AM5630: Foundations of Computational Fluid Dynamics

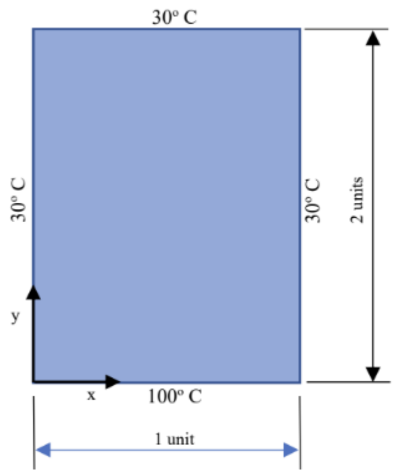
Computer Assignment - 2

Jaswanth VG, MM21B031

25 October 2024

1 Problem Definition

A rectangular plate of unit thickness with dimensions 1 unit by 2 units (2-D steady-state diffusion problem) is given with Dirichlet boundary conditions on all four sides, as shown below:



The steady-state temperature distribution and the magnitude of heat transfer along the bottom boundary have to be found using six different implicit schemes. The thermal conductivity of the plate, k, is given: k = 50 W/mK.

We shall assume that all the points on the plate before the applying boundary conditions were at 30^{o}C.

2 Boundary Conditions

The given Dirichlet boundary conditions (also known as essential boundary conditions) are:

Bottom Boundary: T(x,y=0) = 100^{o}C
All other Boundaries: T(x,y=2) = T(x=0,y) = T(x=1,y) = 30^{o}C

3 Governing Equations

The 2-D steady state heat conduction equation with no heat generation term is:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0 \tag{1}$$

Since B^2 - 4AC < 0, we have an elliptic equation with only spacial variables and no temporal variable, which is why we don't have initial conditions.

4 Numerical Formulation

We shall use the Finite Difference Method (FDM) to discretize equation (1). Discretizing the above Laplace equation using central difference gives us:

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

where Δx and Δy are grid spacing along the X and Y directions, respectively. Substituting $\beta = \frac{\Delta x}{\Delta y}$, we get:

$$-T_{i-1,j} + 2(1+\beta^2)T_{i,j} - T_{i+1,j} = \beta^2(T_{i,j-1} + T_{i,j+1}) \tag{3}$$

OR

$$T_{i,j}^{k+1} = \frac{1}{2(1+\beta^2)}[T_{i-1,j}^{k} + T_{i+1,j}^{k} + \beta^2 T_{i,j-1}^{k} + \beta^2 T_{i,j+1}^{k}] \tag{4}$$

where k+1 and k represent consecutive iterations during the converging process. Now, we shall use various implicit schemes to solve this problem:

4.1 Point Gauss-Seidel Method

$$T_{i,j}^{k+1} = \frac{1}{2(1+\beta^2)}[T_{i-1,j}^{k+1} + T_{i+1,j}^{k} + \beta^2 T_{i,j-1}^{k+1} + \beta^2 T_{i,j+1}^{k}] \tag{5}$$

This method is computationally less intensive than the conventional formulation in equation (4) as we utilize the updated values of $T_{i-1,j}$ and $T_{i,j-1}$ from the $(k+1)^{th}$ iteration instead of taking the values from the $k^{th}4$ iteration.

4.2 Line Gauss-Seidel Method

$$-T_{i-1,j}^{k+1} + 2(1+\beta^2)T_{i,j}^{k+1} - T_{i+1,j}^{k+1} = \beta^2(T_{i,j-1}^{k+1} + T_{i,j+1}^{k}) \tag{6}$$

Iterating over i = 2 to imax-1 and j = 2 to jmax-1 (ignoring the boundaries due to boundary conditions), we get a tridiagonal matrix system, which is solved by using the Tridiagonal Matrix Algorithm (TDMA) or Thomas Algorithm.

