# 제 1 장

# 유한 차분법 (Finite Difference Method)

## 제 1 절 개요

유한차분법은 미분방정식 (differential equation)을 차분방정식 (difference equation)으로 이산화시켜서 수치적인 해를 구하는 방법이다. 이 장에서는 유한차분법을 사용하여 열방정식 (heat equation)과 블랙 숄즈 편미분방정식의 근사해를 구할 것이다. 먼저 유한차분법의 기본 원리를 살펴보자. Taylor의 정리 [1] 를 이용하면 함수 $u(x+h,t)$는 다음과 같이 $(x,t)$ 에서의 $u$ 함수값과 미분값들의 무한 급수로 나타낼 수 있다.

$$u(x+h,t) = u(x,t) + u_x(x,t)h + \frac{u_{xx}(x,t)}{2}h^2 + \frac{u_{xxx}(x,t)}{3!}h^3 + \cdots \tag{1.1}$$

$u_x(x,t)$에 대해서 정리하면,

$$u_x(x,t) = \frac{u(x+h,t) - u(x,t)}{h} + \mathcal{O}(h). \tag{1.2}$$

이것이 전방 차분법 (forward difference method)이다. 마찬가지로, 변수 $t$에 관하여 전방차분을 하면

$$u_t(x,t) = \frac{u(x,t+k) - u(x,t)}{k} + \mathcal{O}(k) \tag{1.3}$$

를 얻는다. 후방차분법 (backward difference method)은

$$u(x-h,t) = u(x,t) - u_x(x,t)h + \frac{u_{xx}(x,t)}{2}h^2 - \frac{u_{xxx}(x,t)}{3!}h^3 + \cdots \tag{1.4}$$

[1] Taylor 정리:함수 $f(x)$가 $x = x_0$에서 $n$번 미분가능하다고 하자.

$$p_n(x) = f(x_0) + f'(x_0)(x-x_0) + \frac{f''(x_0)}{2!}(x-x_0)^2 + \ldots + \frac{f^{(n)}(x_0)}{n!}(x-x_0)^n$$

을 $x = x_0$에서 $f(x)$의 $n$번째 Taylor 다항식이라한다.

식 (1.4)를 $u_x(x,t)$에 대해서 정리하면,

$$u_x(x,t) = \frac{u(x,t) - u(x-h,t)}{h} + \mathcal{O}(h).^2 \tag{1.5}$$

식 (1.1)에서 식(1.4)을 뺀 다음 $u_x(x,t)$에 대해서 정리하면, 다음의 중앙차분방정식 (central difference equation)을 얻는다.

$$u_x(x,t) = \frac{u(x+h,t) - u(x-h,t)}{2h} + \mathcal{O}(h^2). \tag{1.6}$$

식(1.1)에서 식(1.4)을 더한 다음 $u_{xx}(x,t)$에 대해서 정리하면,

$$u_{xx}(x,t) = \frac{u(x+h,t) - 2u(x,t) + u(x-h,t)}{h^2} + \mathcal{O}(h^2). \tag{1.7}$$

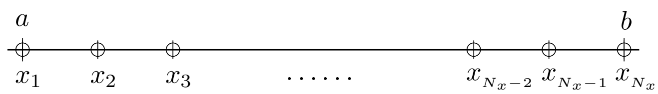



그림 1.1: 유한차분근사를 위한 격자(grid)

그림 1.1와 같이 구간 $[a,b]$를 균등하게 $N_x-1$등분하고 마디점을 $x_i$, 두 점 사이의 간격을 $h$라 하자. $k$만큼 떨어진 동등한 시간 구간의 결절점들로 $t$축을 나누자. 그러면 $h=(b-a)/(N_x-1)$, $x_i = a+(i-1)h$이고 $a = x_1 < x_2 < \cdots < x_{N_x-1} < x_{N_x} = b$와 같이 된다. 여기서 $u_i^n = u(a+(i-1)h, (n-1)k)$ 라고 쓰기로 한다.

## 제 2 절 열 방정식에 대한 유한 차분법

유한차분법을 이용하여 열방정식을 풀어보기로 하자. 열방정식은 식(1.8)과 같은 편미분 방정식이다.

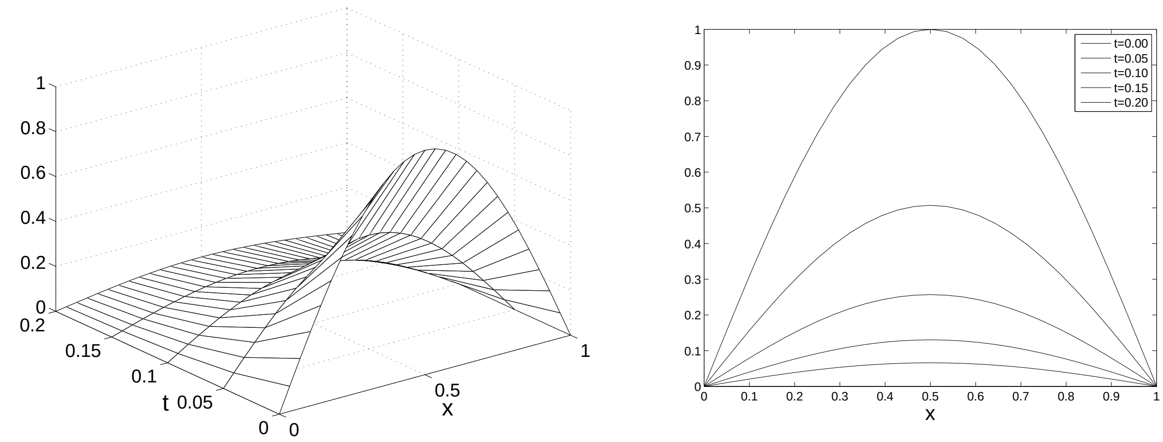
$$u_t(x,t) = u_{xx}(x,t), \qquad 0 < x < 1, \qquad t > 0. \tag{1.8}$$

이 때 경계조건은 $u(0,t) = u(1,t) = 0 \;\; (t > 0)$이고 초기조건은 $u(x,0) = \sin(\pi x) \;\; (0 \le x \le 1)$을 만족한다. 해석해는 $u(x,t) = \sin(\pi x)\mathrm{e}^{-\pi^2 t}$이며 그림1.2처럼 그래프로 나타낼 수 있다. 이 방정식에 대한 근사해를 유한차분법을 이용하여 구해보자.

### 2.1 명시적 (Explicit) 유한 차분법

먼저 정수 $N_x > 0$을 선택하고 $h = 1/(N_x - 1)$이라 정의하면, 식(1.3)와 (1.7)을 이용하여, 열방정식(1.8)에 대해 다음과 같이 유한차분법을 적용할 수 있게 된다. 시간에 대해서 $u_t$을 유한 전방차분 그리고 공간에 대해서 $u_{xx}$에 대한 중앙을 이용하여 열방정식을 다음과 같이 이산화시켜서 나타낼 수 있다.

$$\frac{u_i^{n+1} - u_i^n}{k} + O(k) = \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{h^2} + O(h^2) \tag{1.9}$$




(a) (b)

그림 1.2: 열방정식의 해석해

$$\text{for} \quad i = 2, \ldots, N_x - 1 \quad \text{and} \quad n = 1, 2, \ldots, N_t.$$

$O(k)$와 $O(h^2)$를 무시하면, 식(1.9)를 차분방정식

$$u_i^{n+1} = u_i^n + \alpha(u_{i+1}^n - 2u_i^n + u_{i-1}^n), \ \alpha = \frac{k}{h^2} \tag{1.10}$$

으로 정리할 수 있다. 초기치는 $u_i^1 = \sin(\pi x_i)$ for $i = 1, 2, \ldots, N_x$이다. $u_i^n$을 알고 있다면, 명시적으로 $u_i^{n+1}$을 계산할 수 있다. 이것이 이 방법을 명시적이라 부르는 이유이다. 다음은 $\alpha = 0.45$, $N_x = 30$일 때, 시간에 따른 열방정식의 해를 나타낸 MATLAB 코드이다.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% heatex.m %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clf; clear; clc;
alpha=0.45;
Nx=30;
x=linspace(0,1,Nx);
h=x(2)-x(1);
k=alpha*h^2;
T=0.125;
Nt=round(T/k);
u(1:Nx,1:Nt+1)=0;
u(:,1)=sin(pi*x);

exu=u;

for n=1:Nt
    for i=2:Nx-1
        u(i,n+1) = u(i,n)+alpha*(u(i-1,n)-2*u(i,n)+u(i+1,n));
      exu(i,n+1) = sin(pi*x(i))*exp(-pi^2*(k*n));
    end
end

plot(x,u(:,1),'k*',x,u(:,50),'kd',x,u(:,100),'ks',x,u(:,Nt+1),'ko');
hold
plot(x,exu(:,1),'k',x,exu(:,50),'k',x,exu(:,100),'k',x,exu(:,Nt+1),'k')
legend('initial','n=50','n=100','n=192','exact solution')

xlabel('x','FontSize',20); ylabel('u(x,t)','FontSize',20)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

