COMPUTER ASSIGNMENT 2 REPORT

Pratham Sunkad

ME21B145

IIT Madras

AM 5630 Foundations of CFD

Prof. Arul Prakash

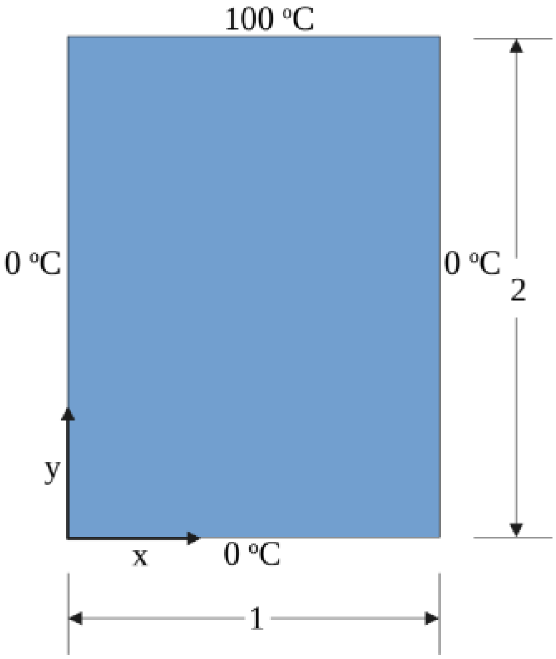
13th April 2024

1. Problem Definition:

Using various schemes derived from the finite difference method to numerically solve a steady state elliptic equation problem.

The schemes used are:

1) Point Gauss Seidel
2) Line Gauss Seidel
3) Point Successive Over- Relaxation
4) Line Successive Over - Relaxation
5) Alternating Direction Implicit
6) Alternating Direction Implicit with Relaxation



2. Governing Equation:

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

Steady State Conduction Equation in 2D

3. Boundary Condition:

Top wall = 100℃

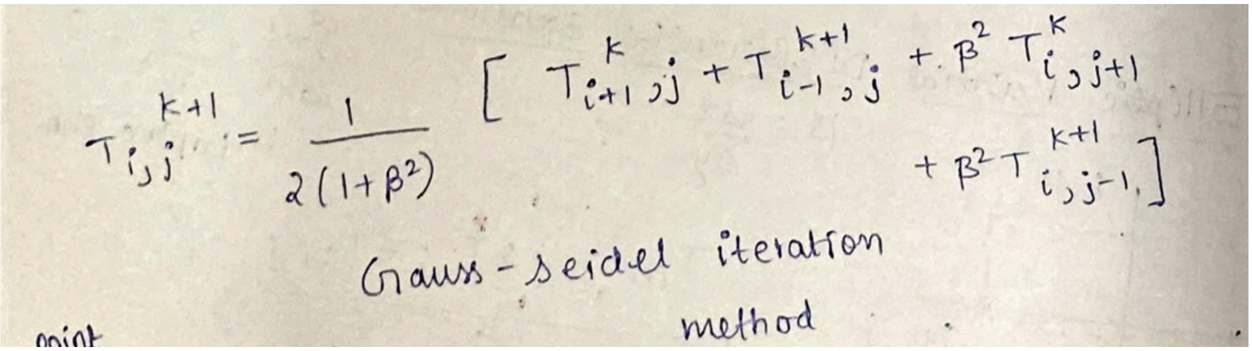
Left wall = 0℃

Right wall = 0℃

Bottom wall = 0℃

4. Numerical Formulation:

1) **Point Gauss Seidel** : This is the finite difference update equation for gauss seidel method.

