전산유체역학

PJ#3: Parabolic Type

항공우주 및 기계공학부

담당교수 : 김문상 교수님

제출일 : 2021-05-11

2016121150 윤준영

1. 서론

대표적인 포물선 특성(Parabolic Type)의 편미분 방정식인 1차원 2차 비정상 열정도 방정식(1-D 2nd Order Unsteady Heat Conduction Equation)을 여러 수치해석 방법을 이용하여 해석하였다.

(*) 1-D 2nd Order Unsteady Heat Conduction Equation

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}, \qquad (\alpha : constant)$$

1-D 2nd Order Unsteady Heat Conduction Equation은 초기 조건(Initial Condition)과 2개의 경계 조건(Boundary Condition)이 필요하다. 주어진 Initial Condition과 Boundary Condition, Exact Solution은 다음과 같다. 주기함수의 peak $A = 1$, diffusion coefficient $\alpha = 2$로 설정하였다.

(1) Initial Condition:

$$u(x, 0) = A \sin x = \sin x, \qquad for\ A = 1$$

(2) Boundary Condition:

$$u(0, t) = u(x_{max}, t) = 0$$

(3) Exact Solution:

$$u(x, t) = Ae^{-\alpha t} \sin x = e^{-t} \sin x, \qquad for\ \alpha = 1$$

아래에서 서술할 식과 코드에서 하첨자 i는 격자점의 위치를 의미하고 상첨자 n은 현재 시간, $n+1$은 미지수가 존재하는 다음 시간 단계(time step)을 의미한다. 격자의 개수는 $gridn$, 그래프의 개수는 gno, 그래프 사이의 시간 간격은 $tint$으로 표기하고, CFL은 다음과 같이 정의한다.

$$CFL = \frac{\alpha \Delta t}{(\Delta x)^2}, \qquad \Delta t = \frac{CFL \cdot (\Delta x)^2}{\alpha}$$

여기서 Δx : space interval, Δt : time interval이다.

2. Explicit Method

2.1. FTCS Method

FTCS Method는 시간에 대해서 1차 정확도를 가지는 전방 차분법, 공간에 대해서는 2차 정확도를 가지는 중심 차분법을 사용하여 유도한 유한 차분 방정식이다. 전체적으로는 1차 정확도를 가지며, 다음과 같이 나타내어진다.

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \alpha \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{(\Delta x)^2}$$

위 식을 미지수 u_i^{n+1}에 대해 표현하면 다음과 같다.

$$u_i^{n+1} = u_i^n + \frac{\alpha \Delta t}{(\Delta x)^2}(u_{i+1}^n - 2u_i^n + u_{i-1}^n)$$

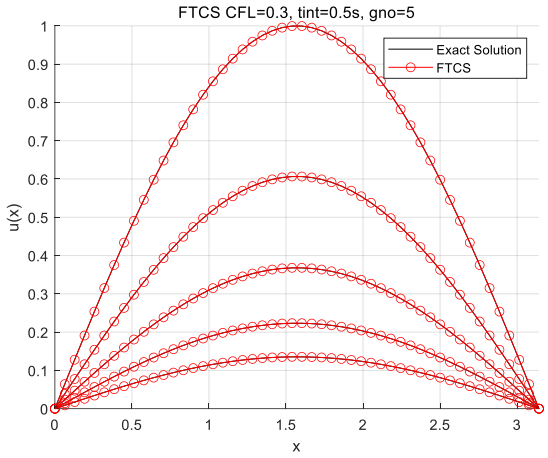
작성한 MATLAB CODE는 다음과 같다.

```
%% FTCS Method
clear; close all;
%% input
prompt = {'gridn', 'gno', 'tint', 'CFL'};
dlgtitle = 'Data'; dims = [1 35];
definput = {'50','5', '0.5', '0.5'}; % set default value
inputdata = inputdlg(prompt,dlgtitle,dims,definput); % input data
dialogue
gridn = str2double(inputdata{1}); % no. of grid
gno = str2double(inputdata{2}); % no. of graph
tint = str2double(inputdata{3}); % time interval between graphs
CFL = str2double(inputdata{4}); % CFL number
A=1; % peak value (amplitude)
a=1; % alpha
wid=pi; % width of wave
dx=wid/(gridn-1); % width of grid dx
dt=CFL*(dx)^2/a; % time interval between u
t=0:dt:tint*(gno-1); % n with dt interval
x=linspace(0,wid,gridn);
no=length(t); % max number of n to calculate
%% initial condition
for i=1:1:gridn
    ue(i,1)=A*sin((i-1)*dx);
end
%% boundary condition
for n=1:1:gno
    ue(1,n)=0;
    ue(gridn,n)=0;
end
%% exact solution
for n=2:1:gno
    for i=2:1:gridn
        ue(i,n)=A*exp(-a*tint*(n-1))*sin(dx*(i-1));
    end
end
%% FTCS method
u=ue;
for n=1:1:no+1
    for i=2:1:gridn-1
        u(i,n+1)=u(i,n)+CFL*(u(i+1,n)-2*u(i,n)+u(i-1,n));
    end
end
for n=1:1:gno
    ui(:,n)=u(:,1+(n-1)*round(tint/dt));
end
%% plot
figure(1)
hold on; grid on;
for n=1:1:gno
    plot(x,ue(:,n),'k') % exact solution
    plot(x,ui(:,n),'-or') % FTCS
end
xlim([0,pi])
title(['FTCS CFL=',num2str(CFL),', tint=',num2str(tint),'s,
gno=',num2str(gno)])
```

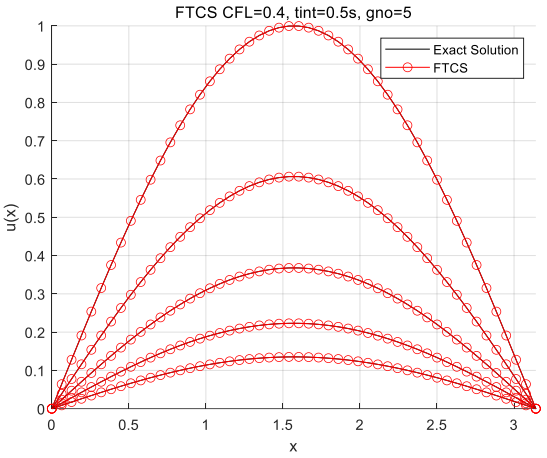
```
xlabel('x')
ylabel('u(x)')
legend({'Exact Solution','FTCS'})
```

< Table. 1 – FTCS Method MATLAB CODE >

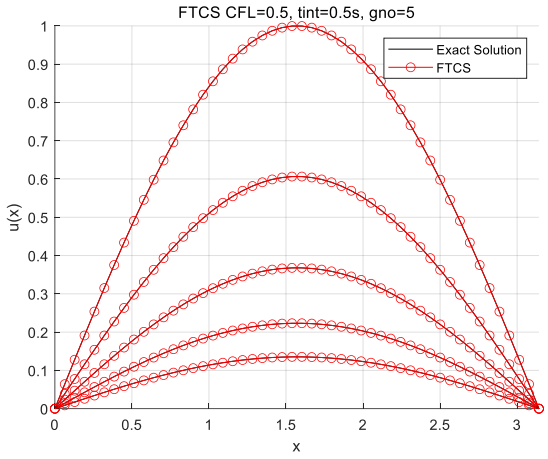
다음 Fig. 1 ~ Fig. 6은 그래프의 시간 간격을 $tint = 0.5s$, 격자의 개수 $gridn = 50$으로 설정한 후 CFL 값을 $0.3, 0.4, 0.5, 0.6, 0.7, 1.0$으로 변화시켜 FTCS Method로 해석한 결과이다.



