전산유체역학

PJ#4: Elliptic Type

항공우주 및 기계공학부

담당교수 : 김문상 교수님

제출일 : 2021-06-08

2016121150 윤준영

1. 서론

타원형 특성(Elliptic Type) 편미분 방정식인 Laplace's Equation, 정상 열전도 방정식(Steady-State Heat Conduction Equation), 속도 포텐셜 방정식(Velocity Potential Equation), 유량함수 방정식(Stream Function Equation) 등의 해를 구할 때 풀이되며 이들은 물리적으로 주변과 영향을 주고받는 성질이 있기 때문에 주변의 모든 위치의 함수 값들과 정보를 공유하면서 균형(Equilibrium)을 이루는 수치 기법을 사용해야 한다. 대표적인 타원형 특성(Elliptic Type)의 편미분 방정식인 정상 2차원 열정도 방정식(Steady-State 2-D Heat Conduction Equation)을 여러 수치해석 방법을 이용하여 해석하였다.

(*) Steady-State 2-D Heat Conduction Equation

$$\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}=0$$

Elliptic Type의 PDE를 해석하는 방법에는 크게 Direct Method와 Iterative Method가 있다. Direct Method는 미지수(unknown)을 구하기 위해 미지수들의 관계식으로 미지수의 개수만큼의 방정식을 만들어 행렬을 이용하여 한번에 풀이하는 방법이고, Iterative Method는 임의의 Initial Condition을 적용하여 여러 횟수에 걸쳐 해를 풀이한 뒤 앞서 지정한 수렴조건에 만족하면 반복을 중단하고 수렴조건에 만족하지 못하면 다시 반복하는 방법이다.

Steady-State 2-D Heat Conduction Equation은 경계 조건(Boundary Condition)이 필요하며 Iterative Method의 경우에는 임의의 초기 조건(Initial Condition)이 추가로 필요하다. 주어진 Boundary Condition과 Initial Condition은 다음과 같다.

(1) Boundary Condition:

$$T_{i,1}=T_1=100°R, \qquad T_{i,j_{max}}=T_3=0°R$$

$$T_{1,j}=T_2=0°R, \qquad T_{i_{max},j}=T_4=0°R$$

(2) Initial Condition:

$$T_{i,j}=0, \qquad for\ i\neq 1,i_{max}\ and\ j\neq 1,j_{max}$$

(3) Exact Solution:

$$T_{i,j}=T_1\left[2\sum_{n=1}^{\infty}\frac{1-(-1)^n}{n\pi}\frac{\sinh\left(\frac{n\pi(H-y)}{L}\right)}{\sinh\left(\frac{n\pi H}{L}\right)}\sin\left(\frac{n\pi x}{L}\right)\right]$$

여기서 L, H는 각각 물체의 x방향 길이와, y방향 길이이다.

아래에서 서술할 식과 코드에서 하첨자 i, j는 격자점의 위치를 의미하고 상첨자는 반복횟수(iteration)를 의미한다. k는 이전에 계산된 이전 반복횟수(previous iteration), $k+1$은 새로 계산하는 현재 반복횟수(present iteration)을 의미한다. $L=1$, $H=2$이며 기본 격자의 개수는 $i_{max}=21$, $j_{max}=41$로 설정하였다. 따라서, $\Delta x=\Delta y=0.05$이다. β와 α는 다음과 같이 정의한다.

$$\beta=\frac{\Delta x}{\Delta y}=1, \qquad \alpha=-2(1+\beta^2)=-4$$

2. Direct Method

2.1. Five-Point Formula

Five-Point Formula는 가장 널리 사용하는 PDE 해석법으로 구하려는 격자점 $u_{i,j}$를 주위 4개의 격자점 $u_{i-1,j}, u_{i+1,j}, u_{i,j-1}, u_{i,j+1}$을 이용하여 구하는 방법으로, 2차 공간 정확도를 가지며, 식은 다음과 같다.

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{(\Delta y)^2} = 0$$

위 식을 정리하고 $\frac{\Delta x}{\Delta y} = \beta, -2(1+\beta^2) = \alpha$로 놓으면 다음과 같이 표현할 수 있다.

$$u_{i+1,j} + u_{i-1,j} + \beta^2 u_{i,j+1} + \beta^2 u_{i,j-1} + \alpha u_{i,j} = 0$$

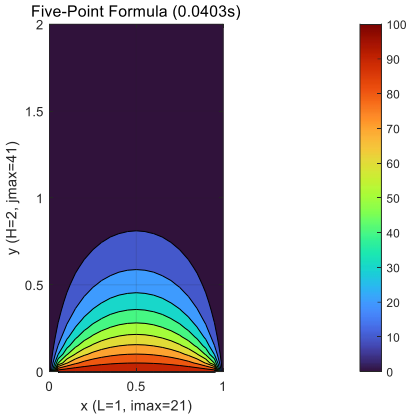
작성한 MATLAB CODE는 다음과 같다.

```
%% Five-Point Formula
%% Data
clear;
imax = 21; jmax = 41; L = 1; H = 2; % number of grid & object length
dx = L/(imax-1); dy = H/(jmax-1); % grid width
T1 = 100; T2 = 0; T3 = 0; T4 = 0; % boundary temp
b = dx/dy; a = -2*(1+b^2);
k = 1;
%% Initial Condition & Boundary Condition
T(imax,jmax,1) = 0; % IC
T(:,1,1) = T1; T(1,:,1) = T2; T(:,jmax,1) = T3; T(imax,:,1) = T4; % BC
%% Five-Point Formula
tic % stopwatch start
m = (imax-2)*(jmax-2);
ab = b^2*diag(ones(1,m-imax+2),imax-2); % above beta diag
bb = b^2*diag(ones(1,m-imax+2),2-imax); % below beta diag
d = diag(a*ones(1,m)); % diagonal
oz(imax-2,jmax-2) = 0; % initialize array of 1 & 0
oz(1:imax-3,:) = ones((imax-3),(jmax-2));
oz = reshape(oz,1,m); % array of 1 & 0
a1 = diag(oz(1:m-1),1); % above 1 & 0
b1 = diag(oz(1:m-1),-1); % below 1& 0
A = d+ab+bb+a1+b1; % known matrix A
C(m)=0; % initialize known matrix C
for l = 1:1:(jmax-2)
    C((imax-2)*(l-1)+1) = C((imax-2)*(l-1)+1)-T(1,l+1);
    C((imax-2)*l-1) = C((imax-2)*l-1)-T(imax,l+1);
    if l == 1 % first imax-2 array
        C(1:imax-2) = C(1:imax-2)-b^2*T(2:imax-1,1)';
    elseif l == jmax-2 % last imax-2 array
        C((imax-2)*(jmax-3)+1:m) = C((imax-2)*(jmax-3)+1:m)-b^2*T(2:imax-
1,jmax)';
    end
end
T(2:imax-1,2:jmax-1) = reshape((A\C')',[(imax-2),(jmax-2)]); % get
unknowns
time = toc % stopwatch stop
%% Plot
contourf(T(:,:,1)); % contour graph
colorbar; colormap('turbo'); view(90,270);
axis equal; hold on; grid on;
xticks([1 0.25*(jmax-1)+1 0.5*(jmax-1)+1 0.75*(jmax-1)+1 jmax])
```

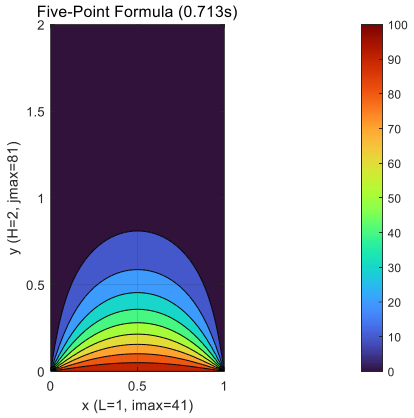
```
xticklabels({'0','0.5','1','1.5','2'})
yticks([1 0.5*(imax-1)+1 imax])
yticklabels({'0','0.5','1'})
xlabel(['y (H=' num2str(H) ', jmax=' num2str(jmax) ')'])
ylabel(['x (L=' num2str(L) ', imax=' num2str(imax) ')'])
title(['Five-Point Formula (',num2str(time,3) 's)'],'FontSize',12)
```

〈 Table. 1 - Five-Point Formula MATLAB CODE 〉

다음은 $L = 1$, $H = 2$에서 $i_{max} = 21$, $j_{max} = 41$일 때와 $i_{max} = 41$, $j_{max} = 81$일 때의 Five-Point Formula로 해석한 결과이다.



