Fin 514: Financial Engineering II

Lecture 13: Finite Difference Methods II

Dr. Martin Widdicks

UIUC

Spring, 2018

Outline

- We have seen the basic idea behind finite-difference methods and introduced the explicit finite difference method. Unfortunately it is an **unstable** scheme that can put limitations on the size of the S- and t-steps.
- Here we will introduce the Crank-Nicolson (CN) method which is a stable method and also has improved convergence.
- Where possible you should use the CN model due its stability and improved convergence.
- In addition we will discuss how to price American options using both of these methods as well as looking how to remove non-linearity error in a variety of cases.

Stability problem

- You have a structured product on a single stock..
- You wish to have one time step per day, so $dt = 1/365$.
- If the volatility of the underlying stock is 0.5 then:

  $$j < \sqrt{\frac{1}{1/365 \times 0.5^2}} = 38$$

  so we can only have 38 S steps, which is very small.
- For example, if $S_0 = \$50$ and $Smax = \$125$ then $dS = 125/38 = \$3.29$ which are very big time steps, especially for getting nonlinearity error down.

Quiz

What is also annoying about the stability constraint?

- Convergence is faster in S than in t
- Convergence is faster in t that in S
- We will need explicit schemes when S is large
- We will need explicit schemes when S is small

Intro to a better method

- The problems with the explicit finite difference method are two fold
    - It is unstable
    - The convergence is only linear in t.
- There is a related finite difference method that overcomes both of these problems.
- It is more complex to show that it overcomes the first but the second is easy to show.

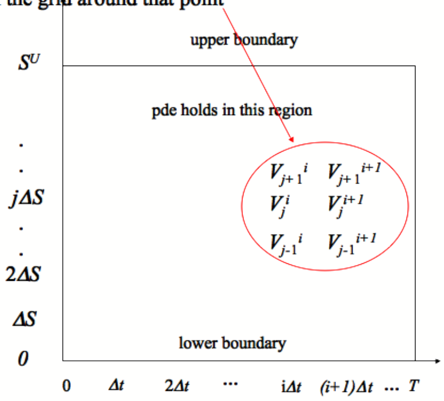
The Crank-Nicolson Method

- The Crank-Nicolson scheme works by evaluating the derivatives at $V(S, t + \Delta t/2)$.
- The first advantages of this is that the error in the time derivative is now $(\Delta t)^2$ rather than Δt.
- Most importantly there are no stability constraints, so we are free to choose any number of time and S steps.
- The only problem with using the Crank-Nicolson method rather than the explicit method is that we will need to use three option values in the future $(t + \Delta t)$ to calculate three option values not (t). This will make the backward iteration scheme harder. Generally it is worth the effort, though.
- The next diagram shows the six points of interest.

Crank-Nicolson grid

Focus attention on *i*, j-th value V_j^i, and a little

piece of the grid around that point



C-N approximations

$$\begin{aligned}
V(S,t+\Delta t) &= V(S,t+\Delta t/2) + \frac{\Delta t}{2}\frac{\partial V}{\partial t} + \frac{1}{4}(\Delta t)^2\frac{\partial^2 V}{\partial t^2} + O((\Delta t)^3) \\
V(S,t) &= V(S,t+\Delta t/2) - \frac{\Delta t}{2}\frac{\partial V}{\partial t} + \frac{1}{4}(\Delta t)^2\frac{\partial^2 V}{\partial t^2} + O((\Delta t)^3)
\end{aligned}$$

- Thus we can estimate the first derivative at $S, t+\Delta t/2$ using central differencing and so,

$$\frac{\partial V}{\partial t} = \frac{V(S,t+\Delta t) - V(S,t)}{\Delta t} + O((\Delta t)^2)$$

- Note that the error term is now $O((\Delta t)^2)$.

