Adaptive Operator Splitting Method

Global KU Frontier Spirit
배신영(betrayer@korea.ac.kr)
김준석(junseok_kim@yahoo.com)

Operator Splitting의 이해

Time splitting 또는 Fractional step방법이라 불리는 operator splitting의 기본 아이디어는 다음과 같다.

다음의 initial value equation 이 있다고 가정하자.

$$\frac{\partial u}{\partial t} = Lu$$

여기서 L 은 operator이다. 이것은 u에 대해 m개의 선형결합으로 다시 쓸 수 있다고 가정하자.

$$Lu = L_1 u + L_2 u + \cdots + L_m u.$$

또한 m개의 operator 각각에 대해 time step n부터 n+1까지 변수 u를 업데이트하는 scheme을 알고 있다고 가정하자. 즉,

$$u^{n+1} = U_1(u^n, \Delta t),$$
$$u^{n+1} = U_2(u^n, \Delta t),$$
$$\vdots$$
$$u^{n+1} = U_m(u^n, \Delta t).$$

이제 operator splitting의 한 열은 다음 열에 의해 연쇄적으로 업데이트되면서 n에서 n+1을 얻게 될 것이다.

$$u^{n+\frac{1}{m}} = U_1(u^n, \Delta t),$$
$$u^{n+\frac{2}{m}} = U_2(u^{n+\frac{1}{m}}, \Delta t),$$
$$\vdots$$
$$u^{n+1} = U_m(u^{n+\frac{m-1}{m}}, \Delta t).$$

OS로 푼 2차원 열방정식

우선, 열방정식을 유한 차분법으로 풀어본다.
다음과 같은 경계조건을 만족시키는 열방정식이 있다고 하자.

초기조건 : $u(x,y,0) = \sin(\pi x)\sin(\pi y)$
경계조건 : $u_x(0,y,t) = u_{xx}(1,y,t) = u_{yy}(x,0,t) = u_{yy}(x,1,t) = 0$

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \quad 0 < x < 1,\ 0 < y < 1,\ t > 0$$

이제 2D 열방정식에 함축적 (implicit) splitting scheme을 적용하여 정리하면 다음과 같다.

$$\frac{\tilde{u}_{ij} - u_{ij}^n}{\Delta t} = \frac{\tilde{u}_{i+1,j} - 2\tilde{u}_{ij} + \tilde{u}_{i-1,j}}{h^2},$$
$$\frac{u_{ij}^{n+1} - \tilde{u}_{ij}}{\Delta t} = \frac{u_{i+1,j}^{n+1} - u_{ij}^{n+1} + u_{i-1,j}^{n+1}}{h^2}, \quad \text{where} \quad h = \Delta x = \Delta y$$

이를 위한 MATLAB Code는 다음과 같다.

```
xleft = 0.0; xright = 1.0; yleft = 0.0; yright = 1.0;
Nx = 50;  Ny = 50;
dx = (xright - xleft)/Nx;  dy = (yright - yleft)/Ny;
h = dx;
T=0.1; Nt=100; dt=T/Nt;

alpha = dt/h^2;

Ax(1:Nx, 1:Nx) = 0.0;      % initalization
Ay(1:Ny, 1:Ny) = 0.0;
bx(1:Nx) = 0.0;
by(1:Ny) = 0.0;
u(1:Nx+2,1:Ny+2) = 0.0;

for i = 1:Nx+2             % cell centered grid
    x(i) = (i-1.5)*h;
end
for j = 1:Ny+2
    y(j) = (j-1.5)*h;
end

for i = 2:Nx+1             %initial condition
    for j = 2:Ny+1
        u(i,j) = sin(pi*x(i))*sin(pi*y(j));
    end
end

u(1,:) = 2.0*u(2,:) - u(3,:); u(Nx+2,:) = 2.0*u(Nx+1,:) - u(Nx,:);   %linear boundary
u(:,1) = 2.0*u(:,2) - u(:,3); u(:,Ny+2) = 2.0*u(:,Ny+1) - u(:,Ny);

for i=1:Nx+2               %analytic solution
    for j=1:Ny+2
        exact_u(i,j)=sin(pi*x(i))*sin(pi*y(j))*exp(-2.0*pi^2*T);
    end
end

for i=2:Nx                 %construct matrices
    Ax(i,i)=1+2*alpha;
    Ax(i,i-1)=-alpha;
    Ax(i-1,i)=-alpha;
end
  % applying linear boundary condition
    Ax(1,1)=1.0; Ax(Nx,Nx)=1.0;
    Ax(1,2)=0.0; Ax(Nx,Nx-1)=0.0;

Ay=Ax;

for k = 1:2*Nt             %OSM
  %%% 1st step (x-direction) %%%
   if (mod(k,2)==1)
     for j = 2:Ny+1
        for i = 2:Nx+1
          bx(i-1) = u(i,j);
        end
      % Solve Ax*U=bx
        u(2:Nx+1,j) = thomas(Ax,bx);
     end
    % linear boundary condition
    u(1,:) = 2.0*u(2,:) - u(3,:); u(Nx+2,:) = 2.0*u(Nx+1,:) - u(Nx,:);
    u(:,1) = 2.0*u(:,2) - u(:,3); u(:,Ny+2) = 2.0*u(:,Ny+1) - u(:,Ny);
  end
  %%% 2nd step (y-direction) %%%
  if (mod(k,2)==0)
     for i = 2:Nx+1
        for j = 2:Ny+1
          by(j-1) = u(i,j);
        end
      % Solve Ay*U=by
        u(i,2:Ny+1) = thomas(Ay,by);
     end
  % linear boundary condition
  u(1,:) = 2.0*u(2,:) - u(3,:); u(Nx+2,:) = 2.0*u(Nx+1,:) - u(Nx,:);
  u(:,1) = 2.0*u(:,2) - u(:,3); u(:,Ny+2) = 2.0*u(:,Ny+1) - u(:,Ny);
 end
end
```