〈 Fig. 1 - Five-Point Formula grid(21,41) 〉



〈 Fig. 2 - Five-Point Formula grid(41,81) 〉

Five-Point Formula로 해석한 결과, $i_{max} = 21$, $j_{max} = 41$일 때에는 0.0403초, $i_{max} = 41$, $j_{max} = 81$일 때에는 0.713초의 연산 시간이 소요되었다. 격자의 개수가 x 방향으로 2배, y 방향으로 2배가 늘었을 때, 총 격자의 개수는 약 4배가 늘었음에도 계산 시간은 훨씬 더 많은 약 18배의 차이를 보여준다. 이는 해를 한번에 구하기 위해 역행렬을 계산할 때의 Fig. 1의 경우 741x741의 역행렬을 구해야 하고, Fig. 2의 경우 3081x3081의 역행렬을 구해야 하기 때문이다.

3. Iterative Method

Iterative Method는 Five-Point Formula의 식을 이용하되 한번에 행렬을 이용하여 식을 구한 것과는 달리 격자점의 값을 하나씩 구하는 방법이다. 모든 격자점의 값을 계산한 뒤 한 iteration이 끝나면, 초기에 설정한 수렴조건을 만족하는지 확인하여 만족하지 못하면 새로운 iteration을 시작하고, 만족한다면 반복계산을 중단한다. 또한 오랫동안 수렴조건을 만족시키지 못할 경우 최대 iteration을 지정해 계산을 중단할 수 있다. 아래 방법에서 사용한 수렴조건과 최대 iteration 조건은 다음과 같다.

(1) Tolerence

$$\frac{\sum_{i=2}^{i_{max}-1}\sum_{j=2}^{j_{max}-1}\left|T_{i,j}^{k+1} - T_{i,j}^{k}\right|}{(i_{max} \times j_{max})} \leq \epsilon\ (Tolerence)$$

(2) 최대 iteration

$$k_{max} = 5000$$

3.1. Point Jacobi Iteration Method

Point Jacobi Iteration Method에서는 이전 k 번째(previous iteration) 값을 이용하여 $(k+1)$ 번째(present iteration)의 값을 구하는 방법으로 첫번째 계산을 위해서 임의의 초기조건이 필요하며, 설정한 수렴조건을 만족하면 반복계산을 중단하는 방법이다. Five-Point Formula의 식을 이용하여 k번째의 $u_{i+1,j}$, $u_{i-1,j}$, $u_{i,j+1}$, $u_{i,j-1}$을 이용하여 $(k+1)$번째의 $u_{i,j}$을 구한다.

$$u_{i+1,j}^{k} + u_{i-1,j}^{k} + \beta^2 u_{i,j+1}^{k} + \beta^2 u_{i,j-1}^{k} - 2(1+\beta^2)u_{i,j}^{k+1} = 0$$

위 식을 $u_{i,j}^{k+1}$에 대해 정리하면 다음과 같다.

$$u_{i,j}^{k+1} = \frac{1}{2(1+\beta^2)}\left[u_{i+1,j}^{k} + u_{i-1,j}^{k} + \beta^2\left(u_{i,j+1}^{k} + u_{i,j-1}^{k}\right)\right]$$

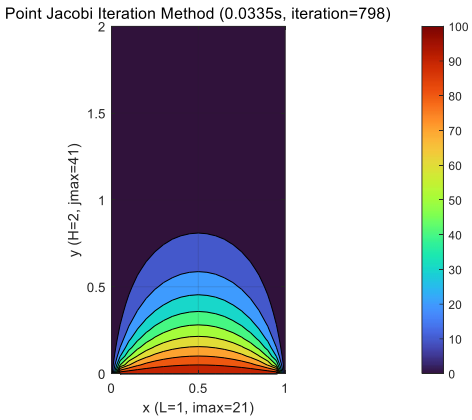
작성한 MATLAB CODE는 다음과 같다.

```
%% Point Jacobi Iteration Method
%% Data
clear;
imax = 21; jmax = 41; L = 1; H = 2; % number of grid & object length
tol = 0.0001; kmax = 5000; % tolerance & max iteration
dx = L/(imax-1); dy = H/(jmax-1); % grid width
T1 = 100; T2 = 0; T3 = 0; T4 = 0; % boundary temp
b = dx/dy; a = -2*(1+b^2);
%% Initial Condition & Boundary Condition
T(imax,jmax,1) = 0; % IC
T(:,1,1) = T1; T(1,:,1) = T2; T(:,jmax,1) = T3; T(imax,:,1) = T4; % BC
%% Point Jacobi Iteration Method
tic; % stopwatch start
for k=1:1:kmax-1
    T(:,1,k+1) = T1; T(1,:,k+1) = T2; T(:,jmax,k+1) = T3; T(imax,:,k+1) =
T4;
    for i=2:1:imax-1
        for j=2:1:jmax-1
            T(i,j,k+1) = 1/(-a)*(T(i+1,j,k)+T(i-
1,j,k)+b^2*(T(i,j+1,k)+T(i,j-1,k)));
        end
    end
    dif = abs(sum(T(2:imax-1,2:jmax-1,k+1)-T(2:imax-1,2:jmax-
1,k),'all'));
    if dif/(imax*jmax)<=tol % tolerence check
        k = k+1;
        break
    end
    k = k+1;
end
time = toc % stopwatch stop
iteration = k
%% Plot
contourf(T(:,:,k)); % contour graph
colorbar; colormap('turbo'); view(90,270);
axis equal; hold on; grid on;
xticks([1 0.25*(jmax-1)+1 0.5*(jmax-1)+1 0.75*(jmax-1)+1 jmax])
xticklabels({'0','0.5','1','1.5','2'})
yticks([1 0.5*(imax-1)+1 imax])
yticklabels({'0','0.5','1'})
xlabel(['y (H=' num2str(H) ', jmax=' num2str(jmax) ')'])
ylabel(['x (L=' num2str(L) ', imax=' num2str(imax) ')'])
```

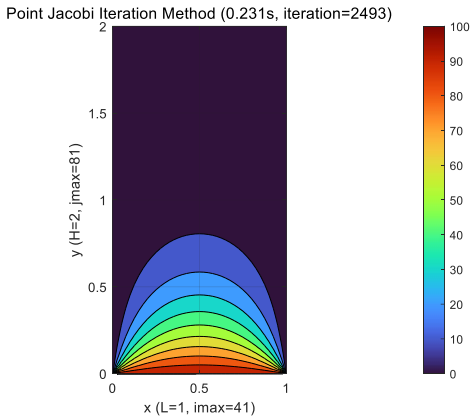
```
title(['Point Jacobi Iteration Method (',num2str(time,3) 's, iteration='
num2str(k) ')'],'FontSize',12)
```

〈 Table. 2 - Point Jacobi Iteration Method MATLAB CODE 〉

다음은 $L = 1$, $H = 2$에서 $i_{max} = 21$, $j_{max} = 41$일 때와 $i_{max} = 41$, $j_{max} = 81$일 때의 Point Jacobi Iteration Method로 해석한 결과이다.



〈 Fig. 3 - Point Jacobi Iteration grid(21,41) 〉



〈 Fig. 4 - Point Jacobi Iteration grid(41,81) 〉

Point Jacobi Iteration Method로 해석한 결과, $i_{max} = 21$, $j_{max} = 41$일 때에는 0.0335초, 798 iteration이 사용되었고, $i_{max} = 41$, $j_{max} = 81$일 때에는 0.231초, 2493 iteration이 수행되었다. 총 격자의 개수가 약 4배가 늘었을 때 계산 시간은 6.90배, 계산 횟수는 3.12배 더 많아졌다.