C-N approximations

- The S derivatives must also be evaluated at $S, t + \Delta t/2$ to give:

$$\begin{aligned}\frac{\partial V}{\partial S} &= \frac{1}{2}\Bigg(\frac{V(S+\Delta S,t) - V(S-\Delta S,t)}{2\Delta S} + \\ &\qquad \frac{V(S+\Delta S,t+\Delta t) - V(S-\Delta S,t+\Delta t)}{2\Delta S}\Bigg) + O((\Delta S)^2)\end{aligned}$$

$$\begin{aligned}\frac{\partial^2 V}{\partial S^2} &= \frac{V(S+\Delta S,t) - 2V(S,t) + V(S+\Delta S,t)}{2(\Delta S)^2} \\ &\quad + \frac{V(S+\Delta S,t+\Delta t) - 2V(S,t+\Delta t) + V(S+\Delta S,t+\Delta t)}{2(\Delta S)^2} + O((\Delta S)^2)\end{aligned}$$

Approximations of the derivatives

- From our new approximations, in terms of V_j^i we have

$$\begin{aligned}
\frac{\partial V}{\partial t} &= \frac{V_j^{i+1} - V_j^i}{\Delta t} \\
\frac{\partial V}{\partial S} &= \frac{1}{4\Delta S}(V_{j+1}^i - V_{j-1}^i + V_{j+1}^{i+1} - V_{j-1}^{i+1}) \\
\frac{\partial^2 V}{\partial S^2} &= \frac{1}{2\Delta S^2}(V_{j+1}^i - 2V_j^i + V_{j-1}^i + V_{j+1}^{i+1} - 2V_j^{i+1} + V_{j-1}^{i+1})
\end{aligned}$$

- Here the V^i values are all unknown, so we rearrange our equations to have the known values on one side and the unknown values on the other.

$$\begin{aligned}
&\frac{1}{4}(\sigma^2 j^2 - rj)V_{j-1}^i + (-\frac{\sigma^2 j^2}{2} - \frac{r}{2} - \frac{1}{\Delta t})V_j^i + \frac{1}{4}(\sigma^2 j^2 + rj)V_{j+1}^i \\
= &-\frac{1}{4}(\sigma^2 j^2 - rj)V_{j-1}^{i+1} - (-\frac{\sigma^2 j^2}{2} - \frac{r}{2} + \frac{1}{\Delta t})V_j^{i+1} - \frac{1}{4}(\sigma^2 j^2 + rj)V_{j+1}^{i+1}
\end{aligned}$$

The problem

We can rewrite the valuation problem in terms of a matrix as follows:

$$\begin{pmatrix} b_0 & c_0 & 0 & 0 & . & & . & 0 \\ a_1 & b_1 & c_1 & 0 & . & & & . \\ 0 & a_2 & b_2 & c_2 & 0 & & & . \\ . & & a_3 & b_3 & c_3 & & & . \\ . & & . & . & . & & & . \\ . & & & & a_j & b_j & c_j & & . \\ . & & & & . & . & . & & . \\ 0 & & & & . & . & 0 & a_{j\max} & b_{j\max} \end{pmatrix} \begin{pmatrix} V_0^i \\ V_1^i \\ V_2^i \\ V_3^i \\ . \\ . \\ V_{j\max\text{-}1}^i \\ V_{j\max}^i \end{pmatrix} = \begin{pmatrix} d_0^i \\ d_1^i \\ d_2^i \\ . \\ . \\ . \\ . \\ d_{j\max}^i \end{pmatrix}$$

where:

$$\begin{aligned} a_j &= \frac{1}{4}(\sigma^2 j^2 - rj) \\ b_j &= (-\frac{\sigma^2 j^2}{2} - \frac{r}{2} - \frac{1}{\Delta t}) \\ c_j &= \frac{1}{4}(\sigma^2 j^2 + rj) \\ d_j^i &= -\frac{1}{4}(\sigma^2 j^2 - rj) V_{j-1}^{i+1} - (-\frac{\sigma^2 j^2}{2} - \frac{r}{2} + \frac{1}{\Delta t}) V_j^{i+1} - \frac{1}{4}(\sigma^2 j^2 + rj) V_{j+1}^{i+1} \end{aligned}$$

Understanding the equations

- In non-matrix form we have, on the lower boundary

  $$b_0 V_0^i + c_0 V_1^i = d_0^i$$

  but usually our boundary condition is

  $$V_0^i = const.$$

  for example, with a call option, $V_0^i = 0$ and so in this case $b_0 = 1, c_0 = 1$ and $d_0^i = 0$.