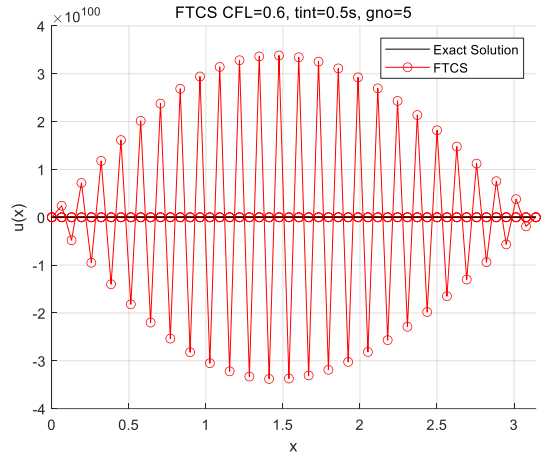
< Fig. 1 – FTCS CFL=0.3 >



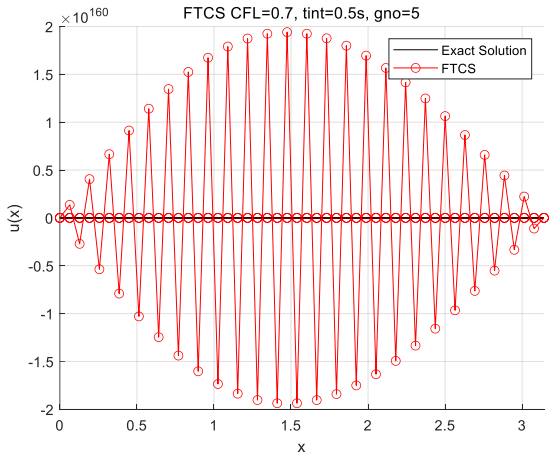
< Fig. 2 – FTCS CFL=0.4 >



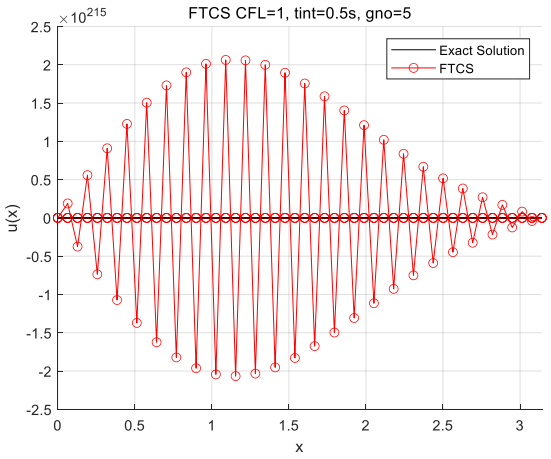
< Fig. 3 – FTCS CFL=0.5 >



< Fig. 4 – FTCS CFL=0.6 >



< Fig. 5 – FTCS CFL=0.7 >



< Fig. 6 – FTCS CFL=1.0 >

Fig. 1, Fig. 2, Fig. 3을 보면 CFL 값이 0.5 이하에서는 FTCS Method를 사용한 해석값이 안정적으로 수렴하며 오차도 그래프 상으로 확인할 수 없을 정도로 매우 적은 것을 확인할 수 있었다. 하지만 Fig. 4, Fig. 5, Fig. 6을 보면 CFL 값이 0.5를 초과하였을 때 FTCS Method의 해석값이 2×10^{100} 이 넘어가는 큰 값을 가지게 되고 해석값은 CFL 값이 커질수록 더 크게 발산하는 것을 확인할 수 있었다. 따라서 FTCS Method는 CFL<=0.5 일 때 수렴하는 해를 가지며 Conditionally Stable함을 알 수 있다.

2.2. Richardson Method

Richardson Method는 시간에 대해 중심 차분법(Central Time)을 사용하고, 공간에 대해서도 중심 차분법(Central Space)을 사용하는 방법으로 공간과 시간에 대해 2차 정확도를 가지도록 유도된 해석 방법이다. Richarson Method로 풀이한 Heat Conduction Equation은 다음과 같다.

$$\frac{u_i^{n+1} - u_i^{n-1}}{2\Delta t} = \alpha \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{(\Delta x)^2}$$

위 식을 미지수 u_i^{n+1}에 대해 표현하면 다음과 같다.

$$u_i^{n+1} = u_i^{n-1} + \frac{2\alpha\Delta t}{(\Delta x)^2}(u_{i+1}^n - 2u_i^n + u_{i-1}^n)$$

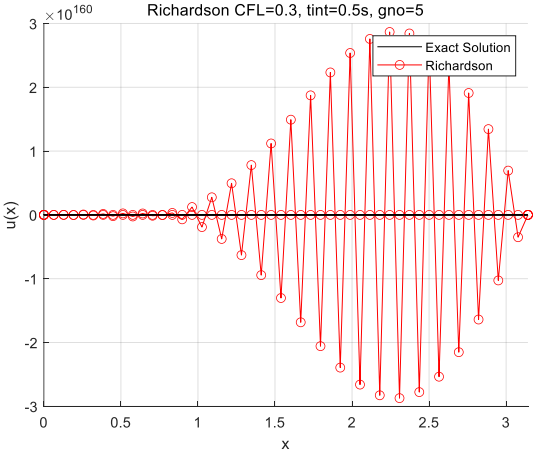
작성한 MATLAB CODE는 다음과 같다.

```
%% Richardson Method
clear; close all;
%% input
prompt = {'gridn', 'gno', 'tint', 'CFL'};
dlgtitle = 'Data'; dims = [1 35];
definput = {'50','5', '0.5', '0.5'}; % set default value
inputdata = inputdlg(prompt,dlgtitle,dims,definput); % input data
```

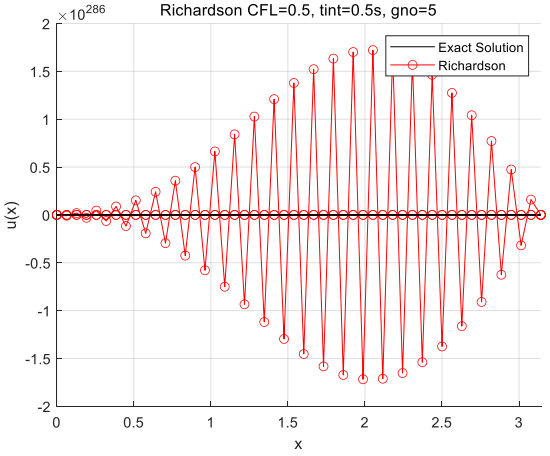
```
dialogue
gridn = str2double(inputdata{1}); % no. of grid
gno = str2double(inputdata{2}); % no. of graph
tint = str2double(inputdata{3}); % time interval between graphs
CFL = str2double(inputdata{4}); % CFL number
A=1; % peak value (amplitude)
a=1; % alpha
wid=pi; % width of wave
dx=wid/(gridn-1); % width of grid dx
dt=CFL*(dx)^2/a; % time interval between u
t=0:dt:tint*(gno-1); % n with dt interval
x=linspace(0,wid,gridn);
no=length(t); % max number of n to calculate
%% initial condition
for i=1:1:gridn
    ue(i,1)=A*sin((i-1)*dx);
end
%% boundary condition
for n=1:1:gno
    ue(1,n)=0;
    ue(gridn,n)=0;
end
%% exact solution
for n=2:1:gno
    for i=2:1:gridn
        ue(i,n)=A*exp(-a*tint*(n-1))*sin(dx*(i-1));
        u1(i)=A*exp(-a*dt)*sin(dx*(i-1)); % n=2
    end
end
%% Richarson method
u=ue; u(:,2)=u1;
for n=2:1:no+1
    for i=2:1:gridn-1
        u(i,n+1)=u(i,n-1)+2*CFL*(u(i+1,n)-2*u(i,n)+u(i-1,n));
    end
end
for n=1:1:gno
    ui(:,n)=u(:,1+(n-1)*round(tint/dt));
end
%% plot
figure(1)
hold on; grid on;
for n=1:1:gno
    plot(x,ue(:,n),'k') % exact solution
    plot(x,ui(:,n),'-or') % Richardson
end
xlim([0,pi])
title(['Richardson CFL=',num2str(CFL),', tint=',num2str(tint),'s,
gno=',num2str(gno)])
xlabel('x')
ylabel('u(x)')
legend({'Exact Solution','Richardson'})
```

