
    def _Binary(self):
        Ind_Call = 1*((self._St_[-1]-self.strike) > 0)
        Binary_call = mean(exp(-self.rate*self.T) * Ind_Call)
        Ind_Put = 1*(self.strike-(self._St_[-1]) > 0)
        Binary_put = mean(exp(-self.rate*self.T) * Ind_Put)
        return [Binary_call, Binary_put]

    # Lookback Options
    def _Lookback(self):
        Opt_Max = self._St_.max(axis=0)
        Opt_Min = self._St_.min(axis=0)
        Lookback_call = mean(exp(-self.rate*self.T) *
                             maximum(0, Opt_Max - self.strike))
        Lookback_put = mean(exp(-self.rate*self.T) *
                            maximum(0, self.strike - Opt_Min))
        return [Lookback_call, Lookback_put]
```

executed in 12ms, finished 15:52:29 2022-09-25

1.2 Euler-Maruyama Schemes

In [284]:

```
class Monte_Carlo_EulerMaruyama:

    def __init__(self, spot, strike, rate, sigma, T, Nsteps, Nsim):

        random.seed(10000)
        self.spot = spot                    # Initial stock price
        self.strike = strike                # Strike price
        self.rate = rate                    # Risk-free rate
        self.T = T                          # Maturity
        self.sigma = sigma                  # Volatility
        self.Nsteps = Nsteps                # Time steps
        self.Nsim = Nsim                    # Times of simulations
        self._dt_ = self.T / self.Nsteps    # Delta t
        self._St_ = zeros((self.Nsteps, self.Nsim))
        self._St_[0] = self.spot

        for i in range(0, self.Nsteps-1):
            w = random.standard_normal(self.Nsim)
            self._St_[i+1] = self._St_[i] * (1 + self.rate*self._dt_
                                             + self.sigma*self._dt_**0.5*w)

        # The __dict__ attribute
        '''
        Contains all the attributes defined for the object itself.
        It maps the attribute name to its value.
        '''
        for i in ['callPrice', 'putPrice']:
            self.__dict__[i] = None

        [self.AsianCall, self.AsianPut] = self._Asian()
        [self.BinaryCall, self.BinaryPut] = self._Binary()
        [self.LookbackCall, self.LookbackPut] = self._Lookback()

    def _Asian(self):
        average = self._St_.mean(axis=0)
        Asian_call = mean(exp(-self.rate*self.T) *
                          maximum(0, average - self.strike))
        Asian_put = mean(exp(-self.rate*self.T) *
                         maximum(0, self.strike - average))
        return [Asian_call, Asian_put]

    def _Binary(self):
        Ind_Call = 1*((self._St_[-1]-self.strike)>0)
        Binary_call = mean(exp(-self.rate*self.T) * Ind_Call)
        Ind_Put = 1*(self.strike-(self._St_[-1])>0)
        Binary_put = mean(exp(-self.rate*self.T) * Ind_Put)
        return [Binary_call, Binary_put]

    def _Lookback(self):
        Opt_Max = self._St_.max(axis=0)
        Opt_Min = self._St_.min(axis=0)
        Lookback_call = mean(exp(-self.rate*self.T) *
                             maximum(0, Opt_Max - self.strike))
        Lookback_put = mean(exp(-self.rate*self.T) *
                            maximum(0, self.strike - Opt_Min))
        return [Lookback_call, Lookback_put]
```

executed in 11ms, finished 15:34:36 2022-09-25

1.3 Milstein Schemes

In [285]:

```
class Monte_Carlo_Milstein:

    def __init__(self, spot, strike, rate, sigma, T, Nsteps, Nsim):

        random.seed(10000)
        self.spot = spot                    # Initial stock price
        self.strike = strike                # Strike price
        self.rate = rate                    # Risk-free rate
        self.T = T                          # Maturity
        self.sigma = sigma                  # Volatility
        self.Nsteps = Nsteps                # Time steps
        self.Nsim = Nsim                    # Times of simulations
        self._dt_ = self.T / self.Nsteps    # Delta t
        self._St_ = zeros((self.Nsteps, self.Nsim))
        self._St_[0] = self.spot

        for i in range(0, self.Nsteps-1):
            w = random.standard_normal(self.Nsim)
            self._St_[i+1] = self._St_[i] * (1 + self.rate*self._dt_
                                             + self.sigma*self._dt_**0.5*w
                                             + 0.5*self.sigma**2*(w**2-1)*self._dt_)

        # The __dict__ attribute
        '''
        Contains all the attributes defined for the object itself.
        It maps the attribute name to its value.
        '''
        for i in ['callPrice', 'putPrice']:
            self.__dict__[i] = None

        [self.AsianCall, self.AsianPut] = self._Asian()
        [self.BinaryCall, self.BinaryPut] = self._Binary()
        [self.LookbackCall, self.LookbackPut] = self._Lookback()

    def _Asian(self):
        average = self._St_.mean(axis = 0)
        Asian_call = mean(exp(-self.rate*self.T) * maximum(0, average - self.strike))
        Asian_put = mean(exp(-self.rate*self.T) * maximum(0, self.strike - average))
        return [Asian_call, Asian_put]

    def _Binary(self):
        Ind_Call = 1*((self._St_[-1]-self.strike)>0)
        Binary_call = mean(exp(-self.rate*self.T) * Ind_Call)
        Ind_Put = 1*(self.strike-(self._St_[-1])>0)
        Binary_put = mean(exp(-self.rate*self.T) * Ind_Put)
        return [Binary_call, Binary_put]

    def _Lookback(self):
        Opt_Max = self._St_.max(axis=0)
        Opt_Min = self._St_.min(axis=0)
        Lookback_call = mean(exp(-self.rate*self.T) *
                             maximum(0, Opt_Max - self.strike))
        Lookback_put = mean(exp(-self.rate*self.T) *
                            maximum(0, self.strike - Opt_Min))
        return [Lookback_call, Lookback_put]
```

executed in 11ms, finished 15:34:39 2022-09-25

2 Results

2.1 Time Steps

In [436]:

```
Nsteps_list = [1,2,4,12,48,72,180,252]
Nsteps_results = []
for i in Nsteps_list:
    ClosedForm = Monte_Carlo_ClosedForm(S0,K,r,sigma,T,i,Nsim)
    results = [ClosedForm.AsianCall, ClosedForm.AsianPut,
               ClosedForm.BinaryCall, ClosedForm.BinaryPut,
               ClosedForm.LookbackCall, ClosedForm.LookbackPut]
    Nsteps_results.append(results)
df_Nsteps_results = pd.DataFrame(Nsteps_results)
df_Nsteps_results.index = Nsteps_list
df_Nsteps_results.index.name = 'Time Steps'
df_Nsteps_results.columns = ['Asian Call', 'Asian Put',
                             'Binary Call', 'Binary Put',
                             'Lookback Call', 'Lookback Put']
```

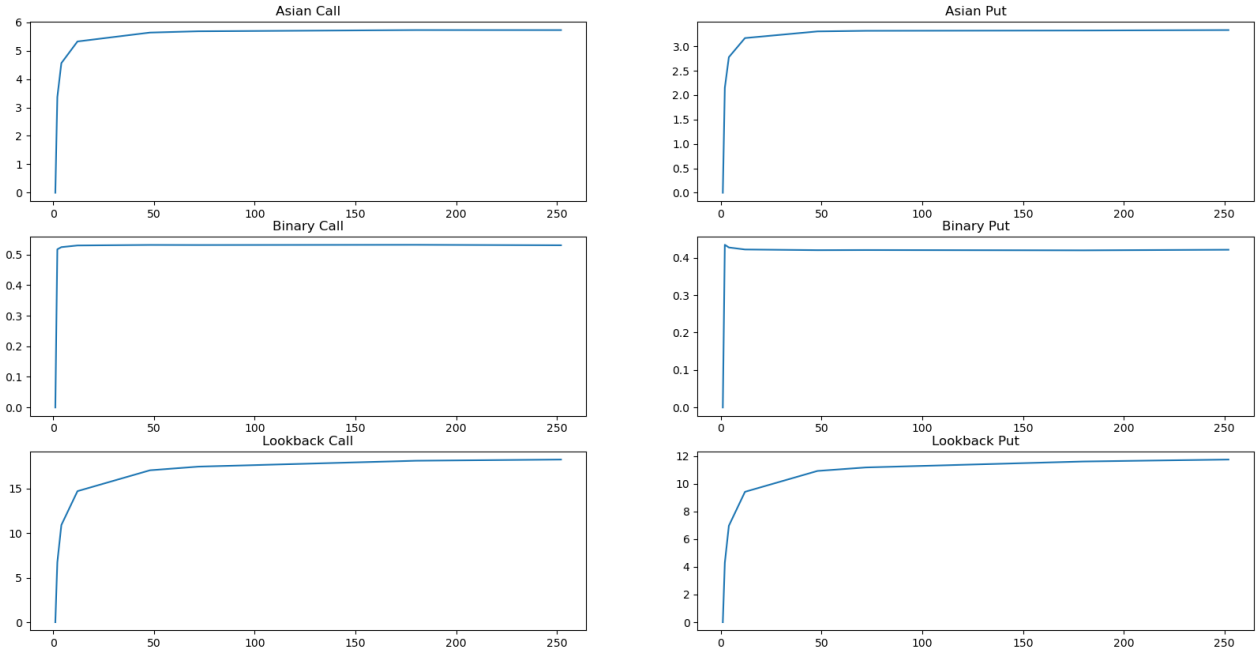
executed in 2.14s, finished 21:07:35 2022-09-25

In [574]:
```
df_Nsteps_results
```
executed in 31ms, finished 07:04:28 2022-09-28

Out[574]:

Time Steps	Asian Call	Asian Put	Binary Call	Binary Put	Lookback Call	Lookback Put
1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
2	3.365105	2.150496	0.516831	0.434398	6.730211	4.300992
4	4.560055	2.776811	0.523937	0.427292	10.910464	6.965515
12	5.321135	3.168353	0.529388	0.421842	14.712957	9.418132
48	5.634464	3.304361	0.531138	0.420091	17.056774	10.925674
72	5.679554	3.317031	0.530834	0.420396	17.465683	11.178371
180	5.722480	3.323130	0.531576	0.419654	18.127193	11.608316
252	5.721539	3.332069	0.530044	0.421185	18.261860	11.749170

In [439]:
```
# Plot graph iteratively
fig, ax = plt.subplots(3,2, figsize=(20,10))

