Derivatives Pricing Course

Lecture 4 – Finite-difference schemes. Advanced topics.

Artem Isaev

2016 CMF

Agenda

- Implicit and Crank-Nicolson schemes.
- Stability issues. Non-equidistant discretization.
- Add new features to American option.
- C++ implementation.
- Compare results with QuantLib.

Finite-difference schemes

Previously discussed

- A variety of schemes allow us to solve PDEs numerically. There are many ways these schemes can be improved in terms of speed and accuracy.

- Fully explicit scheme is easy to implement for vanilla products, including securities with early exercise rights.

Finite-difference schemes

Matrix notation. Implicit scheme

- Recall our discretized equation (no change of variables $x = \log S$):

$$\frac{V\left(t_{i+1}, S_j\right) - V(t_i, S_j)}{\Delta_t} + rS_j\left(\frac{V\left(t_i, S_{j+1}\right) - V\left(t_i, S_{j-1}\right)}{2\Delta_S}\right)$$

$$+\frac{1}{2}{S_j}^2\sigma^2\left(\frac{V\left(t_i, S_{j+1}\right) - 2V\left(t_i, S_j\right) + V\left(t_i, S_{j-1}\right)}{{\Delta_S}^2}\right)$$

$$-rV\left(t_i, S_j\right) + \varepsilon$$

ε – error term

What option values are known?

Finite-difference schemes

Matrix notation. Implicit scheme

- For each pair (i, j) we can write an equation:

$$\underbrace{a(i,j)V\left(t_i, S_{j-1}\right) + b(i,j)V\left(t_i, S_j\right) + c(i,j)V\left(t_i, S_{j+1}\right)}_{\textbf{Unknown values}} = \underbrace{V\left(t_{i+1}, S_j\right)}_{\textbf{Known value}}$$

$$a(i,j) = -\frac{\sigma^2 j^2 \Delta_t}{2} - \frac{rj\Delta_t}{2}$$

$$b(i,j) = 1 + \sigma^2 j^2 \Delta_t + r\Delta_t$$

$$c(i,j) = -\frac{\sigma^2 j^2 \Delta_t}{2} + \frac{rj\Delta_t}{2}$$

Finite-difference schemes

Matrix notation. Implicit scheme

- We want to rewrite our problem as $AV = d$ where:
  $A - squared\ matrix\ (j_{max}+1, j_{max}+1)$

  $V\ - vector\ (j_{max}+1)$

- Where A is a *tri-diagonal matrix*

$$\begin{pmatrix}
b_0(i) & c_0(i) & 0 & 0 & 0 & \cdots & 0 \\
a_1(i) & b_1(i) & c_1(i) & 0 & 0 & \cdots & 0 \\
0 & a_2(i) & b_2(i) & c_2(i) & 0 & \cdots & 0 \\
0 & 0 & a_3(i) & b_3(i) & c_3(i) & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \ddots & \ddots & 0 \\
0 & 0 & 0 & 0 & a_{j_{max}-1}(i) & b_{j_{max}-1}(i) & c_{j_{max}-1}(i) \\
0 & 0 & 0 & 0 & 0 & a_{j_{max}}(i) & b_{j_{max}}(i)
\end{pmatrix}$$

Finite-difference schemes

Matrix notation. Implicit scheme

- V is a *vector* of option values at time t_i
- d is a *vector* of known values at time t_{i+1}

$$\begin{pmatrix} V(t_i, S_0) \\ V(t_i, S_1) \\ \vdots \\ V(t_i, S_{j_{max}-1}) \\ V(t_i, S_{j_{max}}) \end{pmatrix}$$

$$\begin{pmatrix} d_0 \\ d_1 \\ \vdots \\ d_{j_{max}-1} \\ d_{j_{max}} \end{pmatrix}$$

$V(t_i, S_0), V(t_i, S_{j_{max}})$ – values at boundaries

Finite-difference schemes

Matrix notation. Implicit scheme

- After each time step, one should solve the system of linear equations.

- The most straightforward to perform the final calculations is to find the inverse matrix.

