

---

Lab Number : 09

Due Date : Nov 11, 2020

Student Details :

- Name : AB Satyaprakash
  - Roll Number : 180123062
  - Department : Mathematics and Computing
- 

Formulae used:

Apart from the ones needed for estimation of  $\mu$  and  $\sigma$ , we have used the following formulae/algorithms in the code

1.  $N(\mu, \sigma^2) = \mu + \sigma N(0, 1)$
2. Box-Muller algorithm to generate  $N(0, 1)$
3.  $R_{j+1} = -\log(U)/\lambda$
4.  $X(0) = \log(S(0))$
5.  $X(\tau_{j+1}) = X(\tau_j) + (\mu - 1/2\sigma^2)R_{j+1} + \sigma\sqrt{R_{j+1}} Z_{j+1} + \log Y_j$
6.  $\tau_{j+1} = \tau_j + R_{j+1}$
7. Confidence Interval =  $[\mu_b - 1.96 \sigma_b \sqrt{M}, \mu_b + 1.96 \sigma_b \sqrt{M}]$

Question :

Explanation:

----- Basic Info from prev labs -----

First of all we used **pandas** (a python library) to read the csv file, i.e. the file having the daily adjusted closing stock prices for SBI for **July 1, 2020 to Sept 30, 2020**.

Then we moved on to the specific column of the data and stored it as a list. This list is then used to generate the **U**is given by natural logarithm of ratio of the current and previous day stock prices (more like previous entry, since days were missing in the csv accounting for non-working days).

Using the formulae given in the question pdf, corresponding to the gBm model, we obtained the **estimated** values of  $\mu$  and  $\sigma$ .

---

We choose the value of  $S(0)$  as the **adjusted closing stock price on 30th of September**.

Thus, we have  $S(0)=185.399994$  and  $K = 1.1S(0)$  (as given in the question).

### Part (1):

For this part, we need to make **1000** simulations in each of which we find the value of an Asian Option, which is given as:

$$Y_j = \max\left[K - \left(\frac{1}{N} + 1\right) \sum_{i=1}^{N+1} S(t_i)\right], \text{ where } N=300 \text{ (as given in}$$

question)

We get **1000** values of  $Y$  for each  $j$  in  $[1,1000]$ , and then we can obtain the **mean** (=Asian option price) and **sampling variance** as:

$$\mu = 1/1000 \left( \sum_{i=1}^{1000} Y_i \right) \text{ and } \sigma = 1/1000 \left( \sum_{i=1}^{1000} Y_i - \mu Y \right) \text{ -----eq(A)}$$

Also using **formula 7**, we can find the **95% confidence interval**.

For calculating the value of the option we however need the values of  $S(t_i)$ , which we will obtain by using the **Merton's Jump Diffusion Model**, as we did in the previous assignment.

As required by the question **T=30 and N=300**, so we will make the 301 equally spaced  $t_i$ s starting from 0 and going all the way till 30. Since the time points we require might not coincide with those generated from the above equations, the stock prices have been **linearly interpolated** between each adjacent pair of time points. Using the **formulae 1 to 6**, we apply the Jump Diffusion method, and obtain the stock prices.

Also note that the value of  $\lambda = 0.2$ , which is used to generate the  $R$ (time intervals) using formula 3. Finally we obtain the **output** as : (the terminal output is slightly different from this one).

Option Price	Sampling Variance	95% Confidence Interval
18.490053260497668	149.06226451421333	[9.251078, 27.729028]

As we can see, the **sample variance** is a whopping **149.06**, we can reduce this using **control variates**, which we will do in part (2) now.

## Part (2):

In this part we will make use of the price of a standard **European put option** as a control variate.

The payoff of the option is given as follows:

$$X = \max[(K - S(T)), 0] \text{ , where } T=30$$

We will obtain the X values for 1000 cases, and similar to what we did in part (1), we will make use of equation A, to obtain, average and sampling variance for the option.

We know that  $Y_i(b) = (Y_i - b * (X - E[X]))$ , where  $Y_i$  is defined for the Asian option.

Now we find out the mean and variance of the newly defined  $Y_i(b)$ .

- i) The **mean** will remain the **same** since **control variate is an unbiased estimator**.
- ii) The **variance** can be minimised, by taking the optimal value of b,

$$b = \sigma(xy) / \sigma^2(x)$$

However, since we don't have the exact values of  $\sigma(xy)$  and  $\sigma^2(x)$ , we will make use of the corresponding sample values to estimate b.

$$b = \left( \sum_{i=1}^{1000} (X_i - \bar{X})(Y_i - \bar{Y}) \right) / \sum_{i=1}^{1000} (X_i - \bar{X})^2$$

Using the above values we can calculate the Asian option price, the sampling variance and the 95% confidence interval as follows: (the terminal output is slightly different from this one).

