Topic 13

Finite-difference methods

Overview of this topic:

- Solving the Black-Scholes PDE numerically, by discretisation.
- In fact, we present methods for solving more general PDEs:
  - Explicit method
  - Implicit method
  - Crank-Nicolson method
- Material is in Capinski and Zastawniak, Ch. 6.

Many textbooks cover the same material in nearly identical ways. For consistency, we will follow the notation used in Capinski and Zastawniak. We will not discuss details of C++ implementation in the lectures, and so you will definitely need to consult your own copy of this book in the lab sessions and in your own time. As usual, please ask if you need help.

13.1 Parabolic partial differential equations (PDEs)

(Ref: §6.1 of M.J. Capinski and T. Zastawniak, *Numerical Methods in Finance with C++*, CUP, 2012)

- General parabolic PDE (2nd order in x, 1st order in t)

  - Example: Heat equation in physics: $a(t,x) = k$, $b(t,x) = c(t,x) = d(t,x) = 0$.

- Wish to solve for the function $v \equiv v(t,x)$ in the rectangular domain

$$0 \leq t \leq T, \qquad\qquad x_l \leq x \leq x_u.$$

- Also need three (Dirichlet) boundary conditions:

$$\begin{array}{ll} \text{Terminal bc:} & v(T,x) = f(x) \\ \text{Lower bc:} & v(t,x_l) = f_l(t) \\ \text{Upper bc:} & v(t,x_u) = f_u(t) \end{array}$$

where $f(x)$, $f_l(t)$ and $f_u(t)$ are given functions.

13.1 Parabolic partial differential equations (PDEs) (continued)

Diagram:

Black-Scholes equation

The BS equation

where $v \equiv v(t, S)$ is the price of some option, can be written in the previous form:

- S is equivalent to our 'space' coordinate x
- Coefficients given by:

$$\begin{aligned} a(t,x) &= -\tfrac{1}{2}\,\sigma^2 x^2, \\ b(t,x) &= -rx, \\ c(t,x) &= r, \\ d(t,x) &= 0. \end{aligned}$$

Note: Easy to generalise, e.g. to local volatility models where $\sigma \equiv \sigma(x,t)$, see Topic 10.

Boundary conditions

Many different types of option (European calls and puts, some barrier options, etc.) satisfy the same equation (PDE), but with different boundary conditions.

- Terminal boundary condition is set to be payoff of option at time T.
- Upper and lower boundary conditions can usually be derived from heuristic arguments.

Example: European put option with expiry date T and strike K.

- If $S(t)$ is high, then option is almost worthless.
  $\Longrightarrow$ choose x_u big, and set $f_u(t) = 0$.
- If $S(t)$ is low, then assume that option will almost certainly be exercised at expiry, with a payoff equal to K.
  $\Longrightarrow$ choose $x_l = 0$, and set $f_l(t) = e^{-r(T-t)}K$.

Please see your copy of Capinski and Zastawniak for details of call option.

13.2 Numerical solution of the PDE

So far we have described the (exact) mathematical problem.

We now look at several methods for actually solving the problem numerically:

- All these methods involve discretising the problem,
  - i.e. solving the problem on a finite set of points.
- Unlike the binomial and trinomial trees, here the points will be arranged in a rectangular lattice or grid, i.e. (t_i, x_j).
- The difference between the numerical methods lies in the way in which we take the exact PDE and approximate it on the lattice.

Please refer to your copy of Capinski and Zastawniak for full details (and sample C++ code) – we give only an overview of the key ideas here.