3.2. Point Gauss-Seidel Iteration Method

Point Gauss-Seidel Iteration Method은 격자점의 값을 계산할 때에 가장 최신의 값을 사용하는 방법으로, Point Jacobi Iteration Method에서 이전 계산에서 구한 $u_{i-1,j}^{k+1}$, $u_{i,j-1}^{k+1}$의 값을 $u_{i-1,j}^{k}$, $u_{i,j-1}^{k}$ 대신 사용하는 것이다. Jacobi Iteration Method보다 수렴속도가 100% 향상한다고 알려져 있으며, 식은 다음과 같다.

$$u_{i+1,j}^{k} + u_{i-1,j}^{k+1} + \beta^2 u_{i,j+1}^{k} + \beta^2 u_{i,j-1}^{k+1} - 2(1+\beta^2)u_{i,j}^{k+1} = 0$$

위 식을 $u_{i,j}^{k+1}$에 대해 정리하면 다음과 같다.

$$u_{i,j}^{k+1} = \frac{1}{2(1+\beta^2)}\left[u_{i+1,j}^{k} + u_{i-1,j}^{k+1} + \beta^2\left(u_{i,j+1}^{k} + u_{i,j-1}^{k+1}\right)\right]$$

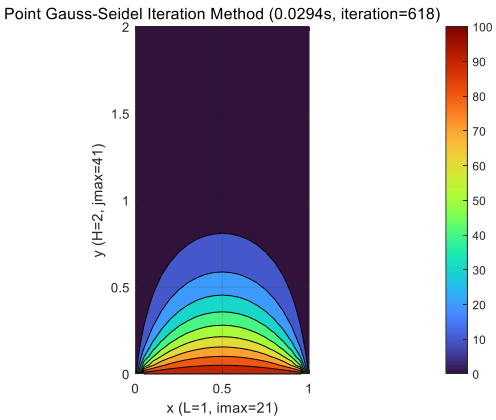
작성한 MATLAB CODE는 다음과 같다.

```
%% Point Gauss-Seidel Iteration Method
%% Data
clear;
imax = 21; jmax = 41; L = 1; H = 2; % number of grid & object length
tol = 0.0001; kmax = 5000; % tolerance & max iteration
dx = L/(imax-1); dy = H/(jmax-1); % grid width
T1 = 100; T2 = 0; T3 = 0; T4 = 0; % boundary temp
b = dx/dy; a = -2*(1+b^2);
%% Initial Condition & Boundary Condition
T(imax,jmax,1) = 0; % IC
T(:,1,1) = T1; T(1,:,1) = T2; T(:,jmax,1) = T3; T(imax,:,1) = T4; % BC
%% Point Gauss-Seidel Iteration Method
```

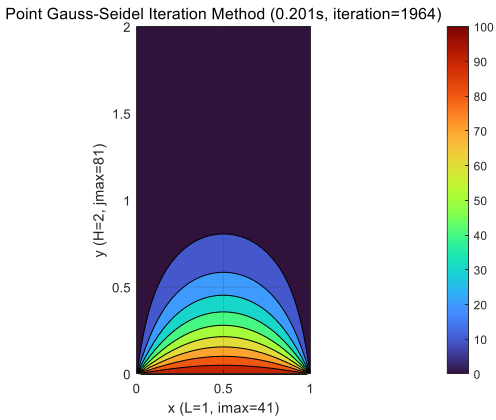
```
tic; % stopwatch start
for k=1:1:kmax-1
    T(:,1,k+1) = T1; T(1,:,k+1) = T2; T(:,jmax,k+1) = T3; T(imax,:,k+1) =
T4;
    for i=2:1:imax-1
        for j=2:1:jmax-1
            T(i,j,k+1) = 1/(-a)*(T(i+1,j,k)+T(i-
1,j,k)+b^2*(T(i,j+1,k)+T(i,j-1,k+1)));
        end
    end
    dif = abs(sum(T(2:imax-1,2:jmax-1,k+1)-T(2:imax-1,2:jmax-
1,k),'all'));
    if dif/(imax*jmax)<=tol % tolerence check
        k = k+1;
        break
    end
    k = k+1;
end
time = toc % stopwatch stop
iteration = k
%% Plot
contourf(T(:,:,k)); % contour graph
colorbar; colormap('turbo'); view(90,270);
axis equal; hold on; grid on;
xticks([1 0.25*(jmax-1)+1 0.5*(jmax-1)+1 0.75*(jmax-1)+1 jmax])
xticklabels({'0','0.5','1','1.5','2'})
yticks([1 0.5*(imax-1)+1 imax]); yticklabels({'0','0.5','1'})
xlabel(['y (H=' num2str(H) ', jmax=' num2str(jmax) ')'])
ylabel(['x (L=' num2str(L) ', imax=' num2str(imax) ')'])
title(['Point Gauss-Seidel Iteration Method (',num2str(time,3) 's,
iteration=' num2str(k) ')'],'FontSize',12)
```

〈 Table. 3 - Point Gauss-Seidel Iteration Method MATLAB CODE 〉

다음은 $L = 1$, $H = 2$에서 $i_{max} = 21$, $j_{max} = 41$일 때와 $i_{max} = 41$, $j_{max} = 81$일 때의 Point Gauss-Seidel Iteration Method로 해석한 결과이다.



〈 Fig. 5 - Point G-S Iteration grid(21,41) 〉



〈 Fig. 6 - Point G-S Iteration grid(41,81) 〉

Point Gauss-Seidel Iteration Method로 해석한 결과, $i_{max} = 21$, $j_{max} = 41$일 때에는 0.0294초, 618 iteration이 사용되었고, $i_{max} = 41$, $j_{max} = 81$일 때에는 0.201초, 1964 iteration이 수행되었다. 총 격자의 개수가 약 4배가 늘었을 때 계산 시간은 6.84배, 계산 횟수는 3.18배 더 많아졌다. Point Gauss-Seidel Iteration Method가 Point Jacobi Iteration Method와 비교해보면 계산 시간은 12.2%, 계산 횟수는 22.6% 감소한 것을 확인할 수 있었다.

3.3 Line Jacobi Iteration Method

Line Jacobi Iteration Method는 Point Jacobi Iteration Method에서는 한 격자점마다 계산한 것과는 달리 같은 미지수를 여러 개 가지는 연립방정식을 이용하여 i나 j열의 격자점을 한번에 계산하는 방법으로, j열을 한번에 계산하는 경우 $(k+1)$번째 $u_{i-1,j}$, $u_{i,j}$, $u_{i+1,j}$을 계산할 때, 이전 iteration에서 얻어진 k번째 $u_{i,j-1}$, $u_{i,j+1}$의 값을 사용하는 방법이다. 식은 다음과 같다.

$$u_{i-1,j}^{k+1} - 2(1+\beta^2)u_{i,j}^{k+1} + u_{i+1,j}^{k+1} = -\beta^2(u_{i,j+1}^{k} + u_{i,j-1}^{k})$$

j열을 한번에 계산하는 경우 식은 다음과 같다.

$$\beta^2 u_{i,j+1}^{k+1} - 2(1+\beta^2)u_{i,j}^{k+1} + \beta^2 u_{i,j-1}^{k+1} = -(u_{i+1,j}^{k} + u_{i-1,j}^{k})$$

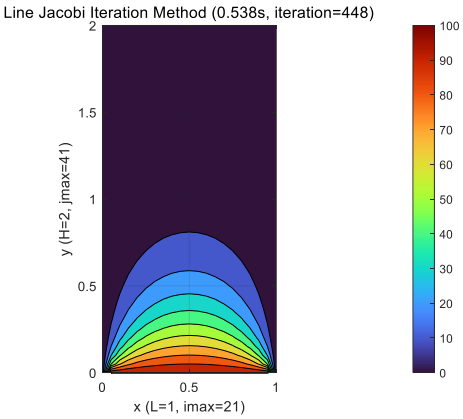
작성한 MATLAB CODE는 다음과 같다.

```
%% Line Jacobi Iteration Method
%% Data
clear;
imax = 21; jmax = 41; L = 1; H = 2; % number of grid & object length
tol = 0.0001; kmax = 5000; % tolerance & max iteration
dx = L/(imax-1); dy = H/(jmax-1); % grid width
T1 = 100; T2 = 0; T3 = 0; T4 = 0; % boundary temp
b = dx/dy; a = -2*(1+b^2);
%% Initial Condition & Boundary Condition
T(imax,jmax,1) = 0; % IC
T(:,1,1) = T1; T(1,:,1) = T2; T(:,jmax,1) = T3; T(imax,:,1) = T4; % BC
%% Line Jacobi Method
tic; % stopwatch start
m = (imax-2);
d = diag(a*ones(1,m)); % diagonal
oz = ones(1,m-1); % array of 1
a1 = diag(oz,1); b1 = diag(oz,-1); % above & below 1
A = d+a1+b1; % known matrix A
for k=1:1:kmax-1
    T(:,1,k+1) = T1; T(1,:,k+1) = T2; T(:,jmax,k+1) = T3; T(imax,:,k+1) =
T4;
    for j=2:1:jmax-1
        C(m) = 0; % initialize known matrix C
        for i=2:1:imax-1
            C(i-1) = -b^2*(T(i,j+1,k)+T(i,j-1,k));
        end
        T(2:imax-1,j,k+1) = reshape((A\C')',[(imax-2),1]);
    end
    dif = abs(sum(T(2:imax-1,2:jmax-1,k+1)-T(2:imax-1,2:jmax-
1,k),'all'));
    if dif/(imax*jmax)<=tol % tolerence check
        k = k+1;
        break
    end
    k = k+1;
end
time = toc % stopwatch stop
iteration = k
%% Plot
contourf(T(:,:,k)); % contour graph
colorbar; colormap('turbo'); view(90,270);
axis equal; hold on; grid on;
xticks([1 0.25*(jmax-1)+1 0.5*(jmax-1)+1 0.75*(jmax-1)+1 jmax])
```

