Numerical Project 1:
The Asian Option

Amie Fall, Dani González

March 6, 2024

Contents

1 Part 1

1.1 Introduction

The Asian option with arithmetic average (with strike K and maturity T) is a non-standard European derivative with pay-off $Y = (\int_0^T \frac{S(t)}{T} dt - K)_+$ for calls. It appeared in 1987 when two analysts, Mark Standish and David Spaughton, were in Tokyo on business and developed the first commercially used formula linked to the average price of crude oil. Compared with the standard European option that we have already seen, Asian options focus on paying the average value of the asset $S(t)$, which can be very useful in volatile markets or assets that can be manipulated: if you used options in a volatile asset, one could find that at the day of expiration the price $S(T)$ is quite different from $S(T-1)$, even if $S(T-1)$ was a better representation of the long term growth of the asset. Therefore, Asian options focus on capturing this average growth and providing a more representative payoff linked to the true value of the underlying. Intuitively, this feature makes Asian options cheaper, because in a Black Scholes market, European options price increase with volatility.

1.2 Exercise 6.18

In an Asian call with geometric average, the payoff at maturity is expressed as:

$$Z = (exp(\frac{1}{T}\int_0^T logS(t)dt) - K)_+$$

According to Exercise 6.18 we need to:

1) Find an exact formula for calls/puts in the Black-Scholes market
2) Derive call-put parity
3) Prove that the Asian call with geometric average is cheaper than the Asian call with arithmetic average.

1) Exact formula

Under the Black-Scholes assumptions we have:

$$S(t) = S(0)e^{(r-\frac{\sigma^2}{2})t+\sigma W(t)}$$

Where $W(t)$ is a Brownian Motion under the risk-neutral measure. And the risk-neutral price of a call is given by

$$AC(t) = e^{-r\tau}E[(Z-K)_+|F_{W(t)}]$$

Where $\tau = T - t$ and the expectation is taken with respect to the risk-neutral measure (we will only refer to risk-neutral measures) To come with a closed form solution we did some algebra with the initial payoff function Z and finally compared it with Theorem 6.6 of the book, yielding to a very similar solution.

$$\int_0^T logS(t)dt = \int_0^T (r - \frac{sigma^2}{2})t + \sigma W(t) = (r - \frac{\sigma^2}{2})T + \sigma \int_0^T W(t)$$

Where by use of stochastic calculus, the Ito integral follows a Normal Distribution:

$$\int_0^T W(t) \sim N(0, \frac{T^3}{3})$$

Therefore

$$\frac{1}{T}\int_0^T logS(t)dt) = S_0 e^{(r-\frac{\sigma^2}{2})+\sigma N(0,1)\sqrt{\frac{T^2}{3}}}$$

And

$$Z = (\frac{1}{T}\int_0^T logS(t)dt - K)_+ = (S_0 e^{(r-\frac{\sigma^2}{2})+\sigma N(0,1)\sqrt{\frac{T^2}{3}}} - K)_+$$

Computing the risk-neutral price

$$AC(t, S_0, r, \sigma, K, T) = e^{-r\tau}E[Z|F_{W(t)}] = e^{-r\tau}E[(S_0 e^{(r-\frac{\sigma^2}{2})+\sigma N(0,1)\sqrt{\frac{\tau^2}{3}}} - K)_+] =$$

$$= \frac{e^{-r\tau}}{\sqrt{2\pi}}\int_R (S_0 e^{(r-\frac{\sigma^2}{2})+\sigma y\sqrt{\frac{\tau^2}{3}}} - K)_+ e^{-\frac{y^2}{2}}dy$$

Which now is very similar to Theorem 6.6 of the book. Following the same steps (but with slightly different definitions for some terms) we arrived to a final closed-form solution similar to the Black-Scholes solution:

$$AC(t, S_0, r, \sigma, K, T) = S_0 e^{(a-r)\tau}\Phi(d_1) - Ke^{-r\tau}\Phi(d_2)$$

Where $a = \frac{r}{2} - \frac{\sigma}{12}$, $d1 = \frac{\sqrt{3}(log(\frac{S_0}{K})+(a+\frac{\sigma^2}{6})T}{\sigma\sqrt{T}}$, $d2 = d1 - \frac{\sigma\sqrt{T}}{\sqrt{3}}$