Discretisation of PDE

(Ref: §6.2 of M.J. Capinski and T. Zastawniak, *Numerical Methods in Finance with C++*, CUP, 2012 – we follow their notation)

We discretise both the time (t) and "space" (x or S) variables:

$$\text{Time:} \qquad \Delta t = \frac{T}{i_{\text{max}}} \qquad t_i = i\,\Delta t \qquad \text{for } i = 0, \ldots, i_{\text{max}},$$

$$\text{Space:} \qquad \Delta x = \frac{x_u - x_l}{j_{\text{max}}} \qquad x_j = x_l + j\,\Delta x \qquad \text{for } j = 0, \ldots, j_{\text{max}}.$$

$\Longrightarrow$ only need to evaluate the coefficients of the PDE, the boundary conditions, and the function v itself, at <u>discrete</u> points. So let us define:

$$\left.\begin{array}{c} a_{ij} = a(t_i, x_j), \quad b_{ij} = b(t_i, x_j), \\ c_{ij} = c(t_i, x_j), \quad d_{ij} = d(t_i, x_j), \\ f_j = f(x_j) \qquad f_{l,i} = f_l(t_i) \qquad f_{u,i} = f_u(t_i). \end{array}\right\} \quad \text{Known}$$

$$\left. v_{ij} = v(t_i, x_j), \right\} \quad \text{To be determined}$$

13.3 Method I: Explicit method

Now we address the first technique for solving the equation numerically.

Using the Taylor series expansion about (t_i, x_j), we approximate the derivatives in the original PDE with differences that can be calculated using the function values on specific gridpoints:

$$\left.\frac{\partial v}{\partial t}\right|_{\substack{t=t_i\\x=x_j}} \approx \boxed{\qquad\qquad\qquad}$$

$$\left.\frac{\partial v}{\partial x}\right|_{\substack{t=t_i\\x=x_j}} \approx \boxed{\qquad\qquad\qquad}$$

$$\left.\frac{\partial^2 v}{\partial x^2}\right|_{\substack{t=t_i\\x=x_j}} \approx \boxed{\qquad\qquad\qquad}$$

We now substitute these into the original PDE:

13.3 Method I: Explicit method (continued)

On rearranging, the discretised PDE becomes

$$\boxed{v_{i-1,j} = A_{ij} v_{i,j-1} + B_{ij} v_{ij} + C_{ij} v_{i,j+1} + D_{ij}}$$

for $1 \le i \le i_{\mathsf{max}}$ and $1 \le j \le j_{\mathsf{max}} - 1$ where

$$A_{ij} = \frac{\Delta t}{\Delta x}\left(\frac{b_{ij}}{2} - \frac{a_{ij}}{\Delta x}\right), \qquad B_{ij} = 1 - \Delta t\, c_{ij} + \frac{2\Delta t\, a_{ij}}{(\Delta x)^2},$$

$$C_{ij} = -\frac{\Delta t}{\Delta x}\left(\frac{b_{ij}}{2} + \frac{a_{ij}}{\Delta x}\right), \qquad D_{ij} = -\Delta t\, d_{ij}.$$

(Recall: We already know the values of A_{ij}, B_{ij}, C_{ij} and D_{ij} for all i and j.)

Furthermore, the v_{ij} on the top, bottom and right boundaries can be calculated immediately from the boundary conditions:

Right:	$v_{i_{\mathsf{max}},j} = f_j$	for $j = 0, \ldots, j_{\mathsf{max}}$
Bottom:	$v_{i,0} = f_{l,i}$	for $i = 0, \ldots, i_{\mathsf{max}} - 1$
Top:	$v_{i,j_{\mathsf{max}}} = f_{u,i}$	for $i = 0, \ldots, i_{\mathsf{max}} - 1.$

We can therefore solve PDE as follows:

1. Populate points on right-hand column for $j = 0, \dots, j_{\text{max}}$, using the final BC:

$$v_{i_{\text{max}},j} = f_j .$$

2. Populate points on top/bottom rows for $i = 0, \dots, i_{\text{max}} - 1$, using the appropriate BCs:

$$v_{i,0} = f_{l,i}, \qquad v_{i,j_{\text{max}}} = f_{u,i}.$$

Now, repeat the following step i_{max} times, starting with $i = i_{\text{max}}$ and finishing with $i = 1$:

3. For each $j = 1, \dots, j_{\text{max}} - 1$, compute v_{i-1}, j using

$$v_{i-1,j} = A_{ij} v_{i,j-1} + B_{ij} v_{ij} + C_{ij} v_{i,j+1} + D_{ij},$$

   and expressions for A_{ij}, B_{ij}, C_{ij} and D_{ij} given on the previous slide.
   (<u>Note</u>: Similar to <u>tri</u>nomial tree considered in coursework!)