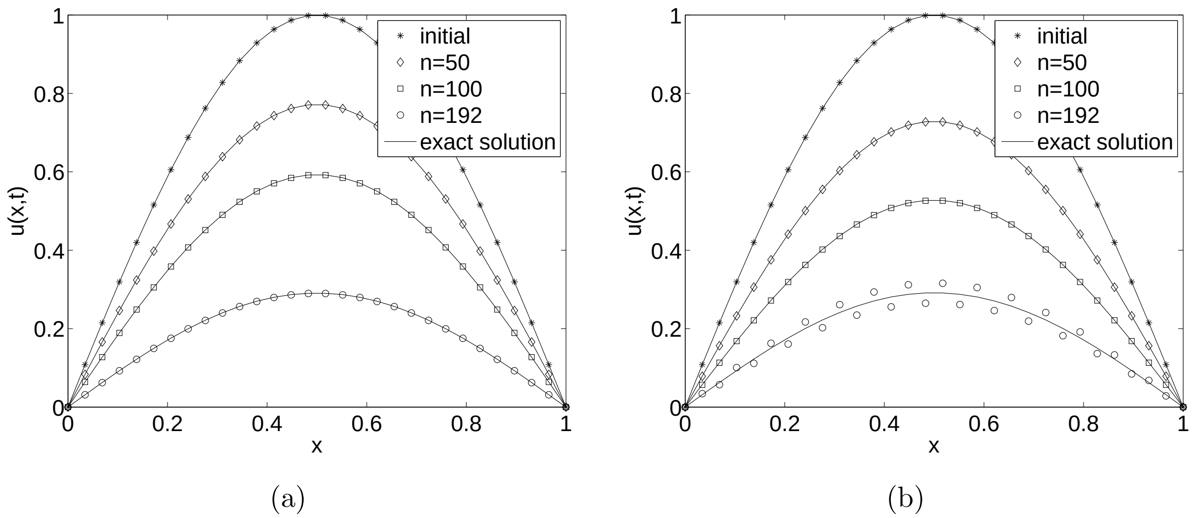



그림 1.3: (a) $\alpha = 0.45$인 안정한 상태 (b) $\alpha = 0.55$인 불안정한 상태

### 2.1.1 명시적방법의 안정성 문제 - 폰 노이만 (von Neumann) 방법

폰노이만 방법은 초기조건을 유한개의 푸리에 급수 (finite Fourier series)로 나타낸 다음, 함수의 성장을 고려하는 것이다. 푸리에 급수는 사인과 코사인의 조합으로 표현할수도 있지만 복소지수함수로 나타내면 계산을 간단하게 할 수 있다. 시간이 지남에 따라서

$$u_k^n = e^{i\beta kh}\xi^n \tag{1.11}$$

이 어떻게 성장하는가 보자. 식 (1.11)을 방정식 (1.10)에 대입을 하면 다음을 얻는다.

$$\begin{aligned} e^{i\beta kh}\xi^{n+1} &= \alpha e^{i\beta(k-1)h}\xi^n + (1-2\alpha)e^{i\beta kh}\xi^n + \alpha e^{i\beta(k+1)h}\xi^n, \\ \xi &= \alpha e^{-i\beta h} + (1-2\alpha) + \alpha e^{i\beta h} = 1 - 4\alpha\sin^2\frac{\beta h}{2}. \end{aligned}$$

양유한차분해가 von Neumann관점에서 안정적이기 위한 필요충분조건은 $|\xi| \leq 1$이다. 그러므로 다음의 부등식이 성립한다.

$$0 \leq \alpha\sin^2\frac{\beta h}{2} \leq \frac{1}{2}. \tag{1.12}$$

따라서, 양유한차분해가 안정적이기 위한 필요충분조건은 다음과 같다.

$$0 < \alpha \leq \frac{1}{2}. \tag{1.13}$$

그림 1.3(a)에서 $\alpha = 0.45$일때 계산된 값은 정확한 해에 가까워지지만, 그림 1.3(b)에서 보듯이 $\alpha = 0.55$일때 계산된 값은 정확한 해로 수렴하지 않는다. 즉 $\alpha$의 값이 커짐에 따라 구한 해가 불안정할 수 있다는 점이 명시적 유한차분법의 단점이다.

## 2.2 함축적 (Implicit) 유한 차분법

명시적 유한차분법의 안정조건인 $0 < \alpha \leq \frac{1}{2}$의 제약을 피하기 위해 함축적 유한 차분법을 이용한다. 함축적 방법은 시간간격을 작게 취하지 않고도 많은 수의 격자점들을 이용할 수 있다. 다만 함축적 방법은 유한차분 연립방정식의 해를 구해야 한다. 보통 함축적 유한차분법이라고 알려진 완전 함축적 유한차분법은 $u_t$에 대한 후방 유한차분근사와 $u_{xx}$에 대한 중앙차분근사를 이용한다. 따라서 다음과 같은 함축적 유한차분 방정식을 이끌어 낼 수 있다.

$$\frac{u_i^{n+1} - u_i^n}{k} = \frac{u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i+1}^{n+1}}{h^2}.$$

다시 정리하면 다음의 함축적 유한차분방정식

$$-\alpha u_{i-1}^{n+1} + (1+2\alpha)u_i^{n+1} - \alpha u_{i+1}^{n+1} = u_i^n,\ \alpha = \frac{k}{h^2} \qquad \text{for each } i = 2, \cdots, N_x - 1 \tag{1.14}$$

을 얻게 된다. (1.14)는 다음과 같은 선형 시스템 (linear system)으로 나타낼 수 있다.

$$\begin{pmatrix} 1+2\alpha & -\alpha & 0 & \dots & 0 \\ -\alpha & 1+2\alpha & -\alpha & & 0 \\ 0 & -\alpha & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & -\alpha \\ 0 & 0 & & -\alpha & 1+2\alpha \end{pmatrix} \begin{pmatrix} u_2^{n+1} \\ u_3^{n+1} \\ \vdots \\ \vdots \\ u_{N_x-1}^{n+1} \end{pmatrix} = \begin{pmatrix} \alpha u_1^{n+1} + u_2^n \\ u_3^n \\ \vdots \\ \vdots \\ u_{N_x-1}^n + \alpha u_{N_x}^{n+1} \end{pmatrix} = \begin{pmatrix} b_2^n \\ b_3^n \\ \vdots \\ \vdots \\ b_{N_x-1}^n \end{pmatrix}. \tag{1.15}$$

(1.15)은 좀 더 압축된 형태인

$$\mathbf{A}\boldsymbol{u}^{n+1} = \boldsymbol{b}^n \tag{1.16}$$

으로 나타낼 수 있다. 여기서 $\boldsymbol{u}^{n+1}$와 $\boldsymbol{b}^n$는 $(N_x - 2)-$차원의 벡터들인

$$\boldsymbol{u}^{n+1} = (u_2^{n+1}, \cdots, u_{N_x-1}^{n+1})^T, \quad \boldsymbol{b}^n = \boldsymbol{u}^n + \alpha(u_1^n, 0, \cdots, 0, u_{N_x}^{n+1})^T$$

을 뜻하며, $\mathbf{A}$은 (1.15)에서 주어진 $(N_x - 2)-$정방대칭행렬을 뜻한다. $\alpha \geq 0$에 대하여 $\mathbf{A}$은 가역 (invertible)이므로

$$\boldsymbol{u}^{n+1} = \mathbf{A}^{-1}\boldsymbol{b}^n \tag{1.17}$$

이다. 그러므로 $\boldsymbol{u}^n$과 경계조건에 의해 구해지는 $\boldsymbol{b}^n$에 의해 $\boldsymbol{u}^{n+1}$를 찾을 수 있다. 초기조건은 $\boldsymbol{u}^1$에 의해 결정되므로, 각각의 $\boldsymbol{u}^{n+1}$을 순차적으로 구할 수 있다.

### 2.2.1 토마스 알고리즘 (Thomas Algorithm)

영이 아닌 원소를 가지는 다음 행렬을 살펴보자.

$$\begin{pmatrix} d_1 & c_1 & & & & & & \\ a_1 & d_2 & c_2 & & & & & \\ & a_2 & d_3 & c_3 & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & a_{i-1} & d_i & c_i & & \\ & & & & \ddots & \ddots & \ddots & \\ & & & & & a_{N_x-2} & d_{N_x-1} & c_{N_x-1} \\ & & & & & & a_{N_x-1} & d_{N_x} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_i \\ \vdots \\ x_{N_x-1} \\ x_{N_x} \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_i \\ \vdots \\ b_{N_x-1} \\ b_{N_x} \end{pmatrix}, \qquad (1.18)$$

여기서 표시되지 않은 원소들은 모두 0이다. 삼중대각(Tridiagonal) 행렬은 $|i-j| \geq 2$일 때, $a_{ij} = 0$가 되는 특징을 가지고 있다. 이제 삼중대각 행렬의 해를 구하는 알고지즘을 구해보자.