- Since our matrix A is tri-diagonal, we can solve the equation using special methods - LU decomposition, SOR, …

Finite-difference schemes
Stability issues

- Ignoring contributions from boundary conditions, our finite-difference scheme can be rewritten:

$$\hat{V}(t_i) = B_i^{\,i+1}\hat{V}(t_{i+1})$$

- The scheme is *stable* if for all k the absolute value of $\hat{V}(t_k)$ is bounded.
- The fully explicit method is quite easy to implement, there are some limitations related to convergence and errors.
- In BS model, we have restriction of the form:

$$\boxed{\sigma^2 \leq \frac{{\Delta_x}^2}{\Delta_t}}$$

which can be quite onerous, often requiring the use of thousands of time steps in the finite difference grid.

Finite-difference schemes

Stability issues

- Returning to the case $\frac{1}{2} \leq \theta \leq 1$

- For these θ values, irrespective of the magnitudes of Δ_x and Δ_t our scheme stays *stable*!

- It should be noted that if there is a discontinuity in the terminal value function high frequency oscillations can creep into numerical solution.

Finite-difference schemes

Non-Equidistant Discretization

- In practice we often wish to align the finite difference grid to particular dates (e.g. on which coupons or dividends are paid) and particular values of x (e.g. those on which strikes and barriers are positioned).

- For the time domain $\Delta_{t,i} \triangleq t_{i+1} - t_i$ and the backward induction algorithm can proceed as before.

- Irregular grid for x:

$$\Delta_{x,j}{}^{+} \triangleq x_{j+1} - x_j, \Delta_{x,j}{}^{-} \triangleq x_j - x_{j-1}$$

- To proceed we should redefine δ_x, δ_{xx} and also A matrix elements a_j, b_j, c_j.

Finite-difference schemes

Option examples

- In our discussion so far, we have assumed that options are characterized by a single terminal payoff function $g(x)$ and a set of spatial boundary conditions determining the option price at the boundaries of the x-domain.

- In reality, many options are more complicated and may involve early exercise decisions, pre-maturity cash-flows, path-dependency and more.

- We will consider some relatively straightforward examples and how to modify the basic finite difference algorithm to deal with them.

