Derivatives Pricing Course

Lecture 3 – Finite-difference schemes

Artem Isaev

2016 CMF

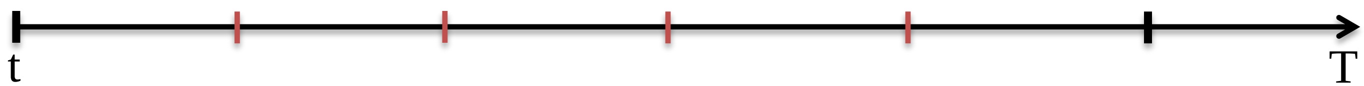
Agenda

- Options with early exercise rights.
- Possible solutions. Introduction to finite-difference schemes.
- C++ implementation.
- Compare results with QuantLib.

Options with early exercise rights

American/Bermudan options

- Such options can be exercised early, prior to maturity.



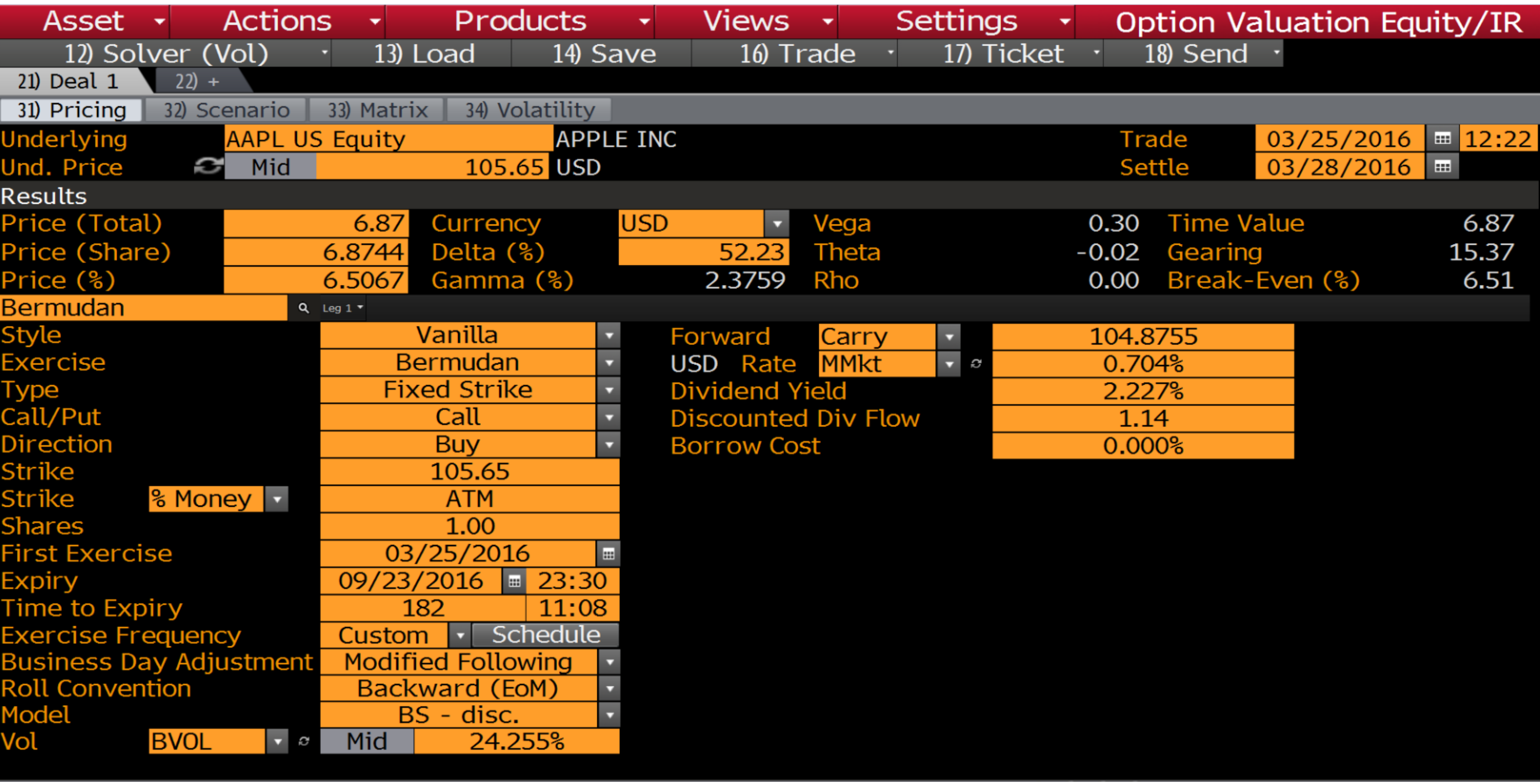
- Bermudan option can be exercised at certain specified times
American – anytime before T

- Can we use PDE/risk-neutral approach to find the price?

