Project 1
Vanilla Options in a Black-Scholes World

Sava Spasojevic
savaspasojevic@g.ucla.edu

November 19, 2019

Abstract

The purpose of this project is to implement the pricing of some vanilla options with the Black-Scholes model by multiple methods. We begin with a brief discussion of the Black-Scholes-Merton partial differential equation for the price of an asset modeled as a geometric Brownian motion. The idea behind this derivation is to determine the initial capital required to perfectly hedge a short position in the option. We next implement the Black-Scholes formulas for various options, i.e. a call option, a put option, a digital-call option, a digital-put option, and a zero-coupon bond. We then run a consistency check to make sure that the formulas have been implemented correctly.

Discussion of the Black-Scholes-Merton PDE.

The Black-Scholes-Merton partial differential equation is the equation

$$c_t(t,x) + rxc_x(t,x) + \frac{1}{2}\sigma^2 x^2 c_{xx}(t) = rc(t,x),\ \forall t \in [0,T),\ x \geq 0$$

which satisfies the terminal condition

$$c(T,x) = (x-K)^+$$

where $x = S(t)$, the stock's value at time t and $c(t,x)$ is the value of a call option as a function of time and the stock's value. This equation comes from the more general Itô-Doeblin formula, which, in its differential form, is given by

$$df(t,W(t)) = f_t(t,W(t))dt + f_x(t,W(t))dW(t) + \frac{1}{2}f_{xx}(t,W(t))dt$$

where $W(t)$ is a Brownian motion. We model the stock's price $S(t)$ using the geometric Brownian motion with drift $r-d$

$$S(t) = S(0)e^{(r-d)T - \frac{1}{2}\sigma^2 T + \sigma\sqrt{T}N(0,1)}$$

This theoretical portion of this project is something that I am actively working on. I am currently working on my proficiency with Information and Conditioning. After that I will cover Brownian motion, then Itô's Integral. But, for now, I have a good enough understanding to code the formulas and produce some actual results.

Code Analysis: Black-Scholes-Merton Formulas.

The goal of this project is to implement the Black-Scholes-Merton formulas for various options. In our list of options, we have a forward, call, put, digital-call, digital-put, and a zero-coupon bond. These formulas are found in the header file BlackScholesFormulas.h given by

```
1  //BlackScholesFormulas.h
2  //Black-Scholes formulas header file
3  //includes forward, call, put, digital-call, digital-put, gamma, and zero-coupon bond.
4  #ifndef BLACK_SCHOLES_FORMULAS_H
5  #define BLACK_SCHOLES_FORMULAS_H
6
7  double BlackScholesForward( double spot,
8                              double strike,
9                              double compoundingRate,
10                             double dividendRate,
11                             double maturity );
```

```
12
13 double BlackScholesCall( double spot,
14                    double strike,
15                    double compoundingRate,
16                    double dividendRate,
17                    double volatility,
18                    double maturity );
19
20 double BlackScholesPut( double spot,
21                   double strike,
22                   double compoundingRate,
23                   double dividendRate,
24                   double volatility,
25                   double maturity );
26
27 double BlackScholesDigitalCall( double spot,
28                         double strike,
29                         double compoundingRate,
30                         double dividendRate,
31                         double volatility,
32                         double maturity );
33
34 double BlackScholesDigitalPut( double spot,
35                         double strike,
36                         double compoundingRate,
37                         double dividendRate,
38                         double volatility,
39                         double maturity );
40
41 double BlackScholesGamma( double spot,
42                     double strike,
43                     double compoundingRate,
44                     double dividendRate,
45                     double volatility,
46                     double maturity );
47
48 double BlackScholesZeroCouponBond( double compoundingRate,
49                           double maturity );
50
51 #endif
```