4.3 Point Successive Over Relaxation (PSOR) Method

In this method, we introduce a relaxation factor, ω, which, depending on the value, varies the weight between the correction part of the equation and the previous iteration value of the Temperature at i,j.

$$T_{i,j}^{k+1} = (1-\omega)T_{i,j}^{k} + \frac{\omega}{2(1+\beta^2)}[T_{i-1,j}^{k+1} + T_{i+1,j}^{k} + \beta^2 T_{i,j-1}^{k+1} + \beta^2 T_{i,j+1}^{k}] \tag{7}$$

Under Relaxation: $0 < \omega < 1$
Over Relaxation: $1 < \omega < 2$
An optimum value of ω is usually chosen through hit-and-trial; The optimum value of ω is obtained when the number of iterations for convergence is the least.

4.4 Line Successive Over Relaxation (LSOR) Method

$$-\omega T_{i-1,j}^{k+1} + 2(1+\beta^2)T_{i,j}^{k+1} - \omega T_{i+1,j}^{k+1} = \omega\beta^2(T_{i,j-1}^{k+1} + T_{i,j+1}^{k}) + 2(1-\omega)(1+\beta^2)T_{i,j}^{k+1} \tag{8}$$

This, too, will yield us a tridiagonal system of linear equations upon Iterating over i = 2 to imax-1 and j = 2 to jmax-1, which is solved by using the Thomas Algorithm.

4.5 Alternating Direction Implicit (ADI) Scheme

ADI Scheme introduces an intermediate iteration step. The temperature matrix at the intermediate step (k+1/2) is found using the Line Gauss-Seidel method along the first direction, and the temperature matrix at (k+1)the iteration is found using the Line Gauss-Seidel method along the second direction.

Along X:

$$-T_{i-1,j}^{k+1/2} + 2(1+\beta^2)T_{i,j}^{k+1/2} - T_{i+1,j}^{k+1/2} = \beta^2(T_{i,j-1}^{k+1/2} + T_{i,j+1}^{k}) \tag{9}$$

Along Y:

$$-\beta^2 T_{i,j-1}^{k+1} + 2(1+\beta^2)T_{i,j}^{k+1} - \beta^2 T_{i,j+1}^{k+1} = T_{i-1,j}^{k+1} + T_{i+1,j}^{k+1/2} \tag{10}$$

This method is solved by using the Thomas algorithm at both along X and along Y sweeps.

4.6 Alternating Direction Implicit (ADI) Scheme with Relaxation

This formulation is similar to the LSOR and PSOR schemes, where a relaxation factor, ω, is introduced to give weights to the previous value and the correction value in equations (9) and (10).

Along X:

$$-\omega T_{i-1,j}^{k+1/2} + 2(1+\beta^2)T_{i,j}^{k+1/2} - \omega T_{i+1,j}^{k+1/2} = \omega\beta^2(T_{i,j-1}^{k+1/2} + T_{i,j+1}^{k}) + 2(1-\omega)(1+\beta^2)T_{i,j}^{k} \tag{11}$$

Along Y:

$$-\omega\beta^2 T_{i,j-1}^{k+1} + 2(1+\beta^2)T_{i,j}^{k+1} - \omega\beta^2 T_{i,j+1}^{k+1} = \omega(T_{i-1,j}^{k+1} + T_{i+1,j}^{k+1/2}) + 2(1-\omega)(1+\beta^2)T_{i,j}^{k+1/2} \tag{12}$$

5 Pseudocode

Algorithm 1 Temperature Distribution Solver

Initialize parameters:
$\Delta x \leftarrow 0.05$, $\Delta y \leftarrow 0.05$
$k \leftarrow 50$ ▷ Thermal conductivity
$rows \leftarrow \frac{2}{\Delta y} + 1$, $columns \leftarrow \frac{1}{\Delta x} + 1$
$\beta \leftarrow \frac{\Delta x}{\Delta y}$
Define error threshold: 0.01

Algorithm 2 tdma_solve (Thomas Algorithm)

Input: a, b, c, d
Apply Thomas algorithm to solve tridiagonal system
for $i \leftarrow 1$ to n **do**
  Modify coefficients b' and d'
end for
for $i \leftarrow n-2$ to 0 **do** ▷ Backward substitution
  Solve for $x[i]$
end for

