Quantitative Finance

Implied Tree

Patrick Hénaff

Version: 06 Feb 2025

Contents


```
get.src.folder <- function() {
  path.expand("../GP/src")
}
```

```
source(file.path(get.src.folder(), 'OptionUtils.R'))
```

The Breeden-Litzenberger formula provides the density of the underlying asset at expiry as a function of vanilla prices. In many instances, for pricing American options for example, it is important to know the density at any time t between the current time and expiry. In this note, we construct the function $\sigma(S_t, t)$, i.e. the instantaneous volatility as a function of the underlying asset and time.

1 The Derman-Kani Implied Tree

In a standard trinomial tree, the transition probabilities are independent of the state. By contrast, the Derman-Kani implied tree involves transition probabilities that are state-dependent. With this extension, one obtain a model that matches the observed prices of vanilla options at various maturities and strikes.

1.1 Principle

We start with a trinomial tree constructed with a constant volatility. This determines the geometry of the tree, according to the formulae derived earlier. We will now modify the transition probabilities in order to match observed option prices.

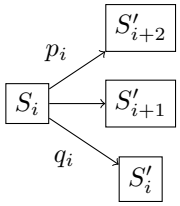

Figure 1: One step of a trinomial tree

Let's focus on one node and its three children nodes. The notation is illustrated in Figure 1.

States S_i are associated with maturity T, and states S'_i with maturity $T + \Delta t$. We will also use the following notation:

λ_i state price for node i
$C(S'_i)$ call price, strike S'_i, maturity $T + \Delta t$

The call can be priced in the trinomial tree, giving:

$$\begin{aligned} C(S'_i) = & \quad \lambda_i e^{-r\Delta t} p_i (S'_{i+2} - S'_{i+1}) \\ & + \sum_{j=i+1}^{2n} \lambda_j e^{-r\Delta t} \left[p_j (S'_{j+2} - S'_{i+1}) + (1 - p_j - q_j)(S'_{j+1} - S'_{i+1}) + q_j (S'_j - S'_{i+1}) \right] \end{aligned}$$

The price process in the tree is a martingale, and thus:

$$\begin{aligned} S'_j e^{r\Delta t} &= F_j \\ &= p_j S'_{j+2} + (1 - p_j - q_j) S'_{j+1} + q_j S'_j - S'_{i+1} \end{aligned}$$

Using this identiy, the call price becomes:

$$\begin{aligned} C(S'_i) = & \quad \lambda_i e^{-r\Delta t} p_i (S'_{i+2} - S'_{i+1}) \\ & + \sum_{j=i+1}^{2n} \lambda_j e^{-r\Delta t} \left[F_j - S'_{i+1} \right] \end{aligned}$$

In the equation above, all quantities except p_i are known, yielding:

$$p_i = \frac{e^{r\Delta t} C(S'_{i+1}) - \sum_{j=i+1}^{2n} \lambda_j e^{-r\Delta t} \left[F_j - S'_{i+1} \right]}{\lambda_i (S'_{i+2} - S'_{i+1})}$$

Using again the martingale property, one can compute the probability q_i:

$$q_i = \frac{F_i - p_i (S'_{i+2} - S'_{i+1}) - S'_{i+1}}{S'_i - S'_{i+1}}$$

The corresponding local volatility at node S'_i is finally obtained by:

$$\sigma(S'_i, T)F_i^2\Delta t = \\ p_i(S'_{i+2} - F_i)^2 + (1 - p_i - q_i)(S'_{i+1} - F_i)^2 + q_i(S'_i - F_i)^2$$

A similar calculation can be performed with put prices.

1.2 Illustration

For the sake of this example, we will use a simple expression for the Black-Scholes volatility, function of time to expiry (T) and spot price (S):

$$\sigma(S, T) = \sigma_0(1 + a(S - 100)) + bT$$

```
sigma <- .3
a <- -.2/100
b <- 0
bsvol <- function(S,T) {
  sigma*(1+a*(S-100)) + b*T
}
```

We now proceed step-by-step in the calculation process. The first step is to construct a classical trinomial tree, with constant probabilities. This determines the geometry of the tree.

