Computational Finance and Risk Management

Assignment 4

Asset Pricing

By

Shabari Girish Kodigepalli Venkata Subramanya

1005627644

Introduction

One of the drawbacks of the CAPM and APT and other models implemented earlier is that they are one-period model and cannot estimate the price of underlying asset beyond one period. However, pragmatically the asset prices are uncertain and fluctuate continuously. Thus, modeling of the asset price dynamics- mainly the drift and volatility is required. the price of option is modelling using the following pricing strategies:

BS_european_price: Black-Scholes pricing formula for European option.

Both 'Call and Put' prices for standard European options, which pays no dividends, are calculated as solutions of Black Scholes equation. The Black Sholes equation is a general form of Black–Scholes–Merton model.

MC_european_price: Monte Carlo pricing procedure for European option.

Both 'Call and Put' prices for standard European options are modeled by Monte Carlo simulations using Geometric Brownian Motion (Geometric Random Walk equation) with constant drift and volatility as mentioned in the assignment.

MC_barrier_knockin_price: Monte Carlo pricing procedure for Barrier knock-in option.

Both 'Call and Put' prices for standard Barrier options are also modeled by Monte Carlo simulation using Geometric Brownian Motion but with an added condition that the price should cross the barrier specified in the assignment. Only if the price crosses the barrier at least once before expiry the option behaves like a European option otherwise it expires worthless.

Implementing Black-Scholes pricing formula

The function BS_european_price which takes the arguments- Initial Price, Strike Price, Maturity Time, Risk-free rate and volatility was developed. The function uses the Black-Scholes options pricing formula derived by assuming that the underlying asset follows the Geometric Brownian Motion and further solving the Black–Scholes partial differential equation:

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0$$

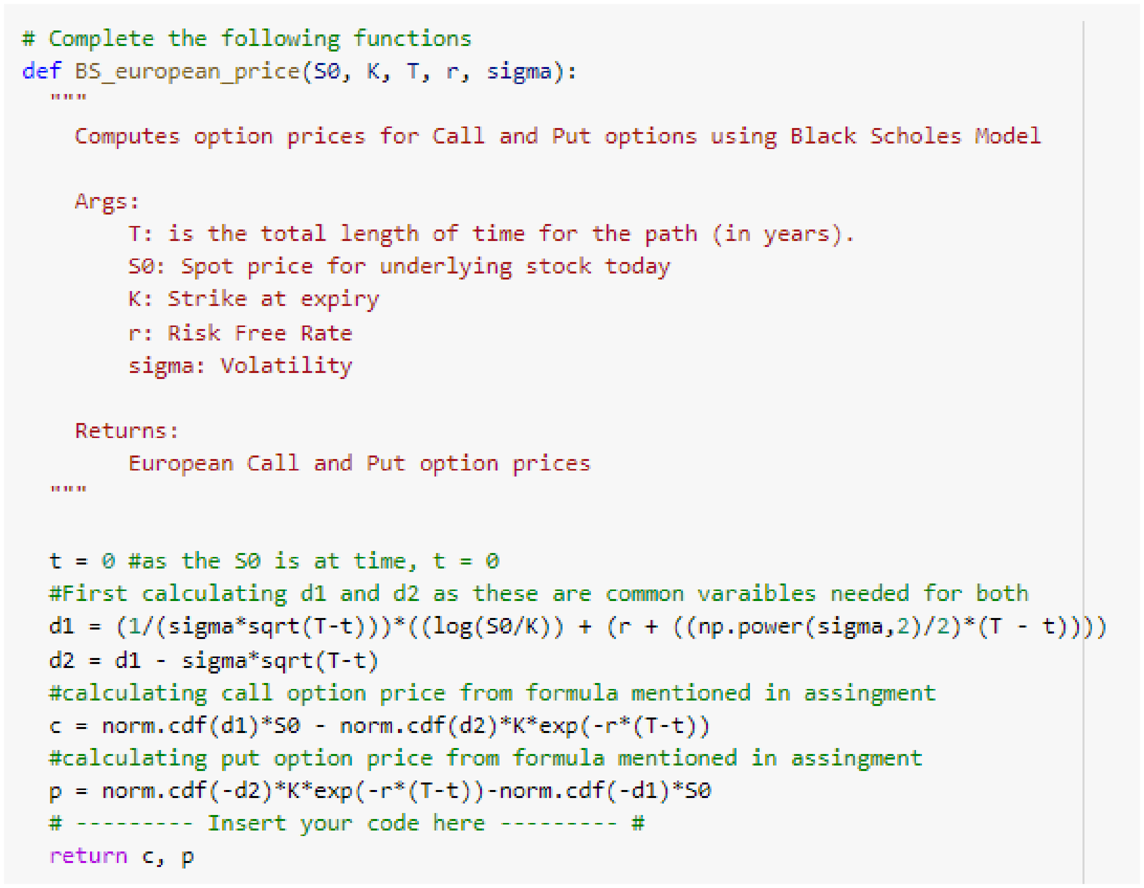
The value of a European call option for a non-dividend-paying underlying stock in terms of the Black–Scholes is as follows:

$$C(S,t) = \mathcal{N}(d_1)S - \mathcal{N}(d_2)Ke^{-r(T-t)}$$
$$d_1 = \frac{1}{\sigma\sqrt{T-t}}\left[\ln\left(\frac{S}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)(T-t)\right]$$
$$d_2 = d_1 - \sigma\sqrt{T-t}$$

The price of a corresponding European put option based on put-call parity is given as:

$$\begin{aligned} P(S,t) &= Ke^{-r(T-t)} - S + C(S,t) \\ &= \mathcal{N}(-d_2)Ke^{-r(T-t)} - \mathcal{N}(-d_1)S \end{aligned}$$