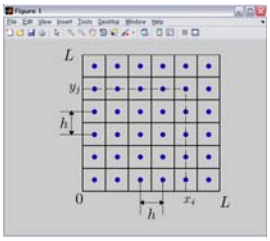
OS를 이용한 2D Black-Scholes equation

2차 Black-Scholes 방정식은 다음과 같다

$$\frac{\partial u}{\partial \tau} = \frac{1}{2}(\sigma_1 x)^2 \frac{\partial^2 u}{\partial x^2} + \frac{1}{2}(\sigma_2 y)^2 \frac{\partial^2 u}{\partial y^2} + \rho\sigma_1\sigma_2 xy \frac{\partial^2 u}{\partial x \partial y} + rx\frac{\partial u}{\partial x} + ry\frac{\partial u}{\partial y} - ru$$

where $\tau = T - t$

Black-Scholes 방정식의 공간미분에 대하여는 **중앙 근사법(central difference)** , **upwind scheme**을, 시간미분에 **후방근사(implicit difference)**을 이용하여 cell center grid.



이제 위의 방정식에 implicit operator splitting scheme을 적용하여 정리하면 다음과 같다.

$$\frac{\tilde{u}_{ij} - u_{ij}^n}{\Delta\tau} = L_{BS}^x \tilde{u}_{ij},$$
$$\frac{u_{ij}^{n+1} - \tilde{u}_{ij}}{\Delta\tau} = L_{BS}^y u_{ij}^{n+1},$$

where

$$L_{BS}^x \tilde{u}_{ij} = \sigma_1^2 x_i^2 \frac{\tilde{u}_{i-1,j} - 2\tilde{u}_{ij} + \tilde{u}_{i+1,j}}{h^2} + rx_i \frac{\tilde{u}_{i+1,j} - \tilde{u}_{ij}}{h}$$
$$+ \lambda_1 \rho\sigma_1\sigma_2 x_i y_j \frac{\tilde{u}_{i+1,j+1} - \tilde{u}_{i+1,j} - \tilde{u}_{ij+1} + \tilde{u}_{ij}}{h^2} - \lambda_2 \tilde{u}_{ij},$$
$$L_{BS}^y u_{ij}^{n+1} = \sigma_2^2 y_j^2 \frac{u_{ij-1}^{n+1} - 2u_{ij}^{n+1} + u_{ij+1}^{n+1}}{h^2} + ry_j \frac{u_{ij+1}^{n+1} - u_{ij}^{n+1}}{h}$$
$$+ (1-\lambda_1)\rho\sigma_1\sigma_2 x_i y_j \frac{u_{i+1,j+1}^{n+1} - u_{ij+1}^{n+1} - u_{i+1,j}^{n+1} + u_{ij}^{n+1}}{h^2} - (1-\lambda_2)u_{ij}^{n+1}.$$

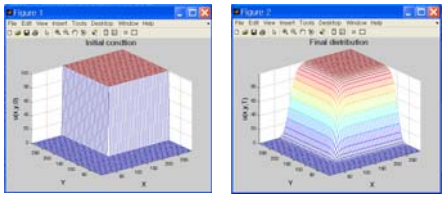
Cash or Nothing option

Cash or Nothing 옵션의 payoff

$$u(x,y,0) = \begin{cases} K & \text{if } x \ge X \text{ and } y \ge Y, \\ 0 & \text{otherwise} \end{cases}$$ ← 초기조건

경계조건 : $u_x(0,y,\tau) = u_{xx}(L,y,\tau) = u_{yy}(x,0,\tau) = u_{yy}(x,L,\tau) = 0$

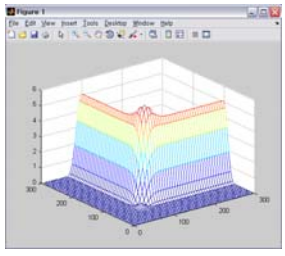
Cash or Nothing Option의 가격계산에 OS방법을 적용하여 보자



초기조건 계산결과

Parameter : $\sigma_1 = 0.5, \sigma_2 = 0.5, \rho = 0.5,$
$r = 0.03, T = 0.1,$
$X = 100, Y = 100, K = 100, L = 300,$
$Nx = 50, Ny = 50, \lambda_1 = 0.5, \lambda_2 = 0.5.$

OS방법을 이용하여 얻은 결과와 해석적 해의 차는 다음과 같이 나타난다.



