Chapter 1

OPTION PRICING WITH EXCEL

Peter Honoré

Nykredit Markets

Nykredit Bank A/S

Kalvebod Brygge 1-3

DK-1780 Copenhagen V, Denmark

pth@nykredit.dk

Rolf Poulsen

Department of Statistics and Operations Research

University of Copenhagen

Universitetsparken 5

DK-2100 Copenhagen Ø, Denmark

rolf@math.ku.dk

Abstract We use spreadsheets to illustrate the concepts and techniques of arbitrage-free option pricing.

Keywords: Excel, no arbitrage, option pricing, binomial model, Black-Scholes model, partial differential equation, finite difference method, hedging.

1. Introduction

We show how to use spreadsheets for financial modelling, or more specifically: How to do option pricing with Excel and Visual Basic for Applications.

Most authors of papers in this volume demonstrate the use of their favorite program (carefully chosen from a long list of peers) in a field where it is less commonly used. This paper differs on two accounts. First, spreadsheets are already widely used in both the teaching of finance and in the finance industry — and have been for some time. There's a saying in the finance community that "a closed-form solution is something you can program in Excel".[1] Second, "spreadsheet" nowadays (early 2002) largely means "Microsoft Excel" — and has done for some time. So there is no refined pro/con analysis of this vs. that spreadsheet. But UNIX/LINUX users[2] with the StarOffice package installed should not abandon the paper at this point. The spreadsheet in that package, StarCalc, and its programming environment, StarBasic, can do most things that Excel can. At least we successfully exported all programs used in this paper to StarCalc.

You need an "all-purpose"-program. One that has a battery of mathematical functions, is able to solve linear and not-too-difficult non-linear equations, has facilities for statistical analysis of data, can produce nice graphs, is endowed with a decent random number generator, offers the possibility of programming, especially a loop-structure, and so on. Hundreds of software packages posses such features, but none is more widespread than the spreadsheet Excel that comes with the Microsoft Office package, and thus is on most PCs. Spreadsheets are typically quite accessible and offer a "hands-on"-feeling; you change something and can see directly what happens. But Excel also offers programming features that allow us to deal with more complex models and analyze more sophisticated questions. This is through the Visual Basic for Applications (VBA) language/environment that — unknown to many, probably — comes with every Excel installation.

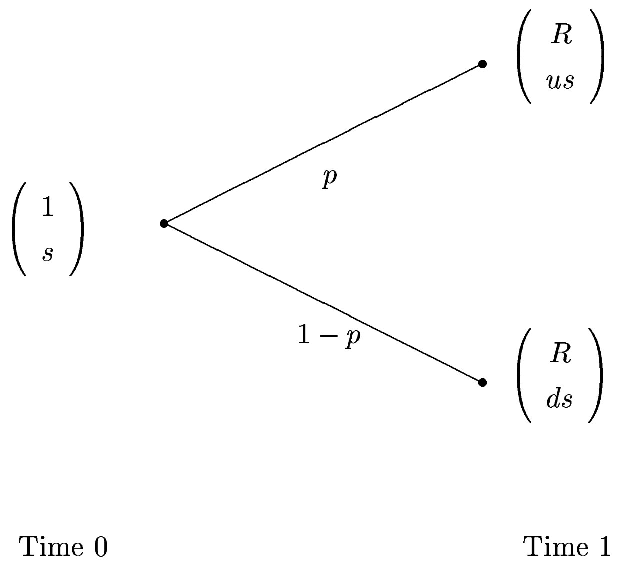
The rest of the paper goes as follows. In Section 2.1 we briefly review the principles of pricing by no arbitrage in binomial models, and show how this can be illustrated and implemented in Excel. Once we are confident with this, we do not need to see the whole model each and every time; we demonstrate how Excel's programming environment can be used to give us just what we want. This allows us to investigate the binomial model. We look at limiting behaviour, and in Section 2.2 that leads us to consider the Black-Scholes model, which is a continuous-time model. After a discussion of parameter estimation issues, we turn again to option pricing, where there are two numerical methods for pricing; Monte-Carlo simulation and solution of partial differential equations (PDEs), which — when it can be used — is typically the faster method. We show how to implement a finite difference PDE solution method. Simulation important too, and in Section 2.3 we use it to study a topic that has recently received some attention: The effects of less-than-perfect hedging, especially the effects from discrete hedging and from model misspecification. In the concluding Section 3 we have a short discussion of what we see as some of the advantages and disadvantages of spreadsheets in general and Excel in particular.

2. Financial Calculations: Models, Problems, & Solutions

2.1 The binomial model

The binomial model is the workhorse when it comes to illustrating the principles of "pricing by no arbitrage".[3] It was first presented in Cox, Ross and Rubinstein (1979) and the fundamental theorems of asset pricing were first formulated in Harrison and Kreps (1979). They are now text-book material; Hull (2000) is a classic in the field, and Pliska (1997) also gives a nice but more abstract treatment.

The building block is the one-period model with two future states of nature. It looks like this:


$\begin{pmatrix} R \\ us \end{pmatrix}$
p
$\begin{pmatrix} 1 \\ s \end{pmatrix}$
$1-p$
$\begin{pmatrix} R \\ ds \end{pmatrix}$
Time 0
Time 1

This model initially contains two assets that can be both bought and sold in any quantity by investors without transaction costs. Specifically, you can sell something you do not have (known as "taking a short position") if you just honor your future obligations to the buyer. The first asset is a *stock* with an initial price of $S(0) = s$. The future price of the stock is uncertain, stochastic; it can either (with probability p) go up to us, or it can go down to ds, where u and d are positive real numbers that are known at time 0. The other asset is a risk-free one; a bank account where an investment of \$1 grows to \$$R$ no matter what. We assume that $u > R > d$. Otherwise (if u and d are both greater than R) investors can borrow money, buy the stock and obtain a risk-free profit after one period. (And if u and d are both smaller than R, they can do the exact opposite.) This would be "a free lunch", or *an arbitrage opportunity.* So what we are assuming is that the model is arbitrage-free.

Now suppose that we introduce into the economy a *European call option* on the stock with exercise, or strike, price K and maturity after one period. This is a contract that gives the owner the right but not the obligation to buy the stock at time 1for the price K. So at time 1 the value of this call is (where x^+ means $\max(x,0)$)

$$C(1) = \begin{cases} (us-K)^+ := c_u \text{ in the up-state} \\ (ds-K)^+ := c_d \text{ in the down-state.} \end{cases}$$

What should the initial price of this call option be? A simple portfolio argument gives the answer: Using only the stock and the bank account we can form a portfolio at time 0 that gives the same pay-off as the call at time 1 regardless of which state occurs. Let (a,b) denote, respectively, the number of stocks and units of the bank account held at time 0 (so in this one-period model b is just how many \$ you have deposited or borrowed in the bank). If the pay-off at time 1 is to match that of the call we must have

$$a(us) + bR = c_u \quad \text{and} \quad a(ds) + bR = c_d.$$

These two equations have the solution

$$a = \frac{c_u - c_d}{s(u-d)}, \quad b = \frac{1}{R}\frac{uc_d - dc_u}{(u-d)}.$$

The number of stocks to invest in is often suggestively written as $a = \frac{\Delta C}{\Delta S}$ and is called the delta-hedge ratio, and we say that the portfolio hedges the call option. The cost of forming the portfolio (a,b) at time 0 is $aS(0) + b$. This is also the only possible initial price, say $C(0)$, of the introduced call option. Any other price would create an arbitrage opportunity: If the price were lower, we could buy the call and sell the replicating portfolio (a,b), receive cash now as a consequence and have no future obligations except to exercise the call if necessary. If the price were higher, the exact opposite could be done. So, $C(0) = aS(0) + b$, which after some simple algebra means:

$$C(0) = \left(\frac{R-d}{u-d}\right)\frac{c_u}{R} + \left(\frac{u-R}{u-d}\right)\frac{c_d}{R}.$$

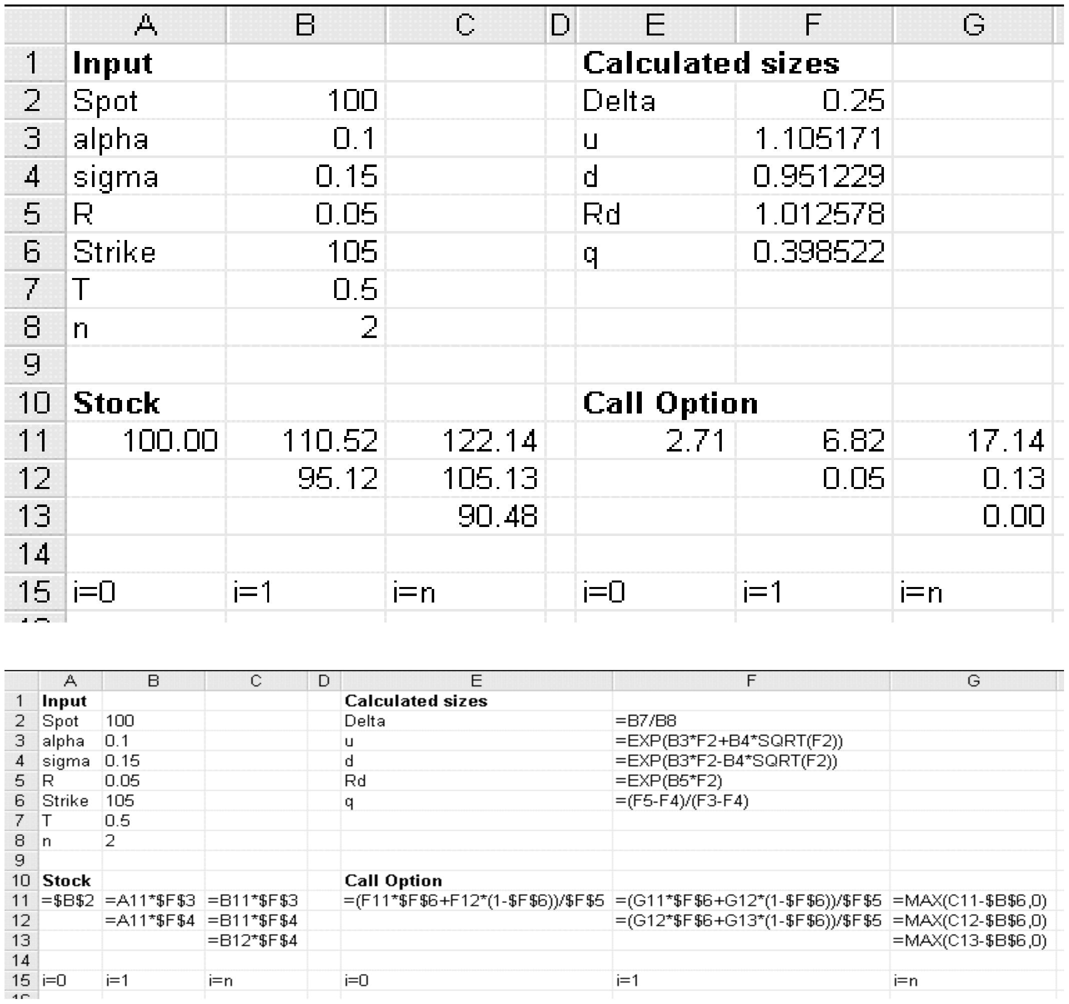
Now let $q = \frac{R-d}{u-d}$, and note that the assumption of no arbitrage means that $q \in]0,1[$, so q can be interpreted as a probability and we can write the call price as

