Lecture 7

Extensions to finite-difference methods

We have seen the basic idea behind finite-difference methods and introduced the explicit finite difference method. Here we will introduce the Crank-Nicolson method which is a stable method and also has improved convergence. In addition we will discuss how to price American options using both of these methods as well as looking how to remove nonlinearity error in a variety of cases.

Example 7.1 (Reminder...)**.** What is the problem with the Explicit Method anyway?

Solution 7.1.

7.1 The Crank-Nicolson Method

As introduced at the end of the last lecture the Crank-Nicolson scheme works by evaluating the derivatives at $V(S, t+\Delta t/2)$. The main advantages of this is that the error in the time derivative is now $(\Delta t)^2$ rather than Δt and that there are no stability constraints. 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 scheme slightly harder.

Example 7.2 (Central Differencing)**.** Use finite difference to generate $O((\Delta S)^2, (\Delta t)^2)$ approximations to the derivatives in the BS PDE.

Solution 7.2.

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

$$\frac{\partial V}{\partial t} \approx \frac{V_j^{i+1} - V_j^i}{\Delta t}$$

$$\frac{\partial V}{\partial S} \approx \frac{1}{4\Delta S}(V^i_{j+1} - V^i_{j-1} + V^{i+1}_{j+1} - V^{i+1}_{j-1})$$
$$\frac{\partial^2 V}{\partial S^2} \approx \frac{1}{2\Delta S^2}(V^i_{j+1} - 2V^i_j + V^i_{j-1} + V^{i+1}_{j+1} - 2V^{i+1}_j + V^{i+1}_{j-1})$$
$$V \approx \frac{1}{2}(V^i_j + V^{i+1}_j)$$

Substituting these approximations into the BS PDE, and given that here the V^i values are all unknown, we rearrange our equations to have the known values on one side and the unknown values on the other.

$$\tfrac{1}{4}(\sigma^2 j^2 - rj)V^i_{j-1} + (-\frac{\sigma^2 j^2}{2} - \frac{r}{2} - \frac{1}{\Delta t})V^i_j + \tfrac{1}{4}(\sigma^2 j^2 + rj)V^i_{j+1} =$$
$$-\tfrac{1}{4}(\sigma^2 j^2 - rj)V^{i+1}_{j-1} - (-\frac{\sigma^2 j^2}{2} - \frac{r}{2} + \frac{1}{\Delta t})V^{i+1}_j - \frac{1}{4}(\sigma^2 j^2 + rj)V^{i+1}_{j+1}$$

The Linear Algebra Problem

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

$$\begin{pmatrix} b_0 & c_0 & 0 & 0 & . & . & . & . & 0 \\ a_1 & b_1 & c_1 & 0 & . & . & . & . & . \\ 0 & a_2 & b_2 & c_2 & 0 & . & . & . & . \\ . & . & a_3 & b_3 & c_3 & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & a_j & b_j & c_j & . & . \\ . & . & . & . & . & . & . & . & . \\ 0 & . & . & . & . & . & . & a_{jmax} & b_{jmax} \end{pmatrix} \begin{pmatrix} V^i_0 \\ V^i_1 \\ V^i_2 \\ V^i_3 \\ . \\ . \\ V^i_{jmax-1} \\ V^i_{jmax} \end{pmatrix} = \begin{pmatrix} d^i_0 \\ d^i_1 \\ d^i_2 \\ d^i_3 \\ . \\ . \\ d^i_{jmax-1} \\ d^i_{jmax} \end{pmatrix}$$

where for $1 \le j < jMax$ we have:

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

For $j = 0$ and $j = jMax$, we will need to use the boundary conditions of the problem.

Behaviour on the boundaries

Again you will note that what happens to the matrix on the boundary is 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.

Example 7.3. Derive the linear algebra equations at the boundary for both a European call and put option.

Solution 7.3.

7.2 The Crank-Nicolson Algorithm

Example 7.4 (Psuedo Code)**.** Outline the general approach to solving a European option problem with the Crank-Nicolson Method.

Solution 7.4.

So 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 (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 does not work particularly well for American options. The SOR (Successive Over Relaxation) approach is easier 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.