In this example, we construct a 3-periods tree, anual volatility is 30%. For simplicity, interest rate and dividend yield is set to 0.

```
## all rates = 0
r <- 0
b <- 0

## time steps
n <- 3
dt <- .01

# number of rows and columns in the price grid
nr <- 2*(n+1)-1
nc <- (n+1)
T <- seq(0, nc) * dt

## Trinomial tree
u <- exp(sigma*sqrt(2*dt))
d <- 1/u
## constant probabilities
pu <- ((exp(r*dt/2)-exp(-sigma*sqrt(dt/2)))/(exp(sigma*sqrt(dt/2))-exp(-sigma*sqrt(dt/2))))^2
```

```
pd <- (1-sqrt(pu))^2
pm <- 1-pu-pd
## grid of prices at each tree node
S <- matrix(data=NA, nrow=nr, ncol=nc)
for(j in seq(1,nc)) {
  S[1:(1+2*(j-1)),j] <- 100 * u^(j-1) * d^seq(0, 2*(j-1))
}
```

The resulting tree is pictured in Figure 2. This is a standard trinomial tree. Branching probabilities are identical at each node, volatility at each node is also constant, 30% annual rate.

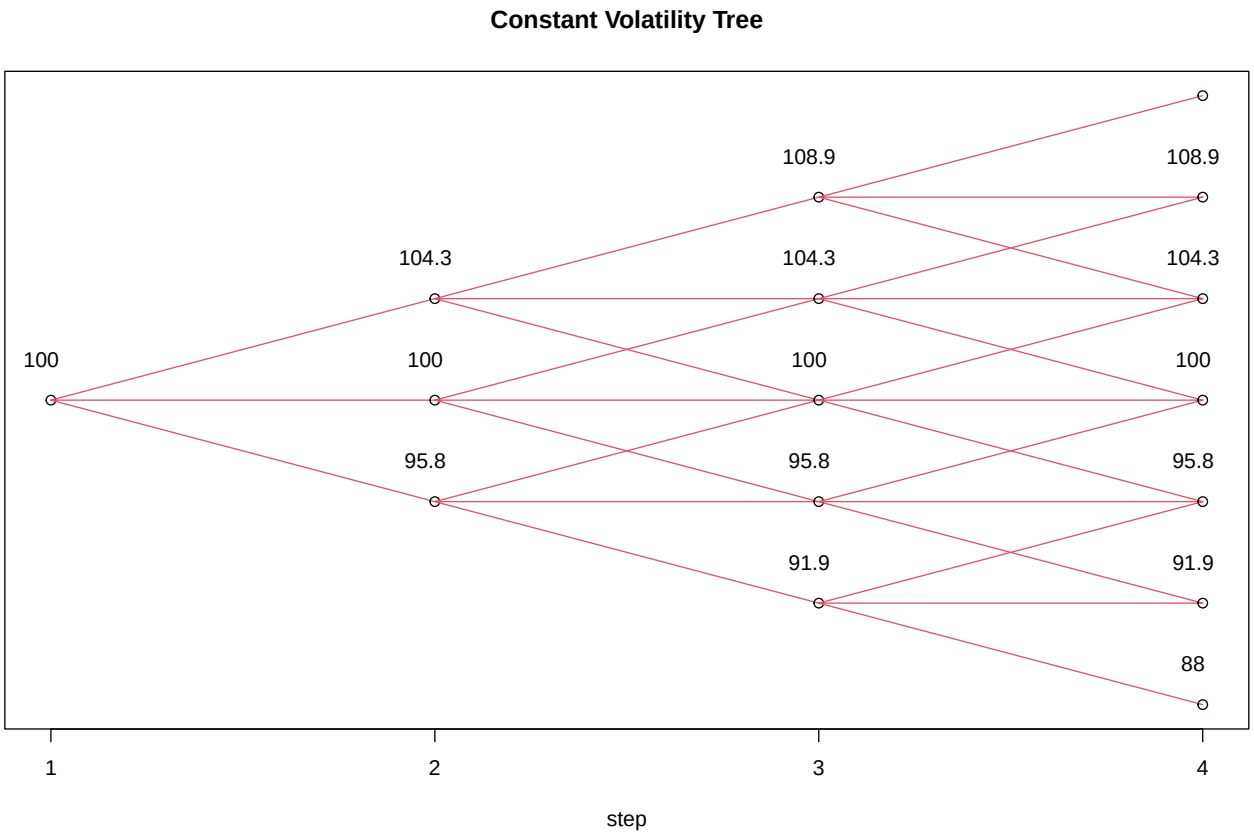

Figure 2: Constant volatility trinomial tree

The next step is to price calls and puts expiring at each time step, and struck at each mode value. The volatility used to price each option is time and strike dependent, per equation ((1.2).