ax[0,0].plot(df_Nsteps_results.iloc[:,[0]])
ax[0,1].plot(df_Nsteps_results.iloc[:,[1]])
ax[1,0].plot(df_Nsteps_results.iloc[:,[2]])
ax[1,1].plot(df_Nsteps_results.iloc[:,[3]])
ax[2,0].plot(df_Nsteps_results.iloc[:,[4]])
ax[2,1].plot(df_Nsteps_results.iloc[:,[5]])

# Set axis title
ax[0,0].set_title('Asian Call'), ax[0,1].set_title('Asian Put'),
ax[1,0].set_title('Binary Call'), ax[1,1].set_title('Binary Put'),
ax[2,0].set_title('Lookback Call'), ax[2,1].set_title('Lookback Put')

plt.show()
```
executed in 543ms, finished 21:07:59 2022-09-25



2.2 Simulation Times

In [456]:
```
Nsim_list = [10,100,1000,10000,100000]
Nsim_results = []
for i in Nsim_list:
    ClosedForm = Monte_Carlo_ClosedForm(S0,K,r,sigma,T,Nsteps,i)
    results = [ClosedForm.AsianCall, ClosedForm.AsianPut,
               ClosedForm.BinaryCall, ClosedForm.BinaryPut,
               ClosedForm.LookbackCall, ClosedForm.LookbackPut]
    Nsim_results.append(results)
df_Nsim_results = pd.DataFrame(Nsim_results)
df_Nsim_results.index = Nsim_list
df_Nsim_results.index.name = 'Time Steps'
df_Nsim_results.columns = ['Asian Call', 'Asian Put',
                           'Binary Call', 'Binary Put',
                           'Lookback Call', 'Lookback Put']
```

executed in 11.8s, finished 22:40:58 2022-09-25

In [457]:
```
round(df_Nsim_results,4)
```

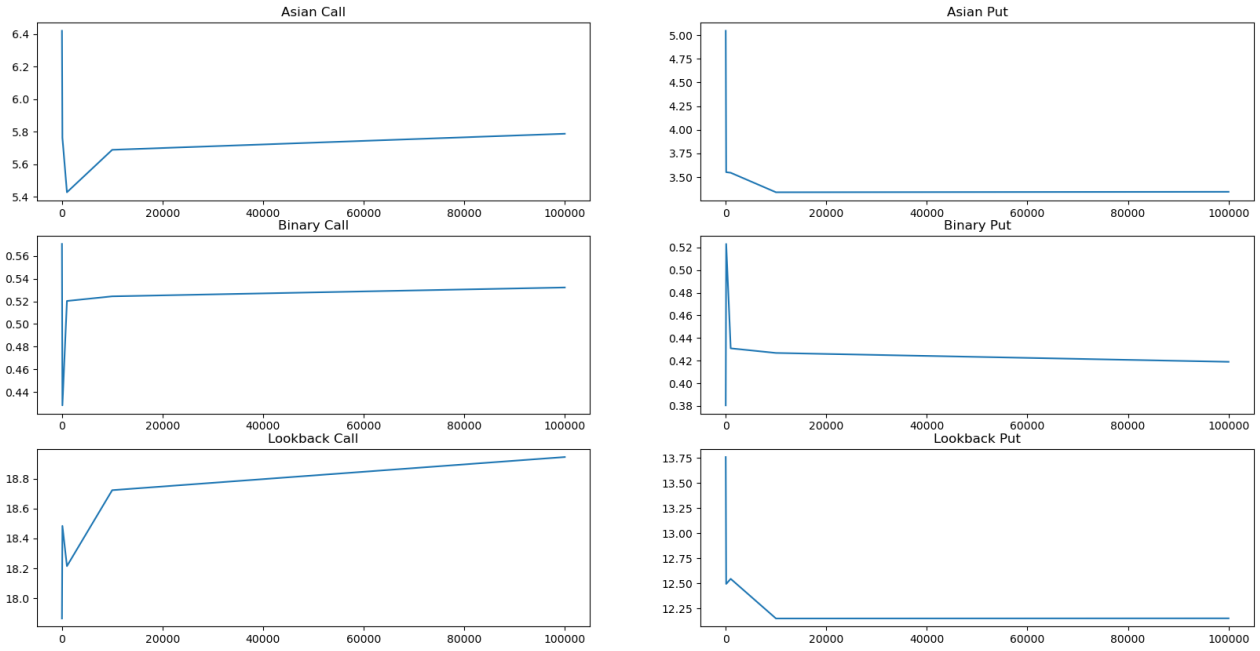
executed in 18ms, finished 22:41:03 2022-09-25

Out[457]:

Time Steps	Asian Call	Asian Put	Binary Call	Binary Put	Lookback Call	Lookback Put
10	6.4201	5.0456	0.5707	0.3805	17.8644	13.7602
100	5.7623	3.5513	0.4281	0.5232	18.4832	12.4950
1000	5.4273	3.5452	0.5203	0.4309	18.2145	12.5459
10000	5.6884	3.3390	0.5244	0.4268	18.7221	12.1506
100000	5.7872	3.3438	0.5322	0.4190	18.9437	12.1522

In [458]:
```
# Plot graph iteratively
fig, ax = plt.subplots(3,2, figsize=(20,10))

ax[0,0].plot(df_Nsim_results.iloc[:,[0]])
ax[0,1].plot(df_Nsim_results.iloc[:,[1]])
ax[1,0].plot(df_Nsim_results.iloc[:,[2]])
ax[1,1].plot(df_Nsim_results.iloc[:,[3]])
ax[2,0].plot(df_Nsim_results.iloc[:,[4]])
ax[2,1].plot(df_Nsim_results.iloc[:,[5]])

# Set axis title
ax[0,0].set_title('Asian Call'), ax[0,1].set_title('Asian Put'),
ax[1,0].set_title('Binary Call'), ax[1,1].set_title('Binary Put'),
ax[2,0].set_title('Lookback Call'), ax[2,1].set_title('Lookback Put')

plt.show()
```

executed in 626ms, finished 22:41:35 2022-09-25



2.3 Risk-free Rate

In [444]:

```
r_list = np.arange(0.01,0.11,0.01)
r_results = []
for i in r_list:
    ClosedForm = Monte_Carlo_ClosedForm(S0,K,i,sigma,T,Nsteps,Nsim)
    results = [ClosedForm.AsianCall, ClosedForm.AsianPut,
               ClosedForm.BinaryCall, ClosedForm.BinaryPut,
               ClosedForm.LookbackCall, ClosedForm.LookbackPut]
    r_results.append(results)
df_r_results = pd.DataFrame(r_results)
df_r_results.index = r_list
df_r_results.index.name = 'Risk-free Rate'
df_r_results.columns = ['Asian Call', 'Asian Put',
                        'Binary Call', 'Binary Put',
                        'Lookback Call', 'Lookback Put']
```

executed in 1m 43.1s, finished 21:14:46 2022-09-25

In [445]:

```
round(df_r_results,4)
```

executed in 21ms, finished 21:14:50 2022-09-25

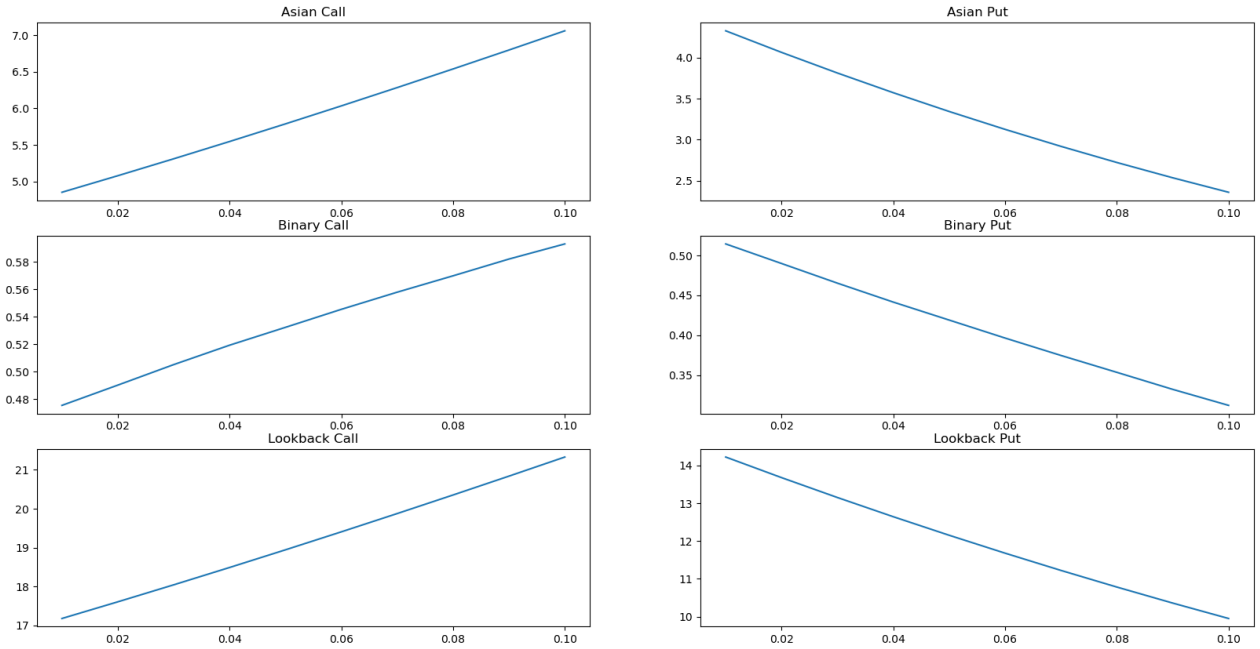
Out[445]:

Risk-free Rate	Asian Call	Asian Put	Binary Call	Binary Put	Lookback Call	Lookback Put
0.01	4.8509	4.3280	0.4755	0.5145	17.1794	14.2212
0.02	5.0775	4.0648	0.4902	0.4900	17.6076	13.6764
0.03	5.3092	3.8131	0.5051	0.4653	18.0444	13.1504
0.04	5.5458	3.5729	0.5193	0.4415	18.4898	12.6425
0.05	5.7872	3.3438	0.5322	0.4190	18.9437	12.1522
0.06	6.0335	3.1259	0.5453	0.3964	19.4057	11.6792
0.07	6.2842	2.9186	0.5578	0.3746	19.8756	11.2230
0.08	6.5392	2.7218	0.5697	0.3534	20.3529	10.7832
0.09	6.7984	2.5353	0.5819	0.3320	20.8370	10.3594
0.10	7.0615	2.3586	0.5929	0.3119	21.3279	9.9519

In [446]:

```
# Plot graph iteratively
fig, ax = plt.subplots(3,2, figsize=(20,10))

ax[0,0].plot(df_r_results.iloc[:,[0]])
ax[0,1].plot(df_r_results.iloc[:,[1]])
ax[1,0].plot(df_r_results.iloc[:,[2]])
ax[1,1].plot(df_r_results.iloc[:,[3]])
ax[2,0].plot(df_r_results.iloc[:,[4]])
ax[2,1].plot(df_r_results.iloc[:,[5]])

# Set axis title
ax[0,0].set_title('Asian Call'), ax[0,1].set_title('Asian Put'),
ax[1,0].set_title('Binary Call'), ax[1,1].set_title('Binary Put'),
ax[2,0].set_title('Lookback Call'), ax[2,1].set_title('Lookback Put')

plt.show()
```

executed in 636ms, finished 21:14:58 2022-09-25



2.4 Volatility

How does the volatility affect the Binary call option price?

In [451]:

```
sigma_list = np.arange(0.1,1.1,0.1)
sigma_results = []
for i in sigma_list:
    ClosedForm = Monte_Carlo_ClosedForm(S0,K,r,i,T,Nsteps,Nsim)
    results = [ClosedForm.AsianCall, ClosedForm.AsianPut,
               ClosedForm.BinaryCall, ClosedForm.BinaryPut,
               ClosedForm.LookbackCall, ClosedForm.LookbackPut]
    sigma_results.append(results)
df_sigma_results = pd.DataFrame(sigma_results)
df_sigma_results.index = sigma_list
df_sigma_results.index.name = 'Volatility'
df_sigma_results.columns = ['Asian Call', 'Asian Put',
                            'Binary Call', 'Binary Put',
                            'Lookback Call', 'Lookback Put']
```

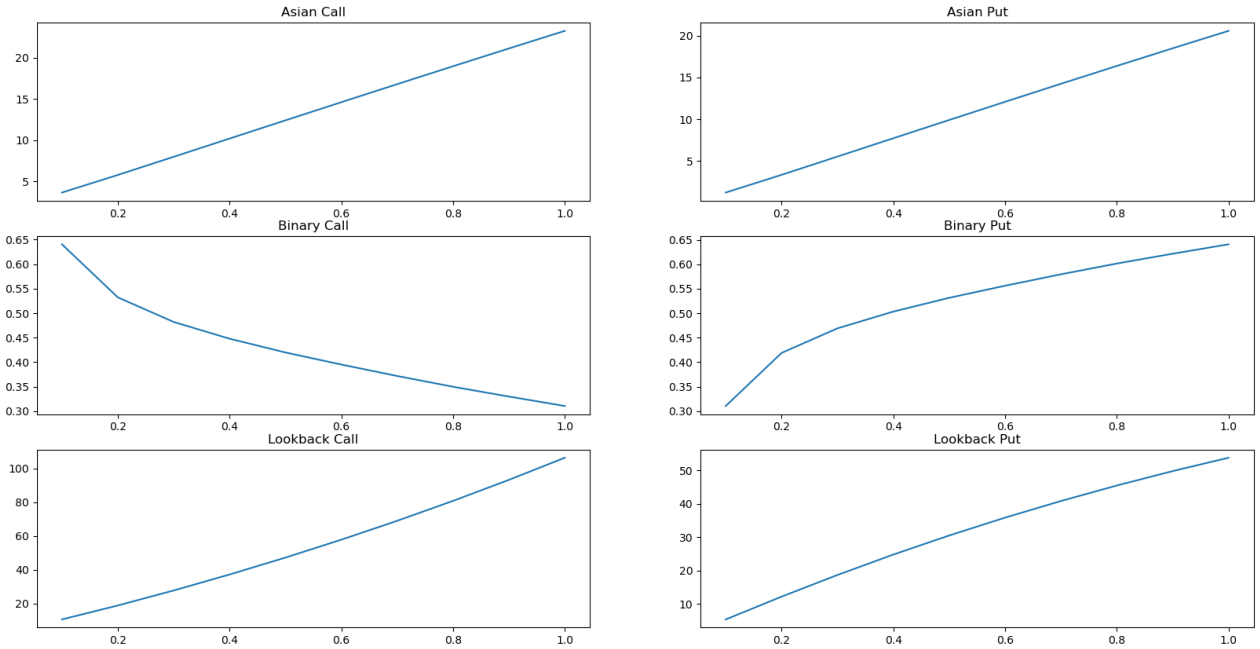
executed in 1m 52.9s, finished 21:21:07 2022-09-25

In [452]:
```
round(df_sigma_results,4)
```
executed in 21ms, finished 21:21:09 2022-09-25

Out[452]:

Volatility	Asian Call	Asian Put	Binary Call	Binary Put	Lookback Call	Lookback Put
0.1	3.6499	1.2208	0.6406	0.3106	10.6814	5.3253
0.2	5.7872	3.3438	0.5322	0.4190	18.9437	12.1522
0.3	7.9891	5.5285	0.4821	0.4691	27.8279	18.6471
0.4	10.2017	7.7207	0.4479	0.5034	37.2754	24.7536
0.5	12.4126	9.9074	0.4198	0.5314	47.2871	30.4732
0.6	14.6154	12.0818	0.3951	0.5561	57.8767	35.8178
0.7	16.8058	14.2394	0.3718	0.5795	69.0635	40.8024
0.8	18.9800	16.3757	0.3498	0.6014	80.8663	45.4438
0.9	21.1362	18.4885	0.3298	0.6214	93.3019	49.7586
1.0	23.2708	20.5738	0.3105	0.6407	106.3934	53.7625

In [453]:
```
# Plot graph iteratively
fig, ax = plt.subplots(3,2, figsize=(20,10))

ax[0,0].plot(df_sigma_results.iloc[:,[0]])
ax[0,1].plot(df_sigma_results.iloc[:,[1]])
ax[1,0].plot(df_sigma_results.iloc[:,[2]])
ax[1,1].plot(df_sigma_results.iloc[:,[3]])
ax[2,0].plot(df_sigma_results.iloc[:,[4]])
ax[2,1].plot(df_sigma_results.iloc[:,[5]])