1행에 $a_1/d_1$을 곱한 값을 2행에서 뺀다. 그러면 $a_1$의 자리에는 0이 위치하게 되고 $d_2$와 $b_2$의 값도 다음과 같이 변하게 된다.

$$d_2 = d_2 - \frac{a_1}{d_1}c_1, \qquad b_2 = b_2 - \frac{a_1}{d_1}b_1$$

하지만 $c_2$는 변하지 않는다.

$$\begin{pmatrix} d_1 & c_1 & & & & & & \\ 0 & d_2 - \dfrac{a_1 c_1}{d_1} & c_2 & & & & & \\ & a_2 & d_3 & c_3 & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & a_{i-1} & d_i & c_i & & \\ & & & & \ddots & \ddots & \ddots & \\ & & & & & a_{N_x-2} & d_{N_x-1} & c_{N_x-1} \\ & & & & & & a_{N_x-1} & d_{N_x} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_i \\ \vdots \\ x_{N_x-1} \\ x_{N_x} \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 - \dfrac{a_1 b_1}{d_1} \\ b_3 \\ \vdots \\ b_i \\ \vdots \\ b_{N_x-1} \\ b_{N_x} \end{pmatrix}.$$

이 과정을 반복하여 적용하면 각각의 단계에서 $d_i$와 $b_i$들은 다음과 같이 바뀌게 된다:

$$d_i = d_i - \frac{a_{i-1}}{d_{i-1}} c_{i-1}, \qquad b_i = b_i - \frac{a_{i-1}}{d_{i-1}} b_{i-1} \qquad (2 \le i \le N_x).$$

전방소거의 결과로, 식(1.18)은 다음의 형태를 갖게 된다.

$$\begin{pmatrix} d_1 & c_1 & & & & & & \\ & d_2 & c_2 & & & & & \\ & & d_3 & c_3 & & & & \\ & & & \ddots & \ddots & & & \\ & & & & d_i & c_i & & \\ & & & & & \ddots & \ddots & \\ & & & & & & d_{N_x-1} & c_{N_x-1} \\ & & & & & & & d_{N_x} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_i \\ \vdots \\ x_{N_x-1} \\ x_{N_x} \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_i \\ \vdots \\ b_{N_x-1} \\ b_{N_x} \end{pmatrix}. \tag{1.19}$$

여기서 $b_i$와 $d_i$들은 처음값과는 다른 값을 가지고 있지만 $c_i$의 값들은 처음값과 동일하다. 이제 후방대입을 통해 $x_{N_x},\ x_{N_x-1},\ \cdots,\ x_1$을 차례로 구할 수 있다:

$$\begin{aligned} x_{N_x} &= \frac{b_{N_x}}{d_{N_x}}, \\ x_i &= \frac{1}{d_i}(b_i - c_i x_{i+1}), \qquad i = N_x - 1,\ N_x - 2,\ \cdots,\ 1. \end{aligned}$$

다음은 열방정식의 수치해를 토마스 알고리즘을 이용하여 구하는 MATLAB 코드이다.

위의 코드 2.2.1를 실행하면 다음의 결과를 얻을 수 있다.

### 2.2.2 함축적 방법의 안정성 문제 - 폰 노이만 방법

명시적 방법의 안정성 문제와 같이 시간이 지남에 따라서

$$u_k^n = e^{i\beta kh}\xi^n \tag{1.20}$$

이 어떻게 성장하는가 보자. 식 (1.20)을 방정식 (1.14)에 대입을 하면 다음을 얻는다.

$$\begin{aligned} -\alpha e^{i\beta(k-1)h}\xi^{n+1} + (1+2\alpha)e^{i\beta kh}\xi^{n+1} - \alpha e^{i\beta(k+1)h}\xi^{n+1} &= e^{i\beta kh}\xi^n, \\ -\alpha e^{-i\beta h}\xi + (1+2\alpha)\xi - \alpha e^{i\beta h}\xi &= 1, \\ (2\alpha(1-\cos(\beta h))+1)\xi &= 1. \end{aligned}$$

따라서,

$$\xi = \frac{1}{4\alpha\sin^2(\beta h/2)+1}. \tag{1.21}$$

$\xi$는 모든 양수 $\alpha$와 모든 $\beta$에 대해서 $\frac{1}{4\alpha+1} \le \xi \le 1$을 만족한다. 따라서 식(1.14)은 무조건 안정적이다. 이는 그림 1.5로 확인할 수 있다.

## 2.3 크랭크 니콜슨 (Crank-Nicolson) 방법

지금까지 언급한 명시적 또는 함축적 방법을 한번에 고려한 방법이 크랭크 니콜슨 방법이다. 크랭크 니콜슨 방법은 시간 격자 $n$과 $n+1$의 중간에 있는 미분 근사값 $u_i^{n+1/2}$을 이용한다. 시점 $n+1/2$에서 , 시간에 대한 1계 편미분을 구하면 다음과 같다.

$$u_t(x_i, t^{n+1/2}) = \frac{u_i^{n+1} - u_i^n}{k} + O(k^2) \tag{1.22}$$

또한 시점 $n+1/2$에서, 공간변수 $x$에 대한 2계 편미분은 시점 $n$와 $n+1$에서 2계 편미분 근사값을 평균해서 구한다.

$$\begin{aligned} u_{xx}(x_i, t^{n+1/2}) &= \frac{1}{2}\left(u_{xx}(x_i,t^n) + u_{xx}(x_i,t^{n+1})\right) + O(h^2) \\ &= \frac{1}{2}\left(\frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{h^2} + \frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}}{h^2}\right) + O(h^2). \end{aligned} \tag{1.23}$$

두 근사식 (1.22)와 (1.23)의 절단오차는 각각 $\mathcal{O}(k^2)$과 $\mathcal{O}(h^2)$으로 근사식의 정확도가 높기 때문에 많은 계산을 하지 않아도 수치분석에서 만족스러운 해를 얻을 수 있다. 식(1.22)와 (1.23)의 우변을 같게 하면 다음과 같은 크랭크 니콜슨 식을 얻을 수 있다.

$$-\alpha u_{i-1}^{n+1} + 2(1+\alpha)u_i^{n+1} - \alpha u_{i+1}^{n+1} = \alpha u_{i-1}^n + 2(1-\alpha)u_i^n + \alpha u_{i+1}^n, \tag{1.24}$$

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% heatim.m %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear; clc; clf;
Nx=12; x=linspace(0,1,Nx); h=x(2)-x(1);
T=0.1; alpha=2;
k = alpha*(h^2); Nt=round(T/k);
u(:,1)=sin(pi*x);

for i=1:Nx-2
    dd(i)= 1 + 2*alpha; c(i)= - alpha; a(i)= - alpha;
end

for n=1:Nt
    d=dd;
    for i=1:Nx-2
           b(i)=u(i+1,n);
    end
            for i=2:Nx-2
                xmult= a(i-1)/d(i-1);
                d(i) = d(i) - xmult*c(i-1);
                b(i) = b(i) - xmult*b(i-1);
            end
            u(Nx-1,n+1) = b(Nx-2)/d(Nx-2);
            for i = Nx-3:-1:1
                u(i+1,n+1) = (b(i) - c(i)*u(i+2,n+1))/d(i);
            end
end
plot(x,u,'ko-')
xlabel('x','FontSize',20); ylabel('u(x,t)','FontSize',20);
title('Heat equation - Implicit(\alpha = 2)','FontSize',20)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

그림 1.4: 토마스 알고리즘을 이용한 함축적 열방정식 MATLAB 코드.

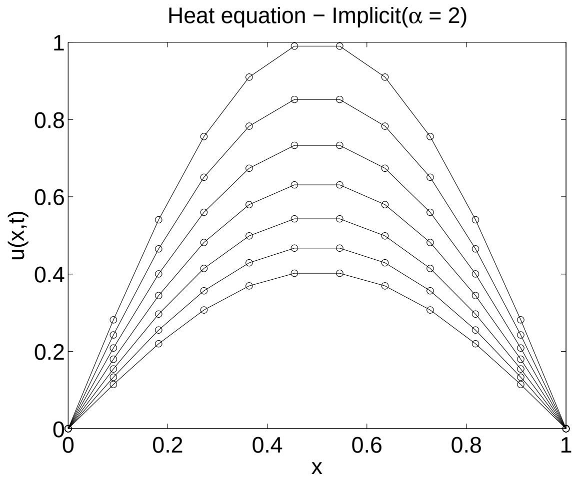



그림 1.5: 함축적 열방정식 $\alpha = 2$인 안정한 상태

여기서 $\alpha = \frac{k}{h^2}$이다. 식 (1.24)를 행렬 형태로 표현하면 다음과 같다.

$$\begin{pmatrix} 2(1+\alpha) & -\alpha & & & & \\ -\alpha & 2(1+\alpha) & -\alpha & & & \\ & -\alpha & 2(1+\alpha) & -\alpha & & \\ & \ddots & \ddots & \ddots & & \\ & & & & 2(1+\alpha) & -\alpha \\ & & & & -\alpha & 2(1+\alpha) \end{pmatrix} \begin{pmatrix} u_2^{n+1} \\ u_3^{n+1} \\ u_4^{n+1} \\ \vdots \\ u_{Nx-2}^{n+1} \\ u_{Nx-1}^{n+1} \end{pmatrix}$$

$$= \begin{pmatrix} \alpha u_1^{n+1} + 2(1-\alpha)u_2^n + \alpha u_3^n \\ \alpha u_2^n + 2(1-\alpha)u_3^n + \alpha u_4^n \\ \alpha u_3^n + 2(1-\alpha)u_4^n + \alpha u_5^n \\ \vdots \\ 2(1-\alpha)u_{Nx-2}^n + \alpha u_{Nx-1}^n \\ \alpha u_{Nx-2}^n + 2(1-\alpha)u_{Nx-1}^n + \alpha u_{Nx}^{n+1} \end{pmatrix} = \begin{pmatrix} b_2^n \\ b_3^n \\ b_4^n \\ \vdots \\ b_{Nx-2}^n \\ b_{Nx-1}^n \end{pmatrix}. \tag{1.25}$$

위의 식(1.25)을 간단하게 다음과 같이 표현할 수 있다.

$$\mathbf{A}\mathbf{u}^{n+1} = \mathbf{b}^n.$$

다음은 열방정식을 크랭크 니콜슨 방법에 의해 푼 MATLAB code이다.