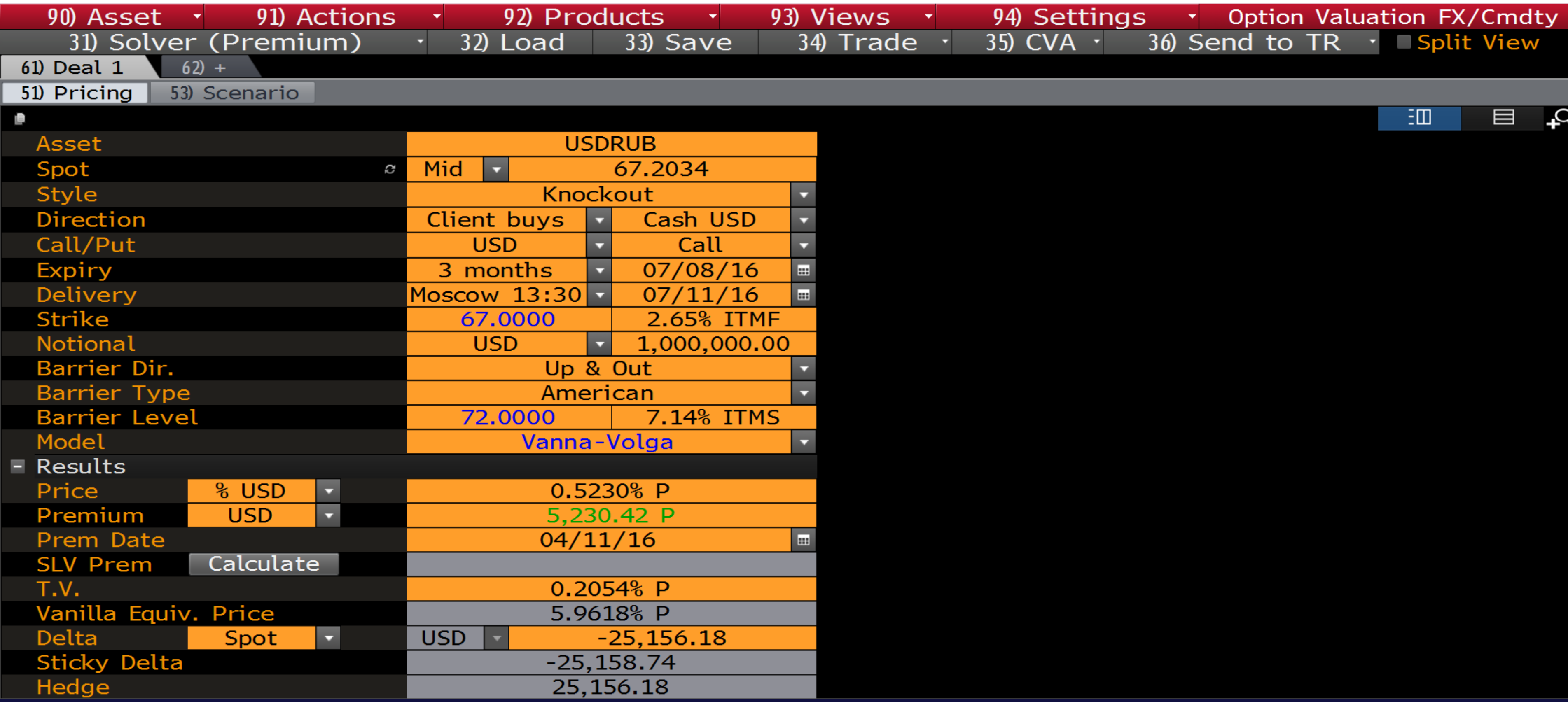
Finite-difference schemes

Continuous Barrier Options

- Barrier options are path-dependent options. They have a payoff that is dependent on the realized asset path.
- Useful for customers with very precise views about the direction of the market.
- Certain aspects of the contract are triggered if the asset price becomes too high or too low.
  - **Up-and-out**: become null if asset moves up higher than the barrier level
  - **Down-and-out**: become null if asset moves down lower than the barrier level
  - **Up-and-in**: activated if asset moves up higher than the barrier level
  - **Down-and-in**: activated if asset moves down lower than the barrier level

