COMISEF
Computational Optimization Methods in Statistics, Econometrics and Finance

- Marie Curie Research and Training Network funded by the EU Commission through MRTN-CT-2006-034270 -

COMISEF WORKING PAPERS SERIES

WPS-008 15/02/2009

Implementing Binomial Trees

M. Gilli
E. Schumann

Implementing Binomial Trees ⋆

Manfred Gilli [a] and Enrico Schumann [b,*]

[a] *Department of Econometrics, University of Geneva and Swiss Finance Institute*
[b] *Department of Econometrics, University of Geneva*

Abstract

This paper details the implementation of binomial tree methods for the pricing of European and American options. Pseudocode and sample programmes for Matlab and R are given.

First version: 11 February 2009. *Last changes*: 15 February 2009.

Key words: Option pricing, Binomial trees, Numerical methods, Matlab, R

JEL codes: G13

⋆ Both authors gratefully acknowledge financial support from the EU Commission through MRTN-CT-2006-034270 COMISEF. See `http://comisef.eu`.

* Corresponding author: Department of Econometrics, University of Geneva, Bd du Pont d'Arve 40, 1211 Geneva 4, Switzerland. Tel.: +41 22 379 8218; fax: +41 22 379 8299.

Email addresses: `Manfred.Gilli@unige.ch` (Manfred Gilli), `Enrico.Schumann@unige.ch` (Enrico Schumann).

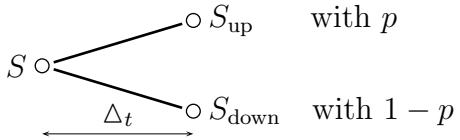
Abbreviations

C	call
P	put
S	spot
X	strike
τ	time to maturity
D	dividend amount (in currency)
q	dividend yield
r	riskfree rate
$\triangle_t$	a small time step
t	a point in 'natural' time
i, j	iterators; points on the lattice

1 Motivation

Binomial trees serve as both an intuitive approach to explain the logic behind option pricing models and as a versatile computational technique. This paper describes briefly how to implement such models. All algorithms are given in pseudocode; some sample codes are given for Matlab and R (R Development Core Team, 2008).[1] We start with the Black–Scholes–Merton option pricing model: for one, this gives a natural benchmark to judge the quality of our trees; secondly, it is easy then to demonstrate the flexibility inherent in tree-models.

The binomial model assumes that, over a short period of time $\triangle_t$, a financial asset's price S either rises by a small amount, or goes down a small amount, as pictured below. The probability of an uptick is p, while a downtick occurs with probability $1 - p$. (Probabilities here are generally risk neutral ones.)



Such a model for price movements may either be additive (ie, absolute movements) or multiplicative (ie, relative movements). 'Additive' and 'multiplicative' here refers to units of the observed price; computationally, a multiplicative model can be changed into an additive one by taking logarithms or similar transformations. There are two objections against additive models: a price could become negative, and the magnitude (in currency units) of a price change does not depend on the current price. Hence, a €1 move is as likely for a stock of €5 as for a stock of €500, which, empirically, is not the case. Still, for very short time periods an additive model may be just as good as a mul-

[1] We used Matlab R2007a and R 2.8.1.

tiplicative one. In fact, the additive model may be even better at capturing market conventions (eg, prices in certain markets may move by fixed currency amounts) or institutional settings (eg, when modelling central banks' interest rate setting). Still, we will discuss multiplicative models; the implementation of additive models requires only minor changes in the procedures described below. We will usually assume that S is a share price, even though the model may be applied to other types of securities as well.

We want to find the price, and later on the Greeks, of plain vanilla call and put options. Now is time 0, the option expires at τ. The time until expiration is divided into a number M of periods; with $M = 1$, we have $\triangle_t = \tau$, else $\triangle_t = \tau/M$. The subscript t differentiates the symbol for a short period of time from the symbol for delta, Δ, that will be used later.

Matching Moments

We start with a tree for a Black–Scholes–Merton world, thus we want to match the mean and variance of the returns of S in our tree to a given mean and variance σ^2 in a continuous time world. Let u and d be the gross returns of S in case of an uptick and downtick, respectively. With $M = 1$ and the current price S_0, we have

$$\mathsf{E}\left(\frac{S_{\triangle_t}}{S_0}\right) = pu + (1-p)d\,, \tag{1a}$$

$$\mathsf{Var}\left(\frac{S_{\triangle_t}}{S_0}\right) = \frac{1}{S_0^2}\,\mathsf{Var}(S_{\triangle_t}) = pu^2 + (1-p)d^2 - \Big(pu + (1-p)d\Big)^2\,. \tag{1b}$$

where we have used

$$\mathsf{Var}(S_{\triangle_t}) = \underbrace{S_0^2\Big(pu^2 + (1-p)d^2\Big)}_{\mathsf{E}(S_{\triangle_t}^2)} - \underbrace{S_0^2\Big(pu + (1-p)d\Big)^2}_{(\mathsf{E}\,S_{\triangle_t})^2}$$

to obtain Equation (1b). Here, E and Var are the expectations and variance operator, respectively.