Finally:

4. Price of the option today is given by $v(0, S_0)$.
   - $(0, S_0)$ might not be a gridpoint $\Longrightarrow$ must use (linear) interpolation.

Accuracy and stability of explicit method

For the explicit method . . .

- For accuracy, must choose a small Δx (and a small time step Δt as well).
  - Minimises "truncation error".
- But even so, explicit method not guaranteed to converge!
  - For stability, we must also have $\boxed{\dfrac{\Delta t}{(\Delta x)^2} \lesssim |2a|^{-1}}$
  - This puts serious limitation on the size of the time step.
  - See, e.g. P. Wilmott, *Derivatives*, pp. 629-633, for further details.

13.4 Method II: Implicit method

(Ref: §6.3 of M.J. Capinski and T. Zastawniak, *Numerical Methods in Finance with C++*, CUP, 2012)

We now present an alternative to the explicit method which is much more stable.

- No restriction on Δt, so can make Δx as small as we want, thereby increasing accuracy.

We use a Taylor series expansion about the point (t_{i-1}, x_j):

$$\left.\frac{\partial v}{\partial t}\right|_{\substack{t=t_{i-1}\\x=x_j}} \approx \boxed{}$$

$$\left.\frac{\partial v}{\partial x}\right|_{\substack{t=t_{i-1}\\x=x_j}} \approx \boxed{}$$

$$\left.\frac{\partial^2 v}{\partial x^2}\right|_{\substack{t=t_{i-1}\\x=x_j}} \approx \boxed{}$$

13.4 Method II: Implicit method (continued)

We insert these into the original PDE as before (see your copy of Capinski and Zastawniak).

- This gives a difference equation involving $v_{i-1,j-1}$, $v_{i-1,j}$, $v_{i-1,j+1}$ and v_{ij}.

Now . . .

- We still have to calculate by iterating through the grid from right to left.
  - This is because we know the boundary condition (payoff) at timestep $i = i_{\text{max}}$ and we wish to find today's price (at timestep $i = 0$).
- However, the equation now involves $v_{i-1,j-1}$, $v_{i-1,j}$ and $v_{i-1,j+1}$ at timestep $i-1$, and only v_{ij} at timestep i.
- It appears that we have three quantities to determine for timestep $i-1$, with only one piece of information about timestep i.

Can this be solved? . . .

13.4 Method II: Implicit method (continued)

Yes! But how?

- We observe that the equations for different values of j involve the same values of the option price at timestep $i-1$.
- So we now have to solve a set of coupled equations.
  - We can no longer just write down an explicit equation for $v_{i-1,j}$ as we did before.
  - Rather, the solution is now implicit.
  - We have to solve a set of simultaneous equations (i.e. for all j), which will involve inverting a matrix.

In fact, we will not study the implicit method in depth, because . . .

13.5 Method III: Crank-Nicolson method

There is an even better method, due to Crank and Nicolson (1947).

- Combines both the explicit and the implicit methods.
- No more difficult to implement than the implicit scheme
  - In fact, the implicit scheme can be implemented using the same coding framework.
- We therefore consider only the Crank-Nicolson scheme in the rest of the module.

We Taylor-expand around the point $(t_i - \frac{1}{2}\Delta t, x_j)$.

