Coursework 1
Report

MATH60082

ID: 10328179

Summary

Overall, this report comprises of two major sections: **Task 2.1: Stock options** and **Task 2.2: Path dependent Options**. C++ and Python codes are implemented to resolve these two Monte-Carlo based questions.

In Section 1, general parameters are maturity date (T) = 1.75, volatility (σ) = 0.27, interest rate (r) = 0.03, dividend rate (D_0) = 0.03, strike price 1 (X_1) = 60000 and strike price 2 (X_2) = 105000. While, different path number (N) is given and will be mentioned in each subsection. This section presents the solution to valuing the portfolio containing one long put option, one short call option and ($2 \times X_2$) binary call options at time t = 0. Specifically, at first, basic Monte-Carlo method is used for the primary approximation of portfolio values as initial stock price equals $S_0 = X_1$ and X_2. Then, against the increasing path number N, two graphs are given with the analytical solution of the portfolio to illustrate the convergence of results. Additionally, the subsection 2.1.3 and 2.1.4 explore the efficiency of Antithetic variates technique with confidence intervals' and path numbers' change. Lastly, the advanced technique of Quasi-Monte Carlo method on the basis of Sobol sequences is employed for the research in its benefits and efficiency.

In Section 2, the maturity date (T) = 0.75, the initial stock price (S_0) = 66500, volatility (σ) = 0.41, interest rate (r) = 0.06, dividend rate (D_0) = 0.06, strike price (X) = 66500, upper barrier (B) = 113200 and timesteps (K) = 35 are provided as constants. This section contains the subsection 2.2.1 to show the path dependent option value after applying the working code in basic method, the subsection 2.2.2 for the demonstration of changing outcome with different path numbers N and time steps K and the subsection 2.2.3 for the innovative equipment of Antithetic variates method to calculate the optimal value precisely within ten seconds.

2.1. Stock Options

▪ 2.1.1

In this subsection, the program calculating portfolio values when $S_0 = X_1$ and X_2 is included in the appendix (*i*).

To improve the approximation accuracy, path number N = 0.7×10^7 is chosen. Thus, the final results using the **basic method** are:

"The value of portfolio as time t=0, S0=X1 is: -670.3

The value of portfolio at time t=0, S0=X2 is: -98951.8"

Which is a good estimate compared to π(S=60000, t=0) = -661$ and π(S=105000, t=0) = -99000$ given in the question.

▪ 2.1.2

The program about analytical formula will be exercised to conduct the comparison. (The code is included in the appendix (*ii*).)

And we obtain the following results using the **analytical formulas**:

"The value of portfolio as time t=0, S0=X1 is: -660.801

The value of portfolio as time t=0, S0=X2 is: -98961.7"

An assumption is made that the analytical formulas offer an exact result on the portfolio values at time t = 0 almost surely. This can be applied in our figures as two horizontal lines ($L1, L2$ in plots) to illustrate the effect from the increasing N. Also, 100 different values ranging from 1000 to 100000 with the increment of 1000 are generated for a better observation of the possible trend.

Store the path number N, portfolio values with $S_0 = X_1$ and X_2 in a Csv file and plot two line graphs with Python:

Figure 1: Portfolio value at time = 0 with S0 = X1

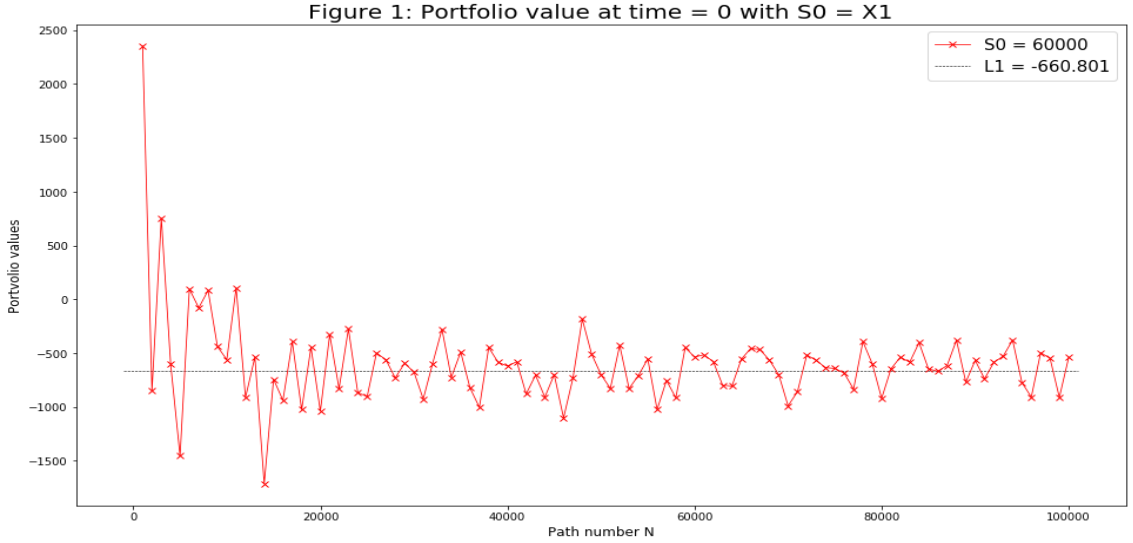
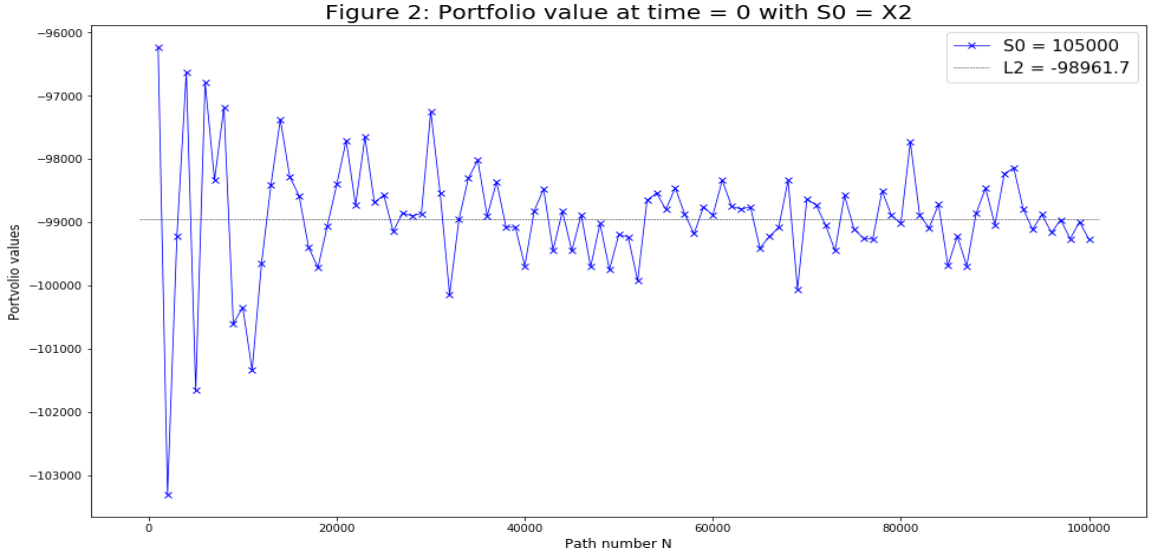


Figure 2: Portfolio value at time = 0 with S0 = X2



Intuitively, ***Figure 1*** *and* ***2*** shows the trend that gradually, the points swing much more closely around the critical line L1 = -660.801 and L2 = -98961.7 respectively as path number N is increasing, illustrating the convergence in portfolio values. This indicates the conclusion that the larger the path number is, the more accurate the portfolio value is becoming.

