Module 3 Assignment Report

Bach Van Hoang Bao

September 21, 2023

1 Outline of the financial problem and numerical procedure

1.1 Financial Problem

Given the expected value of the discounted payoff under the risk-neutral density $\mathbb{Q}$

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

Initial example data:

$$\begin{aligned} \text{Today's stock price } S_0 &= 100 \\ \text{Strike E} &= 100 \\ \text{Time to expiry } (T-t) &= 1 \text{ year} \\ \text{volatility} \sigma &= 20\% \\ \text{constant risk-free interest rate } r &= 5\% \end{aligned}$$

[1]:
```
# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
# Define parameters and variables
S0 = 100
E = 100
T = 1
vol = 0.2
risk_free_rate = 0.05
```

In this report, I will use the Monte Carlo method to simulate risk neutral random walk and then value the option under the risk neutral framework.

The pricing algorithm:

1. Simulate the risk-neutral random walk starting at today's value of the asset over the required time horizon. This gives one realization of the underlying price path.
2. For this realization calculate the option payoff.
3. Perform many more such realizations over the time horizon.

4. Calculate the average payoff over all realizations.
5. Take the present value of this average; this is the option value.

1.2 Simulating path using Euler - Maruyama Scheme

A geometric Brownian motion with a stochastic differential equation (SDE) is given as:

$$dS = rS\,dt + \sigma S\,dW$$

Where:

- S is the price of the underlying
- σ is constant volatility
- r is the constant risk-free interest rate
- X is the brownian motion.

Consider $V(S) = \log S$

First order derivative:

$$\frac{dV}{dS} = \frac{1}{S} \tag{1}$$

Second order derivative:

$$\frac{d^2V}{dS^2} = \frac{-1}{S^2} \tag{2}$$

Using this result for $V = V(S,t)$ [1]:

$$dV = \left(rS\frac{dV}{dS} + \frac{1}{2}\sigma^2 S^2 \frac{d^2V}{dS^2}\right)dt + \left(\sigma S\frac{dV}{dS}\right)dW \tag{3}$$

Subtitude (1) and (2) into (3) we have:

$$d(\log S) = \left(rS\left(\frac{1}{S}\right) + \frac{1}{2}\sigma^2 S^2\left(-\frac{1}{S^2}\right)\right)dt + \sigma S\left(\frac{1}{S}\right)dW$$

$$= \left(r - \frac{1}{2}\sigma^2\right)dt + \sigma dW$$

Integrating both sides between 0 and t

$$\int_0^t d(\log S) = \int_0^t \left(r - \frac{1}{2}\sigma^2\right) d\tau + \int_0^t \sigma dW \quad (t > 0)$$

$$\log \frac{S_t}{S_0} = \left(r - \frac{1}{2}\sigma^2\right) t + \sigma(W(t) - W(0))$$

Assuming $W(0) = 0$ and $S(0) = S_0$, the exact solution becomes:

$$S(t) = S_0 \exp\left(\left(r - \frac{1}{2}\sigma^2\right) t + \sigma W(t)\right) \tag{4}$$

Using Euler - Maruyama scheme [2]:

$$dS = rS\, dt + \sigma S\sqrt{dt}\, \phi$$

where ϕ is from a standardized Normal distribution. This method has an error of $O(\delta t)$.

Using (4) we can derive the Euler - Maruyama scheme for our problem:

$$S_{t+dt} = S_t \exp\left(\left(r - \frac{1}{2}\sigma^2\right) dt + \sigma\sqrt{dt}\, \phi\right) \tag{5}$$

For the random number generator for generating standard normal variable ϕ, I will use the `np.random.standard_normal()` method.

[3]:
```
# define simulation function
def simulate_path(s0, risk_free_rate, vol, horizon, timesteps, n_sims):
    # choose the random seed
    seed = 2023
    rng = np.random.default_rng(seed)
    # read the params
    S0 = s0
    r = risk_free_rate
    T = horizon
    t = timesteps
    n = n_sims
    # define dt
    dt = T/t
    # simulate path
    S = np.zeros((t,n))
    S[0] = S0
    for i in range(0, t-1):
        w = rng.standard_normal(n)
        S[i+1] = S[i] * (1+ r*dt + vol*np.sqrt(dt)*w)
    return S
```