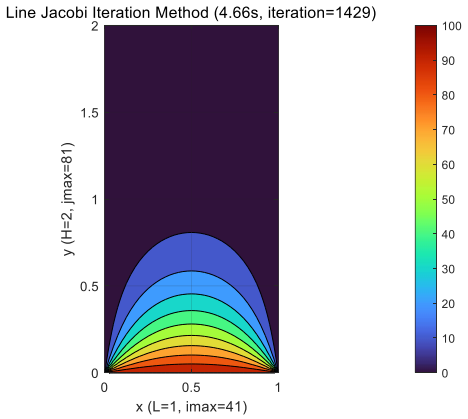
```
xticklabels({'0','0.5','1','1.5','2'})
yticks([1 0.5*(imax-1)+1 imax])
yticklabels({'0','0.5','1'})
xlabel(['y (H=' num2str(H) ', jmax=' num2str(jmax) ')'])
ylabel(['x (L=' num2str(L) ', imax=' num2str(imax) ')'])
title(['Line Jacobi Iteration Method (',num2str(time,3) 's, iteration='
num2str(k) ')'],'FontSize',12)
```

〈 Table. 4 - Line Jacobi Iteration Method MATLAB CODE 〉

다음은 $L = 1$, $H = 2$에서 $i_{max} = 21$, $j_{max} = 41$일 때와 $i_{max} = 41$, $j_{max} = 81$일 때의 Line Jacobi Iteration Method로 해석한 결과이다.



〈 Fig. 7 - Line Jacobi Iteration grid(21,41) 〉



〈 Fig. 8 - Line Jacobi Iteration grid(41,81) 〉

Line Jacobi Iteration Method로 해석한 결과, $i_{max} = 21$, $j_{max} = 41$일 때에는 0.538초, 448 iteration이 사용되었고, $i_{max} = 41$, $j_{max} = 81$일 때에는 4.66초, 1429 iteration이 수행되었다. 총 격자의 개수가 약 4배가 늘었을 때 계산 시간은 8.66배, 계산 횟수는 3.19배 더 많아졌다. Point Jacobi Iteration Method와 비교하면 계산 시간은 약 16.06배 증가하였지만, iteration 횟수는 43.9% 줄었지만 Line별로 계산할 때 역행렬을 구하는 과정에서 계산이 많이 필요하므로 계산 횟수 자체는 더 늘었다고 봐야한다.

3.4 Line Gauss-Seidel Iteration Method

Line Gauss-Seidel Iteration Method는 Line Jacobi Iteration Method의 식에서 같은 iteration 내의 이전 Line에서 계산한 최신의 값을 사용하는 방법으로, j열을 한번에 계산하는 경우 $u_{i-1,j}$, $u_{i,j}$, $u_{i+1,j}$과 더불어 $u_{i,j-1}$의 값 또한 이전 line 계산에서 얻어진 $(k+1)$번째 값을 사용하는 방법이다. 식은 다음과 같다.

$$u_{i-1,j}^{k+1} - 2(1+\beta^2)u_{i,j}^{k+1} + u_{i+1,j}^{k+1} = -\beta^2(u_{i,j+1}^{k} + u_{i,j-1}^{k+1})$$

j열을 한번에 계산하는 경우 식은 다음과 같다.

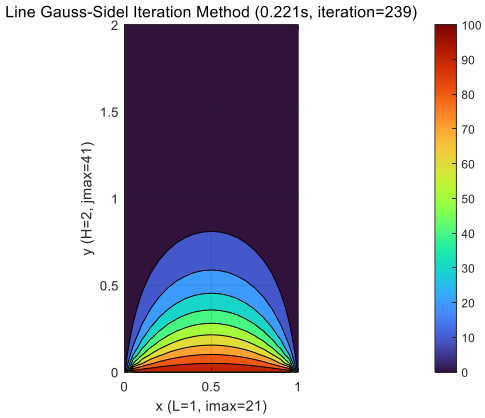
$$\beta^2 u_{i,j+1}^{k+1} - 2(1+\beta^2)u_{i,j}^{k+1} + \beta^2 u_{i,j-1}^{k+1} = -(u_{i+1,j}^{k} + u_{i-1,j}^{k+1})$$

작성한 MATLAB CODE는 다음과 같다.

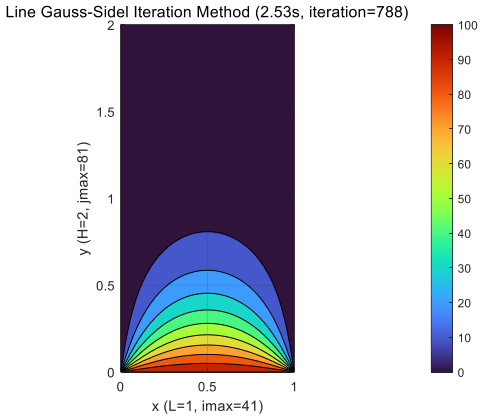
```
%% Line Gauss-Seidel Iteration Method
%% Data
clear;
imax = 21; jmax = 41; L = 1; H = 2; % number of grid & object length
tol = 0.0001; kmax = 5000; % tolerance & max iteration
dx = L/(imax-1); dy = H/(jmax-1); % grid width
T1 = 100; T2 = 0; T3 = 0; T4 = 0; % boundary temp
b = dx/dy; a = -2*(1+b^2);
%% Initial Condition & Boundary Condition
T(imax,jmax,1) = 0; % IC
T(:,1,1) = T1; T(1,:,1) = T2; T(:,jmax,1) = T3; T(imax,:,1) = T4; % BC
%% Line Gauss-Seidel Iteration Method
tic; % stopwatch start
m = (imax-2);
d = diag(a*ones(1,m)); % diagonal
oz = ones(1,m-1); % array of 1
a1 = diag(oz,1); b1 = diag(oz,-1); % above & below 1
A = d+a1+b1; % known matrix A
for k=1:1:kmax-1
    T(:,1,k+1) = T1; T(1,:,k+1) = T2; T(:,jmax,k+1) = T3; T(imax,:,k+1) =
T4;
    for j=2:1:jmax-1
        C(m) = 0; % initialize known matrix C
        for i=2:1:imax-1
            C(i-1) = -b^2*(T(i,j+1,k)+T(i,j-1,k+1));
        end
        T(2:imax-1,j,k+1) = reshape((A\C')',[(imax-2),1]);
    end
    dif = abs(sum(T(2:imax-1,2:jmax-1,k+1)-T(2:imax-1,2:jmax-
1,k),'all'));
    if dif/(imax*jmax)<=tol % tolerence check
        k = k+1;
        break
    end
    k = k+1;
end
time = toc % stopwatch stop
iteration = k
%% Plot
contourf(T(:,:,k)); % contour graph
colorbar; colormap('turbo'); view(90,270);
axis equal; hold on; grid on;
xticks([1 0.25*(jmax-1)+1 0.5*(jmax-1)+1 0.75*(jmax-1)+1 jmax])
xticklabels({'0','0.5','1','1.5','2'})
yticks([1 0.5*(imax-1)+1 imax])
yticklabels({'0','0.5','1'})
xlabel(['y (H=' num2str(H) ', jmax=' num2str(jmax) ')'])
ylabel(['x (L=' num2str(L) ', imax=' num2str(imax) ')'])
title(['Line Gauss-Sidel Iteration Method (',num2str(time,3) 's,
iteration=' num2str(k) ')'],'FontSize',12)
```

〈 Table. 5 - Line Gauss-Seidel Iteration Method MATLAB CODE 〉

다음은 $L = 1$, $H = 2$에서 $i_{max} = 21$, $j_{max} = 41$일 때와 $i_{max} = 41$, $j_{max} = 81$일 때의 Line Gauss-Seidel Iteration Method로 해석한 결과이다.