The volatility grid is computed by evaluating the volatility function for each node:

```
Vol <- matrix(data=NA, nrow=nr, ncol=nc)
for(i in seq(1,nr)) {
  for(j in seq(1, nc))
    if(!is.na(S[i,j])) Vol[i,j] <- bsvol(S[i,j], T[j])
}
```

and this volatility is used to compute the call and put prices expiring at each time step and strike level:

```
## matrice des prix call/put
Call <- matrix(data=NA, nrow=nr, ncol=nc)
Put <- matrix(data=NA, nrow=nr, ncol=nc)
for(i in seq(1,nr)) {
  for(j in seq(2, nc))
    if(!is.na(S[i,j])) {
        Call[i,j] <- CRRTrinomial('ce', S[1,1], S[i,j], T[j], r, b, Vol[i,j], j-1)$price
        Put[i,j] <- CRRTrinomial('pe', S[1,1], S[i,j], T[j], r, b, Vol[i,j], j-1)$price
    }
}
```

Next, we use equations (1.1) and (1.1) to compute the transition probabilities at each node. The probability p_i is computed by:

```
p <- function(i, j) {
  # S'_{i+1} and S'_{i+2}
  SP1 <- S[i+1, j+1]
  SP2 <- S[i, j+1]
  # vector of lambdas
  tmp = 0
  if(i>1) {
    l <- Lambda[1:(i-1),j]
    F <- S[1:(i-1),j]*exp(r*dt)
    #print(paste('F: ', F, ' SP1: ', SP1, ' SP2: ', SP2))
    tmp = t(l) %*% (F-SP1)
  }
  # prob
  (exp(r*dt)*Call[i+1,j+1] - tmp)/(Lambda[i,j]*(SP2-SP1))
}
```

and the probability q_i is determined by:

```
q <- function(i, j) {
  # S'_{i+2}, S'_{i+1} and S'_{i}
  SP2 <- S[i, j+1]
  SP1 <- S[i+1, j+1]
  SP <- S[i+2, j+1]
  F <- S[i,j]*exp(r*dt)
  (F-p(i,j)*(SP2-SP1)-SP1)/(SP-SP1)
}
```

With these two functions, we can proceed recursively, computing the state prices and the transition probabilities one time step at a time. Note that functions p and q above use as input the state prices

up to time step $i-1$ in order to compute the transition probabilities at time step i. The transition probabilites for the root node are computed immediately:

```
Lambda <- matrix(data=NA, nrow=7, ncol=4)
Lambda[1,1] <- 1
Pu <- p(1,1)
Pd <- q(1,1)
Pm <- 1-Pu-Pd
```

and this provides the data for computing the state prices $\lambda_{i,2}, i = 1, \ldots, 3$ for time step Δt.

```
Lambda[1,2] <- Pu * exp(-r*dt)
Lambda[2,2] <- Pm * exp(-r*dt)
Lambda[3,2] <- Pd * exp(-r*dt)
```

The state prices for the other time steps are computed similarly.

```
Lambda[1,3] <- p(1,2)*Lambda[1,2]
Lambda[2,3] <- (1-p(1,2)-q(1,2))*Lambda[1,2] + p(2,2)*Lambda[2,2]
Lambda[3,3] <- q(1,2)*Lambda[1,2] + (1-p(2,2)-q(2,2))*Lambda[2,2] + p(3,2)*Lambda[3,2]
Lambda[4,3] <- (1-p(3,2)-q(3,2))*Lambda[3,2] + q(2,2)*Lambda[2,2]
Lambda[5,3] <- q(3,2)*Lambda[3,2]

Lambda[1,4] <- p(1,3)*Lambda[1,3]
Lambda[2,4] <- (1-p(1,3)-q(1,3))*Lambda[1,3] + p(2,3)*Lambda[2,3]
Lambda[3,4] <- q(1,3)*Lambda[1,3] + (1-p(2,3)-q(2,3))*Lambda[2,3] + p(3,3)*Lambda[3,3]
Lambda[4,4] <- q(2,3)*Lambda[2,3] + (1-p(3,3)-q(3,3))*Lambda[3,3] + p(4,3)*Lambda[4,3]
Lambda[5,4] <- q(3,3)*Lambda[3,3] + (1-p(4,3)-q(4,3))*Lambda[4,3] + p(5,3)*Lambda[5,3]
Lambda[6,4] <- (1-p(5,3)-q(5,3))*Lambda[5,3] + q(4,3)*Lambda[4,3]
Lambda[7,4] <- q(5,3)*Lambda[5,3]
```