< Table. 2 –Richardson Method MATLAB CODE >

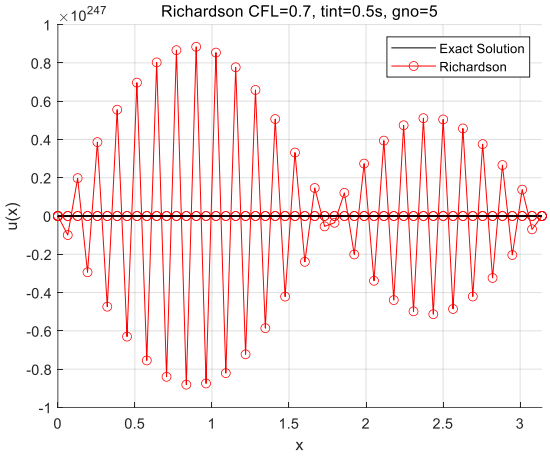
다음 Fig. 7 ~ Fig. 12는 그래프의 시간 간격을 $tint = 0.5s$, 격자의 개수 $gridn = 50$으로 설정한 후 CFL 값을 $0.3, 0.5, 0.7, 1.0, 2.0, 3.0$으로 변화시켜 Richarson Method로 해석한 결과이다.



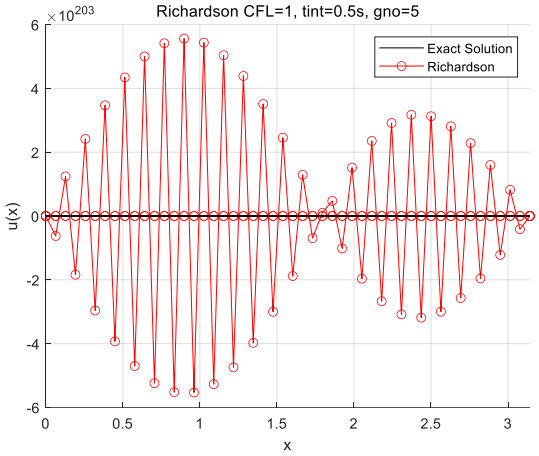
< Fig. 7 – Richardson CFL=0.3 >



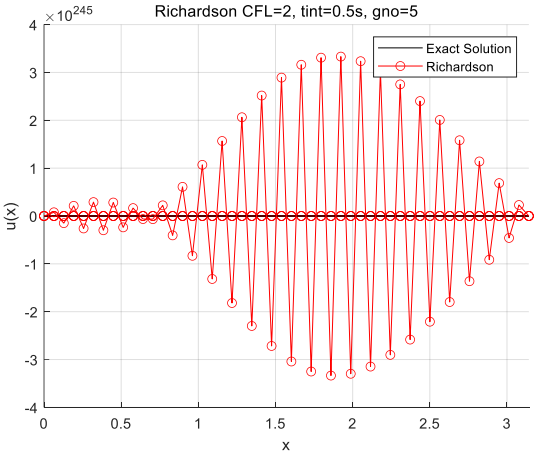
< Fig. 8 – Richardson CFL=0.5 >



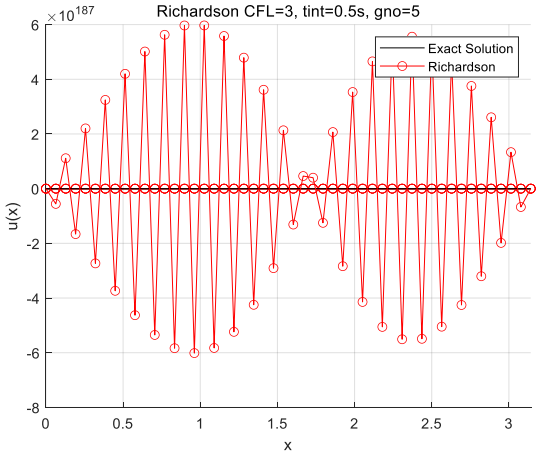
< Fig. 9 – Richardson CFL=0.7 >



< Fig. 10 – Richardson CFL=1.0 >



< Fig. 11 – Richardson CFL=2.0 >



< Fig. 12 – Richardson CFL=3.0 >

위 Fig. 7부터 Fig. 12를 보면 Richardson Method로 구한 해석값은 CFL값과 상관없이 발산하는 것을 확인할 수 있다. 따라서 Richardson Method는 수렴하는 해를 나타내지 못하며 Unconditionally

Unstable함을 알 수 있다.

2.3. Du Fort-Frankel Method

Du Fort-Frankel Method는 Richardson Method에서 u_i^n 대신 u_i^{n+1}과 u_i^{n-1}의 평균값을 사용하는 방법이다. Richardson Method와 마찬가지로 시간과 공간에서 모두 2차 정확도를 가지며, 다음과 같이 나타내어진다.

$$\frac{u_i^{n+1}-u_i^n}{\Delta t}=\alpha\frac{u_{i+1}^n-2\frac{u_i^{n+1}+u_i^{n-1}}{2}+u_{i-1}^n}{(\Delta x)^2}$$

위 식을 미지수 u_i^{n+1}에 대해 표현하면 다음과 같다.

$$u_i^{n+1}=u_i^{n-1}+\frac{2\alpha\Delta t}{(\Delta x)^2}(u_{i+1}^n-u_i^{n+1}-u_i^{n-1}+u_{i-1}^n)$$

$$\left(1+\frac{2\alpha\Delta t}{(\Delta x)^2}\right)u_i^{n+1}=\left(1-\frac{2\alpha\Delta t}{(\Delta x)^2}\right)u_i^{n-1}+\frac{2\alpha\Delta t}{(\Delta x)^2}(u_{i+1}^n+u_{i-1}^n)$$

$$u_i^{n+1}=\frac{1}{1+\frac{2\alpha\Delta t}{(\Delta x)^2}}\left(\left(1-\frac{2\alpha\Delta t}{(\Delta x)^2}\right)u_i^{n-1}+\frac{2\alpha\Delta t}{(\Delta x)^2}(u_{i+1}^n+u_{i-1}^n)\right)$$

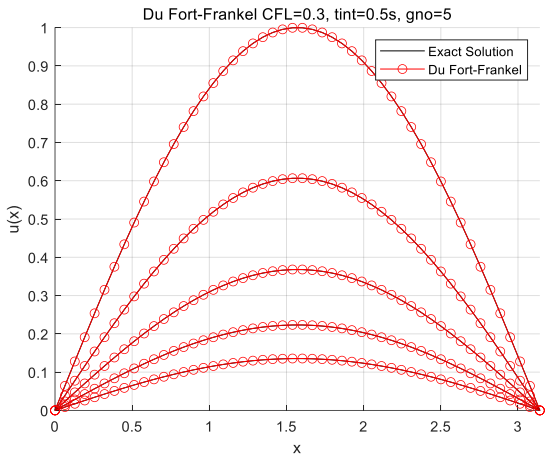
작성한 MATLAB CODE는 다음과 같다.

```
%% Du Fort-Frankel Method
clear; close all;
%% input
prompt = {'gridn', 'gno', 'tint', 'CFL'};
dlgtitle = 'Data'; dims = [1 35];
definput = {'50','5', '0.5', '0.5'}; % set default value
inputdata = inputdlg(prompt,dlgtitle,dims,definput); % input data
dialogue
gridn = str2double(inputdata{1}); % no. of grid
gno = str2double(inputdata{2}); % no. of graph
tint = str2double(inputdata{3}); % time interval between graphs
CFL = str2double(inputdata{4}); % CFL number
A=1; % peak value (amplitude)
a=1; % alpha
wid=pi; % width of wave
dx=wid/(gridn-1); % width of grid dx
dt=CFL*(dx)^2/a; % time interval between u
t=0:dt:tint*(gno-1); % n with dt interval
x=linspace(0,wid,gridn);
no=length(t); % max number of n to calculate
%% initial condition
for i=1:1:gridn
    ue(i,1)=A*sin((i-1)*dx);
end
%% boundary condition
for n=1:1:gno
    ue(1,n)=0;
    ue(gridn,n)=0;
end
```