〈 Fig. 9 - Line G-S Iteration grid(21,41) 〉



〈 Fig. 10 - Line G-S Iteration grid(41,81) 〉

Line Gauss-Seidel Iteration Method로 해석한 결과, $i_{max} = 21$, $j_{max} = 41$일 때에는 0.221초, 239 iteration이 사용되었고, $i_{max} = 41$, $j_{max} = 81$일 때에는 2.53초, 788 iteration이 수행되었다. 총 격자의 개수가 약 4배가 늘었을 때 계산 시간은 11.45배, 계산 횟수는 3.30배 더 많아졌다. Line Gauss-Seidel Iteration Method가 Point Gauss-Seidel Iteration Method와 비교해보면 계산 시간은 7.52배 증가하였고, 계산 횟수는 61.3% 감소한 것을 확인할 수 있었다. Line Gauss-Seidel Iteration Method가 Line Jacobi Iteration Method와 비교해보면 계산 시간은 58.9%, 계산 횟수는 46.7% 감소한 것을 확인할 수 있었다.

따라서 Line Iteration Method는 Point Method과 비교하면 계산 시간은 증가하지만 iteration이 감소하는 효과가 있고, Line Jacobi Iteration Method보다 Line Gauss-Seidel Iteration Method가 계산 속도가 빠르고 iteration 횟수가 적어 효율적이다.

3.5 Point Successive Over-Relaxation Method(PSOR Method)

Point Successive Over-Relaxation Method는 위에서 살펴본 Point Gauss-Seidel Iteration Method에서 변수 ω를 사용하여 수렴속도를 조절하기 위한 방법으로, 수렴되는 방향으로 풀이가 진행될 경우, 새로운 iteration에서의 해를 구할 때 수렴속도를 가속시키는 역할을 한다. 식은 다음과 같다.

$$u_{i,j}^{k+1} = (1-\omega)u_{i,j}^{k} + \frac{\omega}{2(1+\beta^2)}\left[u_{i+1,j}^{k} + u_{i-1,j}^{k+1} + \beta^2\left(u_{i,j+1}^{k} + u_{i,j-1}^{k+1}\right)\right]$$

ω의 크기에 따라, $0 < \omega < 1$의 경우 under-relaxation, $1 < \omega < 2$의 경우 over-relaxation, $\omega = 1$일 경우 Point Gauss-Seidel Iteration Method가 된다.

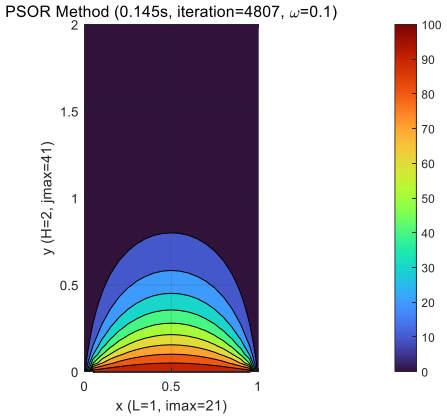
작성한 MATLAB CODE는 다음과 같다.

```
%% Point Successive Over-Relaxation Method
%% Data
clear;
imax = 21; jmax = 41; L = 1; H = 2; % number of grid & object length
tol = 0.0001; kmax = 5000; % tolerance & max iteration
w = 1.5; % relaxation
```

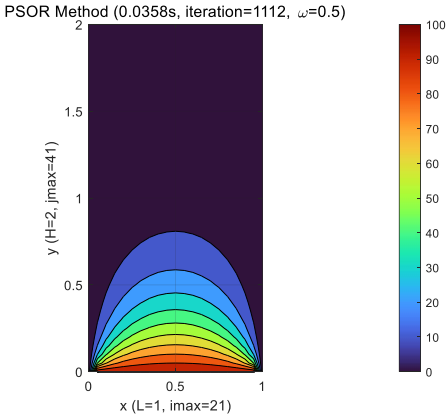
```
dx = L/(imax-1); dy = H/(jmax-1); % grid width
T1 = 100; T2 = 0; T3 = 0; T4 = 0; % boundary temp
b = dx/dy; a = -2*(1+b^2);
%% Initial Condition & Boundary Condition
T(imax,jmax,1) = 0; % IC
T(:,1,1) = T1; T(1,:,1) = T2; T(:,jmax,1) = T3; T(imax,:,1) = T4; % BC
%% PSOR Method
tic; % stopwatch start
for k=1:1:kmax-1
    T(:,1,k+1) = T1; T(1,:,k+1) = T2; T(:,jmax,k+1) = T3; T(imax,:,k+1) =
T4; % BC at k+1
    for i=2:1:imax-1
        for j=2:1:jmax-1
            T(i,j,k+1) = (1-w)*T(i,j,k)+w/(-a)*(T(i+1,j,k)+T(i-
1,j,k+1)+b^2*(T(i,j+1,k)+T(i,j-1,k+1)));
        end
    end
    dif = abs(sum(T(2:imax-1,2:jmax-1,k+1)-T(2:imax-1,2:jmax-
1,k),'all'));
    if dif/(imax*jmax)<=tol % tolerence check
        k = k+1;
        break
    end
    k = k+1;
end
time = toc % stopwatch stop
iteration = k
%% Plot
contourf(T(:,:,k)); % contour graph
colorbar; colormap('turbo'); view(90,270);
axis equal; hold on; grid on;
xticks([1 0.25*(jmax-1)+1 0.5*(jmax-1)+1 0.75*(jmax-1)+1 jmax])
xticklabels({'0','0.5','1','1.5','2'})
yticks([1 0.5*(imax-1)+1 imax])
yticklabels({'0','0.5','1'})
xlabel(['y (H=' num2str(H) ', jmax=' num2str(jmax) ')'])
ylabel(['x (L=' num2str(L) ', imax=' num2str(imax) ')'])
title(['PSOR Method (',num2str(time,3) 's, iteration=' num2str(k) ',
\omega=' num2str(w) ')'],'FontSize',12)
```

〈 Table. 6 - PSOR Method MATLAB CODE 〉

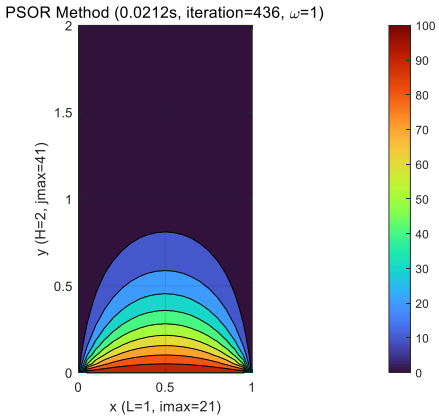
다음은 $L = 1,\ H = 2,\ i_{max} = 21,\ j_{max} = 41$에서 $\omega = 0.1, 0.5, 1.0, 1.5$일 때 Point Successive Over-Relaxation Method로 해석한 결과이다.



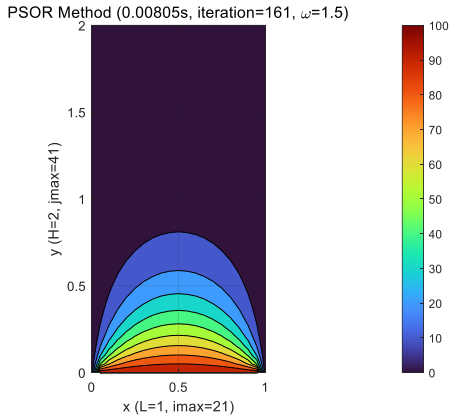
〈 Fig. 11 - PSOR Method $\omega = 0.1$ 〉



〈 Fig. 12 - PSOR Method $\omega = 0.5$ 〉



〈 Fig. 13 - PSOR Method $\omega = 1.0$ 〉



〈 Fig. 14 - PSOR Method $\omega = 1.5$ 〉

Point Successive Over-Relaxation Method로 해석한 결과, $\omega = 0.1$일 때에는 0.145초, 4807 iteration이 수행되었고, $\omega = 0.5$일 때에는 0.0358초, 1112 iteration, $\omega = 1.0$일 때에는 0.0212초, 436 iteration, $\omega = 1.5$일 때에는 0.00805초, 161 iteration이 수행되었다. ω값이 감소할수록 수렴되는 속도가 느려져 계산 시간이 늘어나고 iteration 횟수가 증가하며, ω값이 증가할수록 수렴되는 속도가 빨라져 계산 시간이 감소하고, iteration 횟수도 감소하는 것을 확인할 수 있다.