7.3 LU decomposition

The matrix equation can be written as $A\mathbf{V} = \mathbf{d}$. One method would be to invert A to get $\mathbf{V} = A^{-1}\mathbf{d}$ 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 decompose a tridiagonal matrix (in fact it works for more than this) into a lower triangular matrix and an upper triangular matrix. This makes solving the equation far simpler. To demonstrate this we will look at the individual equations rather than the matrix so that it is easier to program.

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

$$\beta_{j-1}V_{j-1}^j + c_{j-1}V_j^i = D_{j-1}$$

$$a_jV_{j-1}^j + b_jV_j^i + c_jV_{j+1}^i = d_j^i$$

where $\beta_0 = b_0$ and $D_0 = d_0^i$.

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

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

which can be written in the form

$$\beta_j V^i_j + c_j V^i_{j+1} = D_j$$

where $\beta_j = b_j - \frac{a_j c_{j-1}}{\beta_{j-1}}$ and $D_j = d^i_j - \frac{a_j D_{j-1}}{\beta_{j-1}}$ (1)

The last set of simultaneous equations are given by

$$\beta_{jmax-1} V^i_{jmax-1} + c_{jmax-1} V^i_{jmax} = D_{jmax-1}$$

$$a_{jmax} V^i_{jmax-1} + b_{jmax} V^i_{jmax} = d^i_{jmax}$$

and by the same process as before elimination of the V^i_{jmax-1} term yields

$$(b_{jmax} - \frac{a_{jmax} c_{jmax-1}}{\beta_{jmax-1}})V^i_{jmax} = d^i_{jmax} - \frac{a_{jmax} c_{jmax-1}}{\beta_{jmax-1}}$$

i.e.

$$\alpha_{jmax} V^i_{jmax} = D_{jmax} \quad (2)$$

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:

$$V^i_{jmax} = \frac{D_{jmax}}{\beta_{jmax}}$$

$$V^i_j = \frac{1}{\alpha_j}(D_j - c_j V^i_{j+1}) \quad (3)$$

where the βs and D_js are given by

$$\alpha_0 = b_0; \quad \beta_j = b_j - \frac{a_j c_{j-1}}{\beta_{j-1}} \quad (4)$$

$$D_0 = d^i_0; \quad D_j = d^i_j - \frac{a_j}{\beta_{j-1}} D_{j-1}, \quad (5)$$

where $j = 1, 2, 3, ..., jmax$.

Discussion

So this method allows us to move from a known value of V^i_{jmax} up to V^i_0 using the formulae here. Thus we have an explicit algorithm for calculating V^i_j 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^i_j (note that d^I_j uses V^{imax}_j values) and use these together with the equations in the previous subsection to calculate all of the

values of V_j^{imax-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 β_j and D_j using equations (4) and (5) and then work down from $jmax$ to determine V_j^i using equation (3). This is not too difficult to program.

7.4 SOR method

The SOR method is a simpler approach but can be slightly less accurate and take a little longer as it relies upon iteration. If we consider each of the individual equations from $\mathbf{AV} = \mathbf{d}$ we have that

$$\begin{aligned} a_1V_0^i + b_1V_1^i + c_1V_2^i &= d_1^i \\ a_2V_1^i + b_2V_2^i + c_2V_3^i &= d_1^i \\ \ldots\ldots\ldots\ldots &= \ldots \\ a_jV_{j-1}^i + b_jV_j^i + c_jV_{j+1}^i &= d_j^i \\ \ldots\ldots\ldots\ldots &= \ldots \\ a_{jmax-1}V_{jmax-2}^i + b_{jmax-1}V_{jmax-1}^i + c_{jmax-1}V_{jmax}^i &= d_{jmax-1}^i \end{aligned}$$

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

$$V_j^i = \frac{1}{b_j}(d_j^i - a_jV_{j-1}^i - c_jV_{j+1}^i)$$

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

$$V_j^{i,k+1} = \frac{1}{b_j}(d_j^i - a_jV_{j-1}^{i,k} - c_jV_{j+1}^{i,k})$$

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

$$V_j^{i,k+1} = \frac{1}{b_j}(d_j^i - a_jV_{j-1}^{i,k+1} - c_jV_{j+1}^{i,k})$$

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 $V_j^{i,k} \to V_i^j$ monotonically in k. So the SOR algorithm says that

$$y_j^{i,k+1} = \frac{1}{b_j}(d_j^i - a_jV_{j-1}^{i,k+1} - c_jV_{j+1}^{i,k})$$

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

where $1 < \omega < 2$ is called the over-relaxation parameter. This can be shown to converge for our values of a and c.

