A MATLAB and Python implementation of Finite Difference method for Heat and Black-Scholes Partial Differential Equation

This is a series of MATLAB functions and two Python files in object-oriented discipline to implement finite difference method, a common numerical method to get numerical solutions for Partial Differential Equations.

Three common Schemes for FD method: Explicit, Implicit and Crank-Nocolson, the difference among them are largely difference of diffsion method of different terms in the two PDE.

Author: Ruinan Lu

References:

[1]Brandimarte P. Numerical methods in finance and economics: a MATLAB-based introduction[M]. John Wiley & Sons, 2013.

[2]Seydel R, Seydel R. Tools for computational finance[M]. Berlin: Springer, 2006.

[3]Ramalho L. Fluent python: Clear, concise, and effective programming[M]. " O'Reilly Media, Inc.", 2015.

Partial Differential Equations

A dimensionless form of Heat Equation is:

$$\frac{\partial \phi}{\partial t} = \frac{\partial^2 \phi}{\partial x^2}$$

Black-Scholes Partial Differential Equation:

$$\frac{\partial V}{\partial t} + rS\frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} = rV$$

Approximation for the derivative

Forward approximation for the derivative:$f'(x) = \frac{f(x+h)-f(x)}{h} + O(h)$

Forward approximation for partial derivative:$\frac{\partial \phi}{\partial t} = \frac{\phi_{i,j+1}-\phi_{i,j}}{dt}$

Backward approximation for the derivative:$f'(x) = \frac{f(x)-f(x-h)}{h} + O(h)$

Backward approximation for partial derivative:$\frac{\partial \phi}{\partial t} = \frac{\phi_{i,j}-\phi_{i,j-1}}{dt}$

Central approximation for the derivative:$f'(x) = \frac{f(x+h)-f(x-h)}{2h} + O(h^2)$

Central approximation for partial derivative: $\frac{\partial \phi}{\partial t} = \frac{\phi_{i,j+1}-\phi_{i,j-1}}{2dt}$

Second-order derivatives approximation:$f''(x) = \frac{f(x+h)-2f(x)+f(x-h)}{h^2} + O(h^2)$

Second-order partial derivative:$\frac{\partial^2 \phi}{\partial x^2} = \frac{\phi_{i+1,j+}-2\phi_{i,j}+\phi_{i-1,j}}{(dx)^2}$

Schemes

Explicit method for Heat Equation

Use forward approximation for partial derivative:

$$\frac{\phi_{i,j+1}-\phi_{i,j}}{dt} = \frac{\phi_{i+1,j}-2\phi_{i,j}+\phi_{i-1,j}}{(dx)^2} \tag{1}$$

Rearange the equation we get:

$\phi_{i,j+1} = \rho\phi_{i-1,j} + (1-2\rho)\phi_{i,j} + \rho\phi_{i+1,j}$

where $\rho = \frac{dt}{(dx)^2}$

Represent the equation with matrix multiplication we get

$\boldsymbol{\Phi_{j+1}} = \boldsymbol{A\Phi_j} + \rho\boldsymbol{g}_j, j = 0, 1, 2\cdots$

where $\boldsymbol{A} \in R^{N-1,N-1}$,

$$\boldsymbol{A} = \begin{bmatrix} 1-2\rho & \rho & 0 & \cdots & 0 & 0 \\ \rho & 1-2\rho & \rho & \cdots & 0 & 0 \\ 0 & \rho & 1-2\rho & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \rho & 1-2\rho \end{bmatrix}, \boldsymbol{\Phi_j} = \begin{bmatrix} \phi_{1,j} \\ \phi_{2,j} \\ \vdots \\ \phi_{N-1,j} \end{bmatrix}, \boldsymbol{g_j} = \begin{bmatrix} f_{0,j} \\ 0 \\ \vdots \\ f_{N,j} \end{bmatrix}$$

Stability analysis shows that the explicit method is stable when $||\boldsymbol{A}||_\infty \leq 1$.

The code for both MATLAB and Python would give warning when the stability condition is not satisfied.

Explicit method for BS PDE

Use backward approximation for $\frac{\partial V}{\partial t}$, Central approximation for $\frac{\partial V}{\partial S}$, we get the Explicit method for BS PDE:

$$\frac{f_{i,j}-f_{i,j-1}}{dt} + ridS\frac{f_{i+1,j}-f_{i-1,j}}{2dS} + \frac{1}{2}\sigma^2 i^2 (dS)^2 \frac{f_{i+1,j}-2f_{i,j}+f_{i-1,j}}{(dS)^2} = rf_{i,j} \tag{2}$$

rearange to get: $f_{i,j-1} = a_i f_{i-1,j} + b_i f_{i,j} + c_i f_{i+1,j}, j = N-1, N-2, \cdots, 0; i = 1, 2, \cdots, M-1,$

where $a_i = \frac{1}{2}dt(\sigma^2 i^2 - ri), b_i = 1 - dt(\sigma^2 i^2 + r), c_i = \frac{1}{2}dt(\sigma^2 i^2 + ri)$

Please note that Explicit method may have stability issue under certain conditions.

