MIE 1622 Assignment 4 Report

Implement pricing functions in Python

```
# Complete the following functions
def BS_european_price(S0, K, T, r, sigma):
  # --------- Insert your code here --------- #
  d1 = (np.log(S0/K)+(r+(sigma**2)/2)*(T))/(sigma * np.sqrt(T))
  d2 = d1-sigma*(np.sqrt(T))
  c = norm.cdf(d1)*S0-norm.cdf(d2)*K*(np.exp(-r*(T))) # call option
  p = norm.cdf(-d2)*K*(np.exp(-r*(T)))-norm.cdf(-d1)*S0 # put option

  return c, p
```

```
def MC_european_price(S0, K, T, r, mu, sigma, numSteps, numPaths):
  # --------- Insert your code here --------- #
  paths = np.zeros((numSteps+1,numPaths))
  call_payoff = np.zeros((numPaths,1))
  put_payoff = np.zeros((numPaths,1))

  # dT is the time increment (in years)
  dT = T/numSteps
  paths[0] = [S0]*numPaths # initial price of path

  # MC simulation of prices
  for iPath in range(numPaths):
      for iStep in range(numSteps):
          paths[iStep+1,iPath] = paths[iStep,iPath]*np.exp((mu-0.5*sigma**2)*dT+sigma*np.sqrt(dT)*np.random.normal(0,1))

  for iPath in range(numPaths):
      call_payoff[iPath] = np.maximum(paths[numSteps,iPath]-K,0)*np.exp(-r*T)
      put_payoff[iPath] = np.maximum(K-paths[numSteps,iPath],0)*np.exp(-r*T)

  c = np.mean(call_payoff)
  p = np.mean(put_payoff)

  return c, p, paths
```

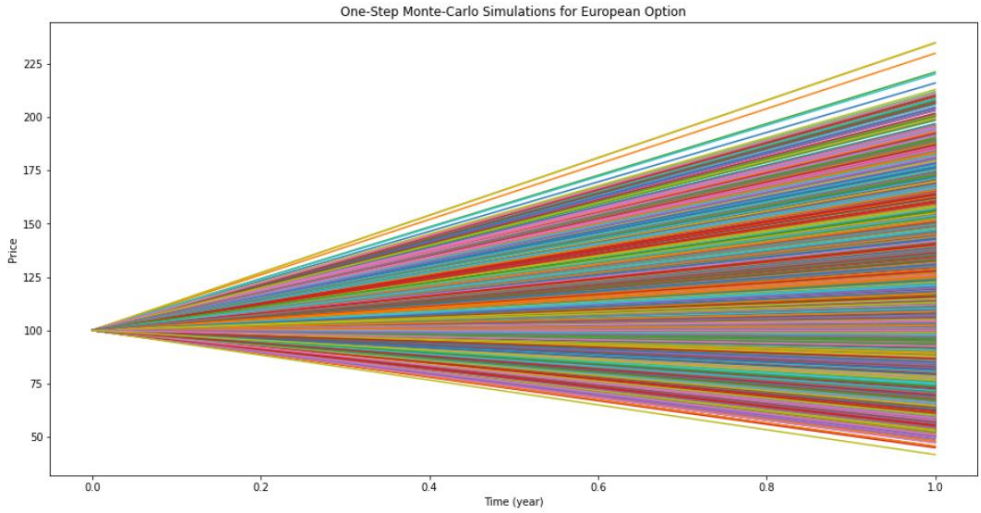
```
def MC_barrier_knockin_price(S0, Sb, K, T, r, mu, sigma, numSteps, numPaths):
  # --------- Insert your code here --------- #
  paths = np.zeros((numSteps+1,numPaths))
  call_payoff = np.zeros((numPaths,1))
  put_payoff = np.zeros((numPaths,1))
  buffers = np.zeros((numPaths,1))

  # dT is the time increment (in years)
  dT = T/numSteps
  paths[0] = [S0]*numPaths # initial price of path


  # MC simulation of prices
  for iPath in range(numPaths):
      for iStep in range(numSteps):
          paths[iStep+1,iPath] = paths[iStep,iPath]*np.exp((mu-0.5*sigma**2)*dT+sigma*np.sqrt(dT)*np.random.normal(0,1))

  # determine if price is higher than barrier
  for iPath in range(numPaths):
      buffers[iPath] = np.sum(paths[:,iPath]>=Sb)
      if buffers[iPath]>0:
          call_payoff[iPath] = np.maximum(paths[numSteps,iPath]-K,0)*np.exp(-r*T)
          put_payoff[iPath] = np.maximum(K-paths[numSteps,iPath],0)*np.exp(-r*T)
      else:
          call_payoff[iPath] = 0
          put_payoff[iPath] = 0

  c = np.mean(call_payoff)
  p = np.mean(put_payoff)

  return c, p
```