$$C(0) = q\frac{c_u}{R} + (1-q)\frac{c_d}{R} = E^Q\left[\frac{C(1)}{R}\right],$$

i.e. as an expected value using q as probability (indicated by the Q-superscript) of the discounted time-1 value of the call. Note that the probability p plays no role in the expression for $C(0)$. Intuitively, this is because we take today's stock price as given and price the call-option relative to this. It is quite plausible that there is some relation between p, u, d and the price of the stock today, but we do not need it in the argument.

A stochastic process with the property that today's value is the expectation of tomorrow's value is called a *martingale*, and what we have shown in this example is that absence of arbitrage implies the existence of a set of probabilities such that the discounted call option price is a martingale when these probabilities are used. And it is not just for the call option that today's price is the Q-expected discounted value of the future price. It is also true for the bank account (trivially) and for the stock (just take $K = 0$). So absence of arbitrage implies the existence of martingale probabilities q and $1 - q$, or in probabilistic terms *an equivalent martingale measure* (EMM) Q. The converse is also true, and these two things combined is (a simple version of) what is known as the 1st fundamental theorem of asset pricing: "No arbitrage $\Leftrightarrow \exists$ EMM Q". Supposing that martingale probabilities exists, we may also verify by linear algebra that they are unique exactly in the case where any stochastic pay-off (not just that of the call option) may be replicated. This is called completeness and we have the 2nd fundamental theorem of asset pricing: "An arbitrage-free model is complete if and only if the EMM Q is unique."

There is a serious objection to this example: The perfect replication argument — especially the explicit expressions for probabilities, prices and hedge-ratios — is really the interesting bit and that breaks down if there are more than 2 future states of the world. With just 3 possible outcomes the model is incomplete: A replicating portfolio must solve 3 equations in 2 unknowns, which is typically impossible. And using a 2-point distribution as a model of stock prices a month, or a year, or ... from now is not very realistic. It would seem that to impose completeness we need to assume that there are as many different assets as there are future states of the world. But there is a different and much more realistic way to remedy things. By piecing together a (large) number of simple one-period models and — and this is the ingenious insight — allowing for dynamic re-adjustments of portfolios, we can make models where we still only need the stock and bank account in the replicating portfolio. Let us look at this in a spreadsheet. The advantage is that you can both see the model and use the program to perform calculations. This is done in Figure 1.1 where we first encounter Excel. The upper panel shows a 2-period binomial



	A	B	C	D	E	F	G
1	**Input**				**Calculated sizes**		
2	Spot	100			Delta	0.25	
3	alpha	0.1			u	1.105171	
4	sigma	0.15			d	0.951229	
5	R	0.05			Rd	1.012578	
6	Strike	105			q	0.398522	
7	T	0.5					
8	n	2					
9							
10	**Stock**				**Call Option**		
11	100.00	110.52	122.14		2.71	6.82	17.14
12		95.12	105.13			0.05	0.13
13			90.48				0.00
14							
15	i=0	i=1	i=n		i=0	i=1	i=n

	A	B	C	D	E	F	G
1	**Input**				**Calculated sizes**		
2	Spot	100			Delta	=B7/B8	
3	alpha	0.1			u	=EXP(B3*F2+B4*SQRT(F2))	
4	sigma	0.15			d	=EXP(B3*F2-B4*SQRT(F2))	
5	R	0.05			Rd	=EXP(B5*F2)	
6	Strike	105			q	=(F5-F4)/(F3-F4)	
7	T	0.5					
8	n	2					
9							
10	**Stock**				**Call Option**		
11	=$B$2	=A11*$F$3	=B11*$F$3		=(F11*$F$6+F12*(1-$F$6))/$F$5	=(G11*$F$6+G12*(1-$F$6))/$F$5	=MAX(C11-$B$6,0)
12		=A11*$F$4	=B11*$F$4			=(G12*$F$6+G13*(1-$F$6))/$F$5	=MAX(C12-$B$6,0)
13			=B12*$F$4				=MAX(C13-$B$6,0)
14							
15	i=0	i=1	i=n		i=0	i=1	i=n

Figure 1.1. A 2-period model in Excel. The upper panel shows the numbers as you see them in a worksheet, while the lower shows how the formulas are typed into Excel. (To display formulas instead of numbers click on `Tools` $\rightarrow$ `Options` $\rightarrow$ `View` and check the `Formulas`-box.) Formulas are created by first typing "="'and then some functions and references to cells. Note that "$"'s indicate absolute references, otherwise these are relative. So: When the formula in cell B12, "=A11*$F$4", is copied to cell C12 it becomes "=B11*$F$4", and when it is copied to C13 we get "=B12*$F$4".

model for the stock price. Today's price is 100 and at time 1 it can be either 110.52 or 95.12. Another "coin is tossed" at time 1 and the stock takes another move up or down (from its time-1 level). We have chosen a very specific parametrization of the up- and down-moves, namely

$$u = \exp(\alpha\Delta t + \sigma\sqrt{\Delta t}), \quad d = \exp(\alpha\Delta t - \sigma\sqrt{\Delta t}),$$

where α and σ (called the volatility) are constants. Assuming that $p = 1/2$, then with $\mu = \alpha + \sigma^2/2$, we have

$$\begin{aligned} E_t^P\left(\frac{S(t+\Delta)-S(t)}{S(t)}\right) &= \mu\Delta t + \mathrm{o}(\Delta t) \quad \text{and} \\ \mathrm{Var}_t^P\left(\frac{S(t+\Delta t)-S(t)}{S(t)}\right) &= \sigma^2\Delta t + \mathrm{o}(\Delta t), \end{aligned}$$

where E_t^P and Var_t^P denote, respectively, mean and variance conditional on the time-t information, and something is "o(Δt)" if it goes to 0 faster than Δt as Δt goes to 0. So σ^2 has natural interpretation as variance (per unit of time) of the stock's rate of return and μ determines the mean return. The model also has the huge advantage that the stock price development can be described by a recombining tree or a lattice; an "up-down" sequence of moves leads to the same stock price as a "down-up"-move. And now for the catch: If after one period we are in the "up"-state (cell B11), then there are only 2 possible future states and we can use our previous replication argument to conclude that a time-2 call option is worth 6.82 at time 1 if the stock goes up. Similarly, it is worth 0.05 if the stock goes down. Taking a step further back to time 0, we now know exactly which to possible time-1 values the call option can take. And the 1-period replication works again (and the arbitrage-free time-0 call option price is 2.71). Clearly, this backward recursive argument also works in 3-,4-, or n-period models, and prices can be found by working backwards through the lattice.

The underlying mathematics is that the fundamental theorems of asset pricing still hold, i.e. in arbitrage-free models we have that arbitrage-free prices at time t, $\pi(t)$, are given by

$$\pi(t) = E_t^Q(\text{discounted pay-off}) = E_t^Q\left(\frac{\pi(T)}{R^{T-t}}\right),$$

and if the equivalent martingale measure Q is unique then the model is complete, i.e. anything can be replicated. (This does not in any way hinge on the recombination.)

By piecing together many small models we can get a more realistic model for the distribution of the future stock price. However, looking at a, say, 100-period lattice in a spreadsheet is cumbersome and impractical. Excel has an environment, Visual Basic for Applications (VBA), for programming macros and user-defined functions. To get to the VBA editor click on `Tools` $\rightarrow$ `Macro` $\rightarrow$ `Visual Basic Editor` in the Excel menu-bar (or press `Alt-F11`). The commands are written in a so-called module (click on `Insert` $\rightarrow$ `Module` in the VBA menu-bar). The VBA editor also helps you debug the code, and you can then call it from the worksheet like any other function. There is an acceptable online help for VBA, but for the serious programmer a book such as Green, Bullen, and Martins (2000) is a must.