Doing the same in the case of puts, where the risk neutral price is given by

$$AP(t, S_0, r, \sigma, K, T) = \frac{e^{-r\tau}}{\sqrt{2\pi}}\int_R (K - S_0 e^{(r-\frac{\sigma^2}{2})+\sigma y\sqrt{\frac{\tau^2}{3}}})_+ e^{-\frac{y^2}{2}}dy$$

We get:

$$AP(t, S_0, r, \tau, K, T) = Ke^{-r\tau}\phi(-d_2) - S_0 e^{(a-r)T}\Phi(-d_1)$$

2- Call-put parity

Having computed the closed-form price for Calls and Puts we can check the call-put parity for Asian options (already proved in Exercise 6.17):

$$AC(t, S_0, r, \sigma, K, T) - AP(t, S_0, r, \sigma, K, T) = S_0 e^{(a-r)\tau}\Phi(d_1) - Ke^{-r\tau}\Phi(d_2) - Ke^{-r\tau}$$

$$\phi(-d_2) + S_0 e^{(a-r)\tau}\Phi(-d_1) = S_0 e^{(a-r)\tau}(\Phi(d_1) + \Phi(-d_1)) - Ke^{-r\tau}(\Phi(d_2) + \Phi(-d_2)) =$$

$$S_0 e^{(a-r)\tau} - Ke^{-r\tau}$$

3- Comparison to geometric average:

To prove that Asian calls with geometric average are cheaper than Asian calls

with arithmetic average is enough to prove that the payoff of the arithmetic average (AM) is greater than the geometric average (GM)

$$AM > GM -> ln(AM) > ln(GM)$$

As both substract K we can forget about it and only focus on the mean:

$$ln(AM) \propto ln(\frac{\int S_t dt}{T}) >^1 \frac{1}{T} ln(\Pi_0^T S_t dt) = \frac{1}{T} \int_0^T logS(t)dt \propto ln(GM)$$

*1: Ignoring the $1/T$ term, some properties allow to proof the remaining inequality.

1.3 Finite Difference Scheme: Crank-Nicolson Method

We are given the following PDE on the domain $(t, x) \in (0, T) \times (-Z, Z)$:

$$-\partial_t u(t, S(t)) + \frac{\sigma^2}{2}(\gamma(t) - z(t))^2 \partial_z^2 u(t, S(t))$$

$u(0, z) = z^+$, $limu(t, z) = 0$, $lim(u(t, z) - z) = 0$
$\gamma(t) = \frac{1-\exp(-rt)}{rT}$

We need to derive the two schemes seem in class:
Forward in time, centered in space:

$$\partial_t u(t, S(t)) = \frac{u(t + \Delta t, z) - u(t, z)}{\Delta t}$$

$$\partial_z^2 u(t, S(t)) = \frac{u(t, z + \Delta z) - 2u(t, z) + u(t, z - \Delta z)}{\Delta x^2}$$

$$\frac{u(t + \Delta t, z) - u(t, z)}{\Delta t} = \frac{\sigma^2}{2}(\gamma(t) - z(t))^2 \frac{u(t, z + \Delta z) - 2u(t, z) + u(t, z - \Delta z)}{\Delta x^2}$$

$$u(t + \Delta t, z) = u(t, z) + \frac{\sigma^2}{2}(\gamma(t) - z(t))^2 \Delta t \frac{u(t, z + \Delta z) - 2u(t, z) + u(t, z - \Delta z)}{\Delta x^2}$$

Where $d = \frac{\Delta t}{\Delta x^2}$ represents the discrete partition of the grid.
$\Delta t = \frac{n}{T}$, where n: number of steps and T: maturity time
$\Delta x = \frac{m}{2Z}$, where m: number of steps and Z: space partition (note that in this case we need to divide the space in the range [-Z, Z])
Finally we discretize the solution, which only depends on previous values

$$u_{i+1,j} = u_{i,j} + \frac{\sigma^2}{2}(\gamma_i - z_j)^2 (d)(u_{i,j+1} - 2u_{i,j} + u_{i,j-1})$$

i=0,..,n-1 and j=1,...,m-1

Backward in time, centered in space: Doing the same implementation for the backward method we got:

$$-\frac{u_{i,j}-u_{i,j-1}}{\delta t}+\frac{\sigma}{2}(\gamma_i-z_j)^2\frac{u_{i+1,j}-2u_{i,j}+u_{i-1,j}}{\delta z^2}$$