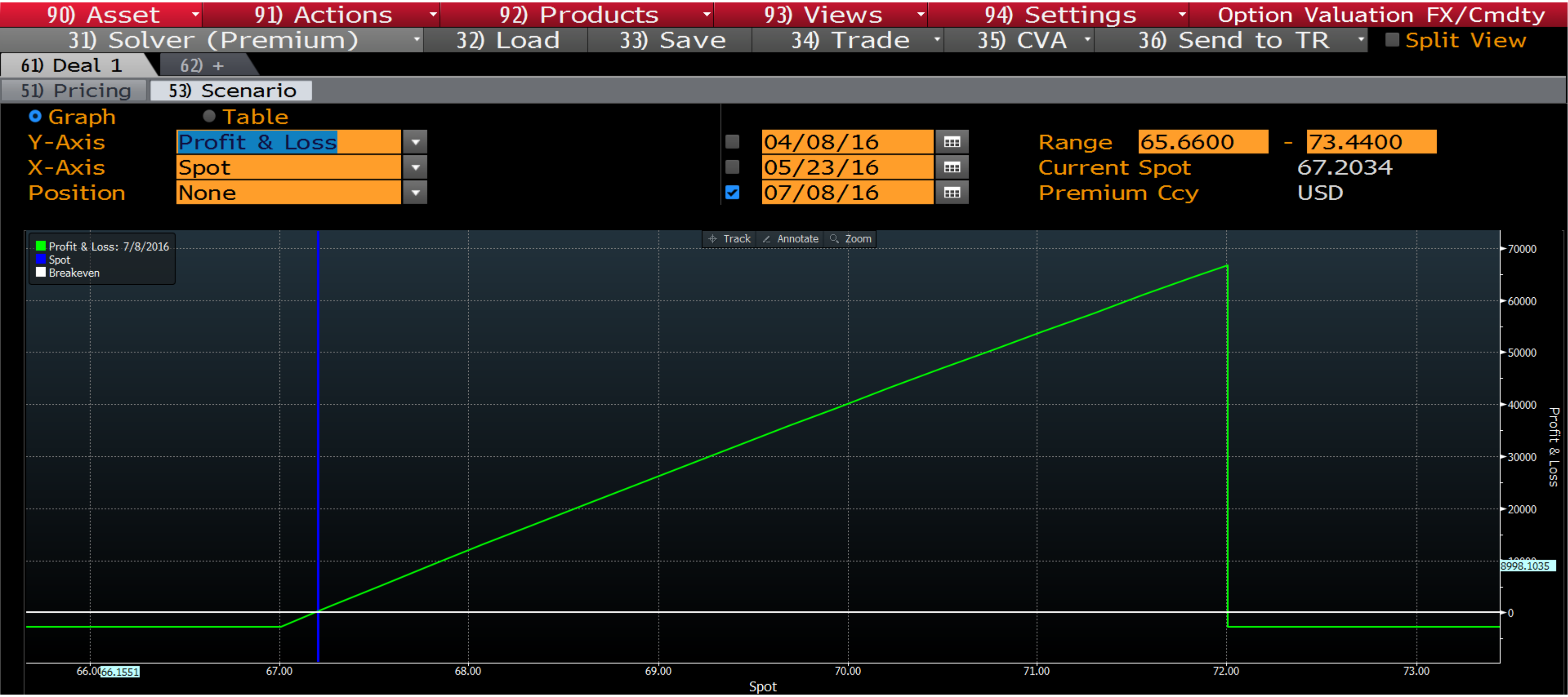
Finite-difference schemes

Barrier Options



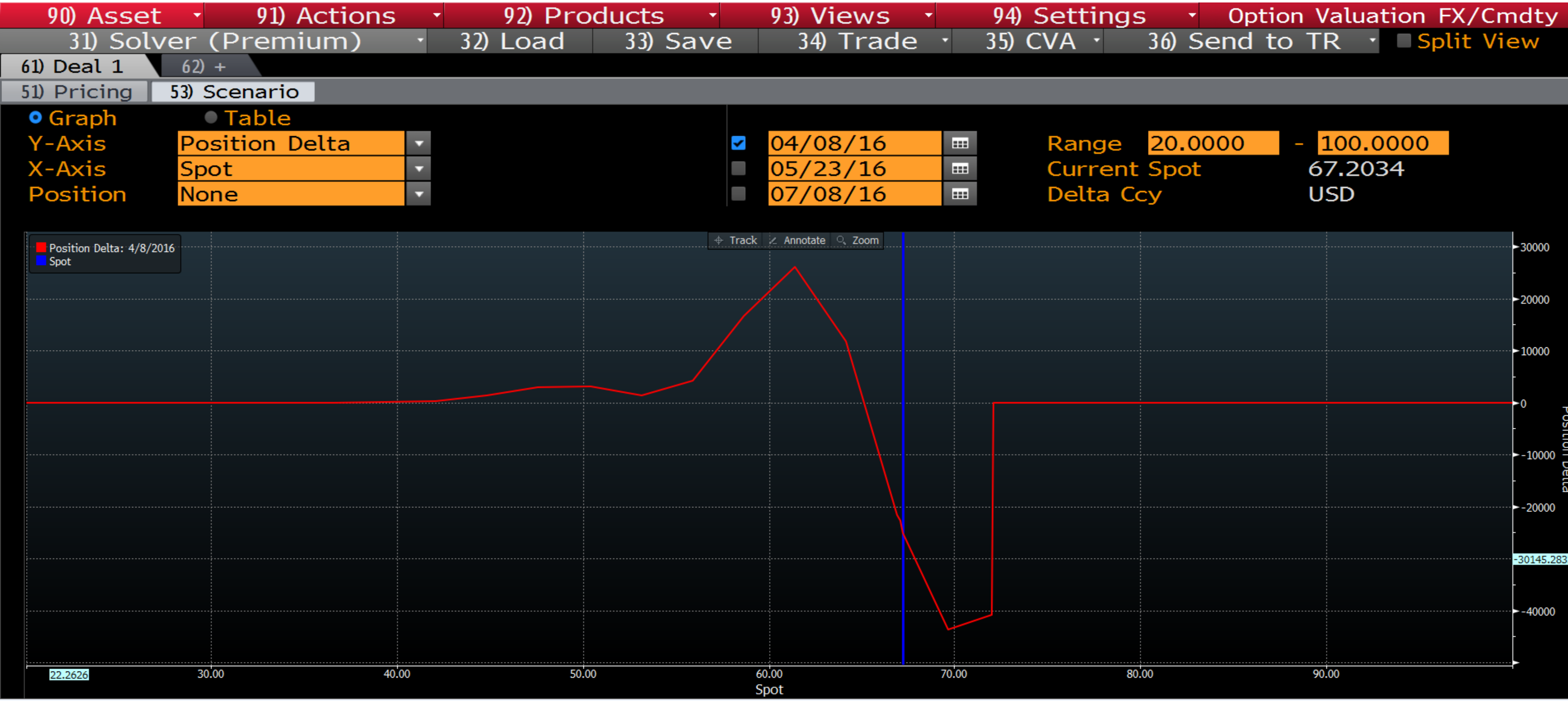
Finite-difference schemes

Barrier Options



Finite-difference schemes

Barrier Options



Finite-difference schemes
Continuous Barrier Options

- Up-and-out knock-out option, critical level H.
- Therefore we must simply solve our PDE on a domain $[\underline{M}, H]$, where $\underline{M}$ represents the lowest attainable value of a process $x(t)$.
- The boundary condition at the upper boundary is $V(t, H) = 0$
- Definitely, in this case we should dimension our spatial grid to have $x_{m+1} = H$.

- In practice, barrier options sometimes involve time-dependent barriers. For instance step-up/step-down barrier options will have piecewise flat barriers that increase/decrease at discrete points in time.

Finite-difference schemes

Continuous Barrier Options

- As an illustration, consider a barrier option where the upper barrier is flat, except for a discontinuous change at time $T^* < T$, at which point the barrier moves from a value of H^* to a value of H, with $H > H^*$.
- We make sure that one level in the spatial grid – say $x_{k+1}, k < m$, - is set exactly at the level H^*.
- Similarly we make sure that one level in the time grid is set exactly to T^*.