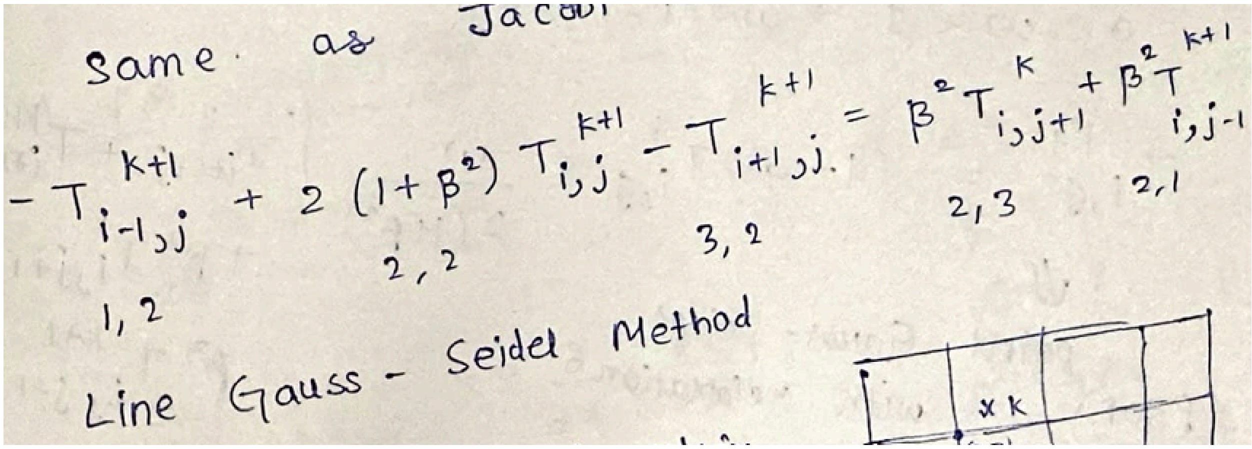


$$T_{i,j}^{k+1} = \frac{1}{2(1+\beta^2)} \left[T_{i+1,j}^{k} + T_{i-1,j}^{k+1} + \beta^2 T_{i,j+1}^{k} + \beta^2 T_{i,j-1}^{k+1} \right]$$

Gauss-seidel iteration method

point

2) **Line Gauss Seidel** : Sweep along the rows and formulate a tridiagonal matrix, solve it and get Temperature values in the row

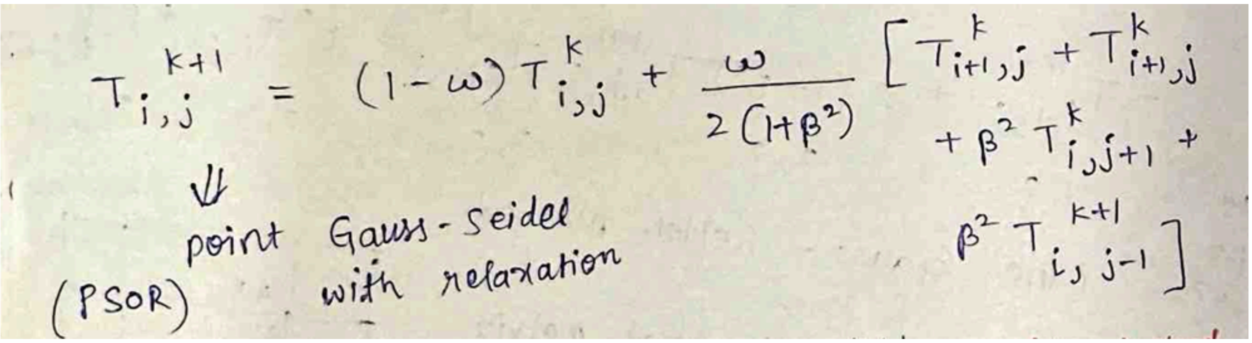


same as Jacobi

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

Line Gauss - Seidel Method

3) **PSOR :** Over relaxation is used to help in converging faster.
0<ω<1- under relaxation, 1<ω<2- over relaxation