위의 코드2.3를 실행하면 다음의 결과를 얻을 수 있다.

### 2.3.1 크랭크 니콜슨 방법의 안정성 문제 - 폰 노이만 방법

## 2.4 수렴성 (convergence) 테스트

국소절단오차 (local truncation error)는 연속해가 노드점에서 수치적 방법을 만족하지 못하는 차이를 측정한 것이다. 수치 기법의 국소절단오차는 연속적인 문제의 정확한 해를 이산적인 수치기법에 대입함으로써 발생한다. $u(x_i, t^n)$는 열방정식의 정확한 해를 나타낸다. 다음은 정확한 해를 수치기법에 대입함으로써 명시적 유한차분법의 국소절단오차를 찾는 과정이다. 노드 $(x_i, t^n)$에서 국소절단오차는 다음과 같이 구한다.

$$T(x_i, t^n) = \frac{u(x_i, t^{n+1}) - u(x_i, t^n)}{k} - \frac{u(x_{i+1}, t^n) - 2u(x_i, t^n) + u(x_{i-1}, t^n)}{h^2}.$$

이제 노드$(x_i, t^n)$에서 테일러전개를 하면 각각의 항을 다음과 같이 나타낼 수 있다.

$$\begin{aligned} T(x_i, t^n) &= u_t(x_i, t^n) + \frac{k}{2} u_{tt}(x_i, t^n) + \mathcal{O}(k^2) \\ &\quad - u_{xx}(x_i, t^n) + \frac{h^2}{12} u_{xxxx}(x_i, t^n) + \mathcal{O}(h^4). \end{aligned}$$

여기서 $u(x_i, t^n)$은 열방정식을 만족하므로 다음이 성립한다.

$$\begin{aligned} T(x_i, t^n) &= \frac{k}{2} u_{tt}(x_i, t^n) + \frac{h^2}{12} u_{xxxx}(x_i, t^n) + \mathcal{O}(k^2) + \mathcal{O}(h^4) \\ &= \mathcal{O}(k) + \mathcal{O}(h^2). \end{aligned} \tag{1.26}$$

수치적 해가 수렴하기 위해 필요한 조건은 수치기법의 국소절단오차가 공간간격과 시간간격을 줄일수록 0에 근사해야 한다는 것이다. 이럴 경우에, 수치기법이 일관적 (consistent)이라고 한다. 정확도의 차수 (order of accuracy)는 절단오차항에서 $h$와 $k$의 승수의 차수로 정의된다. 절단오차항을 $\mathcal{O}(k^l, h^m)$로 가정하면 수치기법이 $l$차 시간 정확 ($l$th order time accurate)하고 $m$차 공간정확 ($m$th order space accurate)하다고 한다. 식(1.31)으로부터 명시적 유한차분법은 1차 시간 정확하고 2차 공간 정확함을 알 수 있다.

### 2.4.1 명시적 유한차분법

열방정식의 명시적 유한 차분법의 수렴성을 알아보기 위해 다음의 테스트를 수행해보자.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% heatCN.m %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear; clc; clf;
Nx=12; x=linspace(0,1,Nx); h=x(2)-x(1);
T=0.1; alpha = 2;
k = alpha*(h^2); Nt=round(T/k);
u(:,1)=sin(pi*x);

for i=1:Nx-2
    dd(i)= 2*(1+alpha); c(i)= - alpha; a(i)= - alpha;
end

for n=1:Nt
    d=dd;
    for i=1:Nx-2
        b(i)=alpha*u(i,n)+2*(1-alpha)*u(i+1,n)+alpha*u(i+2,n);
    end
    for i = 2:Nx-2
        xmult=a(i-1)/d(i-1);
        d(i)=d(i)-xmult*c(i-1);
        b(i)=b(i)-xmult*b(i-1);
    end
    u(Nx-1,n+1) = b(Nx-2)/d(Nx-2);
    for i = Nx-3:-1:1
        u(i+1,n+1) = (b(i) - c(i)*u(i+2,n+1))/d(i);
    end
end
plot(x,u,'ko-');
xlabel('x','FontSize',20); ylabel('u(x,t)','FontSize',20)
title('Heat equation - Crank-Nicolson(\alpha = 2)','FontSize',20)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

그림 1.6: 열방정식의 크랭크 니콜슨 MATLAB 코드.

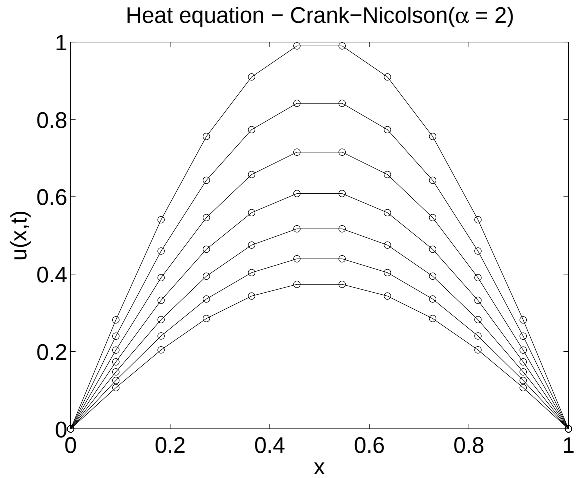



그림 1.7: 크랭크 니콜슨을 이용한 열방정식 $\alpha = 2$인 안정한 상태

초기조건은 $u(x,0) = \sin(x)$, $T = 0.1$, $\alpha = 0.1$, $h = 1/N$, $\Delta t = \alpha h^2$. MATLAB 코드2.4.1을 실행하면 다음의 결과를 얻을 수 있다.

```
>> heatex_convergence_test
---------------------------------------------------
    h          dt         max_error       order
---------------------------------------------------
 0.10000    0.001000      0.001220
 0.05000    0.000250      0.000303      2.008865
 0.02500    0.000063      0.000076      2.002218
 0.01250    0.000016      0.000019      2.000555
 0.00625    0.000004      0.000005      2.000139
---------------------------------------------------
```

### 2.4.2 함축적 유한차분법

열방정식의 함축적 유한 차분법의 수렴성을 알아보기 위해 다음의 테스트를 수행해보자.

```
%%%%%%%%%%%%%%%%%%%%%%%%% heatex_convergence_test.m %%%%%%%%%%%%%%%%%%%%%%%%%
clear; clc;
T=0.1; alpha=0.1;

for iter=1:5
   N=10*2^(iter-1)+1; x=linspace(0,1,N); h=x(2)-x(1);
   k=alpha*h^2; Nt=round(T/k);
   u(1:N,1:Nt+1)=0;
   u(:,1)=sin(pi*x); exact=u(:,1)*exp(-pi^2*T);

   for n=1:Nt
        for i=2:N-1
            u(i,n+1)=alpha*u(i-1,n)+(1-2*alpha)*u(i,n)+alpha*u(i+1,n);
        end
   end
   hh(iter)=h; tt(iter)=k; err(iter) = max(abs(u(:,Nt+1) - exact));
end

Order=[log(err(1)/err(2))/log(2) log(err(2)/err(3))/log(2) ...
    log(err(3)/err(4))/log(2) log(err(4)/err(5))/log(2)]';
fprintf('------------------------------------------------\n')
fprintf('    h          dt        max error       order    \n')
fprintf('------------------------------------------------\n')
fprintf('%8.5f   %8.6f     %8.6f      \n',hh(1), tt(1) ,err(1))
for iter = 2:5
fprintf('%8.5f   %8.6f     %8.6f      %8.6f \n',hh(iter),tt(iter), ...
    err(iter),Order(iter-1))
end
fprintf('------------------------------------------------\n')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

그림 1.8: 열방정식의 명시적 유한차분법의 정확도의 차수를 확인하는 MATLAB 코드.

초기조건은 $u(x, 0) = \sin(x)$, $T = 0.1$, $\alpha = 0.1$, $h = 1/N$, $\Delta t = \alpha h^2$. MATLAB 코드2.4.2을 실행하면 다음의 결과를 얻을 수 있다.

```
>> heatim_convergence_test
------------------------------------------------
    h          dt        max error       order
------------------------------------------------
 0.10000    0.001000     0.004820
 0.05000    0.000250     0.001209      1.995470
 0.02500    0.000063     0.000302      1.998865
 0.01250    0.000016     0.000076      1.999716
 0.00625    0.000004     0.000019      1.999929
------------------------------------------------
```

### 2.4.3 크랭크 니콜슨 유한차분법

열방정식의 크랭크 니콜슨 유한 차분법의 수렴성을 알아보기 위해 다음의 테스트를 수행해보자.