- Away from the boundary we have:

  $$a_j V_{j-1}^i + b_j V_j^i + c_j V_{j+1}^i = d_j^i$$

  where we know the values of a_j, b_j, c_j, d_j^i and we need to solve to find the V_j^i values.

Quiz

Is this system of equations solvable?

- Yes
- No
- it depends

Behavior on the boundaries

- As above the boundary equations are a little different from what happens inside.
- For most PDEs we know the boundary conditions for large and small S and so it is trivial to put in the values of *a*, *b* and *c* on the boundaries.
- For the call option:

$$a_{j\max} = 0, b_{j\max} = 1, d_{j\max} = S^u e^{-\delta(T-i\Delta t)} - Ke^{-r(T-i\Delta t)}$$

$$b_0 = 1, c_0 = 0, d_0 = 0$$

- For the put option

$$b_0 = 1, c_0 = 0, d_0 = Ke^{-r(T-i\Delta t)}$$

$$a_{j\max} = 0, b_{j\max} = 1, d_{j\max} = 0$$

- In general we can always determine the values of $b_0, c_0, d_0, a_{j\max}, b_{j\max}$ and $d_{j\max}$ from our boundary conditions.

Quiz

What are the values of a, b and c at the upper and lower boundaries for:

- American put options
- Down-and-out call option
- Up-and-out put option

The Crank-Nicolson method

- At each point in time we need to solve the matrix equation in order to calculate the V_j^i values. There are two approaches to doing this, the first is to solve the matrix equation directly (technically called LU decomposition) the second is to solve the matrix equation via an iterative method (SOR).
- Typically the LU approach is the preferred approach as it gives you an exact value for V_j^i. However, as we will see it may not work particularly well for American options. The SOR (Successive Over Relaxation) approach is easy to program and can be easily adapted to value American options or more exotic options in general (this is actually PSOR - Projected Successive Over Relaxation).
- We will detail both approaches.

LU decomposition

- The matrix equation can be written as $AV = d$
- One solution would be to invert A to get $V = A^{-1}d$, but this is computationally intensive, especially as most of the entries in A are zero.
- The matrix A is a special type of matrix in that we only have non-zero entries around the diagonal. This type of matrix is called tridiagonal.
- LU decomposition allows us to solve the matrix equation without having to invert it.
- To demonstrate this I will look at the individual equations rather than the matrix so that it is easier to program.

LU decomposition

- In order to solve this the method described by G.D. Smith (1978) is used. This method gives a systematic way of solving the matrix equations. Using the notation from above assume that the following stage of eliminations has been reached:

$$\begin{aligned} \alpha_{j-1} V^i_{j-1} + c_{j-1} V^i_j &= S_{j-1} \\ a_j V^i_{j-1} + b_j V^i_j + c_j V^i_{j+1} &= d^i_j \end{aligned}$$

  where

$$\alpha_0 = b_0 \text{ and } S_0 = d^i_0.$$

- Eliminating V^i_{j-1} and rearranging gives

$$(b_j - \frac{a_j c_{j-1}}{\alpha_{j-1}}) V^i_j + c_j V^i_{j+1} = d^i_j - \frac{a_j S_{j-1}}{\alpha_{j-1}}$$