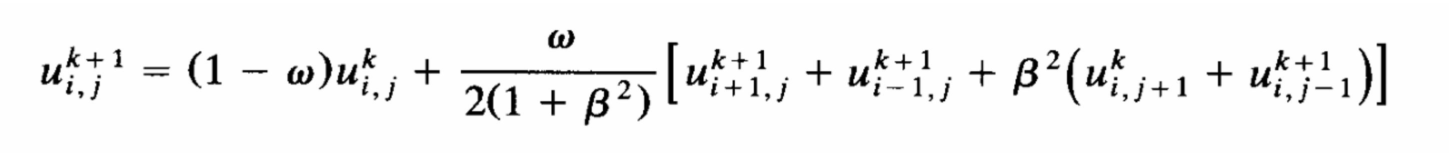
$$T_{i,j}^{k+1} = (1-\omega) T_{i,j}^{k} + \frac{\omega}{2(1+\beta^2)} \left[T_{i+1,j}^{k} + T_{i+1,j}^{k} + \beta^2 T_{i,j+1}^{k} + \beta^2 T_{i,j-1}^{k+1} \right]$$

⇓

point Gauss-Seidel with relaxation

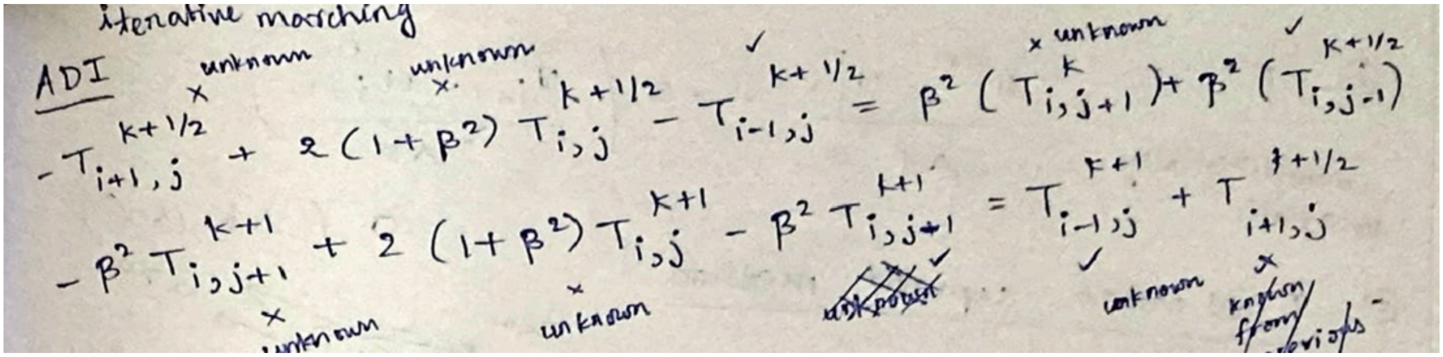
(PSOR)

4) **LSOR** : Sweep through rows or columns. Apply the relaxation parameter for faster convergence. Tridiagonal Matrix algorithm is used to solve for Temperature in each row.



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

5) **ADI** : Introduce T$^{k+½}$ by sweeping over rows first. Use T$^{k+½}$ to calculate T^{k+1} by sweeping over the columns. Solve the Tridiagonal matrix at each row/column sweep to get Temperature values.