# Set axis title
ax[0,0].set_title('Asian Call'), ax[0,1].set_title('Asian Put'),
ax[1,0].set_title('Binary Call'), ax[1,1].set_title('Binary Put'),
ax[2,0].set_title('Lookback Call'), ax[2,1].set_title('Lookback Put')

plt.show()
```
executed in 734ms, finished 21:21:12 2022-09-25



2.5 Maturity

In [461]:
```
T_list = [1/252, 1/12, 0.25, 0.5, 1, 2]
T_results = []
for i in T_list:
    ClosedForm = Monte_Carlo_ClosedForm(S0,K,r,sigma,i,Nsteps,Nsim)
    results = [ClosedForm.AsianCall, ClosedForm.AsianPut,
               ClosedForm.BinaryCall, ClosedForm.BinaryPut,
               ClosedForm.LookbackCall, ClosedForm.LookbackPut]
    T_results.append(results)
df_T_results = pd.DataFrame(T_results)
df_T_results.index = T_list
df_T_results.index.name = 'Maturity'
df_T_results.columns = ['Asian Call', 'Asian Put',
                        'Binary Call', 'Binary Put',
                        'Lookback Call', 'Lookback Put']
```

executed in 1m 4.00s, finished 23:54:50 2022-09-25

In [462]:
```
round(df_T_results,4)
```

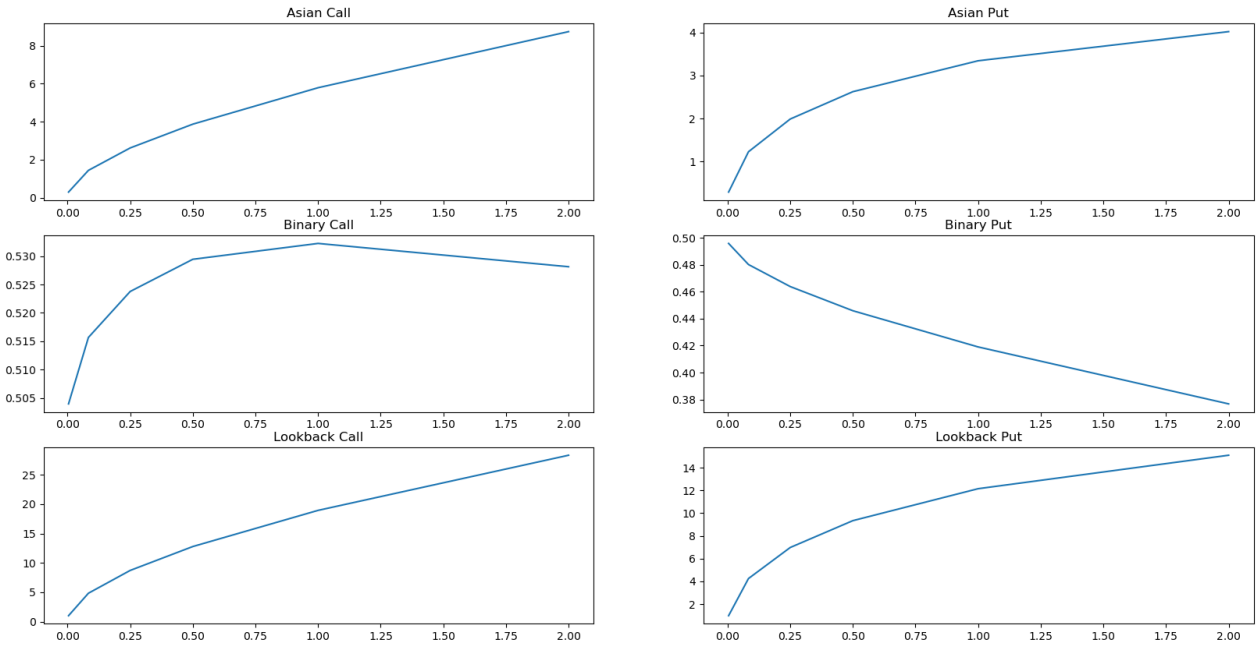
executed in 14ms, finished 23:54:52 2022-09-25

Out[462]:

Maturity	Asian Call	Asian Put	Binary Call	Binary Put	Lookback Call	Lookback Put
0.003968	0.2965	0.2852	0.5040	0.4958	1.0062	0.9768
0.083333	1.4389	1.2246	0.5157	0.4802	4.8321	4.2469
0.250000	2.6181	1.9866	0.5238	0.4638	8.7152	6.9782
0.500000	3.8701	2.6237	0.5294	0.4459	12.7847	9.3407
1.000000	5.7872	3.3438	0.5322	0.4190	18.9437	12.1522
2.000000	8.7393	4.0226	0.5281	0.3767	28.3347	15.0970

In [450]:
```
# Plot graph iteratively
fig, ax = plt.subplots(3,2, figsize=(20,10))

ax[0,0].plot(df_T_results.iloc[:,[0]])
ax[0,1].plot(df_T_results.iloc[:,[1]])
ax[1,0].plot(df_T_results.iloc[:,[2]])
ax[1,1].plot(df_T_results.iloc[:,[3]])
ax[2,0].plot(df_T_results.iloc[:,[4]])
ax[2,1].plot(df_T_results.iloc[:,[5]])

# Set axis title
ax[0,0].set_title('Asian Call'), ax[0,1].set_title('Asian Put'),
ax[1,0].set_title('Binary Call'), ax[1,1].set_title('Binary Put'),
ax[2,0].set_title('Lookback Call'), ax[2,1].set_title('Lookback Put')

plt.show()
```

executed in 638ms, finished 21:18:19 2022-09-25



2.6 Simulated Methods

In [460]:
```
ClosedForm = Monte_Carlo_ClosedForm(S0,K,r,sigma,T,Nsteps,Nsim)
ClosedForm_df = pd.DataFrame([ClosedForm.AsianCall, ClosedForm.AsianPut,
                    ClosedForm.BinaryCall, ClosedForm.BinaryPut,
                    ClosedForm.LookbackCall, ClosedForm.LookbackPut]).T

EulerMaruyama = Monte_Carlo_EulerMaruyama(S0,K,r,sigma,T,Nsteps,Nsim)
EulerMaruyama_df = pd.DataFrame([EulerMaruyama.AsianCall, EulerMaruyama.AsianPut,
                    EulerMaruyama.BinaryCall, EulerMaruyama.BinaryPut,
                    EulerMaruyama.LookbackCall, EulerMaruyama.LookbackPut]).T

Milstein = Monte_Carlo_Milstein(S0,K,r,sigma,T,Nsteps,Nsim)
Milstein_df = pd.DataFrame([Milstein.AsianCall, Milstein.AsianPut,
                    Milstein.BinaryCall, Milstein.BinaryPut,
                    Milstein.LookbackCall, Milstein.LookbackPut]).T

results_df1 = pd.concat([ClosedForm_df, EulerMaruyama_df])
results_df2 = pd.concat([results_df1, Milstein_df])

results_df2.index = ['Closed Form', 'Euler-Maruyama', 'Milstein']
results_df2.index.name = 'Methods'
results_df2.columns = ['Asian Call', 'Asian Put',
                       'Binary Call', 'Binary Put',
                       'Lookback Call', 'Lookback Put']

round(results_df2,6)
```

executed in 35.3s, finished 23:43:20 2022-09-25

Out[460]:

Methods	Asian Call	Asian Put	Binary Call	Binary Put	Lookback Call	Lookback Put
Closed Form	5.787223	3.343797	0.532241	0.418988	18.943660	12.152153
Euler-Maruyama	5.787142	3.343850	0.532327	0.418902	18.943295	12.152118
Milstein	5.787121	3.343720	0.532241	0.418988	18.943288	12.151907

In [591]:
```
EM_diff = EM_CF/result_np[0]*100
EM_diff.round(4)
```

executed in 5ms, finished 22:24:47 2022-09-28

Out[591]:
```
array([0.0014, 0.0016, 0.0161, 0.0204, 0.0019, 0.0003])
```

In [592]:
```
M_diff = M_CF/result_np[0]*100
M_diff.round(4)
```

executed in 5ms, finished 22:24:59 2022-09-28

Out[592]:
```
array([0.0018, 0.0023, 0.    , 0.    , 0.002 , 0.002 ])
```

In [388]:
```
result_np = results_df2.values
EM_CF = abs(result_np[1] - result_np[0])
M_CF = abs(result_np[2] - result_np[0])

comparison = []
for i in range(len(result_np[0])):
    if EM_CF[i] < M_CF[i]:
        results = 'Euler-Maruyama'
    else:
        results = 'Milstein'
    comparison.append(results)

header = ['Asian Call', 'Asian Put', 'Binary Call', 'Binary Put', 'Lookback Call', 'Lookback Put']
table = [[comparison[0],comparison[1],comparison[2],
          comparison[3],comparison[4],comparison[5]]]