3.6 Line Successive Over-Relaxation Method (LSOR Method)

Line Successive Over-Relaxation Method 또한 위에서 살펴본 Line Gauss-Seidel Iteration Method에서 변수 ω를 사용하여 수렴속도를 조절하기 위한 방법으로, 수렴되는 방향으로 풀이가 진행될 경우, 새로운 iteration에서의 해를 구할 때 수렴속도를 가속시키는 역할을 한다. j열을 한번에 계산하는 경우 식은 다음과 같다.

$$\omega u_{i-1,j}^{k+1} - 2(1+\beta^2)u_{i,j}^{k+1} + \omega u_{i+1,j}^{k+1} = -(1-\omega)[2(1+\beta^2)]u_{i,j}^{k} - \omega\beta^2\left(u_{i,j+1}^{k} + u_{i,j-1}^{k+1}\right)$$

i열을 한번에 계산하는 경우 식은 다음과 같다.

$$\omega\beta^2 u_{i,j+1}^{k+1} - 2(1+\beta^2)u_{i,j}^{k+1} + \omega\beta^2 u_{i,j-1}^{k+1} = -(1-\omega)[2(1+\beta^2)]u_{i,j}^{k} - \omega\left(u_{i+1,j}^{k} + u_{i-1,j}^{k+1}\right)$$

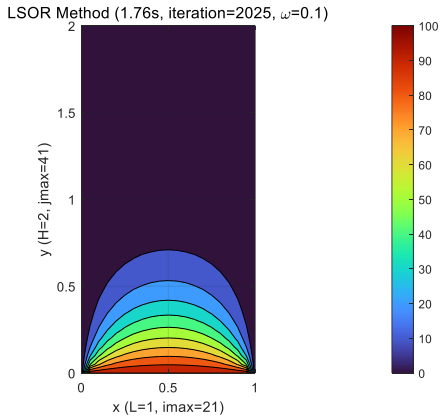
작성한 MATLAB CODE는 다음과 같다.

```
%% Line Successive Over-Relaxation Method
%% Data
clear;
imax = 21; jmax = 41; L = 1; H = 2; % number of grid & object length
tol = 0.001; kmax = 5000; % tolerance & max iteration
w = 1; % relaxation
dx = L/(imax-1); dy = H/(jmax-1); % grid width
T1 = 100; T2 = 0; T3 = 0; T4 = 0; % boundary temp
b = dx/dy; a = -2*(1+b^2);
%% Initial Condition & Boundary Condition
T(imax,jmax,1) = 0; % IC
T(:,1,1) = T1; T(1,:,1) = T2; T(:,jmax,1) = T3; T(imax,:,1) = T4; % BC
%% LSOR Method
tic;
m = (imax-2);
d = diag(a*ones(1,m)); % diagonal
oz = ones(1,m-1); % array of 1
```

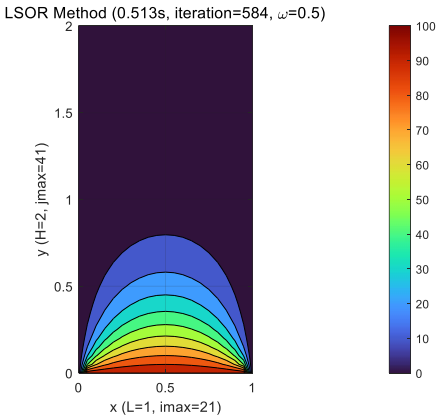
```
a1 = diag(w*oz,1); b1 = diag(w*oz,-1); % above & below w
A = d+a1+b1; % known matrix A
for k=1:1:kmax-1
    T(:,1,k+1) = T1; T(1,:,k+1) = T2; T(:,jmax,k+1) = T3; T(imax,:,k+1) =
T4; % BC at k+1
    for j=2:1:jmax-1
        C(m) = 0; % initialize known matrix C
        for i=2:1:imax-1
            C(i-1) = a*(1-w)*T(i,j,k)-w*b^2*(T(i,j+1,k)+T(i,j-1,k+1));
        end
        C(1) = C(1)-w*T(i-1,j,k+1);
        C(imax-2) = C(imax-2)-w*T(imax,j,k+1);
        T(2:imax-1,j,k+1) = reshape((A\C')',[(imax-2),1]);
    end
    dif = abs(sum(T(2:imax-1,2:jmax-1,k+1)-T(2:imax-1,2:jmax-
1,k),'all'));
    if dif/(imax*jmax)<=tol % tolerence check
        k = k+1;
        break
    end
    k = k+1;
end
time = toc
iteration = k
%% Plot
contourf(T(:,:,k)); % contour graph
colorbar; colormap('turbo'); view(90,270);
axis equal; hold on; grid on;
xticks([1 0.25*(jmax-1)+1 0.5*(jmax-1)+1 0.75*(jmax-1)+1 jmax])
xticklabels({'0','0.5','1','1.5','2'})
yticks([1 0.5*(imax-1)+1 imax])
yticklabels({'0','0.5','1'})
xlabel(['y (H=' num2str(H) ', jmax=' num2str(jmax) ')'])
ylabel(['x (L=' num2str(L) ', imax=' num2str(imax) ')'])
title(['LSOR Method (',num2str(time,3) 's, iteration=' num2str(k) ',
\omega=' num2str(w) ')'],'FontSize',12)
```

〈 Table. 7 - LSOR Method MATLAB CODE 〉

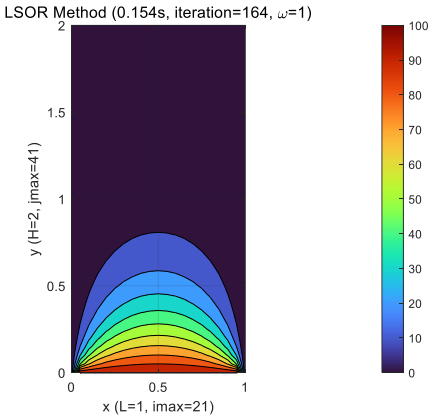
다음은 $L = 1,\ H = 2,\ i_{max} = 21,\ j_{max} = 41$에서 $\omega = 0.1, 0.5, 1.0, 1.34$일 때 Line Successive Over-Relaxation Method로 해석한 결과이다.



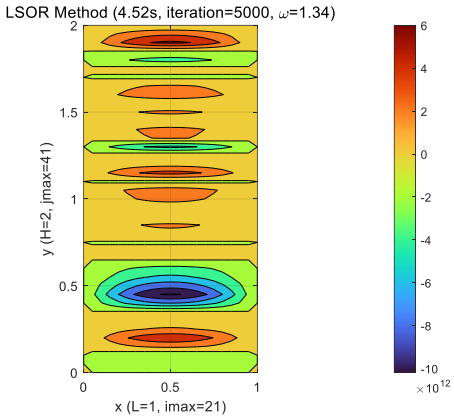
〈 Fig. 15 - LSOR Method $\omega = 0.1$ 〉



〈 Fig. 16 - LSOR Method $\omega = 0.5$ 〉



〈 Fig. 17 - LSOR Method $\omega = 1.0$ 〉



〈 Fig. 18 - LSOR Method $\omega = 1.34$ 〉

Line Successive Over-Relaxation Method로 해석한 결과, $\omega = 0.1$일 때에는 1.76초, 2025 iteration이 수행되었고, $\omega = 0.5$일 때에는 0.513초, 584 iteration, $\omega = 1.0$일 때에는 0.154초, 164 iteration이 수행되었다. $\omega = 1.33$을 넘어가면 해가 발산하여 수렴조건을 만족하지 못하게 되고 최대 iteration인 5000에서 멈추게 된다. 따라서 Line Successive Over-Relaxation Method 또한 PSOR Method와 마찬가지로 ω값이 증가할수록 수렴되는 속도가 빨라져 계산 시간이 감소하고, iteration 횟수도 감소하는 것을 확인할 수 있었지만 일정 ω가 넘어가면 발산하는 것을 확인할 수 있었다.