LU decomposition

- Which can be written in the form

$$\alpha_j V_j^i + c_j V_{j+1}^i = S_j \tag{1}$$

- The last set of simultaneous equations are given by

$$a_{j\max} V_{j\max-1}^i + b_{j\max} V_{j\max}^i = d_{j\max}^i$$

  and by the same process as before elimination of the $V_{j\,\max-1}^i$ term yields

$$\begin{aligned} (b_{j\max} - \frac{a_{j\max} c_{j\max-1}}{\alpha_{j\max-1}}) V_{j\max}^i &= d_{j\max}^i - \frac{a_{j\max} c_{j\max-1}}{\alpha_{j\max-1}} \quad (2) \\ \alpha_{j\max} V_{j\max}^i &= S_{j\max}. \quad (3) \end{aligned}$$

LU decomposition

- The equations are now in the form where one can start obtaining results for the unknown V values. Using equations 1 and 2, the values for V are as follows:

$$V^i_{j\max} = \frac{S_{j\max}}{\alpha_{j\max}} \tag{4}$$

$$V^i_j = \frac{1}{\alpha_j}(S_j - c_j V^i_{j+1}) \tag{5}$$

where the αs and S_js are given by

$$\alpha_0 = b_0;\ \alpha_j = b_j - \frac{a_j c_{j-1}}{\alpha_{j-1}} \tag{6}$$

$$S_0 = d^i_0;\ S_j = d^i_j - \frac{a_j}{\alpha_{j-1}} S_{j-1}, \tag{7}$$

where $j = 1, 2, 3, \ldots., j_{\max}$

Overview of approach

- **Starting at $j = 0$ calculate the α_j and S_j values as follows:**

$$\alpha_0 = b_0; \ \alpha_j = b_j - \frac{a_j c_{j-1}}{\alpha_{j-1}}$$

and

$$S_0 = d_0^i; \ S_j = d_j^i - \frac{a_j}{\alpha_{j-1}} S_{j-1},$$

- **Then continue until, $j = jmax$. Now work backward from $j = jmax$ calculating V_j^i:**

$$\begin{aligned} V_{j\max}^i &= \frac{S_{j\max}}{\alpha_{j\max}} \\ V_j^i &= \frac{1}{\alpha_j}(S_j - c_j V_{j+1}^i) \end{aligned}$$

until $j = 0$.

- **This should not be too difficult to program.**

Comments

- So this method allows us to move from a known value of $V_{j\max}^i$ up to V_0^i using the formulas here. Thus we have an explicit algorithm for calculating V_j^i for all values of i and j that does not involve the inversion of a large, sparse matrix.
- The method will be to start from expiry, then move back a time period, calculate the values of a_j, b_j, c_j and d_j^i (note that d_j^i uses $V_j^{i\max}$ values) and use these together with the equations on the previous page to calculate all of the values of $V_j^{i\max-1}$ and then step back one more time step and repeat the process until we reach $i = 0$ when we have the current option value.
- To calculate the V_j^i values, work up from $j = 0$ to obtain $\alpha_j$and S_j using equations **??** and **??** and then work down from jmax to determine V_j^i using equation **??**.

Quiz

When calculating the values V_j^i using LU decomposition is the value of V_j^i independent of V_{j+1}^i and V_{j-1}^i?

- Yes
- No
- Sometimes

SOR method

- The SOR method is a simpler approach but can be slightly less accurate and takes longer as it relies upon iteration.
- If we consider each of the individual equations from Av = d we have that

$$\begin{array}{rcl} a_1 V_0^i + b_1 V_1^i + c_1 V_2^i & = & d_1^i \\ a_2 V_1^i + b_2 V_2^i + c_2 V_3^i & = & d_2^i \\ & \vdots & \\ a_j V_{j-1}^i + b_j V_j^i + c_j V_{j+1}^i & = & d_j^i \\ & \vdots & \\ a_j V_{j\max-2}^i + b_j V_{j\max-1}^i + c_j V_{j\max-2}^i & = & d_{j\max-1}^i \end{array}$$

- The Jacobi method says that we can trivially rearrange these equations to get:

$$V_j^i = \frac{1}{b_j}\left(d_j^i - a_j V_{j-1}^i - c_j V_{j+1}^i\right)$$

SOR method

- The Jacobi method is an iterative one that relies upon the previous equation. Taking an initial guess for V_j^i, $V_j^{i,0}$ (typically V_j^{i+1}) then we iterate using the formula below for the $(k+1)$th iteration:

$$V_j^{i,k+1} = \frac{1}{b_j}\left(d_j^i - a_j V_{j-1}^{i,k} - c_j V_{j+1}^{i,k}\right)$$

  this is performed until the difference between $V_j^{i,k}$ and $V_j^{i,k+1}$ is sufficiently small for all j.