- What option has the greatest value American/European/Bermudan?

Options with early exercise rights

Bermudan option



Options with early exercise rights

Bermudan option

Start Date 03/25/2016
End Date 09/23/2016

Frequency: Custom
Business Day Adj: Modified Following
Roll Convention: Backward (End of M

Date	Strike
03/25/2016	105.65
03/28/2016	105.65
05/23/2016	105.65
05/28/2016	105.65
07/25/2016	105.65
07/28/2016	105.65
07/29/2016	105.65
09/23/2016	105.65
mm/dd/yyyy	

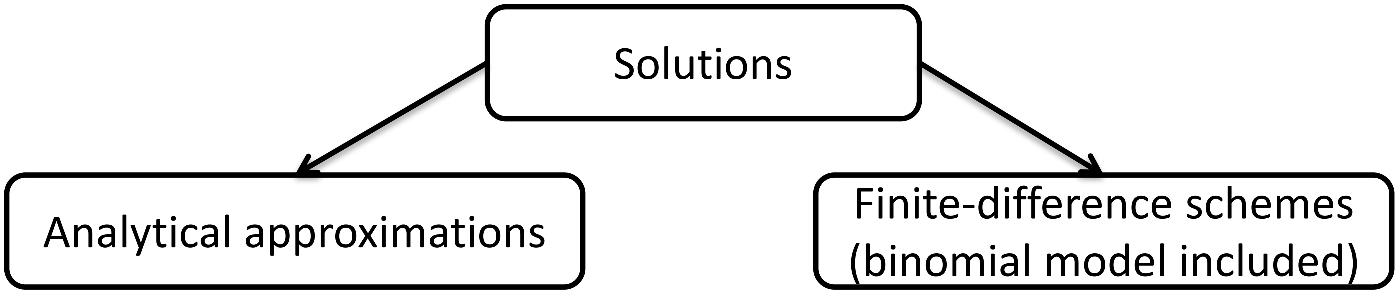
Date	Strike

Date	Strike

Close

Options with early exercise rights

Possible solutions



- When the contract is American the long/short position is asymmetrical, it is the holder of the exercise rights who controls the early exercise feature.

- If V is the value of a long position in an American option then all we can say is that we can earn *no more* than the risk-free rate on our portfolio. Thus we arrive at the *inequality*

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV \leq 0$$

When will we earn less?

Finite-difference methods
Problem formulation

- General PDE:

$$\frac{\partial V}{\partial t} + \mu(t,x)\frac{\partial V}{\partial x} + \frac{1}{2}\sigma(t,x)^2\frac{\partial^2 V}{\partial x^2} - r(t,x)V = 0$$

- where $V = V(t,x)$ satisfies a terminal condition $V(T,x) = g(x)$.

- Underneath the PDE lies a physical model of a state variable:

$$dx(t) = \mu\big(t, x(t)\big)dt + \sigma\big(t, x(t)\big)dW(t)$$

- We have a *Cauchy problem* to be solved for $V(t,x)$ on $(t,x) \in [0,T) \times \mathcal{B}$, where $\mathcal{B}$ is a range of values attainable by $x(t)$.

- For numerical solution we often need to assume that the domain of the state variable is finite, even if our equation supposed to hold for an infinite domain.

Finite-difference methods

Infinite domain

- Suitable truncation of the domain can often be done probabilistically, based on a confidence interval for $x(T)$.
  - What distribution has a stock price $S(T)$ in BS model?
- Consider our Black-Scholes PDE. A common first step is to use transformation $x = \ln S$ (see Chain rule):

$$\frac{\partial V}{\partial t} + \left(r - \frac{1}{2}\sigma^2\right)\frac{\partial V}{\partial x} + \frac{1}{2}\sigma^2 \frac{\partial^2 V}{\partial x^2} - rV = 0,$$

with terminal value (call option) $V(T, x) = (e^x - K)^+$

- The domain of x is here the entire real line, $\mathcal{B} = \mathbb{R}$.