The backward recursive pricing method is easy to program and we show explicit VBA-code for the calculation of the price of a European call option in a binomial model. Note that we do not have to represent the stock price grid as a $(n+1)\times(n+1)$-matrix, but only as a $(n+1)$-vector, so we gain an order memory-requirement-wise.

```
Option Explicit
Option Base 0
Function bineurocall(S As Double, sigma As Double, alpha As Double, _
                     r As Double, Strike As Double, T As Double, _
                     N As Integer) As Double
  Dim u As Double, d As Double, q As Double, Rd As Double, dt As Double
  Dim vec() As Double

  dt = T / N
  Rd = Exp(r * dt)

  ' The up and down specifications
  u = Exp(alpha* dt + sigma * (dt ^ 0.5))
  d = Exp(alpha* dt - sigma * (dt ^ 0.5))

  ' Validate that u < Rd < d
  If (Rd > u) Or (Rd < d) Then
    Call MsgBox("Invalid input data", vbCritical)
    Exit Function
  End If

  ' Calculate the riskneutral probability.
   q = (Rd - d) / (u - d)

  ReDim vec(0 To N) As Double
  Dim i As Integer, j As Integer
  ' Initialise the final pay off.
  For j = 0 To N
    vec(j) = Application.Max(S * (u ^ j) * (d ^ (N - j)) - Strike, 0)
  Next j

  ' Roll backward
  For i = (N - 1) To 0 Step -1
    For j = 0 To i
      vec(j) = (vec(j + 1) * q + vec(j) * (1 - q)) / Rd
    Next j
  Next i

  bineurocall = vec(0)

End Function
```

The function can be called from Excel with valid arguments, e.g. "`=bineurocall(100,0.15,0.07,0.05,105,0.5,50)`" (this returns the value 3.204098) or with references to values in specific cells. When you change the value in an input

Quantity	Symbol	Numerical value
Initial stock price	$S(0)$	100
Volatility	σ	0.15
Interest rate	r	0.05
Strike/exercise price	K	105
Option maturity	T	0.5

Table 1.1. Default parameters. With the symmetry in the model there's no loss of generality in using 100 as initial stock price. Time is measured in years and the interest rate is continuously compounded, i.e. if you put \$1 in the bank, you get \$1*exp(0.05) = \$1.051271 back in a year. These parameters all have the same meaning in both discrete and continuous models. The P-expected rate of return, μ, (and hence α) plays no role for option pricing when portfolios that can be adjusted sufficiently frequently (and cheaply).

cell, then the `bineurocall`-function is automatically recalculated. At least that is the default setting; it may be changed by `Tools` $\rightarrow$ `Options` $\rightarrow$ `Calculation`.

As we said, there are good reasons for looking at models where the number of periods, n, is high (and step sizes are small). What happens in the extreme case, i.e. when $n \rightarrow \infty$? For the specific parametrization it is easy to investigate numerically, and in Figure 1.2 the result can be seen. It appears that the call-price converges. This is true, in fact it can be proved that the model converges (in an certain sense) to the Black-Scholes model where the stock price is lognormally distributed. This model we look at next section. But that's not all. More surprisingly, we see that the limiting value does not depend on μ, the P-expected rate of return on the stock. One way to understand this is to note that when we change to the probability measure Q (which we have to do for pricing) then all expected rates of return are converted to r. Hence "any μ is automatically turned into r". A closer inspection of the argument will show that some higher order terms are missing, but these do disappear in the "$\Delta t \rightarrow 0$"-limit. (Probabilists will see this as being intimately connected to Girsanov's theorem.) We also note that the convergence is quite oscillatory,which is a potential source for problems, for instance if we want to extrapolate. There are different ways to improve on this, see Klassen (2001).

Thus wiser, we can improve and extend the code. First, we cut out the μ-dependence, since it does not matter anyway for small time-steps. We simply use $\exp(\pm\sigma\sqrt{\Delta t})$ as up/down moves. In the literature this is known as the Cox-Ross-Rubinstein-choice. We can also handle other pay-offs (e.g. $(K - S(T))^+$, i.e. the put option), American type features (i.e. the possibility of early exercise) or to return the

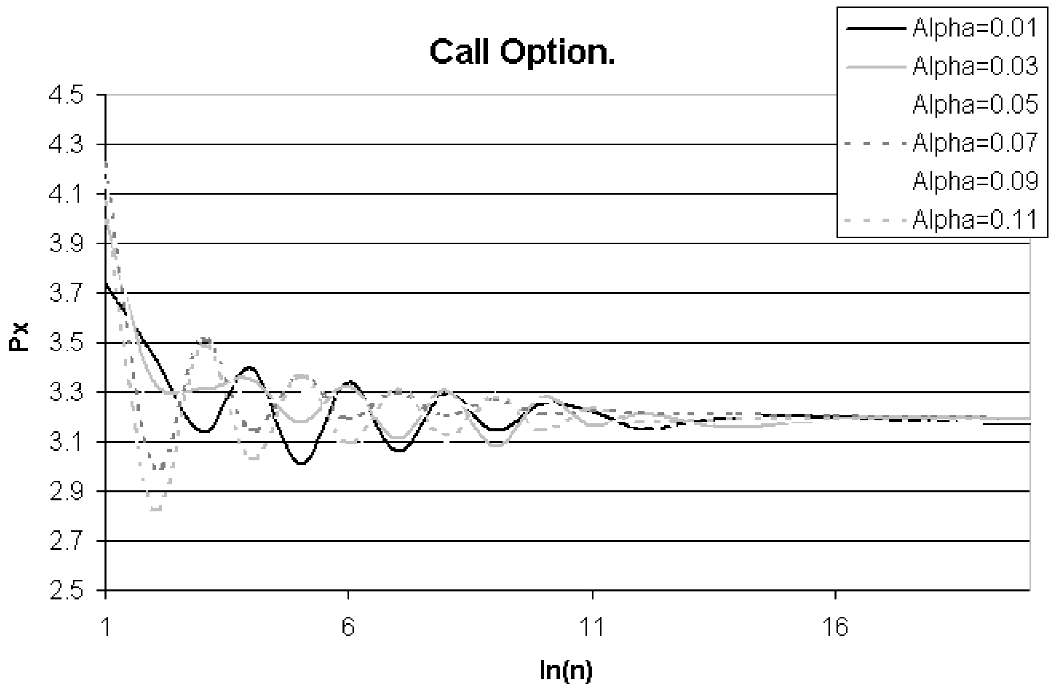


Figure 1.2. The graph shows the convergence (as the number of periods, n, increases) of the call-price in binomial model for different P-expected stock returns (as parametrized α) and otherwise default parameters as given in Table 1.1.

delta-hedge ratio rather than the price. We can also incorporate dividend payments from the stock, although this requires a bit more care, see Hull (2000)[Ch. 14] for instance.[4] Below we have modified the VBA-code such that put options, possibly of American type (American calls on non-dividend-paying stocks are never exercised prematurely, so they are not that interesting) can be priced. The essential modification is that at each node we compare what we get from exercising immediately ($(S(t) - K)^+$ for the call, $(K - S(t))^+$ for the put; this is called the intrinsic value) to the value of holding on to the option. The holding value (aka the time value) is found as discounted Q-expectation of next period's outcomes, and since we are working backwards, we know these. The American option price at the node is then the maximum of the time value and the intrinsic value.

```
' Function calculating option price based on a binomial model.
' Explanation to inputs:
'   opt_type: call/put and _ european/american
' where the default values are the first possibilities.
'
Function bin(S As Double, sigma As Double, r As Double, _
            Strike As Double, T As Double, N As Integer, _
            opt_type As String) As Double
```

```
  Dim u As Double, d As Double, q As Double, Rd As Double, dt As Double
  Dim grid() As Double

  dt = T / N
  Rd = Exp(r * dt)

  ' The up and down specifications
  u = Exp(sigma * (dt ^ 0.5))
  d = Exp(- sigma * (dt ^ 0.5))

  ' Validate that u < Rd < d
  If (Rd > u) Or (Rd < d) Then
    Call MsgBox("Invalid input data", vbCritical)
    Exit Function
  End If

  ' Calculate the riskneutral probability.
   q = (Rd - d) / (u - d)

  ReDim vec(0 To N) As Double
  Dim i As Integer, j As Integer
  ' Initialise the final pay off.
  For j = 0 To N
    vec(j) = pay_off(S * (u ^ j) * (d ^ (N - j)), Strike, opt_type)
  Next j

  ' Roll backward
  For i = (N - 1) To 0 Step -1
    For j = 0 To i
      vec(j) = (vec(j + 1) * q + vec(j) * (1 - q)) / Rd
    Next j

    ' The case with options of the american type
    If UCase(opt_type) Like "*_AMERICAN*" Then
      For j = 0 To i
        vec(j) = Application.Max(vec(j), _
                 pay_off(S * (u ^ j) * (d ^ (i - j)), Strike, opt_type))
      Next j
    End If
  Next i

  ' return the price.
  bin = vec(0)
End Function

' Function for different pay offs.
' Default call option pay-off.
'
Private Function pay_off(S As Double, Strike As Double, opt As String) As Double
  If UCase(opt) Like "PUT*" Then
    pay_off = Application.Max(Strike - S, 0)
  Else
    ' Default
    pay_off = Application.Max(S - Strike, 0)
  End If
End Function
```

2.2 The Black-Scholes model

We now turn to continuous-time financial models, a subject on which Björk (1998), Musiela and Rutkowski (1997) and Duffie (2001) are all recommendable textbooks. The benchmark continuous-time model is the Black-Scholes model. Here the stock price follows a Geometric Brownian motion,

$$dS(t) = \mu S(t)dt + \sigma S(t)dW^P$$

where W^P is a Brownian motion under the "real world"-probability measure P, and similarly to the discrete case μ and σ (the volatility) determine mean and variance of the stock's rate of return

$$\begin{aligned} E_t^P\left(\frac{S(t+\Delta t)-S(t)}{S(t)}\right) &= \mu\Delta t + \mathrm{o}(\Delta t) \quad \text{and} \\ \mathrm{Var}_t^P\left(\frac{S(t+\Delta t)-S(t)}{S(t)}\right) &= \sigma^2\Delta t + \mathrm{o}(\Delta t). \end{aligned}$$

The model also contains a bank-account with a deterministic interest rate, i.e. an asset whose price develops as $\beta(t) = \exp(rt)$ (often written as $d\beta(t) = r\beta(t)dt$).