- A more effective process is the Gauss-Seidel method that uses the fact that when we calculate $V_j^{i,k+1}$ we already know $V_{j-1}^{i,k+1}$ and so we use this information to write a new iterative formula

$$V_j^{i,k+1} = \frac{1}{b_j}\left(d_j^i - a_j V_{j-1}^{i,k+1} - c_j V_{j+1}^{i,k}\right)$$

SOR method

- The SOR method is another slight adjustment. It starts from the trivial observation that

$$V_j^{i,k+1} = V_j^{i,k} + (V_j^{i,k+1} - V_j^{i,k})$$

  and so $(V_j^{i,k+1} - V_j^{i,k})$ is a correction term. It may well be the case that the iterations converge faster to the correct value if we over correct. This is true if $V_j^{i,k} \to V_j^i$ monotonically in k. So the SOR algorithm says that

$$\begin{aligned} y_j^{i,k+1} &= \frac{1}{b_j}\left(d_j^i - a_j V_{j-1}^{i,k+1} - c_j V_{j+1}^{i,k}\right) \\ V_j^{i,k+1} &= V_j^{i,k} + \omega(y_j^{i,k+1} - V_j^{i,k}) \end{aligned}$$

  where $1 < \omega < 2$ is called the over-relaxation parameter.

- This can be shown to converge for our values of a and c.

Overview of approach

- **Start with a guess for the V_j^i values, probably the ones calculated at $i+1$, so $V_j^{i,0} = V_j^{i+1}$**
- **Now choose an overrelaxation parameter, $1 < \omega < 2$.**
- **Then iterate by increasing k starting at $k=1$, and for each k go from $j=0$ up to $j = jmax$ using**

$$\begin{aligned} y_j^{i,k} &= \frac{1}{b_j}\left(d_j^i - a_j V_{j-1}^{i,k} - c_j V_{j+1}^{i,k-1}\right) \\ V_j^{i,k} &= V_j^{i,k-1} + \omega(y_j^{i,k} - V_j^{i,k-1}) \end{aligned}$$

- **For each k check the largest difference between $V_j^{i,k}$ and $V_j^{i,k-1}$ for the different j values. Once this largest difference is below an acceptable level, 10^-7 say, then move back to the next i.**

Quiz

When calculating the values V_j^i using SOR is the value of V_j^i independent of V_{j+1}^i and V_{j-1}^i?

- Yes
- No
- Sometimes

Valuation thoughts

- We now have to think about using out Finite Difference methods to value different types of products.
- We will have the following features to think about:
    - Dividends
    - Early exercise
    - Nonlinearity error from discontinuous payoff
    - Nonlinearity errors from barriers, either discrete or continuous
- In general, where possible we should use the CN method as it
    - Allows us to construct any desired grid - to avoid nonlinearity error.
    - Is completely stable, and so we can choose any number of S or t steps
    - Has the fastest convergence, once we have dealt with nonlinearity error

Continuous dividends

- Here the dividend affects the risk-neutral expected return of the stock, in that now the PDE is

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + (r-\delta)S\frac{\partial V}{\partial S} - rV = 0$$

  which we have used above in the explicit case.

