Lab 6: Monte Carlo method for pricing options

The term *Monte Carlo methods* is used for a broad class of computational algorithms where an answer is obtained by repeated random sampling. It can always be applied for computing expected values (since by Law of Large numbers, the mean of a large random sample converges to the expected value of the underlying random variable).

The method can often also be applied to finding answers to questions related problems which do not contain originally any randomness. For example, numerical computation of various integrals can be related to computing expected values of some continuous random variables and then the expected value can be found by using random sampling. Actually, this was the first practical application of the method (see Wikipedia).

In this course we use the method for finding approximately expected values of various random variables. For this application the main result which allows us to estimate the error of computing a sample mean instead of exact expected value is as follows.

For warm-up, let us see MC method in action in case of some simple examples.

1 Exercise 1

Use MC method for computing approximately EY for $Y = X^2$, where $X \sim U(0,1)$ (uniformly distributed in the interval (0,1)). Use a sample of 10000 values of Y, compute also error estimate which is valid with probability 0.99. Is the actual absolute error smaller than the estimate?

Hints Uniform random values can be generated by the command `np.random.rand(n)`, where `n` is the number of random values generated. The exact value corresponds to the integral $\int_0^1 x^2\,dx$ (do you know why?)

Solution. For checking your solution: the computed error should usually be between 0.0076 and 0.0078, the actual error should be smaller than the error estimate approximately 99 times out of 100 if you run the code many times, so in most cases you should see that the actual error is smaller than the error estimate.

```
In [ ]: import numpy as np
        from scipy import stats
        n=
        X=
        Y=
        #compute approximate expected value
        EY_mc=
        #compute error estimate
        alpha=
        error=
        print("error = {}".format(error))
        EY=
        print(np.abs(EY-EY_mc)<error)
```

2 Exercise 2

Find approximately $E(\sqrt{X^2+Z^2})$, where $X \sim N(0,1)$ and $Z \sim N(0, sd = 2)$ by using 100000 generated pairs of X and Z values. Find also an error estimate which is valid with probability 0.5 (so called probable error).

Solution. For checking: the expected value should be approximately 1.933 and the error estimate should be smaller than 0.003

```
In [ ]:
```

Often it can be shown (or it is assumed in the case of certain market model) that the price of an European option can be expressed as the expected value

$$V = E[e^{-rT} p(S(T))],$$

where $S(T)$ is generated according to a certain stochastic differential equation. If we know the distribution of $S(T)$, we can use Monte Carlo method for computing an approimate price by generating n values of the random variable $S(T)$: $S(T)_1, S(T)_2, \ldots, S(T)_n$ and computing the arithmetic average of the function under the expectation:

$$V \approx \bar{V}_n = \frac{e^{-rT}}{n} \sum_{i=1}^{n} p(S(T)_i).$$

From the Theorem above it follows that

$$P\left(|V - \bar{V}_n| \leq \frac{-\Phi^{-1}(\frac{\alpha}{2})\mathrm{std}(Y)}{\sqrt{n}} \right) \approx 1 - \alpha$$

for large values of n. Here $Y = e^{-rT} p(S(T))$.

3 Exercise 3

If we assume that the Black-Scholes market model with constant volatility holds, then we have to generate $S(T)$ according to the stochastic differential equation

$$dS(t) = S(t)((r - D)\, dt + \sigma\, dB(t).$$

From the lecture we know that the solution of the equation is

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

so $S(T) = S(0) e^{(r - D - \frac{\sigma^2}{2})T + \sigma X}$, where $X \sim N(0, sd = \sqrt{T})$. Write a function MC1, that for given values of $S(0), r, D, \sigma, T, \alpha$ and n and for a given payoff function p computes (which should work with vectors) an approximate option value and its error estimate holding with the probability $(1 - \alpha)$ by Monte-Carlo method, using n generated stock prices. Verify the correctness of the function by Black-Scholes formulas for put and call options in the case $S(0) = 100$, $E = 100$, $\sigma = 0.6$, $T = 0.5$, $r = 0.02$, $D = 0.03$, $\alpha = 0.05$ and $n = 10000$. How often the actual error is larger than the error estimate if you use MC1 1000 times?