Finite-difference methods

Finite domain

- Gaussian random variable with mean $\bar{x} = x(0) + \left(r - \frac{1}{2}\sigma^2\right)T$ and variance $\sigma^2 T$:

$$x(T) = x(0) + \left(r - \frac{1}{2}\sigma^2\right)T + \sigma(W(T) - W(0))$$

- Consider now replacing the domain $(-\infty, \infty)$ with the finite interval $\left[\bar{x} - \alpha\sigma\sqrt{T}, \bar{x} + \alpha\sigma\sqrt{T}\right]$ for some positive constant α.

- If our asset price model is not too complicated, it is possible to write an exact confidence interval for $x(T)$.

- Otherwise approximations should be used, for example, using “average” values for μ and σ.

Finite-difference methods

Discretization

- In order to solve the PDE numerically, we now wish to discretize it on a rectangular domain $(t, x) \in [0, T] \times \left[\underline{M}, \overline{M}\right]$, where $\overline{M}$ and $\underline{M}$ are finite constants.

- We introduce two equidistant grids $\left\{t_i\ , i = \overline{0, n}\right\}$ and $\left\{x_j\ , j = \overline{0, m+1}\right\}$ where

$$t_i = \frac{iT}{n} \triangleq i\Delta_t$$
$$x_j = \underline{M} + j\left(\overline{M} - \underline{M}\right)/m + 1 \triangleq \underline{M} + j\Delta_x$$

- The terminal value $V(T, x) = g(x)$ is imposed at $t_n = T$, and spatial boundary conditions are imposed at x_0 and x_{m+1}.

Finite-difference methods

Discretization in x-direction

- Restrict x to take values in the interior of the spatial grid $x \in \{x_j\}, j = \overline{1, m}$.
- Introduce the difference operators:

$$\delta_x V(t, x_j) \triangleq \frac{V(t, x_{j+1}) - V(t, x_{j-1})}{2\Delta_x},$$

$$\delta_{xx} V(t, x_j) \triangleq \frac{V(t, x_{j+1}) - 2V(t, x_j) + V(t, x_{j-1})}{{\Delta_x}^2}$$

- Lemma 3.1.

$$\delta_x V(t, x_j) \triangleq \frac{\partial V(t, x_j)}{\partial x} + O({\Delta_x}^2),$$

$$\delta_{xx} V(t, x_j) \triangleq \frac{\partial^2 V(t, x_j)}{\partial x^2} + O({\Delta_x}^2)$$

Finite-difference methods

Boundary conditions

- We should also specify the side boundary conditions at x_0 and x_{m+1}:

$$V(x_0, t) = \underline{f}(t, x_0),$$
$$V(x_{m+1}, t) = \overline{f}(t, x_{m+1})$$

- For instance, for the case of a simple call option on a stock paying no dividends:

$$\overline{f}(t, x_{m+1}) = e^{x_{m+1}} - Ke^{-r(T-t)},$$
$$\underline{f}(t, x_0) = 0$$

Finite-difference methods
Boundary conditions

- Deriving asymptotic conditions can be quiet involved for complicated option payouts. One common idea involves making assumptions on the form of the functional dependency between V and x at the grid boundaries, often in terms of spatial derivatives.
- For instance we can impose the condition that the second derivative of V is zero at the upper bound:

$$\frac{V(t, x_{m+1}) - 2V(t, x_m) + V(t, x_{m-1})}{{\Delta_x}^2} = 0$$

$$\Rightarrow V(t, x_{m+1}) = 2V(t, x_m) - V(t, x_{m-1})$$

What does it mean for option price?