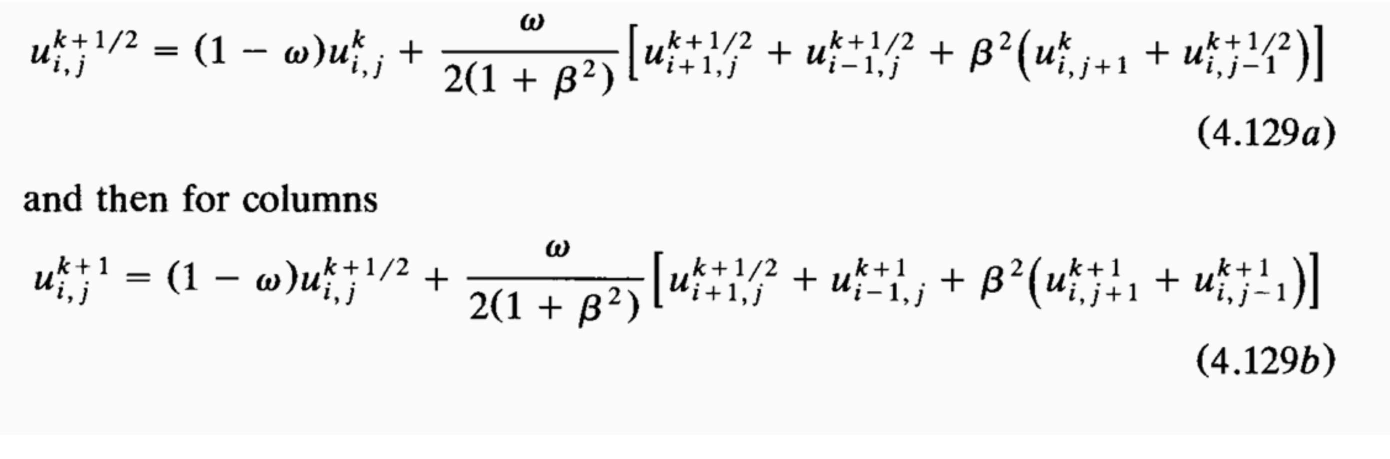
iterative marching

ADI

$$-\overset{\text{unknown}\ \times}{T_{i+1,j}^{k+1/2}} + 2(1+\beta^2)\overset{\text{unknown}\ \times}{T_{i,j}^{k+1/2}} - \overset{\checkmark}{T_{i-1,j}^{k+1/2}} = \beta^2\left(\overset{\times\ \text{unknown}}{T_{i,j+1}^{k}}\right) + \beta^2\left(\overset{\checkmark}{T_{i,j-1}^{k+1/2}}\right)$$

$$-\beta^2\underset{\times\ \text{unknown}}{T_{i,j+1}^{k+1}} + 2(1+\beta^2)\underset{\times\ \text{unknown}}{T_{i,j}^{k+1}} - \beta^2\underset{\checkmark\ \cancel{\text{unknown}}}{T_{i,j+1}^{k+1}} = \underset{\checkmark\ \text{unknown}}{T_{i-1,j}^{k+1}} + \underset{\times\ \text{known from previous}}{T_{i+1,j}^{k+1/2}}$$

6) **ADI with relaxation :** Add the relaxation parameter in both the row and column sweeps.



$$u_{i,j}^{k+1/2} = (1-\omega)u_{i,j}^{k} + \frac{\omega}{2(1+\beta^2)}\left[u_{i+1,j}^{k+1/2} + u_{i-1,j}^{k+1/2} + \beta^2\left(u_{i,j+1}^{k} + u_{i,j-1}^{k+1/2}\right)\right] \tag{4.129a}$$

and then for columns

$$u_{i,j}^{k+1} = (1-\omega)u_{i,j}^{k+1/2} + \frac{\omega}{2(1+\beta^2)}\left[u_{i+1,j}^{k+1/2} + u_{i-1,j}^{k+1} + \beta^2\left(u_{i,j+1}^{k+1} + u_{i,j-1}^{k+1}\right)\right] \tag{4.129b}$$

5. PseudoCode:

i) Point Gauss Seidel:

Step 1: Declare all variables such as Δx , Δy , β , T.

Step 2: Create two 2D matrices T_prev and T_array for storing the value of temperature at k and $k + 1$ iteration of size N × M where, M is number of grid points in x-direction and N is number of grid points in y-direction (here, M = 21, N = 41).

Step 3: Apply Boundary Conditions

```
T_array(1,:) = T_bottom;

T_array(:,1) = T_left;

T_array(:, end) = T_right;

T_array(end, :) = T_top;
```

Step 4 : Apply the Gauss Seidel equation

```
for i = 2:n_x - 1

       for j = 2:n_y-1

           T_array(j,i) = (0.5/(1+beta))*(T_prev(j,i+1) + T_array(j,i-1) + 
beta*T_prev(j+1,i) + beta*T_array(j-1,i));
```

ii) Line Gauss Seidel:

Step 1: Declare all variables such as Δx , Δy , β , T.

Step 2: Create two 2D matrices T_prev and T_array for storing the value of temperature at k and $k + 1$ iteration of size N × M where, M is number of grid points in x-direction and N is number of grid points in y-direction (here, M = 21, N = 41).

Create T_check which is of size N × M with all interior elements 0 and the boundary conditions applied.