Which results in

$$u_{i,j-1}=u_{i,j}+\frac{\delta t,\sigma(\gamma_i-z_j)^2}{(}u_{i+1,j}-2u_{i,j}+u_{i-1,j})\frac{\delta_{z^2}}{delta_t}$$

i=0,..,n-1 and j=1,...,m-1

Crank-Nicolson Finally we take the mean of both methods, giving us:

$$-\frac{u_{i,j+1}-u_{i,j-1}}{2\delta t}+\frac{\sigma}{4}(\gamma_i-z_j)^2(\frac{u_{i+1,j}-2u_{i,j}+u_{i-1,j}}{\delta z^2}+\frac{u_{i+1,j+1}-2u_{i,j+1}+u_{i-1,j+1}}{\delta z^2})$$

Which solves in:

$$u_{i,j+1}=u_{i,j-1}+\frac{\delta t,\sigma(\gamma_i-z_j)^2}{(}\frac{u_{i+1,j}-2u_{i,j}+u_{i-1,j}}{\delta z^2}+\frac{u_{i+1,j+1}-2u_{i,j+1}+u_{i-1,j+1}}{\delta z^2})\frac{\delta_{z^2}}{2delta_t}$$

i=0,..,n-1 and j=1,...,m-1

Some comments on the intuition of boundary conditions:
- $u_{0,z}=z^+$: at maturity $z=max(z,0)$ as we did a change of variable t=T-t.
- $u_{t,-Z}=0$,$u_{t,Z}=z$: for any given time, the first and last solutions of the space are given by 0 and z

2 Part 2: implementation of the code

2.1 Python function

- Write a Matlab function that implements the finite difference scheme derived in Part 1. The parameters $S0,r,\sigma,K,T$ must appear as input variables of your function.

To implement this scheme in Python, we defined an empty grid A that represents the solutions to $u(t,z)$. In this case, it is defined in (n+1)x(m+1) as n steps mean that we will have n+1 values forming the grid. We defined:
$-n=1000(timesteps)$
$-m=1000(spacesteps)$
$-T=1(maturity)$
$-Z=500(spacedimension)$
.

And applied the boundary conditions, which are:
$-A[0][j] = max(space(j), 0)$
$- A[i][0] = 0$
$- A[i][m+1] = space(m+1) = +Z$

The next step was to solve the rest of the grid applying the scheme defined before:

$$A[i+1,j] = A[i,j] + \frac{\sigma^2}{2}(\gamma(time(i)) - space(j))^2(d)(A[i,j+1] - 2A[i,j] + A[i,j-1])$$

i=0,..,n-1 and j=1,...,m-1
(note that this is an implementation of the backward, but for the Crank-Nicolson we substituted the equation by the corresponding formula)
Finally we used Theorem 6.10 to price Asian calls with the given parameters as the risk-neutral expected payoff:

$-S_0 = 100$
$- Q_0 = 0$
$- r = 0.3$
$- \sigma = 0.1$
$- K = 105$
$- T = 1$
$- Z = 500$
.

$$AC(t, S_0, r, \sigma, K, T) = c(t, S(t), Q(t)) = S_0 u((T-t), \frac{1}{rT}(1 - e^{-rt}) + \frac{e^{-rt}}{S_0}(\frac{Q(0)}{T} - K)*$$

*note that we adapted the solution to the change of variable t=T-t

The code for that part was giving weird results, so we decided to implement and accrued Monte Carlo method (explained below) to get approximately 8.70

2.2 Asian call/put plots

- Plot the initial price of the Asian call/put as a function of the volatility σ and of the initial price $S0$. Show in the plot that the Asian call/put is always cheaper than the corresponding European option and less sensitive to volatility. Verify numerically the validity of the put-call parity.

In order to compute both call/puts prices, we decided to implement an accrued Monte Carlo method, in which we simulated the risk neutral payoff and took its discounted expected value.

1. Generate S(t) and Q(t)

$$dS_t = rS_t dt + \sigma S_t dW_t$$

$$dQ_t = Q(t)dt$$

Using the same partition of time defined before:

$$S_{t_{i+1}} = S_t + rS_{t_i}\Delta t + \sigma S_{t_i} dW_i$$

Where $dW_i \sim N(0, d_t)$

$$Q_{ti+1} = Q_t + S_t \Delta t$$

After simulating $Q(T)$ and $S(T)$, we simulated both Asian call and put payoffs and took the expected discounted value: we know that in the risk neutral world the assets grow at the risk free rate and under assumption the discount factor is just a constant that multiplies the expectation.