A Digression on Statistics & Macros

Before turning to the financial theory in the Black-Scholes model, let us look at parameter estimation from data. Suppose we have equidistant observations (Δt apart) of stock prices at times $t_0, \dots, t_n$. In our numerical example we have daily observations (so $\Delta t = 1/250$)[5] of the Microsoft stock price between December 1996 and December 2000 (a little more than 1,000 observations). The time series is shown in Figure 1.3.

To estimate, let us make the distributional results more precise than first order approximation to mean and variance. It can be shown (with the key result of stochastic calculus known as Ito's formula or lemma) that log-returns are independent and normally distributed. Specifically, if we put

$$r_i = \ln\left(\frac{S(t_i)}{S(t_{i-1})}\right),$$

then the r_i's are independent and

$$r_i \sim N((\mu - \sigma^2/2)\Delta t, \sigma^2\Delta t) \quad \text{for all } i.$$

So we transform observed stock-prices to log-returns, put $\alpha = (\mu - \sigma^2/2)$, look (at least formally) at the likelihood function

$$\prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi\Delta t}} \exp\left(-\frac{(r_i - \alpha\Delta t)^2}{2\sigma^2\Delta t}\right),$$

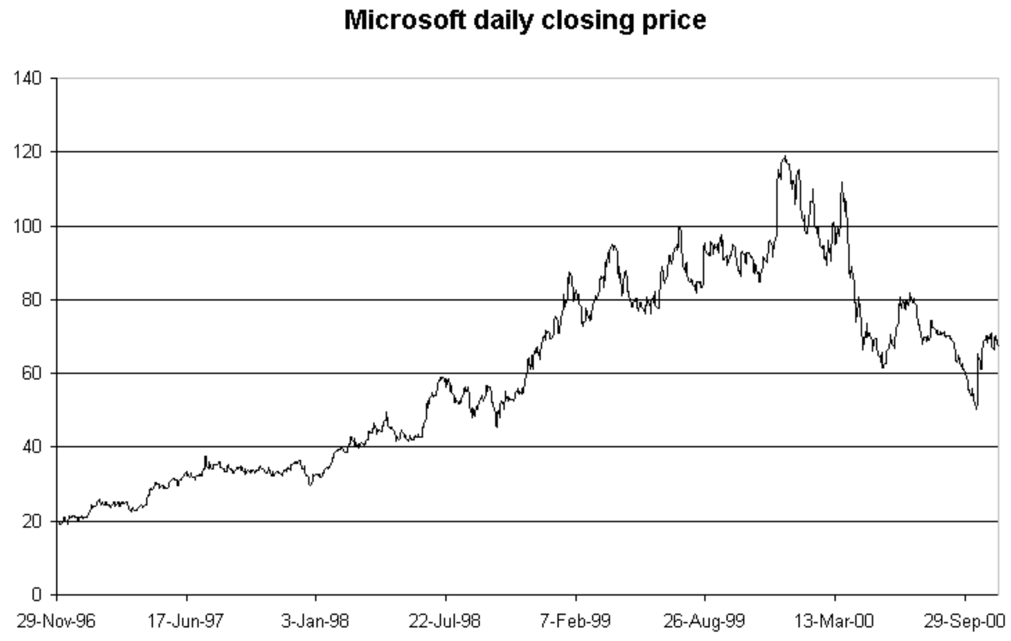


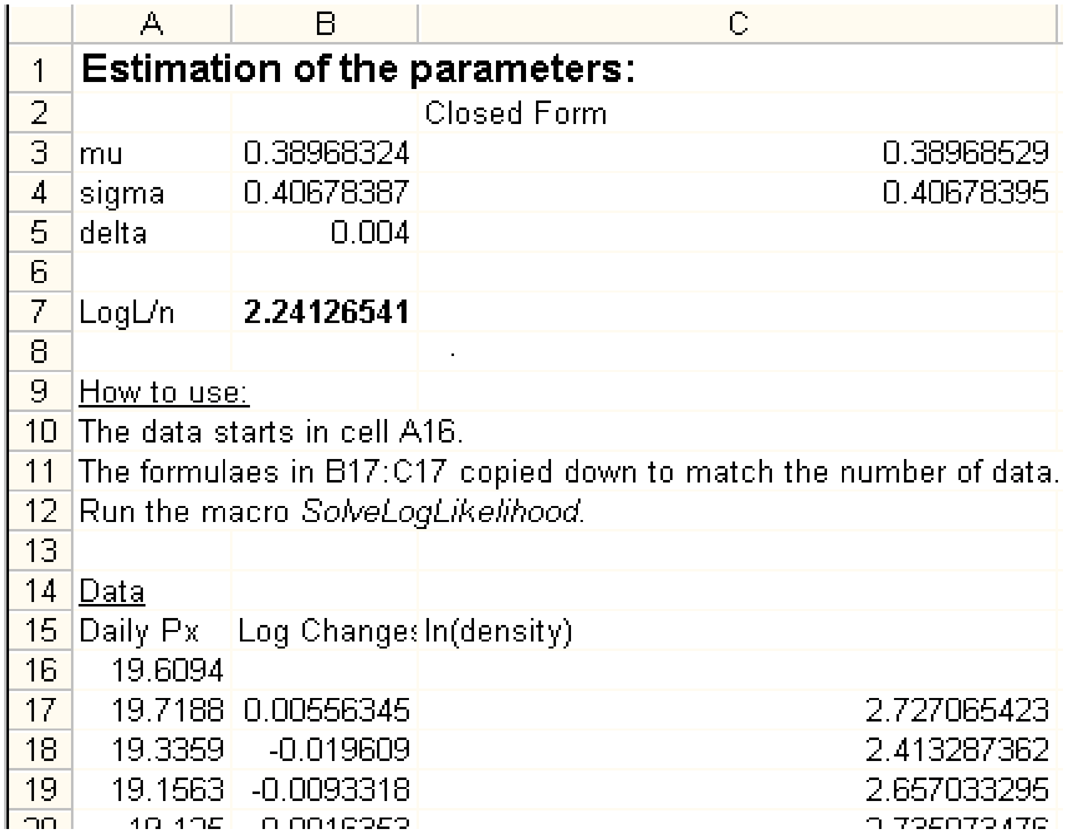
Figure 1.3. Microsoft's stock prices between 1997 and 2000. (Source: Bloomberg.)

and choose as estimators the values of α and σ that maximize this expression (or its logarithm; this is theoretically equivalent and numerically convenient). This is of course completely standard and we know that the estimators are given explicitly by the formulas

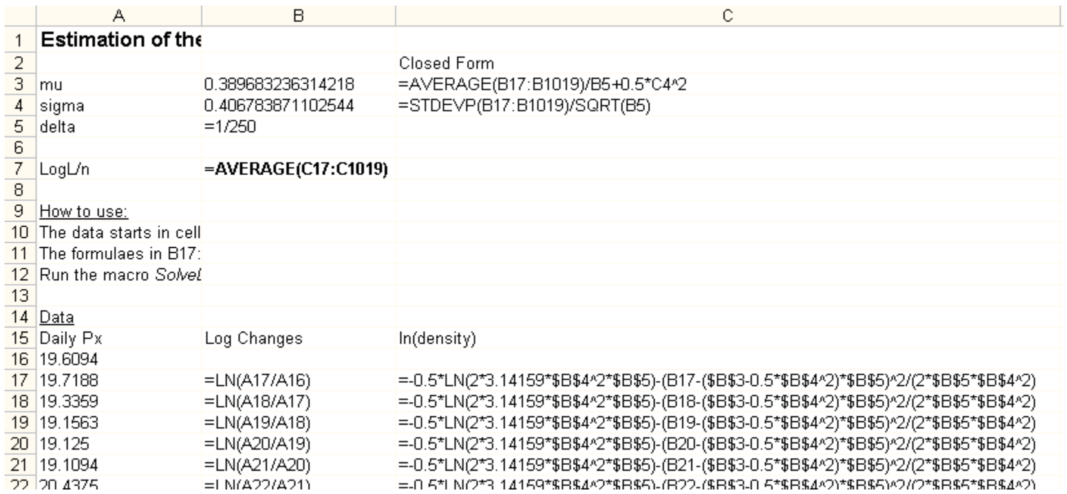
$$\widehat{\alpha} = \frac{1}{n\Delta t}\sum_{i=1}^{n} r_i, \quad \widehat{\sigma} = \sqrt{\frac{1}{n\Delta t}\sum_{i=1}^{n}(r_i - \widehat{\alpha}\Delta t)^2}\ , \text{ and } \widehat{\mu} = \widehat{\alpha} + \widehat{\sigma}^2/2.$$

In other words we just have to calculate means and variances (and remember the scaling factor Δt). Excel has build-in functions, `AVERAGE` and `STDEV/STDEVP`, for this.[6]

One reason we have written down the full likelihood function is that it gives us a chance to illustrate Excel's optimization routine `Solver`. Furthermore, the (numerical) maximum likelihood approach still works for parameter estimation in models with more advanced stock price dynamics. `Solver` may not initially be installed on your computer. Then you need to click `Tools` $\rightarrow$ `Add-Ins`, check the box with `Solver Add-in` and click `OK`. To use `Solver` we set up a cell (B7 in the sheet shown in Figure 1.4) with a formula that calculates our criterion function. In this case this is the log-likelihood function, so the formula will have references



	A	B	C
1	**Estimation of the parameters:**		
2			Closed Form
3	mu	0.38968324	0.38968529
4	sigma	0.40678387	0.40678395
5	delta	0.004	
6			
7	LogL/n	**2.24126541**	
8			
9	How to use:		
10	The data starts in cell A16.		
11	The formulaes in B17:C17 copied down to match the number of data.		
12	Run the macro *SolveLogLikelihood*.		
13			
14	Data		
15	Daily Px	Log Change	ln(density)
16	19.6094		
17	19.7188	0.00556345	2.727065423
18	19.3359	-0.019609	2.413287362
19	19.1563	-0.0093318	2.657033295
20	19.125	0.0016353	2.735073476