Step 3 : Apply Boundary Conditions:

```
T_array(1,:) = T_bottom;

T_array(:,1) = T_left;

T_array(:, end) = T_right;

T_array(end, :) = T_top;
```

Step 4: Sweep along the rows, form the Tridiagonal matrix at each row

```
rhs = transpose(beta*T_prev(j+1, 2: n_x-1) + beta*T_check(j-1, 2: n_x-1));

 tri_diag_mat = zeros((n_x-2),(n_x-2));

       for i = 2:n_x-1

              if i == 2

                  tri_diag_mat(i-1,i-1) = 2*(1+beta);

                  tri_diag_mat(i,i-1) = -1;

                  continue


              if i == n_x - 1

                  tri_diag_mat(i-1,i-1) = 2*(1+beta);

                  tri_diag_mat(i-2,i-1) = -1;

                  continue

              tri_diag_mat(i-1,i-1) = 2*(1+beta);

              tri_diag_mat(i,i-1) = -1;

              tri_diag_mat(i-2,i-1) = -1;
```

Solve the Tridiagonal matrix by using the Tridiagonal Matrix algorithm (tdma) and attain temperature values. Do this one row at a time

iii) PSOR:

Step 1: Declare all variables such as Δx , Δy , β , T.

Step 2: Create two 2D matrices T_prev and T_array for storing the value of temperature at k and $k + 1$ iteration of size N × M where, M is number of grid points in x-direction and N is number of grid points in y-direction (here, M = 21, N = 41).

Step 3: Apply Boundary Conditions

```
T_array(1,:) = T_bottom;

T_array(:,1) = T_left;

T_array(:, end) = T_right;

T_array(end, :) = T_top;
```

Step 4: Apply the PSOR equation with relaxation parameter

```
for i = 2:n_x - 1

       for j = 2:n_y-1

           T_array(j,i) = (1-w)*T_prev(j,i) + (w*0.5/(1+beta))*(T_prev(j,i+1) +
T_array(j,i-1) + beta*T_prev(j+1,i) + beta*T_array(j-1,i));
```

iv) LSOR:

Step 1: Declare all variables such as Δx , Δy , β , T.

Step 2: Create two 2D matrices T_prev and T_array for storing the value of temperature at k and $k + 1$ iteration of size N × M where, M is number of grid points in x-direction and N is number of grid points in y-direction (here, M = 21, N = 41).

Create T_check which is of size N × M with all interior elements 0 and the boundary conditions applied.

Step 3: Apply Boundary Conditions

```
T_array(1,:) = T_bottom;

T_array(:,1) = T_left;

T_array(:, end) = T_right;

T_array(end, :) = T_top;
```

Step 4: Sweep along rows and Form the tridiagonal matrix for each row. The rhs indicates the right hand side of the matrix equation. Solve the matrix using the Tridiagonal Matrix Algorithm to get temperatures of the current row. Repeat this for each row.

```
for j = 2: n_y-1

       rhs = transpose(beta*T_prev(j+1, 2: n_x-1) + beta*T_check(j-1, 2: n_x-1)
+ ((1-w)*(2+2*beta)*T_prev(j, 2: n_x-1)/w));

       tri_diag_mat = zeros((n_x-2),(n_x-2));

       for i = 2:n_x-1

              if i == 2

                  tri_diag_mat(i-1,i-1) = 2*(1+beta)/w;

                  tri_diag_mat(i,i-1) = -1;

                  continue

              if i == n_x - 1

                  tri_diag_mat(i-1,i-1) = 2*(1+beta)/w;

                  tri_diag_mat(i-2,i-1) = -1;

                  continue

              tri_diag_mat(i-1,i-1) = 2*(1+beta)/w;

              tri_diag_mat(i,i-1) = -1;

              tri_diag_mat(i-2,i-1) = -1;
```

v) ADI:

Step 1: Declare all variables such as Δx , Δy , β , T.

Step 2: Create two 2D matrices T_half and T_array for storing the value of temperature at k and $k + 1/2$ iteration of size N × M where, M is number of grid points in x-direction and N is number of grid points in y-direction (here, M = 21, N = 41).