This implementation of the formulas will be discussed in this section. We present the entire BlackScholesFormulas.cpp file and then proceed to examine the code for each function implemented.

```
1  //BlackScholesFormulas.cpp
2  //Black-Scholes formulas cpp file
3  //includes implementation for forward, call, put, digital-call, digital-put, gamma, and zero-coupon bond.
4  #include "BlackScholesFormulas.h"
5  #include "Normals.h"
6  #include <cmath>
7
8  #if !defined(_MSC_VER)
9  using namespace std;
10 #endif
11
12 double BlackScholesForward(double spot, double strike, double compoundingRate, double dividendRate, ↩
       double maturity)
13 {
14    return exp(-compoundingRate*maturity)*(spot*exp((compoundingRate - dividendRate)*maturity) - strike);
15 }
16
17 double BlackScholesCall(double spot, double strike, double compoundingRate, double dividendRate, double ↩
       volatility, double maturity)
18 {
19    double standardDeviation = volatility*sqrt(maturity);
20    double moneyness = log(spot / strike);
21    double d1 = (moneyness + (compoundingRate - dividendRate)*maturity + 0.5*standardDeviation*↩
          standardDeviation) / standardDeviation;
22    double d2 = d1 - standardDeviation;
23    return spot*exp(-dividendRate*maturity)*CumulativeNormal(d1) - strike*exp(-compoundingRate*maturity)*↩
          CumulativeNormal(d2);
24 }
25
26 double BlackScholesPut(double spot, double strike, double compoundingRate, double dividendRate, double ↩
       volatility, double maturity)
```

```
{
    double standardDeviation = volatility*sqrt(maturity);
    double moneyness = log(spot / strike);
    double d1 = (moneyness + (compoundingRate - dividendRate)*maturity + 0.5*standardDeviation*standardDeviation) / standardDeviation;
    double d2 = d1 - standardDeviation;
    return strike*exp(-compoundingRate*maturity)*(1.0 - CumulativeNormal(d2)) - spot*exp(-dividendRate*maturity)*(1.0 - CumulativeNormal(d1));
}

double BlackScholesDigitalCall(double spot, double strike, double compoundingRate, double dividendRate, double volatility, double maturity)
{
    double standardDeviation = volatility*sqrt(maturity);
    double moneyness = log(spot / strike);
    double d2 = (moneyness + (compoundingRate - dividendRate)*maturity - 0.5*standardDeviation*standardDeviation) / standardDeviation;
    return exp(-compoundingRate*maturity)*CumulativeNormal(d2);
}

double BlackScholesDigitalPut(double spot, double strike, double compoundingRate, double dividendRate, double volatility, double maturity)
{
    double standardDeviation = volatility*sqrt(maturity);
    double moneyness = log(spot / strike);
    double d2 = (moneyness + (compoundingRate - dividendRate)*maturity - 0.5*standardDeviation*standardDeviation) / standardDeviation;
    return exp(-compoundingRate*maturity)*(1.0 - CumulativeNormal(d2));
}

double BlackScholesGamma(double spot, double strike, double compoundingRate, double dividendRate, double volatility, double maturity)
{
    double standardDeviation = volatility*sqrt(maturity);
    double moneyness = log(spot / strike);
    double d1 = (moneyness + (compoundingRate - dividendRate)*maturity + 0.5*standardDeviation*standardDeviation) / standardDeviation;
    return (exp(-(d1*d1) / 2) / 0.398942280401433) / (spot*standardDeviation);
}

double BlackScholesZeroCouponBond(double compoundingRate, double maturity)
{
    return exp(-compoundingRate*maturity);
}
```

Forward Price: BlackScholesForward(S, K, r, d, T)

The forward price is a function of the spot S, the strike K, continuously compounding rate r, dividend rate d, and the time-to-maturity, T. We use a fundamental theorem from mathematical finance, which states that if a liquid asset trades today at S_0 with a dividend rate d and the continuously compounding rate r then a forward contract to buy the asset for K with maturity T is worth

$$e^{-rT}\left(e^{(r-d)T}S_0 - K\right)$$

In particular, the contract will have zero value if $K = e^{(r-d)T}S_0$. This is a fact that we will keep in mind when conducting our consistency checks. The forward price is evaluated using the following code:

```
double BlackScholesForward(double spot, double strike, double compoundingRate,double dividendRate, double maturity)
{
    return exp(-compoundingRate*maturity)*(spot*exp((compoundingRate - dividendRate)*maturity) - strike);
}
```