7.5 American options

Adapting the Explicit Method

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

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

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

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

This is similar to how we value American options using the binomial tree, an diagram of this method is shown in figure 7.1.

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

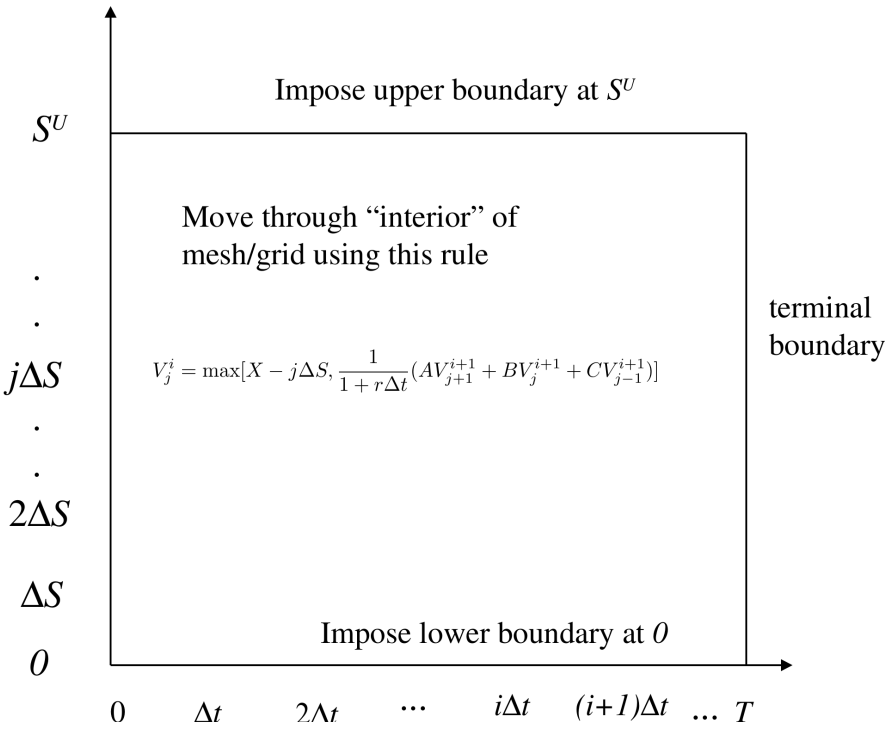

Figure 7.1: An example of the grid for the American put option with an explicit method.

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

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

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

to (in the case of the American put option)

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

$$V_j^{i,k+1} = \max(V_j^{i,k} + \omega(y_j^{i,k+1} - V_j^{i,k}), X - j\Delta S)$$

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 binomial trees, finite-difference 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.

Non-linearity error

The nice thing about finite difference methods in general are that you have the freedom to construct the grid as desired and so it is quite simple to construct the grid so that you have a grid point upon any discontinuities.

For example, if we consider an European call or put option then the only source of non-linearity error is at $S = X$ at expiry. Thus you should always choose ΔS so that $X = j\Delta S$ for some integer value of j. So if in this case $S_0 = 100$ and $X = 95$, you need a suitably large S^U and a ΔS which is a divisor of 95. Thus if you want 5000 S steps then a reasonable choice for ΔS may be 0.076, which give $S^U = 4 \times 95 = 380$ and $j = 1250$ is the position of the exercise price.

Barrier options

As we have seen when pricing barrier options with the binomial method, there is a large amount of non-linearity error that comes from not having the nodes in the tree aligned with the position of the barrier. Thus with barrier options we have two sources of non-linearity error, the error from the barrier and the error from the discontinuous payoff. However, with the finite difference grid it is relatively easy to align the grid so that you have nodes on the barrier and on the exercise price at expiry.

For a down and out barrier option choose S^L (the lower value of S) to be on the barrier and then, as in the previous example, choose ΔS so that the exercise price is also on a node. For example if $B = 90$ and $X = 95$ if you require 5000 steps, choose $\Delta S = 0.05$, so that $S^U = 340$, and the barrier is at $j = 0$ and the exercise price is at $j = 100$.

7.6 Overview

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. Finally, we saw that with finite-difference methods as you can choose the dimensions of the grid so as to remove the nonlinearity error.