C++ CLASSES TO MODEL FINANCIAL INSTRUMENTS

STICLARU, Gabriel[1]
Faculty of Mathematics and Informatics,
Spiru Haret University

Abstract

Our purpose is to use C++ to model financial instruments: Bonds, Stocks, Forwards, Futures, Options and Swaps. This paper concentrates on Options and present an hierarchy of classes to model European Options. Based on Black Scholes extended mathematical model, we calculate the price for Call and Put Option and some measures for sensitivities of the price: delta, gamma, vega, theta and rho.

Key-words: *financial instruments, European Options, Black and Scholes model, C++ classes.*

Computing Reviews classification: D.2.3, J.1, J.4

1. **Introduction**

Most books that discuss financial instruments provide only a theoretical understanding of them as well as their mathematical derivations. While many papers provide numerical results, few supply the details for how to implement the model in real life.

In quantitative finance, most numerical work is done with Excel, MatLab, Mathematica, C++, Visual Basic or Java, but C++ is dominant. We choose C++ for implementing financial models due to the language's object-oriented features, speed, and reusability.

2. **What are Options?**

In this section we will present some result necessary for the rest of the paper. For a complete introduction to options see Hull (2002) and Rubinstein (1999).

An option contract gives the owner of the contract the right, but not the obligation, to buy (Call Option) or sell (Put Option) a security at a specified future date (expiry date or maturity) for a fixed agreed price (strike price). The buyer of

[1] Master student paper presented at the *Conference on Applied Mathematics and Informatics*, 14 December 2007, Faculty of Mathematics and Informatics, *Spiru Haret* University, Bucharest, Romania.

the contract must pay the seller a fee, which is called the option price or *premium*. After the option's expiration date (T = maturity), the contract will cease to exist.

The price of the option is the *premium*. When the option is traded on an organized market, the premium is quoted by the market. Otherwise, the problem is to price the option. Also, even if the option is traded on an organized market, it can be interesting to detect some possible abnormalities in the market.

Options are in many varieties. European Options can only be exercised at maturity T, American Options at any time before T. Option which are only exercisable during a predefined portion of there life of the contract are termed Bermudan.

The following six factors influence the option price (Put or Call):

- S = Spot price of the underlying asset (stock, bond, a currency and so on);
- K = Strike price (the price at which the transaction is done, if the option is exercised);
- T = Time to expiration of the option;
- σ = Expected price volatility of the underlying over the life of the option;
- r = risk-free rate of interest;
- q or f = dividends or foreign currency.

There are two main problems about option: option pricing (evaluation of Premium) and how sensitive the price of an option is to a change in any one of the factors that affect its price.

From the definition of the options, it is clear that at their last possible exercise date, the maturity date, the buyer has cash flows: max (S-K, 0) for Call and max (K-S, 0) for Put.

The question is what formula to use prior to expiration?

Several models have been developed to determine the theoretical value of an option. The most popular one was developed by Fischer Black and Myron Scholes (1973) for valuing European call options.

The classical Black-Scholes model could only be used to value European options (Put or Call) on underlying instruments, which do not generate a cash flow, non-dividend paying stocks. The assumptions used for original Black-Scholes model are:

- the asset price follows the lognormal distribution;
- the underlying asset pays no dividends during the life of the option;
- there are no arbitrage possibilities;
- transactions cost and taxes are zero;
- the risk-free interest rate and the asset volatility are known functions of time over the life of the option;
- there are no penalties for short sales of stock;
- the market operates continuously and the share prices follow a continuous Ito process.

Black-Sholes model was modified by Merton (1973) to work on stock indices with known dividend q, Garman and Kohlhagen (1983) for option on foreign currency f , and Black (1976) for option on future.

The classical Black-Scholes model uses five parameters for option pricing S, K, T, σ, r, but we expand the model with a new parameter b = cost of carry.

3. **Black-Scholes extension model**

We introduce the generalized Black–Scholes formula to calculate the price of a call or put option on some underlying asset. In general, option price is a function f(S, K, T, r, σ, b). The exact formula for C = Call or P = Put is given by (1)-(4):

$$C = S\,e^{(b-r)T}N(d_1) - K\,e^{-rT}N(d_2) \tag{1}$$

$$P = Ke^{-rT}N(-d_2) - S\,e^{(b-r)T}N(-d_1) \tag{2}$$

where

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

and

$$d_2 = \frac{\ln(S/K) + (b-\sigma^2/2)T}{\sigma\sqrt{T}} = d_1 - \sigma\sqrt{T}\ . \tag{4}$$

N(.) = the cumulative normal distribution

We can do as particular to obtain a specific model. When:
B = r Black-Scholes (1973) Stock Option Model;
B = r - q Merton (1973) Stock Option Model – continuous dividend yield q;
B = 0.0 Black (1976) Futures Option Model;
B = r – f Garman-Kohlhagen (1983) Currency Option Model.
To view how sensitive is the price of an option to a change in any one of the factors that affect its price, we can calculate:

- delta – Sensitivity to change in the underlying spot price;
- gamma – option delta's sensitivity to market price changes;
- theta – Sensitivity to time to maturity;
- vega – Change in value for a change in the underlying's volatility;
- rho – Change in value for a change in interest rates.