2. Then, we repeated it 5000 times and computed the mean, obtaining 8.65, which is very similar to the actual price.

3. Finally, we defined a function for computing the Black-Scholes price of European calls and puts:

$$c = S_0 N(d_1) - Ke^{-rT} N(d_2)$$

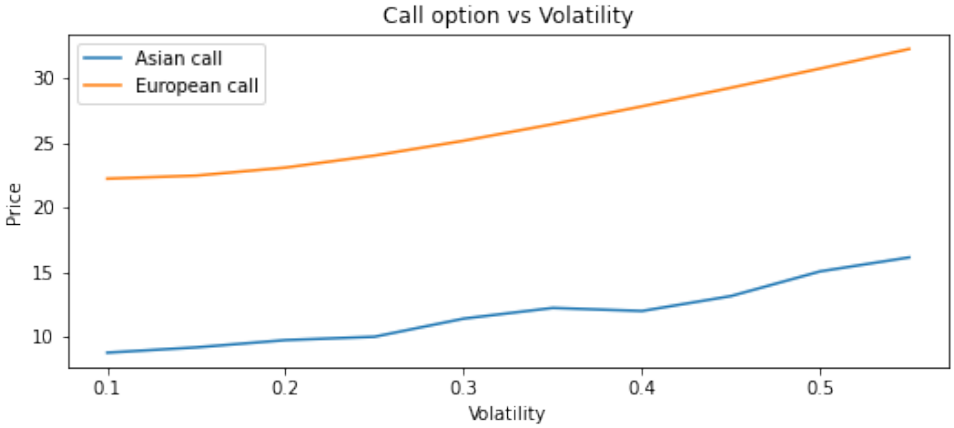
$$p = -Ke^{-rT} N(-d_2) - S_0 N(-d_1)$$

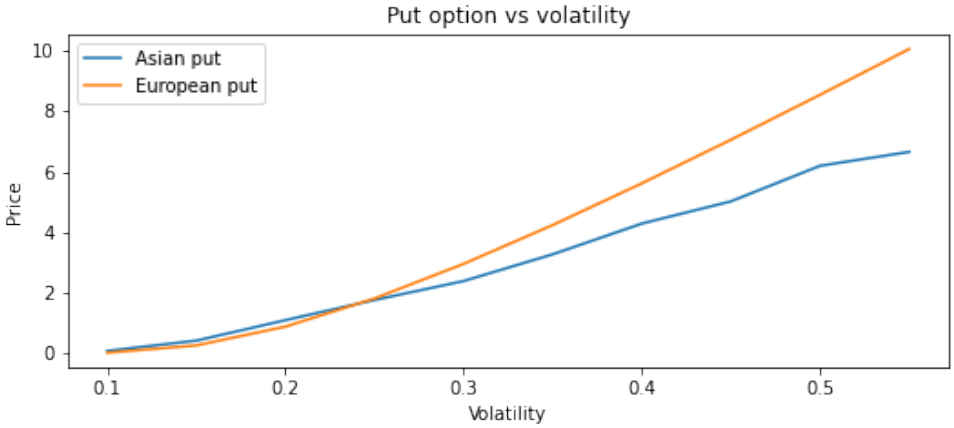
where $d1 = \frac{ln(\frac{S_t}{K}) + (r + \frac{\sigma^2}{2})t}{\sigma\sqrt{t}}$ and $d2 = d1 - \sigma\sqrt{t}$