	A	B	C
1	**Estimation of the**		
2			Closed Form
3	mu	0.389683236314218	=AVERAGE(B17:B1019)/B5+0.5*C4^2
4	sigma	0.406783871102544	=STDEVP(B17:B1019)/SQRT(B5)
5	delta	=1/250	
6			
7	LogL/n	**=AVERAGE(C17:C1019)**	
8			
9	How to use:		
10	The data starts in cell		
11	The formulaes in B17:		
12	Run the macro *SolveL*		
13			
14	Data		
15	Daily Px	Log Changes	ln(density)
16	19.6094		
17	19.7188	=LN(A17/A16)	=-0.5*LN(2*3.14159*$B$4^2*$B$5)-(B17-($B$3-0.5*$B$4^2)*$B$5)^2/(2*$B$5*$B$4^2)
18	19.3359	=LN(A18/A17)	=-0.5*LN(2*3.14159*$B$4^2*$B$5)-(B18-($B$3-0.5*$B$4^2)*$B$5)^2/(2*$B$5*$B$4^2)
19	19.1563	=LN(A19/A18)	=-0.5*LN(2*3.14159*$B$4^2*$B$5)-(B19-($B$3-0.5*$B$4^2)*$B$5)^2/(2*$B$5*$B$4^2)
20	19.125	=LN(A20/A19)	=-0.5*LN(2*3.14159*$B$4^2*$B$5)-(B20-($B$3-0.5*$B$4^2)*$B$5)^2/(2*$B$5*$B$4^2)
21	19.1094	=LN(A21/A20)	=-0.5*LN(2*3.14159*$B$4^2*$B$5)-(B21-($B$3-0.5*$B$4^2)*$B$5)^2/(2*$B$5*$B$4^2)
22	20.4375	=LN(A22/A21)	=-0.5*LN(2*3.14159*$B$4^2*$B$5)-(B22-($B$3-0.5*$B$4^2)*$B$5)^2/(2*$B$5*$B$4^2)

Figure 1.4. Sheet for maximum likelihood estimation (on Microsoft stock prices) of parameters in the Black-Scholes model. Note that the estimates found numerically with `Solver` are the same as the closed-form ones.

to both data and parameters (our initial parameter-guesses have be entered in two separate cells, B3 and B4 in this case). For numerical stability it is advisable to normalize the log-likelihood function by the number of observations, since it is then safe to use the same absolute termination criterion in the optimization procedure irrespective of whether you have 100 or 100,000 observations. To optimize, click `Tools` → `Solver`, specify the formula cell as `target cell`, choose `Max`, tell `Solver` that the maximization is to be done by changing the two cells with the parameter guesses, and click `OK`. Figure 1.4 shows how this looks in Excel. We see that the closed-form expressions and the numerical optimization give the same results (to an accuracy of about 10^{-7}), as of course they must if the procedures work properly. We see that the Microsoft stock has had an annual expected growth rate of around 40%, and a volatility also around 40%. (Qualitatively, both these numbers are quite high. And it is a coincidence that there are about equal.)

An important thing to know about `Solver` is that a recalculation does not automatically take place when you change one or more cells in the spreadsheet. If we decided that "a year is 255 business days", we would (in principle) have to click `Tools` → `...` again to re-optimize. But there is a way to avoid such long series of key-strokes/mouse-clicks: Macros. A macro is simply a recording of a series of key-strokes. To record one, click `Tools` → `Macro` → `Record New Macro`, give the macro a name and click `OK`. Now do exactly as you did the first time you wanted to optimize, and (immediately) after that click `Tools` → `Macro` → `Stop Recording`. You now have the possibility to redo the optimization again and again with only a few clicks: `Tools` → `Macro` → `Macros`, choose it macro from the list and `run`. If you're really fond of the macro you can give a shortcut (`CTRL`-"something"), "iconize" it and put it in the tool-bar or in a button on the sheet.

A quite smart thing is that the macro is generated as commands in VBA code. For instance when we recorded the macro described above (and called it `SolveLogLikelihood`) the following code was generated:

```
Sub SolveLogLikelihood()
    Range("B7").Select
    SolverOk SetCell:="$B$7", MaxMinVal:=1, ValueOf:="0", ByChange:="$B$3:$B$4"
    SolverSolve
End Sub
```

Recorded macro-code easily becomes quite messy. For instance: Cell B7 with the formula was highlighted before `Solver` was called; this generates the superfluous line `Range("B7").Select`. And of course the code is un-annotated. So it's advisable to clean up your code immediately. Doing this, we arrived at the following ("call" just indicates then a procedure is being called)

```
' Macro used to calculate the maximum likelihood estimates.
' Assumption:
'   i) the log-likelihood function in cell B7.
'   ii) the parameters in the range B3:B4.
'
' To run this macro be on the relevant sheet.
Sub SolveLogLikelihood()
  Call SolverOk(SetCell:="$B$7", MaxMinVal:=1, ByChange:="$B$3:$B$4")
  Call SolverSolve
End Sub
```

Macros can change objects in the spreadsheet (e.g. overwrite cells) whereas a user-defined *function* can only return a specific value (a number/vector). This is something of a double-edged sword. Some things are (almost) impossible to do with functions but easy with macros, while on the other hand when things go wrong with macros, they can go horribly wrong.

But Back to the Option Pricing ...
After this digression on statistics, let us get back to the option pricing in the Black-Scholes model. The fundamental theorems of asset pricing still hold (at least with some extra technical requirements). Therefore "no arbitrage" means that the arbitrage-free prices at time t, $\pi(t)$, are given by

$$\pi(t) = E_t^Q(\text{discounted pay-off}),$$

where Q is a probability measure such that the discounted stock price, S/β, is a martingale. A result from stochastic calculus (Girsanov's theorem) tells us that the Q-dynamics of S must be

$$dS(t) = rS(t)dt + \sigma S(t)dW^Q,$$

where W^Q is a Q-Brownian motion that is (in some sense) unique because the model is complete. The representation of prices as expected values suggests using simulation for calculating prices by appealing to the law of large numbers. This works for many different types of contracts (though not for American-type options without *considerable* sleight of hand, see Fu, Laprise, Madan, Su and Wu (2001)) and is conceptually pretty straightforward. We will not look at that now. In the next section we tell you "how to" when it comes to simulation in Excel, but use it in a more advanced context. Rather, we focus at a more cunning approach that involves solving partial differential equations numerically. A much more thorough treatment of this is given in Morton and Mayers (1994), and Wilmott (1998) shows how many, many problems in finance can be attacked and solved with a PDE approach. In the Black-Scholes model it can be shown (the Feynman-Kac representation) that a contract whose terminal pay-off depends only on the terminal stock price, say through the function h, has a time-t price of the form $\pi(t) = F(t, S(t))$, where F is a deterministic function that solves the PDE

$$F_t + rxF_x + \frac{1}{2}x^2\sigma^2 F_{xx} = rF \text{ for } t < T, \tag{1.1}$$

with the terminal condition $F(T, x) = h(x)$. For the call option ($h(x) = (x - K)^+$), the PDE can be solved in closed form to give the Black-Scholes formula (Black and Scholes (1973))

$$BS^{call}(t) = S(t)\Phi(z_+) - e^{-r(T-t)}K\Phi(z_-),$$

where $z_\pm = \left(\ln(\frac{S(t)}{K}) + (r \pm \frac{\sigma^2}{2})(T-t)\right) / \sigma\sqrt{T-t}$ and Φ is the cumulative density of the standard normal distribution. Below the Black-Scholes formula is shown in VBA-code. The function can return put price or the delta-hedge ratio if the flag `out_type` is set appropriately.

```
' Black Scholes formula.
' Put or Call price/delta.
' The default is the price of a call option.
'
Function BS(S As Double, sigma As Double, r As Double, Strike As Double, _
            T As Double, Optional opt_type = "Call", _
            Optional out_type = "Price") As Double
  Dim d1 As Double, nd1 As Double, nd2 As Double
  d1 = (Application.Ln(S / Strike) + (r + sigma ^ 2 * T) / 2) / (sigma * T ^ 0.5)
  nd1 = Application.NormDist(d1, 0, 1, True)
  nd2 = Application.NormDist(d1 - sigma * T ^ 0.5, 0, 1, True)
  If UCase(opt_type) Like "PUT*" Then
    If UCase(out_type) Like "DELTA*" Then
      BS = nd1 - 1
    Else
      BS = S * (nd1 - 1) - Strike * Exp(-r * T) * (nd2 - 1)
    End If
  Else
    If UCase(out_type) Like "DELTA*" Then
      BS = nd1
    Else
      BS = S * nd1 - Strike * Exp(-r * T) * nd2
    End If
  End If
End Function
```

Finite Difference Methods

Often it is not possible to find closed-form solutions to the PDEs we encounter. So we need to look for numerical solution techniques, which is what we do next. The Black-Scholes call-price formula will be our test-case.