Create T_check which is of size N × M with all interior elements 0 and the boundary conditions applied. T_check will have elements at $k + 1$ iteration at the end of a for loop.

Step 3: Apply Boundary Conditions

```
T_array(1,:) = T_bottom;
T_array(:,1) = T_left;
T_array(:, end) = T_right;
T_array(end, :) = T_top;
```

Step 4: Form Tridiagonal Matrix using row sweep for each row. Solve this to get T at $k + 1/2$ iteration.

Now sweep along the columns. Use Temperature values obtained at $k + 1/2$ iteration to form the Tridiagonal Matrix and solve for each column to get T $k + 1$.

Code for row sweep same as LSOR to get T at $k + 1/2$ iteration.

Code for column sweep as follows:

```
for i = 2: n_x-1
    rhs = T_check(2: n_y-1, i-1) + T_half(2: n_y-1, i+1);
    tri_diag_mat = zeros((n_y-2),(n_y-2));
    for j = 2:n_y-1
        if j == 2
          tri_diag_mat(j-1,j-1) = 2*(1+beta);
          tri_diag_mat(j,j-1) = -beta;
          rhs(j-1) = rhs(j-1) + beta* T_bound(j-1,i);
          continue
        if j == n_y - 1
          tri_diag_mat(j-1,j-1) = 2*(1+beta);
          tri_diag_mat(j-2,j-1) = -beta;
          rhs(j-1) = rhs(j-1) + beta*T_bound(j+1, i);
```

```
                   continuu
               tri_diag_mat(j-1,j-1) = 2*(1+beta);
               tri_diag_mat(j,j-1) = -beta;
               tri_diag_mat(j-2,j-1) = -beta;
```

Solve the TDM obtained here to get Temp. values for the current column. Repeat this for all columns.

vi) ADI with relaxation:

Step 1: Declare all variables such as Δx , Δy , β , T.

Step 2: Create two 2D matrices T_half and T_array for storing the value of temperature at k and $k + 1/2$ iteration of size N × M where, M is number of grid points in x-direction and N is number of grid points in y-direction (here, M = 21, N = 41).

Create T_check which is of size N × M with all interior elements 0 and the boundary conditions applied. T_check will have elements at $k + 1$ iteration at the end of a for loop.

Step 3: Apply Boundary Conditions

```
T_array(1,:) = T_bottom;
T_array(:,1) = T_left;
T_array(:, end) = T_right;
T_array(end, :) = T_top;
```

Step 4: Form Tridiagonal Matrix using row sweep for each row. Solve this to get T at $k + 1/2$ iteration. Now sweep along the columns. Use Temperature values obtained at $k + 1/2$ iteration to form the Tridiagonal Matrix and solve for each column to get T $k + 1$. Code exactly the same as ADI but with the relaxation parameter.

RHS value for row sweep. Includes the relaxation parameter 'w'

```
rhs = transpose(beta*T_array(j+1, 2: n_x-1) + beta*T_half(j-1, 2: n_x-1) +
(2*(1-w)*(1+beta)*T_array(j, 2: n_x-1)/w));
```

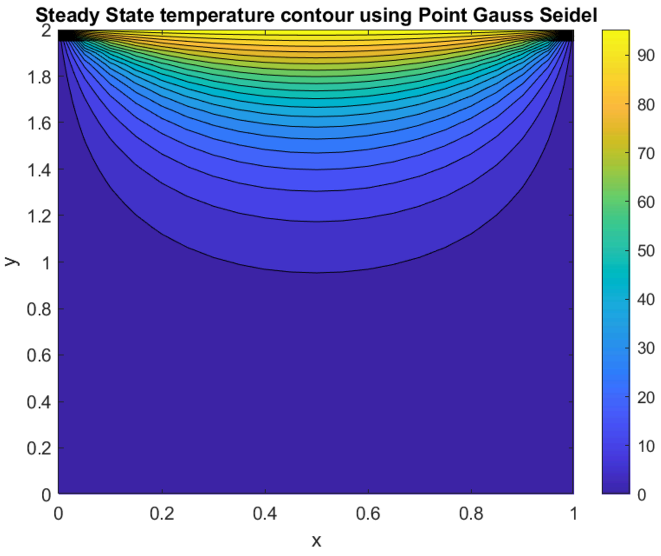
RHS value for column sweep. Includes the relaxation parameter ‘w’