- Starting at time T, we then iterate backwards in time (m −dimensional systems). The moment we hit T^*, the PDE now only applies to the smaller region $\left[\underline{M}, H^*\right]$, covered by the reduced spatial grid with $x_{k+1} = H^*$ (k −dimensional systems).

Finite-difference schemes
Discrete Barrier Options

- In practice, monitoring the barrier condition continuously can be impractical, and it may be imposed on a discrete set of dates.
- In this case we should allow the value function to "diffuse" above the barrier levels between dates in the monitoring set. So we discretize the PDE on a larger domain - $[\underline{M}, \overline{M}]$.
- Between barrier observation dates, we solve our PDE by the standard methods.
- At each barrier observation time T_k, we must impose a *barrier jump condition:*

$$V(T_k-, x) = V(T_k+, x)1_{\{x<H\}}, k = 1, \dots, K$$

- This jump condition will generally produce a discontinuity in V as a function of x, around the barrier level H.

C++ code

BarrierOptionCN(.h)

```
#ifndef BARRIEROPTIONCN_H
#define BARRIEROPTIONCN_H

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

class AmericanBarrierCallCN : public Option
{
public:
            AmericanBarrierCallCN(double, 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
            double DividendYield; //in % annualized
            int TimeSteps; //number of steps
            double deltaT; //step length
            int UnderlyingSteps; //number of steps
            double deltaS; //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
```