```
%%%%%%%%%%%%%%%%%%%%%%%% heatim_convergence_test.m %%%%%%%%%%%%%%%%%%%%%%%%
clear; clc; T=0.1; alpha=0.1;
for iter=1:5
   N=10*2^(iter-1)+1; x=linspace(0,1,N); h=x(2)-x(1); k=alpha*h^2;
   Nt=round(T/k); u(1:N,1:Nt+1)=0; u(:,1)=sin(pi*x);
   exact=u(:,1)*exp(-pi^2*T);
   for i=1:N-2
       dd(i)= 1 + 2*alpha; c(i)= - alpha; a(i)= - alpha;
   end
   for n=1:Nt
       d=dd;
       for i=1:N-2
           b(i)=u(i+1,n);
       end
       for i=2:N-2
           xmult= a(i-1)/d(i-1);
           d(i) = d(i) - xmult*c(i-1);
           b(i) = b(i) - xmult*b(i-1);
       end
       u(N-1,n+1) = b(N-2)/d(N-2);
       for i = N-3:-1:1
           u(i+1,n+1) = (b(i) - c(i)*u(i+2,n+1))/d(i);
       end
   end
   hh(iter)=h; tt(iter)=k; err(iter) = max(abs(u(:,Nt+1) - exact));
end
Order=[log(err(1)/err(2))/log(2) log(err(2)/err(3))/log(2) ...
    log(err(3)/err(4))/log(2) log(err(4)/err(5))/log(2)]';
fprintf('-----------------------------------------------\n')
fprintf('    h           dt        max error       order    \n')
fprintf('-----------------------------------------------\n')
fprintf('%8.5f    %8.6f     %8.6f      \n',hh(1), tt(1) ,err(1))
for iter = 2:5
fprintf('%8.5f    %8.6f     %8.6f      %8.6f \n',hh(iter),tt(iter), ...
    err(iter),Order(iter-1))
end
fprintf('-----------------------------------------------\n')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

그림 1.9: 열방정식의 함축적 유한차분법의 정확도의 차수를 확인하는 MATLAB 코드.

초기조건은 $u(x,0) = \sin(x)$, $T = 0.1$, $\alpha = 0.1$, $h = 1/N$, $\Delta t = \alpha h^2$. MATLAB 코드2.4.3을 실행하면 다음의 결과를 얻을 수 있다.

```
>> heatcn_convergence_test
--------------------------------------------------
    h          dt          max error        order
--------------------------------------------------
 0.10000    0.001000    0.003025
 0.05000    0.000250    0.000756      1.999772
 0.02500    0.000063    0.000189      1.999942
 0.01250    0.000016    0.000047      1.999985
 0.00625    0.000004    0.000012      1.999996
--------------------------------------------------
```

# 제 3 절 Black-Scholes 편미분방정식에 대한 유한 차분법

유러피언 콜 옵션의 값을 구하기 위해서 Black-Scholes 편미분방정식을 유한차분법으로 풀어서 구한다. 편미분방정식은 Dirichlet 경계조건을 갖는 포물선형 편미분방정식이다. 특히 초기조건보다 만기시의 조건이 주어진다. $\tau = T - t$를 잔존기간으로 놓음으로써, 더 자연스러운 시간의 방정식으로 바꿀 수 있다. 그러면 편미분방정식은 다음과 같이 정리된다.

$$u_\tau = \frac{1}{2}\sigma^2 x^2 u_{xx} + rxu_x - ru.$$

이 때, 정의역은 $x \geq 0$ 이고, 시간의 범위는 $0 \leq \tau \leq T$, 초기값은 $u(x,0) = \max(x - E, 0)$ 이고, 경계조건은 $u(0,\tau) = 0$, 값이 큰 $x$에 대해서 $u(x,\tau) \approx x - Ee^{-r\tau}$를 갖는다. $x$를 $0 \leq x \leq L$의 범위로 두고 $h = L/(N_x - 1)$와 $k = T/N_t$의 간격을 갖는 유한 차분 격자를 사용함으로써, 이산해 $u_i^n \approx u((i-1)h,(n-1)k) = u(x_i, t^n)$를 계산할 수 있다. 모든 $1 \leq n \leq N_t$에서 초기 데이터에 의해 지정된 값 $u_i^1 = \max(x_i - E, 0)$ for $1 \leq i \leq N_x$, 그리고 경계조건에 의해 지정된 경계값 $u_1^n = 0$와 $u_{N_x}^n = L - Ee^{-rt^n}$을 갖게 된다.

## 3.1 명시적 방법에 의한 옵션 가격 결정

시간 미분에 대해서 전방 차분, 그리고 공간 미분에 대해서 중앙 차분을 사용함으로써 다음과 같이 명시적 방법을 적용할 수 있다.

