Derivatives Pricing Course

Lecture 2 – Vanilla products

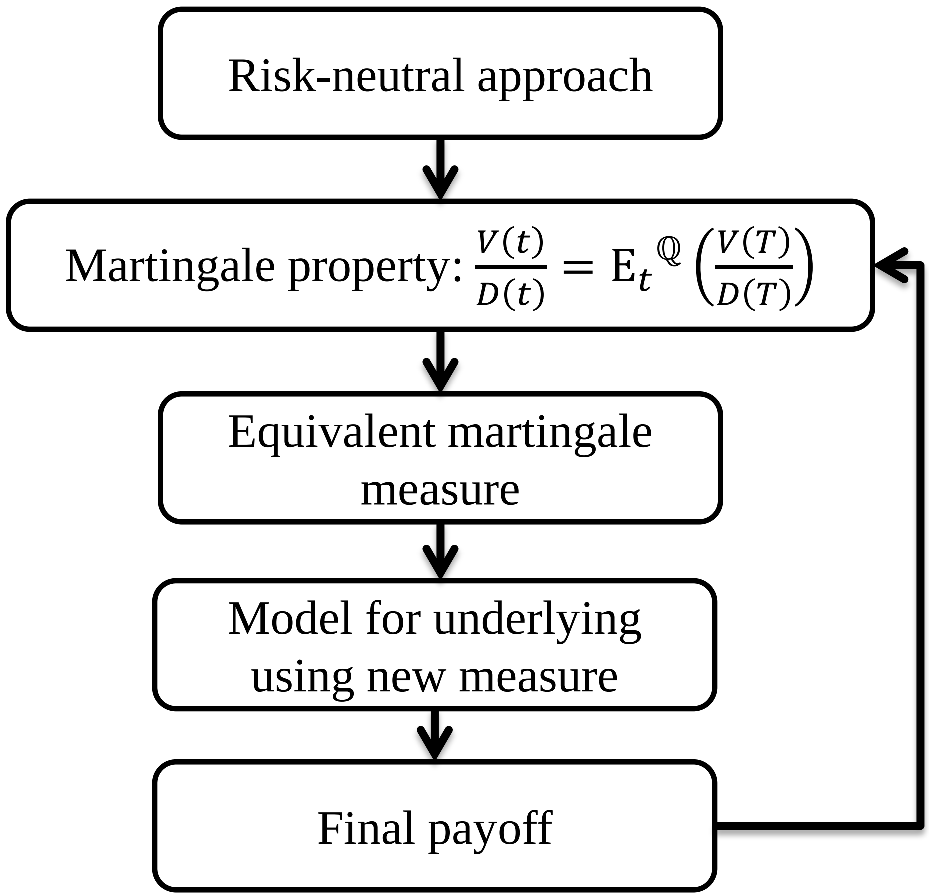
Artem Isaev

2016 CMF

Agenda

- Closed-form solution for BS vanilla call/put price derivation.
- Closed-form solution for BS vanilla call/put Greeks derivation.
- Bruteforce C++ implementation.
- Compare hardcoded formulas with QuantLib results.

Closed-form solutions
Vanilla option price



Closed-form solutions

Vanilla option price

- In the basic BSM economy, two assets are traded: a money market account β and a stock S - $X(t)$=($\beta(t), S(t))^T$.

- The dynamics for β:

$$d\beta(t)/\beta(t) = r\,dt, \beta(0) = 1$$

- The stock dynamics are assumed to satisfy GBMD:

$$dS(t)/S(t) = \boldsymbol{\mu}\,dt + \boldsymbol{\sigma}\,dW(t)$$

- Deflated stock price:

$$S^{\beta}(t) = S(t)/\beta(t)$$

- By Ito’s lemma:

$$dS^{\beta}(t)/S^{\beta}(t) = (\mu - r)\,dt + \sigma\,dW(t)$$

Closed-form solutions
Vanilla option price

- Applying Girsanov's theorem:

$$d\varsigma(t)/\varsigma(t) = -\theta\, dW(t), \qquad \theta = \frac{\mu - r}{\sigma}$$

- Under new measure $\mathbb{Q}$, $W^{\beta}(t) = W(t) + \theta t$ is a Brownian motion:

$$dS^{\beta}(t)/S^{\beta}(t) = \sigma\, dW^{\beta}(t)$$

$$dS(t)/S(t) = r dt + \sigma\, dW^{\beta}(t)$$

- Hence stock dynamics:

$$S(T) = S(t)e^{\left(r - \frac{1}{2}\sigma^2\right)(T-t) + \sigma\left(W^{\beta}(T) - W^{\beta}(t)\right)},\ t \in [0, T]$$

- Our final payoff depends on the final value of the underlying.
- Plug our stock dynamics into $E_t{}^{\mathbb{Q}}$ and get the price.