Algorithm 3 solve_and_plot (Main Function)

Initialize temperature arrays: $temp\_old[rows][columns]$, $temp\_new[rows][columns]$
for $i \leftarrow 0$ to $rows - 1$ **do**
  for $j \leftarrow 0$ to $columns - 1$ **do**
    $temp\_new[i][j] \leftarrow 30$
  end for
end for
Set boundary conditions:
Bottom boundary $\leftarrow 100°C$
Left, Right, Top boundaries $\leftarrow 30°C$
$temp\_old \leftarrow temp\_new$
Initialize: $error \leftarrow 1$, $step \leftarrow 0$
while $error \geq 0.01$ **do**
  $error \leftarrow 0$
  Apply $scheme(temp\_old, temp\_new)$
  for $i \leftarrow 1$ to $rows - 2$ **do**
    for $j \leftarrow 1$ to $columns - 2$ **do**
      $error \leftarrow error + |temp\_new[i][j] - temp\_old[i][j]|$
      $temp\_old[i][j] \leftarrow temp\_new[i][j]$
    end for
  end for
  $step \leftarrow step + 1$
end while
Calculate heat transfer:
$dT\_dx \leftarrow \frac{temp\_new[rows-1][columns/2]-temp\_new[rows-2][columns/2]}{\Delta y}$
$heat\_transfer \leftarrow -k \cdot dT\_dx$
Plot temperature distribution

Algorithm 4 Point_Gauss_Seidel (Point Gauss-Seidel Method)

for $i \leftarrow 1$ to $rows - 2$ **do**
  for $j \leftarrow 1$ to $columns - 2$ **do**
    Update $temp\_new[i][j]$ using Gauss-Seidel formula
  end for
end for

Algorithm 5 PSOR (Point Successive Over-Relaxation)

Set relaxation factor: $\omega \leftarrow 1.79$
for $i \leftarrow 1$ to $rows - 2$ **do**
  for $j \leftarrow 1$ to $columns - 2$ **do**
    Update $temp\_new[i][j]$ using SOR formula
  end for
end for

Algorithm 6 Line_Gauss_Seidel (Line Gauss-Seidel Method)

for $j \leftarrow 1$ to $columns - 2$ **do**
  Prepare tridiagonal system along rows
  Solve using TDMA_SOLVE
end for

Algorithm 7 LSOR (Line Successive Over-Relaxation)

for $j \leftarrow 1$ to $columns - 2$ **do**
  Prepare tridiagonal system with SOR adjustments
  Solve using TDMA_SOLVE
end for

Algorithm 8 ADI (Alternating Direction Implicit Scheme)

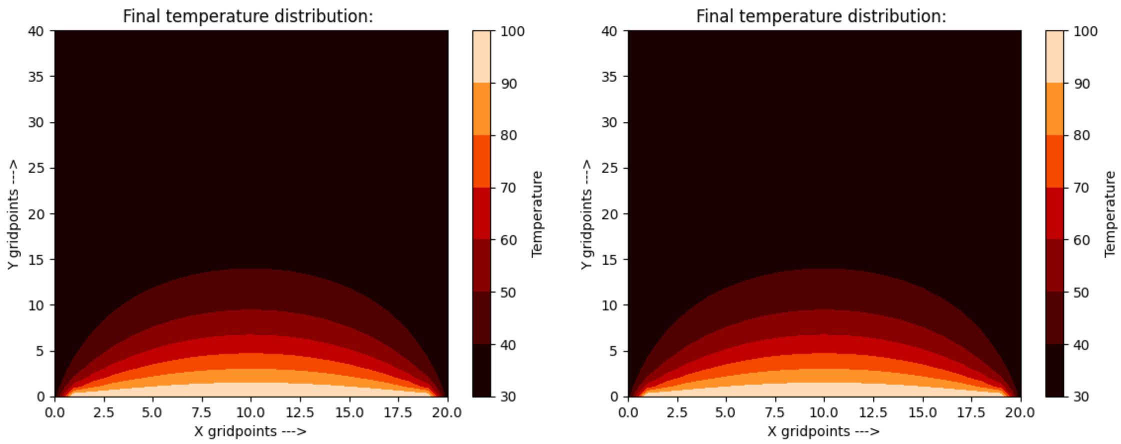
Split into two sweeps:
for X-direction sweep **do**
  Solve tridiagonal system along columns
end for
for Y-direction sweep **do**
  Solve tridiagonal system along rows
end for