13.5 Method III: Crank-Nicolson method (continued)

Equations are:

$$\left.\frac{\partial v}{\partial t}\right|_{\substack{t=t_i-\Delta t/2\\ x=x_j}} \approx \boxed{\quad}$$

$$\left.\frac{\partial v}{\partial x}\right|_{\substack{t=t_i-\Delta t/2\\ x=x_j}} \approx \boxed{\quad}$$

$$\left.\frac{\partial^2 v}{\partial x^2}\right|_{\substack{t=t_i-\Delta t/2\\ x=x_j}} \approx \boxed{\quad}$$

$$v\left(t_i - \tfrac{\Delta t}{2}, x_j\right) \approx \boxed{\quad}$$

Substitute these into the PDE, and rearrange (see Capinski and Zastawniak for details):

$$\boxed{E_{ij} v_{i-1,j-1} + F_{ij} v_{i-1,j} + G_{ij} v_{i-1,j+1} = A_{ij} v_{i,j-1} + B_{ij} v_{ij} + C_{ij} v_{i,j+1} + D_{ij} \qquad (*)}$$

for $1 \le i \le i_{\text{max}}$ and $1 \le j \le j_{\text{max}} - 1$, where

$$A_{ij} = \frac{\Delta t}{2\Delta x}\left(\frac{b_{i-1/2,j}}{2} - \frac{a_{i-1/2,j}}{\Delta x}\right), \qquad E_{ij} = -A_{ij},$$

$$B_{ij} = \frac{\Delta t}{2}\left(\frac{2a_{i-1/2,j}}{(\Delta x)^2} - c_{i-1/2,j}\right) + 1, \qquad F_{ij} = 2 - B_{ij},$$

$$C_{ij} = -\frac{\Delta t}{2\Delta x}\left(\frac{b_{i-1/2,j}}{2} + \frac{a_{i-1/2,j}}{\Delta x}\right), \qquad G_{ij} = -C_{ij},$$

$$D_{ij} = -\Delta t \, d_{i-1/2,j}.$$

We also have the boundary conditions

$$v_{i,0} = f_{l,i}, \qquad v_{i,j_{\text{max}}} = f_{u,i} \qquad \text{for } 0 \le i \le i_{\text{max}} - 1, \qquad (*)$$

$$v_{i_{\text{max}},j} = f_j \qquad \text{for } 0 \le j \le j_{\text{max}}. \qquad (*)$$

Can combine equations $(*)$ into a set of matrix equations (one for each $1 \leq i \leq i_{\mathsf{max}}$)

$$\boxed{\mathbf{B}_i \mathbf{v}_{i-1} = \mathbf{A}_i \mathbf{v}_i + \mathbf{w}_i}$$

or equivalently

$$\boxed{\mathbf{v}_{i-1} = \mathbf{B}_i^{-1} \left(\mathbf{A}_i \mathbf{v}_i + \mathbf{w}_i \right)}$$

where $\mathbf{v}_i$ and $\mathbf{w}_i$ are vectors of size $j_{\mathsf{max}} - 1$, given by

$$\mathbf{v}_i = \begin{pmatrix} v_{i,1} \\ \vdots \\ v_{i,j_{\mathsf{max}}-1} \end{pmatrix}, \qquad \mathbf{w}_i = \begin{pmatrix} D_{i,1} + A_{i,1} f_{l,i} - E_{i,1} f_{l,i-1} \\ D_{i,2} \\ \vdots \\ D_{i,j_{\mathsf{max}}-2} \\ D_{i,j_{\mathsf{max}}-1} + C_{i,j_{\mathsf{max}}-1} f_{u,i} - G_{i,j_{\mathsf{max}}-1} f_{u,i-1} \end{pmatrix},$$

(note that $D_{ij} = 0$ for the BS equation)

...and where $\mathbf{A}_i$ and $\mathbf{B}_i$ are square tridiagonal matrices, given by

$$\mathbf{A}_i = \begin{pmatrix} B_{i,1} & C_{i,1} & 0 & 0 & \cdots & 0 \\ A_{i,2} & B_{i,2} & C_{i,2} & 0 & \cdots & 0 \\ 0 & A_{i,3} & B_{i,3} & C_{i,3} & \ddots & \vdots \\ 0 & 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & C_{i,j_{\mathsf{max}}-2} \\ 0 & 0 & \ldots & 0 & A_{i,j_{\mathsf{max}}-1} & B_{i,j_{\mathsf{max}}-1} \end{pmatrix},$$

$$\mathbf{B}_i = \begin{pmatrix} F_{i,1} & G_{i,1} & 0 & 0 & \cdots & 0 \\ E_{i,2} & F_{i,2} & G_{i,2} & 0 & \cdots & 0 \\ 0 & E_{i,3} & F_{i,3} & G_{i,3} & \ddots & \vdots \\ 0 & 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & G_{i,j_{\mathsf{max}}-2} \\ 0 & 0 & \ldots & 0 & E_{i,j_{\mathsf{max}}-1} & F_{i,j_{\mathsf{max}}-1} \end{pmatrix}.$$