Option Price	Sampling Variance	95% Confidence Interval
18.490053260497668	34.28108643323387	[16.365289, 20.614817]

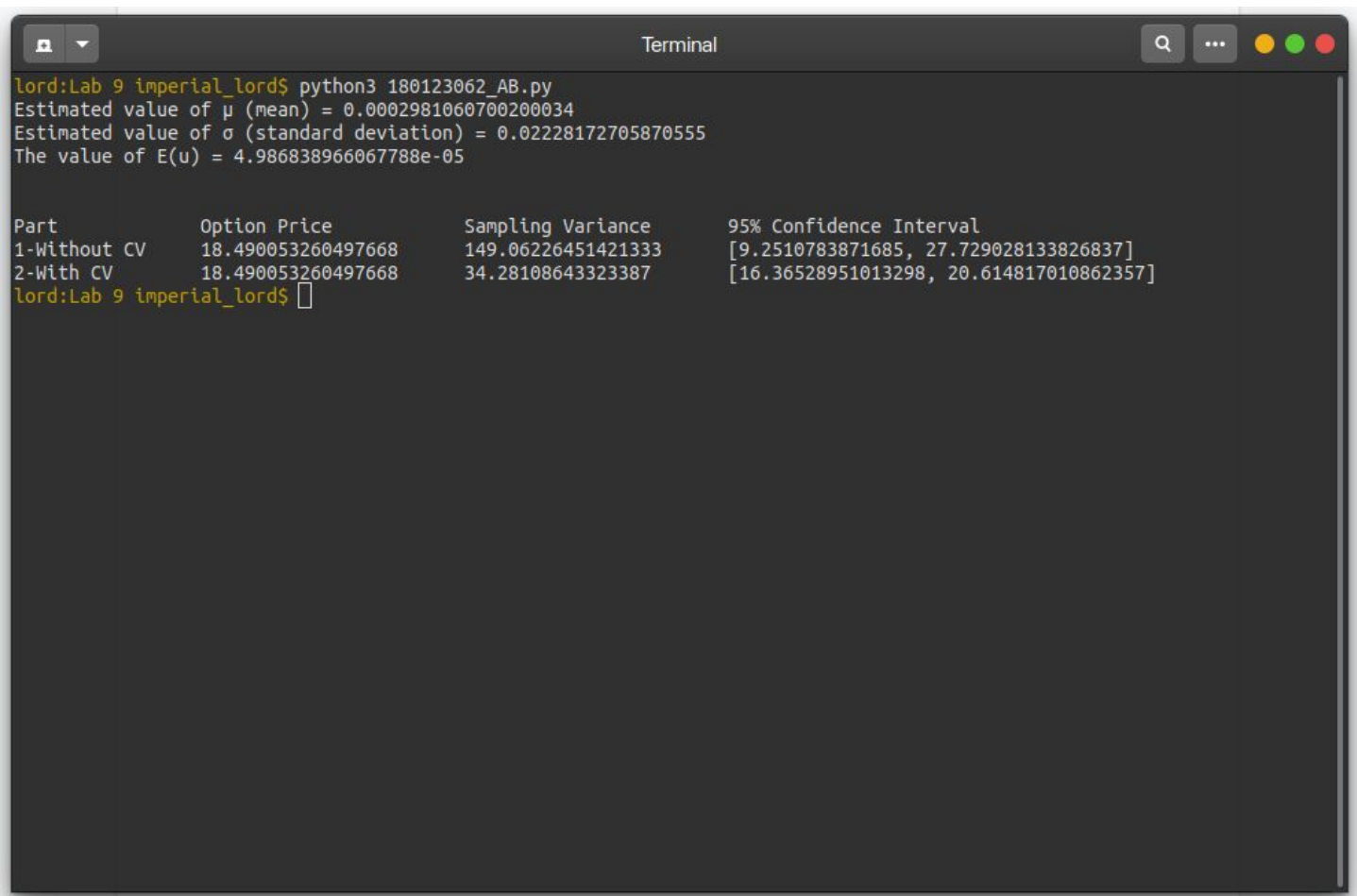
## Output:

Please make sure that the file '**SBIN.NS.csv**' is in the same directory as well for pandas to do its work.

Also, the following command will install **pandas** on the linux system. One can use pip if one doesn't use python3.

```
pip3 install pandas
```

The following output is obtained in the terminal after running the program **180123062\_AB.py**.

A terminal window titled "Terminal" with standard macOS window controls (search, close, zoom, and window management buttons). The terminal shows the execution of a Python script. The output includes statistical estimates for mean, standard deviation, and expected value, followed by a table comparing option prices and variances with and without correlation (CV).

```
lord:Lab 9 imperial_lord$ python3 180123062_AB.py
Estimated value of  $\mu$  (mean) = 0.0002981060700200034
Estimated value of  $\sigma$  (standard deviation) = 0.02228172705870555
The value of  $E(u)$  = 4.986838966067788e-05

Part          Option Price          Sampling Variance          95% Confidence Interval
1-Without CV   18.490053260497668         149.06226451421333        [9.2510783871685, 27.729028133826837]
2-With CV     18.490053260497668         34.28108643323387        [16.36528951013298, 20.614817010862357]
lord:Lab 9 imperial_lord$
```

Additionally one can note that the above values don't vary a lot on running the program multiple times, since we have taken it for **M=1000 simulations**, which in a sense averages out the different paths generated by the **Jump Diffusion Process**!