ME426

Mini Project Report

Comparison of 3 schemes for unsteady heat transfer in a 2d plate with Dirichlect boundary conditions

Submitted to: Dr Ranjith M.

By: Achintya Thantry
Roll no: 17ME103

Introduction:

Steady state heat transfer has been simulated and time for reaching the steady state is compared between the time schemes to get to the steady state result. The computational time is also calculated as well.The 3 popular time schemes that are used to model this process are explicit,implicit and Crank Nicolson methods. This project aims to compare the results between the 3 methods on a two dimensional plate that is imposed with Dirichlect boundary conditions on all sides and simulate the process until steady state is achieved.

Heat Equation:

The heat equation for conduction through any material is given by:

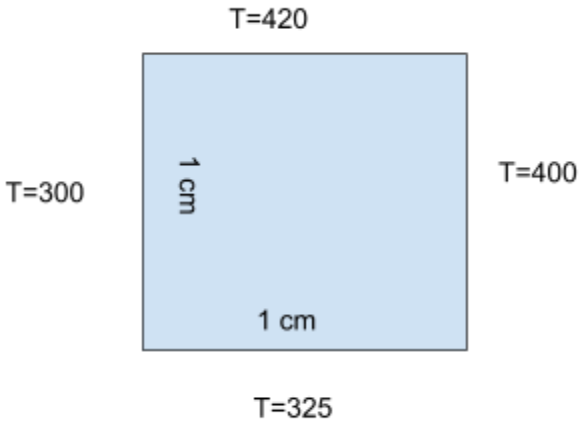
$$\frac{\partial T}{\partial t} = \alpha\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y 2}\right)$$

where α is the thermal diffusivity of the material = $\frac{k}{\rho c}$

k = thermal conductivity
ρ = density
c = specific heat capacity

Model:

The model considered for simulation is described below. Dirichlet boundary conditions have been imposed on all the four walls of a square of size (1cmx1cm). The thermal diffusivity of aluminium is used for the simulation. α =0.97 cm^2 /s.



Discretization for different schemes:

As mentioned previously, the aim of this project is to compare the results obtained between the 3 schemes. First we will discretize the governing equation using the explicit scheme and then proceed to do the same using the implicit and Crank Nicolson schemes.
The method of indexing are as follows:
¡- along the x direction
j- along the y direction
n- along the time axis

The discretization using the explicit scheme is as follows:

$$\frac{\partial T}{\partial t} = \frac{T_{i,j}^{n+1} - T_{i,j}^{n}}{\Delta t}$$

$$\frac{\partial^2 T}{\partial x^2} = \frac{T_{i+1,j}^{n} + T_{i-1,j}^{n} - 2T_{i,j}^{n}}{\Delta x^2}$$

$$\frac{\partial^2 T}{\partial y^2} = \frac{T_{i,j+1}^{n} + T_{i,j-1}^{n} - 2T_{i,j}^{n}}{\Delta y^2}$$

$$T_{i,j}^{n+1} = \frac{(1-4\lambda)T_{i,j}^{n} + \lambda(T_{i+1,j}^{n} + T_{i-1,j}^{n} + T_{i,j+1}^{n} + T_{i,j-1}^{n})}{(1+4\lambda)}$$

It is to be noted that for a stable solution:

$$\lambda = \frac{\alpha \Delta t}{\Delta x^2} \leq \frac{1}{4}$$

The LHS discretization of (1) does not change among all methods, it is only the RHS that changes as we use different schemes. A drawback of this scheme is that the time and space discretization is restricted.. This implies that as we increase the grid size, we have to increase the time steps by a lot to get a stable solution.This is not desired as it increases computational time involved.The implicit methods get rid of this problem.

For the implicit method the RHS discretization of terms is:

$$\frac{\partial^2 T}{\partial x^2} = \frac{T_{i+1,j}^{n+1} + T_{i-1,j}^{n+1} - 2T_{i,j}^{n+1}}{\Delta x^2}$$

$$\frac{\partial^2 T}{\partial y^2} = \frac{T_{i,j+1}^{n+1} + T_{i,j-1}^{n+1} - 2T_{i,j}^{n+1}}{\Delta y^2}$$

Using this method of discretization there is one temperature value from the previous time step and 5 temperature values of the current time step. The equation is rewritten with $T_{i,j}^{n+1}$ as unknown and it is shown below:

$$T_{i,j}^{n+1} = \frac{T_{i,j}^{n} + \lambda(T_{i+1,j}^{n+1} + T_{i-1,j}^{n+1} + T_{i,j+1}^{n+1} + T_{i,j-1}^{n+1})}{1+4\lambda}$$

The $T_{i,j}^{n+1}$ values are constantly updated within an iteration to use the latest value of temperature available at the (n+1)th time step. The limitation on value of λ is lifted in this method and thus time is saved. The same applies to the Crank Nicolson scheme as well.