Closed-form solutions
Vanilla option price

- *Discount bond* – paying at time T \$1 for certain. Application of basic derivative pricing equation immediately gives:

$$P(t,T) = \beta(t)\, E_t^{\mathbb{Q}}\left(\frac{1}{\beta(T)}\right) = E_t^{\mathbb{Q}}\left(e^{-r(T-t)}\right) = e^{-r(T-t)}$$

- *European call option* – paying $c(T) = (S(T) - K)^+$:

$$c(t) = e^{-r(T-t)} E_t^{\mathbb{Q}}\left((S(T) - K)^+\right)$$

$$c(t) = P(t,T) \int_{-\infty}^{\infty} \left(S(t) e^{\left(r - \frac{1}{2}\sigma^2\right)(T-t) + z\sigma\sqrt{T-t}} - K\right)^+ \varphi(z) dz$$

Closed-form solutions

Vanilla option price

- Theorem 2.1 In the BS economy, the arbitrage-free time t price of a K-strike call option maturing at time T is

$$c(t) = S(t)N(d_1) - KP(t,T)N(d_2),$$

$$d_{1,2} \triangleq \frac{\ln(S(t)/K) + (r \pm \sigma^2/2)(T-t)}{\sigma\sqrt{T-t}}, t < T$$

where $N(\cdot)$ is the Gaussian cumulative distribution function

Closed-form solutions

The Greeks

- Lemma 2.2 In BS notation the following result holds:

$$SN'(d_1) = Ke^{-r(T-t)}N'(d_2)$$

Proof: recall that $d_2 = d_1 - \sigma\sqrt{T-t}$ and open brackets in the exponent.

- Greeks to derive:
  - Delta (Δ) – sensitivity of option price to underlying price
  - Gamma (Γ) – sensitivity of option delta to underlying price
  - Vega (υ) – sensitivity of option price to volatility

Closed-form solutions

The Greeks - Delta

- Initial formula

$$c(t) = S(t)N(d_1) - Ke^{-r(T-t)}N(d_2)$$

- First step

$$\Delta_{call} = \frac{\partial c}{\partial S} = N(d_1) + SN'(d_1)\frac{\partial d_1}{\partial S} - Ke^{-r(T-t)}N'(d_2)\frac{\partial d_2}{\partial S}$$

- (Lemma 2.2)

$$\Delta_{call} = N(d_1) + \left(\frac{\partial d_1}{\partial S} + \frac{\partial d_2}{\partial S}\right) \cdot 0 = N(d_1)$$

$$\boxed{\Delta_{call} = N(d_1)}$$

Closed-form solutions

The Greeks - Gamma

- Initial formula

$$c(t) = S(t)N(d_1) - Ke^{-r(T-t)}N(d_2)$$

- First step

$$\Gamma_{call} = \frac{\partial^2 c}{\partial S^2} = \frac{\partial \Delta_{call}}{\partial S} = N'(d_1)\frac{\partial d_1}{\partial S} = N'(d_1)\frac{K}{S\sigma\sqrt{T-t}}$$

$$\boxed{\Gamma_{call} = \frac{N'(d_1)}{S\sigma\sqrt{T-t}}}$$

Closed-form solutions

The Greeks - Vega

- Initial formula

$$c(t) = S(t)N(d_1) - Ke^{-r(T-t)}N(d_2)$$

- First step

$$\upsilon_{call} = \frac{\partial c}{\partial \sigma} = SN'(d_1)\frac{\partial d_1}{\partial \sigma} - Ke^{-r(T-t)}N'(d_2)\frac{\partial d_2}{\partial \sigma}$$

- (Lemma 2.2)