Where N(.) is cumulative distribution function of standard Normal distribution



```
# Complete the following functions
def BS_european_price(S0, K, T, r, sigma):
  """
    Computes option prices for Call and Put options using Black Scholes Model

    Args:
        T: is the total length of time for the path (in years).
        S0: Spot price for underlying stock today
        K: Strike at expiry
        r: Risk Free Rate
        sigma: Volatility

    Returns:
        European Call and Put option prices
  """

  t = 0 #as the S0 is at time, t = 0
  #First calculating d1 and d2 as these are common varaibles needed for both
  d1 = (1/(sigma*sqrt(T-t)))*((log(S0/K)) + (r + ((np.power(sigma,2)/2)*(T - t))))
  d2 = d1 - sigma*sqrt(T-t)
  #calculating call option price from formula mentioned in assingment
  c = norm.cdf(d1)*S0 - norm.cdf(d2)*K*exp(-r*(T-t))
  #calculating put option price from formula mentioned in assingment
  p = norm.cdf(-d2)*K*exp(-r*(T-t))-norm.cdf(-d1)*S0
  # --------- Insert your code here --------- #
  return c, p
```

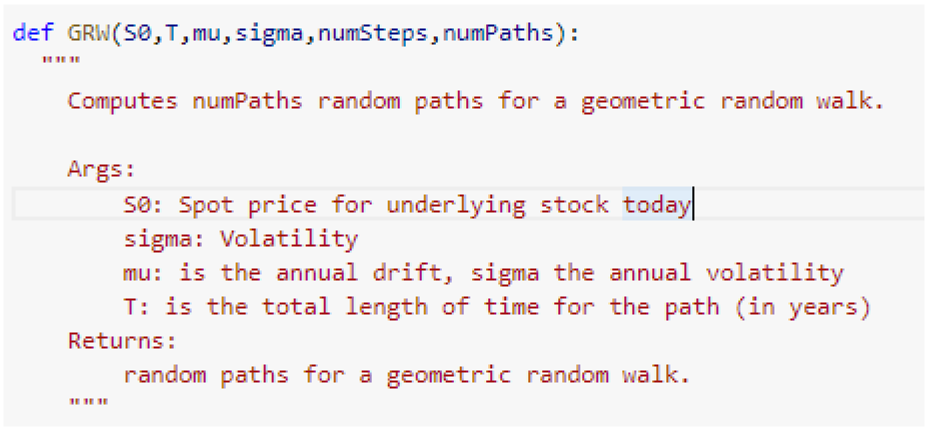
Implementing one-step Monte Carlo (MC) pricing procedure for European option

Stochastic process can be discretized into finite time steps and then Monte Carlo simulation method can be used to the generate number of scenarios and draw the results from the same. In one-step Monte Carlo pricing method, thus, the time increment dT is taken equal to the maturity time T which in this case is one year. The given mu and sigma were assumed to be annual drift and volatility respectively.

The function MC_european_price which takes the arguments- Initial Price, Strike Price, Maturity Time, Risk-free rate, mean, volatility, number of steps and the number of paths was developed. The function uses Geometric Random Walk Model to get the price of the underlying asset in future and further obtain the price of the call and put option based on the strike price.

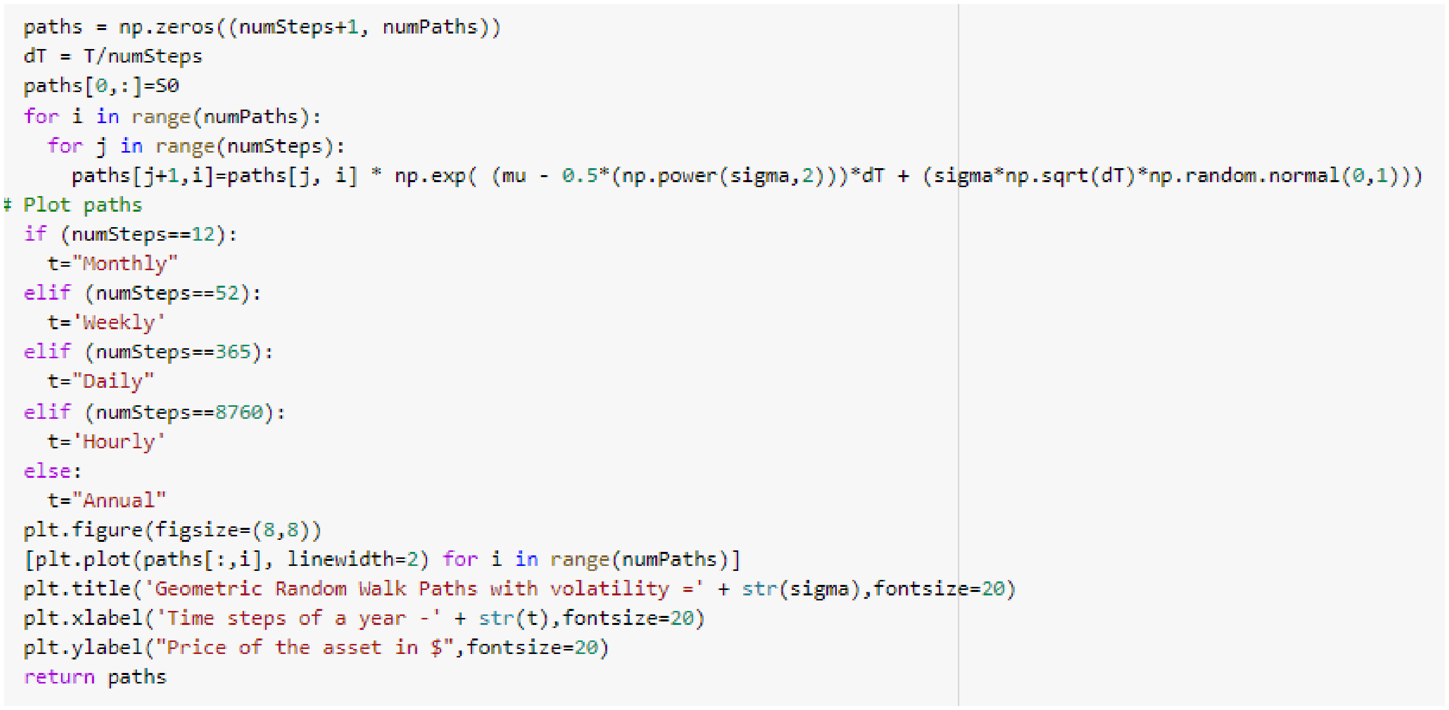
The steps involved in computing the one step MC pricing of European options are as follows:

1. Call the *MC_european_price*
2. Declare a variable to identify if the pricing in with or without Barrier
3. Call the *GRWPaths* function with the arguments as *Initial Price, mu, sigma, Time to maturity, number of paths or scenarios* and the *pricing identity i.e. with or without barrier.*
4. In the *GRWPaths* function apart from calculating the price of the underlying asset using GRW model, number of steps and the pricing identifier (shown in step 3) was used set appropriate title to the plot of the GRW paths of the asset price as shown below:



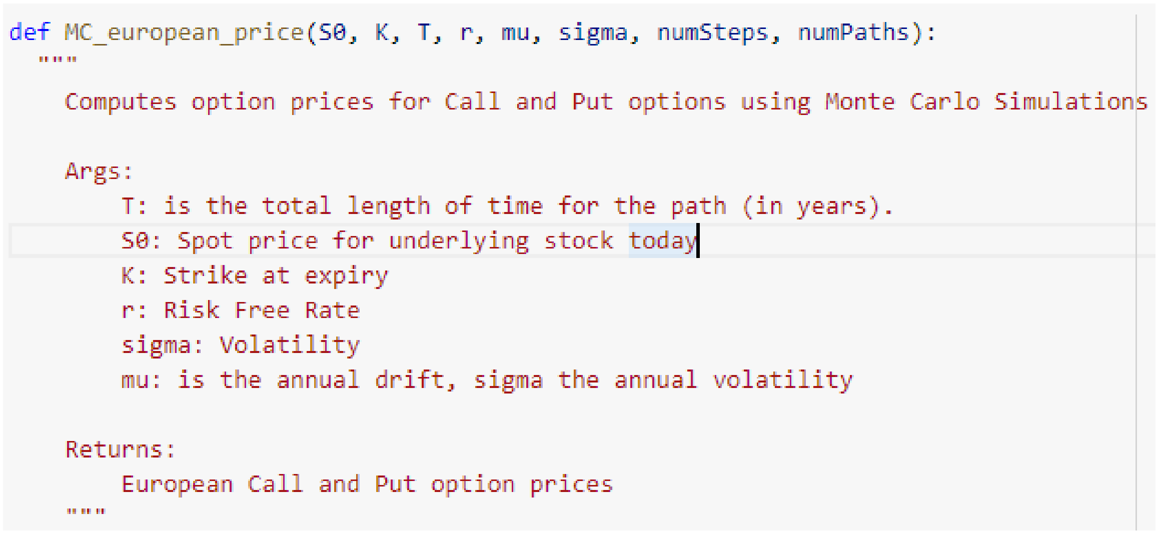
```python
def GRW(S0,T,mu,sigma,numSteps,numPaths):
  """
    Computes numPaths random paths for a geometric random walk.

    Args:
        S0: Spot price for underlying stock today
        sigma: Volatility
        mu: is the annual drift, sigma the annual volatility
        T: is the total length of time for the path (in years)
    Returns:
        random paths for a geometric random walk.
    """
```



```python
  paths = np.zeros((numSteps+1, numPaths))
  dT = T/numSteps
  paths[0,:]=S0
  for i in range(numPaths):
    for j in range(numSteps):
      paths[j+1,i]=paths[j, i] * np.exp( (mu - 0.5*(np.power(sigma,2)))*dT + (sigma*np.sqrt(dT)*np.random.normal(0,1)))
# Plot paths
  if (numSteps==12):
    t="Monthly"
  elif (numSteps==52):
    t='Weekly'
  elif (numSteps==365):
    t="Daily"
  elif (numSteps==8760):
    t='Hourly'
  else:
    t="Annual"
  plt.figure(figsize=(8,8))
  [plt.plot(paths[:,i], linewidth=2) for i in range(numPaths)]
  plt.title('Geometric Random Walk Paths with volatility =' + str(sigma),fontsize=20)
  plt.xlabel('Time steps of a year -' + str(t),fontsize=20)
  plt.ylabel("Price of the asset in $",fontsize=20)
  return paths
```

5. Since it's the European option, the payoff is calculated from the last row (number of steps+1) of the matrix shown in step 5. The payoff in the call and put option is obtained in the MC_european_price function which is then discounted to the current time as shown below:



```python
def MC_european_price(S0, K, T, r, mu, sigma, numSteps, numPaths):
  """
    Computes option prices for Call and Put options using Monte Carlo Simulations

    Args:
        T: is the total length of time for the path (in years).
        S0: Spot price for underlying stock today
        K: Strike at expiry
        r: Risk Free Rate
        sigma: Volatility
        mu: is the annual drift, sigma the annual volatility

    Returns:
        European Call and Put option prices
    """
```

```
S=GRW(S0,T,mu,sigma,numSteps,numPaths)
PutPayoffT = np.maximum(K - S[-1,:], 0)
#Calculate the payoff for each path for a Call
CallPayoffT = np.maximum(S[-1,:]- K, 0)
#Discount back
p= np.mean(PutPayoffT)*exp(-r*T)
c= np.mean(CallPayoffT)*exp(-r*T)

return c, p
```