C++ code

BarrierOptionCN(.cpp) (1/5)

```
#include "BarrierOptionCN.h"

AmericanBarrierCallCN::AmericanBarrierCallCN(double Spot_, double Strike_, double Rate_, double Vol_,
double Time_, double DividendYield_) : Spot(Spot_), Strike(Strike_), Time(Time_){
            Rate = Rate_ / 100.0;
            Vol = Vol_ / 100.0;
            TimeSteps = 500; //Pay attention to the number of steps
            UnderlyingSteps = 750; //Pay attention to the number of steps
            DividendYield = DividendYield_ / 100.0;

            //Step 0. Add barrier. Barrier type is fixed - Up-and-out
            double barrier = 120.0;

            //Step 1. Reserve memory for a grid, reduce infinite domain and width of steps
            Grid.resize(TimeSteps);

            //Finite domain - zero to double spot, or barrier
            //double s_max = 2 * Spot;
            double s_max = barrier;
            double s_min = 0;

            deltaS = (s_max - s_min) / (UnderlyingSteps - 1);
            deltaT = Time / (TimeSteps - 1);

            //Step 2. Fill values at boundaries
            for (int i = 0; i < TimeSteps; i++){
                        Grid[i].resize(UnderlyingSteps);
            }
```

C++ code

BarrierOptionCN(.cpp) (2/5)

```
//The first line in case of no barriers
	for (int i = 0; i < TimeSteps; i++){
//Grid[i][UnderlyingSteps - 1] = s_max*exp(-DividendYield *(Time - i * deltaT)) -Strike*exp(-Rate*(Time-i*deltaT));
		Grid[i][UnderlyingSteps - 1] = 0;
		Grid[i][0] = 0;
	}
	for (int j = 0; j < UnderlyingSteps - 1; j++){
		Grid[TimeSteps - 1][j] = (s_min + deltaS*j) > Strike ? (s_min + deltaS*j) - Strike : 0;
	}
	//define matrix diagonals
	std::vector<double> a;
	std::vector<double> b;
	std::vector<double> c;
	std::vector<double> d;
	std::vector<double> temp;

	a.resize(UnderlyingSteps);
	b.resize(UnderlyingSteps);
	c.resize(UnderlyingSteps);
	d.resize(UnderlyingSteps);
	temp.resize(UnderlyingSteps);
	//Step 3. fill matrix A
	for (int i = 1; i < UnderlyingSteps - 1; i++){
		a[i] = 0.25 * (Vol*Vol*i*i - (Rate - DividendYield) * i);
		b[i] = -Vol*Vol*i*i*0.5 - Rate*0.5 - 1 / deltaT;
		c[i] = 0.25 * (Vol*Vol*i*i + (Rate - DividendYield) * i);
	}
```

C++ code

BarrierOptionCN(.cpp) (3/5)

```
//i = 0
b[0] = 1;
c[0] = 0;
a[0] = 0;


//i = underlyingSteps - 1;
a[UnderlyingSteps - 1] = 0;
b[UnderlyingSteps - 1] = 1;
c[UnderlyingSteps - 1] = 0;

//Values known at maturity time
d[0] = Grid[TimeSteps - 1][0];
d[UnderlyingSteps - 1] = Grid[TimeSteps - 2][UnderlyingSteps - 1];
for (int i = 1; i < UnderlyingSteps - 1; i++){
            d[i] = -a[i] * Grid[TimeSteps - 1][i - 1] - (b[i] + 2 / deltaT) * Grid[TimeSteps - 1][i] -
                    c[i] * Grid[TimeSteps - 1][i + 1];
}
```

Barrier option using QuantLib

BS(.cpp) (1/2)

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

using namespace QuantLib;

double* AmericanBarrier(){
            double* myarr = new double[0];
            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 = 0.02;
            Rate riskFreeRate = 0.05;
            Volatility volatility = 0.20;
            Date maturity(12, Jan, 2016);
            DayCounter dayCounter = Actual365Fixed();
            //DayCounter dayCounter = SimpleDayCounter();
            Barrier::Type barrierType = Barrier::UpOut;
            Real barrier = 120.0;
            Real rebate = 0.0;
            std::string method;