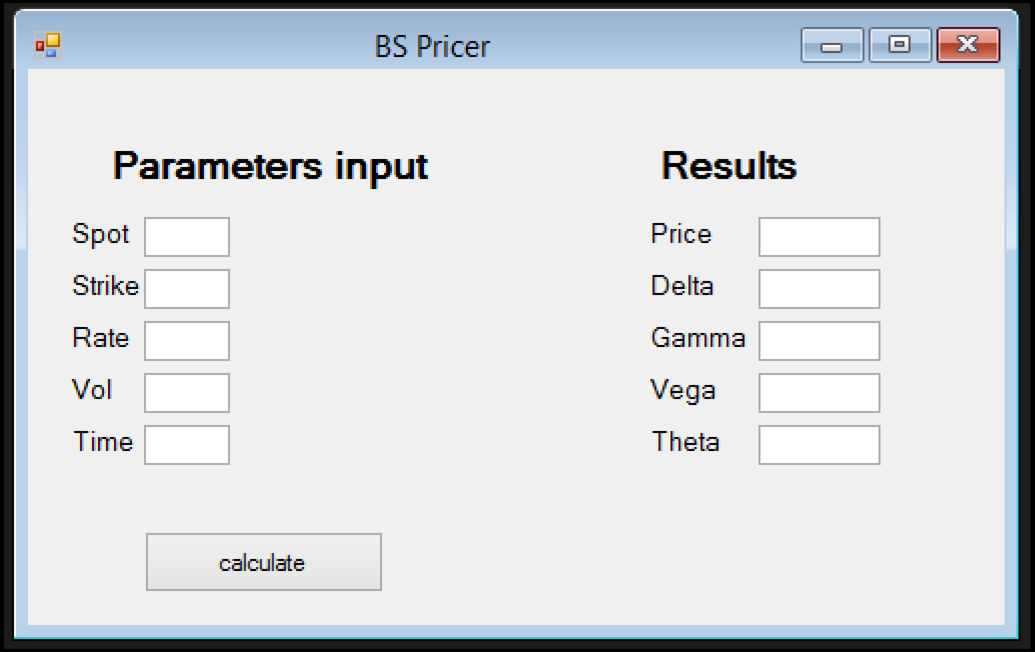
$$\upsilon_{call} = SN'(d_1)\left(\frac{\partial d_1}{\partial \sigma} - \frac{\partial d_2}{\partial \sigma}\right)$$

- Recall that $d_2 = d_1 - \sigma\sqrt{T-t}$, hence

$$\boxed{\upsilon_{call} = SN'(d_1)\sqrt{T-t}}$$

Basic C++ code

Bruteforce implementation



Basic C++ code

OptionClass(.h)

```
#ifndef OPTIONCLASS_H
#define OPTIONCLASS_H

//Abstract class for Option - defined interface, overload payoff function in inherited classes
class Option
{
public:
            Option(){};
            virtual double getPrice() const = 0;
            virtual double getDelta() const = 0;
            virtual double getGamma() const = 0;
            virtual double getVega() const = 0;
            virtual double getTheta() const = 0;
private:
};

class VanillaCall : public Option
{
public:
            VanillaCall(double, double, double, double, double);
            virtual double getPrice() const;
            virtual double getDelta() const;
            virtual double getGamma() const;
            virtual double getVega() const;
            virtual double getTheta() const;
private:
            double Spot;
            double Strike;
            double Rate; //in % annualized
            double Vol; //in % annualized
            double Time; //time to maturity in years
};
#endif
```

Basic C++ code
OptionClass(.cpp)

```
#include "OptionClass.h"
#include "Random.h"
#include <cmath>

//Function definition

VanillaCall::VanillaCall(double Spot_, double Strike_, double Rate_, double Vol_, double Time_) :
Spot(Spot_), Strike(Strike_),Time(Time_){
            Rate = Rate_ / 100.0;
            Vol = Vol_ / 100.0;
}

double VanillaCall::getPrice() const{

            double d1 = 0.0;
            double d2 = 0.0;
            double price = 0.0;

            d1 = (log(Spot / Strike) + (Rate + Vol * Vol / 2.0) * Time) / (Vol * sqrt(Time));
            d2 = d1 - Vol * sqrt(Time);

            price = Spot * CDF(d1) - Strike * exp(-Rate * Time) * CDF(d2);

            return price;
}

double VanillaCall::getDelta() const{

            double d1 = (log(Spot / Strike) + (Rate + Vol * Vol / 2.0) * Time) / (Vol * sqrt(Time));
            double result = CDF(d1);

            return result;
}
```

Basic C++ code

Random(.h)

```
#ifndef RANDOM_H
#define RANDOM_H

//Cumulative Normal distribution function
double CDF(double);

//Probability density function
double PDF(double);

#endif
```

Basic C++ code

Random(.cpp)