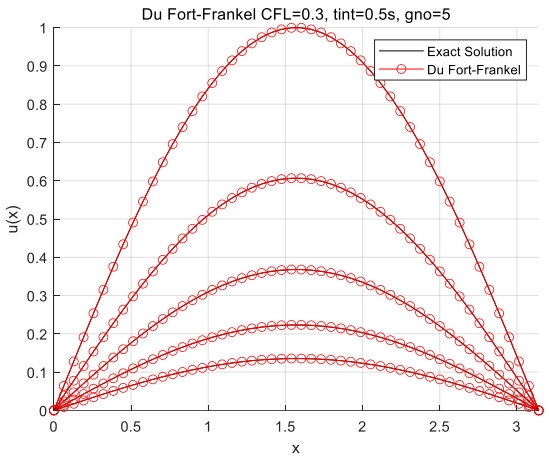
```
%% exact solution
for n=2:1:gno
    for i=2:1:gridn
        ue(i,n)=A*exp(-a*tint*(n-1))*sin(dx*(i-1));
        u1(i)=A*exp(-a*dt)*sin(dx*(i-1)); % n=2
    end
end
%% Du Fort-Frankel method
u=ue; u(:,2)=u1;
for n=2:1:no+1
    for i=2:1:gridn-1
        u(i,n+1)=((1-2*CFL)*u(i,n-1)+2*CFL*(u(i+1,n)+u(i-1,n)))/(1+2*CFL);
    end
end
for n=1:1:gno
    ui(:,n)=u(:,1+(n-1)*round(tint/dt));
end
%% plot
figure(1)
hold on; grid on;
for n=1:1:gno
    plot(x,ue(:,n),'k') % exact solution
    plot(x,ui(:,n),'-or') % Du Fort-Frankel
end
xlim([0,pi])
title(['Du Fort-Frankel CFL=',num2str(CFL),', tint=',num2str(tint),'s,
gno=',num2str(gno)])
xlabel('x')
ylabel('u(x)')
legend({'Exact Solution','Du Fort-Frankel'})
```

< Table. 3 – Du Fort-Frankel Method MATLAB CODE >

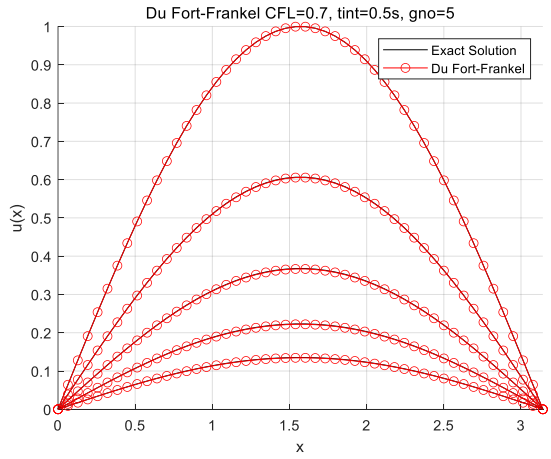
다음 Fig. 13 ~ Fig. 18는 마찬가지로 그래프의 시간 간격을 $tint = 0.5s$, 격자의 개수 $gridn = 50$으로 설정한 후 CFL 값을 $0.3, 0.5, 0.7, 1.0, 2.0, 3.0$으로 변화시켜 Du Fort-Frankel Method로 해석한 결과이다.



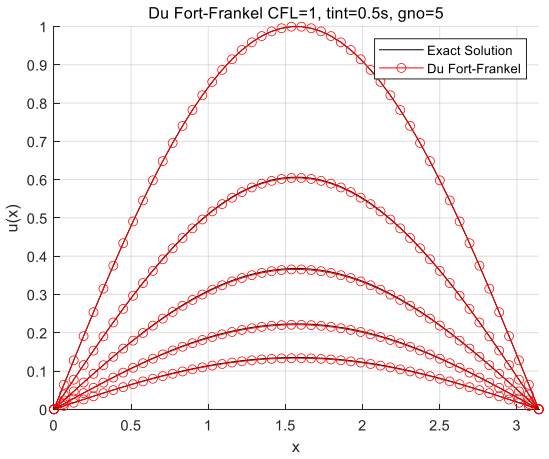
< Fig. 13 – Du Fort-Frankel CFL=0.3 >



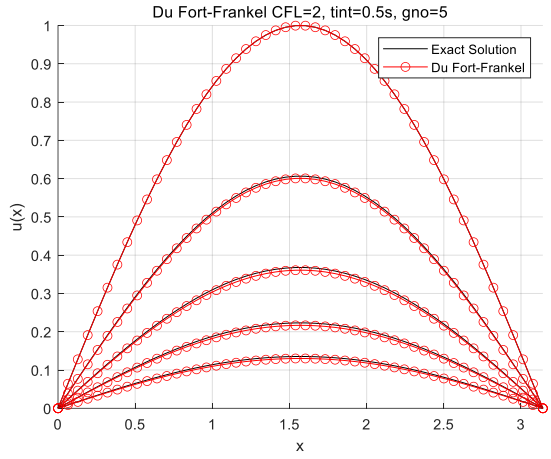
< Fig. 14 – Du Fort-Frankel CFL=0.5 >



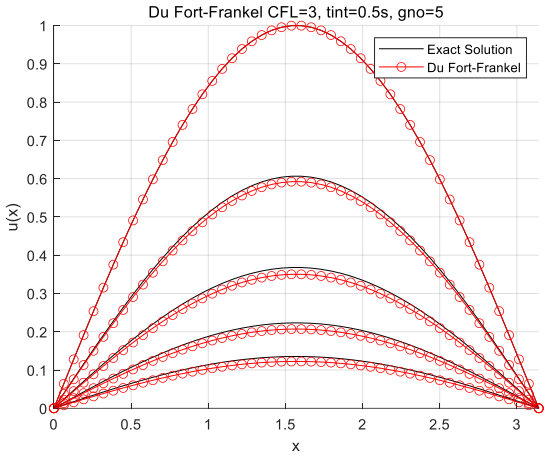
< Fig. 15 – Du Fort-Frankel CFL=0.7 >



< Fig. 16– Du Fort-Frankel CFL=1.0 >



< Fig. 17 – Du Fort-Frankel CFL=2.0 >



< Fig. 18 – Du Fort-Frankel CFL=3.0 >

위 그래프 Fig. 13에서 Fig. 18을 보면 CFL 값과 관계없이 해석값이 수렴하는 것을 확인할 수 있다. Fig. 13, Fig. 14, Fig. 15를 보면, CFL=0.3, 0.5, 0.7 일 때의 그래프는 그래프 상으로 확인하기 어려울 정도로 오차가 작게 나타났으며 Fig. 16, Fig. 17, Fig. 18을 보면, CFL=1.0, 2.0, 3.0 일 때에는 그래프 상으로도 오차를 확인할 수 있었다. 5번째 그래프의 25번째 격자를 기준으로 오차를 계산해보니 CFL=0.3, 0.5, 0.7, 1.0, 2.0, 3.0일 때 오차는 각각 0.219%, 0.087%, 0.603%, 1.348%, 3.826%, 9.581%로 CFL=0.5 일 때 가장 오차가 적게 나왔고 CFL=0.5 에서 멀어질수록 오차가 커지는 것을 확인할 수 있었다. 따라서 CFL=0.5 일 때 가장 정확한 해답을 가지며, CFL 값과 관계없이 수렴한 해를 가지는 Unconditionally Stable임을 알 수 있다.