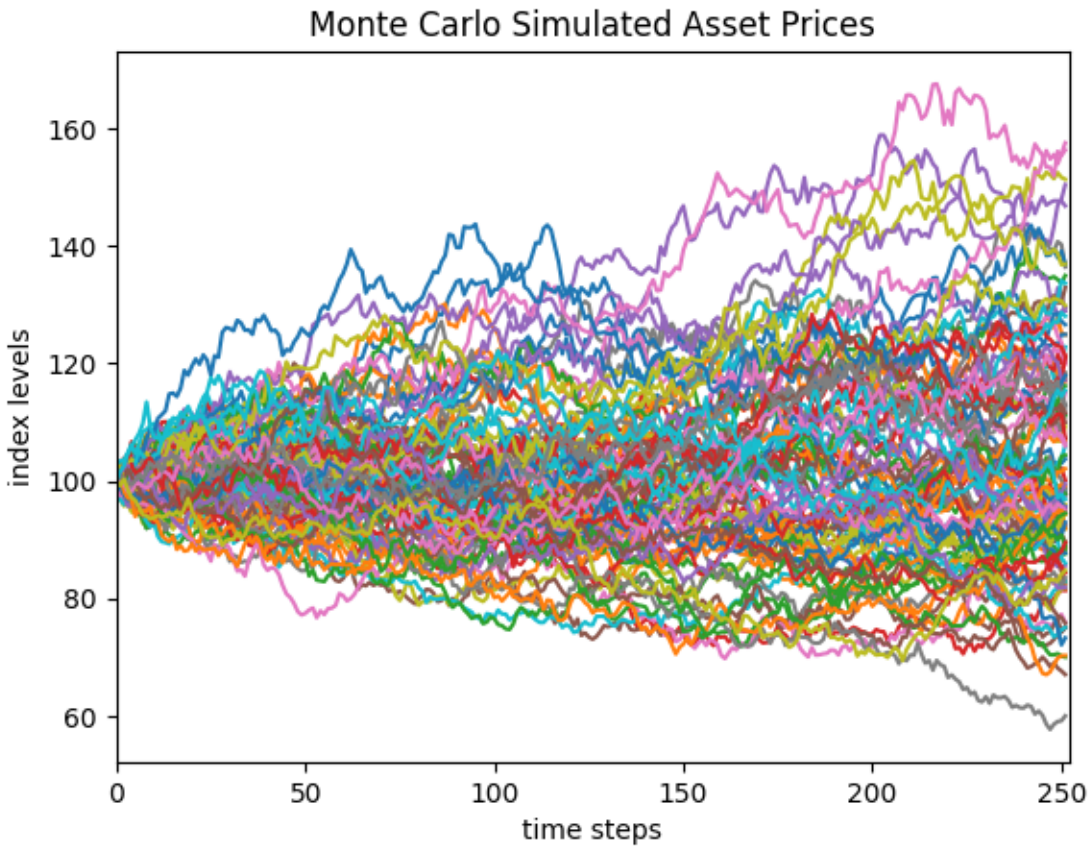
We will choose:

1. Number of paths: 100,000
2. Time steps: 252 trading days

[4]:
```
# Monte Carlo parameters
n = 100000
t = 252
```

[5]:
```
# Assign simulated price path to dataframe for analysis and plotting
S = pd.DataFrame(simulate_path(S0,risk_free_rate,vol,T,t,n))
```

[6]:
```
# Plot initial 100 simulated path using matplotlib
plt.plot(S.iloc[:,:100])
plt.xlabel('time steps')
plt.xlim(0,t)
plt.ylabel('index levels')
plt.title('Monte Carlo Simulated Asset Prices');
```



1.3 Exotic Options Pricing

Using the simulation from the previous section, I will calculate the payoff of the options for all realizations of the asset path. Then, I will take the expectation of the option price, discounted back to the present value.

1.3.1 Asian Options Pricing

An Asian option is an option where the payoff depends on the average price of the underlying asset over a certain period of time. Averaging can be either be Arithmetic or Geometric. There are two types of Payoff types: average rate, where averaging price is used in place of underlying price; and erage strike, where averaging price is used in place of strike [3]

Average strike call:

$$\max(S - A, 0)$$

Average strike put:

$$\max(A - S, 0)$$

Average rate call:

$$\max(A - E, 0)$$

Average rate put:

$$\max(E - A, 0)$$

The Average tracking variable:

$$A_i = \frac{1}{i} \sum_{k=1}^{i} S\left(t_k\right)$$

Where i is the total number of sampling dates.

In this function, I will use:

1. Payoff Type: Average rate
2. Type of Average: Athrimetic Average

Under the risk-neutral framework, we assume the asset is going to earn, on average, the risk-free interest rate. Hence, the option value at time t would simply be the discounted value of the expected payoff.