In a risk neutral world with continuous time, the mean gross return is $\mathrm{e}^{r\triangle_t}$, where r is the riskfree rate. So with Equation (1a),

$$pu + (1-p)d = \mathrm{e}^{r\triangle_t}\,. \tag{2}$$

Thus we have an equation that links our first target moment, the mean return, with the tree parameters p, u and d.

In the Black–Scholes–Merton model we have lognormally distributed stock

prices with variance

$$\mathsf{Var}(S_{\triangle_t}) = S_0^2 \mathrm{e}^{2r\triangle_t}(\mathrm{e}^{\sigma^2\triangle_t} - 1) = S_0^2(\mathrm{e}^{2r\triangle_t + \sigma^2\triangle_t} - \mathrm{e}^{2r\triangle_t})\,.$$

Dividing by S_0^2 and equating to Equation (1b), we obtain

$$pu^2 + (1-p)d^2 = \mathrm{e}^{2r\triangle_t + \sigma^2\triangle_t}\,, \tag{3}$$

which links our second target moment, σ^2, with our tree parameters. We now have two equations, (2) and (3), from which we try to infer the values for p, u, and d, hence there is an infinity of possible specifications for the tree. Different authors introduce different restrictions to obtain a solution. One possible (and probably the best-known) assumption, made by Cox *et al.* (1979), is that

$$ud = 1\,.$$

We obtain

$$p = \frac{\mathrm{e}^{r\triangle_t} - d}{u - d}$$

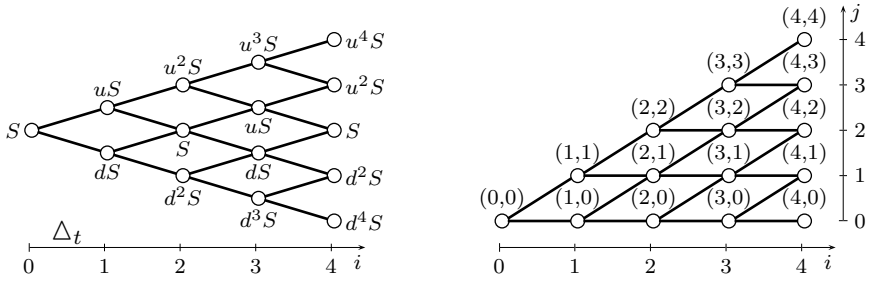
from Equation (2); for u and d, Cox *et al.* (1979) suggest the approximative solutions

$$u = \mathrm{e}^{\sigma\sqrt{\triangle_t}}\,,$$
$$d = \mathrm{e}^{-\sigma\sqrt{\triangle_t}}\,.$$

These parameter settings ensure that Equation (2) is satisfied exactly, and Equation (3) approximately (with the approximation improving for decreasing $\triangle_t$ and becoming exact in the limit). See Jabbour *et al.* (2001) for a discussion of these and other possible parameter settings.

2 Growing the tree

When we increase the number of periods, $\triangle_t = \tau/M$ becomes smaller. The following figure shows the resulting prices for $M = 4$.



The graphic on the right gives a convenient form to visualise and work with the resulting tree. The node (i, j) represents the asset price after i periods and j upticks, that is

$$S_{i,j} = S u^j d^{i-j} \,.$$

A time step on the grid is labelled i, which translates into a 'real' time point as

$$t = \tau \frac{i}{M} = i \, \triangle_t \;\; .$$

Note that there is also an equally-shaped tree for the option prices; in this option tree every node corresponds exactly to a node in the stock price tree. Furthermore, in this tree, the parameters (u, d, p) stay unchanged at every node. Thus, the volatility (ie, the volatility that is inserted in Equation (1b) to obtain the model parameters) is constant, as in the Black–Scholes–Merton model. (The volatility at a certain node is also called 'local volatility'.)

2.1 Implementing a tree

We start with computing the current price C_0 of a European call; Algorithm 1 gives the procedure for given values of the spot price S, the strike X, the volatility σ, the riskfree rate r and time to maturity τ.

