Option Pricing with Java

Peter von Tessin

December 20, 2004

1 Introduction

Option Pricing has lost much of it's former allure as "rocket science" and nowdays almost any high-school kid with a computer can calculate the price of the most exotic of financial products. It is probably no coincidence that this development coincides with a reduction of profits generated at option trading desks. Most international financial institutions are only able to make money with either very unsophisticated customers, who are not able to determine the value of some (mostly intentionally) quite obscure products offered to them or by taking larger and larger bets in a way only hedge-funds used to do.

Two factors have contributed to this equalisation of the financial playing field: On the one hand the publication of very good books to introduce almost any interested person to the field of option pricing theory (e.g. "Paul Wilmott on Quantitative Finance", by, well, Paul Wilmott, Wiley 2000 or "An Introduction to the Mathematics of Financial Derivatives", by Salih Neftci, Academic Press 2000). On the other hand the proliferation of very powerful personal computers on which even sophisticated programs can be run in very efficiently.

In this paper we try to give an introduction to the basic concepts of option pricing using Java.

2 Financial Assets and Instruments

The currently most widly accepted model for financial asset S in contionous time is the following stochastic differential equation:

$$dS = \mu S dt + \sigma S dX \tag{1}$$

We will use this basic model to derive the Black-Scholes equation for valuing an option on this asset $V(S, t; \sigma, \mu; E, T; r)$.

3 Deriving the Black Scholes Equation

Starting with a portfolio Π of a long position in this option and a short position (of Δ) in the underlying asset:

$$\Pi = V(S,t) - \Delta S \tag{2}$$

Now looking at a small change in this portfolio we get the following equation:

$$d\Pi = dV(S,t) - \Delta dS \tag{3}$$

Using Ito's lemma on V:

$$dV = \frac{\delta V}{\delta t}dt + \frac{\delta V}{\delta S}dS + \frac{1}{2}\sigma^2 S^2 \frac{\delta^2 V}{\delta S^2}dt \tag{4}$$

we can see that a small change in the portfolio is equal to:

$$d\Pi = \frac{\delta V}{\delta t}dt + \frac{\delta V}{\delta S}dS + \frac{1}{2}\sigma^2 S^2 \frac{\delta^2 V}{\delta S^2}dt - \Delta dS \tag{5}$$

A little bit of classic delta-hedging ($\Delta = \frac{\delta V}{\delta S}$) will reduced this to:

$$\delta\Pi = (\frac{\delta V}{\delta t} + \frac{1}{2}\sigma^2 S^2 \frac{\delta^2 V}{\delta S^2})dt \tag{6}$$

The assumption of no-arbitrage dictates that:

$$d\Pi = r\Pi dt \tag{7}$$

and therfore we get the Black-Scholes equation of:

$$\frac{\delta V}{\delta t} + \frac{1}{2}\sigma^2 S^2 \frac{\delta^2 V}{\delta S^2} + rS\frac{\delta V}{\delta S} - rV = 0 \tag{8}$$

Now, in order to use this equation for valuation purposes we need to specify the final conditions. These are for a call: $V(S,T) = max(S-E,0)$ and for a put $V(S,T) = max(E-S,0)$.

/sectoinSolution of the Black Scholes Formula Now the famous formula for the call is a solution to this equation:

The value of a call on stock S is:

$$SN(d_1) - Ee^{-r(T-t)}N(d_2) \tag{9}$$

where:

$$d_1 = \frac{\log \frac{S}{E} + (r + \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}} \tag{10}$$

$$d_2 = d_1 - \sigma\sqrt{T-t} \tag{11}$$

Here, log denotes the natural logarithm, and:

S = the price of the underlying stock (the order-size)

E = the strike price (the amount of stock in the warehouse)

r = the continuously compounded risk free interest rate (depreciation rate for stock)

t = the time in years until the expiration of the option (well, what ever)

σ = the implied volatility for the underlying stock (volatility of orders)

N = the standard normal cumulative distribution function.

4 Finite Difference Method

But we are not interested in these special cases that have a "simple solution", we are interested in a more general case, where a numerical solution has to be found. We will find our first numerical solution using the explicit finite difference method. In order to do so we rewrite the Black-Scholes equation in a more genearl way:

$$\frac{\delta V}{\delta t} + a(S,t)\frac{\delta^2 V}{\delta S^2} + b(S,t)\frac{\delta V}{\delta S} + c(S,t)V = 0 \tag{12}$$

Using finite differences to approximate the derivatives this gives:

$$\frac{V_i^k - V_i^{k+1}}{\delta t} + a_i^k(\frac{V_{i+1}^k - 2V_i^k + V_{i-1}^k}{\delta S^2}) + b_i^k(\frac{V_{i+1}^k - V_{i-1}^k}{2\delta S}) + c_i^k V_i^k = O(\delta t, \delta S^2) \tag{13}$$

Rearanging this equation we get:

$$V_i^{k+1} = A_i^k V_{i-1}^k + (1 + B_i^k)V_i^k + C_i^k V_{i+1}^k \tag{14}$$

where

$$A_i^k = \frac{\delta t}{\delta S^2}a_i^k - \frac{1}{2}\frac{\delta t}{\delta S}b_i^k \tag{15}$$

$$B_i^k = -2\frac{\delta t}{\delta S^2}a_i^k + \delta t c_i^k \tag{16}$$

$$C_i^k = \frac{\delta t}{\delta S^2}a_i^k + \frac{1}{2}\frac{\delta t}{\delta S}b_i^k \tag{17}$$