4. Results:

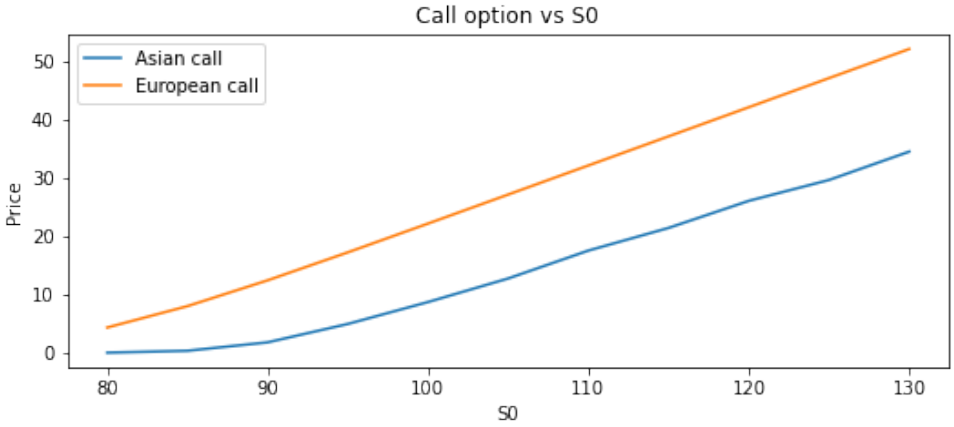
Volatility plots

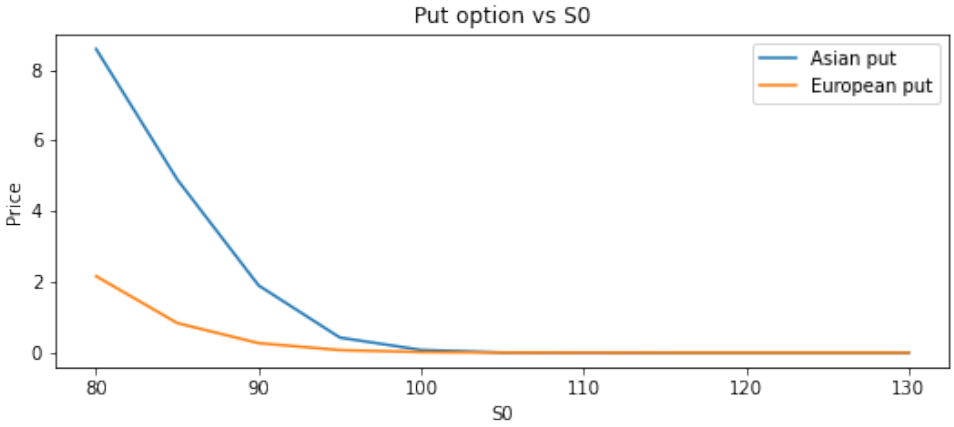
We first plotted the price of European/Asian options as a function of volatility. We observed that both calls and puts increase with volatility, being Asian options cheaper than the European ones.





Initial price plots In the case of initial stock price, calls increase with it (Asian calls being cheaper) while puts decrease (being European puts cheaper).





5. Put-call parity: We also used the formula given in Exercise 6.17, that corresponds to the put-call parity for Asian options to verify the parity numerically.
We generated some call/put prices with same parameters and compute the parity of each simulated pair. To verify the parity we computed the error between the simulated parity and the theoretical one. As each pair was an inexact simulation, each difference was not exactly 0, but taking the mean across all the comparisons we were very close to 0 (0.02), which verified it.

2.3 Control variate

- Compare the price obtained by the finite difference method and by the control variate Monte Carlo method for different value of σ and compare the efficiency of the two methods, e.g., by performing speed tests. Present your results using tables.

In order to implement the control variate chose to use the underlying asset S_T as the control variate, which is correlated to the payoff and has an expectation of $E[S_T] = S_0 e^{rT}$. We defined the new estimator as

$$\theta = Z + c(S_T - E[S_T])$$

where $c = -\frac{Cov(Y,Z)}{Var(Z)}$

One can check that $E[\theta] = E[Z]$ and $Var(\theta) <= Var(Y)$

In order to compare this 3 methods we saw that the Monte Carlo methods time was in the order of a few seconds, and it increased linearly with the number of simulation, which are required to get a better estimate. On the other hand, the finite difference method was in the order of tenths of second and it gave the same exact result every time, being much more reliable and accurate.To

summarize:

- Finite difference: Error comes from round-off error and discretization. In general it converges faster, it is deterministic, more accurate and stable. However, it requires more sophisticated methods, which are harder to implement.

- Monte Carlo: Error comes from the randomness of each simulation. It can implement path dependence and handle discontinuities, which makes it ideal for exotic options. The estimates are random, and implementing variance reduction techniques (like control variate) help to reduce the standard deviation of the price and get more accurate estimations in less time.