Algorithm 1 European call for S, X, r, σ, τ and M time steps

1: initialise $\triangle_t = \tau/M$, $\quad S_{0,0} = S$, $\quad v = \mathrm{e}^{-r\triangle_t}$
2: compute $u = \mathrm{e}^{\sigma\sqrt{\triangle_t}}$, $\quad d = 1/u$, $\quad p = (\mathrm{e}^{r\triangle_t} - d)/(u - d)$
3: $S_{M,0} = S_{0,0} d^M$
4: **for** $j = 1 : M$ **do**
5: $\quad S_{M,j} = S_{M,j-1}\, u/d$ # initialise asset prices at maturity
6: **end for**
7: **for** $j = 0 : M$ **do**
8: $\quad C_{M,j} = \max(S_{M,j} - X, 0)$ # initialise option values at maturity
9: **end for**
10: **for** $i = M - 1 : -1 : 0$ **do**
11: $\quad$ **for** $j = 0 : i$ **do**
12: $\quad\quad C_{i,j} = v\left(p\, C_{i,j+1} + (1-p)\, C_{i,j}\right)$ # step back through the tree
13: $\quad$ **end for**
14: **end for**
15: $C_0 = C_{0,0}$

For the price of a put, we only have to replace $\max(S_{M,j} - X, 0)$ in Statement 8 by $\max(X - S_{M,j}, 0)$.

2.1.1 Numerical implementation

An implementation in Matlab may look as follows.

```
function C0 = EuropeanCall(S0,X,r,tau,sigma,M)
% version: 14 Feb 2009
% compute constants
f7 = 1;  dt = tau / M;  v = exp(-r * dt);
u = exp(sigma*sqrt(dt));   d = 1 /u;
p = (exp(r * dt) - d) / (u - d);

% initialise asset prices at maturity (period M)
S = zeros(M + 1,1);
S(f7+0) = S0 * d^M;
for j = 1:M
    S(f7+j) = S(f7+j - 1) * u / d;
end

% initialise option values at maturity (period M)
C = max(S - X, 0);

% step back through the tree
for i = M-1:-1:0
    for j = 0:i
        C(f7+j) = v * (p * C(f7+j + 1) + (1-p) * C(f7+j));
    end
end
C0 = C(f7+0);
```

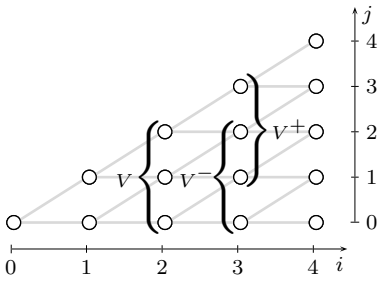
A few remarks: Several of the loops in Algorithm 1 start counting from or to 0, which is a convenient way for notation, but does not conform well with how vectors and matrices are indexed in Matlab or R. A helpful 'trick' is to use an offsetting constant (`f7`). This is added to an iterator and allows to quickly code an algorithm while reducing errors (later on one may dispose of the constant).

It is not necessary to store the complete matrices of stock and option prices, it is enough to keep one vector that is updated while stepping through the tree. Note in particular that we do not need to update the stock prices, only the option prices are computed afresh in every time step. The algorithm could be accelerated by also precomputing quantities like $(1-p)$ or, since we discount in every period by $\mathrm{e}^{-r\Delta_t}$, by leaving out the periodic discount factor `v` and instead discount `C0` by $\mathrm{e}^{-r\tau}$. Such changes would, however, only marginally improve the performance while rather obscuring the code; thus, we leave them out here.

Vectorisation We do not exploit any special structure in Matlab (ie, we do not vectorise), even though the inner loop could be avoided by realising that for every time step i, we have

$$\mathrm{e}^{r\Delta_t} \underbrace{\begin{bmatrix} C_{i,i} \\ C_{i,i-1} \\ \vdots \\ C_{i,1} \\ C_{i,0} \end{bmatrix}}_{V} = p \underbrace{\begin{bmatrix} C_{i+1,i+1} \\ C_{i+1,i} \\ \vdots \\ C_{i+1,2} \\ C_{i+1,1} \end{bmatrix}}_{V^+} + (1-p) \underbrace{\begin{bmatrix} C_{i+1,i} \\ C_{i+1,i-1} \\ \vdots \\ C_{i+1,1} \\ C_{i+1,0} \end{bmatrix}}_{V^-},$$

which is illustrated below for $i = 2$.