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

//Available in other code files?
const double PI = 3.141592653589793238463;
//What is this about?
double CDF(double x){
            double L = 0.0;
            double K = 0.0;
            double dCDF = 0.0;
            const double a1 = 0.31938153;
            const double a2 = -0.356563782;
            const double a3 = 1.781477937;
            const double a4 = -1.821255978;
            const double a5 = 1.330274429;
            L = abs(x);
            K = 1.0 / (1.0 + 0.2316419 * L);
            dCDF = 1.0 - 1.0 / sqrt(2 * PI) *
                        exp(-L * L / 2.0) * (a1 * K + a2 * K * K + a3 * pow(K, 3.0) +
                        a4 * pow(K, 4.0) + a5 * pow(K, 5.0));

            if (x < 0){
               return 1.0 - dCDF;
            }
            else      {
               return dCDF;
            }
}
```

Basic C++ code

BS_Pricer(.h)

```
#include <cmath>
#include "OptionClass.h"
 //Can we just pass "Option theOption" to the function
double forPolymorphism(Option* theOption){
             return theOption -> getPrice();

 (…)

private: System::Void button1_Click(System::Object^  sender, System::EventArgs^  e) {
                          //Get inputs
                          double Spot = Convert::ToDouble(textBox1->Text);
                          double Strike = Convert::ToDouble(textBox2->Text);
                          double Rate = Convert::ToDouble(textBox3->Text);
                          double Vol = Convert::ToDouble(textBox4->Text);
                          double Time = Convert::ToDouble(textBox5->Text);

                          //Option pointer to show polymorphism example. Where the object is stored?
                          Option* theOptionPointer = new VanillaCall(Spot, Strike, Rate, Vol, Time);

                          //Inherited object
                          VanillaCall theOption(Spot, Strike, Rate, Vol, Time);