Algorithm 9 ADIwRlx (ADI with Relaxation)

Set relaxation factor: $\omega \leftarrow 1.31$
Split into two sweeps with relaxation:
for X-direction sweep **do**
  Solve tridiagonal system with SOR adjustments
end for
for Y-direction sweep **do**
  Solve tridiagonal system with SOR adjustments
end for

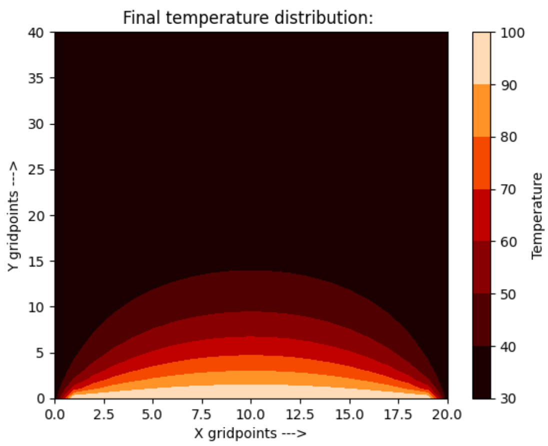
6 Results and Discussion

A Python script was created to implement the above algorithm. A rectangular temperature matrix was created with rows and columns representing the length of the 2D plate along the X and Y directions, respectively.
Upon running the script using a grid size of 0.05, all the schemes yielded similar temperature distributions obtained using the contourf() function in matplotlib, as shown:

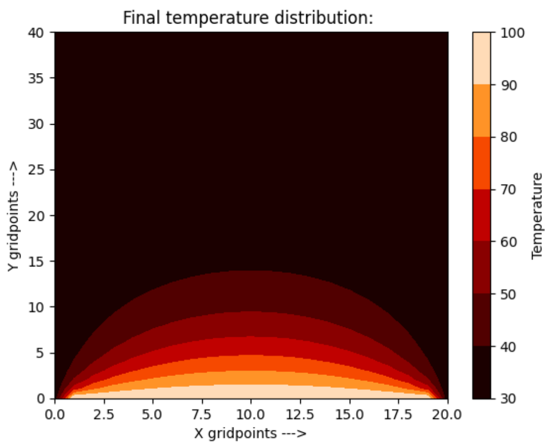


(a) Line Gauss Seidel

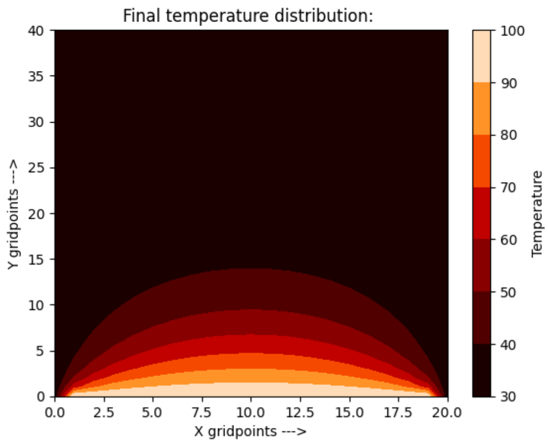
(b) Point Gauss Seidel



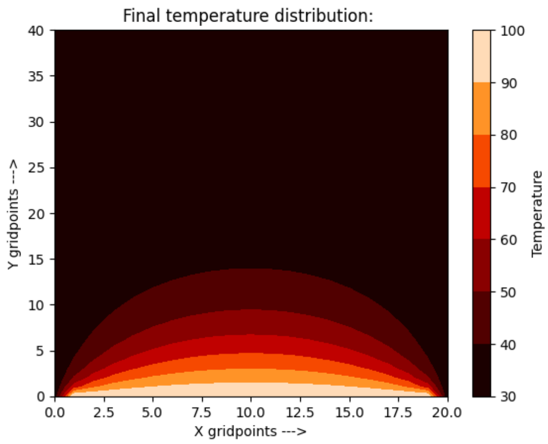
(c) Point Successive Over Relaxation



(d) Line Successive Over Relaxation



(e) Alternating Direction Implicit (ADI) scheme



(f) ADI with relaxation

Heat transfer at the centre-most point on the bottom boundary of the plate (at a grid size of 0.05 using ADI with relaxation scheme) is calculated as follows by using the backward difference method:

$$q = -k\left(\frac{\partial T}{\partial y}\right) = -50\left(\frac{T_{imax/2,jmax} - T_{imax/2,jmax-1}}{\Delta y}\right) = -50\left(\frac{6.9994}{0.05}\right) = -6999.40W/m^2 \tag{13}$$

Similarly, the below plot shows the heat transfer variation along the bottom boundary at all points along the bottom boundary:

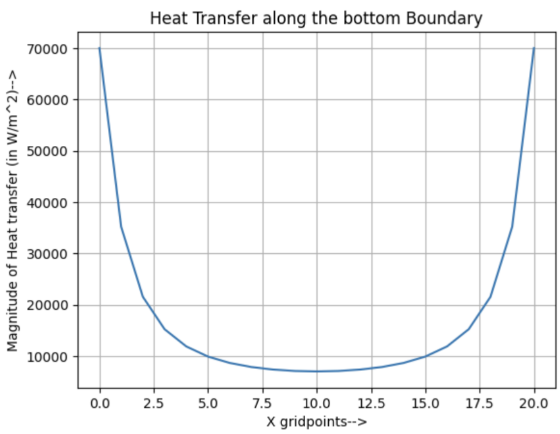


Figure 1: Variation of heat transfer along the bottom boundary in X direction

The table shown below summarises the performance of various schemes used at different grid sizes:

S.No.	Scheme Used	Grid Size	No.of Iterations	Relaxation factor (ω)
1	Point Gauss Seidel	0.05 (21 x 41)	570	-
2	Line Gauss Seidel	0.05 (21 x 41)	303	-
3	ADI	0.05 (21 x 41)	168	-
4	PSOR	0.05 (21 x 41)	71	1.79
5	LSOR	0.05 (21 x 41)	42	1.27
6	ADI with relaxation	0.05 (21 x 41)	25	1.31
7	Point Gauss Seidel	0.03 (34 x 67)	1542	-
8	Line Gauss Seidel	0.03 (34 x 67)	825	-
9	ADI	0.03 (34 x 67)	449	-
10	PSOR	0.03 (34 x 67)	221	1.79
11	LSOR	0.03 (34 x 67)	134	1.27
12	ADI with relaxation	0.03 (34 x 67)	47	1.31
13	Point Gauss Seidel	0.02 (51 x 101)	3528	-
14	Line Gauss Seidel	0.02 (51 x 101)	1893	-
15	ADI	0.02 (51 x 101)	1025	-
16	PSOR	0.02 (51 x 101)	521	1.79
17	LSOR	0.02 (51 x 101)	327	1.27
18	ADI with relaxation	0.02 (51 x 101)	89	1.31

As one can notice, the ADI with relaxation scheme is the best-performing (converging almost 20 times quicker than the point Gauss seidel scheme) scheme with a relaxation factor of 1.31, which was found out by hit-and-trial. The relaxation factors for the PSOR and LSOR schemes, 1.79 and 1.0, respectively, were also found using the same method for the quickest convergence.

Also, overall, the point Gauss seidel scheme takes the most number of iterations, followed by the line Gauss seidel scheme and the ADI scheme. Each of these schemes dramatically improves when used with the optimum relaxation factor obtained by hit-and-trial.