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

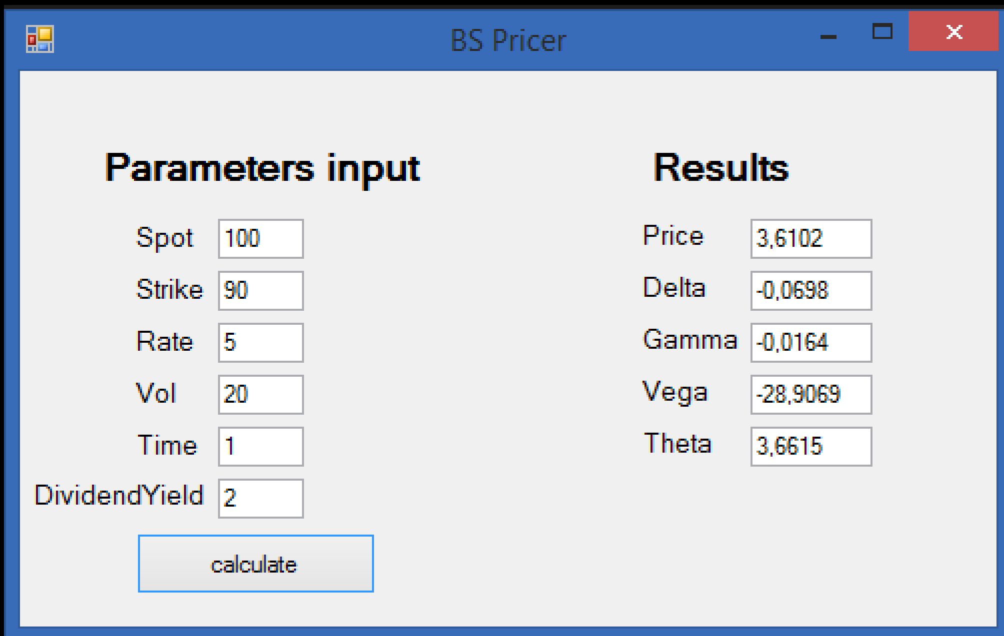
Barrier option using QuantLib

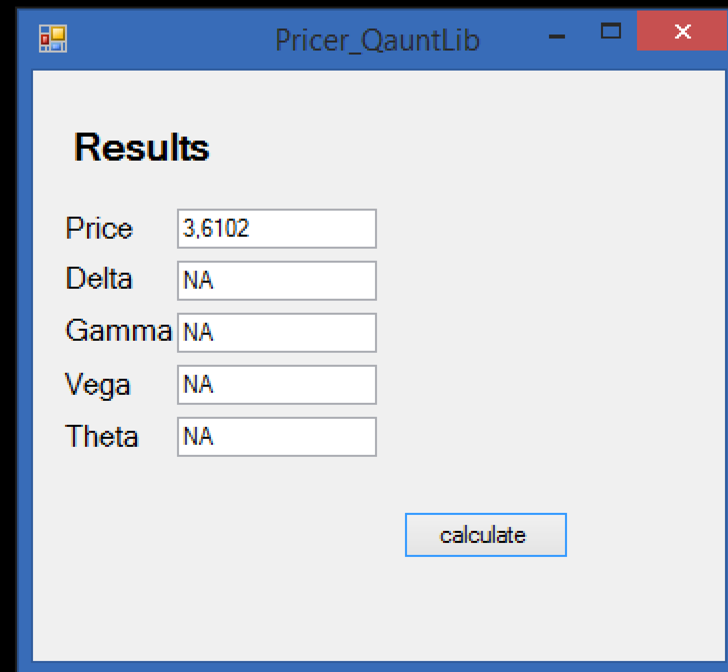
BS(.cpp) (2/2)

```
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));

BarrierOption barrierOption(barrierType, barrier, rebate,   payoff, americanExercise);

method = "Crank-Nicolson scheme:  ";
//No Finite-difference engines for barrier American options… no greek coefficients…
barrierOption.setPricingEngine(boost::shared_ptr<PricingEngine>(new AnalyticBarrierEngine(bsmProcess)));

myarr[0] = barrierOption.NPV();

return  myarr;

}
```

QuantLib vs Manual

Comparing results. Call option, how can delta be negative?





Homework assignment 3

- Modify program to price Down-and-out put option.
- Scheme should be Implicit.

- **Deadline** – 27th April EOD