Because this is the simplest method, I used double circulation nesting in my MATLAB code to iterate forward in the grid to make it easier to understand.

Implicit method for Heat Equation

Use backward approximation for $\frac{\partial V}{\partial t}$, we get a very different method:

$$\frac{\phi_{i,j}-\phi_{i,j-1}}{dt} = \frac{\phi_{i+1,j}-2\phi_{i,j}+\phi_{i-1,j}}{(dx)^2} \tag{3}$$

Rearange to:

$-\rho\phi_{i-1,j} + (1+2\rho)\phi_{i,j} - \rho\phi_{i+1,j} = \phi_{i,j-1}$

where $\rho = \frac{dt}{(dx)^2}$

In matrix form we have:

$\boldsymbol{B\Phi_{j+1}} = \boldsymbol{\Phi_j} + \rho \boldsymbol{g}_j, j = 0, 1, 2\cdots$, where $\boldsymbol{B} \in R^{N-1,N-1}$

$$\boldsymbol{B} = \begin{bmatrix} 1+2\rho & -\rho & 0 & \cdots & 0 & 0 \\ -\rho & 1+2\rho & -\rho & \cdots & 0 & 0 \\ 0 & -\rho & 1+2\rho & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -\rho & 1+2\rho \end{bmatrix}$$

and solved by $\boldsymbol{\Phi_{j+1}} = \boldsymbol{B}^{-1}(\boldsymbol{\Phi_j} + \rho \boldsymbol{g}_j), j = 0, 1, 2\cdots$

Also, it should be noted that the Implicit scheme is unconditionally stable.

Implicit method for BS PDE

Use forward approciation for $\frac{\partial V}{\partial t}$, we get the Implicit method for BS PDE:

$$\frac{f_{i,j+1} - f_{i,j}}{dt} + ridS\frac{f_{i+1,j} - f_{i-1,j}}{2dS} + \frac{1}{2}\sigma^2 i^2 (dS)^2 \frac{f_{i+1,j} - 2f_{i,j} + f_{i-1,j}}{(dS)^2} = rf_{i,j} \qquad (4)$$, rearrange to

$a_i f_{i-1,j} + b_i f_{i,j} + c_i f_{i+1,j} = f_{i,j+1}, i = 1, 2, \cdots, M-1; j = 1, 2, \cdots, N-1$

where $a_i = \frac{1}{2}dt(ri - \sigma^2 i^2), b_i = 1 + dt(\sigma^2 i^2 + r), c_i = -\frac{1}{2}dt(\sigma^2 i^2 + ri)$

The iteration process could be vectorised by

$$\begin{bmatrix} b_1 & c_1 & 0 & \cdots & 0 & 0 \\ a_2 & b_2 & c_2 & \cdots & 0 & 0 \\ 0 & a_3 & b_3 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & a_{M-2} & b_{M-2} & c_{M-2} \\ 0 & 0 & \cdots & 0 & a_{M-1} & b_{M-1} \end{bmatrix} \cdot \begin{bmatrix} f_{1,j} \\ f_{2,j} \\ f_{3,j} \\ f_{4,j} \\ \vdots \\ f_{M-1,j} \end{bmatrix} = \begin{bmatrix} f_{1,j+1} \\ f_{2,j+1} \\ f_{3,j+1} \\ f_{4,j+1} \\ \vdots \\ f_{M-1,j+1} \end{bmatrix} - \begin{bmatrix} a_1 f_{0,j} \\ 0 \\ 0 \\ \vdots \\ 0 \\ c_{M-1} f_{M,j} \end{bmatrix}$$

Also, in MATLAB code I used LU decomposition to make the calculation faster.

Crank-Nicolson method for Heat Equation

Push the implicit method equation (3) one step forward in terms of j to

$$\frac{\phi_{i,j+1} - \phi_{i,j}}{dt} = \frac{\phi_{i+1,j+1} - 2\phi_{i,j+1} + \phi_{i-1,j+1}}{(dx)^2} \qquad (5)$$

Let $\frac{1}{2}(3) + \frac{1}{2}(5)$ for both left hand side and right hand side, we have

$$\frac{\phi_{i,j+1} - \phi_{i,j}}{dt} = \frac{1}{2}\frac{\phi_{i+1,j+1} - 2\phi_{i,j+1} + \phi_{i-1,j+1}}{(dx)^2} + \frac{1}{2}\frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{(dx)^2}$$

and could be rearanged to

$-\rho\phi_{i-1,j+1} + 2(1+\rho)\phi_{i,j+1} - \rho\phi_{i+1,j+1} = \rho\phi_{i-1,j} + 2(1-\rho)\phi_{i,j} + \rho\phi_{i+1,j}$, where $\rho = \frac{dt}{(dx)^2}$

In matrix form we have:

$\boldsymbol{C\Phi_{j+1}} = \boldsymbol{D\Phi_j} + \rho(\boldsymbol{g_{j+1}} + \boldsymbol{g_j}), j = 0, 1, 2\cdots$, where $\boldsymbol{C}, \boldsymbol{D} \in R^{N-1,N-1}$ and

$$\boldsymbol{C} = \begin{bmatrix} 2(1+\rho) & -\rho & 0 & \cdots & 0 & 0 \\ -\rho & 2(1+\rho) & -\rho & \cdots & 0 & 0 \\ 0 & -\rho & 2(1+\rho) & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -\rho & 2(1+\rho) \end{bmatrix},$$

$$\boldsymbol{D} = \begin{bmatrix} 2(1-\rho) & \rho & 0 & \cdots & 0 & 0 \\ \rho & 2(1-\rho) & \rho & \cdots & 0 & 0 \\ 0 & \rho & 2(1-\rho) & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \rho & 2(1-\rho) \end{bmatrix}$$

Also, it should be noted that the Crank-Nicolson scheme is unconditionally stable.

Crank-Nicolson method for PS PDE

Push the implicit method equation (4) one step backward in terms of j to

$$\frac{f_{i,j}-f_{i,j-1}}{dt} + ridS\frac{f_{i+1,j-1}-f_{i-1,j-1}}{2dS} + \frac{1}{2}\sigma^2 i^2 (dS)^2 \frac{f_{i+1,j-1}-2f_{i,j-1}+f_{i-1,j-1}}{(dS)^2} = rf_{i,j-1} \qquad (6)$$

Let $\frac{1}{2}(4) + \frac{1}{2}(6)$ for both left hand side and right hand side, we have

$$\frac{f_{i,j}-f_{i,j-1}}{dt} + \frac{ridS}{2}\left(\frac{f_{i+1,j-1}-f_{i-1,j-1}}{2dS}\right) + \frac{ridS}{2}\left(\frac{f_{i+1,j}-f_{i-1,j}}{2dS}\right) + \frac{\sigma^2 i^2 (dS)^2}{4}\left(\frac{f_{i+1,j-1}-2f_{i,j-1}+f_{i-1,j-1}}{(dS)^2}\right) + \frac{\sigma^2 i^2 (dS)^2}{4}\left(\frac{f_{i+1,j}-2f_{i,j}+f_{i-1,j}}{(dS)^2}\right) = \frac{r}{2}f_{i,j-1} + \frac{r}{2}f_{ij}$$

and could be rearanged to $-a_i f_{i-1,j-1} + (1-b_i)f_{i,j-1} - c_i f_{i+1,j-1} = a_i f_{i-1,j} + (1+b_i)f_{ij} + c_i f_{i+1,j}$, where

$a_i = \frac{dt}{4}(\sigma^2 i^2 - ri)$, $b_i = -\frac{dt}{2}(\sigma^2 i^2 + r)$, $c_i = \frac{dt}{4}(\sigma^2 i^2 + ri)$

The iteration process could be vectorised by

$\boldsymbol{M_1 f_{j-1}} = \boldsymbol{M_2 f_j}$, where

$$\boldsymbol{M}_1 = \begin{bmatrix} 1-b_1 & -c_1 & 0 & \cdots & 0 & 0 \\ -a_2 & 1-b_2 & -c_2 & \cdots & 0 & 0 \\ 0 & -a_3 & 1-b_3 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & -a_{M-2} & 1-b_{M-2} & -c_{M-2} \\ 0 & 0 & \cdots & 0 & -a_{M-1} & 1-b_{M-1} \end{bmatrix},$$

$$\boldsymbol{M}_2 = \begin{bmatrix} 1+b_1 & c_1 & 0 & \cdots & 0 & 0 \\ a_2 & 1+b_2 & c_2 & \cdots & 0 & 0 \\ 0 & a_3 & 1+b_3 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & a_{M-2} & 1+b_{M-2} & c_{M-2} \\ 0 & 0 & \cdots & 0 & a_{M-1} & 1+b_{M-1} \end{bmatrix}$$

Handling American option

For Explicit and Implicit method, handling American option is simple. At each time node, record the payoff of exercising the option at this time node and compare it with the value of European option in the grid, keep the maximum number for each node.

i.e. at the end of each iteration, simply calculate $\boldsymbol{f}_i = max(\boldsymbol{f}_i, payoff)$ and keep iterating will give us the price.

For Crank-Nicolson method, the equation become

$$\boldsymbol{M_1 f_{j-1}} = \boldsymbol{M_2 f_j} + \begin{bmatrix} a_1(f_{0,j-1} + f_{0,j}) \\ 0 \\ \vdots \\ c_{M-1}(f_{M,j-1} + f_{M,j}) \end{bmatrix}$$, which could simply be modified as

$$\boldsymbol{M_1 f_{j-1}} - \begin{bmatrix} a_1 f_{0,j-1} \\ 0 \\ \vdots \\ c_{M-1} f_{M,j-1} \end{bmatrix} = \boldsymbol{M_2 f_j} + \begin{bmatrix} a_1 f_{0,j} \\ 0 \\ \vdots \\ c_{M-1} f_{M,j} \end{bmatrix}$$, which could be acieved in code by modifying M_1 and M_2

Also, pricing American options with Finite Difference method could also be regarded as a non-linear least-square optimization problem with MATLAB in-built optimizer.