7 Appendix

7.1 Python Script:

```python
import numpy as np
import matplotlib.pyplot as plt

del_x = 0.05
del_y = 0.05
beta = del_x / del_y
k = 50  #thermal conductivity
rows = int((2/del_y)) + 1
columns = int((1/del_x)) + 1
heat_transfer = []

def solve_and_plot(scheme):
    #initial temperature of all points except the bottom line are set to 30 deg.C for faster convergence
    temp_old = np.zeros([rows, columns])
    temp_new = np.zeros([rows, columns])
    for i in range(rows):
        for j in range(columns):
            temp_new[i][j] = 30
    temp_new[-1, :] = 100  # Bottom boundary
    temp_old = temp_new.copy()  # Initialize temp_old with boundary conditions

    error = 1 #dummy error to get into the loop
    step = 0

    print("Solving using the", scheme.__name__ ,"scheme:\n")
    #Iterate
    while (error >= 0.01):
        error = 0
        scheme(temp_old, temp_new)
        for i in range(1, rows-1):
            for j in range(1, columns-1):
                error += abs(temp_new[i][j] - temp_old[i][j])
                temp_old[i][j] = temp_new[i][j]
        step += 1
    print("Number of iterations taken to converge: ",step)
    #Calculate heat transfer
    for i in range(columns):
        heat_transfer.append(50.0*(temp_new[rows-1][i] - temp_new[rows-2][i]) / del_y)
    print("Heat transfer along the bottom boundary: ",heat_transfer)
    #Plot temperature contour
```

```python
    X = np.arange(0,columns)
    Y = np.flip(np.arange(0,rows))
    plt.contourf(X,Y,temp_new, cmap = 'gist_heat')
    plt.colorbar(label = 'Temperature')
    plt.xlabel("X gridpoints --->")
    plt.ylabel("Y gridpoints --->")
    plt.title("Final temperature distribution: ")
    plt.show()
    #plot heat transfer
    plt.title("Heat Transfer along the bottom Boundary")
    plt.plot(heat_transfer)
    plt.grid(True)
    plt.xlabel("X gridpoints-->")
    plt.ylabel("Magnitude of Heat transfer (in W/m^2)-->")
    plt.show()

#Function for thomas algorithm
def tdma_solve(a,b,c,d):
    n = len(a)
    bprime = b.copy()
    dprime = d.copy()
    for i in range(1,n+1):
        factor = a[i-1] / bprime[i-1]
        bprime[i] -= factor*c[i-1]
        dprime[i] -= factor*dprime[i-1]

    nx = len(d)
    x = np.zeros(nx)
    x[-1] = dprime[-1] / bprime[-1]
    for i in range(nx-2,-1,-1):
        x[i] = (dprime[i] - c[i]*x[i+1]) / bprime[i]

    return x

#Point gauss-seidel method:
def Point_Gauss_Seidel(temp_old, temp_new):
    for i in range(1, rows-1):
        for j in range(1, columns-1):
            temp_new[i][j] = (0.5 / (1 + beta**2))*(temp_new[i-1][j] + temp_old[i+1][j] + (beta**2
                )*temp_old[i][j+1] + (beta**2)*temp_new[i][j-1])