Call Option Price: BlackScholesCall(S, K, r, d, σ, T)

The call option price is a function of the spot S, the strike K, continuously compounding rate r, dividend rate d, volatility σ and the time-to-maturity, T. The solution to the Black-Scholes-Merton equation for a call option is

$$C(S, K, r, d, \sigma, T) = SN(d_1) - Ke^{-r(T-t)}N(d_2)$$

where

$$d_1 = \frac{\log\left(\frac{S}{K}\right) + (r - d + \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}}$$

and

$$d_2 = \frac{\log\left(\frac{S}{K}\right) + (r - d - \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}}$$

where $N(x)$ is the cumulative normal function,

$$N(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{x^2}{2}} ds$$

This price of a call option is evaluated using the following code:

```
double BlackScholesCall(double spot, double strike, double compoundingRate, double dividendRate, double volatility, double maturity)
{
   double standardDeviation = volatility*sqrt(maturity);
   double moneyness = log(spot / strike);
   double d1 = (moneyness + (compoundingRate - dividendRate)*maturity + 0.5*standardDeviation*standardDeviation) / standardDeviation;
   double d2 = d1 - standardDeviation;
   return spot*exp(-dividendRate*maturity)*CumulativeNormal(d1) - strike*exp(-compoundingRate*maturity)*CumulativeNormal(d2);
}
```

where we have taken $t = 0$ and defined the standard deviation to be the product of the volatility and the square root of the expiry. This of course comes from the fact that the variance is $\sigma^2 T$.

The cumulative normal function is found in the implementation of Normals.h, appropriately titled Normals.cpp. Here is also found the normal density, as well as the inverse cumulative normal function obtained by the Moro algorithm:

```
#include <cmath>
#include "Normals.h"

#if !defined(_MSC_VER)
using namespace std;
#endif

const double Normalizer = 0.398942280401433; //Normalizer = \frac{1}{\sqrt{2\pi}}

double NormalDensity(double x) //Probability denisty for a standard Gaussian distribution
{
   return Normalizer*exp(-x*x / 2);
}

double InverseCumulativeNormal(double u) //The inverse cumulative normal function via the Beasley-Springer/Moro Approximation
{
   static double a[4] = { 2.50662823884, -18.61500062529, 41.39119773534, -25.44106049637 };
   static double b[4] = { -8.47351093090, 23.08336743743, -21.06224101826, 3.13082909833 };
   static double c[9] = { 0.3374754822726147, 0.9761690190917186, 0.1607979714918209, 0.0276438810333863,
                          0.0038405729373609, 0.0003951896511919, 0.0000321767881768, 0.0000002888167364,
                          0.0000003960315187 };
   double x = u - 0.5;
   double r;

   if (fabs(x) < 0.42) //Beasley-Springer
   {
      double y = x*x;
      r = x*(((a[3]*y + a[2])*y + a[1])*y + a[0]) / ((((b[3]*y + b[2])*y + b[1])*y + b[0])*y + 1.0);
   }

   else //Moro
   {
      r = u;

      if (x > 0.0)
      {
         r = 1.0 - u;
      }

```

```
      r = log(-log(r));
      r = c[0] + r*(c[1] + r*(c[2] + r*(c[3] + r*(c[4] + r*(c[5] + r*(c[6] + r*(c[7] + r*c[8])))))));

      if (x < 0.0)
      {
         r = -r;
      }
   }

   return r;
}

double CumulativeNormal(double x)
{
   static double a[5] = { 0.319381530, -0.356563782, 1.781477937, -1.821255978, 1.330274429 };
   double result;
   if (x < -7.0)
   {
      result = NormalDensity(x) / sqrt(1.0 + x*x);
   }

   else
   {
      if (x > 7.0)
      {
         result = 1.0 - CumulativeNormal(-x);
      }
      else
      {
         double temp = 1.0 / (1.0 + 0.2316419*fabs(x));

         result = 1 - NormalDensity(x)* (temp*(a[0] + temp*(a[1] + temp*(a[2] + temp*(a[3] + temp*a[4]))))
             ));

         if (x <= 0.0)
            result = 1.0 - result;
      }
   }
   return result;
}
```

