Computational Finance with C++

Lecture 6: Finite Difference Methods

Panos Parpas
Imperial College London
p.parpas@imperial.ac.uk

Outline

- Parabolic partial differential equations
- Explicit Methods
- Implicit Methods
- Reading:
    - ✦Chapter 6
      Capinski+Zastawniak,Numerical
      Methods in finance with C++

Parabolic Partial Differential Equations

- Consider a general parabolic PDE:

$$\frac{\partial v(t,x)}{\partial t} = a(t,x)\frac{\partial^2 v(t,x)}{\partial x^2} + b(t,x)\frac{\partial v(t,x)}{\partial x} c(t,x)v(t,x) + d(t,x),$$
$$V(T,X) = f(x),$$
$$V(t,x_l) = f_l(x),$$
$$V(t,x_u) = f_u(x),$$

- We look for solution in: $[0,T] \times [x_l, x_u]$
- Last three equations are called **boundary conditions:** (terminal, lower and upper)

Parabolic PDE and Black-Scholes Model

- Denote price of asset at time t: $S(t)$
- Price of financial derivative at time t: $H(t) = u(t, S(t))$
- Black-Scholes PDE:

$$\frac{\partial u(t,z)}{\partial t} + \frac{\sigma^2}{2} z^2 \frac{\partial^2 u(t,z)}{\partial z^2} + rz \frac{\partial u(t,z)}{\partial z} - ru(t,z) = 0$$

- Related to general parabolic PDE via:

$$a(t,z) = -\frac{\sigma^2}{2} z^2 \quad b(t,z) = -rz$$

$$c(t,z) = r \quad d(t,z) = 0$$

Still need to **derive boundary conditions**.

Boundary Conditions for the Black-Scholes Model

- Suppose the payoff is given by:
- e.g. for European Put: $u(T, z) = \max(K - z, 0)$
- Lower/upper bounds depend on option, usually derived via some financial insight e.g. arbitrage

Example: Upper Lower conditions for European Put.

- If S(t) is "high" then option is worthless e.g.

$$u(t, z_u) = h_u^{\text{put}}(t) = 0$$

- If S(t) is "low" then option is equal to (discounted) payoff:

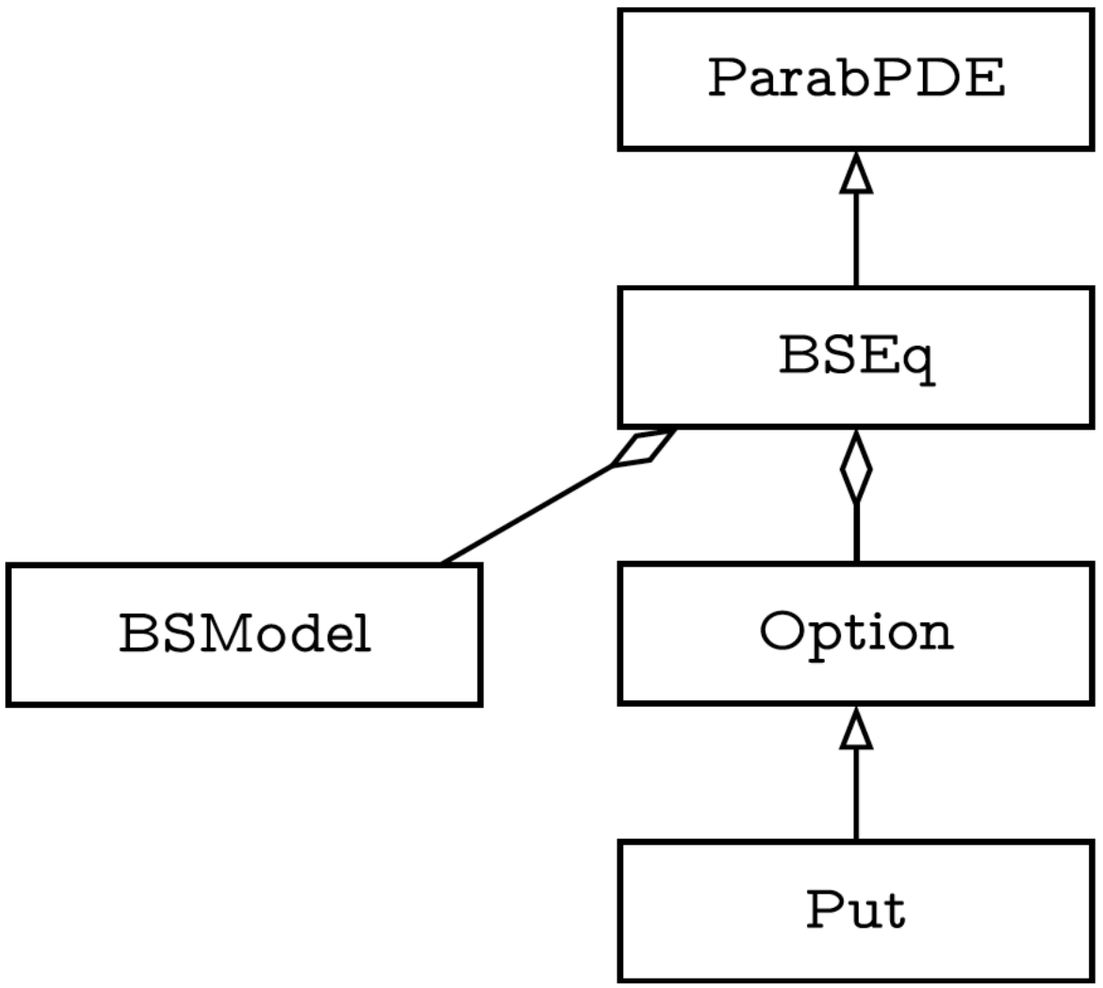
$$u(t, z_l) = h_l^{\text{put}}(t) = e^{-r(T-t)}K$$