3.7 Alternating Direction Implicit Method(ADI Method)

Alternating Direction Implicit Method는 2-D Hyperbolic Type의 PDE 해석 시 많이 사용되는 방법으로, 2-D Elliptic Type PDE 해석 시 응용할 수 있다. Line Gauss-Seidel Iteration Method의 Line sweep 방법을 사용하여 한 sweep 당 $\frac{1}{2}$ iteration으로 한 iteration 당 두번의 sweep을 방향을 바꾸어 계산한다. 다음 식은 x-sweep 후 y-sweep의 계산식을 나타낸다.

$$u_{i-1,j}^{k+1/2} - 2(1+\beta^2)u_{i,j}^{k+1/2} + u_{i+1,j}^{k+1/2} = -\beta^2(u_{i,j+1}^{k} + u_{i,j-1}^{k+1/2})$$

$$\beta^2 u_{i,j-1}^{k+1} - 2(1+\beta^2)u_{i,j}^{k+1} + \beta^2 u_{i,j+1}^{k+1} = -(u_{i+1,j}^{k+1/2} + u_{i-1,j}^{k+1})$$

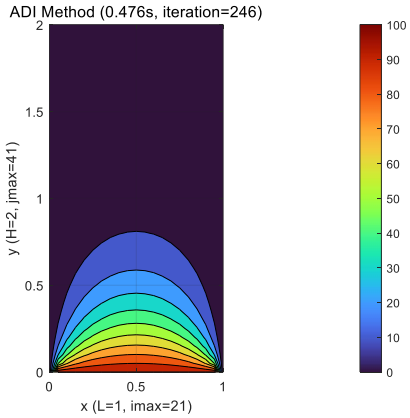
작성한 MATLAB CODE는 다음과 같다.

```
%% Alternating Direction Implicit Method
%% Data
clear;
imax = 21; jmax = 41; L = 1; H = 2; % number of grid & object length
tol = 0.0001; kmax = 5000; % tolerance & max iteration
dx = L/(imax-1); dy = H/(jmax-1); % grid width
T1 = 100; T2 = 0; T3 = 0; T4 = 0; % boundary temp
b = dx/dy; a = -2*(1+b^2);
%% Initial Condition & Boundary Condition
T(imax,jmax,1) = 0; % IC
T(:,1,1) = T1; T(1,:,1) = T2; T(:,jmax,1) = T3; T(imax,:,1) = T4; % BC
%% ADI Method
tic;
mx = (imax-2); my = (jmax-2);
dx = diag(a*ones(1,mx)); dy = diag(a*ones(1,my)); % diagonal
ozx = ones(1,mx-1); ozy = ones(1,my-1); % array of 1
a1x = diag(ozx,1); b1x = diag(ozx,-1);
```

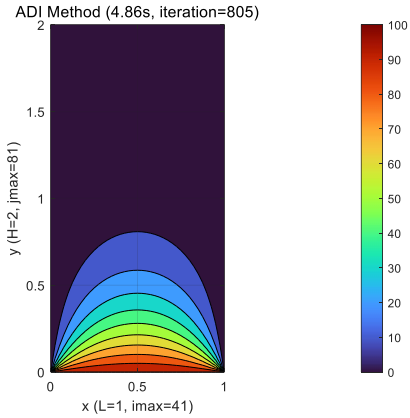
```
a1y = diag(ozy,1); b1y = diag(ozy,-1); % above & below w
Ax = dx+a1x+b1x; % known matrix A (x sweep)
Ay = dy+b^2*(a1y+b1y); % known matrix A (y sweep)
for k=1:1:kmax-1
    T(:,1,k+1) = T1; T(1,:,k+1) = T2; T(:,jmax,k+1) = T3; T(imax,:,k+1) =
T4; % BC at k+1/2
    for j=2:1:jmax-1 % known matrix C (x sweep)
        for i=2:1:imax-1
            Cx(i-1) = -b^2*(T(i,j+1,k)+T(i,j-1,k+1));
        end
        Cx(1) = Cx(1)-T(1,j,k+1);
        Cx(imax-2) = Cx(imax-2) -T(imax,j,k+1);
        T(2:1:imax-1,j,k+1) = (Ax\Cx')';
    end
    for i=2:1:imax-1 % known matrix C (y sweep)
        for j=2:1:jmax-1
            Cy(j-1) = -(T(i+1,j,k)+T(i-1,j,k+1));
        end
        Cy(1) = Cy(1)-b^2*T(i,1,k+1);
        Cy(jmax-2) = Cy(jmax-2)-b^2*T(i,jmax,k+1);
        T(i,2:1:jmax-1,k+1) = (Ay\Cy')';
    end
    dif = abs(sum(T(2:imax-1,2:jmax-1,k+1)-T(2:imax-1,2:jmax-
1,k),'all'));
    if dif/(imax*jmax)<=tol % tolerence check
        k = k+1;
        break
    end
    k = k+1;
end
time = toc
iteration = k
%% Plot
contourf(T(:,:,k)); % contour graph
colorbar; colormap('turbo'); view(90,270);
axis equal; hold on; grid on;
xticks([1 0.25*(jmax-1)+1 0.5*(jmax-1)+1 0.75*(jmax-1)+1 jmax])
xticklabels({'0','0.5','1','1.5','2'})
yticks([1 0.5*(imax-1)+1 imax])
yticklabels({'0','0.5','1'})
xlabel(['y (H=' num2str(H) ', jmax=' num2str(jmax) ')'])
ylabel(['x (L=' num2str(L) ', imax=' num2str(imax) ')'])
title(['ADI Method (',num2str(time,3) 's, iteration=' num2str(k)
')'],'FontSize',12)
```

〈 Table. 8 - ADI Method MATLAB CODE 〉

다음은 $L = 1$, $H = 2$에서 $i_{max} = 21$, $j_{max} = 41$일 때와 $i_{max} = 41$, $j_{max} = 81$일 때의 Alternating Direction Implicit Method로 해석한 결과이다.



〈 Fig. 19 - ADI Method grid(21,41) 〉



〈 Fig. 20 - ADI Method grid(41,81) 〉

Alternating Direction Implicit Method로 해석한 결과, $i_{max} = 21$, $j_{max} = 41$일 때에는 0.476초, 246 iteration이 사용되었고, $i_{max} = 41$, $j_{max} = 81$일 때에는 4.86초, 805 iteration이 수행되었다. 총 격자의 개수가 약 4배가 늘었을 때 계산 시간은 10.21배, 계산 횟수는 3.27배 더 많아졌다. Line Gauss-Seidel Iteration Method와 비교해보면 계산 시간은 2.17배 증가하였고, 계산 횟수는 1.03배 증가한 것을 확인할 수 있었다. 이는 한 iteration에서 x-sweep과 y-sweep이 일어나기 때문에 Line Gauss-Seidel Iteration Method보다 시간은 약 2배, iteration은 유사하게 가지는 것으로 해석할 수 있다.

3.8 Alternating Direction Implicit Over-Relaxation Method(ADIOR Method)

Alternating Direction Implicit Over-Relaxation Method는 Line Successive Over-Relaxation Method에 ADI 방법을 적용한 것으로 ADI Method보다 수렴속도를 가속시키기 위해 사용한다. x-sweep 후 y-sweep을 사용한 ADIOR의 식은 다음과 같다.

$$\omega u_{i-1,j}^{k+1/2} - 2(1+\beta^2)u_{i,j}^{k+1/2} + \omega u_{i+1,j}^{k+1/2} = -(1-\omega)[2(1+\beta^2)]u_{i,j}^{k} - \omega\beta^2(u_{i,j+1}^{k} + u_{i,j-1}^{k+1/2})$$

$$\omega\beta^2 u_{i,j-1}^{k+1} - 2(1+\beta^2)u_{i,j}^{k+1} + \omega\beta^2 u_{i,j+1}^{k+1} = -(1-\omega)[2(1+\beta^2)]u_{i,j}^{k+1/2} - \omega(u_{i-1,j}^{k+1/2} + u_{i+1,j}^{k+1/2})$$

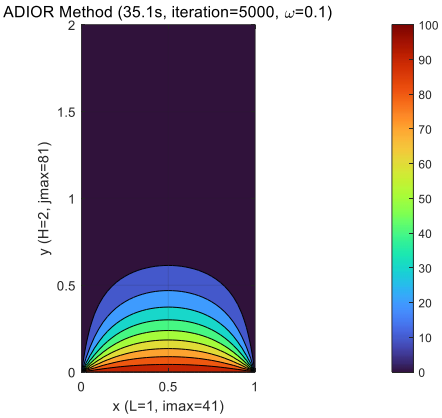
작성한 MATLAB CODE는 다음과 같다.

```
%% Alternating Direction Implicit Over-Relaxation Method
%% Data
clear;
imax = 41; jmax = 81; L = 1; H = 2; % number of grid & object length
tol = 0.0001; kmax = 5000; % tolerance & max iteration
w = 1; % relaxation
dx = L/(imax-1); dy = H/(jmax-1); % grid width
T1 = 100; T2 = 0; T3 = 0; T4 = 0; % boundary temp
b = dx/dy; a = -2*(1+b^2);
%% Initial Condition & Boundary Condition
T(imax,jmax,1) = 0; % IC
T(:,1,1) = T1; T(1,:,1) = T2; T(:,jmax,1) = T3; T(imax,:,1) = T4; % BC
%% ADIOR Method
tic;
mx = (imax-2); my = (jmax-2);
```