The payoff of the options is given by

$$C(S,t) = e^{-r(T-t)}(\mathbb{E}[\max(A - E), 0])$$

[7]:
```
# Average price
A = S.mean(axis=0)
C0 = np.exp(-risk_free_rate*T) * np.mean(np.maximum(A-E,0))
P0 = np.exp(-risk_free_rate*T) * np.mean(np.maximum(E-A,0))
# Print the values
```

```
print(f"Asian Call Option Value is {C0:0.4f}")
print(f"Asian Put Option Value is {P0:0.4f}")
```

```
Asian Call Option Value is 5.7603
Asian Put Option Value is 3.3464
```

1.3.2 Lookback Options Pricing

The lookback option has a payoff that depends on the maximum or minimum of the realized asset price. There are two types of payoff: The rate and the strike option, also called the **fixed strike** and the **floating strike** respectively [4].

Variable $M_{\max}$ as the realized maximum and $M_{\min}$ as the realized minimum of the asset from the start of the sampling period $t = 0$ until the current time t:

$$M_{\max} = \max_{0 \le \tau \le t} S(\tau)$$

$$M_{\min} = \min_{0 \le \tau \le t} S(\tau)$$

The payoff function becomes

Floating strike lookback call:

$$\max(S - M_{\min}, 0)$$

Floating strike lookback put:

$$\max(M_{\max} - S, 0)$$

Fixed strike lookback call:

$$\max(M_{\max} - E, 0)$$

Fixed strike lookback put:

$$\max(E - M_{\min}, 0)$$

The value of our lookback option is a function of three variables, $V(S, M, t)$. In this function, I will use:

1. Payoff Type: Fixed strike
2. Maximum measurement: Continous

The algorithm is to price option of each path and then calculate the average payoff over all realizations. Take the present value of this average which is the value of the option.

[8]:
```
# Tracking Variable for Maximum and Minimum
M_max = np.max(S,axis=0)
M_min = np.min(S,axis=0)
# Pricing Options
C0 = np.exp(-risk_free_rate*T) * np.mean(np.maximum(M_max-E,0))
P0 = np.exp(-risk_free_rate*T) * np.mean(np.maximum(E-M_min,0))
# Print the values
```

```
print(f"Lookback Call Option Value is {C0:0.4f}")
print(f"Lookback Put Option Value is {P0:0.4f}")
```

```
Lookback Call Option Value is 18.3002
Lookback Put Option Value is 11.7159
```

2 Varying the Initial Data

Using previous code for our Monte Carlo simulation and Asian options pricing, I will create an `asian` function to price an option values for different parameters.

[9]:
```
# Setting the random seed
seed = 2023
rng = np.random.default_rng(seed)
```

[10]:
```
def asian(s0, strike, risk_free_rate, vol, horizon, timesteps, n_sims,␣
↪option_type = "C"):
    # Euler - Maruyama Scheme
    S0 = s0
    E = strike
    r = risk_free_rate
    T = horizon
    t = timesteps
    n = n_sims
    dt = T/t
    S = np.zeros((t,n))
    S[0] = S0
    for i in range(0, t-1):
        w = rng.standard_normal(n)
        S[i+1] = S[i] * (1+ r*dt + vol*np.sqrt(dt)*w)
    S = pd.DataFrame(simulate_path(S0,risk_free_rate,vol,T,t,n))     # Building a␣
↪Dataframe for all simulated path
    # Average price
    A = S.mean(axis=0)
    # Option pricing Algorithm
    if option_type == "C":
        return np.exp(-r*T) * np.mean(np.maximum(A-E,0))
    elif option_type == "P":
        return np.exp(-r*T) * np.mean(np.maximum(E-A,0))
    else:
        print("Wrong Option Type. Use `C` for Call option and `P` for Put option.
↪")
```

And doing the same to create `lookback` function to price Lookback options

[11]:
```
def lookback(s0, strike, risk_free_rate, vol, horizon, timesteps, n_sims,␣
↪option_type = "C"):
    # Euler - Maruyama Scheme
```

```
    S0 = s0
    E = strike
    r = risk_free_rate
    T = horizon
    t = timesteps
    n = n_sims
    dt = T/t
    S = np.zeros((t,n))
    S[0] = S0
    for i in range(0, t-1):
        w = rng.standard_normal(n)
        S[i+1] = S[i] * (1+ r*dt + vol*np.sqrt(dt)*w)
    S = pd.DataFrame(simulate_path(S0,risk_free_rate,vol,T,t,n))     # Building a␣
↪Dataframe for all simulated path

    # Tracking Variable for Maximum and Minimum
    M_max = np.max(S,axis=0)
    M_min = np.min(S,axis=0)

    # Discounted back to PV
    if option_type == "C":
        return np.exp(-r*T) * np.mean(np.maximum(M_max-E,0))
    elif option_type == "P":
        return np.exp(-risk_free_rate*T) * np.mean(np.maximum(E-M_min,0))
    else:
        print("Wrong Option Type. Use `C` for Call option and `P` for Put option.
↪")
```