▪ 2.1.3

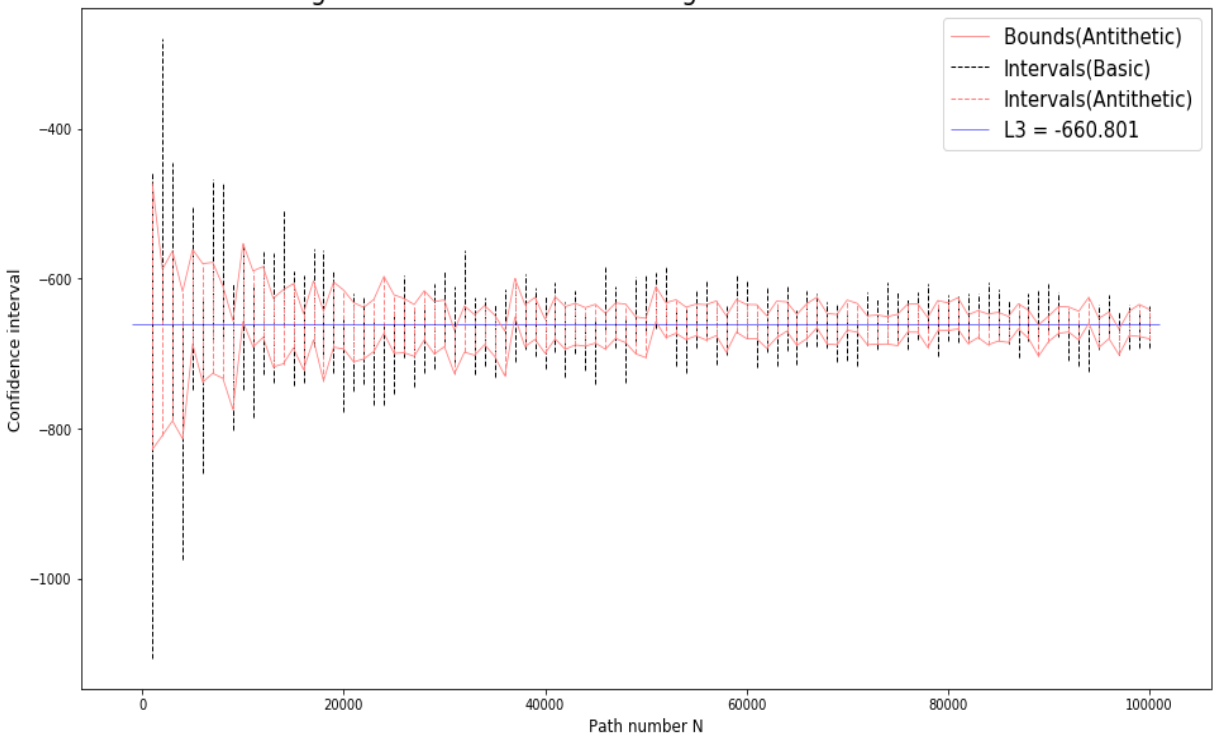
Utilizing the basic method, the exact **confidence interval** can be produced by the code in appendix (***iii***). While sample number is fixed at 100 in each vector.

The result when N = 1000, M =100 is: *"[-1105.38, -457.235]"*

To demonstrate the relationship between path number N and confidence interval, we use the loop to generate the data as N rises from 1000 to 100000 with increment of 1000.

Plot these confidence intervals out with multiple vertical lines and insert a L3 $=X_1$, we obtain:

Figure 3: Confidence interval against N for two methods



The black dashed lines in ***Figure 3*** provides the information that especially before N = 20000, the low precision of confidence interval to predict the exact portfolio value (L3 = -660.801) can be realized. However, as N goes up, the confidence interval turns out to be getting narrower, meaning high confidence and accuracy in estimation. Positioned as a denominator, the increasing N has dispersed the influence from the wrongly deviant results given by Monte-Carlo.