Since interest rate is 0, the state prices at each time step should sum up to 1. This is verified by:

```
z <- apply(Lambda, 2, function(x){sum(x[!is.na(x)])})
print(z)
```

```
## [1] 1 1 1 1
```

Having determined the state prices, we record the transition probabilites in grids, in order to facilitate the display. The up and down probabilities associated with each node are displayed in Figure 3.

```
Pup <- matrix(data=NA, nrow=nr, ncol=nc)
Pdn <- matrix(data=NA, nrow=nr, ncol=nc)
Pmd <- matrix(data=NA, nrow=nr, ncol=nc)
for(i in seq(1,nr)) {
  for(j in seq(1, nc-1))
    if(!is.na(S[i,j])) {
      Pup[i,j] <- p(i,j)
      Pdn[i,j] <- q(i,j)
      Pmd[i,j] <- 1-Pup[i,j]-Pdn[i,j]
    }
}
```

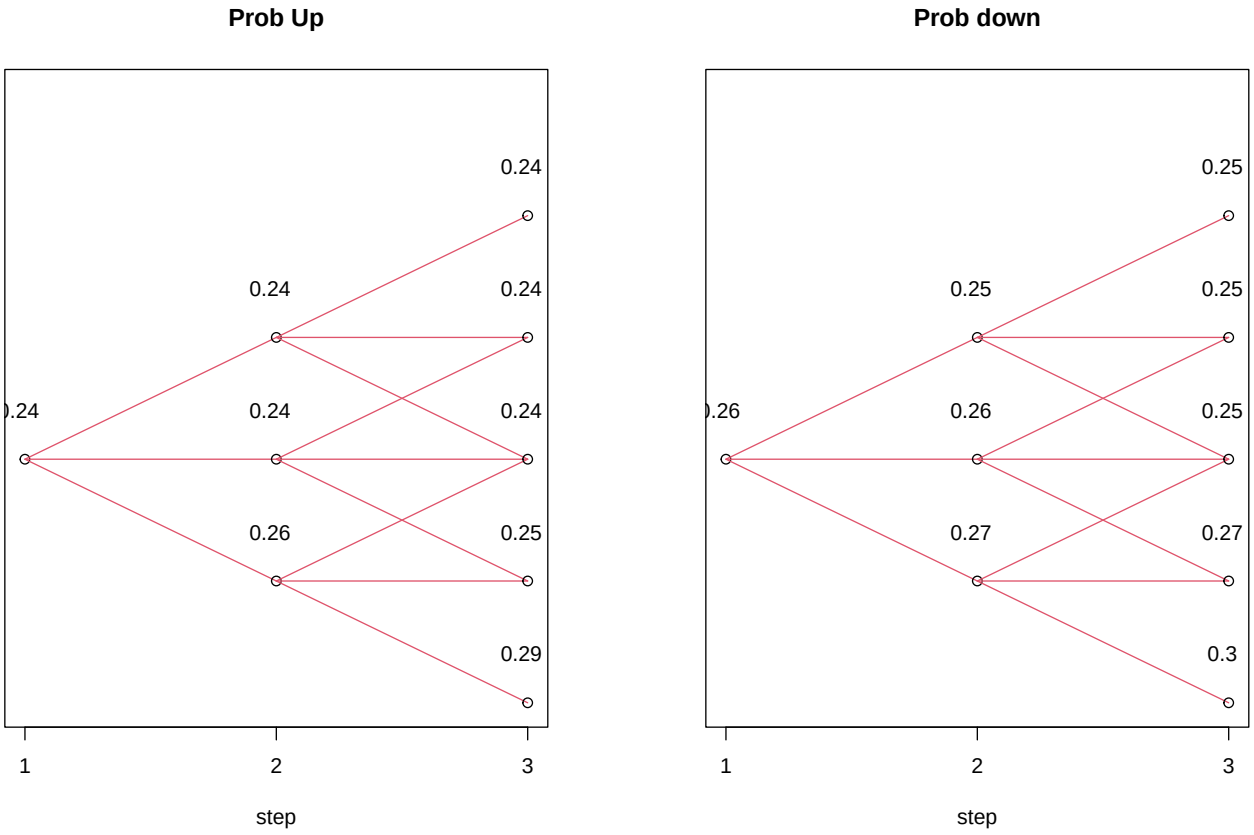

Figure 3: Up and down probabilities