- The adaptation to C-N is also straightforward as now:

$$\begin{aligned}
a_j &= \frac{1}{4}(\sigma^2 j^2 - (r-\delta)j) \\
b_j &= (-\frac{\sigma^2 j^2}{2} - \frac{r}{2} - \frac{1}{\Delta t}) \\
c_j &= \frac{1}{4}(\sigma^2 j^2 + (r-\delta)j) \\
d_j^i &= -\frac{1}{4}(\sigma^2 j^2 - (r-\delta)j)V_{j-1}^{i+1} - (-\frac{\sigma^2 j^2}{2} - \frac{r}{2} + \frac{1}{\Delta t})V_j^{i+1} - \frac{1}{4}(\sigma^2 j^2 + (r-\delta)j)V_{j+1}^{i+1}
\end{aligned}$$

Proportional Dividends

- These dividends can be modeled in the same way as with the binomial tree. When constructing the grid on the ex-dividend date, the stock price becomes $j\Delta S - Dj\Delta S = j\Delta S(1 - D)$ and so these become the new stock prices in the grid on the ex-dividend date.
- As we can think of finite difference methods as analogous to binomial trees then the A, B, C terms do not change and we can proceed by using the same finite difference equations just with updated stock prices.

Fixed dividends

- These are the most problematic but they are, however, the most realistic when the underlying is a single stock.
- There are many solutions, we will use the one we used for the binomial model, formally described in Vellekoop and Nieuwenhuis (2006).
- Construct the stock grid as usual using the standard finite difference formulas. Value the option backwards in time until we hit an ex-dividend date, t_d.
- We will assume that we have a time step, $i = k$ say, that coincides with this ex-dividend date, stock prices here are given by $j\Delta S$. At this point the stock price should fall from $j\Delta S$ to $j\Delta S - D$. So, here we need to replace the option values $V_j^k = V(j\Delta S, k\Delta t)$ with $V(j\Delta S - D, k\Delta t)$.

Fixed dividends: algorithm

1. Value the option using the standard finite difference formula at $i = k$ to obtain $V(j\Delta S, k\Delta t))$, then we want to replace the $V(j\Delta S, k\Delta t)$ values with $V(j\Delta S - D, k\Delta t)$.
2. Interpolate across the known values from the grid. For example if S_d is the largest value in the tree where $S < j\Delta S - D$ and S_u is the smallest value in the tree where $S > j\Delta S - D$ then we calculate a new $V(j\Delta S - D, k\Delta t)$, call it $V^*(j\Delta S, k\Delta t)$ or V_j^* as

$$V^*(j\Delta S, k\Delta t) = V(S_d, k\Delta t) + \frac{j\Delta S - D - S_d}{S_u - S_d} \times (V(S_u, k\Delta t) - V(S_d, k\Delta t))$$

3. Replace all of the $V(j\Delta S, k\Delta t)$ values with $V^*(j\Delta S, k\Delta t)$ or, in the simpler notation $V_j^k = V_j^*$.
4. You need to be careful at the bottom of the grid. If $j\Delta S - D < S_L$ (below the bottom of the grid) then the interpolation will not work. If $S_L = 0$ then this is unrealistic anyway as a dividend will not be paid that is larger than the value of the firm. If $S_L = 0$, set the dividend to be ΔS when $j = 1$.

Fixed dividends

5. If $S_L \neq 0$ there is a boundary condition at $S = S_L$ which also applies below S_L. For example, if S_L is a continuous barrier then $V = 0$ below the barrier also and so, you can interpolate between V_j^i and the known option value $V(j\Delta S - D, k\Delta t)$
6. Calculate back through the grid applying this rule whenever there are dividend payments.
7. If the option is American then if it is a call you will wish to exercise before the dividend payment and so $V_j^k = \max(j\Delta S - K, V_j^*)$ whereas for a put it will be after and so $V_j^k = \max(K - j\Delta S - D, V_j^*)$.