Testing the function with our previous results for Asian Options

[12]:
```
asian(100,100,0.05,0.2,1,252,100000,"C"), asian(100,100,0.05,0.
↪2,1,252,100000,"P")
```

[12]:
```
(5.760317815971038, 3.346444480615885)
```

And for the Lookback Options

[13]:
```
lookback(100,100,0.05,0.2,1,252,100000,"C"), lookback(100,100,0.05,0.
↪2,1,252,100000,"P")
```

[13]:
```
(18.300180586891152, 11.715854637463458)
```

Both functions are working as expected. In the next section, I will vary the parameters of the functions and observe the effect on option prices.

2.1 Vary the Volatility σ

Now, let's vary the volatility for three different levels: $\sigma_1 = 10$, $\sigma_2 = 20$, and $\sigma_3 = 30$.

[14]:
```
vol_set = [0.15,0.2,0.25]
asian_calls = [asian(100,100,0.05,vol,1,252,100000,"C") for vol in vol_set]
asian_puts = [asian(100,100,0.05,vol,1,252,100000,"P") for vol in vol_set]
lookback_calls = [lookback(100,100,0.05,vol,1,252,100000,"C") for vol in vol_set]
lookback_puts = [lookback(100,100,0.05,vol,1,252,100000,"P") for vol in vol_set]
```

[15]:
```
# Create a dictionary with the sets and headers
data1 = {
    'Volatility': vol_set,
    'Asian Call': asian_calls,
    'Asian Put': asian_puts,
    'Lookback Call': lookback_calls,
    'Lookback Put': lookback_puts
}
# Create a DataFrame
df1 = pd.DataFrame(data1).set_index('Volatility')

df1
```

[15]:

Volatility	Asian Call	Asian Put	Lookback Call	Lookback Put
0.15	4.683266	2.271560	14.255136	8.430421
0.20	5.760318	3.346444	18.300181	11.715855
0.25	6.849325	4.432830	22.482347	14.921957

It appears that both Asian and Lookback options prices increase when volatility rises and decrease when volatility falls. However, it seems that Asian options are less sensitive to changes in volatility. This could be due to the effect of averaging the price of the underlying, which mitigates the impact of sudden changes.

The price of Lookback options is increasing significantly. This could be attributed to the high volatility that increases the probability of extreme prices, consequently raising the maximum value $M_{\max}$ and minimum value $M_{\min}$, thereby increasing the price of the options.

2.2 Vary the inital stock price S_0

Let's vary the initial stock price at three different levels: 90, 100, and 110.

[16]:
```
S0_set = [90,100,110]
asian_calls = [asian(S0,100,0.05,0.2,1,252,100000,"C") for S0 in S0_set]
asian_puts = [asian(S0,100,0.05,0.2,1,252,100000,"P") for S0 in S0_set]
lookback_calls = [lookback(S0,100,0.05,0.2,1,252,100000,"C") for S0 in S0_set]
lookback_puts = [lookback(S0,100,0.05,0.2,1,252,100000,"P") for S0 in S0_set]
```

[17]:
```
# Create a dictionary with the sets and headers
data2 = {
    'S0': S0_set,
    'Asian Call': asian_calls,
    'Asian Put': asian_puts,
```

```
    'Lookback Call': lookback_calls,
    'Lookback Put': lookback_puts
}
# Create a DataFrame
df2 = pd.DataFrame(data2).set_index('S0')

df2
```

[17]:

S0	Asian Call	Asian Put	Lookback Call	Lookback Put
90	1.562839	8.902648	8.920297	20.056563
100	5.760318	3.346444	18.300181	11.715855
110	13.051017	0.883462	29.642493	5.699979

The price of Out of The Money (OTM) options at $S_0 = 90$ is significantly lower compared to At the Money (ATM) and In the Money (ITM) options, as expected. For Asian Options, the price is notably lower. This can be attributed to the effect of averaging the price of the underlying, which makes it more challenging for the average to increase.

In the case of Lookback Options, since the payoff depends on the maximum and minimum of the underlying, it appears that the price of the options doesn't change as drastically

2.3 Vary the strike price E

Let's vary the strike price at three different levels: 90, 100, and 110.