In particular, the interval progresses from **[-1105.38, -457.235] (N=1000)** to **[-692.378, -635.496] (N=100000).**

▪ 2.1.4

This subsection is trying to discover the efficiency of **Antithetic variates** in variance reduction.

Under new method, the confidence interval as N = 1000, M =100 is:

"[-827.728, -473.074]"

And $|-827.728-(-473.074)| = 354.654 < |-1105.38-(-457.235)| = 648.145$

Initial N, M values remained, the new confidence interval clearly reveals the advantage of Antithetic variates in approaching the best estimate. This can also be noticed in ***Figure 3***, where red bounds and intervals show that the confidence intervals produced by Antithetic variates are strictly shorter than those generated with basic method (black dashed lines). This figure witnesses that the accuracy of estimate in new method is highly promoted.

Moreover, to construct a fair comparison, the running time of the code can be taken into account. We roughly record the time before the confidence interval reaches an expected precision for estimation. For instance, a standard deviation (if standard deviation < =15) is determined to test the time elapsed. The result is:

"Total time elapsed for basic method is: 271.771 seconds,

Total time elapsed for Antithetic variates method is: 66.075 seconds"

This presents the great superiority of Antithetic variates method.

Additionally, another feasible method is that we only compute the confidence interval for once, while 10000 paths are set for basic method and 5000 paths for Antithetic method. Then, merely the time elapsed for one simulation is returned, which is **0.821082** and **0.689409** seconds respectively. Antithetic variates method still outperforms the basic one.

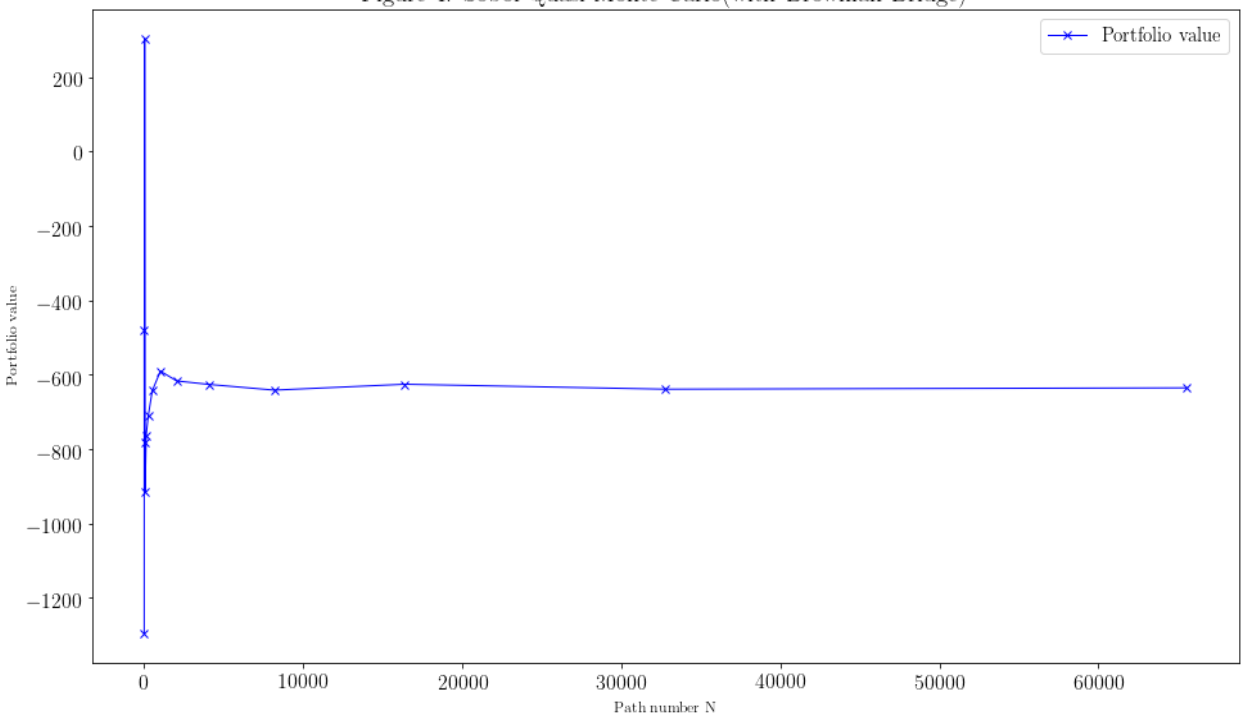
After two types of examinations above, it is almost certain that the Antithetic variates method is much more efficient than basic method, considering both time and estimation accuracy.