3. Implicit Method

3.1. Laasonen Method

Laasonene Method는 음해법으로 시간에 대해 후방 차분법(Backward Time)을 사용하고 공간에 대

해서는 중심 차분법(Central Space)을 사용하는 방법으로 FTCS Method에서 우변의 시간을 n에서 n+1로 변경한 것과 같다. 식은 다음과 같이 나타난다.

$$\frac{u_i^{n+1}-u_i^n}{\Delta t}=\alpha\frac{u_{i+1}^{n+1}-2u_i^{n+1}+u_{i-1}^{n+1}}{(\Delta x)^2}$$

위 식에서 미지수만 좌변 남기도록 이항하면 다음과 같다.

$$\frac{\alpha\Delta t}{(\Delta x)^2}u_{i-1}^{n+1}-\left[1+2\frac{a\Delta t}{(\Delta x)^2}\right]u_i^{n+1}+\frac{\alpha\Delta t}{(\Delta x)^2}u_{i+1}^{n+1}=-u_i^n$$

위 식에서 계수를 간단하게 나타내면 다음과 같다.

$$b_i^n u_{i-1}^{n+1}+d_i^n u_i^{n+1}+a_i^n u_{i+1}^{n+1}=c_i^n$$

$$b_i^n=\frac{\alpha\Delta t}{(\Delta x)^2},\qquad d_i^n=-\left[1+2\frac{a\Delta t}{(\Delta x)^2}\right],\qquad a_i^n=\frac{\alpha\Delta t}{(\Delta x)^2},\qquad c_i^n=-u_i^n$$

미지수가 여러 개 존재하는 식이므로 행렬로 나타내어 구하면 다음과 같다.

$$\begin{bmatrix} d_2^n & a_2^n & 0 & 0 & 0 & 0 & 0 & 0\\ b_3^n & d_3^n & a_3^n & 0 & 0 & 0 & 0 & 0\\ 0 & b_4^n & d_4^n & a_4^n & 0 & 0 & 0 & 0\\ 0 & 0 & \ddots & \ddots & \ddots & 0 & 0 & 0\\ 0 & 0 & 0 & \ddots & \ddots & \ddots & 0 & 0\\ 0 & 0 & 0 & 0 & b_{imax-3}^n & d_{imax-3}^n & a_{imax-3}^n & 0\\ 0 & 0 & 0 & 0 & 0 & b_{imax-2}^n & d_{imax-2}^n & a_{imax-2}^n\\ 0 & 0 & 0 & 0 & 0 & 0 & b_{imax-1}^n & d_{imax-1}^n \end{bmatrix}\begin{bmatrix} u_2^{n+1}\\ u_3^{n+1}\\ u_4^{n+1}\\ \vdots\\ \vdots\\ u_{imax-3}^{n+1}\\ u_{imax-2}^{n+1}\\ u_{imax-1}^{n+1} \end{bmatrix}=\begin{bmatrix} c_2^n-b_2^n u_1^{n+1}\\ c_3^n\\ c_4^n\\ \vdots\\ \vdots\\ c_{imax-3}^n\\ c_{imax-2}^n\\ c_{imax-1}^n-a_{imax-1}^n u_{imax-1}^{n+1} \end{bmatrix}$$

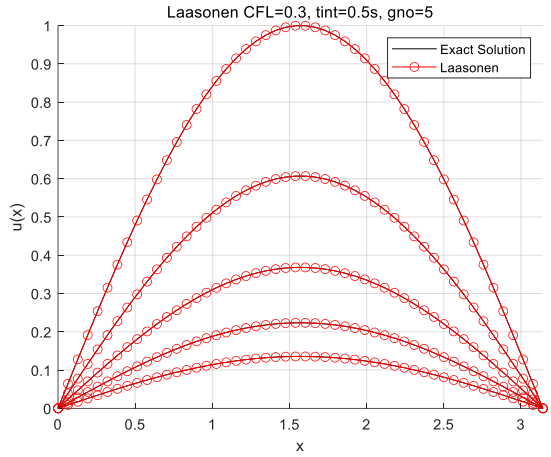
작성한 MATLAB CODE는 다음과 같다.

```
%% Laasonen Method
clear; close all;
%% input
prompt = {'gridn', 'gno', 'tint', 'CFL'};
dlgtitle = 'Data'; dims = [1 35];
definput = {'50','5', '0.5', '0.5'}; % set default value
inputdata = inputdlg(prompt,dlgtitle,dims,definput); % input data
dialogue
gridn = str2double(inputdata{1}); % no. of grid
gno = str2double(inputdata{2}); % no. of graph
tint = str2double(inputdata{3}); % time interval between graphs
CFL = str2double(inputdata{4}); % CFL number
A=1; % peak value (amplitude)
a=1; % alpha
wid=pi; % width of wave
dx=wid/(gridn-1); % width of grid dx
dt=CFL*(dx)^2/a; % time interval between u
t=0:dt:tint*(gno-1); % n with dt interval
x=linspace(0,wid,gridn);
```

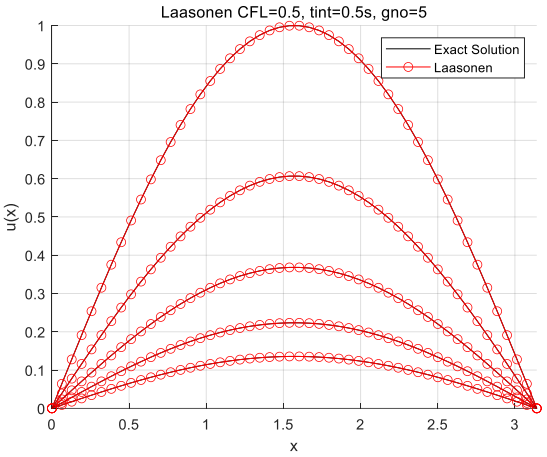
```
no=length(t); % max number of n to calculate
%% initial condition
for i=1:1:gridn
    ue(i,1)=A*sin((i-1)*dx);
end
%% boundary condition
for n=1:1:no
    ue(1,n)=0;
    ue(gridn,n)=0;
end
%% exact solution
for n=2:1:no+2
    for i=2:1:gridn-1
        ue(i,n)=A*exp(-a*dt*(n-1))*sin(dx*(i-1));
    end
end
for n=1:1:gno
    uee(:,n)=ue(:,1+(n-1)*round(tint/dt));
end
%% Laasonen method
c=-ue;
b=CFL; d=-(1+2*CFL); a=CFL;
z1=linspace(0,0,gridn-3);
z2=(cat(2,z1,[0]))';
A=diag(linspace(a,a,gridn-3));
A=cat(1,A,z1); A=cat(2,z2,A);
B=diag(linspace(b,b,gridn-3));
B=cat(1,z1,B); B=cat(2,B,z2);
D=diag(linspace(d,d,gridn-2));
AA=A+B+D;
for n=2:1:no+1
    for i=2:1:gridn-3
        CC(i,n)=c(i+1,n);
    end
    CC(1,n)=c(2,n); % b*u(1,n+1)=0
    CC(gridn-2,n)=c(gridn-1,n); % a*u(gridn,n+1)=0
end
u=AA\CC;
u=cat(1,linspace(0,0,length(u)),u,linspace(0,0,length(u)));
u=cat(2,ue(:,1),u);
for n=1:1:gno
    ui(:,n)=u(:,1+(n-1)*round(tint/dt));
end
%% plot
figure(1)
hold on; grid on;
for n=1:1:gno
    plot(x,uee(:,n),'k') % exact solution
    plot(x,ui(:,n),'-or') % Laasonen
end
xlim([0,pi])
title(['Laasonen CFL=',num2str(CFL),', tint=',num2str(tint),'s,
gno=',num2str(gno)])
xlabel('x')
ylabel('u(x)')
legend({'Exact Solution','Laasonen'})
ee=(uee(gridn/2,gno)-ui(gridn/2,gno))/uee(gridn/2,gno)*100 % error(middle
grid, last graph)
```