Finite-difference methods

Time-Discretization

- By a Taylor expansion:

$$\frac{\partial V(t'_i(\theta))}{\partial t} = \frac{V(t_{i+1}) - V(t_i)}{\Delta_t} + 1_{\left\{\theta \neq \frac{1}{2}\right\}} O(\Delta_t) + O\left({\Delta_t}^2\right)$$

$$t'_i(\theta) = (1-\theta)t_{i+1} + \theta t_i$$

- This result on the convergence order is intuitive since only in the case $\theta = \frac{1}{2}$ is the difference coefficient precisely central.
  - $\theta = 1$ – fully implicit scheme
  - $\theta = 0$ – fully explicit scheme
  - $\theta = \frac{1}{2}$ – Crank-Nicolson scheme

Finite-difference methods

Fully-explicit scheme

- Theta scheme:

$$\frac{V(t_{i+1},x_j)-V(t_i,x_j)}{\Delta_t}+O(\Delta_t)$$

- Spatial scheme:

$$\left(r-\frac{1}{2}\sigma^2\right)\left(\frac{V(t_{i+1},x_{j+1})-V(t_{i+1},x_{j-1})}{2\Delta_x}\right)$$

$$+\frac{1}{2}\sigma^2\left(\frac{V(t_{i+1},x_{j+1})-2V(t_{i+1},x_j)+V(t_{i+1},x_{j-1})}{{\Delta_x}^2}\right)$$

$$-rV(t_{i+1},x_j)+O({\Delta_x}^2)$$

Finite-difference methods

Convergence

- <u>Proposition 3.2</u> The aforementioned scheme recovers $\hat{V}(0)$ in $O(mn)$ operations. If the scheme converges, the error is of order:
$$O\left({\Delta_x}^2\right) + O(\Delta_t)$$

- If one has enough skill to implement the Crank-Nicolson scheme the order will be:

$$O\left({\Delta_x}^2\right) + O\left({\Delta_t}^2\right)$$

- To get next value of the option, we need to solve a set of linear equations. Each value is linked to its spatial neighbors.

Basic C++ code

AmericanOption(.h)

```
#ifndef AMERICANOPTION_H
#define AMERICANOPTION_H

#include "OptionClass.h"
#include <vector>

class AmericanCall : public Option
{
public:
		AmericanCall(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
		int TimeSteps; //number of steps
		double deltaT; //step length
		int UnderlyingSteps; //number of steps
		double deltaX; //step length
		std::vector < std::vector<double> > Grid; //pricing grid
		std::vector <double> SpotArray; //possible spot values
		int SpotPosition; //position in SpotArray corresponding to the first element greater than Spot

};
#endif
```

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:
};

//Any potential problems because of such definition? (what if we already have hundreds of products)

class AmericanCall : public Option { … }
```

Basic C++ code

AmericanOption(.cpp) (3/4)

```
double AmericanCall::getPrice() const{
            //Can we build a Grid inside this function?
            //Find corresponding price using linear interpolation
            double Price;
            Price = Grid[0][SpotPosition - 1] + (Grid[0][SpotPosition] - Grid[0][SpotPosition - 1]) *
                   (Spot - SpotArray[SpotPosition - 1]) /
                   (SpotArray[SpotPosition] - SpotArray[SpotPosition - 1]);
            return Price;
}
double AmericanCall::getDelta() const{
            double resultLeft, resultRight, result;
            //How to calculate derivative with respect to S?
            resultLeft = (Grid[1][(SpotPosition - 1) + 1] - Grid[1][(SpotPosition - 1) - 1]) /
                         2.0 / deltaX / Spot; //!!!
            resultRight = (Grid[1][SpotPosition + 1] - Grid[1][SpotPosition - 1]) /
                         2.0 / deltaX / Spot; //!!!

            result = resultLeft + (resultRight - resultLeft) * (Spot - SpotArray[SpotPosition - 1]) /
                    (SpotArray[SpotPosition] - SpotArray[SpotPosition - 1]);
            return result;
}
```

Basic C++ code

AmericanOption(.cpp) (4/4)

```
double AmericanCall::getGamma() const{

            double resultLeft, resultRight, result;
            //What about the second derivative?
            resultLeft = (Grid[1][(SpotPosition - 1) + 1] - 2 * Grid[1][(SpotPosition - 1)] +
                          Grid[1][(SpotPosition - 1) - 1]) / deltaX / deltaX / Spot / Spot -
                         (Grid[1][(SpotPosition - 1) + 1] - Grid[1][(SpotPosition - 1) - 1]) / 2.0 /
                         deltaX / Spot / Spot; //!!!
            resultRight = (Grid[1][SpotPosition + 1] - 2 * Grid[1][SpotPosition] +
                           Grid[1][SpotPosition - 1]) / deltaX / deltaX / Spot / Spot -
                          (Grid[1][SpotPosition + 1] - Grid[1][SpotPosition - 1]) / 2.0 /
                         deltaX / Spot / Spot; //!!!