Some testing showed that implementing this approach does not improve performance (in version R2007a), even though in older versions of Matlab it did (see Higham (2002)). Hence, we keep the double-loop structure.

The same does not hold in R (version 2.8.1); here the nested loops are much slower than the vectorised version.

```
1  EuropeanCall <- function(S0,X,r,tau,sigma,M)
2  {
3  # version: 14 Feb 2009
4  # compute constants
5  f7 <- 1;  dt <- tau / M; v <- exp(-r * dt)
6  u  <- exp(sigma * sqrt(dt)); d <- 1 /u
7  p  <- (exp(r * dt) - d) / (u - d)
8
9  # initialise asset prices at maturity (period M)
10 S               <- numeric(M + 1)
11 S[f7+0]         <- S0 * d^M
12 for (j in 1:M){
13         S[f7+j] <- S[f7+j - 1] * u / d
14 }
15
16 # initialise option values at maturity (period M)
17 C               <- pmax(S - X, 0)
18
19 # step back through the tree
20 for (i in seq(M-1,0,by=-1)){
21       C <- v * (p * C[(1+f7):(i+1+f7)] + (1-p) * C[(0+f7):(i+f7)])
22 }
23 return(C)
24 }
```

The prices at period `M` (Statements 10–14 in the R-code) could be initialised by the more efficient vectorised command `S <- S0 * u^(0:M) * d^(M:0)`.

2.2 Binomial expansion

Algorithm 1 only uses the stock prices at τ. These prices are, in turn, computed from the current price, $S_{0,0}$. Since the number of stock paths reaching final node j is given by

$$\binom{M}{j} = \frac{M!}{(M-j)!j!}$$

we can write

$$C_{0,0} = \mathrm{e}^{-r\tau} \sum_{k=0}^{M} \binom{M}{k} p^k (1-p)^{M-k} C_{M,k} \tag{4}$$

where $C_{M,k} = \max(u^k d^{M-k} S - X, 0)$ is the payoff of the call option, evaluated at the respective final node. There is, in fact, no need to sum over all end nodes, but it suffices to select those where the option expires in the money.

Such an implementation of a 'tree' loses the possibility of including early exercise features (see below). It demonstrates the flexibility of the binomial method, though, since nothing limits us to plain vanilla payoffs $\max(S-X, 0)$ (or $\max(X-S, 0)$ for the put). An example may be a 'power option' whose payoff (for the call) is given by

$$\max(S^z - X, 0)$$

where z is a real number (usually an integer) greater than one.

2.2.1 Numerical implementation

A straightforward implementation of Equation (4) may lead to an overflow since, as the number of time steps grows, ever larger integers are required for the binomial coefficient (see Higham (2002) for a detailed discussion of ways to circumvent these problems in Matlab; see Staunton (2003) for Excel experiences).

The following Matlab-implementation is based on Higham (2002).

```
function C0 = EuropeanCallBE(S0,X,r,tau,sigma,M)
% version: 14 Feb 2009
% compute constants
dt = tau / M;
u = exp(sigma*sqrt(dt));   d = 1 /u;
p = (exp(r * dt) - d) / (u - d);

% initialise asset prices at maturity (period M)
C = max(S0*d.^((M:-1:0)').*u.^((0:M)') - X,0);

% log/cumsum version
csl= cumsum(log([1;[1:M]']));
tmp= csl(M+1) - csl - csl(M+1:-1:1) + log(p)*((0:M)') + log(1-p)*((M:-1:0)');
C0 = exp(-r*tau)*sum(exp(tmp).*C);
```

An R-adaptation of this code follows.

```
EuropeanCallBE <- function(S0,X,r,tau,sigma,M)
{
# version: 14 Feb 2009
# compute constants
dt      <- tau / M
u       <- exp(sigma*sqrt(dt))
d       <- 1 /u
p       <- (exp(r * dt) - d) / (u - d)

```

```
# initialise asset prices at maturity (period M)
C       <- pmax(S0 * d^(M:0) * u^(0:M) - X, 0)

# log/cumsum version
csl     <- cumsum(log(c(1,1:M)))
tmp     <- csl[M+1] - csl - csl[(M+1):1] + log(p)*(0:M) + log(1-p)*(M:0)
C0      <- exp(-r*tau)*sum(exp(tmp)*C)
return(C0)
}
```

3 Early excercise

Early exercise features are easily implemented in the binomial method; the original paper by Cox *et al.* (1979) advocated the method exactly because of this possibility. What is required is that when we compute the new option price at a specific node, we check whether the payoff from exercising is greater than the current value of the option. Thus we need to go through the tree, we cannot use the binomial expansion. It is also necessary to update the spot price at every node. Algorithm 2 details the necessary changes in the procedure.