$$\frac{u_i^{n+1} - u_i^n}{k} = \frac{1}{2}\sigma^2 x_i^2 \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{h^2} + rx_i \frac{u_{i+1}^n - u_{i-1}^n}{2h} - ru_i^n.$$

```
%%%%%%%%%%%%%%%%%%%%%%%% heatcn_convergence_test.m %%%%%%%%%%%%%%%%%%%%%%%%%%
clear; clc; T=0.1; alpha=0.1;
for iter=1:5
   N=10*2^(iter-1)+1; x=linspace(0,1,N); h=x(2)-x(1); k=alpha*(h^2);
   Nt=round(T/k); u(1:N,1:Nt+1)=0; u(:,1)=sin(pi*x);
   exact=u(:,1)*exp(-pi^2*T);
   for i=1:N-2
       dd(i)= 2*(1+alpha); c(i)= - alpha; a(i)= - alpha;
   end
    for n=1:Nt
        d=dd;
        for i=1:N-2
          b(i)=alpha*u(i,n)+2*(1-alpha)*u(i+1,n)+alpha*u(i+2,n);
        end
        for i = 2:N-2
          xmult=a(i-1)/d(i-1);
          d(i)=d(i)-xmult*c(i-1); b(i)=b(i)-xmult*b(i-1);
        end
        u(N-1,n+1) = b(N-2)/d(N-2);
        for i = N-3:-1:1
            u(i+1,n+1) = (b(i) - c(i)*u(i+2,n+1))/d(i);
        end
    end
   hh(iter)=h; tt(iter)=k; err(iter) = max(abs(u(:,Nt+1) - exact));
end
Order=[log(err(1)/err(2))/log(2) log(err(2)/err(3))/log(2) ...
    log(err(3)/err(4))/log(2) log(err(4)/err(5))/log(2)]';
fprintf('------------------------------------------------\n')
fprintf('    h          dt        max error       order    \n')
fprintf('------------------------------------------------\n')
fprintf('%8.5f    %8.6f     %8.6f      \n',hh(1), tt(1) ,err(1))
for iter = 2:5
fprintf('%8.5f    %8.6f     %8.6f      %8.6f \n',hh(iter),tt(iter), ...
    err(iter),Order(iter-1))
end
fprintf('------------------------------------------------\n')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

그림 1.10: 열방정식의 크랭크 니콜슨 유한차분법의 정확도의 차수를 확인하는 MATLAB 코드.

$u_i^{n+1}$에 대해서 정리하면 다음과 같은 식을 얻는다.

$$\begin{aligned} u_i^{n+1} &= u_i^n + k\left(\frac{1}{2}\sigma^2 x_i^2 \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{h^2} + r x_i \frac{u_{i+1}^n - u_{i-1}^n}{2h} - r u_i^n\right) \\ & \quad \text{for} \;\; 2 \le i \le N_x - 1. \end{aligned}$$

Black-Scholes 방정식을 명시적 방법으로 수치해를 구하는 MATLAB 코드가 그림3.1에 있다.

그림1.12에는 무위험이자율이 $r = 0.03$, 변동성이 $\sigma = 0.5$, 현재시점이 $t = 0$, 만기 시점이 $T = 1$, 그리고 행사가격이 $E = 230$인 유럽형 콜옵션의 가격이 그려져 있다.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% BSex.m %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clf; clear;
E=230; L=800; sigma=0.5; r=0.03;
T=1; Nx=50; Nt=1000; k=T/Nt;
x=linspace(0,L,Nx); h=x(2)-x(1);
u(1:Nx,1:Nt+1)=0;

for i=1:Nx
    if x(i)<= E
        u(i,1)=0;
    else
        u(i,1)=x(i)-E;
    end
end

for n=2:Nt+1
    u(Nx,n)=L-E*exp(-r*k*(n-1));
end

for n=1:Nt
    for i=2:Nx-1
        u(i,n+1)=u(i,n) + k*((1/2)*(sigma^2)*((i-1)*h)^2*...
            ((u(i+1,n)-2*u(i,n)+u(i-1,n))/(h^2)) +...
            r*(i-1)*h*((u(i+1,n)-u(i-1,n))/(2*h))-r*u(i,n));
    end
end

plot(x,u(:,1:200:Nt+1),'ko-')
axis image
axis([0 L 0 600])
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

그림 1.11: 명시적 방법에 의한 유러피언 옵션 가격결정을 위한 MATLAB 코드.

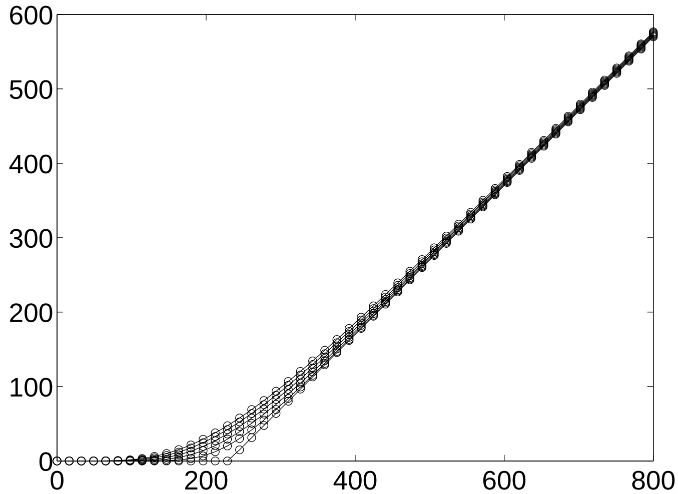



그림 1.12: 명시적 방법에 의한 블렉숄츠 방정식의 수치해

### 3.2 함축적 방법에 의한 옵션 가격 결정

시간에 대한 전방 차분을 이용한 명시적 방법에서 일어날 수 있는 불안정성 문제를 해결하기 위해서 후방차분을 이용하여 함축적 방법을 적용하면 다음의 식을 얻게 된다.

$$\frac{u_i^{n+1}-u_i^n}{k} = \frac{1}{2}\sigma^2 x_i^2 \frac{u_{i+1}^{n+1}-2u_i^{n+1}+u_{i-1}^{n+1}}{h^2} + rx_i \frac{u_{i+1}^{n+1}-u_{i-1}^{n+1}}{2h} - ru_i^{n+1}. \tag{1.27}$$

이러한 함축적 방법은 시간의 크기에 영향을 받지 않을 뿐만 아니라 수치계산의 정확도를 높이는 장점이 있다. 이제 위의 식 (1.27)을 정리하면

$$\alpha_i u_{i-1}^{n+1} + \beta_i u_i^{n+1} + \gamma_i u_{i+1}^{n+1} = \frac{u_i^n}{k}, \tag{1.28}$$

여기서

$$\alpha_i = \frac{rx_i}{2h} - \frac{\sigma^2 x_i^2}{2h^2}, \quad \beta_i = \frac{1}{k} + \frac{\sigma^2 x_i^2}{h^2} + r, \quad \gamma_i = -\frac{rx_i}{2h} - \frac{\sigma^2 x_i}{h^2}$$

이다.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% BSim.m %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clf; clear;
E=230; L=800; sigma=0.5; r=0.03;
T=1; Nx=50; Nt=100; k=T/Nt;
x=linspace(0,L,Nx); h =x(2)-x(1);
u(1:Nx,1:Nt+1)=0;
N=Nx-2;
for i=1:Nx
    if x(i) < E
        u(i,1)= 0;
    else
        u(i,1)= x(i)-E;
    end
end
for i=1:N
   dd(i)=1/k+(sigma*i)^2+r; c(i)=-r*i/2-((sigma*i)^2)/2;
   a(i)=r*(i+1)/2-((sigma*(i+1))^2)/2;
end
for n=1:Nt
    d=dd;
    for i=1:N-1
         b(i)=u(i+1,n)/k;
    end
     u(Nx,n+1)=L - E*exp(-r*k*n); b(N)=u(N+1,n)/k - c(N)*u(Nx,n+1);
    for i = 2:N
        xmult= a(i-1)/d(i-1);
        d(i) = d(i) - xmult*c(i-1); b(i) = b(i) - xmult*b(i-1);
    end
    u(N+1,n+1) = b(N)/d(N);
    for i = N-1:-1:1
        u(i+1,n+1) = (b(i) - c(i)*u(i+2,n+1))/d(i);
    end
end
    plot(x,u(:,1:20:Nt+1),'ko-')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

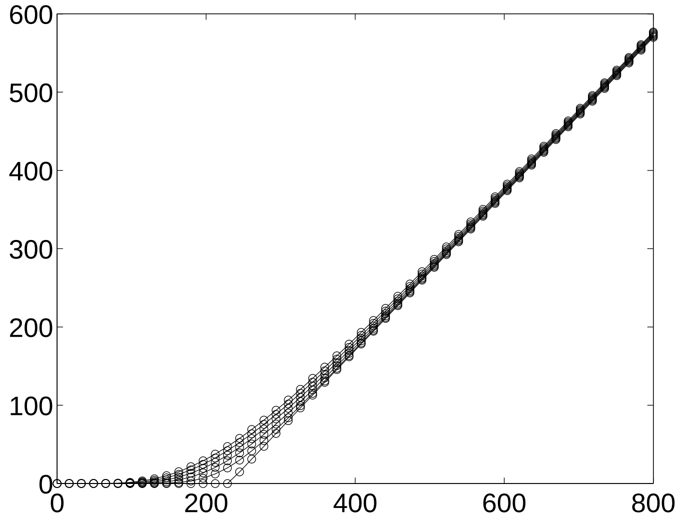



그림 1.13: 함축적 방법에 의한 옵션 가격 결정

## 3.3 크랭크 니콜슨 방법방법에 의한 옵션 가격 결정

크랭크 니콜스방법은 명시적방법과 함축적방법을 조합하여 정확도를 향상시킨 방법이다. 이 아이디어를 Black-Scholes방정식에 적용하면 다음과 같은 방정식을 얻는다.