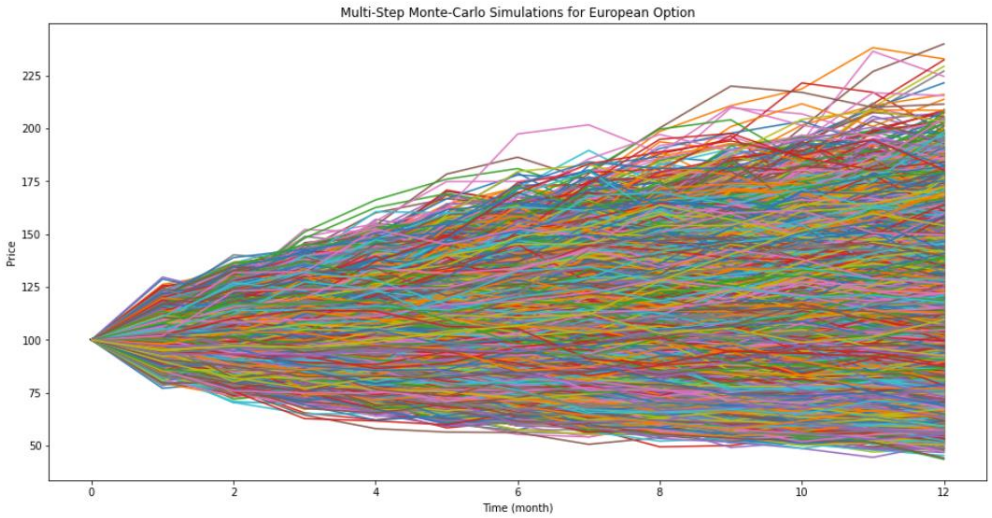
Analyze your results

```
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 price of an European call option is 7.987924298161481
One-step MC price of an European put option is 7.876371202013491
Multi-step MC price of an European call option is 8.047139859126094
Multi-step MC price of an European put option is 7.918108553539543
One-step MC price of an Barrier call option is 7.840802060391148
One-step MC price of an Barrier put option is 0.0
Multi-step MC price of an Barrier call option is 7.954775089739049
Multi-step MC price of an Barrier put option is 1.2792102640954988
```

The number of scenarios and the number of steps for the multi-step Monte Carlo pricing procedure for any option are 100,000 and 12.

The number of scenarios and the number of steps for the one-step Monte Carlo pricing procedure for any option are 100,000 and 1.





There three pricing strategies for the European option, Black-Scholes, one-step Monte Carlo, multi-step Monte Carlo. For Black-Scholes and multi-step Monte Carlo, both prices of call and put option are very close that the difference is around 0.01. On the other hand, one-step Monte Carlo is have bigger difference around 0.1 in comparison with those two.