Algorithm 2 Testing for early exercise: An American put

```
...
for i = M − 1 : −1 : 0 do
  for j = 0 : i do
    C_{i,j} = v (p C_{i,j+1} + (1 − p) C_{i,j})
    S_{i,j} = S_{i,j}/d
    C_{i,j} = max(C_{i,j}, X − S_{i,j})
  end for
end for
...
```

The following Matlab code can be used to price an American put.

```
function P0 = AmericanPut(S0,X,r,tau,sigma,M)
% version: 14 Feb 2009
% compute constants
f7 = 1;  dt = tau / M;  v = exp(-r * dt);
u = exp(sigma * sqrt(dt));   d = 1 / u;
p = (exp(r * dt) - d) / (u - d);

% initialise asset prices at maturity (period M)
S = zeros(M + 1,1);
S(f7+0) = S0 * d^M;
for j = 1:M
    S(f7+j) = S(f7+j - 1) * u / d;
end

% initialise option values at maturity (period M)
P = max(X - S, 0);

% step back through the tree
for i = M-1:-1:0
    for j = 0:i
        P(f7+j) = v * (p * P(f7+j + 1) + (1-p) * P(f7+j));
        S(f7+j) = S(f7+j) / d;
```

```
23          P(f7+j) = max(P(f7 + j),X - S(f7+j));
24      end
25  end
26  P0 = P(f7+0);
```

4 Dividends

4.1 Continuous dividends

If the dividend of the asset can be approximated by a continuous dividend yield q, the algorithms need only slightly be changed. In a risk neutral world, the drift of a dividend-paying asset changes from r to $r - q$; hence we replace any r by $r - q$ in the tree parameters u, d and p.

4.2 Discrete dividends

Assume the asset pays a discrete dividend, that is a fixed currency amount D, at some future time τ_D (where $0 < \tau_D < \tau$). Algorithm 3 describes how to implement the 'escrowed dividend model' for an American call option.

Algorithm 3 American call for S, X, r, σ, τ, τ_D, D and M time steps

initialise $\triangle_t = \tau/M$, $S_{0,0} = S$, $v = \mathrm{e}^{-r\triangle_t}$
compute $u = \mathrm{e}^{\sigma\sqrt{\triangle_t}}$, $d = 1/u$, $p = (\mathrm{e}^{r\triangle_t} - d)/(u - d)$
compute $S_{0,0} = S - D\mathrm{e}^{-r\tau_D}$ # adjust spot for dividend
$S_{M,0} = S_{0,0}d^M$
for $j = 1 : M$ **do**
    $S_{M,j} = S_{M,j-1}\, u/d$
end for
for $j = 0 : M$ **do**
    $C_{M,j} = \max(S_{M,j} - X, 0)$
end for
for $i = M - 1 : -1 : 0$ **do**
    for $j = 0 : i$ **do**
        $C_{i,j} = v\left(p\, C_{i,j+1} + (1 - p)\, C_{i,j}\right)$
        $S_{i,j} = S_{i,j}/d$
        compute $t = i\tau/M$ # compute current time
        if $t < \tau_D$
            $C_{i,j} = \max(C_{i,j}, S_{i,j} + D\mathrm{e}^{-r(\tau_D - t)} - X)$ # before dividend
        else
            $C_{i,j} = \max(C_{i,j}, S_{i,j} - X)$ # after dividend
        end if
    end for
end for
$C_0 = C_{0,0}$

In Matlab:

```
function C0 = AmericanCallDiv(S0,X,r,tau,sigma,D,tauD,M)
% version: 14 Feb 2009
% compute constants
f7 = 1;  dt = tau / M;  v = exp(-r * dt);
u = exp(sigma*sqrt(dt));   d = 1 /u;
p = (exp(r * dt) - d) / (u - d);

% adjust spot for dividend
S0 = S0 - D * exp(-r * tauD);

% initialise asset prices at maturity (period M)
S = zeros(M + 1,1);
S(f7+0) = S0 * d^M;
for j = 1:M
    S(f7+j) = S(f7+j - 1) * u / d;
end

% initialise option values at maturity (period M)
C = max(S - X, 0);

% step back through the tree
for i = M-1:-1:0
    for j = 0:i
        C(f7+j) = v * ( p * C(f7+j + 1) + (1-p) * C(f7+j));
        S(f7+j) = S(f7+j) / d;
        t = tau * i / M;
        if t > tauD
            C(f7+j) = max(C(f7 + j), S(f7+j) - X);
        else
            C(f7+j) = max(C(f7 + j), S(f7+j) + D*exp(-r * (tauD-t)) - X);
        end
    end
end
C0 = C(f7+0);
```

5 The Greeks

The Black–Scholes–Merton option price is a function v of the spot price S, the strike X, the (constant) volatility σ, the riskfree rate r and time to maturity τ. (If the underlyer pays dividends, these will also affect the value of v.)