$$\begin{aligned}\frac{u_i^{n+1}-u_i^n}{k} &= \frac{1}{2}\left(\frac{1}{2}\sigma^2 x_i^2 \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{h^2} + rx_i \frac{u_{i+1}^n - u_{i-1}^n}{2h} - ru_i^n\right) \qquad (1.29)\\ &+ \frac{1}{2}\left(\frac{1}{2}\sigma^2 x_i^2 \frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}}{h^2} + rx_i \frac{u_{i+1}^{n+1} - u_{i-1}^{n+1}}{2h} - ru_i^{n+1}\right).\end{aligned}$$

이러한 함축적 방법은 시간의 크기에 영향을 받지 않는 장점이 있다. 이제 위의 식 (1.30)를 정리하면,

$$\begin{aligned}\alpha_i u_{i-1}^{n+1} + \beta_i u_i^{n+1} + \gamma_i u_{i+1}^{n+1} &= \frac{u_i^n}{k} + \frac{1}{4}\sigma^2 x_i^2 \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{h^2}\\ &+ rx_i \frac{u_{i+1}^n - u_{i-1}^n}{4h} - \frac{r}{2}u_i^n, \qquad (1.30)\end{aligned}$$

여기서

$$\alpha_i = \frac{rx_i}{4h} - \frac{\sigma^2 x_i^2}{4h^2}, \quad \beta_i = \frac{1}{k} + \frac{\sigma^2 x_i^2}{2h^2} + r, \quad \gamma_i = -\frac{rx_i}{4h} - \frac{\sigma^2 x_i}{4h^2}$$

이다.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% BScn.m %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clf; clear;
E=230; sigma=0.5; r=0.03; T=1; Nx=50; Nt=100; L=800; k=T/Nt;
x=linspace(0,L,Nx); h =x(2)-x(1);
u(1:Nx,1:Nt+1)=0;
for i=1:Nx
    if x(i) < E
        u(i,1)= 0;
    else
        u(i,1)= x(i)-E;
    end
end
N = Nx-2;
for i=1:N
     dd(i)=1/k+(sigma*i)^2/2+r; c(i)=-r*i/4 - ((sigma*i)^2)/4;
     a(i)=r*(i+1)/4-((sigma*(i+1))^2)/4;
end
for n=1:Nt
    d=dd;
    for i=1:N
         b(i) = u(i+1,n)/k + (sigma*i)^2*(u(i+2,n)-2*u(i+1,n)+u(i,n))/4 ...
             + r*i*(u(i+2,n)-u(i,n))/4 - r*u(i+1,n)/2;
    end
     u(Nx,n+1)= L - E*exp(-r*k*n); b(N) = b(N) - c(N)*u(Nx,n+1);
    for i = 2:N
        xmult= a(i-1)/d(i-1); d(i) = d(i) - xmult*c(i-1);
        b(i) = b(i) - xmult*b(i-1);
    end
    u(N+1,n+1) = b(N)/d(N);
    for i = N-1:-1:1
        u(i+1,n+1) = (b(i) - c(i)*u(i+2,n+1))/d(i);
    end
end
plot(x,u(:,1:20:Nt+1),'ko-')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

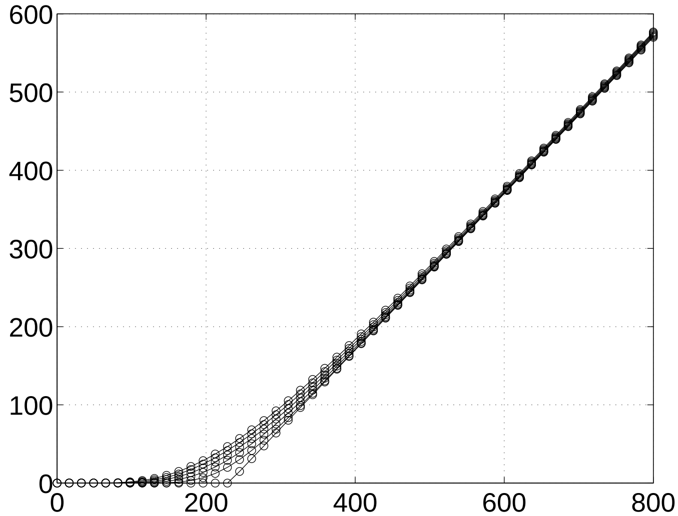



그림 1.14: 크랭크 니콜슨 방법에 의한 옵션 가격 결정

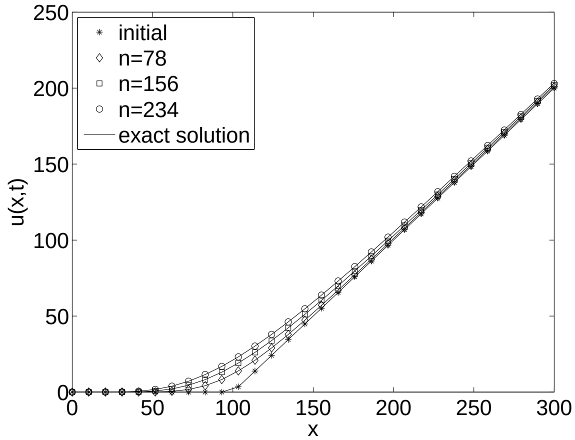
## 3.4 안정성 테스트

### 3.4.1 명시적 유한차분법

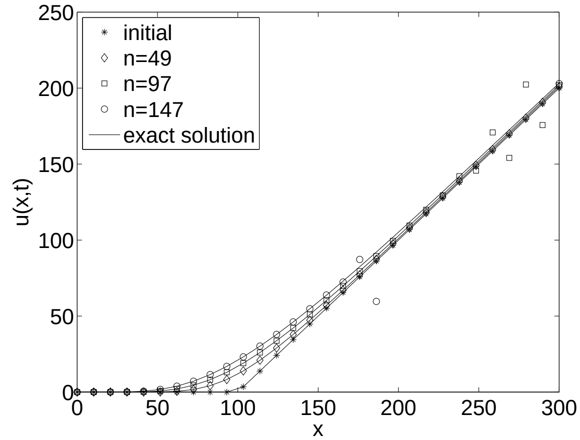
다음은 무위험이자율이 $r = 0.03$, 변동성이 $\sigma = 0.5$, 현재시점이 $t = 0$, 만기 시점이 $T = 1$, 그리고 행사가격이 $E = 100$인 유럽형 콜옵션의 가격을 구하는 MATLAB 코드이다.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%% BSex_stability.m %%%%%%%%%%%%%%%%%%%%%%%%%%%
clf; clear; E=100; sigma=0.5; r=0.03; L=300;
alpha = 0.000005; Nx = 30; x=linspace(0,L,Nx); h=x(2)-x(1);
k = 2*alpha*(h/sigma)^2; T = 1.0; Nt = round(T/k); u(1:Nx,1:Nt+1)=0;
for i=1:Nx
    if x(i)<= E
        u(i,1)=0;
    else
        u(i,1)=x(i)-E;
    end
end
for n=2:Nt+1
    u(Nx,n)=L-E*exp(-r*k*(n-1));
end
exu=u;
for n=1:Nt
    for i=2:Nx-1
        u(i,n+1)=u(i,n) + k*((1/2)*(sigma^2)*((i-1)*h)^2*...
            ((u(i+1,n)-2*u(i,n)+u(i-1,n))/(h^2)) +...
            r*(i-1)*h*((u(i+1,n)-u(i-1,n))/(2*h))-r*u(i,n));
    end
    for i=1:Nx
        d1(i)=(log(x(i)/E)+(r+sigma^2/2)*k*n)/(sigma*sqrt(k*n));
        d2(i)=d1(i)-sigma*sqrt(k*n);
        exu(i,n+1)=x(i)*normcdf(d1(i))-E*exp(-r*k*n)*normcdf(d2(i));
    end
end
plot(x,u(:,1),'k*',x,u(:,round(Nt/3)),...
    'kd',x,u(:,round(2*Nt/3)),'ks',x,u(:,Nt+1),'ko'); hold
plot(x,exu(:,1),'k',x,exu(:,round(Nt/3)),...
    'k',x,exu(:,round(2*Nt/3)),'k',x,exu(:,Nt+1),'k')
legend('initial','n=78','n=156','n=234','exact solution',2)
xlabel('x','FontSize',20); ylabel('u(x,t)','FontSize',20)
axis([0 L 0 L-0.5*E])
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

앞의 코드를 이용하여 $\Delta t = 0.0043$인 경우와 $\Delta t = 0.0068$인 경우에 다음과 같은 결과를 얻을 수 있다.




(a)




(b)

그림 1.15: (a) $\Delta t = 0.0043$인 안정한 상태 (b) $\Delta t = 0.0068$인 불안정한 상태

그림 1.15 (a) 에 비해 그림 1.15 (b)의 계산된 값은 정확한 해로 수렴하지 않는 불안정한 상태임을 확인할 수 있다.

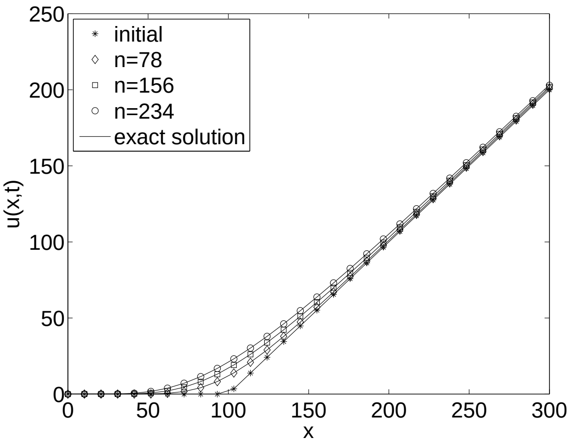
### 3.4.2 함축적 유한차분법

다음은 무위험이자율이 $r = 0.03$, 변동성이 $\sigma = 0.5$, 현재시점이 $t = 0$, 만기 시점이 $T = 1$, 그리고 행사가격이 $E = 100$인 유럽형 콜옵션의 가격을 구하는 MATLAB 코드이다.

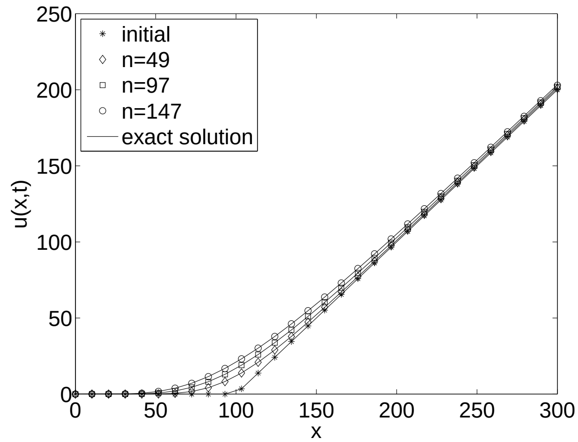
```
%%%%%%%%%%%%%%%%%%%%%%%%%% BSim_stability.m %%%%%%%%%%%%%%%%%%%%%%%%%%
clf; clear; E=100; L=300; alpha = 0.000005; %alpha = 0.000008;
Nx = 30; x=linspace(0,L,Nx); h=x(2)-x(1); sigma=0.5; r=0.03; T = 1.0;
k = 2*alpha*(h/sigma)^2; Nt = round(T/k); u(1:Nx,1:Nt+1)=0; N=Nx-2;
u(:,1)= max(0,x-E);
for n=2:Nt+1
    u(Nx,n)=L-E*exp(-r*k*(n-1));
end
exu=u;
for i=1:N
   dd(i)=1/k+(sigma*i)^2+r; c(i)=-r*i/2-((sigma*i)^2)/2;
   a(i)=r*(i+1)/2-((sigma*(i+1))^2)/2;
end
for n=1:Nt
    d=dd;
    for i=1:N-1
         b(i)=u(i+1,n)/k;
    end
    b(N)=u(N+1,n)/k - c(N)*u(Nx,n+1);
    for i = 2:N
        xmult= a(i-1)/d(i-1);
        d(i) = d(i) - xmult*c(i-1); b(i) = b(i) - xmult*b(i-1);
    end
    u(N+1,n+1) = b(N)/d(N);
    for i = N-1:-1:1
        u(i+1,n+1) = (b(i) - c(i)*u(i+2,n+1))/d(i);
    end
    for i=1:Nx
        d1(i)=(log(x(i)/E)+(r+sigma^2/2)*k*n)/(sigma*sqrt(k*n));
        d2(i)=d1(i)-sigma*sqrt(k*n);
        exu(i,n+1)=x(i)*normcdf(d1(i))-E*exp(-r*k*n)*normcdf(d2(i));
    end
end
plot(x,u(:,1),'k*',x,u(:,Nt/3),'kd',x,u(:,2*Nt/3),'ks',x,u(:,Nt+1),'ko');
hold
```