위의 그래프에서 알 수 있듯이 Cash or Nothing Option의 가격함수의 특성상 Strike Price근처에서 jump현상이 발생하며 비교적 큰 오차가 발생.

Reference

• Space-time adaptive finite difference method for European multi-asset options[computers&mathematics with Appl.53(2007)]Per Lostedt,Jonas Persson,Lina von Sydow,Johan Tysk.
• Pricing European multi-asset options using a space-time adaptive FD-method[comput.Visual Sci(2007)10;173-183] Jonas Persson,Lina von Sydow.

Adaptive OS를 이용한 2D Black-Scholes equation

기울기의 변화가 큰 Strike Price근처에서 오차가 크게 발생하며 나머지 구간에서는 오차가 비교적 작으므로 mesh의 size를 달리하여 OS방법을 적용해보자.

Cash or Nothing Option

초기조건 :

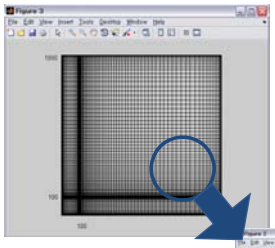
$$u(x,y,0) = \begin{cases} K & \text{if } x \ge X \text{ and } y \ge Y, \\ 0 & \text{otherwise} \end{cases}.$$

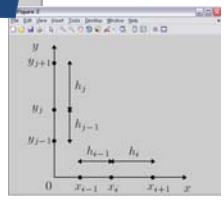
where
$x, y \in [0,1000], \ K = 100.$

경계조건 : Linear Boundary

$$u_{xx}(0,y,\tau) = u_{xx}(1000,y,\tau) = u_{yy}(x,0,\tau) = u_{yy}(x,1000,\tau) = 0$$

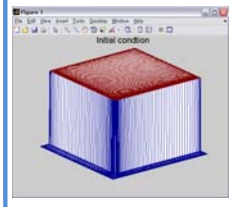
Adaptive Mesh

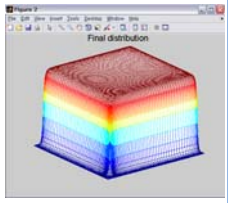




다음과 같이
기울기 변화가 큰 구간에서는
mesh size를 작게,
오차가 비교적 작게 발생하는 구간에서는
mesh size를 크게 하여
계산상의 효율성이 높아지도록 구성.
Mesh size의 jump현상을 피하기 위해
polynomial 함수를 이용.

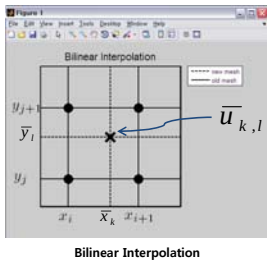
초기조건을 이용하여 adaptive mesh를 구성하여 계산하여 보자.





초기조건 계산결과

정확성을 높이기 위해 각 time step마다 adaptive mesh를 재구성 후, Bilinear interpolation을 이용하여 이전의 계산된 값을 새로 구성된 mesh에 적용.



Bilinear Interpolation

$$\bar{u}_{k,l} = \frac{(x_{i+1} - \bar{x}_k)(y_{j+1} - \bar{y}_l)}{(x_{i+1} - x_i)(y_{j+1} - y_j)} u_{i,j} + \frac{(\bar{x}_k - x_i)(y_{j+1} - \bar{y}_l)}{(x_{i+1} - x_i)(y_{j+1} - y_j)} u_{i+1,j}$$
$$+ \frac{(x_{i+1} - \bar{x}_k)(\bar{y}_l - y_j)}{(x_{i+1} - x_i)(y_{j+1} - y_j)} u_{i,j+1} \frac{(\bar{x}_k - x_i)(\bar{y}_l - y_j)}{(x_{i+1} - x_i)(y_{j+1} - y_j)} u_{i+1,j+1}.$$

Future Research

대부분의 옵션은 단조함수 형태의 Payoff를 가지므로 이에 우수한 Bi-cubic interpolation을 이용하여 각 step에서 adaptive mesh를 구성할 때 적용.
Time step에 대해서도 변화가 큰 부분에서는 작은 time step을 작게 잡아 정확성 제고.