Put Option Price: BlackScholesPut(S, K, r, d, σ, T)

The put option price is a function of the spot S, the strike K, continuously compounding rate r, dividend rate d, volatility σ and the time-to-maturity, T.. The solution to the Black-Scholes-Merton equation for a put option is

$$P(S, K, r, d, \sigma, T) = Ke^{-r(T-t)}(1 - N(d_2)) - Se^{dT}(1 - N(d_1))$$

where

$$d_1 = \frac{\log\left(\frac{S}{K}\right) + (r - d + \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}}$$

and

$$d_2 = \frac{\log\left(\frac{S}{K}\right) + (r - d - \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}}$$

Note that d_1 and d_2 stay the same. The price of a put option is evaluated using the following code:

```
double BlackScholesPut(double spot, double strike, double compoundingRate, double dividendRate, double
    volatility, double maturity)
{
   double standardDeviation = volatility*sqrt(maturity);
   double moneyness = log(spot / strike);
   double d1 = (moneyness + (compoundingRate - dividendRate)*maturity + 0.5*standardDeviation*
       standardDeviation) / standardDeviation;
   double d2 = d1 - standardDeviation;
   return strike*exp(-compoundingRate*maturity)*(1.0 - CumulativeNormal(d2)) - spot*exp(-dividendRate*
       maturity)*(1.0 - CumulativeNormal(d1));
}
```

Digital-Call Option Price: BlackScholesDigitalCall(S, K, r, d, σ, T)

The digital-call option price is a function of the spot S, the strike K, continuously compounding rate r, dividend rate d, volatility σ and the time-to-maturity, T. The solution to the Black-Scholes-Merton equation for a digital-call option is

$$DC(S, K, r, d, \sigma, T) = e^{-r(T-t)}N(d_2)$$

where

$$d_2 = \frac{\log\left(\frac{S}{K}\right) + (r - d - \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}}$$

This price of a digital-call option is reflected in the following code:

```
double BlackScholesDigitalCall(double spot, double strike, double compoundingRate, double dividendRate, double volatility, double maturity)
{
   double standardDeviation = volatility*sqrt(maturity);
   double moneyness = log(spot / strike);
   double d2 = (moneyness + (compoundingRate - dividendRate)*maturity - 0.5*standardDeviation*standardDeviation) / standardDeviation;
   return exp(-compoundingRate*maturity)*CumulativeNormal(d2);
}
```

Digital-Put Option Price: BlackScholesDigitalPut(S, K, r, d, σ, T)

The digital-put option price is a function of the spot S, the strike K, continuously compounding rate r, dividend rate d, volatility σ and the time-to-maturity, T. The solution to the Black-Scholes-Merton equation for a digital-put option is

$$DC(S, K, r, d, \sigma, T) = e^{-r(T-t)}(1 - N(d_2))$$

where

$$d_2 = \frac{\log\left(\frac{S}{K}\right) + (r - d - \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}}$$

This price of a digital-put option is evaluated by the following code:

```
double BlackScholesDigitalPut(double spot, double strike, double compoundingRate, double dividendRate, double volatility, double maturity)
{
   double standardDeviation = volatility*sqrt(maturity);
   double moneyness = log(spot / strike);
   double d2 = (moneyness + (compoundingRate - dividendRate)*maturity - 0.5*standardDeviation*standardDeviation) / standardDeviation;
   return exp(-compoundingRate*maturity)*(1.0 - CumulativeNormal(d2));
}
```

Zero-Coupon Bond Price: BlackScholesZeroCouponBond(r, T)

The digital-call option price is a function of the continuously compounding rate r, and the time-to-maturity T, denoted expiry in the code. The equation for the price of zero-coupon bond is

$$B(r, T) = e^{-r(T-t)}$$

The price of a zero-coupon bond is evaluated using the following code:

```
double BlackScholesZeroCouponBond(double compoundingRate, double maturity)
{
   return exp(-compoundingRate*maturity);
}
```

Consistency Checks.