Consider a PDE of the slightly more general form (it's known as a linear parabolic PDE)

$$F_t = \mathcal{L}F, \tag{1.2}$$

where $\mathcal{L}$ is a differential operator that maps a function f into the function $\mathcal{L}f$, where $(\mathcal{L}f)(x) = rf(x) - a(x)f_x(x) - \frac{b^2(x)}{2}f_{xx}(x)$, where $a(\cdot)$ and $b(\cdot)$ are known functions (and it is implicitly understood that $\mathcal{L}$ on the right hand side (RHS) of (1.2) works on the second argument of F). The pricing problem is to determine the time 0 value of a contingent claim with final maturity T. The idea with finite difference is to discretize (1.2) by dividing the (x, t) plane into a uniformly spaced mesh, or grid, with $N + 1$ discrete points in the time dimension and $M + 1$ points in the x space:

$$x_i = x_0 + i\Delta x \quad \text{and } t_j = j\Delta t.$$

A probabilistic argument can be used for setting appropriate values for the lower and upper values of x. Set x_0 and x_M such that a certain confidence interval of the state space in x is reached given an initial value for x. Let f_i^j denote the value of a function on the grid at node (x_i, t_j). We then approximate the differential operators in (1.2) with difference operators on the grid. While F_t is naturally approximated by $(f_i^{j+1} - f_i^j)/\Delta t$, there are several — non-equivalent, so it turns out — ways to approximate the operator $\mathcal{L}$; one with 'a degree of freedom' is to use the following on the interior of the grid:

$$\begin{aligned}\mathcal{L}F \quad &\approx \quad (1-\theta)\left(rf_i^j - a(x_i)\frac{f_{i+1}^j - f_{i-1}^j}{2\Delta x} - \frac{b^2(x_i)}{2}\frac{f_{i+1}^j - 2f_i^j + f_{i-1}^j}{(\Delta x)^2}\right)\\ &+ \quad \theta\left(rf_i^{j+1} - a(x_i)\frac{f_{i+1}^{j+1} - f_{i-1}^{j+1}}{2\Delta x} - \frac{b^2(x_i)}{2}\frac{f_{i+1}^{j+1} - 2f_i^{j+1} + f_{i-1}^{j+1}}{(\Delta x)^2}\right),\end{aligned}$$

where θ is a parameter between 0 and 1. If $\theta = 1$ the resulting scheme is the so-called explicit (trinomial) finite difference method, whereas we have the fully implicit finite difference method for $\theta = 0$. The Crank-Nicolson scheme is a sort of average of the first two schemes with $\theta = 1/2$. It is only when we use $\theta \in [0; 1/2]$ that the numerical solution converges without further restrictions on the relation between time- and space-stepsizes. (We refrain from illustrating this in our examples.)

Let us consider the log-transformed Black-Scholes PDE, i.e. the PDE that the function defined by $g(x) := F(e^x)$ solves. This means that $a(x) = r - \sigma^2/2$ and $b(x) = \sigma$, and explains in part why we use the transform: We want as many things as possible to be constant. To solve the discretized system we get by plugging our approximations into (1.2) — and thus get an approximation to F and ultimately the time-0 contingent claim price — we have to solve a sequence of N systems of linear equations of the form

$$Af^j = d^{j+1} \quad \text{for } j = N-1, \dots, 0, \tag{1.3}$$

where A is an $(M+1) \times (M+1)$-matrix, $f^j = (f_0^j, f_1^j, \dots, f_M^j)^\top$ and the $(M+1)$-vectors d^{j+1} depend only on quantities with time-index $j+1$ (or greater). The matrix A is sparse; all its entries except those we mention below are 0. Calculations that are more tedious than hard give us the non-zero entries of A and the RHSs. We find

$$\begin{aligned}
a_{i,i-1} &= \frac{1-\theta}{2\Delta x}\left((r-\sigma^2/2) - \frac{\sigma^2}{\Delta x}\right) \equiv a_{-1}, \\
a_{i,i} &= \frac{1}{\Delta t} + (1-\theta)\left(r + \frac{\sigma^2}{(\Delta x)^2}\right) \equiv a_0, \quad \text{and} \\
a_{i,i+1} &= \frac{1-\theta}{2\Delta x}\left(-(r-\sigma^2/2) - \frac{\sigma^2}{\Delta x}\right) \equiv a_1 \quad \text{for } i = 1, \dots, M-1,
\end{aligned}$$

Note that the the a's are the same for all time- and space-points as well as for all option types. The "inner point" RHSs are given by

$$\begin{aligned}
d_i^{j+1} &= -\frac{\theta}{2\Delta x}\left((r-\sigma^2/2) - \frac{\sigma^2}{\Delta x}\right) f_{i-1}^{j+1} + \left(\frac{1}{\Delta t} - \theta\left(r + \frac{\sigma^2}{(\Delta x)^2}\right)\right) f_i^{j+1} \\
&+ \frac{\theta}{2\Delta x}\left((r-\sigma^2/2) + \frac{\sigma^2}{\Delta x}\right) f_{i+1}^{j+1} \quad \text{i.e. for } i = 1, \dots, M-1,
\end{aligned}$$

and these do depend of the type of option considered. It is not immediately obvious what we should do on the upper and lower boundaries. We suggest the following explicit boundary conditions. For the call option on the upper boundary use the approximation $F_S \approx 1$, which in the log-transformed world means: $g_x(X_M) = S_0 \exp(M\Delta x)$, and $F_{SS} = 0$. The reason for this is that if the stock price is high, the $(\cdot)^+$ does not matter very much and then $F(s,t) = e^{-r(T-t)}E^Q((S(T)-K)^+|S_t = s) \approx e^{-r(T-t)}E^Q((S(T)-K)|S_t = s) = s - e^{-r(T-t)}K$. Similar arguments can be used at the low boundary and for put options, and in compact notation we end up with $F_{SS}(\text{"any bd."}) = 0$ and

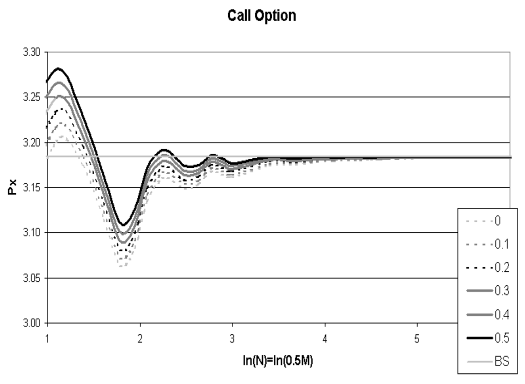
$$F_S(\text{"upper bd."}) = \mathbf{1}_{\{\text{option = call}\}}, \text{ and } F_S(\text{"lower bd."}) = -\mathbf{1}_{\{\text{option = put}\}},$$

where $\mathbf{1}_{\{\cdot\}}$ denotes the indicator function. It is possible to use so-called implied boundary conditions (see Vetzal (1998)), where derivatives at the boundary are estimated using only points on one side of the boundary . However, this can make the solution procedure unstable if drift does not dominate volatility (i.e. $|a/b|$ is large) at the boundaries. For the Black-Scholes model the drift does not dominate. The boundary elements of the equation systems for call and put options become

$$\begin{aligned}
a_{0,1} &= a_{M,M-1} = 0, \\
a_{0,0} &= a_{M,M} = \frac{1}{dt} + r(1-\theta), \\
d_0^{j+1} &= \left(\frac{1}{\Delta t} - r\theta\right) f_0^{j+1} + \mathbf{1}_{\{\text{option = put}\}}(r - \sigma^2/2) S_0 \exp(x_0), \text{ and} \\
d_M^{j+1} &= \left(\frac{1}{\Delta t} - r\theta\right) f_M^{j+1} - \mathbf{1}_{\{\text{option = call}\}}(r - \sigma^2/2) S_0 \exp(x_0 + M\Delta x).
\end{aligned}$$

Since A is tridiagonal each of the systems in (1.3) can be solved with a computational effort that grows only linearly in the size of A. Now we just have to roll backward. Remember the final pay-off of the contingent claim is the initial condition for f^N. Just as with the binomial model we can work with American-type features and other pay-off structures, as it is done in the following VBA-code. We do not explicitly show the algorithm, `SolveLinearSystem`, for solving tridiagonal linear systems, but refer the reader to Press, Teukolsky, Vetterling and Flannery (1993), for instance. It is possible to use the build-in Excel functions `MINVERSE` and `MMULT` for the matrix calculations. But this is much slower because the tridiagonal structure is not exploited.

```
Option Explicit
Option Base 0
' Function calculating option price based on a Black-Scholes model.
' Explanation to inputs:
'   opt_type:  call/put and _ european/american
' where the default values are the first possibilities.
'
Function FD(S As Double, sigma As Double, r As Double, _
           K As Double, t As Double, N As Integer, M As Integer, _
           opt_type As String, Optional theta = 0.5) As Double
  Dim a() As Double, f() As Double, d() As Double, dx As Double, dt As Double
  Dim i As Integer, j As Integer
  ' The increments in the grid.
  dx = 6 * sigma * (t ^0.5) / M: dt = t / N
  ' Set-up the a coefficients.
  ReDim a(-M To M, -1 To 1)
  Dim s2 As Double, s2f As Double, rs As Double
  s2 = sigma ^2:  s2f = s2 / dx:  rs = r - s2 / 2
  For i = 1 - M To M - 1
    a(i, -1) = ((1 - theta) / (2 * dx)) * (rs - s2f)
    a(i, 0) = 1 / dt + (1 - theta) * (r + (sigma / dx) ^2)
```



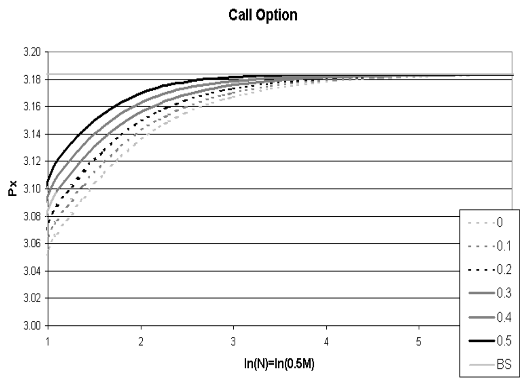


Figure 1.5. Convergence of finite difference solutions to call-price PDEs for different θ-values and with default parameters as given in Table 1.1. Note that we keep the number of time and space steps proportional, as is optimal for methods that have the same convergence order in the two directions. For the graph on the left we used the "raw" pay-off function as terminal condition, while we smoothed the pay-off over the last time-step with the Black-Scholes formula to produce the graph on the right. This makes quite a difference; the convergence in the smoothed case is faster and "nicer" in the sense that it is monotone. Notice also the clearly superior accuracy of the Crank-Nicolson method ($\theta = 1/2$) in the smoothed case.

```
  a(i, 1) = ((1 - theta) / (2 * dx)) * (-rs - s2f)
Next i
a(-M, 0) = (1 / dt) + (1 - theta) * r
a(-M, 1) = 0
a(M, -1) = 0
a(M, 0) = (1 / dt) + (1 - theta) * r
' The initial pay-off:
ReDim f(-M To M)
For i = -M To M
  f(i) = pay_off(S * Exp(i * dx), K, opt_type)
Next i
' Option type used for the explicit boundaries:
Dim isPut As Boolean
isPut = (UCase(opt_type) Like "PUT*")
' Roll backward
ReDim d(-M To M)
For j = N - 1 To 0 Step -1
  ' Calculate the d-vector.
  For i = 1 - M To M - 1
    d(i) = -(theta / (2 * dx)) * (rs - s2f) * f(i - 1) + (1 / dt - theta * (r + s2f / dx)) * f(i) _
         + (theta / (2 * dx)) * (rs + s2f) * f(i + 1)
    Next i
    d(-M) = ((1 / dt) - theta * r) * f(-M) + rs * S * Exp(-M * dx) * isPut
    d(M) = ((1 / dt) - theta * r) * f(M) - rs * S * Exp(M * dx) * (1 - isPut)
    ' Solve A'f = d
    Call SolveLinearSystem(a, f, d)
    ' The case with options of the american type
    If UCase(opt_type) Like "*_AMERICAN*" Then
      For i = -M To M
        f(i) = Application.Max(f(i), pay_off(S * Exp(i * dx), K, opt_type))
      Next i
    End If
  Next j
  ' Return the price.
  FD = f(0)
End Function
```