- **2.1.5**

In this subsection, **Sobol Quasi-Monte Carlo method** will be employed with Python (*Giles, 2018*) as an extension of the basic method. Combined with the knowledge in digital scrambling and Brownian bridge, this method is computationally efficient for its *lower error*, sequential *filling in the vacant spots* (*Dias, 2004*) and replacement in pseudo-random numbers.

Here is the figure of portfolio values:

Figure 4: Sobol Quasi-Monte Carlo(with Brownian Bridge)



Noticed from what the code produces, Quasi-Monte Carlo method with sobol sequences presents the advantage in performing the least simulations (2^{13} = 8912 < 10000 paths) to reach an almost optimal output: **-641.933858**, which is very efficient compared to the paths taken in 2.1.1 (After around 100000 steps, an optimal estimate occurs).

However, the disadvantage is obvious that it does not reach a value close to the analytical solution. It may be due to the less scrambles and simulations or errors in predetermined sobol sequences.

While, to obtain a more accurate one, extremely longer time is needed to verify the result with larger N, for the only path numbers accepted in this method are the powers of 2. In this case, the relatively less time (733.671875 seconds) is

used with (2^16=) 65536 paths, but for the (2^17=) 131072 paths, the time will be extended to over 20 minutes to find the solution, drastically lowering the computing efficiency.

Nevertheless, it is still a supremely efficient method to predict portfolio values if the simulation time is limited.

2.2. Path Dependent Options

- ## 2.2.1

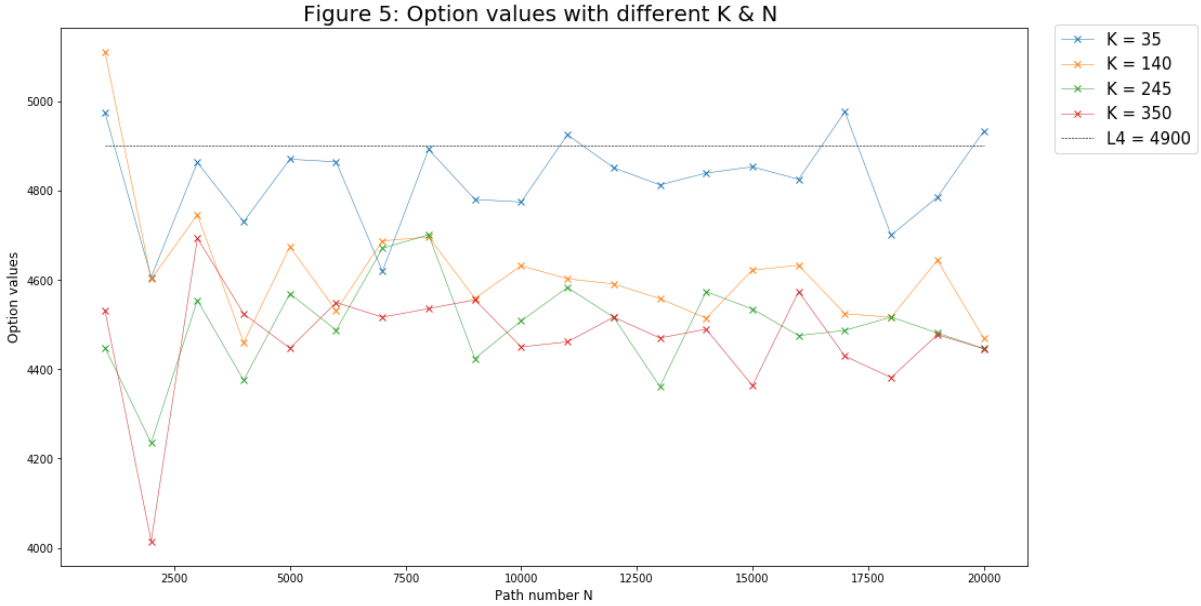
The code will be attached to the appendix (*iv*). And through the csv file produced, the output when k =35, N =100000 is:

"The up-and-out barrier Call Option value is: 4938.59 (barrier hit for 8224 times)"

- ## 2.2.2

This subsection will concern about the relationship among the option value, time steps K and the path numbers N. Initially, a series of K (35 to 350 with the interval of 35) and N (1000 to 10000 with the increment of 1000), i.e. 100 values will be generated. And, the numbers with K = 35, 140, 245,350 will be plotted as shown:

Figure 5: Option values with different K & N



To capture the significant movement, we should select more time steps K and path numbers N that are notably distinct to assist with the exploration. For instance, a table of new option values with N =100000, K =35, 350,3500,35000 can be created for the test:

Table 6: Option values with selected numbers

Path (N)	100000			
Time step (K)	35	350	3500	35000
Option value	4848.68	4469.58	4337.21	4347.24

From both answers provided, the values at K =35 are all very close to the rough estimate 4900 (L4 = 4900). Starting from the tendency in respect of the path numbers N, it is visible in ***Figure 5*** that the line of any K moves stochastically against the growing path N while the volatility of option values is declining, i.e. the fluctuation of values is growing smaller. To be concrete, we can compare the huge waves on the left of figure to the steady swings on the right. This indicates the decreasing probability to produce abnormal option values with the larger number of simulations.

However, focusing on the parameter K, ***Table 6*** reflects a clear trend that the rise of K may lower the option values returned by the code. From what ***Figure 5*** shows, the line 'K =35' is notably above lines with larger K. Furthermore, though it is difficult to detect the exact differences among other three lines, we can still estimate that roughly, the line 'K =140' is over the line 'K =245' and the line 'K = 245' is over the line 'K=350'.

This phenomenon happens owing to the higher sensitivity when the increasing K causes a higher volatility (*Wang, 2011*). In this scenario, the initial spot price will obtain a higher probability to reach the knocked-out price and breach the upper barrier, leading to more occurrence of knocked-out options and the drop in average option value.

- **2.2.3**

In this subsection, Black-Scholes barrier option price formula (*Davis, 2008*) will be introduced to provide the analytical solution and help verify the efficiency of Antithetic variates method. The formula is followed:

$$S_0 \left[N(d_1) - N(x_1) + \left({}^{B}/_{S_0} \right)^{2\lambda} \left(N(-y) - N(-y_1) \right) \right] + \mathrm{K}\, e^{-rT} \left[-N(d_2) + N\left(x_1 - \sigma\sqrt{T} \right) - \left({}^{B}/_{S_0} \right)^{2\lambda-2} \left(N(-y + \sigma\sqrt{T}) - N(-y_1 + \sigma\sqrt{T}) \right) \right] \quad (1)$$

Where d_1, d_2 are the coefficients in Black-Scholes model including dividends and

$$x_1 = \frac{\log(S_0/B)}{\sigma\sqrt{T}} + \lambda\sigma\sqrt{T}, \qquad y_1 = \frac{\log(\mathrm{B}/S_0)}{\sigma\sqrt{T}} + \lambda\sigma\sqrt{T}$$

$$y = \frac{\log(B^2/(S_0 K))}{\sigma\sqrt{T}} + \lambda\sigma\sqrt{T}, \qquad \lambda = \frac{r - D_0 + \sigma^2/2}{\sigma^2} \quad (2)$$

(The code is attached in the appendix (***v***).)