< Table. 4 – Laasonen Method MATLAB CODE >

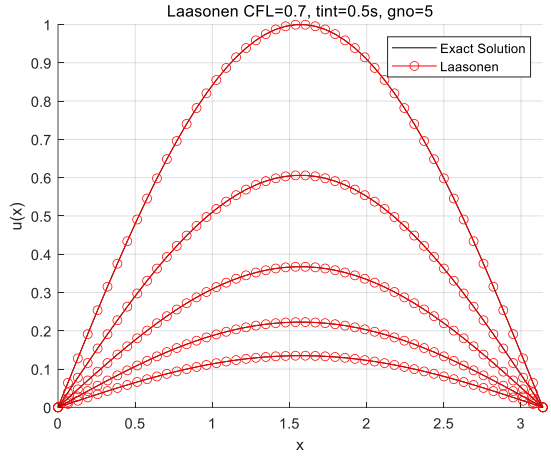
다음 Fig. 19 ~ Fig. 24는 그래프의 시간 간격을 $tint = 0.5s$, 격자의 개수 $gridn = 50$으로 설정한 후 CFL 값을 $0.3, 0.5, 0.7, 1.0, 2.0, 3.0$으로 변화시켜 Du Fort-Frankel Method로 해석한 결과이다.



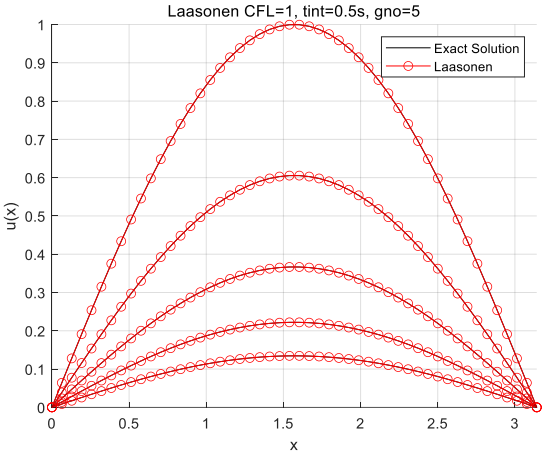
< Fig. 19 – Laasonen CFL=0.3 >



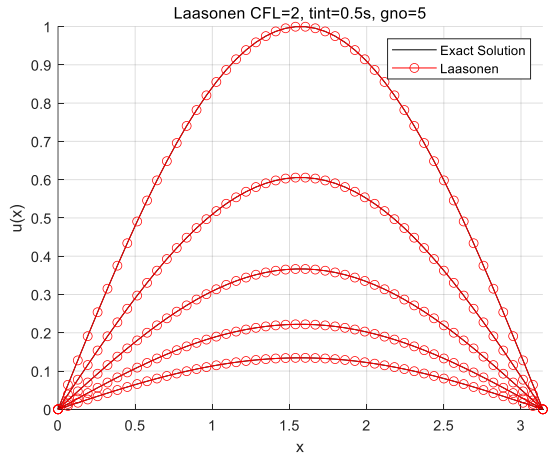
< Fig. 20 – Laasonen CFL=0.5 >

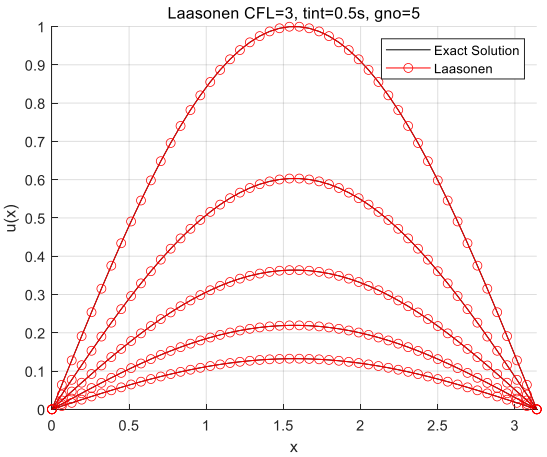


< Fig. 21 – Laasonen CFL=0.7 >



< Fig. 22– Laasonen CFL=1.0 >





< Fig. 23 – Laasonen CFL=2.0 >

< Fig. 24 – Laasonen CFL=3.0 >

Fig. 19에서 Fig. 24를 보면 CFL=0.3 부터 CFL=3.0 까지 CFL 값과 관계없이 해석값이 수렴하는 것을 확인할 수 있다. 또한 위 그래프 모두에서 그래프 상으로 오차를 확인하기 어려웠으며, 오차값은 CFL=0.3, 0.5, 0.7, 1.0, 2.0, 3.0 에서 각각 0.000118%, 0.000281%, 0.000511%, 0.000983%, 0.0036%, 0.0080%의 오차값을 가지며 CFL 값이 낮아질수록 오차가 적은 것을 확인할 수 있었다. 따라서 Laasonen Method는 Unconditionally Stable하며, CFL=1.0 이하에서 0.001% 이하의 오차를 가지며 매우 정확한 해석법임을 알 수 있다.

3.2. Crank-Nicolson Method

Crank-Nicolson Method는 가장 많이 사용하는 음해법(Implicit method)으로 항상 수렴된 해(Unconditionally Stable)를 가지는 것이 특징이다. 2차 정확도를 가지며 음해법과 양해법을 모두 이용하는 방법이다. Time step의 절반은 양해법을 사용하고 절반은 음해법을 사용하는 것이 특징이다. 두 식은 다음과 같다.

$$\frac{u_i^{n+\frac{1}{2}} - u_i^n}{\frac{\Delta t}{2}} = \alpha \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{(\Delta x)^2}$$

$$\frac{u_i^{n+1} - u_i^{n+\frac{1}{2}}}{\frac{\Delta t}{2}} = \alpha \frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}}{(\Delta x)^2}$$

두 식을 정리하여 미지수만 좌변으로 정리하면 다음과 같다.

$$-\frac{\alpha\Delta t}{2(\Delta x)^2}u_{i-1}^{n+1} + \left(1 + \frac{\alpha\Delta t}{(\Delta x)^2}\right)u_i^{n+1} - \frac{\alpha\Delta t}{2(\Delta x)^2}u_{i+1}^{n+1} = u_i^n - \frac{\alpha\Delta t}{2(\Delta x)^2}(u_{i-1}^n - 2u_i^n + u_{i+1}^n)$$

3.1절과 같이 위 식에서 계수를 간단하게 정리하면 다음과 같다.

$$b_i^n u_{i-1}^{n+1} + d_i^n u_i^{n+1} + a_i^n u_{i+1}^{n+1} = c_i^n$$

$$b_i^n = -\frac{\alpha\Delta t}{2(\Delta x)^2}, \quad d_i^n = \left(1 + \frac{\alpha\Delta t}{(\Delta x)^2}\right), \quad a_i^n = -\frac{\alpha\Delta t}{2(\Delta x)^2}, \quad c_i^n = u_i^n - \frac{\alpha\Delta t}{2(\Delta x)^2}(u_{i-1}^n - 2u_i^n + u_{i+1}^n)$$