Loosely speaking, $\mathbf{A}_i$ is the explicit part, and $\mathbf{B}_i$ is the implicit part.

Our challenge, then, is to solve a problem of the following generic form (for each time step):

Find the vector $\mathbf{p}$ where

$$\boxed{\mathbf{p} = \mathbf{B}^{-1}\mathbf{q}}$$

and matrix $\mathbf{B}$ and vector $\mathbf{q}$ are known.

If solving numerically, it turns out that

1. We do not need to determine the inverse of the matrix $\mathbf{B}$ explicitly.
2. Our problem is made (much) easier by the fact that the matrix $\mathbf{B}$ is tridiagonal.

Consequently, we can use a simple iterative method for finding $\mathbf{p}$.

- The method used is a particular case of the LU decomposition.
- See Capinski and Zastawniak (pp. 144 and 162) for details of the algorithm (also known as the Thomas algorithm), including proof.

13.5 Method III: Crank-Nicolson method (continued)

The complete algorithm for CN method is similar to explicit case (see slide 11),

except that, in Step 3, we use (for each time step i)

$$\mathbf{v}_{i-1} = \mathbf{B}_i^{-1}(\mathbf{A}_i\mathbf{v}_i + \mathbf{w}_i)$$

to generate the $v_{i-1,j}$ for all values of j simultaneously ($1 \le j \le j_{\text{max}} - 1$), by using the LU decomposition method.

Please work through the code in Capinski and Zastawniak by yourselves.

13.6 Coordinate changes

(Ref: §6.4 of M.J. Capinski and T. Zastawniak, *Numerical Methods in Finance with C++*, CUP, 2012)

We can change variables in the PDE (e.g. from S to $\log S$), and solve the new equation numerically.

- This can give more accurate results, and increase the stability of the method.
- Very important in practice.
- Please work through this section of Capinski and Zastawniak in the labs.

13.7 American options

(Ref: §6.5 of M.J. Capinski and T. Zastawniak, *Numerical Methods in Finance with C++*, CUP, 2012)

Recall: For an American option, the price can never fall below the early exercise value.

Suppose $g_{ij} \equiv g(t_i, x_j)$ is the early exercise value at node (i, j).

Then, in the explicit method, we have simply

$$\boxed{v_{i-1,j} = \max\left[A_{ij}v_{i,j-1} + B_{ij}v_{ij} + C_{ij}v_{i,j+1} + D_{ij} \ ,\ g_{i-1,j}\right]}$$

replacing the corresponding formula earlier (see slide 11).

We also modify the boundary conditions in the same way.

So, for each time step i, we proceed exactly as for a European option, except that:

- If the value $v_{i-1,j}$ that we calculate at any node is less than the early exercise payoff ...
- ... then we merely set $v_{i-1,j}$ to be that payoff instead.

Please work through the details of this section of Capinski and Zastawniak in the labs.

Pricing American options with the implicit or Crank-Nicolson schemes is more complicated.

- Cannot do the 'backward' induction step and the early exercise step as 2 separate stages.
  - At least, not whilst retaining accuracy.

We have to use an iterative scheme at each timestep.

- e.g. the successive over-relaxation (SOR) method (and, in particular, projected SOR).
- This replaces the LU decomposition method that we saw earlier.

See, e.g. P. Wilmott, *Derivatives*, §47.3.6, §47.9.2 and §50.3 for details of the SOR method, and an implementation in Visual Basic.