A Taylor series expansion can be used to estimate the sensitivity of v to a given change in one of the parameters. The change in the option price is then a function of the (mathematical) derivatives of v, evaluated at the current values of the arguments. These derivatives are known as the 'Greeks', and for Black–Scholes–Merton the most common ones are available in closed form (see the Appendix). For the binomial model, the Greeks need in general be

approximated by finite differences, that is we need to compute

$$v_x(x,y) \simeq \frac{v(x+h,y)-v(x,y)}{h}\,, \tag{5a}$$

$$v_x(x,y) \simeq \frac{v(x,y)-v(x-h,y)}{h} \quad \text{or} \tag{5b}$$

$$v_x(x,y) \simeq \frac{v(x+h,y)-v(x-h,y)}{2h}\,. \tag{5c}$$

These equations give the forward, backward and central finite differences, respectively. The symbol v_x is the partial derivative of v with respect to x, with h a small change in x and all other arguments collected in y. One advantage of this approach over the analytical expressions is that h can be set to a 'meaningful' change: for instance, the Θ may be computed for one day hence, which may be more reasonable (and easier to communicate to a trader) than a change of infinitesimally small size.

Unfortunately, such a straightforward implementation of finite differences requires to step through the tree two or even three times. If time is of the essence, some Greeks can also be approximated directly from the original tree. (These direct approximations are again finite differences.)

5.1 Greeks from the tree

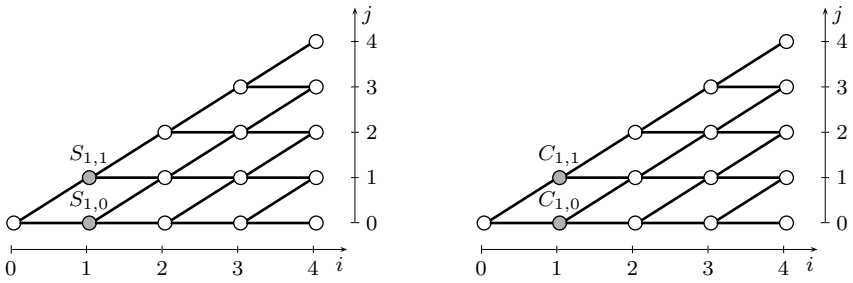
5.1.1 Delta Δ

The Δ is the change in v for a given small change in S. An estimate of Δ can be read directly from tree:

$$\Delta_{0,0} = \frac{C_{1,1} - C_{1,0}}{S_{1,1} - S_{1,0}} \tag{6}$$

where we have used a subscript to Δ to indicate the node for which it is computed (here the root node, that is now).

The following figure shows the nodes required to compute the value.



For an arbitrary node (i, j), we have

$$\Delta_{i,j} = \frac{C_{i+1,j+1} - C_{i+1,j}}{S_{i+1,j+1} - S_{i+1,j}} \,. \tag{7}$$

5.1.2 Gamma Γ

The Γ is the change in the Δ given a small change in the spot price S. Thus, the current Γ may be approximated as

$$\Gamma_{0,0} = \frac{\Delta_{1,1} - \Delta_{1,0}}{S_{1,1} - S_{1,0}} \quad \text{or} \quad \Gamma_{0,0} = \frac{\Delta_{1,1} - \Delta_{1,0}}{\frac{1}{2}(S_{2,2} - S_{2,0})} \tag{8}$$

where the second possibility uses the midpoint of the prices after two upticks and two downticks, respectively.