print(tabulate(table,header))
```

executed in 7ms, finished 19:50:52 2022-09-25

```
Asian Call      Asian Put       Binary Call    Binary Put     Lookback Call    Lookback Put
--------------  --------------  -------------  -------------  ---------------  --------------
Euler-Maruyama  Euler-Maruyama  Milstein       Milstein       Euler-Maruyama   Euler-Maruyama
```

3 Interesting Observations

3.1 How does Volatility Affect the Binary Call Option Prices?

In [486]:
```
vol_list = np.arange(0,5.1,0.05)
```

executed in 3ms, finished 12:22:58 2022-09-26

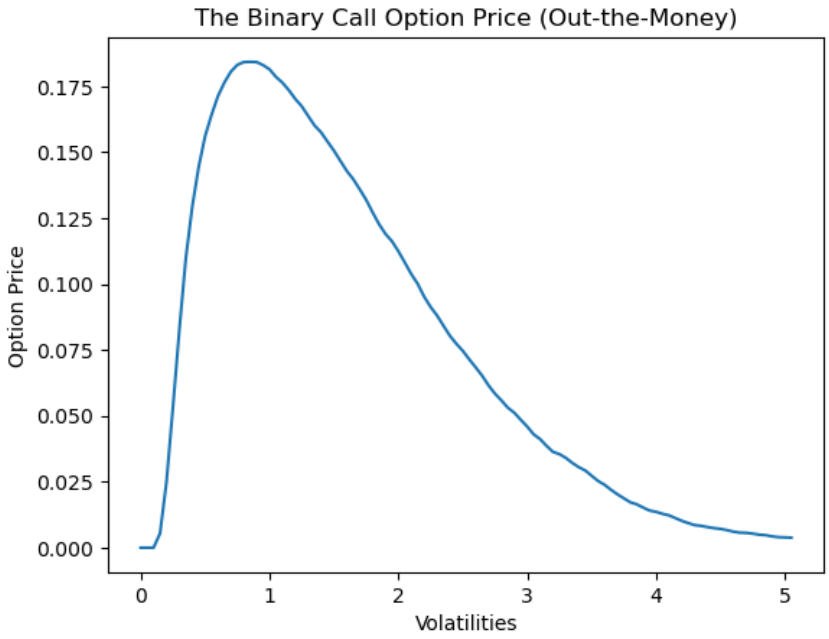
Out the Money

In [491]:
```
K_OTM = 150
price_OTM = []
for i in vol_list:
    Vol_OTM = Monte_Carlo_ClosedForm(S0,K_OTM,r,i,T,Nsteps,10000)
    results = [Vol_OTM.BinaryCall, Vol_OTM.BinaryPut]
    price_OTM.append(results)
df_OTM_results = pd.DataFrame(price_OTM)
df_OTM_results.index = vol_list
df_OTM_results.index.name = 'Volatility'
df_OTM_results.columns = ['Binary Call', 'Binary Put']
```

executed in 1m 43.4s, finished 12:25:24 2022-09-26

In [497]:
```
plt.plot(df_OTM_results.iloc[:,[0]])
plt.xlabel('Volatilities')
plt.ylabel('Option Price')
plt.title('The Binary Call Option Price (Out-the-Money)')
plt.show()
```

executed in 212ms, finished 12:33:05 2022-09-26



In the Money

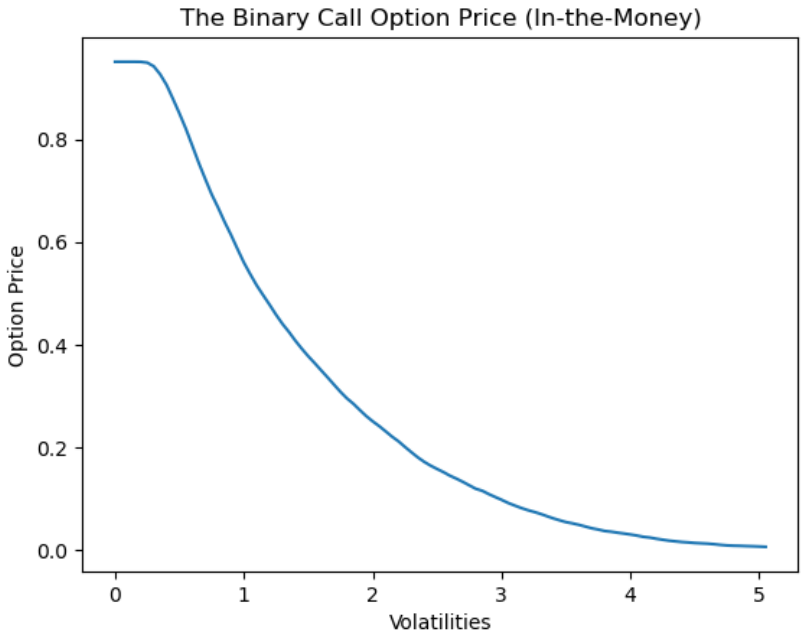
In [514]:
```
K_ITM = 50
price_ITM = []
for i in vol_list:
    Vol_ITM = Monte_Carlo_ClosedForm(S0,K_ITM,r,i,T,Nsteps,10000)
    results = [Vol_ITM.BinaryCall, Vol_ITM.BinaryPut]
    price_ITM.append(results)
df_ITM_results = pd.DataFrame(price_ITM)
df_ITM_results.index = vol_list
df_ITM_results.index.name = 'Volatility'
df_ITM_results.columns = ['Binary Call', 'Binary Put']
```

executed in 1m 35.7s, finished 12:56:26 2022-09-26

In [515]:

```
plt.plot(df_ITM_results.iloc[:,[0]])
plt.xlabel('Volatilities')
plt.ylabel('Option Price')
plt.title('The Binary Call Option Price (In-the-Money)')
plt.show()
```

executed in 97ms, finished 12:56:29 2022-09-26



At the Money

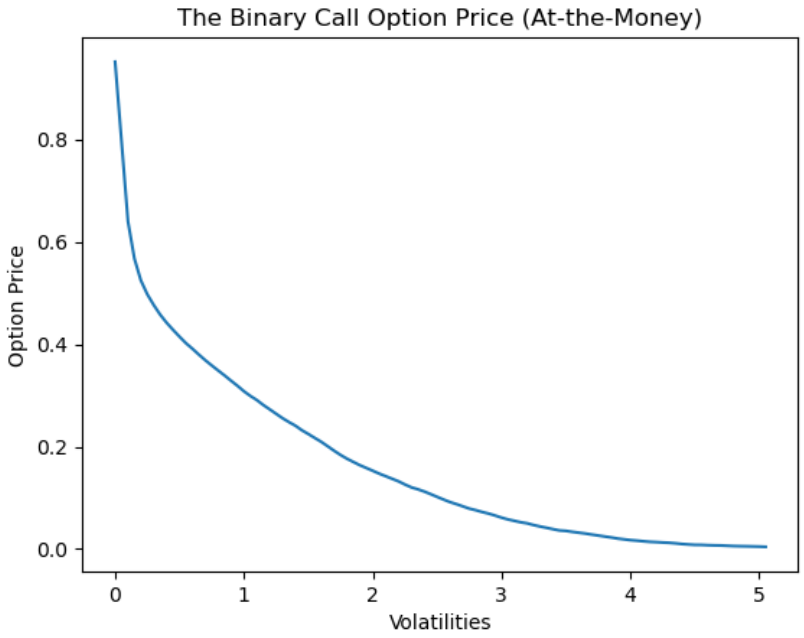
In [516]:

```
K_ATM = 100
price_ATM = []
for i in vol_list:
    Vol_ATM = Monte_Carlo_ClosedForm(S0,K_ATM,r,i,T,Nsteps,10000)
    results = [Vol_ATM.BinaryCall, Vol_ATM.BinaryPut]
    price_ATM.append(results)
df_ATM_results = pd.DataFrame(price_ATM)
df_ATM_results.index = vol_list
df_ATM_results.index.name = 'Volatility'
df_ATM_results.columns = ['Binary Call', 'Binary Put']
```

executed in 1m 36.1s, finished 12:58:16 2022-09-26

In [517]:
```
plt.plot(df_ATM_results.iloc[:,[0]])
plt.xlabel('Volatilities')
plt.ylabel('Option Price')
plt.title('The Binary Call Option Price (At-the-Money)')
plt.show()
```

executed in 100ms, finished 12:58:19 2022-09-26



3.2 How does Maturity Affect the Binary Call Option Prices?

In [555]:
```
maturity_list = np.arange(0.1,2.1,0.05)
```

executed in 3ms, finished 13:28:18 2022-09-26

Out the Money

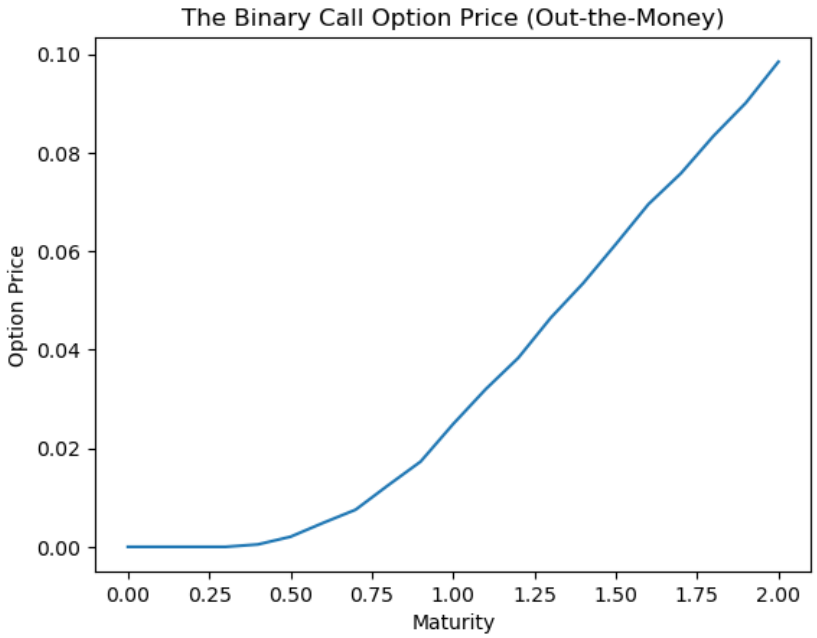
In [546]:
```
K_OTM = 150
price_OTM = []
for i in maturity_list:
    Mat_OTM = Monte_Carlo_ClosedForm(S0,K_OTM,r,sigma,i,Nsteps,10000)
    results = [Mat_OTM.BinaryCall, Mat_OTM.BinaryPut]
    price_OTM.append(results)
df_OTM_results = pd.DataFrame(price_OTM)
df_OTM_results.index = maturity_list
df_OTM_results.index.name = 'Maturity'
df_OTM_results.columns = ['Binary Call', 'Binary Put']
```

executed in 20.0s, finished 13:26:06 2022-09-26

In [547]:

```
plt.plot(df_OTM_results.iloc[:,[0]])
plt.xlabel('Maturity')
plt.ylabel('Option Price')
plt.title('The Binary Call Option Price (Out-the-Money)')
plt.show()
```

executed in 124ms, finished 13:26:09 2022-09-26



In the Money

In [548]:

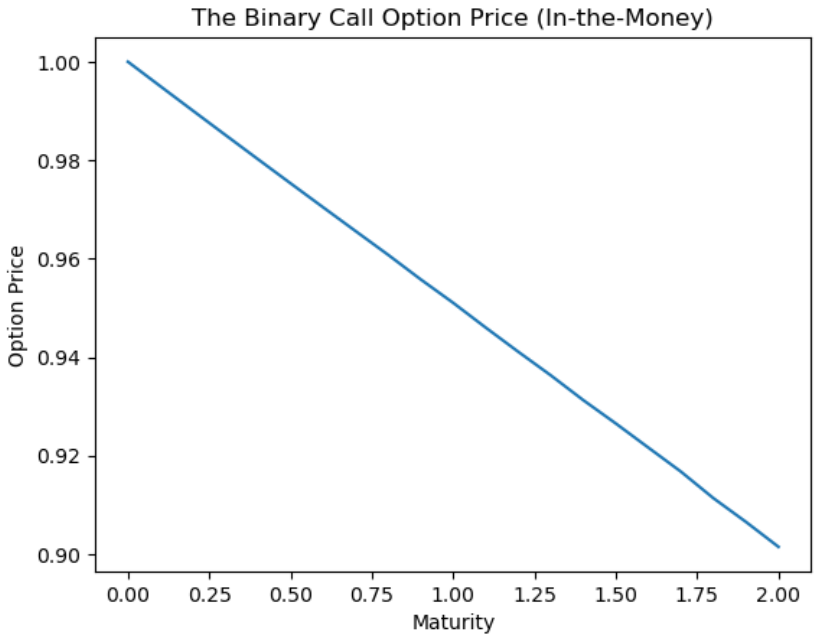
```
K_ITM = 50
price_ITM = []
for i in maturity_list:
    Mat_ITM = Monte_Carlo_ClosedForm(S0,K_ITM,r,sigma,i,Nsteps,10000)
    results = [Mat_ITM.BinaryCall, Mat_ITM.BinaryPut]
    price_ITM.append(results)
df_ITM_results = pd.DataFrame(price_ITM)
df_ITM_results.index = maturity_list
df_ITM_results.index.name = 'Maturity'
df_ITM_results.columns = ['Binary Call', 'Binary Put']
```

executed in 20.5s, finished 13:26:35 2022-09-26

In [549]:

```
plt.plot(df_ITM_results.iloc[:,[0]])
plt.xlabel('Maturity')
plt.ylabel('Option Price')
plt.title('The Binary Call Option Price (In-the-Money)')
plt.show()
```

executed in 113ms, finished 13:26:38 2022-09-26



At the Money

In [558]:

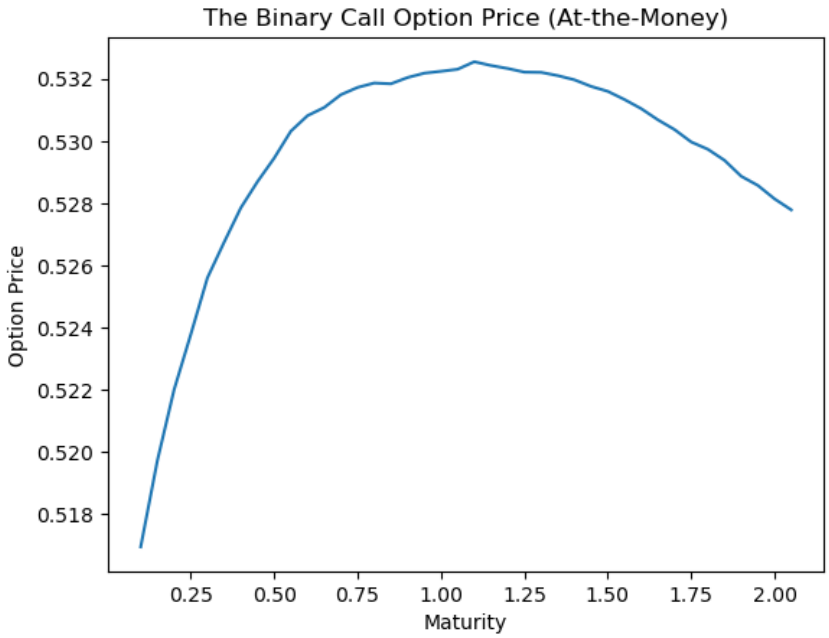
```
K_ATM = 100
price_ATM = []
for i in maturity_list:
    Mat_ATM = Monte_Carlo_ClosedForm(S0,K_ATM,r,sigma,i,Nsteps,100000)
    results = [Mat_ATM.BinaryCall, Mat_ATM.BinaryPut]
    price_ATM.append(results)
df_ATM_results = pd.DataFrame(price_ATM)
df_ATM_results.index = maturity_list
df_ATM_results.index.name = 'Maturity'
df_ATM_results.columns = ['Binary Call', 'Binary Put']
```

executed in 6m 48s, finished 13:36:05 2022-09-26

In [559]:

```
plt.plot(df_ATM_results.iloc[:,[0]])
plt.xlabel('Maturity')
plt.ylabel('Option Price')
plt.title('The Binary Call Option Price (At-the-Money)')
plt.show()
```

executed in 152ms, finished 13:36:08 2022-09-26



3.3 Convergence

In [562]:

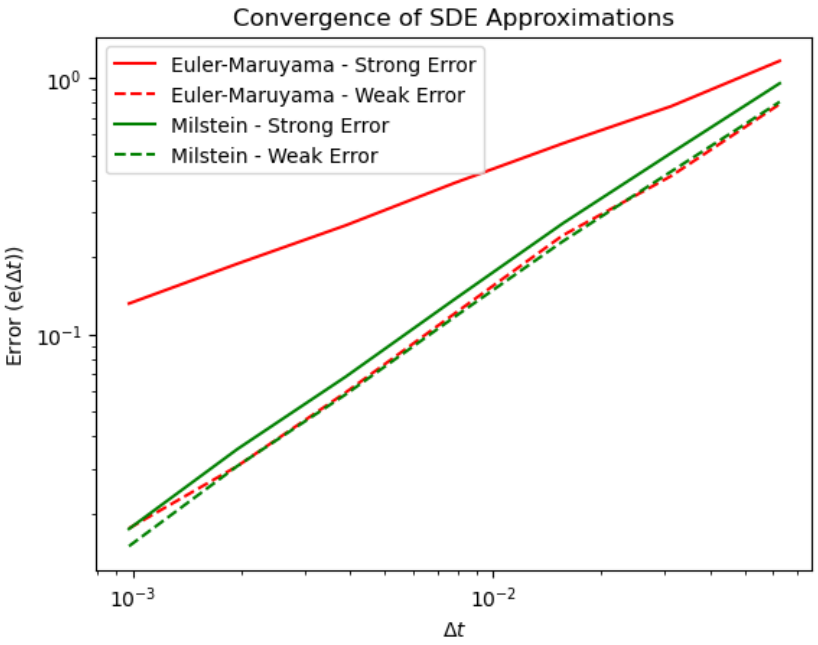
```python
# SDE model parameters
mu, sigma, X0 = 2, 1, 1

# Initiate dt grid and lists to store errors
str_err_em, str_err_mil, weak_err_em, weak_err_mil = [], [], [], []
dt_grid = [2 ** (R-10) for R in range(7)]
mc = 10000

# Loop over values of dt
for Dt in dt_grid:

    # Setup discretized grid
    t = np.arange(Dt, 1 + Dt, Dt)
    n = len(t)

    # Initiate vectors to store errors and time series
    err_em, err_mil = np.zeros(n), np.zeros(n)
    Y_sum, Xem_sum, Xmil_sum = np.zeros(n), np.zeros(n), np.zeros(n)

    # Generate many sample paths
    for i in range(mc):

        # Create Brownian Motion
        np.random.seed(i)
        dB = np.sqrt(Dt) * np.random.randn(n)
        B  = np.cumsum(dB)

        # Exact solution
        Y = X0 * np.exp((mu - 0.5*sigma**2)*t + sigma * B)

        # Simulate stochastic processes
        Xemt, Xmilt, Xem, Xmil = X0, X0, [], []
        for j in range(n):

            # Euler-Maruyama
            Xemt += mu*Xemt* Dt + sigma * Xemt * dB[j]
            Xem.append(Xemt)

            # Milstein
            Xmilt += mu*Xmilt*Dt + sigma*Xmilt*dB[j]
                     + 0.5*sigma**2*Xmilt*(dB[j]**2 - Dt)
            Xmil.append(Xmilt)