Warning: in the code, the given value of the initial stock price $S(0)$ should be denoted by a suitable name like `S0`. It is not possible to use commands like `S(0)=100` since in python writing `S(0)` means that we want to compute the value of a function `S` when it's argument is 0 and it does not make sense to try to define what a value of a function is for a particular agument outside of the definition of the function.

Solution

```
In [ ]: #define the function
        def MC1(S0,r,D,sigma,T,alpha,n,p):
            #generate the stock prices
            ST=
            #compute the discounted payoff values
            Y=
            price=
            error=
            return [price,error]
```

To verify the correctness of the code, we need the function that computes the price of call options according to Black-Scholes formula. If this function is in a file, it can be imported from it. Alternatively, just copy the code for the call option here

```
In [ ]: #import sys
        #indicate the location of the file
        #sys.path.append("locatin_of_your_file")
        #import BSformulas as bs
```

Verify the correctness

```
In [ ]: #define the payoff function
        def p_call(s):
            return np.maximum()
        #compute the exact price
        exact_price=
        #compute the approximate price

        Print values
        print("Exact price: {}".format(exact_price))
        print("MC price with error estimate: {}".format(???))
```

Is the error estimate valid for this computation?

To verify further the correctness of the function, let us do 1000 computations and check, how often the error estimate is wrong. If the code is correct, the result should be wrong approximately 50 times. More precisely, the number of times the estimate is wrong should behave like random variable with binomial distribution Bin(1000,0.05), having mean 50 and standard deviation $\sqrt{1000 \cdot 0.05 \cdot 0.95} \approx 6.892$. This means that the observed number of wrong estimates should usually (aobut 19 times out of 20) between $50 - 13.8$ and $50 + 13.8$.

```
In [ ]: N=1000
        prices=np.zeros(N)#vector for storing approximate prices
```

```
errors=???#vector vor error estimates
for i in range(??):
    #compute the result
    result=???
    #store the components of the result in the corresponding vectors
print(np.sum(np.abs(prices-exact_price)>errors))
```

Is the result in the expected range?

Very often it is not possible to generate $S(T)$ values that correspond exactly to the stochastic differential equation; then it is necessary to use some approximation methods. One such method is the Euler's method (or more precisely, Euler-Maruyama method), where we divide the interval $[0, T]$ into m equal subintervals and use the approximations (int the case of Black-Scholes market model)

$$S_{i+1} = S_i(1 + (r - D)\,\Delta t + \sigma(S_i, t_i)X_i), i = 0, \ldots, m - 1,$$

where S_i are approximations to $S(i\Delta t)$, $\Delta t = \frac{T}{m}$ and $X_i \sim N(0, \sqrt{\Delta t})$. Instead of $S(T)$ we use S_m, thus we use Monte-Carlo method to compute an approximate value of $\hat{V}$, where

$$\hat{V}_m = E[e^{-rT} p(S_m)].$$

Since S_m for a fixed m does not have exactly the same distribution as $S(T)$, we have in general $\hat{V}_m \neq V$ and therefore Monte-Carlo method converges to a value that is different from the option price.

4 Exercise 4

Write a function MC2 that computes approximate option prices so that the stock prices are generated according to Euler's method. Determine how large is the difference between V_m (the expected value of $e^{-rT} p(S_m)$, where S_m is computed by Euler's method) and the correct option price in the case of European call option, using the same parameters as in the previous exercise for $m = 2, 4, 8, 16$. In order to see the difference, large enough value for n should be used (if possible, the corresponding MC error should be at least 5 times smaller than the computed difference).

Solution

```
In [ ]: def MC2(S0,r,D,sigma,T,alpha,n,p,m):
            #use Euler's method to compute ST
            #assume that sigma is a function of stock price and time
            delta_t=T/m
            S=np.zeros(shape=(m+1,n))
            S[0,:]=S0
            t=np.linspace(0,T,m+1)
            for i in range(1,m+1):
                S[i,:]=S[i-1,:]*(1+(r-D)*delta_t+sigma(S[i-1,:],t[i-1])*np.sqrt(delta_t)*np.rand
            ST=S[m,:]
            Y=np.exp(-r*T)*p(ST)
            price=np.mean(Y)
            error=-phi_inv(alpha/2)*np.std(Y)/np.sqrt(n)
            return [price,error]
```