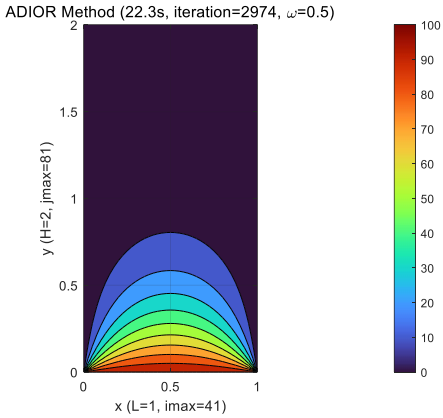
```
dx = diag(a*ones(1,mx)); dy = diag(a*ones(1,my)); % diagonal
ozx = w*ones(1,mx-1); ozy = w*ones(1,my-1); % array of 1
a1x = diag(ozx,1); b1x = diag(ozx,-1);
a1y = diag(ozy,1); b1y = diag(ozy,-1); % above & below w
Ax = dx+a1x+b1x; % known matrix A (x sweep)
Ay = dy+b^2*(a1y+b1y); % known matrix A (y sweep)
for k=1:1:kmax-1
    T(:,1,k+1) = T1; T(1,:,k+1) = T2; T(:,jmax,k+1) = T3; T(imax,:,k+1) =
T4; % BC at k+1/2
    for j=2:1:jmax-1 % known matrix C (x sweep)
        for i=2:1:imax-1
            Cx(i-1) = a*(1-w)*T(i,j,k)-w*b^2*(T(i,j+1,k)+T(i,j-1,k+1));
        end
        Cx(1) = Cx(1)-w*T(1,j,k+1);
        Cx(imax-2) = Cx(imax-2) -w*T(imax,j,k+1);
        T(2:1:imax-1,j,k+1) = (Ax\Cx')';
    end
    for i=2:1:imax-1 % known matrix C (y sweep)
        for j=2:1:jmax-1
            Cy(j-1) = a*(1-w)*T(i,j,k)-w*(T(i+1,j,k)+T(i-1,j,k+1));
        end
        Cy(1) = Cy(1)-w*b^2*T(i,1,k+1);
        Cy(jmax-2) = Cy(jmax-2)-w*b^2*T(i,jmax,k+1);
        T(i,2:1:jmax-1,k+1) = (Ay\Cy')';
    end
    dif = abs(sum(T(2:imax-1,2:jmax-1,k+1)-T(2:imax-1,2:jmax-
1,k),'all'));
    if dif/(imax*jmax)<=tol % tolerence check
        k = k+1;
        break
    end
    k = k+1;
end
time = toc
iteration = k
%% Plot
contourf(T(:,:,k)); % contour graph
colorbar; colormap('turbo'); view(90,270);
axis equal; hold on; grid on;
xticks([1 0.25*(jmax-1)+1 0.5*(jmax-1)+1 0.75*(jmax-1)+1 jmax])
xticklabels({'0','0.5','1','1.5','2'})
yticks([1 0.5*(imax-1)+1 imax])
yticklabels({'0','0.5','1'})
xlabel(['y (H=' num2str(H) ', jmax=' num2str(jmax) ')'])
ylabel(['x (L=' num2str(L) ', imax=' num2str(imax) ')'])
title(['ADIOR Method (',num2str(time,3) 's, iteration=' num2str(k) ',
\omega=' num2str(w) ')'],'FontSize',12)
```

〈 Table. 9 - ADIOR Method MATLAB CODE 〉

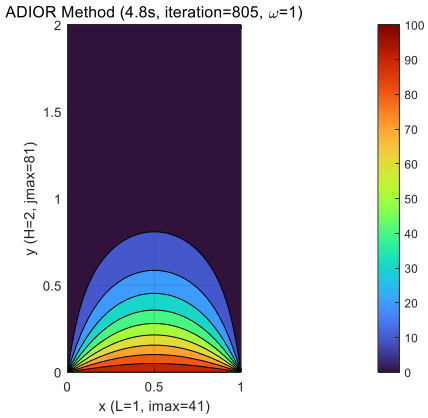
다음은 $L = 1,\ H = 2,\ i_{max} = 21,\ j_{max} = 41$에서 $\omega = 0.1, 0.5, 1.0, 1.34$일 때 Alternating Direction Implicit Over-Relaxation Method로 해석한 결과이다.



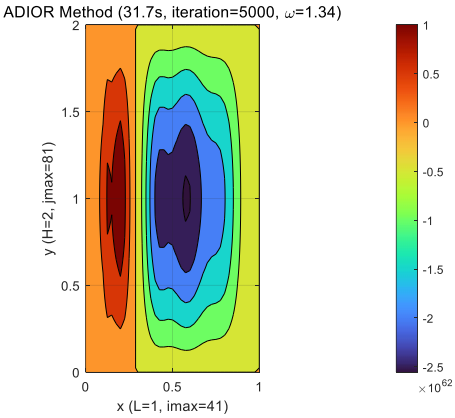
〈 Fig. 21 - ADIOR Method $\omega = 0.1$ 〉



〈 Fig. 22 - ADIOR Method $\omega = 0.5$ 〉



〈 Fig. 23 - ADIOR Method $\omega = 1.0$ 〉



〈 Fig. 24 - ADOR Method $\omega = 1.34$ 〉

Alternating Direction Implicit Over-Relaxation Method로 해석한 결과, $\omega = 0.1$일 때에는 35.1초, 5000 iteration이 수행되었고 그래프를 확인해보면 최대 iteration 값을 초과하여 정확한 해를 가지기 전에 중단되었다. $\omega = 0.5$일 때에는 22.3초, 2974 iteration, $\omega = 1.0$일 때에는 4.80초, 805 iteration이 수행되었다. ADIOR Method에서도 $\omega = 1.33$을 넘어가면 해가 발산하여 수렴조건을 만족하지 못하게 되고 최대 iteration인 5000에서 중단하게 된다. 따라서 Alternating Direction Implicit Over-Relaxation Method 또한 LSOR Method와 마찬가지로 ω값이 증가할수록 수렴되는 속도가 빨라져 계산 시간이 감소하고, iteration 횟수도 감소하는 것을 확인할 수 있었지만 일정 ω가 넘어가면 발산하는 것을 확인할 수 있었다.

4. 결론

이번에는 대표적인 타원형 특성(Elliptic Type)의 편미분 방정식인 정상 2차원 열전도 방정식(Steady-State 2-D Heat Conduction Equation)을 여러 수치해석 방법을 이용하여 해석하였다.

Elliptic Type의 PDE를 해석하는 방법에는 크게 Direct Method와 Iterative Method가 있다. Direct Method는 미지수의 개수만큼의 방정식을 만들어 행렬을 이용하여 한번에 풀이하는 방법이고, Iterative Method는 임의의 Initial Condition을 적용하여 여러 횟수에 걸쳐 해를 풀이한 뒤 앞서 지정한 수렴조건에 만족하면 반복을 중단하고 수렴조건에 만족하지 못하면 다시 반복하는 방법이다.

Direct Method로는 Five-Point Formula를 사용하였으며, Iterative Method로는 Point Iteration Method, Line Iteration Method를 Jacobi Iteration과 Gauss-Seidel Iteration을 사용하여 해석해 보았다. 또한, relaxation parameter를 사용하는 Point Successive Over-Relaxation Method, Line Successive Over-Relaxation Method을 사용하여 해석해보았다. 한 Iteration을 방향을 바꾸어 두 번 실시하는 Alternating Direction Implicit Method와 마지막으로 이 방법에 relaxation까지 부여하는 Alternating Direction Implicit Over-Relaxation Method까지 사용하여 해석해보았다.

$L = 1,\ H = 2,\ i_{max} = 21,\ j_{max} = 41$일 때, Iteration Method에서 Point Method와 Line Method 중 Point Method가 계산 속도가 더 빨랐으며, Gauss-Sidel Iteration Method가 Jacobi Iteration Method보다 계산 속도가 증가하는 특성을 보였다. 이 계산 속도는 Over-Relaxation 사용하였을 때 $\omega = 1.33$을 사용하였을 때 가장 빠른 것으로 나타났으며, $\omega = 1.33$을 넘어가면 해석 결과가 발산하는 특성을 보였다.