In order to check that formulas have been implemented correctly, we run the following consistency checks with values $S = 100,\ K = 110,\ r = 0.05,\ d = 0,\ \sigma = 0.99,$ and $T = 1$:

(i) Satisfaction of the Put-Call Parity.

The put-call parity is the relationship between a call, put, and forward. It is the statement that the payoff of a forward contract agrees with the payoff of a portfolio that is long one call and short one put. Since the value at expiry of the forward agree with the value of the portfolio that is long one call and short one put, these values must agree at all previous times $0 \leq t \leq T$ by the Generalized Law of One Price,

$$F(t) = C(t) - P(t)$$

This is verified in main.cpp:

```
#include<iostream>
#include "Normals.h"
#include "BlackScholesFormulas.h"
using namespace std;

int main()
{
const int cap = 10000;
   const double spot = 100, strike = 110, ra = 0.05, d = 0, vol = 0.99, expiry = 1;
   double call = BlackScholesCall(spot, strike, ra, d, vol, expiry);
   double put = BlackScholesPut(spot, strike, ra, d, vol, expiry);
   double forward = BlackScholesForward(spot, strike, ra, d, expiry);
   double digital_call = BlackScholesDigitalCall(spot, strike, ra, d, vol, expiry);
   double digital_put = BlackScholesDigitalPut(spot, strike, ra, d, vol, expiry);
   double zero_coupon_bond = BlackScholesZeroCouponBond(ra, expiry);
   cout << "List of parameters: " << endl;
   cout << "------------------- " << endl;
   cout << "Spot price = $" << spot << endl;
   cout << "Strike = $" << strike << endl;
   cout << "Continuously compounding rate = " << ra << endl;
   cout << "Dividend rate = " << d << endl;
   cout << "Time to maturity = " << expiry << endl;
   cout << "Volatility = " << vol << endl;
   cout << "------------------- " << endl;
   cout << "Price of call option: " << call << endl;
   cout << "Price of put option: " << put << endl;
   cout << "Value of forward: " << forward << endl;
   cout << "Put-Call Parity: " << call << " - " << put << " = " << forward << endl;
   return 0;
}
```

The output is

```
List of parameters:
-------------------
Spot price = $100
Strike = $110
Continuously compounding rate = 0.05
Dividend rate = 0
Time to maturity = 1
Volatility = 0.99
-------------------
Price of call option: 36.5391
Price of put option: 41.1743
Value of forward: -4.63524
Put-Call Parity: 36.5391 - 41.1743 = -4.63524
```

Thus, we have satisfied the put-call parity.

(ii) C(S, K, r, d, σ, T) is Monotone Decreasing With Strike.

We add the following for loop which iterates through increasing values of strike by a magnitude of 10 up to 10^4

```
cout << "Price of a call option should be monotone decreasing with strike." << endl;
   cout << "Test: ";
   for (int i = 1; i < cap; i *= 10) {
      cout << BlackScholesCall(spot, i, ra, d, vol, expiry);
      if (i * 10 < cap)
```

```
            cout << " > ";
        else
            cout << "." << endl;
    }
```

and get an output of

```
Price of a call option should be monotone decreasing with strike.
Test: 99.0488 > 90.5683 > 39.4968 > 1.08762.
```

Which satisfies the consistency check.

(iii) S - Ke^{-rT} $\leq$ C(S, K, r, d, σ, T) $\leq$ S.