Finite difference methods can be much faster than simulation, but are numerically more delicate. Small changes can make a big difference, as it is shown in Figure 1.5. Here we smoothed the (otherwise non-differentiable) terminal condition using the Black-Scholes formula in the next to last step and then working backwards from $T - 2\Delta t$. So it is then “`For j = N-2 To 0 Step -1`” with terminal condition "`f(i) = BS(S * Exp(i * dx), sigma, r, K, dt, opt_type, "price")`" at $T - \Delta t$, and otherwise the same code. In this case the small change works to our benefit, for instance because the smooth convergence makes it possible to determine the order of convergence empirical and to extrapolate. Another advantage of finite difference methods over simulation is that the partial derivatives are directly available. These are important for hedging purposes, as the next section will show.

The results of applying the PDE solver to American and European type put options can be seen in Figure 1.6. No closed-form solution for the American put option is known, but many approximations have been suggested. The numerical PDE solution method provides a yardstick for these.

2.3 Less-than-perfect Hedging

What makes pricing by arbitrage work is the possibility of creating perfectly replicating portfolios. This requires that you know the true model and its parameters (except μ) with certainty, and in the Black-Scholes world that you can adjust your portfolio continuously. We now try to relax these assumptions, and in this way investigate the robustness of the Black-Scholes framework. A theoretical analysis of the topic is given in El Karoui, Jeanblanc-Picqué and Shreve (1998), but we look at

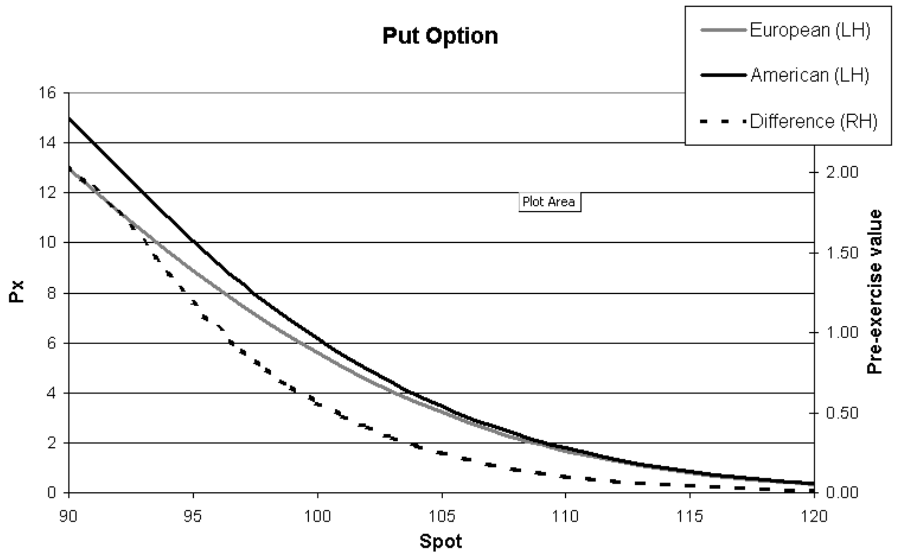


Figure 1.6. Price difference between American and European type put options in the Black-Scholes model. Except for the varying initial stock-price (on the x-axis) we use the default parameters from Table 1.1.

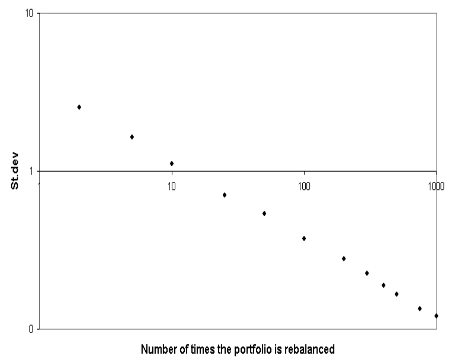
it in a simulation study. So imagine now that we have sold a call option to someone. From this we have received some funds, but taken on a (possible) future liability. We want to delta-hedge to off-set this liability, but are subject to certain constraints.

First, suppose that the Black-Scholes model is indeed the true one, but we are only able to adjust our portfolio discretely, say once every day or every week. Let's say that when we adjust, we make sure to keep the number of stocks prescribed by the theory, namely $\Delta(t_j) = \Phi(z_1(S_{t_j}, t_j))$ and use the bank account to finance or deposit the cash-flows from the stock strategy. How bad are we off then? The VBA function below can be used to answer that question. The code simulates `NumOfHedgeInLifeTime` points on the stock price path and creates a portfolio that *i*) is adjusted at time t_j in such a way that $\Phi(z_1(S_{t_j}, t_j))$ units of the stock is held (the Black-Scholes function shown earlier is used for that) and *ii*) requires no net cash in- or out-flow between time 0 and time T (money are borrowed or deposited in the bank). For each path the terminal hedge error is recorded, the experiment is repeated over many (`NumOfReplications`) paths and summary statistics of the simulated hedge errors are returned.

```
Option Explicit
Option Base 0

Function hedge_call(S As Double, sigma As Double, r As Double, k As Double, T As Double, _
                    PDrift As Double, PSigma As Double, NumOfHedgeInLifeTime As Integer, _
                    NumOfReplications As Integer) As Variant
  Dim BankAccount As Double, Stock As Double, NumOfStocks As Double, dt As Double
  Dim CallValue As Double, i As Integer, j As Integer, pfValue() As Double
  Dim Delta As Double, ChangeNumOfStocks As Double

  ' Initialize random-number generator. The seed is the Time.
  Call Randomize
  ReDim pfValue(1 To NumOfReplications)

  ' Pre-calculating
  dt = T / NumOfHedgeInLifeTime: CallValue = BS(S, sigma, r, k, T, "Call_European", "Price")

  For i = 1 To NumOfReplications
    ' Initial values.
    Stock = S: NumOfStocks = 0: BankAccount = CallValue
    For j = 1 To NumOfHedgeInLifeTime
      ' The Delta hedge
      Delta = BS(Stock, sigma, r, k, T - (j - 1) * dt, "Call_European", "Delta")
      ' Adjust the number of stocks, such the delta hedge is applied.
      ChangeNumOfStocks = Delta - NumOfStocks
      ' The remaining into the bank account
      BankAccount = BankAccount - ChangeNumOfStocks * Stock
      NumOfStocks = Delta
      ' Simulate the Stock value at time t + j*dt
      Stock = Stock * Exp((PDrift - 0.5 * PSigma ^ 2) * dt + PSigma * normal(dt))
      ' Accrue the bank account for the time period dt.
      BankAccount = BankAccount * Exp(r * dt)
    Next j

    ' The net value of the pf
    pfValue(i) = BankAccount + NumOfStocks * Stock - Application.Max(Stock - k, 0)
  Next i

  Dim res(1 To 2, 1 To 1) As Variant
  res(1, 1) = Application.WorksheetFunction.Average(pfValue)
  res(2, 1) = Application.WorksheetFunction.StDev(pfValue)
  hedge_call = res
End Function

' Generate random numbers, based on the functionality in Excel.
Function normal(variance) As Double
  normal = Application.WorksheetFunction.NormSInv(Rnd) * (variance ^ 0.5)
End Function
```

The output from this VBA-code is a range of size 2×1. To use the function in a spreadsheet you have to mark a 2×1 range (by dragging with the left mouse-button down), enter the formula as you would normally, and finishing by pressing `CTRL+SHIFT+ENTER` (not just `ENTER`). When editing the formula later, it is also important to remember to use `CTRL+SHIFT+ENTER` to finish; Excel simply will not let you continue if you do not.