3 Appendix

References

https://en.wikipedia.org/wiki/Asian_option\protect\@normalcr\relax
https://core.ac.uk/reader/234678964

Code

```
import numpy as np
import matplotlib.pyplot as plt

#Parameters
T=1
S0=100
Q0=0
r=0.3
sigma=0.1
K=105

#1. Finite difference scheme
#Time partition [0,T]
n=1000 #steps
delta_t=T/n
time=np.zeros(n+1)
for i in range (1,n+1):
    time[i]=time[i-1]+delta_t

#Space partition [-Z, Z]
Z=100
m=1000
delta_z=(2*Z)/m
```

```
space=np.zeros(m+1)
space[0]=-Z
for j in range (1,m+1):
    space[j]=space[j-1]+delta_z

#uniform spatial partition
d=delta_t/(delta_z*delta_z)

#Boundary conditions
A=np.zeros((n+1,m+1))

#the first and last column are 0, so we dont have to do anything
#the first row u(0,z)=z (first and last value=0)
for j in range (0,m+1):
    var=space[j]
    if var<0:
        var=0
    A[0][j]=var

for i in range(0,n+1):
    A[i][1000]=space[1000]

gamma=(1-np.exp(-r*time))/(r*T)

for i in range(1,n+1): #start at 1 and finish at n
    gamma_t=gamma[i] #the gamma given at each time
    for j in range(1,m): #start at m=1 and finish at m-1 (0 and m = boundary conditions)
        a=((sigma*sigma)/2)*((gamma_t-space[j])**2)
        b=d
        c=A[i-1,j+1]-2*A[i-1,j]+A[i-1,j-1]
        A[i][j]=A[i-1,j] + (a*b*c)

t=0
t_change=T-t
index_i=int(t_change/delta_t)

index_j=(((1/(r*T))*(1-np.exp(-r*(T-t_change)))) +
         ((np.exp(-r*(T-t_change))/S0) * ((Q0/T)-K)))
index_j=int(index_j/delta_z)

call=S0*A[index_i,index_j]

#2.Plots
#function for computing Black Scholes price
from scipy import stats
def bs_price(T,S0,r,sigma,K):
```

```
    d1=np.log(S0/K) + (r+(sigma*sigma)/2)*T
    d1=d1/(sigma*np.sqrt(T))
    if (np.isinf(d1).any())==True:
        return -1
      #  return np.nan()

    d2=d1 - sigma*np.sqrt(T)

    vanilla_call=S0*stats.norm.cdf(d1)- K*np.exp(-r*T)*stats.norm.cdf(d2)
    vanilla_put=K*np.exp(-r*T)*stats.norm.cdf(-d2)-S0*stats.norm.cdf(-d1)
    return vanilla_call, vanilla_put

#Accrued Monte Carlo
def simulate(T,S0,r,sigma,K):
    n=1000 #steps
    delta_t=T/n
    time=np.zeros(n+1)
    S=np.zeros(n+1)
    Q=np.zeros(n+1)
    for i in range (1,n+1):
        time[i]=time[i-1]+delta_t
    S[0]=S0
    Q[0]=0

    #simulate S and Q
    for i in range (1,n+1):
        W=np.random.normal(0,np.sqrt(delta_t))
        dS=(r*S[i-1]*delta_t)+(sigma*S[i-1]*W)
        dQ=S[i-1]*delta_t
        S[i]=S[i-1]+dS
        Q[i]=Q[i-1]+dQ
    return Q[-1]

T=1
S0=100
Q0=0
r=0.3
sigma=0.1
K=105

simulations=10000

def price(simulations,T,S0,r,sigma,K):
    payoffs_call=np.zeros(simulations)
    payoffs_put=np.zeros(simulations)
```

```
    for i in range(simulations):
        Q=simulate(T,S0,r,sigma,K)
        payoffs_call[i]=np.exp(-r*(T)) * max((Q/T)-K,0)
        payoffs_put[i]=np.exp(-r*(T)) * max(K-(Q/T),0)

    return np.mean(payoffs_call),np.mean(payoffs_put)

call,put= price(simulations,T,S0,r,sigma,K)
print(call,put)

#Volatility Plots
T=1
S0=100
Q0=0
r=0.3
K=105

simulations=1000

volatility=np.arange(0.1,0.6,0.05)
asian_call=np.zeros(volatility.size)
asian_put=np.zeros(volatility.size)
european_call=np.zeros(volatility.size)
european_put=np.zeros(volatility.size)