More formally, the price of a call should be bounded from below and above by $S - Ke^{-rT} = -4.63524$ and $S = 100$ respectively. This check is satisfied since $C = 36.541$.

(iv) C(S, K, r, d, σ, T) is Monotone Increasing in Volatility.

We add the following for loop which iterates through increasing values of volatility by a magnitude of 10 up to 10^4

```
cout << "A call option should be monotone increasing in volatility." << endl;
    cout << "Test: ";
    for (int i = 1; i < cap; i *= 10) {
        cout << BlackScholesCall(spot, strike, ra, d, i*0.001, expiry);
        if (i * 10 < cap)
            cout << " < ";
        else
            cout << "." << endl;
    }
```

and get an output of

```
A call option should be monotone increasing in volatility.
Test: 0 < 6.07001e-07 < 2.17395 < 36.8993.
```

Which satisfies the consistency check.

(v) If d = 0, C(S, K, r, d, σ, T) Should Be Increasing With T.

We add the following for loop which iterates through increasing values of volatility by a magnitude of 10 up to 10^4

```
cout << "If d = 0, the call option price should be increasing with T." << endl;
    cout << "Test: ";
    for (int i = 1; i < cap; i *= 5) {
        cout << BlackScholesCall(spot, strike, ra, 0, vol, i);
        if (i * 10 < cap)
            cout << " < ";
        else
            cout << "." << endl;
    }
```

and get an output of

```
If d = 0, the call option price should be increasing with T.
Test: 36.5391 < 90.4556 < 100 < 100.
```

Which satisfies the consistency check.

(vi) C(S, K, r, d, σ, T) Should Be a Convex Function of Spot.

This is equivalent to verifying that second derivative of the Black-Scholes price of a call option with respect to spot is strictly positive. For this we consider the Gamma. The Gamma is the derivative of the Delta with respect to spot. The Delta

is the most fundamental Greek. The Delta of a call option is given by

$$\Delta = \frac{\partial C}{\partial S}(S,t) = N(d_1)$$

where

$$d_1 = \frac{\log\left(\frac{S}{K}\right) + (r - d + \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}}$$

Using this, we obtain the Gamma by the formula

$$\Gamma = \frac{\partial^2 C}{\partial S^2}(S,t) = \frac{N'(d_1)}{S\sigma\sqrt{T-t}}$$

where $N'(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$. This is implemented in BlackScholesFormulas.cpp as

```
double BlackScholesGamma(double spot, double strike, double compoundingRate, double dividendRate, double volatility, double maturity)
{
   double standardDeviation = volatility*sqrt(maturity);
   double moneyness = log(spot / strike);
   double d1 = (moneyness + (compoundingRate - dividendRate)*maturity + 0.5*standardDeviation*standardDeviation) / standardDeviation;
   return (exp(-(d1*d1) / 2) / 0.398942280401433) / (spot*standardDeviation);
}
```

where 0.398942280401433 is just $\frac{1}{\sqrt{2\pi}}$. We add to main.cpp

```
cout << "The call option should be a convex function of spot (Gamma > 0)." << endl;
   cout << "Test: " << BlackScholesGamma(spot, strike, ra, d, vol, expiry) << " > 0" << endl;
```

which outputs

```
The call option should be a convex function of spot (Gamma > 0).
Test: 0.0228893 > 0
```

Since the second derivative of C with respect to spot is strictly positive, it must be convex.

(vii) The Price of a Digital-Call Option Plus a Digital-Put Option Is Equal to the Price of a Zero-Coupon Bond.

We add the following code

```
cout << "The price of a digital-call option plus a digital-put option is equal to the price of a zero-coupon bond." << endl;
   cout << "Test: " << digital_call << " + " << digital_put << " = " << digital_call + digital_put << " = " << zero_coupon_bond << endl;
```

which outputs

```
The price of a digital-call option plus a digital-put option is equal to the price of a zero-coupon bond.
Test: 0.279979 + 0.67125 = 0.951229 = 0.951229
```

which satisfies the consistency check.