Introduction to Modern C++

Homework 1 : Black-Scholes formula for basic option pricing

1 Context

In finance, a *European call option* is a contract which gives the buyer the right (but not the obligation) to buy some asset at a specified price K and date T.

Let S_t be the price of the underlying asset at time t. If the call option is exercised at time T, it provides payoff

$$C_T = \max\{0, S_T - K\} \tag{1}$$

In 1973, Black and Scholes showed that if we know the interest rate r, the variability of the underlying asset σ, and the time to maturity $(T - t)$ of the option, then we can know everything about this option pricing. At the same time, their model generates hedge parameters (a.k.a. the "Greeks") necessary for effective risk management of option holdings. For this work Scholes received in 1997 the Nobel Prize in Economics.

This homework is about implementing the Black-Scholes solution in C++, and computing the Greeks.

2 Black-Scholes formula

The price of an option at time t is given by the formula:

$$c = S \times \Phi(d_1) - K \times \exp(-r \times (T - t)) \times \Phi(d_2) \tag{2}$$

with

$$d_1 = \frac{\ln(S/K) + (T - t) \times (r + \sigma^2/2)}{\sigma\sqrt{T - t}} \tag{3}$$

$$d_2 = d_1 - \sigma\sqrt{T - t} \tag{4}$$

where the function Φ is the cumulative distribution function of the normal distribution, and T and t are measured in years.

Remember that the standard library provides useful mathematical functions:

- `sqrt` for the square root;
- `exp` for the exponential;
- `log` for the natural logarithm.

Task 1. Create the file `blackscholes.cpp` containing these lines:

```cpp
#include <iostream>
#include <cmath>

double option_price(double S, double K, double r, double sigma, double T, double t) {
  double c, d1, d2;

  // Your code here

  return c;
```

```
}

int main() {
  // Nothing for the moment
}
```

Task 2. Write code to compute d_1 and d_2 inside the `option_price` function.

Task 3. Assume that you have access a function Φ called `Phi`. Write code to compute c.

Note: You can use the function `Phi` provided in Appendix A. To do that simply add its code into your file.

3 Example

Stock in company C++4EVER is currently trading at $S = 50$. Consider a call option on C++4EVER with an exercise price of $K = 50$ and a time to maturity $(T - t)$ of 6 months. The volatility of C++4EVER stock has been estimated to be $\sigma = 30\%$ and the current risk-free interest rate for six month borrowing is $r = 10\%$.

Task 4. Use your program to compute the C++4EVER option price. You should find 5.45325

Recall that to compile your program you can use the following terminal command:
`g++ blackscholes.cpp -o homework --std=c++1y`

4 The Greeks

In option trading, a number of important partial derivatives of the option price formula appear:

- Delta $\Delta = \frac{\partial c}{\partial S}$
- Vega $\nu = \frac{\partial c}{\partial \sigma}$
- Gamma $\Gamma = \frac{\partial^2 c}{\partial S^2}$
- Rho $\rho = \frac{\partial c}{\partial r}$

Task 5. Compute analytically Vega and Rho. Note that $\Phi'(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$.

Task 6. Implement C++ functions Vega and Rho that perform these computations:

```
double Vega(double S, double K, double r, double sigma, double T, double t) {

  // Your code here

}

double Rho(double S, double K, double r, double sigma, double T, double t) {

  // Your code here

}
```

You may use the code provided in Appendix B for Φ'.

Task 7. Test your code on the same example as for the option price. You should find `Vega = 13.3046` and `Rho = 13.1168`

You can compute and implement the Delta and Gamma as well if you like. You can test them on the example, you should find `Delta = 0.633737` and `Gamma = 0.0354789`.

A Code for computing Φ

```
double Phi(double z) {
    double b1 =  0.31938153;
    double b2 = -0.356563782;
    double b3 =  1.781477937;
    double b4 = -1.821255978;
    double b5 =  1.330274429;
    double p  =  0.2316419;
    double c2 =  0.3989423;

    if (z >  6.0) {
      return 1.0;
    }

    if (z < -6.0) {
      return 0.0;
    }

    double a = fabsl(z);
    double t = 1.0/(1.0+a*p);
    double b = c2*expl((-z)*(z/2.0));
    double n = ((((b5*t+b4)*t+b3)*t+b2)*t+b1)*t;
    n = 1.0 - b * n;
    if (z < 0.0) {
      n = 1.0 - n;
    }
    return n;
}
```

B Code for computing Φ'

```
double PhiPrime(double x) {
 return 1.0/sqrt(2*M_PI)*expl(-x*x/2);
}
```