                          //Output results
                          textBox6->Text = theOption.getPrice().ToString("0.##");
                          textBox7->Text = theOption.getDelta().ToString("0.##");
                          textBox8->Text = theOption.getGamma().ToString("0.##");
                          textBox9->Text = theOption.getVega().ToString("0.##");
                          textBox10->Text = theOption.getTheta().ToString("0.##");
                          //Pay attention to pointer passed
                          MessageBox::Show(forPolymorphism(theOptionPointer).ToString("0.##"));
                          delete theOptionPointer;
```

CMF MSU

Vanilla products using QuantLib

http://quantlib.org/

QuantLib

A free/open-source library for quantitative finance

QuantLib User Meeting 2015: slides from the presentations are available.

Get QuantLib

Head to our download page to get the latest official release, or check out the latest development version from our git repository. QuantLib is also available in other languages.

Documentation

Documentation is available in several formats from a number of sources. You can also read our Installation instructions to get QuantLib working on your computer.

Need Help?

If you need to ask a question, subscribe to our mailing list and post it there. Before doing that, though, you might want to look at the FAQ and check if it was already answered.

Found a bug?

If you have a patch, open a pull request on GitHub or post it to our patch manager. Otherwise, report the bug on our issue tracker.

Want to contribute?

Just fork our repository on GitHub and start coding (instructions are here). Please have a look at our developer intro and guidelines.

More info

Here is the QuantLib license, the list of contributors, and the version history. The project page on Sourceforge is available at this link.

OSI certified

Hosted by sourceforge

Supported by statpro

The QuantLib project is aimed at providing a comprehensive software framework for quantitative finance. QuantLib is a free/open-source library for modeling, trading, and risk management in real-life.

QuantLib is written in C++ with a clean object model, and is then exported to different languages such as C#, Objective Caml, Java, Perl, Python, GNU R, Ruby, and Scheme. The reposit project facilitates deployment of object libraries to end user platforms and is used to generate QuantLibXL, an Excel addin for QuantLib, and QuantLibAddin, QuantLib addins for other platforms such as LibreOffice Calc. Bindings to other languages and porting to Gnumeric, Matlab/Octave, S-PLUS/R, Mathematica, COM/CORBA/SOAP architectures, FpML, are under consideration. See the extensions page for details.

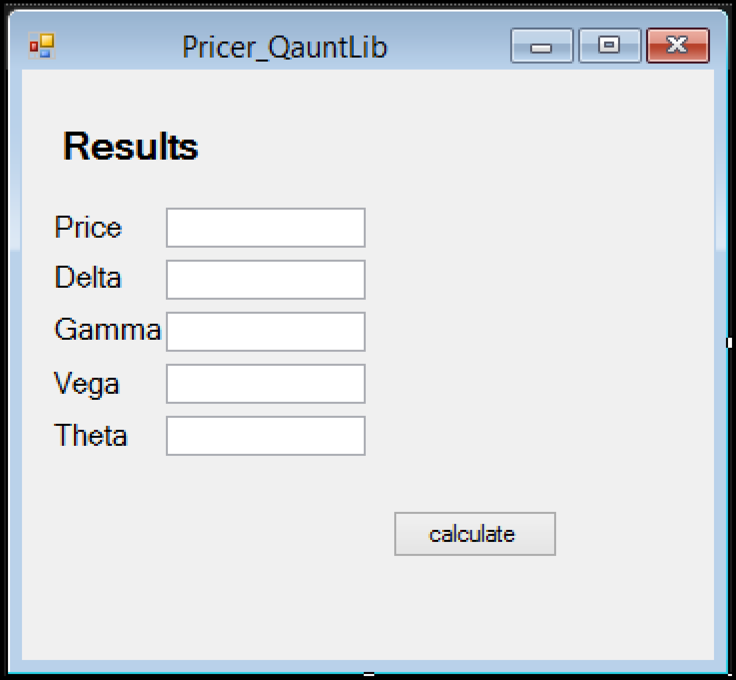
Appreciated by quantitative analysts and developers, it is intended for academics and practitioners alike, eventually promoting a stronger interaction between them. QuantLib offers tools that are useful both for practical implementation and for advanced modeling, with features such as market conventions, yield curve models, solvers, PDEs, Monte Carlo (low-discrepancy included), exotic options, VAR, and so on.

Finance is an area where well-written open-source projects could make a tremendous difference:

- any financial institution needs a solid, time-effective, operative implementation of cutting edge pricing models and hedging tools. However, to get there, one is currently forced to re-invent the wheel every time. Even standard decade-old models, such as Black-Scholes, still lack a public robust implementation. As a consequences many good quants are wasting their time writing C++ classes which have been already written thousands of times.
- By designing and building these tools in the open, QuantLib will both encourage peer review of the tools themselves, and demonstrate how this ought to be done for scientific and commercial software. Dan Gezelter's talk at the first Open Source/Open Science conference discussed how the scientific tradition of peer review fits well with the philosophy of the Open Source movement. Open standards are the only fair way for science and technology to evolve.

Vanilla products using QuantLib

Verified code



Vanilla products using QuantLib

BS (.cpp) (1/2)

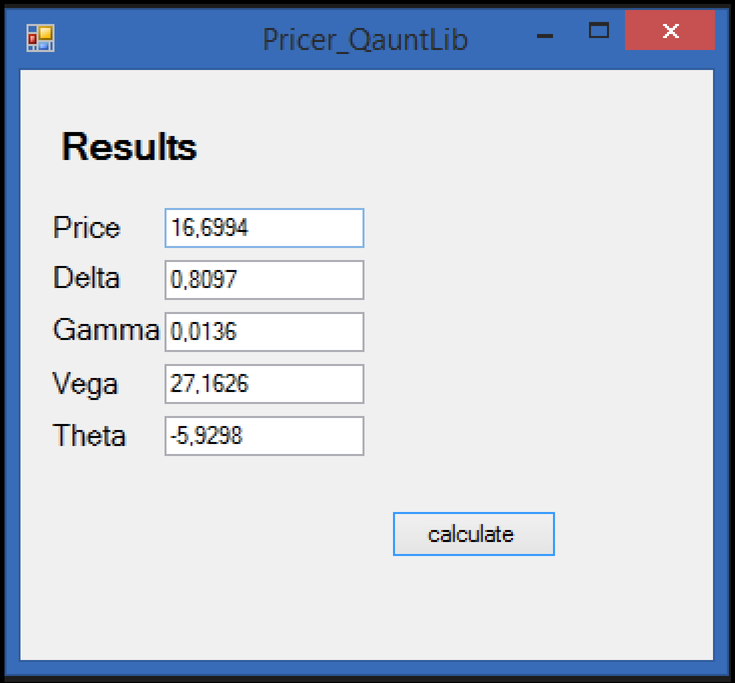
```
#include "BS.h"
#include <ql/quantlib.hpp>

using namespace QuantLib;

double* European_BS1(){

            double* myarr = new double[5];