i=0
for vol in volatility:
    asian_call[i],asian_put[i]=price(simulations,T,S0,r,vol,K)
    european_call[i], european_put[i]=bs_price(T,S0,r,vol,K)
    i+=1

#Calls
plt.rcParams["figure.figsize"] = [7.50, 3.50]
plt.rcParams["figure.autolayout"] = True
plt.title("Call option vs Volatility")
# plot lines
plt.plot(volatility, asian_call, label = "Asian call")
plt.plot(volatility, european_call, label = "European call")
plt.xlabel("Volatility")
plt.ylabel("Price")
plt.legend()
plt.savefig("Call-vol.png")
plt.show()


#Puts
```

```
plt.rcParams["figure.figsize"] = [7.50, 3.50]
plt.rcParams["figure.autolayout"] = True
plt.title("Put option vs volatility")
# plot lines
plt.plot(volatility, asian_put, label = "Asian put")
plt.plot(volatility, european_put, label = "European put")
plt.xlabel("Volatility")
plt.ylabel("Price")
plt.legend()
plt.savefig("Put-Vol.png")
plt.show()


T=1
Q0=0
r=0.3
K=105
sigma=0.1

simulations=1000

price_p=np.arange(80,135,5)
asian_call_p=np.zeros(price_p.size)
asian_put_p=np.zeros(price_p.size)
european_call_p=np.zeros(price_p.size)
european_put_p=np.zeros(price_p.size)

i=0
for p in price_p:
    asian_call_p[i], asian_put_p[i]=price(simulations,T,p,r,sigma,K)
    european_call_p[i], european_put_p[i]=bs_price(T,p,r,sigma,K)
    i+=1

#Calls
plt.rcParams["figure.figsize"] = [7.50, 3.50]
plt.rcParams["figure.autolayout"] = True
plt.title("Call option vs S0")
# plot lines
plt.plot(price_p, asian_call_p, label = "Asian call")
plt.plot(price_p, european_call_p, label = "European call")
plt.legend()
plt.xlabel("S0")
plt.ylabel("Price")
plt.savefig("Call-S0.png")
plt.show()
```

```
#Puts
plt.rcParams["figure.figsize"] = [7.50, 3.50]
plt.rcParams["figure.autolayout"] = True
plt.title("Put option vs S0")
# plot lines
plt.plot(price_p, asian_put_p, label = "Asian put")
plt.plot(price_p, european_put_p, label = "European put")
plt.legend()
plt.xlabel("S0")
plt.ylabel("Price")
plt.savefig("Put-S0.png")
plt.show()


#Call-put parity
T=1
S0=100
Q0=0
r=0.3
K=105

parity=0
for i in range(np.size(asian_call)):
    a=asian_call[i]-asian_put[i]
    b=(Q0/T)*np.exp(-r*T) + (S0/(r*T))*(1-np.exp(-r*T)) - K*np.exp(-r*T)
    parity=parity+(a-b)

print(parity/np.size(asian_call))

#Control Variate
def simulate_cv(T,S0,r,sigma,K):
    n=1000 #steps
    delta_t=T/n
    time=np.zeros(n+1)
    S=np.zeros(n+1)
    Q=np.zeros(n+1)
    for i in range (1,n+1):
        time[i]=time[i-1]+delta_t
    S[0]=S0
    Q[0]=0

    #simulate S and Q
    for i in range (1,n+1):
        W=np.random.normal(0,np.sqrt(delta_t))
        dS=(r*S[i-1]*delta_t)+(sigma*S[i-1]*W)
```

```
        dQ=S[i-1]*delta_t
        S[i]=S[i-1]+dS
        Q[i]=Q[i-1]+dQ
    return Q[-1],S[-1] #use S as control variate

simulations=500
payoffs=np.zeros(simulations)
control_variate_S=np.zeros(simulations)
for i in range(simulations):
    Q,S=simulate_cv(T,S0,r,sigma,K)

    payoffs[i]=np.exp(-r*(T)) * max((Q/T)-K,0)
    control_variate_S[i]=S

c=np.cov(payoffs,control_variate_S)[0][0]/np.var(control_variate_S)
exp_S=S0*np.exp(r*T)

control_estimator=payoffs+c*(control_variate_S-exp_S)


print(np.mean(control_estimator))
```