Thus, the solution returned by the program is:

"The barrier option ends up with the value: 1647.17 "

This result surely corresponds to the primary features of up-and-out barrier options: low purchasing prices to attract the investors and high flexibility to hedge the exposure risks.

Then, applying the **Antithetic variates method** again, we may generate another estimation. To realize this, we delete the redundant sentences in the program and add in the 'simple timer' to guarantee that only 10 seconds is permitted for the whole loop.

Taking path number N = 1000 in consideration of the maximization in sample numbers, the expected outcome then is (Check the program in appendix (***vi***) for more. The results may vary but they are all approaching analytical solution):

"The average barrier option price is: 1844.29 with 291 times simulations"

(This means total simulations are 291 × 1000 = 291000 times.)

This is a reasonable estimate compared to the analytical solution.

Conclusion

Monte-Carlo method is an efficient method to manage the option pricing in expectation. Though the minor drifts exist, various variance reduction techniques can be employed to guarantee both the accuracy and efficiency.

References

Davis, M.H.A (2008), Black-Scholes Barrier Option Value [Online] Available at: http://wwwf.imperial.ac.uk/~mdavis/course_material/NMF/BARRIER.PDF (Accessed: 13 March 2019)

Dias, M.A.G. (2004), *Visual FAQ's on Real Options/FAQ number 15*. [Online] Available at: http://marcoagd.usuarios.rdc.puc-rio.br/faq15.html (Accessed: 10 March 2019)

Giles, M (2018), *Monte-Carlo Methods* [Online] Available at: http://people.maths.ox.ac.uk/~gilesm/mc/ (Accessed: 7 March 2019)

Wang, B &Wang, L(2011), *Pricing Barrier Options using Monte Carlo Methods* [Online] Available at: https://pdfs.semanticscholar.org/542f/6e1e9338632e3bc5b56dad2515854e34190f.pdf (Accessed: 13 March 2019)

Appendix

The source codes are attached here:

i. *https://raw.githubusercontent.com/hunterkillerbb/Computational-Finance/master/Coursework%201/Coursework%201%20Source%20code/2.1.1.cpp* ***(code for question 2.1.1)***

```
#include <iostream>

#include <random>

#include <cmath>

#include <vector>

#include <algorithm>

#include <vector>

#include <fstream>

#include <conio.h>

using namespace std;


double putoption(double S, double X)

{

	return max(X - S, 0.);

}


double calloption(double S, double X)


{

	return max(S - X, 0.);
```

```
}

double binarycalloption(double S, double X)
{
	if (S>X)
	{
		return 1;
	}

	else
	{
		return 0;
	}
}

double payoff(double S, double X1, double X2)
{
	return putoption(S, X1) - calloption(S, X2) - 2 * X2 * binarycalloption(S, X2);
}

//the option value at time t = 0
double monteCarlo(double S, double X1, double X2, double r, double D0, double
sigma, double T, int N)
```

```
{
        // declare the random number generator
        static mt19937 rng;
        // declare the distribution
        normal_distribution<> ND(0., 1.);
        ND(rng);
        // initialise sum
        double sum = 0.;
        for (int i = 0; i<N; i++)
        {
                double phi = ND(rng);
                // calculate stock price at T
                double ST = S * exp((r - D0 - 0.5*sigma*sigma)*T +
phi*sigma*sqrt(T));
                // add in payoff
                sum = sum + payoff(ST, X1, X2);
        }
        // return discounted value
        return sum / N*exp(-r*T);
}

int main()
{
```

cout << "Press any key to generate the value as S0 = X1 and X2:\n" << endl;

_getch();

*cout << "\nThe value of portfolio as time t=0, S0=X1 is: " << monteCarlo(60000, 60000, 105000, 0.03, 0.03, 0.27, 1.75, 0.7*1e7) << endl;*

*cout << "The value of portfolio at time t=0, S0=X2 is: " << monteCarlo(105000, 60000, 105000, 0.03, 0.03, 0.27, 1.75, 0.7*1e7) << endl;*

return 0;