            Calendar calendar = TARGET();
            Date todaysDate(12, Jan, 2015);
            Date settlementDate(12, Jan, 2015);
            Settings::instance().evaluationDate() = todaysDate;
            Option::Type type(Option::Call);
            Real underlying = 100.0;
            Real strike = 90.0;
            Spread dividendYield = 0.0;
            Rate riskFreeRate = 0.05;
            Volatility volatility = 0.20;
            Date maturity(12, Jan, 2016);
            DayCounter dayCounter = Actual365Fixed();
            //DayCounter dayCounter = SimpleDayCounter();
            std::string method;

            boost::shared_ptr<Exercise> europeanExercise(new EuropeanExercise(maturity));
            Handle<Quote> underlyingH(boost::shared_ptr<Quote>(new SimpleQuote(underlying)));
```

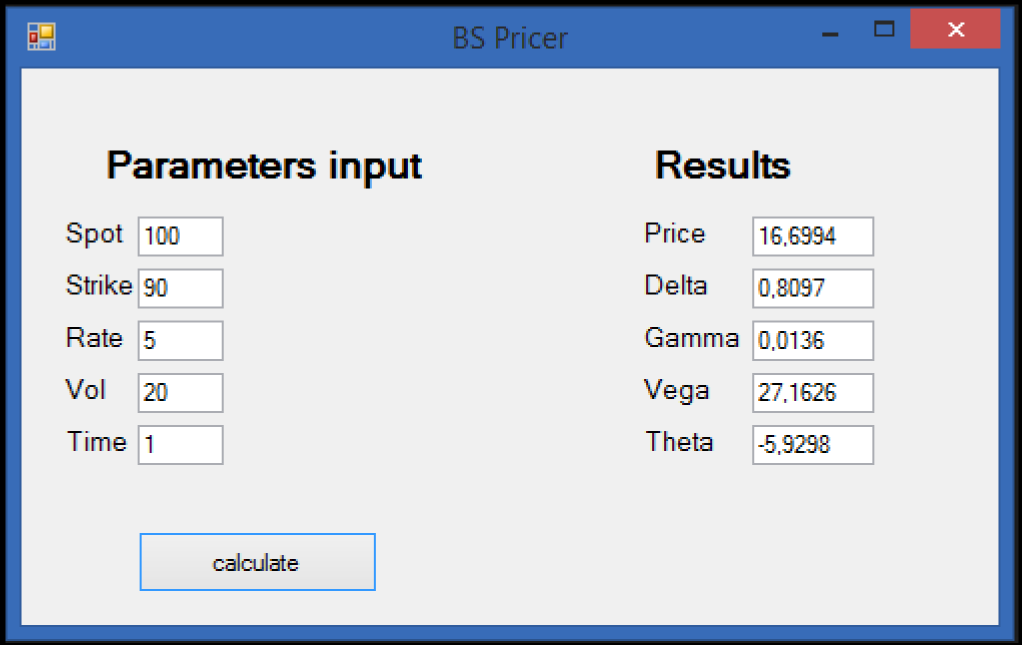
Vanilla products using QuantLib

BS (.cpp) (2/2)

```
Handle<YieldTermStructure> flatTermStructure(boost::shared_ptr<YieldTermStructure>(new
             FlatForward(settlementDate, riskFreeRate, dayCounter)));
Handle<YieldTermStructure> flatDividendTS(boost::shared_ptr<YieldTermStructure>(new
             FlatForward(settlementDate, dividendYield, dayCounter)));
Handle<BlackVolTermStructure> flatVolTS(boost::shared_ptr<BlackVolTermStructure>(new
             BlackConstantVol(settlementDate, calendar, volatility, dayCounter)));
boost::shared_ptr<StrikedTypePayoff> payoff(new PlainVanillaPayoff(type, strike));
boost::shared_ptr<BlackScholesMertonProcess> bsmProcess(new BlackScholesMertonProcess(underlyingH,
             flatDividendTS, flatTermStructure, flatVolTS));

VanillaOption europeanOption(payoff, europeanExercise);
method = "Black-Scholes:  ";
europeanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(new
             AnalyticEuropeanEngine(bsmProcess)));

myarr[0] = europeanOption.NPV();
myarr[1] = europeanOption.delta();
myarr[2] = europeanOption.gamma();
myarr[3] = europeanOption.vega();
myarr[4] = europeanOption.theta();

return  myarr;
}
```

QuantLib vs Hardcode
Comparing results





Homework assignment 1

- Modify program to allow user to choose type of an option – call/put.
- Take into account dividends.
- Use any alternative implementation of the CDF.
- Program should be object-oriented.

- **Deadline** – 27th March EOD