```

```
#Point Successive Over Relaxation:
def PSOR(temp_old, temp_new):
    omega = 1.79
    for i in range(1, rows-1):
        for j in range(1, columns-1):
            temp_new[i][j] = (1 - omega)*temp_old[i][j] + ((0.5*omega) / (1 + beta**2))*(temp_new[i-1][j] +
                temp_old[i+1][j] + (beta**2)*temp_old[i][j+1] + (beta**2)*temp_new[i][j-1])

#Line gauss-seidel method:
def Line_Gauss_Seidel(temp_old, temp_new):
    for j in range(1,columns-1):
        a = np.ones(rows-3) * -1.0
        b = np.ones(rows-2) * 2 * (1.0 + beta**2)
        c = np.ones(rows-3) * -1.0
        d = np.zeros(rows-2)
        for i in range(1,rows-1):
            d[i-1] = beta**2 * (temp_new[i][j-1] + temp_old[i][j+1])
        d[0] += 30
        d[-1] += 100

        new_values = tdma_solve(a,b,c,d)
        temp_new[1:rows-1, j] = new_values

#Line successive over-relaxation method:
def LSOR(temp_old, temp_new):
    omega = 1.27
    for j in range(1,columns-1):
        a = np.ones(rows-3) * -1.0 * omega
        b = np.ones(rows-2) * 2 * (1.0 + beta**2)
        c = np.ones(rows-3) * -1.0 * omega
        d = np.zeros(rows-2)
        for i in range(1,rows-1):
            d[i-1] = (omega * beta**2 * (temp_new[i][j-1] + temp_old[i][j+1])) + (2*(1 - omega)*(1 + beta**2
                )*temp_old[i][j])
        d[0] += 30*omega
        d[-1] += 100*omega

        new_values = tdma_solve(a,b,c,d)
        temp_new[1:rows-1, j] = new_values

#ADI Scheme:
def ADI(temp_old, temp_new):
```

```
    temp_mid = temp_old.copy()
    #Along X:
    for j in range(1,columns-1):
        a = np.ones(rows-3) * -1.0
        b = np.ones(rows-2) * 2 * (1.0 + beta**2)
        c = np.ones(rows-3) * -1.0
        d = np.zeros(rows-2)
        for i in range(1,rows-1):
            d[i-1] = beta**2 * (temp_mid[i][j-1] + temp_old[i][j+1])
        d[0] += 30
        d[-1] += 100
        new_values = tdma_solve(a,b,c,d)
        temp_mid[1:rows-1, j] = new_values

    #ALong Y:
    for i in range(1,rows-1):
        a = np.ones(columns-3) * -1.0 * beta**2
        b = np.ones(columns-2) * 2 * (1.0 + beta**2)
        c = np.ones(columns-3) * -1.0 * beta**2
        d = np.zeros(columns-2)
        for j in range(1,columns-1):
            d[j-1] = temp_new[i-1][j] + temp_mid[i+1][j]
        d[0] += 30 * beta**2
        d[-1] += 30 * beta**2
        new_values = tdma_solve(a,b,c,d)
        temp_new[i, 1:columns-1] = new_values

#ADI Scheme with relaxation:
def ADIwRlx(temp_old, temp_new):
    omega = 1.31
    temp_mid = temp_old.copy()
    #Along X:
    for j in range(1,columns-1):
        a = np.ones(rows-3) * -1.0 * omega
        b = np.ones(rows-2) * 2 * (1.0 + beta**2)
        c = np.ones(rows-3) * -1.0 * omega
        d = np.zeros(rows-2)
        for i in range(1,rows-1):
            d[i-1] = omega * beta**2 * (temp_mid[i][j-1] + temp_old[i][j+1]) + 2*(1-omega)*(1+beta**2
                )*temp_old[i][j]
        d[0] += omega * 30
```

```
        d[-1] += omega * 100
        new_values = tdma_solve(a,b,c,d)
        temp_mid[1:rows-1, j] = new_values

    #ALong Y:
    for i in range(1,rows-1):
        a = np.ones(columns-3) * -1.0 * beta**2 * omega
        b = np.ones(columns-2) * 2 * (1.0 + beta**2)
        c = np.ones(columns-3) * -1.0 * beta**2 * omega
        d = np.zeros(columns-2)
        for j in range(1,columns-1):
            d[j-1] = omega * (temp_new[i-1][j] + temp_mid[i+1][j]) + 2*(1-omega)*(1+beta**2)*temp_mid[i][j]
        d[0] += 30 * beta**2 * omega
        d[-1] += 30 * beta**2 * omega
        new_values = tdma_solve(a,b,c,d)
        temp_new[i, 1:columns-1] = new_values

solve_and_plot(Point_Gauss_Seidel)
solve_and_plot(Line_Gauss_Seidel)
solve_and_plot(ADI)
solve_and_plot(PSOR)
solve_and_plot(LSOR)
solve_and_plot(ADIwRlx)
```