4. **C++ classes to model European Options**

In this section we wish to describe how we have applied this body of knowledge to produce a software system that is able to price European Option.

We start by creating an Abstract Base Class from which specific instruments will be derived: Bonds, Stocks, Forwards, Futures, Options, Swaps an so on. Classes must be related in a hierarchy that moves from general to the specific:

```
class FinancialInstrument
{ // Abstract Base Class for all financial instruments
};
```

Because many financial models use statistical distributions, we can make a library with all the necessary functions. Here we use namespace to group together two Gaussian functions and a record for parameters:

```
using namespace std;
namespace library
{
 struct      OptionParameters
 {
 double S  ;            // Price for underlying asset
 double K;              // Strike price
 double T  ;            // Time to maturity (in fractional years)
 double r;              // Interest rate
 double sigma;          // Volatility
 double b;              // Cost of carry
 string optionType;     // Option name (Call, Put)
 string UName;          // Underlying asset name
 };
 double  n(const double&  x)
 { // The normal distribution function
  return 0.398942280401433*exp(-0.5*x*x);
 }
 double  N(const double&  x)
 { // The approximation to the cumulative normal distribution
  //  see  Abramowitch, M. and L.A. Stegun (1972)
 double a1=0.4361836, a2=-0.1201676, a3=0.9372980;
 double k=1.0/(1.0 + (0.33267 * x));
 if (x >=0.0)
  return 1.0 - n(x)* (a1*k + (a2*k*k) + (a3*k*k*k));
  else  return 1.0 - N(-x);
 }
};
```

We can replace the parameter of Gaussian functions by double x, but for efficiency, argument x is given as constant reference.

Option is implemented as a new class deriving from FinancialInstrument. The data in the base class is declared as protected because we wish to be able to access this data in derived classes (EuropeanOption, AmericanOption and so on).

Having defined the object's data we must extract information from the object. Because it is possible for a single instrument to be priced in different ways, we make virtual Price() and Payoff(). To set or to extract protected data in the child class, we have created two virtual function Set() and Show().

```
using namespace  library;
class   Option:  public  FinancialInstrument
{// Abstract Base Class for all options (European, American, …)
protected:
  double S, K, T, r, sigma, b;
  string optionType, UName;
public:
```

```
  virtual double Price() const=0;              // option price
  virtual double PayOff() const=0; // payments balance
  virtual OptionParameters Show() const=0;
  virtual void Set(const OptionParameters& p)=0;
};
```

A derived class EuropeanOption, add functionality beyond that of the base class, and override methods of its base class. Price() implements the analytic Black-Scholes formula for European options and with method switchType() we can change option type. Also, we added methods to calculate parameters for sensitivies: Delta(), Gamma(), Vega, Theta(), Rho().

```
class EuropeanOption: public Option
{
public:
  EuropeanOption();     // constructor, default
  EuropeanOption(const EuropeanOption& o);  // Copy constructor
  void copy(const EuropeanOption& o);
  virtual ~EuropeanOption();
  EuropeanOption& operator = (const EuropeanOption& o);
  double Price () const;
  double PayOff() const;
  void switchType();  // Change "Call" to "Put" and revers
  void Set(const OptionParameters& p) ;    // modifiers
  OptionParameters Show() const;
  // Functions that calculate sensitivities
  double Delta() const;
  double Gamma() const;
  double Vega () const;
  double Theta() const;
  double Rho() const;
};
```