American options: Explicit

- As we well know the American option pricing problem requires us to calculate the optimal early exercise strategy. To do this you typically compare the continuation value with the early exercise value, if the latter is larger then you exercise.
- To value an American option using the explicit finite difference method is pretty straightforward, you calculate the continuation value CoV_j^i by the valuation formula below

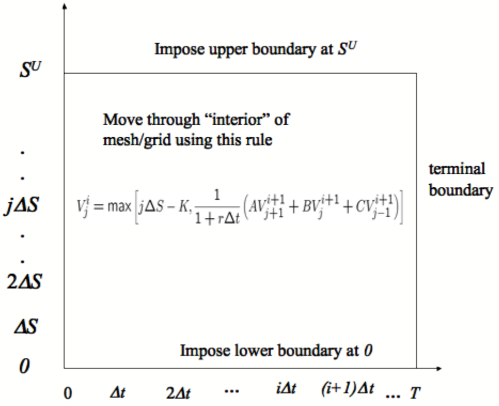
$$CoV_j^i = \frac{1}{1 + r\Delta t}\left(AV_{j+1}^{i+1} + BV_j^{i+1} + CV_{j-1}^{i+1}\right)$$

  and then compare this to the early exercise payoff. Thus for a put:

$$V_j^i = \max\left[j\Delta S - K, \frac{1}{1 + r\Delta t}\left(AV_{j+1}^{i+1} + BV_j^{i+1} + CV_{j-1}^{i+1}\right)\right]$$

- This is similar to how we value American options using the binomial tree

American put option: Explicit



Impose upper boundary at S^U
S^U
Move through "interior" of mesh/grid using this rule
$V_j^i = \max\left[j\Delta S - K, \frac{1}{1+r\Delta t}\left(AV_{j+1}^{i+1} + BV_j^{i+1} + CV_{j-1}^{i+1}\right)\right]$
terminal boundary
$j\Delta S$
$2\Delta S$
ΔS
0
Impose lower boundary at 0
0
Δt
$2\Delta t$
...
$i\Delta t$
$(i+1)\Delta t$
...
T

Quiz

Will we be able to apply the same method for the Crank-Nicolson finite difference scheme?

- Yes
- No
- Sometimes

Explain.

American put option: C-N

- The American option pricing problem is slightly more complex for the Crank-Nicolson method. To see this consider the process of calculating V_j^i.
- The value of the option V_j^i, for all values of j, depends also upon the value of V_{j-1}^i and V_{j+1}^i. Thus, when we are determining where to early exercise we also need to simultaneously know these values. However as we change one of these values then the other values will also change.
- This means that it is not possible to simply compare the early exercise value to the continuation value as, if you decide to early exercise this will change all of the values of V_j^i and then this decision may well have been suboptimal.

PSOR

- One, simple solution to this problem is to adapt our SOR method to PSOR (Projected SOR) which enables us to deal with early exercise constraints.
- The only difference between SOR and PSOR is that for each calculation in the iteration you also check whether or not it would be optimal to exercise. Thus our previous iteration technique

$$
\begin{aligned}
y_j^{i,k+1} &= \frac{1}{b_j}\left(d_j^i - a_j V_{j-1}^{i,k+1} - c_j V_{j+1}^{i,k}\right) \\
V_j^{i,k+1} &= V_j^{i,k} + \omega(y_j^{i,k+1} - V_j^{i,k})
\end{aligned}
$$

to (in the case of the American put option)

$$
\begin{aligned}
y_j^{i,k+1} &= \frac{1}{b_j}\left(d_j^i - a_j V_{j-1}^{i,k+1} - c_j V_{j+1}^{i,k}\right) \\
V_j^{i,k+1} &= \max\left(V_j^{i,k} + \omega(y_j^{i,k+1} - V_j^{i,k}), j\Delta S - K\right)
\end{aligned}
$$

LU decomposition

- With some work, it is also possible to adapt the LU decomposition approach to price American options.
- Let's consider a put option. Start by making a guess for the first stock price grid point above the early exercise boundary j_b^i. The first guess should be $j_b^{i+1} - 1$ as the boundary moves down as we move backward in time.
- For this choice, set all option values for $j < j_b^i$ to be $K - S_j^i$ and all values above to be solved via the standard C-N method. Then check the option value at $j = j_b^i$ if it is less than $K - S$ then the correct boundary is at $j_b^i + 1$ and you can perform standard calculation, otherwise reduce the value of j_b^i by one until the condition is satisfied.
- This is slightly time consuming but will give you accurate estimates for the American put option price without the often slow iteration of a PSOR scheme.