위 식을 행렬로 나타내면 다음과 같다.

$$\begin{bmatrix} d_2^n & a_2^n & 0 & 0 & 0 & 0 & 0 & 0 \\ b_3^n & d_3^n & a_3^n & 0 & 0 & 0 & 0 & 0 \\ 0 & b_4^n & d_4^n & a_4^n & 0 & 0 & 0 & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & 0 & 0 & 0 \\ 0 & 0 & 0 & \ddots & \ddots & \ddots & 0 & 0 \\ 0 & 0 & 0 & 0 & b_{imax-3}^n & d_{imax-3}^n & a_{imax-3}^n & 0 \\ 0 & 0 & 0 & 0 & 0 & b_{imax-2}^n & d_{imax-2}^n & a_{imax-2}^n \\ 0 & 0 & 0 & 0 & 0 & 0 & b_{imax-1}^n & d_{imax-1}^n \end{bmatrix} \begin{bmatrix} u_2^{n+1} \\ u_3^{n+1} \\ u_4^{n+1} \\ \vdots \\ \vdots \\ u_{imax-3}^{n+1} \\ u_{imax-2}^{n+1} \\ u_{imax-1}^{n+1} \end{bmatrix} = \begin{bmatrix} c_2^n - b_2^n u_1^{n+1} \\ c_3^n \\ c_4^n \\ \vdots \\ \vdots \\ c_{imax-3}^n \\ c_{imax-2}^n \\ c_{imax-1}^n - a_{imax-1}^n u_{imax-1}^{n+1} \end{bmatrix}$$

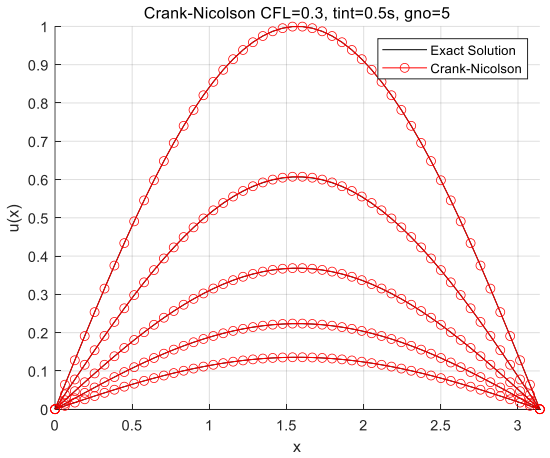
작성한 MATLAB CODE는 다음과 같다.

```
%% Crank-Nicolson Method
clear; close all;
%% input
prompt = {'gridn', 'gno', 'tint', 'CFL'};
dlgtitle = 'Data'; dims = [1 35];
definput = {'50','5', '0.5', '0.5'}; % set default value
inputdata = inputdlg(prompt,dlgtitle,dims,definput); % input data
dialogue
gridn = str2double(inputdata{1}); % no. of grid
gno = str2double(inputdata{2}); % no. of graph
tint = str2double(inputdata{3}); % time interval between graphs
CFL = str2double(inputdata{4}); % CFL number
A=1; % peak value (amplitude)
a=1; % alpha
wid=pi; % width of wave
dx=wid/(gridn-1); % width of grid dx
dt=CFL*(dx)^2/a; % time interval between u
t=0:dt:tint*(gno-1); % n with dt interval
x=linspace(0,wid,gridn);
no=length(t); % max number of n to calculate
%% initial condition
for i=1:1:gridn
    ue(i,1)=A*sin((i-1)*dx);
end
%% boundary condition
for n=1:1:no
    ue(1,n)=0;
    ue(gridn,n)=0;
end
%% exact solution
for n=2:1:no+2
    for i=2:1:gridn-1
        ue(i,n)=A*exp(-a*dt*(n-1))*sin(dx*(i-1));
    end
end
for n=1:1:gno
    uee(:,n)=ue(:,1+(n-1)*round(tint/dt));
end
%% Crank-Nicolson method
c=-ue;
for n=2:1:no+2
    for i=2:1:gridn-1
        c(i,n)=ue(i,n)-CFL/2*(ue(i-1,n)-2*ue(i,n)+ue(i+1,n));
    end
end
b=-CFL/2; d=1+CFL; a=-CFL/2;
z1=linspace(0,0,gridn-3);
z2=(cat(2,z1,[0]))';
A=diag(linspace(a,a,gridn-3));
A=cat(1,A,z1); A=cat(2,z2,A);
B=diag(linspace(b,b,gridn-3));
B=cat(1,z1,B); B=cat(2,B,z2);
D=diag(linspace(d,d,gridn-2));
AA=A+B+D;
for n=2:1:no+1
```

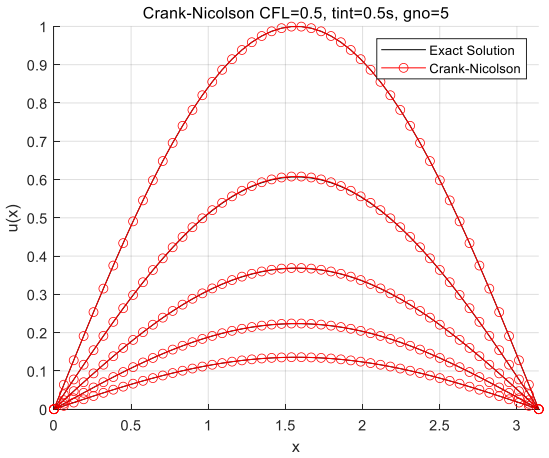
```
    for i=2:1:gridn-3
        CC(i,n)=c(i+1,n);
    end
    CC(1,n)=c(2,n); % b*u(1,n+1)=0
    CC(gridn-2,n)=c(gridn-1,n); % a*u(gridn,n+1)=0
end
u=AA\CC;
u=cat(1,linspace(0,0,length(u)),u,linspace(0,0,length(u)));
u=cat(2,ue(:,1),u);
for n=1:1:gno
    ui(:,n)=u(:,1+(n-1)*round(tint/dt));
end
%% plot
figure(1)
hold on; grid on;
for n=1:1:gno
    plot(x,uee(:,n),'k') % exact solution
    plot(x,ui(:,n),'-or') % Crank-Nicolson
end
xlim([0,pi])
title(['Crank-Nicolson CFL=',num2str(CFL),', tint=',num2str(tint),'s,
gno=',num2str(gno)])
xlabel('x')
ylabel('u(x)')
legend({'Exact Solution','Crank-Nicolson'})
ee=(uee(gridn/2,gno)-ui(gridn/2,gno))/uee(gridn/2,gno)*100 % error(middle
grid, last graph)
```

< Table. 5 – Crank-Nicolson Method MATLAB CODE >

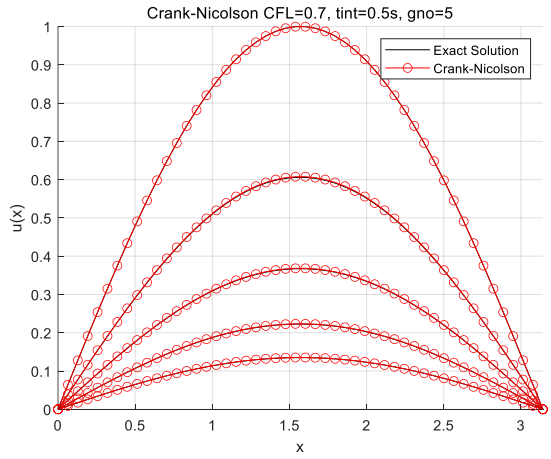
다음 Fig. 25 ~ Fig. 30은 그래프의 시간 간격을 $tint = 0.5s$, 격자의 개수 $gridn = 50$으로 설정한 후 CFL 값을 $0.3, 0.5, 0.7, 1.0, 2.0, 3.0$으로 변화시켜 Crank-Nicolson Method로 해석한 결과이다.