$\text{Typically } z_l = 0, \ z_u = 2S(0).$

Exercise 6.1: Use the boundary conditions derived for European puts in the previous slide and put-call parity to derive formulae for the upper and lower boundary conditions for a European call option.

Reminder put call parity: C-P=S-DK (where C is call price, P is put price, S is spot price, K is strike, and D is discount factor from expiry e^(-rT)).

Code Review: `Option.h/Cpp, ParabPDE.h/cpp, BSE.h/cpp`



Basic UML (Universal Modelling Language notation)

arrow = "is a"

rhombus= "has a"

Exercise 6.2: Expand the code to include European calls.

Finite Difference Methods

- Not always possible to find a closed form solution
- Can only find a solution on a finite set of points:

$$\{(t_i, x_j) \mid i = 0, \ldots, i_{\max} \text{ and } j = 0, \ldots, j_{\max}\}$$

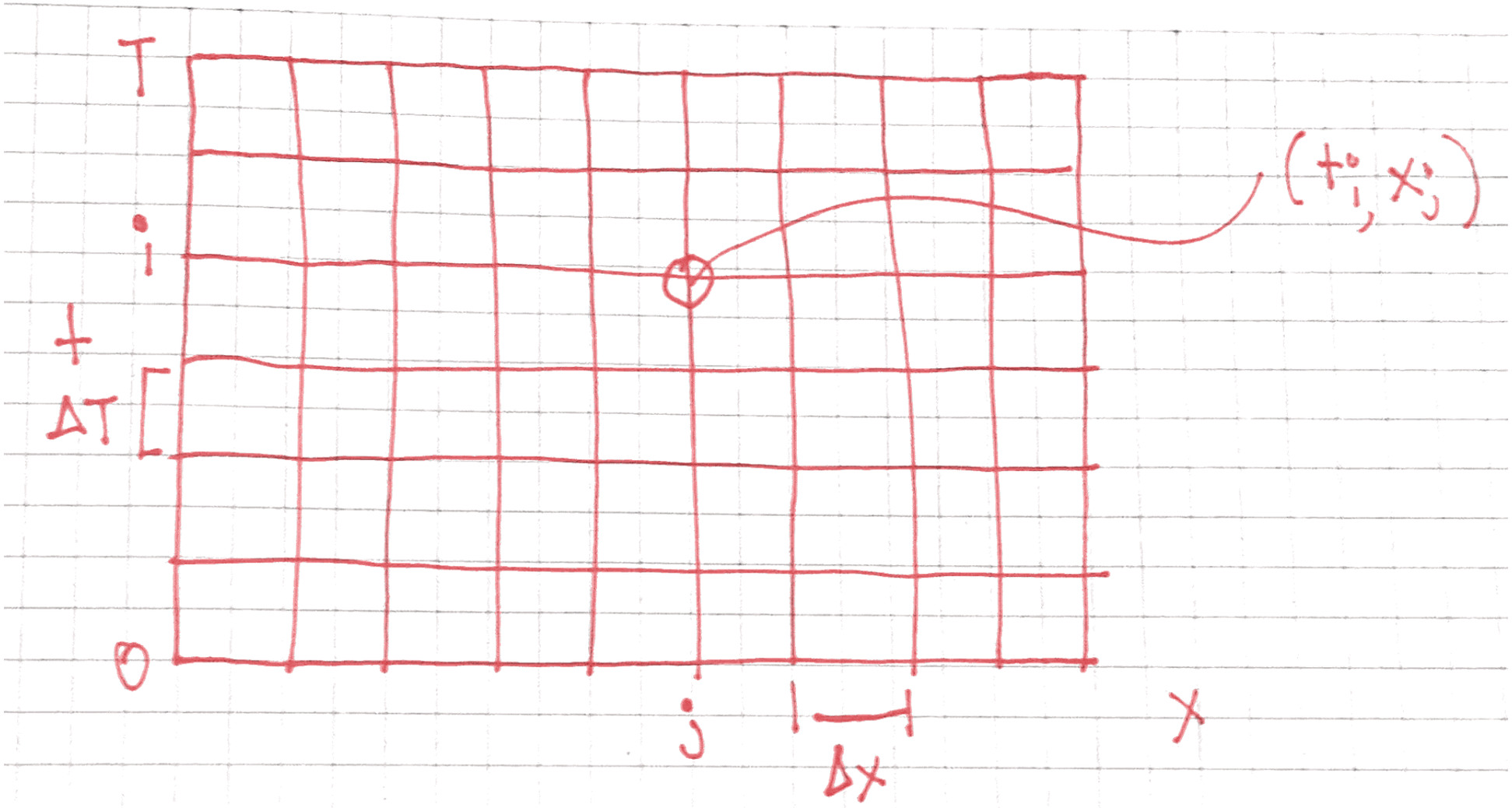
- Due to the type of approximations we call the techniques ***finite difference*** methods
- In engineering/science applications ***finite element*** methods are more appropriate
- When boundary conditions are relatively simple (as they usually are in finance) finite difference methods are more than enough.

Explicit Finite Difference Methods

$$\{(t_i, x_j) \mid i = 0, \ldots, i_{\max} \text{ and } j = 0, \ldots, j_{\max}\}$$

Let $t_i = i\Delta t \qquad x_j = x_l + j\Delta x$

Where $\Delta t = \dfrac{T}{i_{\max}} \qquad \Delta x = \dfrac{x_u - x_l}{j_{\max}}$



Finite Difference Methods

- Notation to represent functions on grid:

$$\nu_{i,j} = \nu(t_i, x_j) \text{ :option price}$$

- PDE coefficients:

$$a_{i,j} = a(t_i, x_j)$$
$$b_{i,j} = b(t_i, x_j)$$
$$c_{i,j} = c(t_i, x_j)$$
$$d_{i,j} = d(t_i, x_j)$$

- Coefficients for boundary conditions:

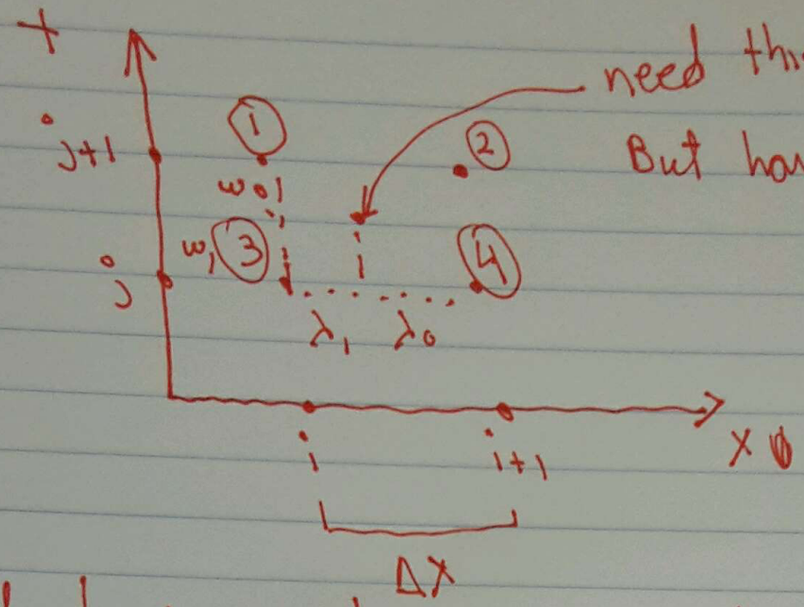
$$f_j = f(x_j), \quad f_{l,i} = f_l(t_i), \quad f_{u,i} = f_u(t_i)$$

Code Review: `FDMethod.h/Cpp`

- Type casting: (changes type from double to int)

```
int i = (int)(t/dt);
int j = (int)((x-PtrPDE->xl)/dx);
```

- Casting removes “floating” part same effect as rounding down (note that 2.1->2, 2.999->2 etc..)
- Derivation of weights in next slide



need this $v(t,x)$

But have: 1 $v_{i,j+1}$
2 $v_{i+1,j+1}$
3 $v_{i,j}$
4 $v_{i+1,j}$

find $\lambda_0+\lambda_1=1$, $w_0+w_1=1$

such that $t=\lambda_0 t_i+\lambda_1 t_{i+1}$

$x=w_0 x_j+w_1 x_{j+1}$

$\lambda_0=1-\lambda_1 \Rightarrow t=(1-\lambda_1)t_i+\lambda_1 t_{i+1}$

$(t_{i+1}-t_i)\lambda_1=t-t_i$

ie $\lambda_1=\frac{t-t_i}{t_{i+1}-t_i}=\frac{t-t_i}{\Delta t}$

In a similar way

$w_1=\frac{x-x_j}{\Delta x}$ $\quad w_0=1-w_1$

we then Approximate

$v(t,x)\approx \lambda_1 w_1 v_{i+1,j+1}+\lambda_1 w_0 v_{i+1,j}$

$+\lambda_0 w_1 v_{i,j+1}+\lambda_0 w_0 v_{i,j}$

Discretisation

$$\frac{\partial v(t,x)}{\partial t} = a(t,x)\frac{\partial^2 v(t,x)}{\partial x^2} + b(t,x)\frac{\partial v(t,x)}{\partial x} c(t,x)v(t,x) + d(t,x),$$

$$V(T,X) = f(x),$$

$$V(t,x_l) = f_l(x),$$

$$V(t,x_u) = f_u(x),$$

Discretised boundary conditions, we do the derivatives next:

$$\frac{\partial \nu(t_i,x_j)}{\partial t} \approx \frac{\nu_{i,j} - \nu_{i-1,j}}{\Delta t}$$

Forward differencing for time derivative is what makes the method **explicit**

$$\frac{\partial v(t_i,x_j)}{\partial x} \approx \frac{v_{i,j+1} - v_{i,j-1}}{2\Delta x}$$

$$\frac{\partial^2 \nu(t_i,x_j)}{\partial x^2} \approx \frac{\nu_{i,j+1} - 2\nu_{i,j} + \nu_{i,j-1}}{\Delta x^2}$$

Approximation with *central differencing*

Discretisation

Using the finite difference approximations we get:

$$\frac{v_{i,j} - v_{i-1,j}}{\Delta t} = a_{i,j}\frac{v_{i,j+1} - 2v_{i,j} + v_{i,j-1}}{\Delta x^2} + b_{i,j}\frac{v_{i,j+1} - v_{i,j-1}}{2\Delta x} + c_{i,j}v_{i,j} + d_{i,j}$$