}

ii. *https://raw.githubusercontent.com/hunterkillerbb/Computational-Finance/master/Coursework%201/Coursework%201%20Source%20code/2.1.2.cpp*

iii. *(a). https://raw.githubusercontent.com/hunterkillerbb/Computational-Finance/master/Coursework%201/Coursework%201%20Source%20code/2.1.3/2.1.3%20basic%20method.cpp* ***(basic method)***

(b). https://github.com/hunterkillerbb/Computational-Finance/blob/master/Coursework%201/Coursework%201%20Source%20code/2.1.3/2.1.3%20antithetic%20variates%20method.cpp ***(antithetic variates method)***

iv. *https://raw.githubusercontent.com/hunterkillerbb/Computational-Finance/master/Coursework%201/Coursework%201%20Source%20code/2.2.1* ***(code for question 2.2.1)***

#include <iostream>

#include <iomanip>

#include <cmath>

#include <vector>

#include <fstream>

#include <random>

```
#include <algorithm>
#include <conio.h>
using namespace std;


double barriercallOption(double S, double  sigma, double  r, double D0, double  T, double  X, double B, int K, int N)
{
	static mt19937 rng;
	normal_distribution<> ND(0., 1.);
	double sum = 0.;
	double total = 0;
	int barriernum = 0;
	for (int n = 0; n < N; n++)
	{
		// now create a path
		double dt = T / K;
		vector<double> stockPath(K + 1);
		stockPath[0] = S;
		for (int i = 1; i <= K; i++)
		{
			double phi = ND(rng);
			stockPath[i] = stockPath[i - 1] * exp((r - D0 - 0.5*sigma*sigma)*dt + phi*sigma*sqrt(dt));
		}
```

```
// and calculate A
double A = S;
for (int i = 1; i <= K; i++)
{
    A = A + (stockPath[i] - stockPath[i - 1]);

    // add in the payoff to the sum

    if (A > B)

    {
        barriernum += 1;
        cout << "\nHit the barrier " << barriernum << "
time(s)..." << endl;
        sum = 0;
        break;
    }

    sum = max(A - X, 0.);

}

total += sum;
```

```
            cout << "\nIteration for path number: " << N << " is finished..." <<
endl;
        }
        return total / N*exp(-r*T);
}

int main()
{

        double S0 = 66500, sigma = 0.41, r = 0.06, T = 0.75, X = 66500, B = 113200,
D0 = 0.06;
        int K = 35;
        int N = 100000;
        cout << "\nThe up-and-out barrier Call Option value is: " <<
barriercallOption(S0, sigma, r, D0, T, X, B, K, N) << endl;

        return 0;
}
```

v. *https://raw.githubusercontent.com/hunterkillerbb/Computational-Finance/master/Coursework%201/Coursework%201%20Source%20code/2.2.3%20BS-barrier%20option%20pricing*

vi. *https://raw.githubusercontent.com/hunterkillerbb/Computational-Finance/master/Coursework%201/Coursework%201%20Source%20code/2.2.3%20time%20limit%20test*

PS:

vii. ***Python code for plots in question 2.1.2***: *https://github.com/hunterkillerbb/Computational-Finance/tree/master/Coursework%201/Coursework%201%20Source%20code/2.1.2%20plot*

viii. ***Python code for plots and csv files in question 2.1.3***: *https://github.com/hunterkillerbb/Computational-Finance/tree/master/Coursework%201/Coursework%201%20Source%20code/2.1.3*

ix. ***Python code for Sobol Quasi-Monte Carlo method***: *https://github.com/hunterkillerbb/Computational-Finance/tree/master/Coursework%201/Coursework%201%20Source%20code/2.1.4/QMC%20Method*

x. ***Python code for plots and csv files in question 2.2.2***: *https://github.com/hunterkillerbb/Computational-Finance/tree/master/Coursework%201/Coursework%201%20Source%20code/2.2.2%20plot%20and%20table*