            result = resultLeft + (resultRight - resultLeft) * (Spot - SpotArray[SpotPosition - 1]) /
                     (SpotArray[SpotPosition] - SpotArray[SpotPosition - 1]);

            return result;
}
//Alternative solution? Disadvantages of the following code
double AmericanCall::getVega() const{
            double deltaVol = 1;
            AmericanCall dummy(Spot, Strike, Rate * 100.0, Vol * 100.0 + deltaVol, Time);

            double result = (dummy.getPrice() - getPrice()) / (deltaVol / 100.0);

            return result;
}
```

Finite-difference scheme using QuantLib

BS(.cpp) (1/2)

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

using namespace QuantLib;

double* AmericanFD(){

            double* myarr = new double[3];

            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();
            std::string method;

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

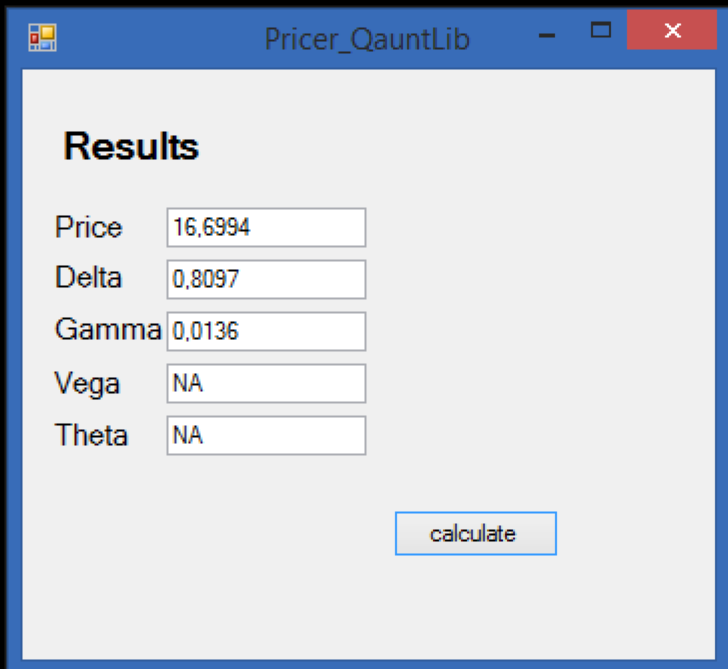
Finite-difference scheme using QuantLib

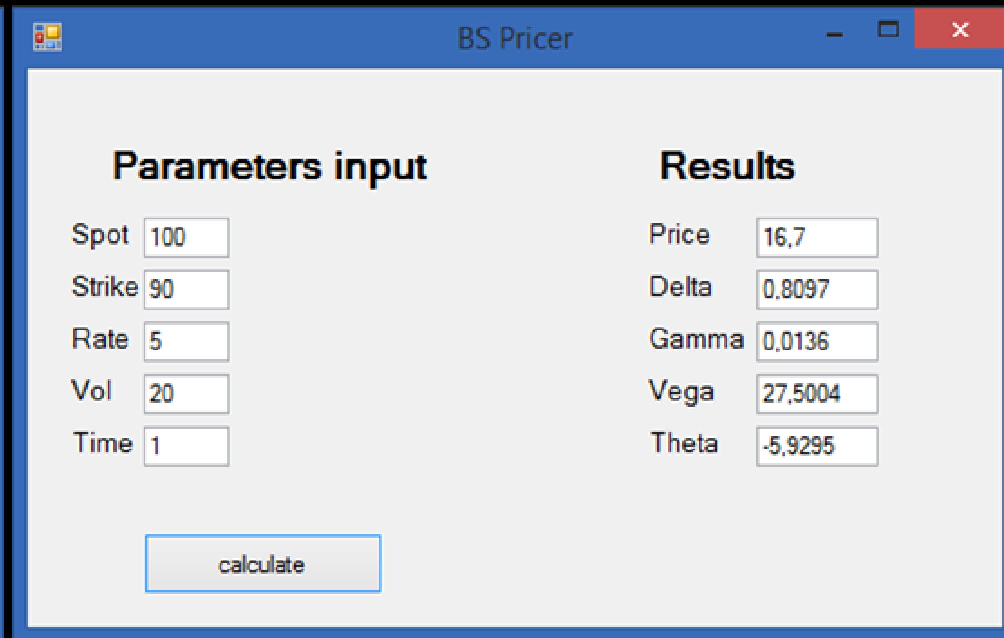
BS(.cpp) (2/2)