Finally, the local volatility at each node can be computed from the transition probabilities:

```
lvol <- function(i,j) {
  SP2 <- S[i, j+1]
  SP1 <- S[i+1, j+1]
  SP <- S[i+2, j+1]
  F <- S[i,j]*exp(r*dt)
  sqrt((Pup[i,j]*(SP2-F)^2 + Pdn[i,j]*(SP-F)^2 + Pmd[i,j]*(SP1-F)^2)/(F^2*dt))
```

```
}

LVol <- matrix(data=NA, nrow=nr, ncol=nc)
for(i in seq(1,nr)) {
  for(j in seq(1, nc-1))
    if(!is.na(S[i,j])) {
      LVol[i,j] <- lvol(i,j)
    }
}
```

Figure 4 shows the local volatility associated with each node, and, for comparison, the Black-Scholes volatility, which is the average volatility from time 0 to expiry associated with a particular strike and expiry date.

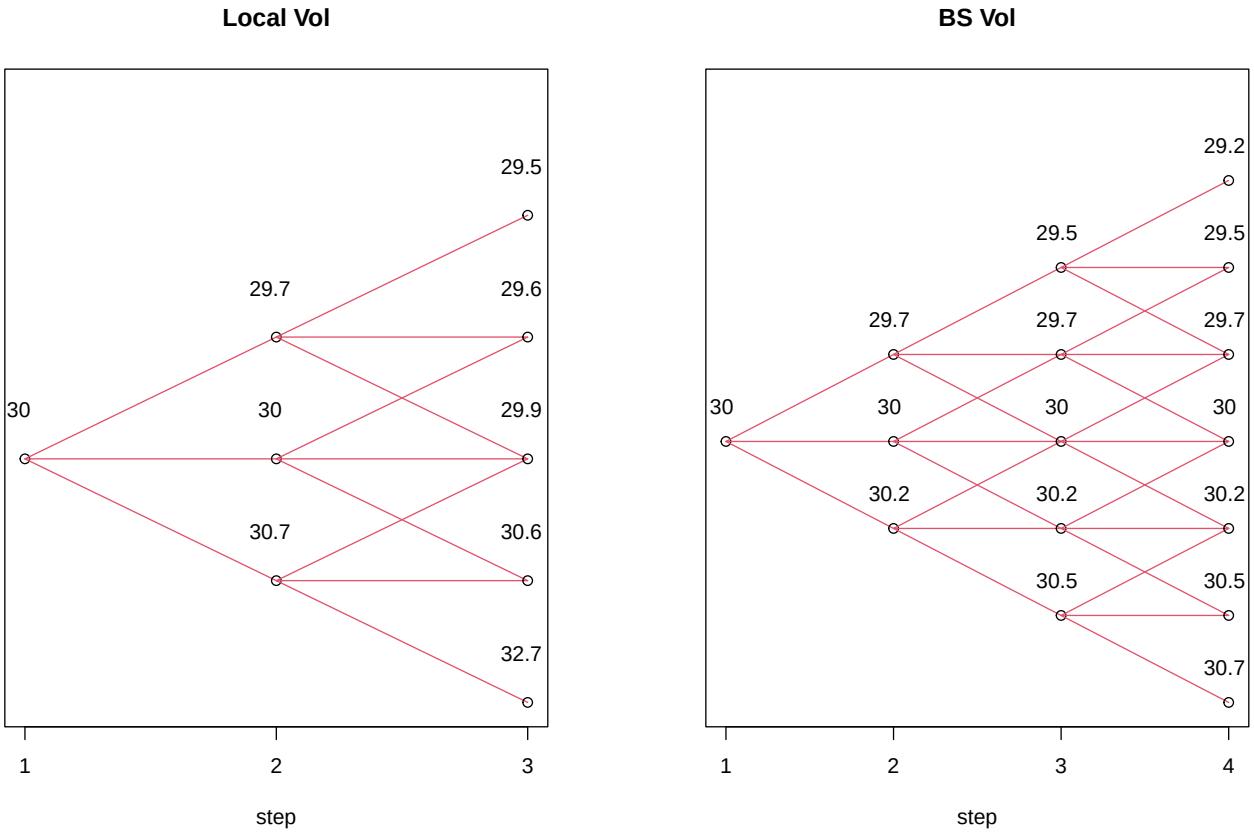

Figure 4: Local volatility and Black-Scholes average volatility at each node