Implementing multi-step Monte Carlo pricing procedure for European option

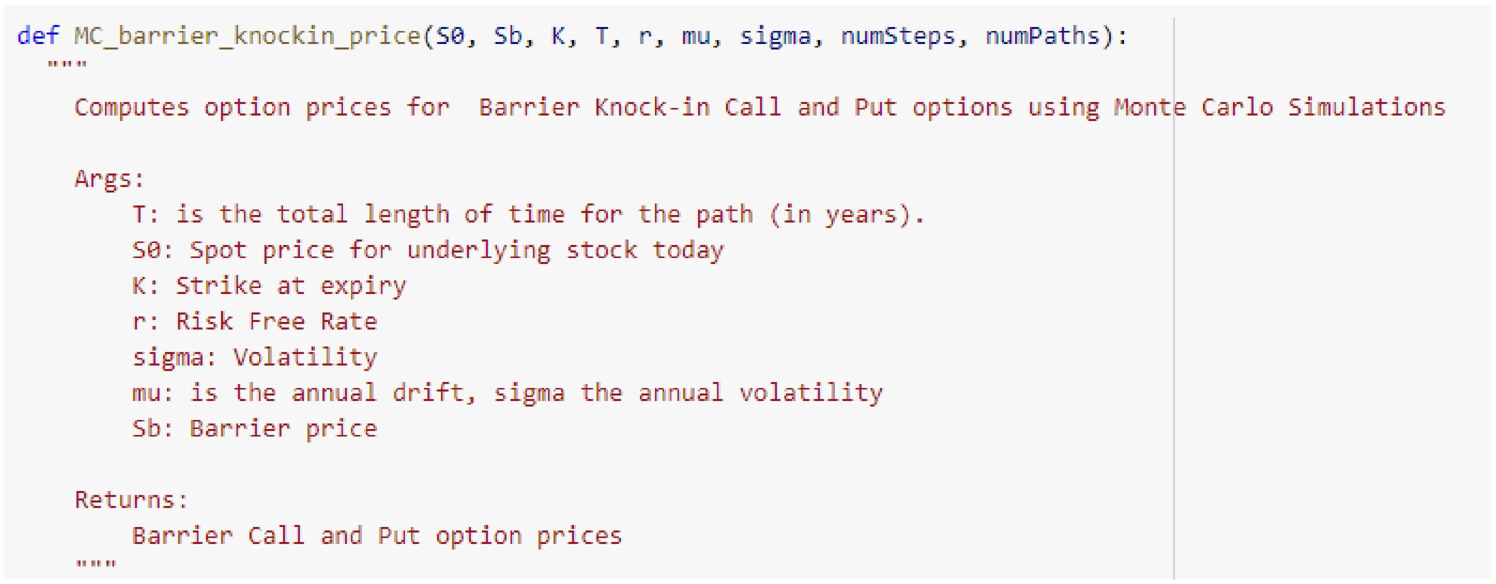
The price of the underlying asset can be computed every week, every month, every day of the year until the maturity, in other words the time interval (dt) is chopped into as many units (number of steps) as we need. Since calculation is performed multiple times throughout the life of an option, this is referred as multiple-step MC. To check the accuracy of the results and in-depth analysis here the price is computed on monthly, weekly and daily basis i.e. number of steps will be 12, 52 and 365 respectively.

```
# # Implement your multi-step Monte Carlo pricing procedure for European option
callMC_European_Price_multi_step=np.zeros((len(numSteps)))
putMC_European_Price_multi_step=np.zeros((len(numSteps)))
for i in range(len(numSteps)):
  callMC_European_Price_multi_step[i], putMC_European_Price_multi_step[i] = MC_european_price (S0, K, T, r, mu, sigma, numSteps[i], numPaths)
```

Implementing one-step Monte Carlo pricing procedure for Barrier option

A barrier option is a type of option whose payoff depends on whether the underlying asset has reached or exceeded a predetermined price – barrier. There are two general types of barrier options, in and out options. A knock-out option only pays so if the stock does not hit the barrier level throughout the life of the option. If the stock hits a specified barrier, then it has knocked out and expires worthless. A knock-in option on the other hand only pays out if the barrier is crossed during the life of the option.

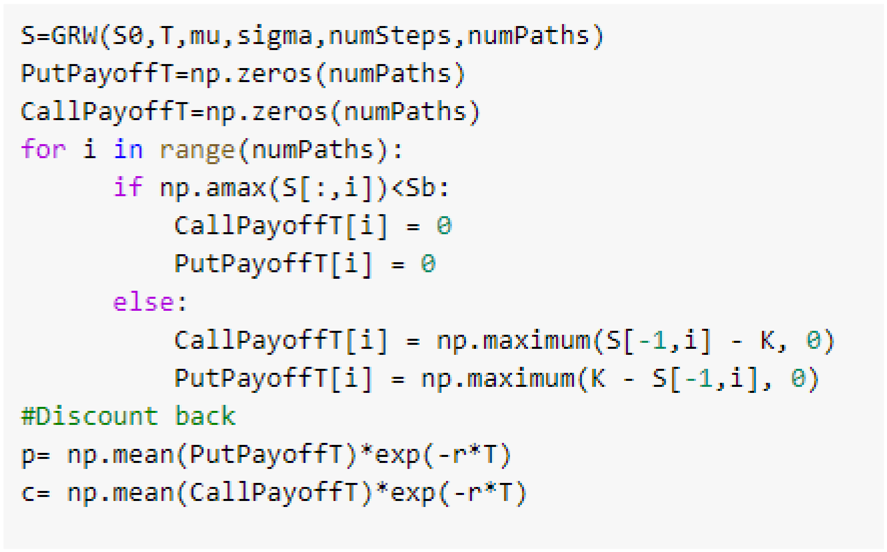
If the barrier price is above the spot price, then the option is an up option; if the barrier is below the spot price then it is a down option. Therefore, we can categorize barrier options (put/call) as follows: down-and-out, down-and-in, up-and-out and up-and-in. Monte Carlo simulations can be used to price barrier options in a similar manner as European options. Once multiple asset paths for GBM have been simulated, the next step is to determine the payoff for each asset path. This is done by evaluating each path to see whether it has hit the barrier. The payoff is then dependent on the type of barrier option and the knowledge of whether the barrier level has been hit during the life of the option. Some options will expire worthless if the barrier is reached, others will be worthless unless the barrier is reached.