The necessary code to implement these classes is presented below.

```
EuropeanOption::EuropeanOption()
{// Initialise default values (original Black-Scholes)
  r=0.08;     sigma= 0.30;       K=65.0;
  T=0.25;    S=60.0;                    b=r;
  optionType = "Call";          UName = "stock";
 }
void EuropeanOption::copy(const EuropeanOption& o)
{
  r      =o.r;  sigma=o.sigma;  K=o.K;  T=o.T;  S=o.S;  b=o.b;
  optionType=o.optionType;  UName=o.UName;
}
EuropeanOption::EuropeanOption(const EuropeanOption& o)
{  copy(o); }
EuropeanOption& EuropeanOption::operator=(const EuropeanOption& o)
{
  if (this==&o) return *this;
  copy(o);
  return *this;
```

```
}
OptionParameters EuropeanOption::Show() const
{
OptionParameters p;
p.S=S;  p.K=K;  p.T=T;  p.r=r;  p.sigma=sigma;  p.b=b;
p.optionType=optionType;  p.UName = UName;
return p;
}
void EuropeanOption::Set(const OptionParameters& p)
{
S=p.S;   K=p.K;  T=p.T;  r=p.r;  sigma=p.sigma;  b = p.b;
optionType=p.optionType;  UName=p.UName;
}
EuropeanOption::~EuropeanOption(){}
double EuropeanOption::Price() const
{
 double d1, d2, aux, price=0.0;
 aux=sigma * sqrt(T);
 d1 = (log(S / K) + (b + (sigma*sigma)* 0.5) * T)/aux;
 d2 = d1 - aux;
 if ( optionType == "Call" )
   price = S * exp((b - r) * T) * N(d1) - K * exp(-r * T) * N(d2);
 else if ( optionType == "Put" )
   price = K*exp(-r * T)*N(-d2) - S*exp((b - r)*T)*N(-d1);
 return price;
}

double EuropeanOption::PayOff() const
{
 double payoff=0.0;
 if (( optionType=="Call" ) && (S >K))  payoff=S - K;
 if (( optionType=="Put" )  &&  (K>S))  payoff=K - S;
 return payoff;
}
double EuropeanOption::Delta() const
{
 double d1, aux, delta=0.0;
 aux = sigma * sqrt(T);
 d1 = ( log(S/K) + (b+ (sigma*sigma)*0.5 ) * T )/ aux;
 if ( optionType == "Call" ) delta = exp((b-r)*T) * N(d1);
   else if ( optionType == "Put" )  delta = exp((b-r)*T) * (N(d1) - 1.0);
 return delta;
}
double EuropeanOption::Gamma() const
{
 double d1, aux;
 aux = sigma * sqrt(T);
 d1 = ( log(S/K) + (b + (sigma*sigma)*0.5 ) * T )/ aux;
```

```
 return ( n(d1) * exp((b-r)*T) ) / (S * aux);
}

double EuropeanOption::Vega() const
{
 double d1, aux=sigma*sqrt(T);
 d1 = ( log(S/K) + (b + (sigma*sigma)*0.5 ) * T )/ aux;
 return (S * exp((b-r)*T) * n(d1) * sqrt(T) );
}

double EuropeanOption::Theta() const
{
 double d1, d2, t1, t2, t3, theta=0.0, aux=sigma*sqrt(T);
 d1 = ( log(S/K) + (b + (sigma*sigma)*0.5 ) * T )/ aux;
 d2 = d1 - aux;
 t1 =  (S * exp((b-r)*T ) * n(d1) * sigma * 0.5 )/ sqrt(T);
 t2 =  (b-r)*(S * exp((b-r)*T) * N(d1));
 t3 =  r * K * exp(-r * T) * N(d2);
 if ( optionType == "Call" )  theta = -(t1 + t2 + t3);
 t2 =  (b-r)*(S * exp((b-r)*T) * N(-d1));
 t3 =  r * K * exp(-r * T) * N(-d2);
 if ( optionType == "Put" )  theta =  t2 + t3 - t1;
 return theta;
}
double EuropeanOption::Rho() const
{
 double d1, d2, t1, t2, t3, rho=0.0, aux=sigma*sqrt(T);
 d1 = ( log(S/K) + (b + (sigma*sigma)*0.5 ) * T )/ aux;
 d2 = d1 - aux;
 if ( optionType == "Call"  && b != 0.0 )
 rho = T * K * exp(-r * T) * N(d2);
 if (( optionType == "Put" ) && (b != 0.0))
 rho = - T * K * exp(-r * T) * N(-d2);
 if (b == 0.0)  rho = - T * Price();
 return rho;
}
void EuropeanOption::switchType()
{
 if (optionType == "Call") optionType = "Put";
   else optionType = "Call";
}
```

5. **Summaries and conclusion**

One of the two objectives of this paper was to present the theory of options and how could be priced.

The second objective of this work was to model European Option with C++ classes. The source code was compiled with Borland C++ Builder and it is entirely presented here, in order to allow readers to replicate and validate these results.

To illustrate the applicability of our design, the results are shown below. We take the following example (original Black-Scholes model):

S = 60.0	K = 65.0	T = 0.25(three months)
R = 0.08	σ = 0.30	b = r = 0.08
optionType = “Call”		UName = “stock”

The outputs of the software are:
Call price = 2.13 and for sensitivities of the price:

Delta = 0.37;
Gamma = 0.04;
Vega = 11.35;
Theta=-8.43;
Rho=5.05.

REFERENCES

1. Hull, J. (2002), *Options, Futures and other Derivatives*, Prentice-Hall, Fifth edition
2. Rubinstein, M. (1999), *Rubinstein on derivatives*, Risk Books
3. Black, F. and Scholes, M. S. (1973), *The Pricing of Options and Corporate Liabilities*, J. Political Economy
4. Garman, M. B. and Kohlhagen, S. W. (1983), *Foreign Currency Option Values*, J. International Money and Finance 2
5. Wilmott, P., Howison, S., Dewynne, J. (1995), *The mathematics of financial derivatives*
6. Abramowitch, M. and Stegun, L. A. (1972), *Handbook of Mathematical Functions*, Dover Publications, New York