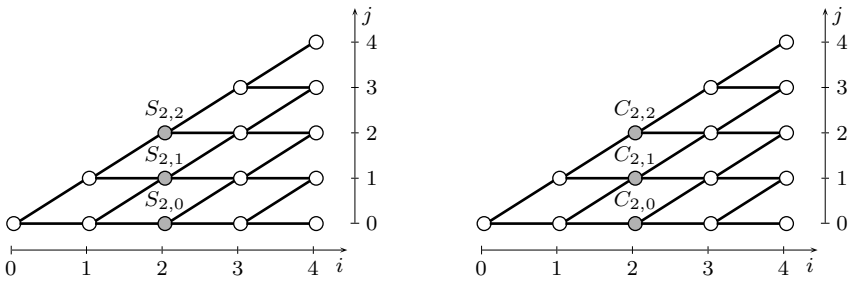
The Δ-values, following Equation (7), are

$$\Delta_{1,1} = \frac{C_{2,2} - C_{2,1}}{S_{2,2} - S_{2,1}} \quad \text{and} \quad \Delta_{1,0} = \frac{C_{2,1} - C_{2,0}}{S_{2,1} - S_{2,0}} \,.$$

Hence we obtain

$$\Gamma_{0,0} = \frac{\dfrac{C_{2,2} - C_{2,1}}{S_{2,2} - S_{2,1}} - \dfrac{C_{2,1} - C_{2,0}}{S_{2,1} - S_{2,0}}}{\frac{1}{2}\left(S_{2,2} - S_{2,0}\right)} \,, \tag{9}$$

where we have used the second approach from Equation (8) to approximate a small change in S. The figure below shows the nodes that are required to compute the Γ.



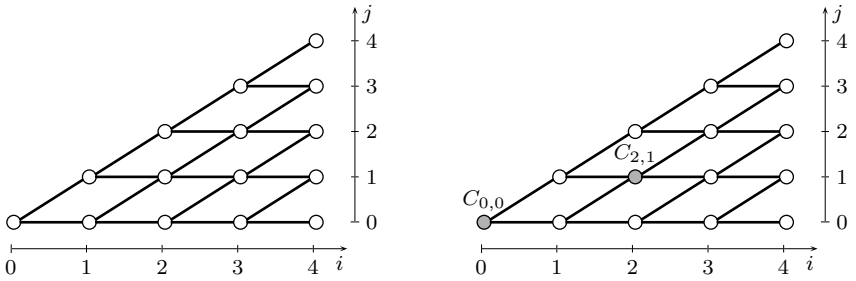
5.1.3 Theta Θ

The Θ, or 'bleed', is the change in the price of the option for a small change in τ. Since τ naturally decreases, the sign is usually switched for the analytical derivative. (So for a long position in an option, the Θ is negative.) This sign-change is not necessary for a finite difference method if we let h be a small negative quantity.

If the tree fulfils the condition $ud = 1$ (as is the case for Cox *et al.* (1979)), the spot price will arrive at its initial value at time step 2. Thus, a simple method to approximate Θ is

$$\Theta = \frac{C_{2,1} - C_{0,0}}{2\ \triangle_t}. \tag{10}$$

The following figure indicates the required nodes.



If the 'centring on the spot' condition (ie, $ud = 1$) is not fulfilled, then Θ may either be approximated by one of the finite difference schemes in Equations (5) or, as suggested by Rubinstein (1994), by

$$\Theta = rC_{0,0} - rS_{0,0}\Delta_{0,0} - \frac{1}{2}\sigma^2 S_{0,0}^2 \Gamma \tag{11}$$

which uses the Black–Scholes–Merton differential equation. The Δ and Γ can be taken from the tree as described above, hence one loop through the tree suffices to obtain Θ.

Some remarks: When computing the Greeks, the spot price must be updated in the tree. If there is a discrete dividend, it must be added back to the spot (as in the case of early exercise).

A Matlab programme (we chose a European call to make comparison with the analytical Greeks easier):

```
function [C0,deltaE,gammaE,thetaE] = EuropeanCallGreeks(S0,X,r,tau,sigma,M)
% version: 14 Feb 2009
% compute constants
f7 = 1;  dt = tau / M;  v = exp(-r * dt);
u = exp(sigma*sqrt(dt));   d = 1 /u;
p = (exp(r * dt) - d) / (u - d);

% initialise asset prices at maturity (period M)
S = zeros(M + 1,1);
S(f7+0) = S0 * d^M;
for j = 1:M
    S(f7+j) = S(f7+j - 1) * u / d;
end

% initialise option values at maturity (period M)
C = max(S - X, 0);

% step back through the tree
for i = M-1:-1:0
    for j = 0:i
```

```
21          C(f7+j) = v * ( p * C(f7+j + 1) + (1-p) * C(f7+j));
22          S(f7+j) = S(f7+j) / d;
23      end
24      if i==2
25          %gamma
26          gammaE = ((C(2+f7) - C(1+f7)) / (S(2+f7) - S(1+f7)) - ...
27              (C(1+f7) - C(0+f7)) / (S(1+f7) - S(0+f7))) / ...
28              (0.5 * (S(2+f7) - S(0+f7)));
29          %theta (aux)
30          thetaE = C(1+f7);
31      end
32      if i==1
33          %delta
34          deltaE = (C(1+f7) - C(0+f7)) / (S(1+f7) - S(0+f7));
35      end
36      if i==0
37          %theta (final)
38          thetaE = (thetaE - C(0+f7)) / (2 * dt);
39      end
40  end
41  C0 = C(f7+0);
```

6 Conclusion

In this paper we briefly described the implementation of standard binomial trees for European and American options. All Matlab and R programmes can be downloaded from `http://comisef.eu`.