Using the above formulas into the BSE we obtain,

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

where,

$$\begin{aligned}
A_{i,j} &= \frac{\Delta t}{\Delta x}(b_{i,j/2} - a_{i,j}/\Delta x) \\
B_{i,j} &= 1 - \Delta t c_{i,j} + 2\Delta t a_{i,j}/\Delta x)^2 \\
C_{i,j} &= -\frac{\Delta t}{\Delta x}(b_{i,j/2} + a_{i,j}/\Delta x) \\
D_{i,j} &= -\Delta t d_{i,j}
\end{aligned}$$

Boundary Conditions

Payoff at time T:

$$v_{i_{\max},j} = v(t_{\max}, x_j) = v(T, x_j) = f(x_j) = f_j$$

Lower Boundary:

$$v_{i,0} = v(t_i, x_0) = v(t_i, x_l) = f_l(t_i) = f_{l,i}$$

Upper Boundary:

$$v_{i,j_{\max}} = v(t_i, x_{j_{\max}}) = v(t_i, x_u) = f_u(t_i) = f_{u,i}$$

Explicit Finite Difference Method

Start at $i = i_{\max}$ and finish at i=0 i.e. work backwards (like binomial tree)

(i) For $i = i_{\max}$ compute $v_{i_{\max},j} = f_j,\ j = 0, \ldots, j_{\max}$
(ii) Apply lower boundary and upper boundary from previous slide
(iii) Compute $v_{i-1,j}$ using:

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

Method called explicit because we have an explicit formula from time i to time i-1.

Code Review: `ExplicitMethod.h/Cpp, Main24.cpp`

Exercise 6.3: Write functions to compute the replicating strategy (i.e. the delta)

Exercise 6.4: Modify the code to price European call options using the explicit scheme. Compare the results using analytic formulae. Experiment with different `i_max` and `j_max` to see when it becomes unstable

Implicit Finite Difference Method

Explicit Method can become **unstable** if stepsize is too small!

e.g. try:

```
int imax=2400, jmax=1000;
int imax=2500, jmax=1000;
```

in `Main24.cpp` also see **Exercise 6.4**

Backward Finite differencing in **time** fixes the stability issue.

$$\frac{\partial v(t_{i-1}, x_j)}{\partial t} \approx \frac{v_{i,j} - v_{i-1,j}}{\Delta t}$$

$$\frac{\partial v(t_{i-1}, x_j)}{\partial x} \approx \frac{v_{i-1,j+1} - v_{i-1,j-1}}{2\Delta x}$$

$$\frac{\partial^2 v(t_{i-1}, x_j)}{\partial x^2} \approx \frac{v_{i-1,j+1} - 2v_{i-1,j} + v_{i-1,j-1}}{\Delta x^2}$$

Implicit Finite Difference Method

Backward Finite differencing in **time** fixes the stability issue.

$$\frac{\partial v(t_{i-1}, x_j)}{\partial t} \approx \frac{v_{i,j} - v_{i-1,j}}{\Delta t}$$

$$\frac{\partial v(t_{i-1}, x_j)}{\partial x} \approx \frac{v_{i-1,j+1} - v_{i-1,j-1}}{2\Delta x}$$

$$\frac{\partial^2 v(t_{i-1}, x_j)}{\partial x^2} \approx \frac{v_{i-1,j+1} - 2v_{i-1,j} + v_{i-1,j-1}}{\Delta x^2}$$

Implicit

$$\frac{\partial \nu(t_i, x_j)}{\partial t} \approx \frac{\nu_{i,j} - \nu_{i-1,j}}{\Delta t}$$

$$\frac{\partial v(t_i, x_j)}{\partial x} \approx \frac{v_{i,j+1} - v_{i,j-1}}{2\Delta x}$$

$$\frac{\partial^2 \nu(t_i, x_j)}{\partial x^2} \approx \frac{\nu_{i,j+1} - 2\nu_{i,j} + \nu_{i,j-1}}{\Delta x^2}$$

Explicit

Crank-Nicolson Method

- Depending on the choice of discretisation can get different methods
- Crank-Nicolson is very popular
- Both Crank-Nicolson and Implicit methods require the solution of a linear system of equations
- In your textbook LU decomposition is discussed
- Conjugate Gradient (from coursework) can also be used