This experiment shows that hedging works, and that it does not matter much what the expected rate of return on the stock, μ or `Pdrift`, is as it can be seen from the right hand graph in Figure 1.7. (If hedging takes place very infrequently,



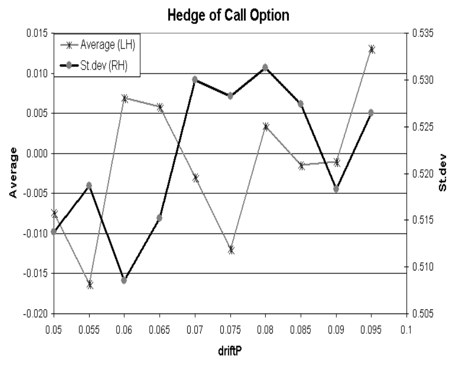


Figure 1.7. The graph on the left shows the standard deviation of the terminal hedging error for different hedging frequencies when the drift of the stock, `Psigma`, is 0.07 (and the risk-free rate is 0.05). The graph on the right shows the average and standard deviation of the hedging error with 52 rebalancings, `NoOfHedgeInLifeTime` = 52, for different drifts of the stock.

then μ may play a role in determining a "variance optimal" hedge, see Wilmott (1998)[Ch. 20].) In the left hand graph in 1.7 we have used a log/log-scale to plot the standard deviation of the hedging error (that is, the terminal value of our hedge portfolio less the pay-off of the call option) against the number of rebalancings for an expected stock return of 0.07 (when the risk-free rate is 0.05). The average hedging error is very close to 0, and does not depend very much on the number of rebalancings, so we haven't plotted it. Not only do we see that the hedge becomes more and more accurate, but the fact that the points make up a straight line with slope -1/2 indicates that the standard deviation of hedging error is proportional to $1/\sqrt(\#$ times we rebalance). Indeed this is true, but in general it depends critically on the smoothness of the pay-off function, see Gobet and Temam (2001). A variation of this experiment is to say that we adjust our hedge portfolio not every day, but only when our stock holdings are sufficiently far from the Black-Scholes delta-hedge ratio. This leads so-called bandwidth hedging, see Wilmott (1998)[Ch. 21] and the references therein.

Another interesting question is: What happens if we hedge with a wrong volatility? In Figure 1.8 we can see what happens. (For simplicity we have $\mu = r$.) For each graph, we have done the hedging 1,000 times (with a weekly hedging frequency) and for each path we plotted the realized pay-off of the call $((S(T) - K)^+)$ against the terminal value of hedge portfolio. The hedge portfolios are constructed using volatilities (called `Qsigma`) of 0.1, 0.15, and 0.2, while the true volatility is 0.15. If the hedge were perfect (i.e. if we use the true σ and adjust our portfolio continuously), then all these points should fall exactly on the pay-off function curve. They

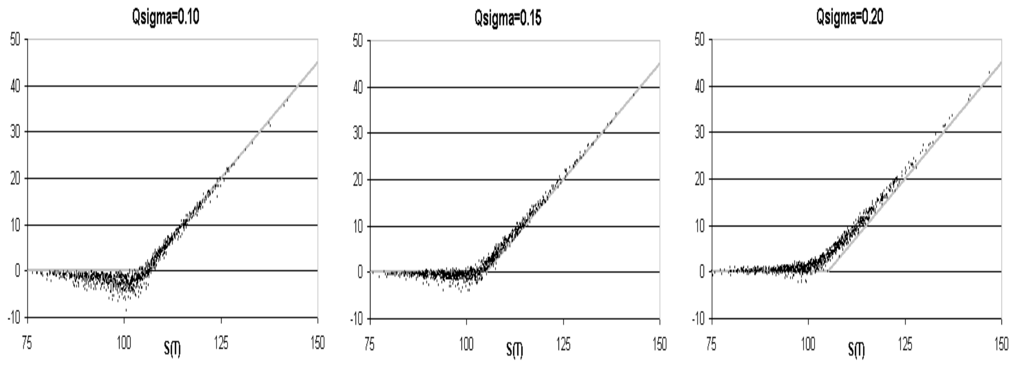


Figure 1.8. Terminal hedge portfolio values for different hedge volatilities. The true volatility is 0.15, so the middle graph corresponds to hedging with the correct volatility.

do not; we see how the points form different "clouds" around the pay-off function. When the hedging volatility is correct, the cloud most closely resembles the pay-off function, while the clouds spread out when we move away from the true volatility. There is a vertical movement that reflects different initial investments in the hedge: A high hedge volatility means that you invest more than the true Black-Scholes call price, and thus points should be above the call pay-off on average. The clouds also become more disperse, in fact if we look closely, we see that largest dispersion occurs in the vicinity of the kink in the pay-off function (i.e. at the strike price). The standard deviation of the hedge error is minimal around (although not exactly at) the true volatility. But a centered second moment probably is not the most informative measure of risk in this case. First, the distribution of the hedge error is far from normal. Second a hedger's main worry is shortfall risk. So to really analyze what the optimal hedge (volatility) is, we would have specify a utility function for the hedger, as well as his wealth and the total position he is trying to hedge, and that leads to interesting (and truly non-linear) problems.

3. Conclusion

"Sheer volume" means that Excel cannot be ignored, and in economics/finance it has applications beyond "simple accounting". We showed how to use it for option pricing calculations. There are many relevant and interesting topics that we did not touch. We could have looked further into PDEs, investigated Black-Scholes-like delta-hedging in the case where the stock price in not a Geometric Brownian motion, or discussed "implied modelling" where option prices observed in the market are used to create models with stock price dynamics that reproduce these option prices. This can all readily be done with Excel.

A further argument in favor of Excel is its strong interfacing opportunities (to e.g. the database Access or Reuter/Bloomberg financial services). And it shouldn't be ignored that a number of "Excel-Finance"-books have recently appeared, Benninga (2000) and Jackson and Staunton (2001) for instance. Of course the Devil's advocate would say that you should learn finance from a book on finance, not from a book on Excel (and vice versa).

Limitations of Spreadsheets

As high-level software, spreadsheets are generally slow for numerical computations. Even the latest Excel versions may leave something to be desired when it comes to accuracy of large-scale computations, see McCullough and Wilson (1999). The computations in this paper can all be done in "real-time" on a fairly new (early 2002) PC, but they shouldn't be much larger before we would recommend switching to something like C++ for the "number crunching", make an xll or com object and then use Excel on top as front end software, i.e. for graphical interfacing, such that you can still see what is happening without a degree in computer science.

Excel comes "in your own language". This creates peculiar compatibility problems that can be quite a nuisance. For instance matrices must be semicolon-separated on the US-version, and colon-separated in the Danish version. Further, there spelling differences for some standard functions (such as `EKSP` in Danish for the exponential function) and Danish versions do not accept US-spelling. It must be said that Microsoft have been very thorough in the translation, and sometimes that makes quite difficult to find the Danish term: Translating `RAND()` (the generation of a $U(0, 1)$-variable) to `SLUMP()` (an old Danish word used to describe "something we do not quite know where is") is very charming, but means that guessing the Danish syntax requires some imagination.

Notes

1. We have not been able to precisely locate its origin and this may be the first time that it appears in print.
2. Over the last half year we have seen several cases of computer-offers aimed at "ordinary users" where StarOffice is the default office package. So things may be changing.
3. We prefer the term "pricing by no arbitrage" to "risk-neutral pricing", since the latter could give the impression that there is an assumption of risk-neutrality among investors. There isn't.
4. First, if you are careless you get a non-recombining model, which is a computational nightmare. Second, it is cumulative gains, i.e. stock price + dividends reinvested, that should be Q-martingales when discounted to prevent arbitrage.

5. We use 1/250 because there are roughly 250 business days in a year. “Why business days?” is actually a very good question. The short answer is: That gives the best results. An indication of what is meant by this you can get from estimating the variance of Friday-to-Monday returns and comparing it to (for instance) the Monday-to-Tuesday return variance. If physical time is the real clock, the former should be about 3 times higher, but they are about the same. Intuitively, nothing is going on, when the stock markets are closed.

6. `STDEV` divides by $n-1$ rather than n (as `STDEVP` does), thus ensuring that we get an unbiased σ^2-estimate. But if n is large this is of no practical importance.

References

Benninga, S. (2000), *Financial Modeling*, 2/e, MIT Press.

Björk, T. (1998), *Arbitrage Theory in Continuous Time*, Oxford University Press.

Black, F. and Scholes, M. (1973), The Pricing of Options and Corporate Liabilities, *Journal of Political Economy*, Vol. 81, pp 637-654.

Cox, J., Ross, S. and Rubinstein, M. (1979), Option Pricing: A Simplified Approach, *Journal of Financial Economics*, Vol. 7, pp 229-263.

Duffie, D. (2001), *Dynamic Asset Pricing Theory*, 3/e, Princeton University Press.

El Karoui, N., Jeanblanc-Picqué, M. and Shreve, S. (1998), Robustness of the Black and Scholes Formula, *Mathematical Finance*, Vol. 8, pp 93-126.

Fu, M. C., Laprise, S. B., Madan, D. B., Su, Y., and Wu, R. (2001), Pricing American options: a comparison of Monte Carlo simulation approaches, *Journal of Computational Finance*, Vol. 4, pp 38-88.

Gobet, E. and Temam, E. (2001), Discrete time hedging errors for options with irregular payoffs, *Finance and Stochastics*, Vol. 5, pp 357-367.

Green, J., Bullen, S. and Martins, F. (2000), *Excel 2000 VBA Programmer's Reference*, Wrox Press.

Harrison, M. and Kreps, D. (1979), Martingales and Arbitrage in Multiperiod Securities Markets, *Journal of Economic Theory*, Vol. 20, pp 381-408.

Hull, J. (2000), *Options, Futures, and Other Derivative Securities*, 4/e, Prentice-Hall.

Jackson, M. and Staunton, M. (2001), *Advanced modelling in finance using Excel and VBA*, Wiley.

Klassen, T. R. (2001), Simple, fast and flexible pricing of Asian options, *Journal of Computational Finance*, Vol. 4, pp 89-124.

McCullough, B. D. and Wilson, B. (1999), On the Accuracy of Statistical Procedures in Microsoft Excel 97, *Computational Finance Statistics and Data Analysis*, Vol. 31, pp 27-37.

Morton, K. W. and Mayers, D. F. (1994), *Numerical Solution of Partial Differential Equations*, Cambridge University Press.

Musiela, M. and Rutkowski, M. (1997), *Martingale Methods in Financial Modelling*, Springer-Verlag.

Pliska, S. (1997), *Introduction to Mathematical Finance*, Blackwell.

Press, W., Teukolsky, S., Vetterling, W. and Flannery, B. (1993), *Numerical Recipes in C*, Cambridge University Press.

Vetzal, K. R. (1998) 'An Improved Finite Difference to Fitting the Initial Term Structure', *Journal of Fixed Income*, March, pp 62-81.

Wilmott, P. (1998), *Derivatives*, Wiley.