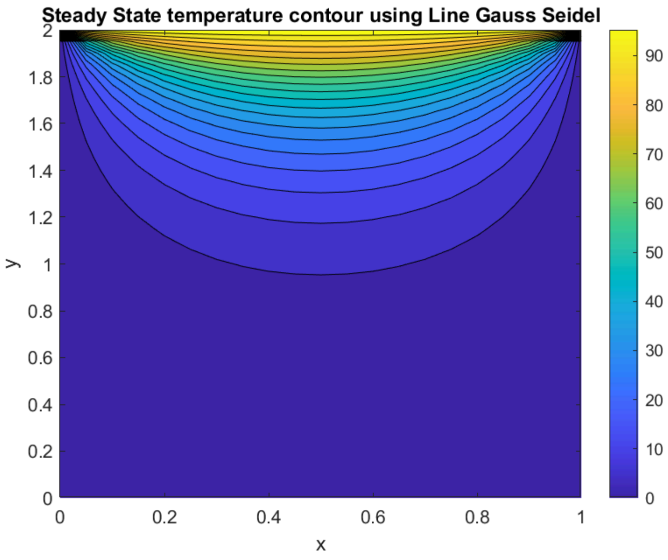
```
rhs = T_check(2: n_y-1, i-1) + T_half(2: n_y-1, i+1) +
(2*(1-w)*(1+beta)*T_half(2: n_y-1, i)/w);
```

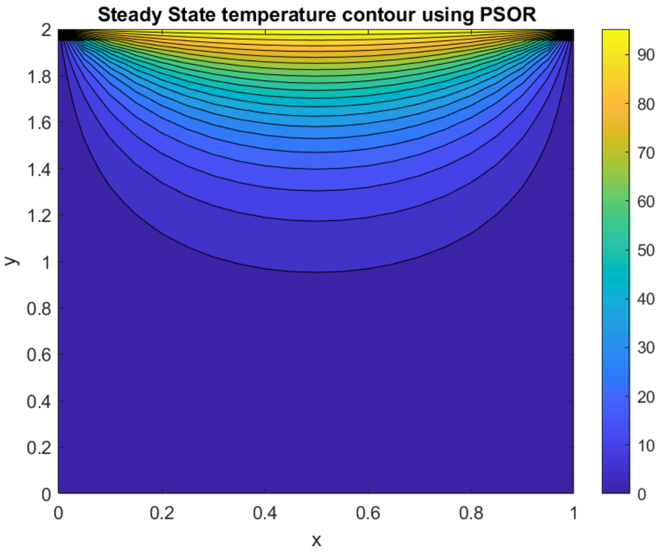
TABLE:

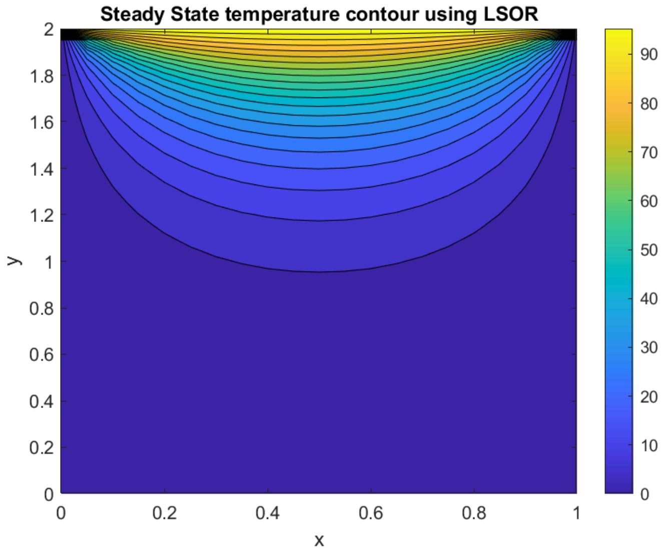
S No.	Scheme Used	Grid Size	omega(w)	No. of Iterations	CPU Time (s)	Computational Time (s)
1	Point Gauss Seidel	21 x 41	-	593	3.99	0.01
2	Line Gauss Seidel	21 x 41	-	327	11.15	0.06
3	PSOR	21 x 41	1.85	93	0.27	0.0
4	LSOR	21 x 41	1.25	76	1.19	0.03
5	ADI	21 x 41	-	174	5.00	0.06
6	ADI with relaxation	21 x 41	1.3	28	0.55	0.03