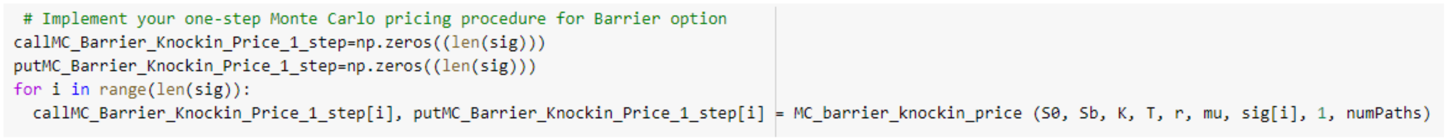
```
def MC_barrier_knockin_price(S0, Sb, K, T, r, mu, sigma, numSteps, numPaths):
  """
    Computes option prices for  Barrier Knock-in Call and Put options using Monte Carlo Simulations

    Args:
        T: is the total length of time for the path (in years).
        S0: Spot price for underlying stock today
        K: Strike at expiry
        r: Risk Free Rate
        sigma: Volatility
        mu: is the annual drift, sigma the annual volatility
        Sb: Barrier price

    Returns:
        Barrier Call and Put option prices
    """
```



```
S=GRW(S0,T,mu,sigma,numSteps,numPaths)
PutPayoffT=np.zeros(numPaths)
CallPayoffT=np.zeros(numPaths)
for i in range(numPaths):
      if np.amax(S[:,i])<Sb:
          CallPayoffT[i] = 0
          PutPayoffT[i] = 0
      else:
          CallPayoffT[i] = np.maximum(S[-1,i] - K, 0)
          PutPayoffT[i] = np.maximum(K - S[-1,i], 0)
#Discount back
p= np.mean(PutPayoffT)*exp(-r*T)
c= np.mean(CallPayoffT)*exp(-r*T)
```

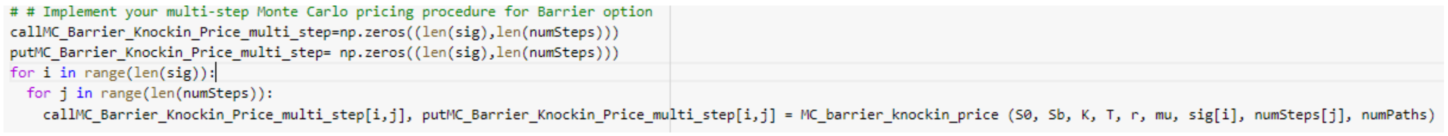


```
 # Implement your one-step Monte Carlo pricing procedure for Barrier option
callMC_Barrier_Knockin_Price_1_step=np.zeros((len(sig)))
putMC_Barrier_Knockin_Price_1_step=np.zeros((len(sig)))
for i in range(len(sig)):
  callMC_Barrier_Knockin_Price_1_step[i], putMC_Barrier_Knockin_Price_1_step[i] = MC_barrier_knockin_price (S0, Sb, K, T, r, mu, sig[i], 1, numPaths)
```

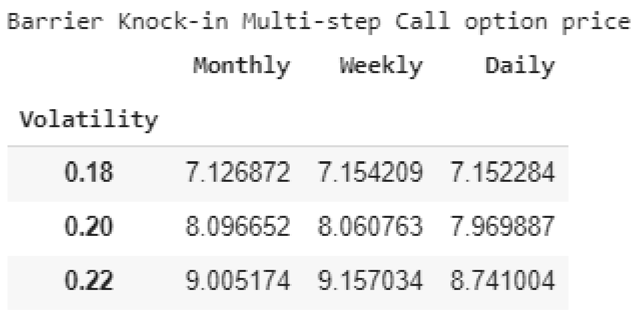
As seen from above, here new function MC_barrier_knockin_price was developed. This function remained the same as MC_european_price function, thus only the changes made to this function in development of the MC_barrier_knockin_price is detailed here. The strike price is 105 and the barrier price is 110, thus for the call option, the call payoff will be zero until the barrier level is reached. Once, the barrier is crossed, the payoff will be (Asset price at maturity- Strike price). For the put payoff, since the barrier price is greater than the strike price, once the asset price at the maturity goes below the strike price, the payoff starts with (Strike price- Asset price).

Implementing multi-step Monte Carlo pricing procedure for Barrier option

The implementation of the multi-step MC pricing for the barrier option is same as that of multi-step option pricing detailed for European option having multiple steps per volatility value. The only difference being that here the price of call and put options are computed for sigma of 0.2 as well as with a 10% increase and decrease in the sigma. Accordingly, an array of sigma is defined as sig= [0.18 0.2 0.22]. The dimensionality of the resulting output changes from those in previous section in the sense that instead of a single row of a matrix of a cell, here we have three row each for a given sigma. Also, a ‘for’ loop is added to loop through each value of sigma as shown below:

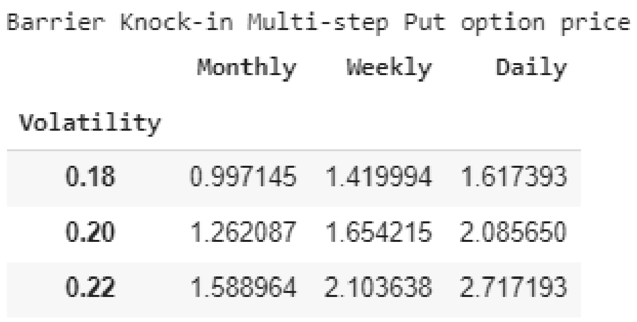


```
# # Implement your multi-step Monte Carlo pricing procedure for Barrier option
callMC_Barrier_Knockin_Price_multi_step=np.zeros((len(sig),len(numSteps)))
putMC_Barrier_Knockin_Price_multi_step= np.zeros((len(sig),len(numSteps)))
for i in range(len(sig)):
  for j in range(len(numSteps)):
    callMC_Barrier_Knockin_Price_multi_step[i,j], putMC_Barrier_Knockin_Price_multi_step[i,j] = MC_barrier_knockin_price (S0, Sb, K, T, r, mu, sig[i], numSteps[j], numPaths)
```



Barrier Knock-in Multi-step Call option price

Volatility	Monthly	Weekly	Daily
0.18	7.126872	7.154209	7.152284
0.20	8.096652	8.060763	7.969887
0.22	9.005174	9.157034	8.741004



Barrier Knock-in Multi-step Put option price

Volatility	Monthly	Weekly	Daily
0.18	0.997145	1.419994	1.617393
0.20	1.262087	1.654215	2.085650
0.22	1.588964	2.103638	2.717193