Remark: Since we need only the value of S_m, we do not need to fill the full matrix. It is enough if we have only the values
for the previous time moment available to compute the next one. Advanced students may define an alternative function for `MC2` which uses only a vector with `n` components to store stock prices

Let us verify the correctness of the function `MC2`.

```
In [ ]: n=100000
        m=2
        #define sigma as a constant function
        def sigma(s,t):
            return ??
        #compute the actual error and compare it to the error estimate
```

The first component of the result of MC2 is $V_m + MC\_error$, so if we compute the difference of it and the exact price, we see the $V_m - V + MC\_error$. When the error of the MC method is relatively large compared to the difference we computed, we do not see clearly what we wanted to see (the actual difference $V_m - V$). So we have to use much larger n. You should see that the actual error of the answer is not 5 times larger than the error estimate, so please repeat the computations with large `n`

```
In [ ]: n=10000000
        m=2


        #store the computed price in variable price_2 and actual error in the variable error_2
        price_2=
        error_2=
```

```
In [ ]: print([error_4/error_2,error_8/error_4,error_16/error_8])
```

Theoretically, for large enough m we should have $\frac{error_{2m}}{error_m} \approx 0.5$. Is this property visible from the results?

It is known that if p is continuous and has bounded first derivative (ie it is Lipshitz continuous), then

$$\hat{V}_m = V + \frac{C}{m} + o(\frac{1}{m}),$$

where C is a constant that does not depend on m and $m \cdot o(\frac{1}{m}) \to 0$ as $m \to \infty$. Thus, if we knew the value of the constant C, we could choose large enough m (so that the absolute value of the term $\frac{C}{m}$ is small enough, for example less than $\frac{\varepsilon}{2}$, where ε is the desired accuracy) and then use MC method with large enough n so that the MC error estimate is also small enough (less than $\frac{\varepsilon}{2}$). There is one trouble: we do not know C. One possibility to estimate C is by computing V_m approximately (by MC method) for several values of m and then find the best values of V and C such that $V + \frac{C}{m}$ are as close as possible to the obtained results. In statistics, such fitting is called linear regression and there is a function `linregress` in the `stats` subpackage of `scipy` for computing the regression parameters.

5 Exercise 5

Find an approximate value of C by fitting a linear regression line to the approximate option prices V_2, V_4, V_8, V_{16} computed in the previous exercise. Using the value of C, find a value of m such that $\frac{|C|}{m} \leq 0.03$.

Solution We have to take into account that in the command `stats.linregress` the fitted line has a form $y = a + b \cdot x$, where x values should be in the first parameter of the command and y values should be in the second parameter of the command. So the first parameter should be $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}$. The value of constant C we want to estimate is the coefficient of x in the equation and this is also called slope of the regression line.

```
In [ ]: #define array of computed prices
        prices=

        #define a vector of m values which were used for computing prices
        m_values=
        model=stats.linregress(m_values,prices)
        print(model)
```

Determine the slope, the absolute value of it should be used for estimating the value of m for the final computation

```
In [ ]: #compute m such that |C|/m is small enough. Make sure that the result is an integer.
        m=
        print(m)
```

6 Exercise 6

With the value of m found in the previous exercise, compute V_m so that MC error is less than 0.03 with probability 0.95. Find also the actual difference of V_m you found and the exact option price. Is the actual difference smaller than 0.06?

Solution Do a computation with a reasonably large value of n:

```
In [ ]:
```

If MC error is not small enough, the value of `n`should be increased. If we look at how the MC error estimate depends on n, it is clear that the error estimate is reduced k times, if we multiply n by k^2. Use this observation to compute the value of `n` which is needed for the final computation

```
In [ ]: print("New value of n: {}".format(n))
```

The value should be approximately 4 million. Compute the final answer

```
In [ ]: final_answer=
        print("The final answer:",final_answer)
        print("Actual error:", abs(final_answer[0]-exact_price))
```

Did you get answer with the desired accuracy?

It may happen that the actual error of the final answer is slightly larger than the allowed error. There are two possible causes for this happening: we may have used too small values for m for estimating the constant C (the behaviour C/m holds asymptotically for large enough m) or we just had a bad luck (the error estimate should hold with probability 0.95, so maybe we see an outcome which happens with probability less than 0.05)