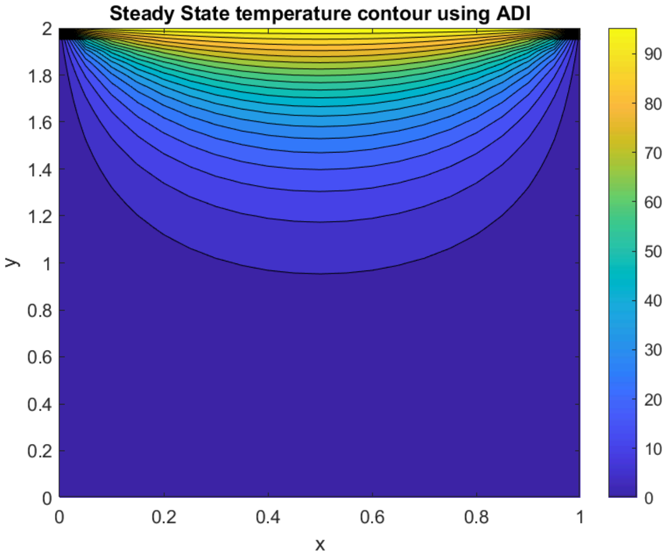
6. Results:

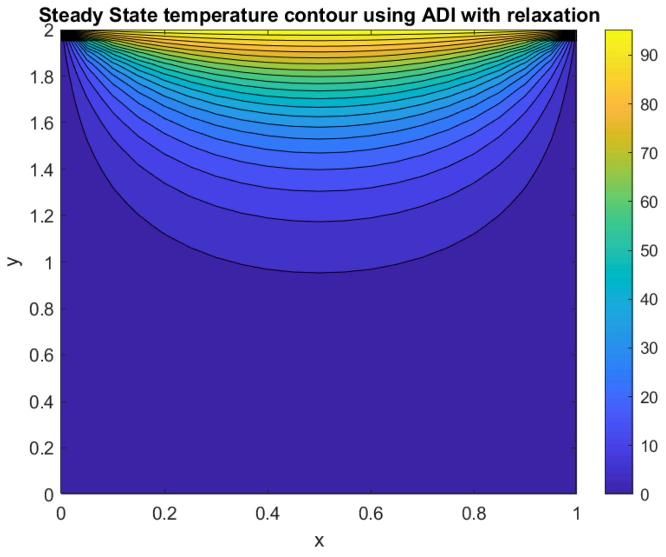

Steady State temperature contour using Point Gauss Seidel


Steady State temperature contour using Line Gauss Seidel


Steady State temperature contour using PSOR


Steady State temperature contour using LSOR


Steady State temperature contour using ADI


Steady State temperature contour using ADI with relaxation

Discussion:

1) ADI with relaxation converged in least number of iterations - 28
2) Point Gauss Seidel converged in highest number of iterations - 593
3) Adding relaxation helps the algorithm to converge faster
4) Line Gauss seidel has the highest CPU time.
5) PSOR has the lowest CPU time.

Computer Code:

S No.	Scheme Used	Name of the Matlab File
1	Point Gauss Seidel	Point_GS.mlx
2	Line Gauss Seidel	Line_GS.mlx
3	PSOR	PSOR.mlx
4	LSOR	LSOR.mlx
5	ADI	ADI.mlx
6	ADI with relaxation	ADI_relax.mlx
7	Tridiagonal Matrix Algorithm	tdma.mlx
8	Running all the codes	Main_A2.mlx

—--end—---