Analysis of Results

The results of the implementation of the code explained above is analyzed here:

Black-Scholes call and put price for the given European option

The function BS_european_price detailed in section 2 which takes the arguments- Initial Price, Strike Price, Maturity Time, Risk-free rate and volatility was used to compute the price of European option using the Black-Scholes formula. The function uses the Black-Scholes options pricing formula derived by assuming that the underlying asset follows the Geometric Brownian Motion and further solving the Black–Scholes partial differential equation.

```
Black-Scholes price of an European call option is 8.021352235143176
Black-Scholes price of an European put option is 7.9004418077181455
```

One-step MC call and put price for the given European option

In one-step Monte Carlo pricing method, the time increment dT is taken equal to the maturity time T which in this case is one year. The function MC_european_price as detailed in section 3 which takes the arguments- Initial Price, Strike Price, Maturity Time, Risk-free rate, mean, volatility, number of steps and the number of paths was used to obtain the required price. The function uses Geometric Random Walk Model to get the price of the underlying asset in future and further obtain the price of the call and put option based on the strike price.

The number of paths for the one and Multi-step MC price was kept as 10000 to make the results comparable. However, since the one-step MC calculates the price of the option directly at the maturity period, in this case, one year, it is advisable to keep the number of paths of the one step MC higher than the multi-step to make for the discretization error due to higher time step and also lead to a reduction in the variability of the results with the increase number of samples.

```
One-step MC price of an European call option is 8.068838813284108
One-step MC price of an European put option is 7.871860075539126
```

Multi-step MC call and put price for the given European option.:

The price of the underlying asset can be computed every week, every month, every day of the year until the maturity, in other words the time interval (dt) is chopped into as many units (number of steps) as we need. Since calculation is performed multiple times throughout the life of an option, this is referred as multiple-step MC. To check the accuracy of the results and in-depth analysis here the price was computed on monthly, weekly and daily basis i.e. number of steps used were 12, 52 and 365 respectively.

When the pricing is done by simulation (discretization of the stochastic process), discretization error occurs as we cannot simulate future prices at infinitely small-time intervals. This discretization error can be reduced and the distribution of the price after many steps will indeed approximate a lognormal distribution with the correct parameters (but it is computationally intensive). Thus, in multi-step MC pricing we reduce the error by generating the whole path even if we are only interested in the end of time horizon. Accordingly, as explained in previous section, the number of paths here can be taken less than that of the one-step MC pricing, however, to keep the pricing of one-step and multi-step MC pricing comparable the number of paths of 10000 was used for both single and multi-step MC pricing.

Computing multi-step MC price of European option:

```
Multi-step Monthly  MC price of an European call option is 8.205955348950111
Multi-step Monthly  MC price of an European put option is 7.884391753872634
Multi-step Weekly  MC price of an European call option is 8.01188219455434
Multi-step Weekly  MC price of an European put option is 7.885089774834908
Multi-step Daily  MC price of an European call option is 8.055057810889023
Multi-step Daily  MC price of an European put option is 7.840025524401902
```

One-step MC call and put price for the given Barrier option

The objective of the task here was to obtain the price of knock-in option. The function MC_barrier_knockin_price as detailed in section 5 was used to compute the price of the Barrier options. The strike price was 105 and the barrier price was 110, thus for the call option, the call payoff will be zero until the barrier level is reached. Once, the barrier is crossed, the payoff will be (Asset price at maturity- Strike price). For the put payoff, since the barrier price is greater than the strike price, once the asset price at the maturity goes below the strike price, the payoff starts with (Strike price- Asset price).

Thus, here, the put payoff will behave in similar way as put European option.

It is advisable to keep the number of paths in the barrier option to be greater than the European options because this ensures that there in sufficient number of paths in the MC simulation which cross the barrier and the ones that do not to clear identify the difference in the behavior of the barrier option from the European option. Accordingly, the number of paths in this case was kept as 10000. The results of the implementation of this function with one step is as follows:

Computing One-step MC prices for given sigma values:

```
One-step MC price of an Barrier call option (volatility = 0.18 ) is 6.944149735261419
One-step MC price of an Barrier put option (volatility = 0.18 ) is 0.0
One-step MC price of an Barrier call option (volatility = 0.2 ) is 7.90748293646925
One-step MC price of an Barrier put option (volatility = 0.2 ) is 0.0
One-step MC price of an Barrier call option (volatility = 0.22 ) is 8.472923985824547
One-step MC price of an Barrier put option (volatility = 0.22 ) is 0.0
```

Multi-step MC call and put price for the given Barrier option

The motivation behind using multi-step is as detailed in section 7.3 i.e. to reduce the discretization error. Accordingly, as explained in section 7.2, the number of paths here can be taken less than that of the one-step MC barrier pricing, however, to keep the pricing of one-step and multi-step MC barrier pricing comparable the number of paths of 10000 was used for both single and multi-step MC barrier pricing. To check the accuracy of the results and in-depth analysis here the price was computed on monthly, weekly and daily basis i.e. number of steps used were 12, 52 and 365 respectively.

The results of the implementation of this function with multiple steps is as follows:

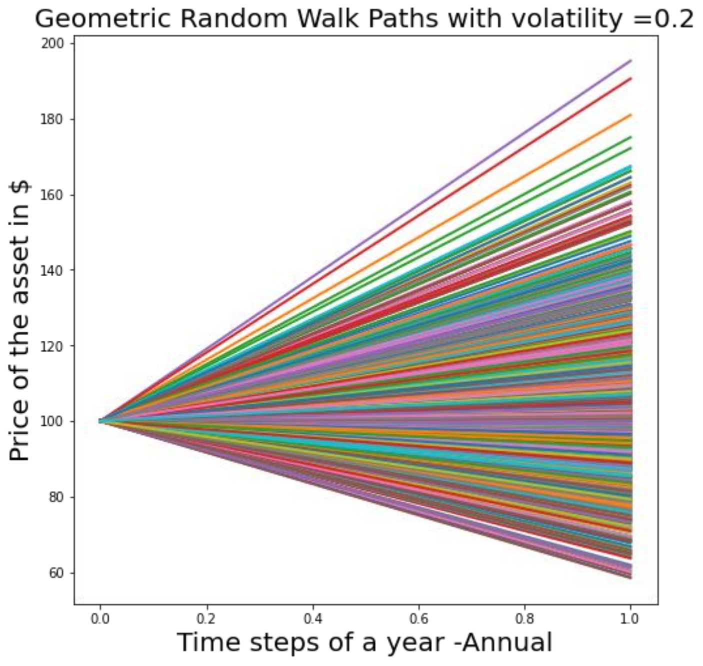
Computing multi-step MC price of Barrier option with different volatility values:

```
Multi-step Monthly  MC price of an Barrier call option (volatility= 0.18 ) is 7.222074564455849
Multi-step Monthly  MC price of an Barrier put option (volatility= 0.18 ) is 1.0458917318627028
Multi-step Weekly  MC price of an Barrier call option (volatility= 0.18 ) is 6.904798437312771
Multi-step Weekly  MC price of an Barrier put option (volatility= 0.18 ) is 1.5138029815204765
Multi-step Daily  MC price of an Barrier call option (volatility= 0.18 ) is 7.192576620822846
Multi-step Daily  MC price of an Barrier put option (volatility= 0.18 ) is 1.9250631076850915
Multi-step Monthly  MC price of an Barrier call option (volatility= 0.2 ) is 7.609140037380542
Multi-step Monthly  MC price of an Barrier put option (volatility= 0.2 ) is 1.2175203962343428
Multi-step Weekly  MC price of an Barrier call option (volatility= 0.2 ) is 7.569661703170689
Multi-step Weekly  MC price of an Barrier put option (volatility= 0.2 ) is 1.8160273148056443
Multi-step Daily  MC price of an Barrier call option (volatility= 0.2 ) is 7.287504868557493
Multi-step Daily  MC price of an Barrier put option (volatility= 0.2 ) is 1.8462128695596722
Multi-step Monthly  MC price of an Barrier call option (volatility= 0.22 ) is 8.8994542825267
Multi-step Monthly  MC price of an Barrier put option (volatility= 0.22 ) is 1.780855771539562
Multi-step Weekly  MC price of an Barrier call option (volatility= 0.22 ) is 8.459224302869556
Multi-step Weekly  MC price of an Barrier put option (volatility= 0.22 ) is 1.9556974052245653
Multi-step Daily  MC price of an Barrier call option (volatility= 0.22 ) is 8.742600384227577
Multi-step Daily  MC price of an Barrier put option (volatility= 0.22 ) is 2.7967771372611963
```

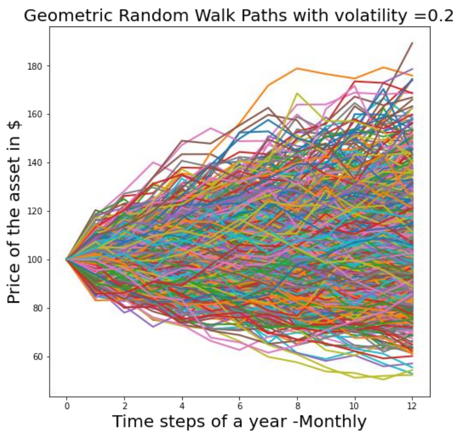
Plot

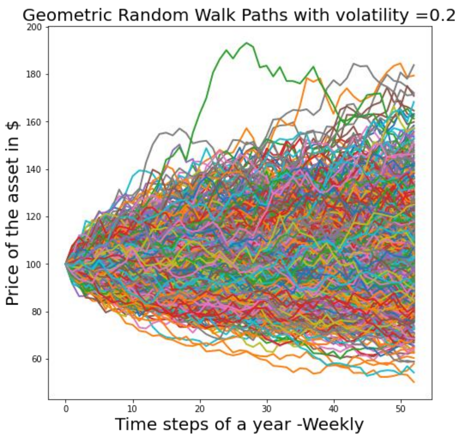
The plot obtained for the pricing strategies-

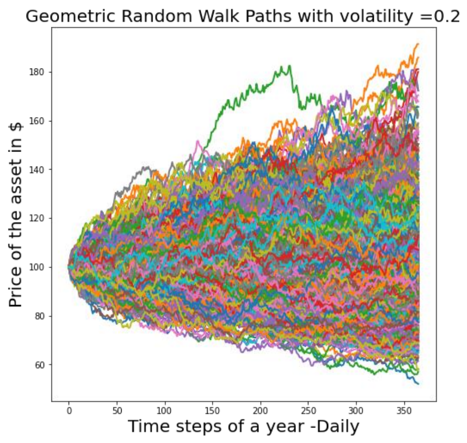
MC European Pricing:



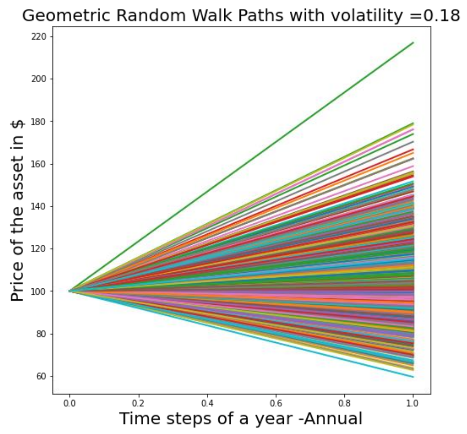
Multi-Step MC European option:

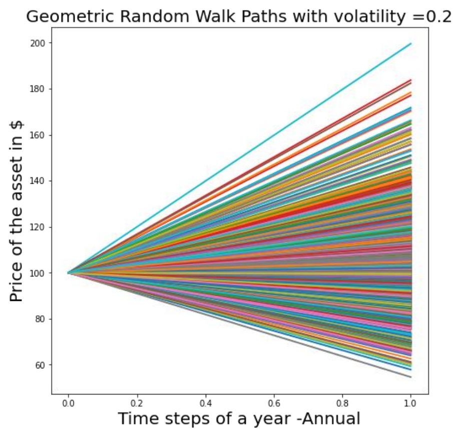


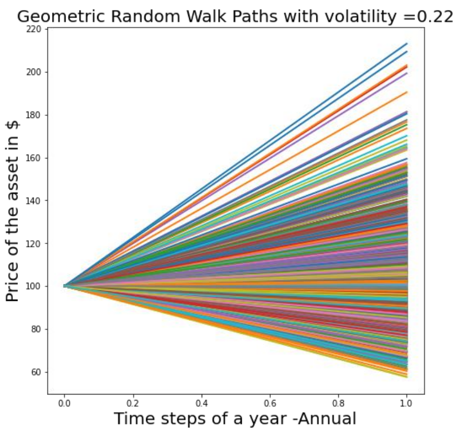




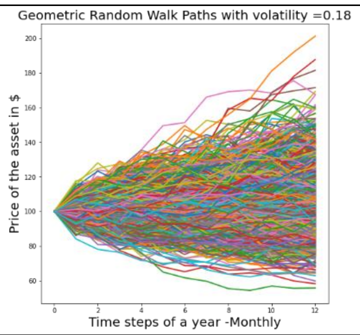
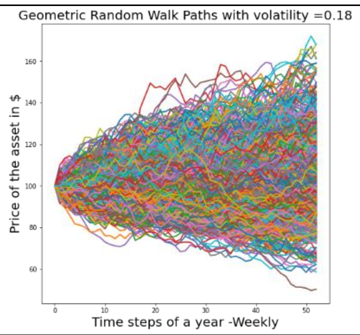
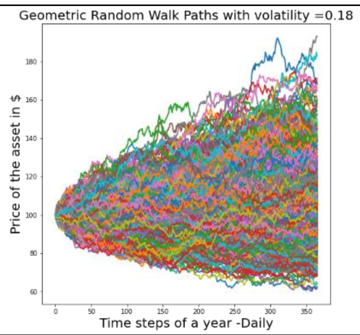
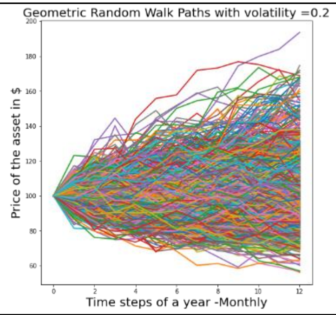
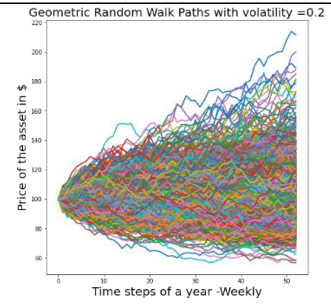
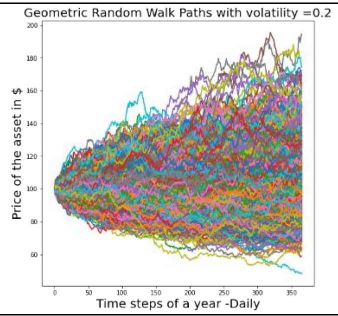
One-Step MC Barrier option (different volatilities):

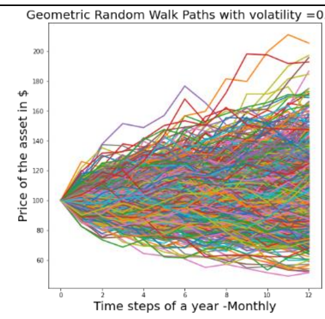
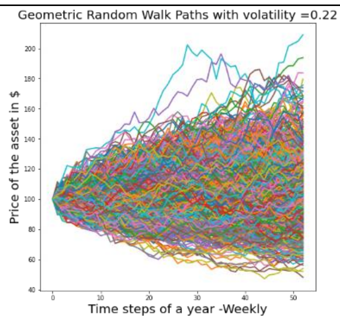
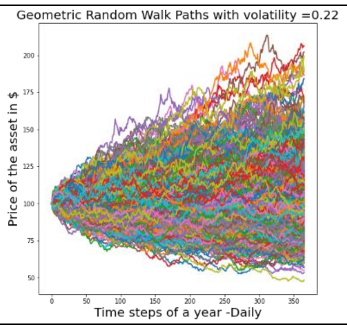






Multi-Step MC Barrier option (different volatilities):

Volatility/Time steps	Monthly	Weekly	Daily
0.18			
0.2			

0.22			

Comparing the pricing strategies for European option also considering volatilities

1. Since the Black-Scholes formula is obtained from solving the differential equation of the underlying stochastic process, it represents the most accurate approximate of the options pricing.
2. When the pricing is done by simulation (discretization of the stochastic process), discretization error occurs as we cannot simulate future prices at infinitely small-time intervals. This discretization error can be reduced and the distribution of the price after many steps will indeed approximate a lognormal distribution with the correct parameters (but it is computationally intensive). Thus, in multi-step MC pricing we reduce the error by generating the whole path even if we are only interested in the end of time horizon.
3. The price of the barrier option was lower than that of the European option, this is because the in the knock-in barrier option the payoff is only obtained when the price of the underlying asset crosses the barrier.
4. The price change is proportional to the change in volatility for call as well as put option.
5. On an average, the percentage change in price as compared to the original inputs of volatility by 10% was about 10% increase or decrease in the direction of the volatility.

Choosing number of time steps and number of scenarios in MC pricing to get the same price as given by the Black-Scholes formula.

1. When the pricing is done by simulation (discretization of the stochastic process), discretization error occurs as we cannot simulate future prices at infinitely small-time intervals.
2. Here, the distribution of the price approximated is normal rather being lognormal as obtained in the by the Black-Scholes formula. This discretization error can be reduced and the distribution of the price after many steps will indeed approximate a lognormal distribution with the correct parameters (but it is computationally intensive).
3. To check the accuracy of the results and in-depth analysis a procedure was designed to compute the price on monthly, weekly and daily basis i.e. number of steps used were 12, 52 and 365 respectively. Thus, was done to see the sensitivity of the number to steps to the accuracy of the approximation.