        # Compute strong errors and add to those across from other sample paths
        err_em  += abs(Y - Xem)
        err_mil += abs(Y - Xmil)

        # Add Y and X values to previous sample paths
        Y_sum += Y
        Xem_sum += Xem
        Xmil_sum += Xmil

    # Compute mean of absolute errors and find maximum (strong error)
    str_err_em.append(max(err_em / mc))
    str_err_mil.append(max(err_mil / mc))

    # Compute error of means and find maximum (weak error)
    weak_err_em.append(max(abs(Y_sum - Xem_sum)/mc))
    weak_err_mil.append(max(abs(Y_sum - Xmil_sum)/mc))
```

executed in 54.5s, finished 17:32:12 2022-09-27

In [569]:

```
# Plot
plt.loglog(dt_grid, str_err_em, label="Euler-Maruyama - Strong Error",color='red')
plt.loglog(dt_grid, weak_err_em, label="Euler-Maruyama - Weak Error",color='red',ls='--')
plt.loglog(dt_grid, str_err_mil, label="Milstein - Strong Error",color='green')
plt.loglog(dt_grid, weak_err_mil, label="Milstein - Weak Error",color='green',ls='--')
plt.title('Convergence of SDE Approximations')
plt.xlabel('$\Delta t$'); plt.ylabel('Error (e($\Delta t$))'); plt.legend(loc=2);
plt.show()
```

executed in 512ms, finished 17:35:24 2022-09-27



3.4 Antithetic Sampling

In [4]:

```
def normfit(data, confidence=0.95):
    a = 1.0 * np.array(data)
    n = len(a)
    m, se = np.mean(a), si.sem(a)
    h = se * si.t.ppf((1 + confidence) / 2., n - 1)
    var = np.var(data, ddof=1) # Divide by n-1
    sigma = np.sqrt(var)
    return m, sigma, np.hstack((m-h,m+h))
```

executed in 4ms, finished 09:16:55 2022-09-29

In [5]:

```
# define simulation function
def simulate_path(s0,E,mu,sigma,horizon,timesteps,n_sims,sim_method='Closed Form',option='Asian',COP='

    # set the random seed for reproducibility
    random.seed(10000)

    # read parameters
    S0 = s0                 # initial spot price
    K = E                   # strike price
    r = mu                  # mu = rf in risk neutral framework
    T = horizon             # time horizon
    t = timesteps           # number of time steps
    n = n_sims              # number of simulation

    # define dt
    dt = T/t                # length of time interval

    # simulate 'n' asset price path with 't' timesteps
    S = zeros((t,n))
    S[0] = S0

    for i in range(0, t-1):
        w = random.standard_normal(n)

        if sim_method == 'Closed Form':
            S[i+1] = S[i] * np.exp((r - 0.5*sigma**2)*dt + sigma*sqrt(dt)*w)

        if sim_method == 'Euler-Maruyama':
            S[i+1] = S[i] * (1 + r*dt + sigma*sqrt(dt)*w)

        if sim_method == 'Milstein':
            S[i+1] = S[i] * (1 + r*dt + sigma*sqrt(dt)*w + 0.5*sigma**2*(w**2-1)*dt)

    if option == 'Asian':
        average = S.mean(axis = 0)

        if COP == 'Call':
            payout = exp(-r*T) * maximum(0, average - K)
        else:
            payout = exp(-r*T) * maximum(0, K - average)

    if option == 'Binary':

        if COP == 'Call':
            Ind = 1*((S[-1]-K)>0)
        else:
            Ind = 1*(K-(S[-1])>0)

        payout = exp(-r*T) * Ind

    if option == 'Lookback':
        Opt = S.max(axis = 0)

        if COP == 'Call':
            payout = exp(-r*T) * maximum(0, Opt - K)
        else:
            payout = exp(-r*T) * maximum(0, K - Opt)


    price, SigPrice, CI = normfit(payout)

    std_error = (CI[1]-CI[0])/price

    return S, price, std_error
```

executed in 10ms, finished 09:16:56 2022-09-29

In [31]:

```
# define simulation function
def simulate_path_av(s0,E,mu,sigma,horizon,timesteps,n_sims,sim_method='Closed Form',option='Asian',CO

    # set the random seed for reproducibility
    random.seed(10000)

    # read parameters
    S0 = s0                 # initial spot price
    K = E                   # strike price
    r = mu                  # mu = rf in risk neutral framework
    T = horizon             # time horizon
    t = timesteps           # number of time steps
    n = n_sims              # number of simulation
    n_av = int(n/2)

    # define dt
    dt = T/t                # length of time interval

    # simulate 'n' asset price path with 't' timesteps
    Su = zeros((t,n_av))
    Sd = zeros((t,n_av))
    Su[0] = S0
    Sd[0] = S0

    for i in range(0, t-1):
        w = random.standard_normal(n_av)

        if sim_method == 'Closed Form':
            Su[i+1] = Su[i] * np.exp((r - 0.5*sigma**2)*dt + sigma*sqrt(dt)*w)
            Sd[i+1] = Su[i] * np.exp((r - 0.5*sigma**2)*dt + sigma*sqrt(dt)*(-w))

        if sim_method == 'Euler-Maruyama':
            Su[i+1] = Su[i] * (1 + r*dt + sigma*sqrt(dt)*w)
            Sd[i+1] = Sd[i] * (1 + r*dt + sigma*sqrt(dt)*(-w))

        if sim_method == 'Milstein':
            Su[i+1] = Su[i] * (1 + r*dt + sigma*sqrt(dt)*w + 0.5*sigma**2*(w**2-1)*dt)
            Sd[i+1] = Sd[i] * (1 + r*dt + sigma*sqrt(dt)*(-w) + 0.5*sigma**2*((-w)**2-1)*dt)

    S = np.hstack((Su,Sd))

    if option == 'Asian':
        average = S.mean(axis = 0)

        if COP == 'Call':
            payout = exp(-r*T) * maximum(0, average - K)
        else:
            payout = exp(-r*T) * maximum(0, K - average)

    if option == 'Binary':

        if COP == 'Call':
            Ind = 1*((S[-1]-K)>0)
        else:
            Ind = 1*(K-(S[-1])>0)

        payout = exp(-r*T) * Ind

    if option == 'Lookback':
        Opt = S.max(axis = 0)

        if COP == 'Call':
            payout = exp(-r*T) * maximum(0, Opt - K)
        else:
            payout = exp(-r*T) * maximum(0, K - Opt)


    price, SigPrice, CI = normfit(payout)

    std_error = (CI[1]-CI[0])/price

    return S, price, std_error
```

executed in 14ms, finished 10:02:05 2022-09-29

Asian Options

In [32]:

```
m_list = [1,2,3,4,5,6]
N = [10** m for m in m_list]
```

executed in 3ms, finished 10:02:06 2022-09-29

In [54]:

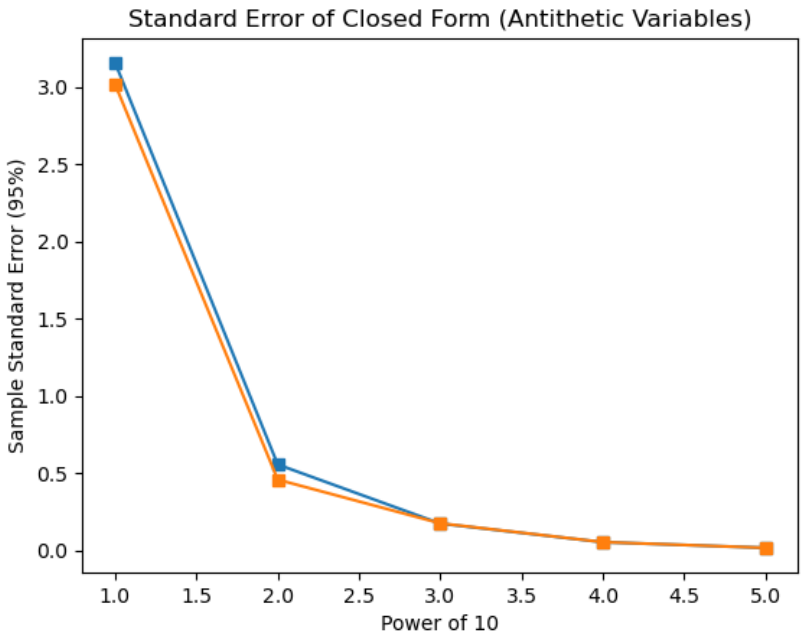
```
option = 'Asian'
COP = 'Call'
sim_method = 'Closed Form'
# price_list = []
# path_list = []
stderror_list = []
stderror_list_av = []
sim_times = [10**1, 10**2, 10**3, 10**4, 10**5]
nn = [1, 2, 3, 4, 5]
for i in sim_times:
#     sim_price = simulate_path(S0,K,r,sigma,T,Nsteps,i,sim_method,option,COP)[1]
#     sim_path  = simulate_path(S0,K,r,sigma,T,Nsteps,i,sim_method,option,COP)[0]
    sim_error    = simulate_path(S0,K,r,sigma,T,Nsteps,i,sim_method,option,COP)[2]
    sim_error_av = simulate_path_av(S0,K,r,sigma,T,Nsteps,i,sim_method,option,COP)[2]
#     price_list.append(sim_price)
#     path_list.append(sim_path)
    stderror_list.append(sim_error)
    stderror_list_av.append(sim_error_av)

stderror_df = pd.DataFrame(stderror_list)
stderror_av_df = pd.DataFrame(stderror_list_av)
stderror_sample = pd.concat((stderror_df,stderror_av_df), axis=1)
stderror_sample.columns = ['Standard', 'Antithetic Variables ']
stderror_sample.index = nn
```

executed in 20.6s, finished 12:15:16 2022-09-29

In [55]:

```
plt.plot(stderror_sample, marker='s')
plt.xlabel('Power of 10')
plt.ylabel('Sample Standard Error (95%)')
plt.title('Standard Error of Closed Form (Antithetic Variables)')
plt.show()
```

executed in 130ms, finished 12:15:17 2022-09-29



In [52]:

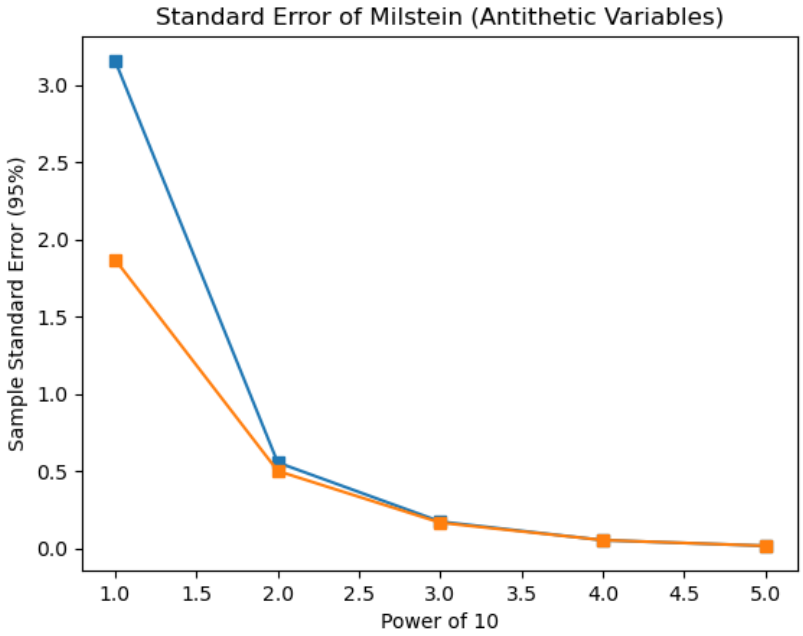
```
option = 'Asian'
COP = 'Call'
sim_method = 'Milstein'
# price_list = []
# path_list = []
stderror_list = []
stderror_list_av = []
sim_times = [10**1, 10**2, 10**3, 10**4, 10**5]
nn = [1, 2, 3, 4, 5]
for i in sim_times:
#     sim_price = simulate_path(S0,K,r,sigma,T,Nsteps,i,sim_method,option,COP)[1]
#     sim_path  = simulate_path(S0,K,r,sigma,T,Nsteps,i,sim_method,option,COP)[0]
    sim_error    = simulate_path(S0,K,r,sigma,T,Nsteps,i,sim_method,option,COP)[2]
    sim_error_av = simulate_path_av(S0,K,r,sigma,T,Nsteps,i,sim_method,option,COP)[2]
#     price_list.append(sim_price)
#     path_list.append(sim_path)
    stderror_list.append(sim_error)
    stderror_list_av.append(sim_error_av)

stderror_df = pd.DataFrame(stderror_list)
stderror_av_df = pd.DataFrame(stderror_list_av)
stderror_sample = pd.concat((stderror_df,stderror_av_df), axis=1)
stderror_sample.columns = ['Standard', 'Antithetic Variables ']
stderror_sample.index = nn
```

executed in 22.1s, finished 12:13:51 2022-09-29

In [53]:

```
plt.plot(stderror_sample, marker='s')
plt.xlabel('Power of 10')
plt.ylabel('Sample Standard Error (95%)')
plt.title('Standard Error of Milstein (Antithetic Variables)')
plt.show()
```

executed in 196ms, finished 12:13:53 2022-09-29



In [56]:

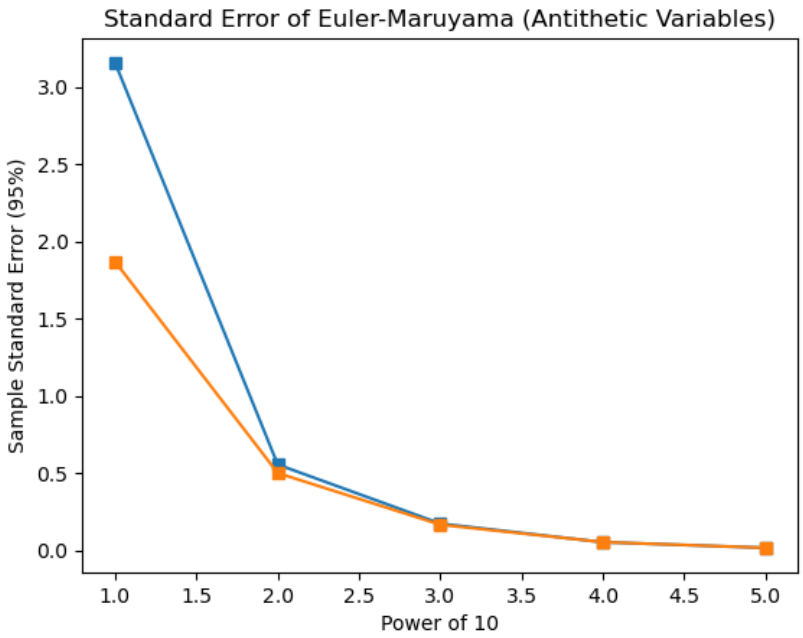
```
option = 'Asian'
COP = 'Call'
sim_method = 'Euler-Maruyama'
# price_list = []
# path_list = []
stderror_list = []
stderror_list_av = []
sim_times = [10**1, 10**2, 10**3, 10**4, 10**5]
nn = [1, 2, 3, 4, 5]
for i in sim_times:
#     sim_price = simulate_path(S0,K,r,sigma,T,Nsteps,i,sim_method,option,COP)[1]
#     sim_path  = simulate_path(S0,K,r,sigma,T,Nsteps,i,sim_method,option,COP)[0]
    sim_error    = simulate_path(S0,K,r,sigma,T,Nsteps,i,sim_method,option,COP)[2]
    sim_error_av = simulate_path_av(S0,K,r,sigma,T,Nsteps,i,sim_method,option,COP)[2]
#     price_list.append(sim_price)
#     path_list.append(sim_path)
    stderror_list.append(sim_error)
    stderror_list_av.append(sim_error_av)

stderror_df = pd.DataFrame(stderror_list)
stderror_av_df = pd.DataFrame(stderror_list_av)
stderror_sample = pd.concat((stderror_df,stderror_av_df), axis=1)
stderror_sample.columns = ['Standard', 'Antithetic Variables ']
stderror_sample.index = nn
```

executed in 15.0s, finished 12:15:34 2022-09-29

In [57]:

```
plt.plot(stderror_sample, marker='s')
plt.xlabel('Power of 10')
plt.ylabel('Sample Standard Error (95%)')
plt.title('Standard Error of Euler-Maruyama (Antithetic Variables)')
plt.show()
```

executed in 128ms, finished 12:15:35 2022-09-29



References

Boyle, P. & Potapchik, A. (2008), 'Prices and sensitivities of asian options: A survey', *Insurance: Mathematics and Economics* **42**(1), 189–211.

Buchen, P. & Konstandatos, O. (2005), 'A NEW METHOD OF PRICING LOOKBACK OPTIONS', *Mathematical Finance* **15**(2), 245–259.

Fadugba, S., Adegboyegun, B., Ogunbiyi, O. et al. (2013), 'On the convergence of euler maruyama method and milstein scheme for the solution of stochastic differential equations', *International Journal of Applied Mathematics and Modeling* **1**(0), 1.

Higham, D. J. (2015*a*), 'An introduction to multilevel Monte Carlo for option valuation', *International Journal of Computer Mathematics* **92**(12), 2347–2360.

Higham, D. J. (2015*b*), 'An introduction to multilevel monte carlo for option valuation', *International Journal of Computer Mathematics* **92**(12), 2347–2360.

Jabbour, G. M. & Liu, Y.-K. (2011), 'Option Pricing And Monte Carlo Simulations', *Journal of Business & Economics Research (JBER)* **3**(9).

Thavaneswaran, A., Appadoo, S. S. & Frank, J. (2013), 'Binary option pricing using fuzzy numbers', *Applied Mathematics Letters* **26**(1), 65–72.

Wikipedia (2022*a*), 'Asian option — Wikipedia, the free encyclopedia', http://en.wikipedia.org/w/index.php?title=Asian%20option&oldid=1049722431. [Online; accessed 25-September-2022].

Wikipedia (2022*b*), 'Lookback option — Wikipedia, the free encyclopedia', http://en.wikipedia.org/w/index.php?title=Lookback%20option&oldid=1101781227. [Online; accessed 25-September-2022].