A Analytical Black–Scholes–Merton

A.1 Prices

$$C = \quad S\mathrm{e}^{-q\tau}\,\Phi(\quad d_1) - X\mathrm{e}^{-r\tau}\,\Phi(\quad d_2) \tag{A.1a}$$
$$P = -S\mathrm{e}^{-q\tau}\,\Phi(-d_1) + X\mathrm{e}^{-r\tau}\,\Phi(-d_2) \tag{A.1b}$$

with

$$d_1 = \frac{1}{\sigma\sqrt{\tau}}\left(\ln\left(\frac{S}{X}\right) + \left(r - q + \frac{\sigma^2}{2}\right)\tau\right) \tag{A.2a}$$
$$d_2 = \frac{1}{\sigma\sqrt{\tau}}\left(\ln\left(\frac{S}{X}\right) + \left(r - q - \frac{\sigma^2}{2}\right)\tau\right) = d_1 - \sigma\sqrt{\tau} \tag{A.2b}$$

Here, Φ is the Gaussian distribution function, ϕ is the Gaussian density. (In the financial engineering literature, one often finds the symbols $\mathrm{N}(\cdot)$ and $\mathrm{n}(\cdot)$, respectively).

A.2 Greeks

A.2.1 Delta Δ

$$\Delta_C = \mathrm{e}^{-q\tau}\,\Phi(d_1) \tag{A.3a}$$
$$\Delta_P = \mathrm{e}^{-q\tau}\Big(\,\Phi(d_1) - 1\Big) \tag{A.3b}$$

A.2.2 Gamma Γ

$$\Gamma_{C,P} = \frac{\mathrm{e}^{-q\tau}\,\phi(d_1)}{S\sigma\sqrt{\tau}} \tag{A.4}$$

A.2.3 adjusted Gamma Γ^a

$$\Gamma^a_{C,P} = \frac{S}{100}\Gamma \tag{A.5}$$

A.2.4 Theta Θ

$$\Theta_C = -\frac{S\mathrm{e}^{-q\tau}\phi(d_1)\sigma}{2\sqrt{\tau}} + qS\mathrm{e}^{-q\tau}\Phi(\quad d_1) - rX\mathrm{e}^{-r\tau}\Phi(\quad d_2) \tag{A.6a}$$

$$\Theta_P = -\frac{S\mathrm{e}^{-q\tau}\phi(d_1)\sigma}{2\sqrt{\tau}} - qS\mathrm{e}^{-q\tau}\Phi(-d_1) + rX\mathrm{e}^{-r\tau}\Phi(-d_2) \tag{A.6b}$$

A.2.5 Vega $\mathcal{V}$

$$\mathcal{V}_{C,P} = S\mathrm{e}^{-q\tau}\,\phi(d_1)\sqrt{\tau} \tag{A.7}$$

A.2.6 Rho R

$$R_C = \quad \tau X\mathrm{e}^{-r\tau}\Phi(\quad d2) \tag{A.8a}$$

$$R_P = -\tau X\mathrm{e}^{-r\tau}\Phi(-d2) \tag{A.8b}$$

References

Cox, John C., Stephen A. Ross and Mark Rubinstein (1979). Option Pricing: A Simplified Approach. *Journal of Financial Economics* **7**(3), 229–263.

Higham, Desmond J. (2002). Nine Ways to Implement the Binomial Method for Option Valuation in MATLAB. *SIAM Review* **44**(4), 661–677.

Jabbour, George M., Marat V. Kramin and Stephen D. Young (2001). Two-State Option Pricing: Binomial Models Revisited. *The Journal of Futures Markets* **21**(11), 987–1001.

R Development Core Team (2008). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. ISBN 3-900051-07-0.

Rubinstein, Mark (1994). Implied Binomial Trees. *The Journal of Finance* **49**(3), 771–818.

Staunton, Mike (2003). From Floating Points to Binomial Trees. *Wilmott* **1**, 44–47.