```
plot(x,exu(:,1),'k',x,exu(:,Nt/3),'k',x,exu(:,2*Nt/3),'k',x,exu(:,Nt+1),'k')
legend('initial','n=78','n=156','n=234','exact solution',2)
xlabel('x','FontSize',20); ylabel('u(x,t)','FontSize',20);
axis([0 L 0 L-0.5*E])
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

앞의 코드를 이용하여 $\Delta t = 0.0043$인 경우와 $\Delta t = 0.0068$인 경우에 다음과 같은 결과를 얻을 수 있다.




(a)




(b)

그림 1.16: (a) $\Delta t = 0.0043$인 안정한 상태 (b) $\Delta t = 0.0068$인 안정한 상태

앞서 명시적 유한차분법은 $\alpha$의 값에 따라 안정한 상태와 불안정한 상태의 결과를 얻었지만 함축적 유한차분법은 위의 그림 1.16에서 볼 수 있듯이 (a)와 (b) 모두 안정한 상태임을 확인할 수 있다.

### 3.4.3 크랭크 니콜슨 유한차분법

## 3.5 수렴성 테스트

$u(x_i, t^n)$는 Black-Scholes 방정식의 정확한 해를 나타낸다. 다음은 정확한 해를 수치기법에 대입함으로써 명시적 유한차분법의 국소절단오차를 찾는 과정이다. 노드 $(x_i, t^n)$에서 국소절단오차는 다음과 같이 구한다.

$$
\begin{aligned}
T(x_i, t^n) \;\; = \;\; & \frac{u(x_i, t^{n+1}) - u(x_i, t^n)}{k} - \frac{\sigma^2 x_i^2}{2} \frac{u(x_{i+1}, t^n) - 2u(x_i, t^n) + u(x_{i-1}, t^n)}{h^2} \\
& - r x_i \frac{u(x_{i+1}, t^n) - u(x_{i-1}, t^n)}{2h} + r u(x_i, t^n).
\end{aligned}
$$

이제 노드$(x_i, t^n)$에서 테일러전개를 하면 각각의 항을 다음과 같이 나타낼 수 있다.

$$
\begin{aligned}
T(x_i, t^n) \;\; = \;\; & u_t(x_i, t^n) + \frac{k}{2} u_{tt}(x_i, t^n) + \mathcal{O}(k^2) \\
& - \frac{\sigma^2 x_i^2}{2} \left[ u_{xx}(x_i, t^n) + \frac{h^2}{12} u_{xxxx}(x_i, t^n) + \mathcal{O}(h^4) \right] \\
& - r x_i \left[ u_x(x_i, t^n) + \frac{h^2}{3} u_{xxx}(x_i, t^n) + \mathcal{O}(h^4) \right] + r u(x_i, t^n).
\end{aligned}
$$

여기서 $u(x_i, t^n)$은 Black-Scholes 방정식을 만족하므로 다음이 성립한다.

$$\begin{aligned} T(x_i, t^n) &= \frac{k}{2}u_{tt}(x_i, t^n) - \frac{\sigma^2 h^2 x_i^2}{24}u_{xxxx}(x_i, t^n) - rx_i\frac{h^2}{3}u_{xxx}(x_i, t^n) + \mathcal{O}(k^2) + \mathcal{O}(h^4) \\ &= \mathcal{O}(k) + \mathcal{O}(h^2). \end{aligned} \tag{1.31}$$

식(1.31)으로부터 명시적 유한차분법은 1차 시간 정확하고 2차 공간 정확함을 알 수 있다. 다음 결과를 확인하기 위해 다음의 테스트를 수행해보자.

### 3.5.1 명시적 유한차분법

### 3.5.2 함축적 유한차분법

유러피언 콜옵션방정식의 함축적 유한 차분법의 수렴성을 알아보기 위해 다음의 테스트를 수행해보자. 무위험이자율이 $r = 0.03$, 변동성이 $\sigma = 0.5$, 현재시점이 $t = 0$, 만기 시점이 $T = 1$, 그리고 행사가격이 $E = 350$인 유럽형 콜옵션의 가격을 구하는 MATLAB 코드이다. 이 때, 초기 조건은 $u(x, 0) = \max(0, x - E)$, $T = 0.1$, $h = 1/N$, $\Delta t = h/375$을 이용하였다. MATLAB 코드3.5.2을 실행하면 다음의 결과를 얻을 수 있다.

```
%%%%%%%%%%%%%%%%%%%%%%% BSex_convergence_test.m %%%%%%%%%%%%%%%%%%%%%%%%
clear; clc;
E=100; sigma=0.5; r=0.03; L=300; T=0.1;
for iter=1:5
   N = 16*(2^iter);
   x=linspace(0,L,N); h=x(2)-x(1); k=h/375;
   Nt=round(T/k); u(1:N,1:Nt+1)=0;
   for i=1:N
       if x(i)<= E
           u(i,1)=0;
       else
        u(i,1)=x(i)-E;
       end
   end
   for n=2:Nt+1
    u(N,n)=L-E*exp(-r*k*(n-1));
   end
exact=u;
    for n=1:Nt
        for i=2:N-1
            u(i,n+1)=u(i,n) + k*((1/2)*(sigma^2)*((i-1)*h)^2*...
                      ((u(i+1,n)-2*u(i,n)+u(i-1,n))/(h^2)) +...
                      r*(i-1)*h*((u(i+1,n)-u(i-1,n))/(2*h))-r*u(i,n));
        end
        for i=1:N
            d1(i)=(log(x(i)/E)+(r+sigma^2/2)*k*n)/(sigma*sqrt(k*n));
            d2(i)=d1(i)-sigma*sqrt(k*n);
            exact(i,n+1)=x(i)*normcdf(d1(i))-E*exp(-r*k*n)*normcdf(d2(i));
        end
    end
   hh(iter)=h; tt(iter)=k; F = u(:,Nt+1) - exact(:,Nt+1);
   err(iter) = sqrt(sum(sum(F.^2)))/N;
end
Order=[log(err(1)/err(2))/log(2) log(err(2)/err(3))/log(2) ...
    log(err(3)/err(4))/log(2) log(err(4)/err(5))/log(2)]';
```

```
fprintf('-----------------------------------------------\n')
fprintf('    h          dt        l2 error       order    \n')
fprintf('-----------------------------------------------\n')
fprintf('%8.5f    %8.6f     %8.6f      \n',hh(1), tt(1) ,err(1))
for iter = 2:5
fprintf('%8.5f    %8.6f     %8.6f      %8.6f \n',hh(iter),tt(iter), ...
    err(iter),Order(iter-1))
end
fprintf('-----------------------------------------------\n')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
>> bsim_convergence_test
-------------------------------------------------
    h           dt         l2 error       order
-------------------------------------------------
25.80645    0.068817     0.068311
12.69841    0.033862     0.026456      1.368545
 6.29921    0.016798     0.009680      1.450521
 3.13725    0.008366     0.003479      1.476111
 1.56556    0.004175     0.001240      1.488203
-------------------------------------------------
```

### 3.5.3 크랭크 니콜슨 유한차분법

유러피언 콜옵션방정식의 크랭크 니콜슨 유한 차분법의 수렴성을 알아보기 위해 다음의 테스트를 수행해보자. 무위험이자율이 $r = 0.03$, 변동성이 $\sigma = 0.5$, 현재시점이 $t = 0$, 만기 시점이 $T = 1$, 그리고 행사가격이 $E = 350$인 유럽형 콜옵션의 가격을 구하는 MATLAB 코드이다. 이 때, 초기조건은 $u(x,0) = \max(0, x-E)$, $T = 0.1$, $h = 1/N$, $\Delta t = h/375$을 이용하였다. MATLAB 코드3.5.3을 실행하면 다음의 결과를 얻을 수 있다.

```
%%%%%%%%%%%%%%%%%%%%%%%% BScn_convergence_test.m %%%%%%%%%%%%%%%%%%%%%%%%%
clear; clc;
E=350; sigma=0.5; r=0.03; L=800; T=0.1;
for iter=1:5
   Nx = 16*(2^iter);
   x=linspace(0,L,Nx); h=x(2)-x(1); k=h/375;
   Nt=round(T/k); u(1:Nx,1:Nt+1)=0;
   N=Nx-2;
   u(:,1)= max(0,x-E);
   for n=2:Nt+1
       u(Nx,n)=L-E*exp(-r*k*(n-1));
   end
   exact=u;
   for i=1:N
     dd(i)=1/k+(sigma*i)^2/2+r; c(i)=-r*i/4 - ((sigma*i)^2)/4;
     a(i)=r*(i+1)/4-((sigma*(i+1))^2)/4;
   end
   for n=1:Nt
       d=dd;
       for i=1:N
         b(i) = u(i+1,n)/k + (sigma*i)^2*(u(i+2,n)-2*u(i+1,n)+u(i,n))/4 ...
               + r*i*(u(i+2,n)-u(i,n))/4 - r*u(i+1,n)/2;
       end
       u(Nx,n+1)= L - E*exp(-r*k*n); b(N) = b(N) - c(N)*u(Nx,n+1);
       for i = 2:N
          xmult= a(i-1)/d(i-1); d(i) = d(i) - xmult*c(i-1);
          b(i) = b(i) - xmult*b(i-1);
       end
       u(N+1,n+1) = b(N)/d(N);
       for i = N-1:-1:1
          u(i+1,n+1) = (b(i) - c(i)*u(i+2,n+1))/d(i);
       end
       for i=1:Nx
       d1(i)=(log(x(i)/E)+(r+sigma^2/2)*k*n)/(sigma*sqrt(k*n));
       d2(i)=d1(i)-sigma*sqrt(k*n);
```

```
        exact(i,n+1)=x(i)*normcdf(d1(i))-E*exp(-r*k*n)*normcdf(d2(i));
        end
    end
    hh(iter)=h; tt(iter)=k; F = u(:,Nt+1) - exact(:,Nt+1);
    err(iter) = sqrt(sum(sum(F.^2)))/Nx;
end
Order=[log(err(1)/err(2))/log(2) log(err(2)/err(3))/log(2) ...
    log(err(3)/err(4))/log(2) log(err(4)/err(5))/log(2)]';
fprintf('------------------------------------------------\n')
fprintf('    h          dt         l2 error       order    \n')
fprintf('------------------------------------------------\n')
fprintf('%8.5f    %8.6f     %8.6f      \n',hh(1), tt(1) ,err(1))
for iter = 2:5
fprintf('%8.5f    %8.6f     %8.6f      %8.6f \n',hh(iter),tt(iter), ...
    err(iter),Order(iter-1))
end
fprintf('------------------------------------------------\n')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
>> bscn_convergence_test
-------------------------------------------------
    h           dt          l2 error       order
-------------------------------------------------
25.80645    0.068817     0.052859
12.69841    0.033862     0.026682      0.986274
 6.29921    0.016798     0.018524      0.526444
 3.13725    0.008366     0.013077      0.502435
 1.56556    0.004175     0.009243      0.500497
-------------------------------------------------
```