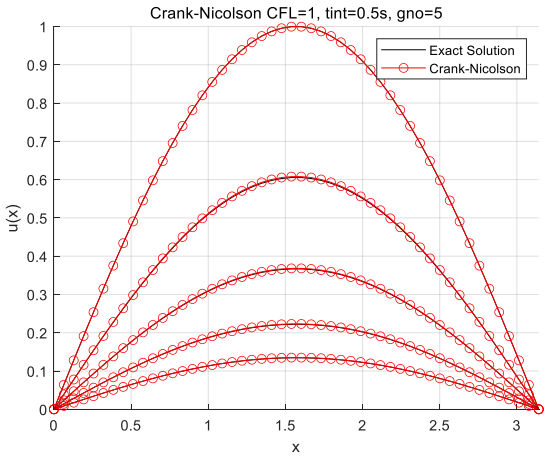
< Fig. 25 – C-N CFL=0.3 >



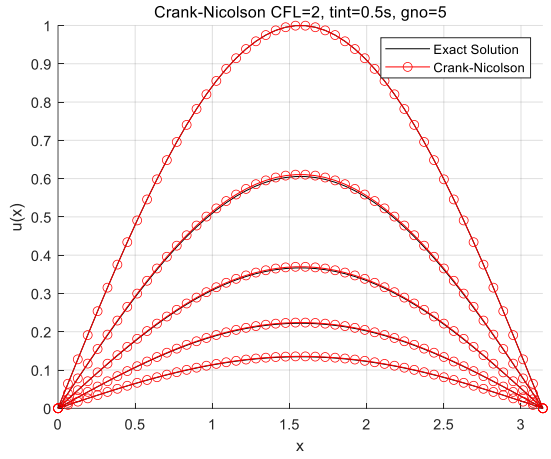
< Fig. 26 – C-N CFL=0.5 >



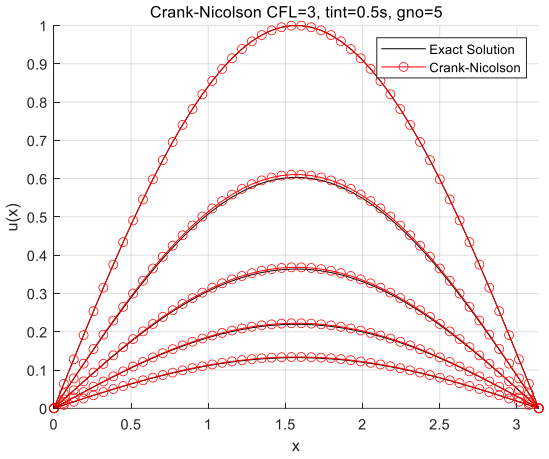
< Fig. 27 – C-N CFL=0.7 >



< Fig. 28– C-N CFL=1.0 >



< Fig. 29 – C-N CFL=2.0 >



< Fig. 30 – C-N CFL=3.0 >

Fig. 25에서 Fig. 30을 보면 CFL=0.3 부터 CFL=3.0 까지 CFL 값과 관계없이 해석값이 수렴하는 것을 확인할 수 있다. 또한 위 그래프 모두에서 그래프 상으로 오차를 확인하기 어려웠으며, 오차값은 CFL=0.3, 0.5, 0.7, 1.0, 2.0, 3.0에서 각각 0.123%, 0.206%, 0.288%, 0.412%, 0.826%, 1.241%로 CFL이 낮아질수록 오차가 작은 것을 확인할 수 있다. 따라서 Crank-Nicolson Method는 Unconditionally Stable하며, CFL 값이 낮을수록 정확한 해석값을 가지는 것을 알 수 있다.

Laasonen Method와 Crank-Nicolson Method와 비교해 보면 두 방법 모두 수렴하는 결과를 얻을 수 있으며, CFL 값이 낮을 때 더 정확한 값을 가지는 것을 확인할 수 있었다. Laasonen Method의 결과가 Crank-Nicolson Method의 결과보다 더 정확한 값을 가지는 것 또한 확인할 수 있었다.

3.3 Beta Formulation

Beta Formulation은 β값에 따라 Implicit Method와 Explicit Method의 특성을 변화시킬 수 있는 방법으로 식은 다음과 같다.

$$\frac{u_i^{n+1}-u_i^n}{\Delta t}=\alpha\left[\beta\frac{u_{i-1}^{n+1}-2u_i^{n+1}+u_{i+1}^{n+1}}{(\Delta x)^2}+(1-\beta)\frac{u_{i-1}^{n}-2u_i^{n}+u_{i+1}^{n}}{(\Delta x)^2}\right],\qquad 0\le\beta\le 1$$

위 식에서 β값이 증가할수록 Implicit Method의 성질을 가지며 $\beta=0$일 때 FTCS Method, $\beta=0.5$일 때 Crank-Nicolson Method, $\beta=1$일 때 Laasonen Method인 것을 알 수 있다.

위 식을 정리하면 다음과 같다.

$$\beta\frac{\alpha\Delta t}{(\Delta x)^2}u_{i-1}^{n+1}-\left(\beta\frac{2\alpha\Delta t}{(\Delta x)^2}+1\right)u_i^{n+1}+\beta\frac{\alpha\Delta t}{(\Delta x)^2}u_{i+1}^{n+1}=-u_i^n+(\beta-1)\frac{\alpha\Delta t}{(\Delta x)^2}(u_{i-1}^n-2u_i^n+u_{i+1}^n)$$

4. 결론

이번에는 대표적인 포물선 특성(Parabolic Type)의 편미분 방정식인 1차원 2차 비정상 열정도 방정식(1-D 2nd Order Unsteady Heat Conduction Equation)을 여러 수치해석 방법을 이용하여 해석하였다.

Explicit Method로는 FTCS Method와 Richardson Method, Du Fort-Frankel Method를 사용하였고, Implicit Method로는 Laasonen Method와 Crank-Nicolson Method를 사용하였다.

Explicit Method에서 FTCS Method는 Fig. 1에서 Fig. 6을 보면 CFL 값이 0.5이하일 때 수렴하는 해를 가지며 CFL 값이 0.5을 초과할 때 발산하는 것을 확인할 수 있고, Unconditionally Stable함을 알 수 있었고, Richardson Method는 Fig. 7에서 Fig. 12를 보면 CFL 값에 관계없이 수렴하는 해를 나타내지 못하며 Unconditionally Unstable함을 알 수 있다. Du Fort-Frankel Method에서는 Richardson Method에서 u_i^n 대신 u_i^{n+1}과 u_i^{n-1}의 평균값을 사용하는 방법으로, CFL 값과 관계없이 수렴하여 Unconditionally Stable함을 확인할 수 있었다.

Implicit Method인 Laasonen Method는 n time step의 i번째의 격자점과 n+1 time step의 i-1, i, i+1번째 격자점의 관계를 음함수로 나타낸 방법으로 CFL 값과 관계없이 수렴된 해를 가져 Unconditionally Stable하며, CFL 값이 작아질수록 더 정확한 해를 나타내었다. Crank-Nicolson Method는 n time step에서 n+0.5 time step을 먼저 양함수로 계산하고, n+0.5 time step에서 n+1 time step을 음함수로 계산하는 방법으로 해석 결과를 보면 Laasonene Method와 마찬가지로 CFL 값과 관계없이 수렴된 해를 가져 Unconditionally Stable하며, CFL 값이 작아질수록 더 정확한 해를 나타내었다. Laasonen Method와 Crank-Nicolson Method의 해석값을 비교해 본 결과 Laasonen Method에서 exact solution과의 오차가 더 적은 것을 확인할 수 있었다.