[18]:
```
strike_set = [90,100,110]
asian_calls = [asian(100,strike,0.05,0.2,1,252,100000,"C") for strike in␣
 ↪strike_set]
asian_puts = [asian(100,strike,0.05,0.2,1,252,100000,"P") for strike in␣
 ↪strike_set]
lookback_calls = [lookback(100,strike,0.05,0.2,1,252,100000,"C") for strike in␣
 ↪strike_set]
lookback_puts = [lookback(100,strike,0.05,0.2,1,252,100000,"P") for strike in␣
 ↪strike_set]
```

[19]:
```
# Create a dictionary with the sets and headers
data3 = {
    'Strike': strike_set,
    'Asian Call': asian_calls,
    'Asian Put': asian_puts,
    'Lookback Call': lookback_calls,
    'Lookback Put': lookback_puts
}
# Create a DataFrame
df3 = pd.DataFrame(data3).set_index('Strike')

df3
```

[19]:

Strike	Asian Call	Asian Put	Lookback Call	Lookback Put
90	12.596374	0.670206	27.812475	4.714733
100	5.760318	3.346444	18.300181	11.715855
110	1.987817	9.086238	10.585264	21.228149

These results closely resemble our earlier observations when we varied the initial stock price data. We can still draw the same conclusions for both Asian Options and Lookback Options. The pricing patterns remain consistent, with Asian Options generally having lower prices compared to Lookback Options under similar parameters.

3 Observations and problems encountered

Observations:

- The volatility of the average stock price over time is lower than the volatility of the individual stock. This results in significantly lower prices for Asian Options compared to Lookback Options under the same parameters. As a result, the reduction in the upfront premium in an option contract tends to make Asian Options more appealing to investors.
- The price of Lookback Options is significantly higher than that of Asian Options. This can be attributed to the extreme payoff structure of Lookback Options, which tends to make the contracts more expensive.
- The Risk Neutral Framework makes the pricing of these options relatively straightforward, especially when using the Monte Carlo Method.
- The Euler-Maruyama Scheme is relatively easy to program, but the quality of the entire algorithm depends on the quality of the pseudo-random number generator (RNG) methods. To maintain consistent results, I have adopted a method that creates a global numpy RNG and passes the seed only once. This approach ensures ease of reproduction without affecting the randomness of the results [5].

Problems:

- In the Asian Options pricing function, I haven't utilized the Average Strike payoff and Geometric Average sampling technique.
- In the Lookback Options pricing function, I haven't incorporated the Lookback Floating Strike Payoff and Discrete sampling technique.
- The output of the Euler-Maruyama Scheme in (5) depends on the quality of the pseudo-random number generator (RNG) for ϕ. I only used the `np.random` method and didn't have the time to compare the effects of different RNG methods on the output of our Monte Carlo model.
- The Euler-Maruyama Scheme has an error of $O(\delta t)$. Better approximations, such as the Milstein method with an error of $O(\delta t^2)$, could be implemented to improve the model.

4 Conclusion

Under the risk-neutral framework, the fair value of an option is the present value of the expected payoff at expiry under a risk-neutral random walk for the underlying.

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

In this expression, we see the short-term interest rate playing two distinct roles. First, it is used for discounting the payoff to the present. This is the term $e^{-r(T-t)}$ outside the expectation. Second, the return on the asset in the risk-neutral world is expected to be $rS\,dt$ in a time step dt.

I have successfully priced Asian Options and Lookback Options using the Euler-Maruyama Scheme. The Monte Carlo algorithm is easy to program. The more simulations I use, the better accuracy I will achieve. The algorithm is also useful for pricing strong-path-dependent contracts, such as Asian Options and Lookback Options. The code is easy to reproduce, and I can create a Python class for later use.

After pricing the options, I noticed that the price of Asian Options is lower compared to Lookback Options with similar parameters. I also observed the effect of varying the volatility and strike price on the option price.

I believe using the Finite Difference Method for pricing these options should be optimal since it is well-suited for three-dimensional problems. If I were to do this again, I would like to try the Finite Difference Method and compare the results between the Monte Carlo Simulation and Finite Different Method.

5 References

[1] Module 1 - Lecture 4,Apply Ito III for $V = V(t, X)$

[2] Paul Wilmott on Quantitative Finance, Chapter 80, Page 1266

[3] Paul Wilmott on Quantitative Finance, Chapter 25, Page 428

[4] Paul Wilmott on Quantitative Finance, Chapter 26, Page 445

[5] Good practices with numpy random number generators, Albert Thomas (https://albertcthomas.github.io/good-practices-random-number-generators/)