Nonlinearity error

- We know that even for vanilla European options, discontinuities in the slope or value at maturity can create nonlinearity errors. Fortunately, as we can choose any grid in S that we want we can always ensure that the strike price, or up to three features are always in the same place relative to the grid. For example, at maturity, we can choose dS such that the strike price is alwayson a grid point.
- Similarly, we can do the same with a continuous barrier - always on a grid point, or a discrete barrier - always halfway between two grid points by choosing out three S grid parameters: S_L, S_U and $jmax$ such that the grid is aligned with up to three sources of error.

Bottom of the grid not at $S = 0$

- One implication of this is that we may have the lower value of S not at $S = 0$ but at $S = S_L > 0$. This is useful when there is a continuous barrier and you would like to align the grid with the barrier.
- However, in this case, the finite difference equations do not simplify so easily, as now $S = S_L + j\Delta S$ and so, in the explicit method we have:

$$\begin{aligned} A &= \left(\frac{1}{2}\sigma^2(S_L/\Delta S + j)^2 + \frac{(r-\delta)(S_L/\Delta S + j)}{2}\right)\Delta t \\ B &= 1 - \sigma^2(S_L/\Delta S + j)^2\Delta t \\ C &= \left(\frac{1}{2}\sigma^2(S_L/\Delta S + j)^2 - \frac{(r-\delta)(S_L/\Delta S + j)}{2}\right)\Delta t \end{aligned}$$

- Note that this also changes the stability constraint as now $\sigma^2(S_L/\Delta S + j)^2\Delta t < 1$.

Bottom of the grid not at $S = 0$

- And in the C-N case we have:

$$
\begin{aligned}
a_j &= \frac{1}{4}\left(\sigma^2(S_L/\Delta S + j)^2 - (r-\delta)(S_L/\Delta S + j)\right) \\
b_j &= \left(-\frac{\sigma^2(S_L/\Delta S + j)^2}{2} - \frac{r}{2} - \frac{1}{\Delta t}\right) \\
c_j &= \frac{1}{4}\left(\sigma^2(S_L/\Delta S + j)^2 + (r-\delta)(S_L/\Delta S + j)\right) \\
d_j^i &= -\frac{1}{4}\left(\sigma^2(S_L/\Delta S + j)^2 - r(S_L/\Delta S + j)\right) V_{j-1}^{i+1} \\
&\quad - \left(-\frac{\sigma^2(S_L/\Delta S + j)^2}{2} - \frac{r}{2} + \frac{1}{\Delta t}\right) V_j^{i+1} \\
&\quad - \frac{1}{4}\left(\sigma^2(S_L/\Delta S + j)^2 + r(S_L/\Delta S + j)\right) V_{j+1}^{i+1}
\end{aligned}
$$

Convergence and accuracy

- If the option price and the derivatives are well behaved then we know that the error of the Explicit method should be $O(\Delta t, (\Delta S)^2)$ and the Crank-Nicolson method should be $O((\Delta t)^2, (\Delta S)^2)$. These can be considered similar to the distribution error for the binomial tree as this will be the general convergence to the correct option value.
- If convergence is smooth like this then the results are also amenable to **extrapolation**.
- Unfortunately just as for the explicit finite difference method, Crank-Nicolson methods will also suffer from non-linearity error if the grid is not correctly aligned with respect to any discontinuities in the option value, or the derivatives of the option value.

Conclusion

- We have introduced the Crank-Nicolson finite difference method. This is slightly harder to program than the explicit method in that you have to solve a matrix equation at each time point but it has faster convergence and is stable.
- Applying the method to American options requires the use of PSOR which again is more complex than the method for valuing American options using the explicit method.