```
Handle<YieldTermStructure> yieldTermStructure(boost::shared_ptr<YieldTermStructure>(new
            FlatForward(settlementDate, riskFreeRate, dayCounter)));
Handle<YieldTermStructure> dividendTermStructure(boost::shared_ptr<YieldTermStructure>(new
            FlatForward(settlementDate, dividendYield, dayCounter)));
Handle<BlackVolTermStructure> volatilityTermStructure(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,
                                    dividendTermStructure, yieldTermStructure, volatilityTermStructure));

VanillaOption americanOption(payoff, americanExercise);
method = "Explicit scheme:  ";
americanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(new
            FDAmericanEngine<ExplicitEuler>(bsmProcess,100000,50)));

myarr[0] = americanOption.NPV();
myarr[1] = americanOption.delta();
myarr[2] = americanOption.gamma();

return  myarr;

}
```

QuantLib vs Hardcode

Comparing results

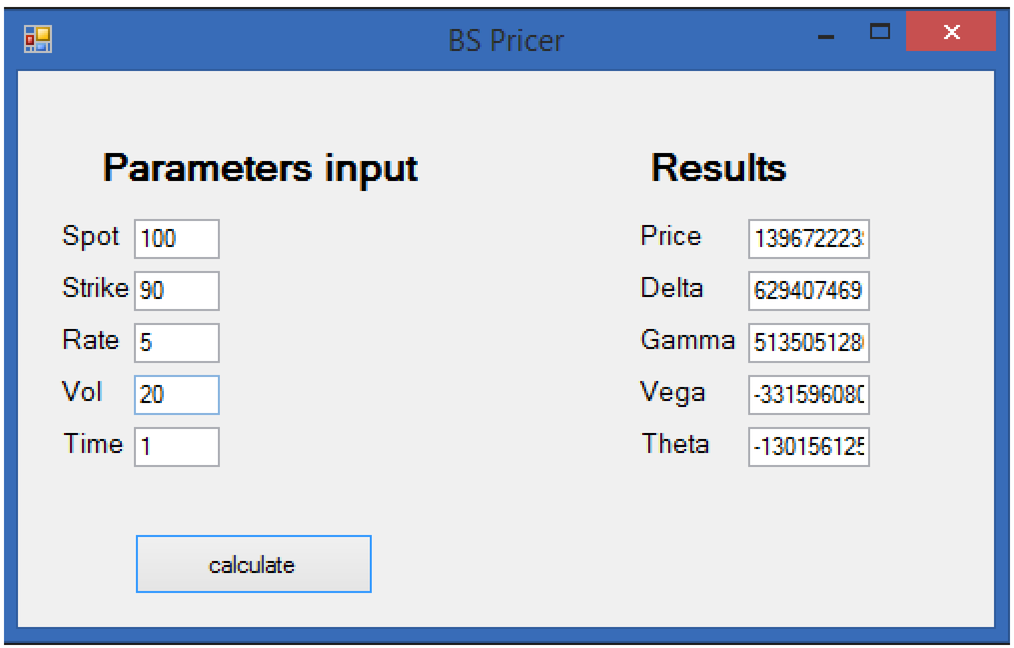




Recall European option

Analyzing results

- Why results are equal for the call option?
- Results with TimeSteps = 1000, UnderlyingSteps = 500



Homework assignment 2

- Modify program to price Bermudan put option with a floating strike.
- Take into account dividends.
- Modify upper boundary - payoff is at most linear in the underlying.
- Answer questions in `green` from the code.

- **Deadline** – 8th April EOD