For Barrier option, both one-step and multi-step Monte Carlo's call prices are similar to one-step Monte Carlo call price for European option. However, the put prices from any pricing procedure for Barrier option is very different from any European option's estimated put prices by being around 1. The reason for this is that the barrier is set to be 110 when the strike is set to be 105. Since the barrier option is a knock-in option and the barrier is higher than the strike, investor certainly get paid off like European option if the call option knocks in and investor can hardly get paid well if the put option knocks in.

```
One-step MC price of an Barrier call option with volatility increased by 10%:8.628128520979281
One-step MC price of an Barrier put option with volatility increased by 10%:0.0
Multi-step MC price of an Barrier call option with volatility increased by 10%:8.746913089748702
Multi-step MC price of an Barrier put option with volatility increased by 10%:1.5786705884331969
One-step MC price of an Barrier call option with volatility decreased by 10%:7.018445426792787
One-step MC price of an Barrier put option with volatility decreased by 10%:0.0
Multi-step MC price of an Barrier call option with volatility decreased by 10%:7.097571545276242
Multi-step MC price of an Barrier put option with volatility decreased by 10%:0.9749655638134963
```

For volatility increases by 10%, except the put price from one-step Monte Carlo, every other price in one-step and multi-step Monte Carlo have increased in a certain amount. Oppositely, volatility decreases by 10%, except the put price from one-step Monte Carlo, every other price in one-step and multi-step Monte Carlo have decreased in a certain amount. The reason for this is that volatility is close related to the possibility that the barrier is crossed. Hence, higher volatility and higher chance that the barrier is crossed, lower volatility and lower chance that the barrier is crossed.

Discuss possible strategies to obtain the same prices from two procedures

```
paths = [10, 100, 1000, 5000, 10000, 1000000]
res_call_opti, res_put_opti = 0.01, 0.01
num_path_call_opti = 0
num_path_put_opti = 0

for Path in paths:
    callMC_opti, putMC_opti, pathsMC_opti = MC_european_price(S0, K, T, r, mu, sigma, numSteps, Path)

    res_call = abs(callMC_opti - call_BS_European_Price)
    if res_call < res_call_opti:
        res_call_opti = res_call
        num_path_call_opti = Path
        callMC_opti = callMC_opti


    res_put = abs(putMC_opti - putBS_European_Price)
    if res_put < res_put_opti:
        res_put_opti = res_put
        num_path_put_opti = Path
        putMC_opti = putMC_opti
```

```
print('optimal number of scenarios:', max(num_path_call_opti, num_path_put_opti))
print('Error in price:', max(res_call_opti,res_put_opti))
```

```
optimal number of scenarios: 1000000
Error in price: 0.004449967887824258
```

```
steps = [1, 4, 8, 12, 16, 20, 24, 28, 32, 36, 40]
res_call_opti, res_put_opti = 0.01, 0.01
num_step_call_opti = 0
num_step_put_opti = 0

for Step in steps:
    callMC_opti, putMC_opti, pathsMC_opti = MC_european_price(S0, K, T, r, mu, sigma, Step, numPaths)

    res_call = abs(callMC_opti - call_BS_European_Price)
    if res_call < res_call_opti:
        res_call_opti = res_call
        num_step_call_opti = Step
        callMC_opti = callMC_opti


    res_put = abs(putMC_opti - putBS_European_Price)
    if res_put < res_put_opti:
        res_put_opti = res_put
        num_step_put_opti = Step
        putMC_opti = putMC_opti
```

```
print('optimal number of time-steps:', max(num_step_call_opti, num_step_put_opti))
print('Error in price:', max(res_call_opti,res_put_opti))
```

```
optimal number of time-steps: 40
Error in price: 0.007672450435513234
```

The method I use to find the optimal number of steps or number of scenarios to get the same prices for multi-step Monte Carlo and Black-Scholes price procedure, is to use the original number of steps or number of scenarios and then set the difference between Monte-Carlo prices and Black-Scholes prices under 0.01 as the threshold while looping through a list of steps or scenarios.