5 Monte Carlo Method

Using the Monte-Carlo method is even easier. We know that the value of an option (at least in the happy world of Black and Scholes) is the present value of the expected payoff at expiry under a risk-neutral random walk. The risk neutral random walk for S is:

$$dS = rSdt + \sigma SdX \tag{18}$$

therefore

$$V(S,t) = e^{r(T-t)}E[payoff(S,T)] \tag{19}$$

But using the log-normal random walk we can write

$$d(logS) = (r - \frac{1}{2}\sigma^2)dt + \sigma dX \tag{20}$$

which cnan be integrated to

$$S(t+\delta t) = S(t)\exp((r - \frac{1}{2}\sigma^2)t + \sigma\sqrt{\delta t}\phi \tag{21}$$

6 The Java Programs

```
public class Main {

    public Main() {
        double asset = 100;
        double strike = 105;
        double drift = 0.05;
        double volatility = 0.2;
        double time = 1.0;
        double interestRate = 0.05;

        Calc calc = new Calc(asset, strike, drift, volatility, time,
                interestRate);
        System.out.println("formula: " + calc.getBlackScholesSolution());
        System.out.println("finit diff: " + calc.getFiniteDifference());
        System.out.println("monte carlo: " + calc.getMonteCarlo());
    }

    public static void main(String[] args) {
```

```
        new Main();
    }
}



public class Calc {

    double assetValue, strikeValue, mu, sig2, timeToExpiry, r;

    public Calc(double asset, double strike, double drift, double volatility,
            double time, double interest) {
        assetValue = asset;
        strikeValue = strike;
        mu = drift;
        sig2 = volatility;
        timeToExpiry = time;
        r = interest;
    }

    public double getFiniteDifference() {
        int noAssetSteps = 100;

        double[] oldValue = new double[noAssetSteps];
        double[] newValue = new double[noAssetSteps];
        double[] delta = new double[noAssetSteps];
        double[] gamma = new double[noAssetSteps];
        double[] asset = new double[noAssetSteps];

        double assetstep = 2 * strikeValue / noAssetSteps;
        int nearestGridPoint = (int) (assetValue / assetstep);
        double dummy = (assetValue - nearestGridPoint * assetstep) / assetstep;
        double timestep = assetstep * assetstep
                / (sig2 * 4 * strikeValue * strikeValue);
        int noTimeSteps = (int) (timeToExpiry / timestep) + 1;
        timestep = timeToExpiry / noTimeSteps;

        for (int i = 0; i < noAssetSteps; i++) {
            asset[i] = i * assetstep;
            oldValue[i] = Math.max(asset[i] - strikeValue, 0);
```

```
    }

    for (int j = 1; j < noTimeSteps; j++) {

        for (int i = 1; i < noAssetSteps - 1; i++) {
            delta[i] = (oldValue[i + 1] - oldValue[i - 1])
                    / (2 * assetstep);
            gamma[i] = (oldValue[i + 1] - 2 * oldValue[i] + oldValue[i - 1])
                    / (assetstep * assetstep);
            newValue[i] = oldValue[i]
                    + timestep
                    * (0.5 * sig2 * asset[i] * asset[i] * gamma[i] + r
                            * asset[i] * delta[i] - r * oldValue[i]);
        }
        newValue[0] = 0;
        newValue[noAssetSteps - 1] = 2 * newValue[noAssetSteps - 2]
                - newValue[noAssetSteps - 3];

        for (int i = 0; i < noAssetSteps; i++) {
            oldValue[i] = newValue[i];
        }
    }

    double value = (1 - dummy) * oldValue[nearestGridPoint] + dummy
            * oldValue[nearestGridPoint + 1];

    return value;
}

public double getMonteCarlo() {

    Random rand = new Random();

    int numTimeSteps = 100;
    int numSimulationRuns = 5000;
    double timestep = timeToExpiry / numTimeSteps;

    double sumOfValues = 0;
    for (int j = 0; j < numSimulationRuns; j++) {
        double asset = assetValue;
        for (int i = 0; i < numTimeSteps; i++) {
```

```
                asset = asset
                        * Math.exp((r - 0.5 * sig2) * timestep
                                + Math.sqrt(sig2 * timestep)
                                * rand.nextGaussian());
            }
            sumOfValues += Math.max(asset - strikeValue, 0);
        }

        double value = sumOfValues / numSimulationRuns
                * Math.exp(-r * timeToExpiry);

        return value;
    }

    public double getBlackScholesSolution() {
        double d1 = d1();
        double Nd1 = Probability.normal(0.0, 1.0, d1);
        double Nd2 = Probability.normal(0.0, 1.0, d2(d1));
        double v = assetValue * Nd1 - strikeValue * Math.exp(-r * timeToExpiry)
                * Nd2;

        return v;
    }

    public double d1() {
        double d1 = (Math.log(assetValue / strikeValue) + (r + 0.5 * sig2)
                * timeToExpiry)
                / (Math.sqrt(sig2 * timeToExpiry));

        return d1;
    }

    public double d2(double d1) {
        double d2 = d1 - Math.sqrt(sig2 * timeToExpiry);
        return d2;
    }
}
```