The Crank Nicolson method is a combination of both the explicit and implicit schemes and take the average value of $T_{i,j}$ at the (n+1)th and nth time levels. The discretization is as follows:

$$\frac{\partial^2 T}{\partial y^2} = \frac{1}{2}\left(\frac{T_{i,j+1}^{n+1} + T_{i,j-1}^{n+1} - 2T_{i,j}^{n+1}}{\Delta y^2} + \frac{T_{i,j+1}^{n} + T_{i,j-1}^{n} - 2T_{i,j}^{n}}{\Delta y^2} \right)$$

$$\frac{\partial^2 T}{\partial x^2} = \frac{1}{2}\left(\frac{T_{i+1,j}^{n+1} + T_{i-1,j}^{n+1} - 2T_{i,j}^{n+1}}{\Delta x^2} + \frac{T_{i+1,j}^{n} + T_{i-1,j}^{n} - 2T_{i,j}^{n}}{\Delta x^2} \right)$$

On rearranging the terms once again such that $T_{i,j}^{n+1}$ is on the LHS the equation is as:

$$T_{i,j}^{n+1} = \frac{(1-2\lambda)T_{i,j}^{n} + \frac{\lambda}{2}(T_{i+1,j}^{n+1} + T_{i-1,j}^{n+1} + T_{i,j+1}^{n+1} + T_{i,j-1}^{n+1} + T_{i+1,j}^{n} + T_{i-1,j}^{n} + T_{i,j+1}^{n} + T_{i,j-1}^{n})}{(1+2\lambda)}$$

Results and Figures:

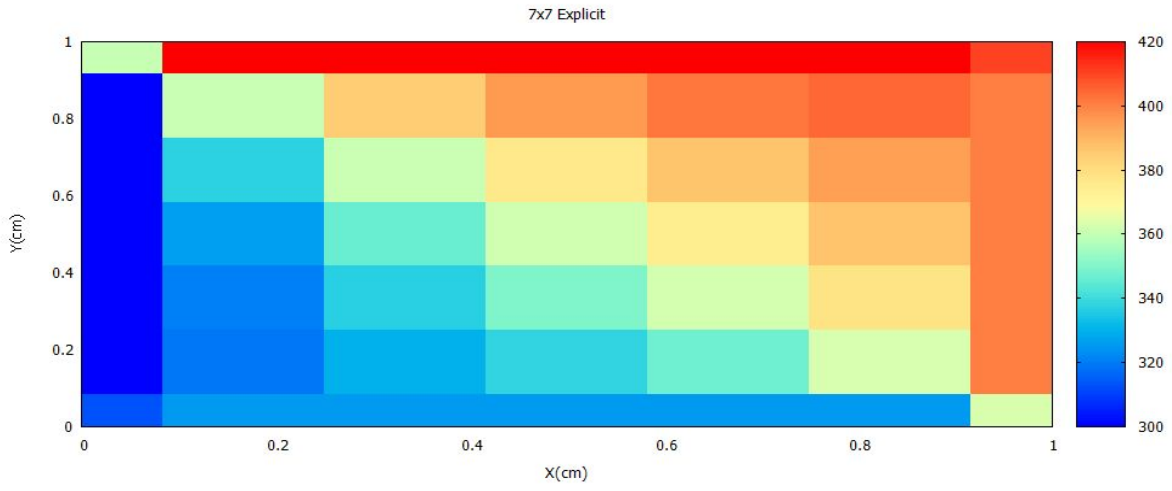
Grid independent studies have been carried out for all 3 methods and figures obtained are compared.The parameters taken constant are:
$\alpha = 0.97$, $\Delta t = 0.000533$, total time=3.0 seconds

For constant $\Delta t = 0.000533s$ the grid sizes are only changed and the results when $t = 3s$ is observed. The time taken and total iterations are as follows:

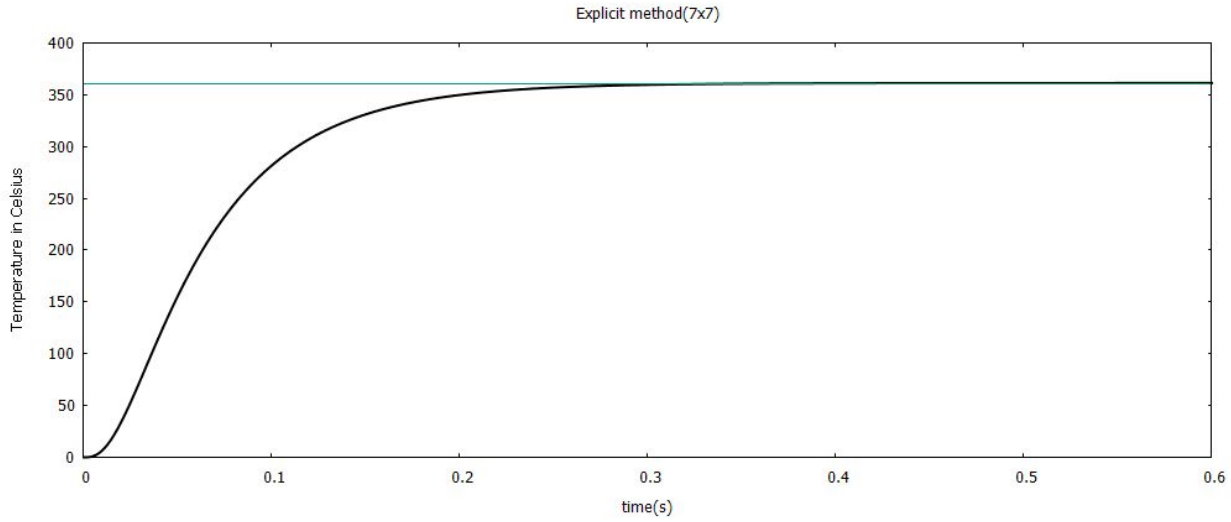
	Iterations			Time Taken(seconds)		
No. of nodes	Explicit	Implicit	Crank-Nicolson	Explicit	Implicit	Crank-Nicolson
7	57400	103902	103958	1	3	1
25	NA	102709	106086	NA	74	50
50	NA	110128	141639	NA	304	316
75	NA	115219	168698	NA	460	873
100	NA	107785	183175	NA	546	1575

It was observed that in the explicit method when the number of nodes were 25 and greater, unreasonable values were obtained which were shown as NaN values. This reinforces the fact of the limitation of the stability of the explicit method at lower grid sizes. The graphs obtained for various grid sizes are shown below:

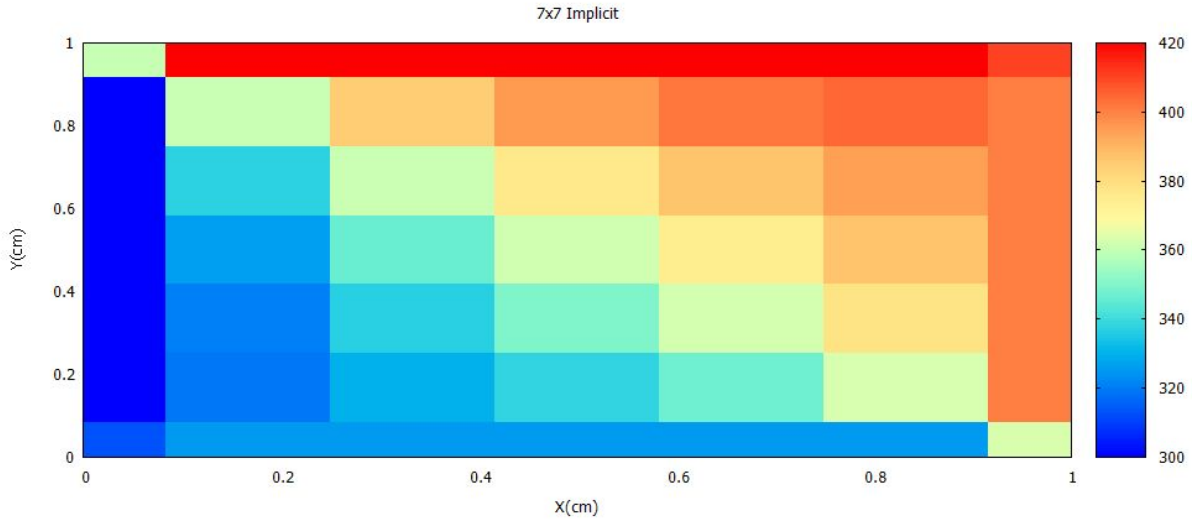
Explicit Scheme:

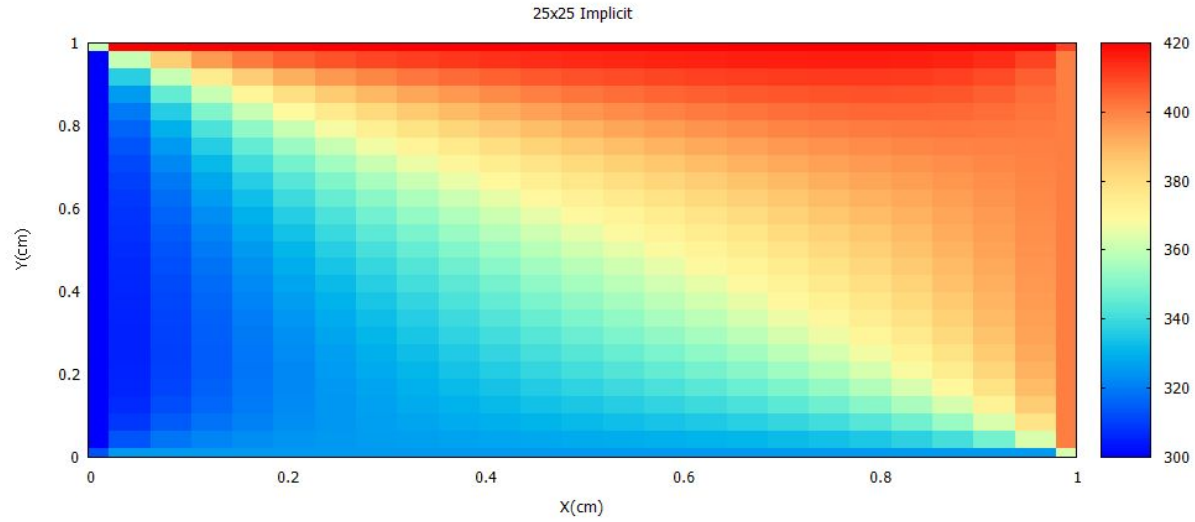


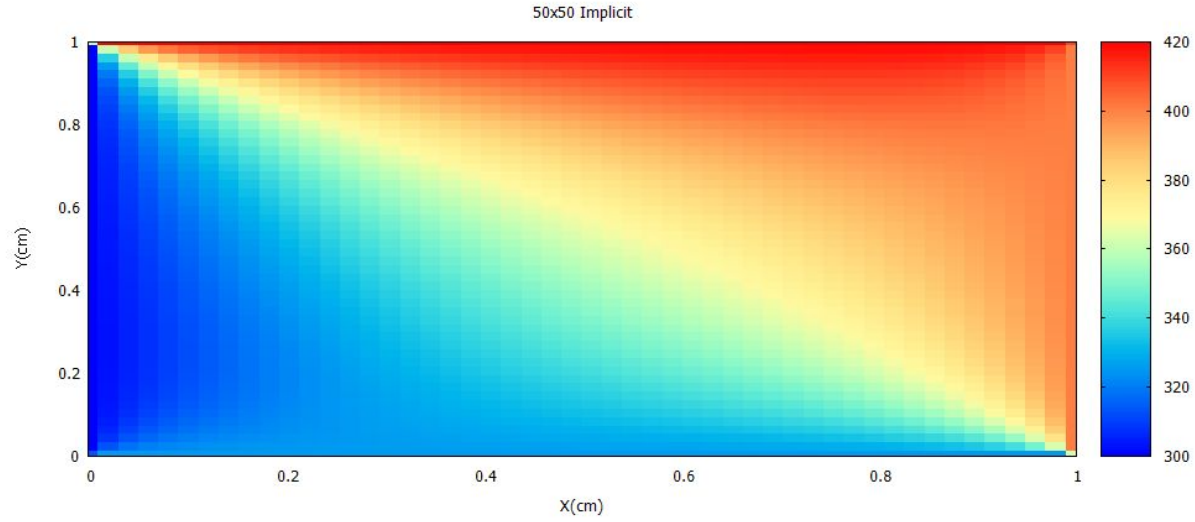
As grids are refined more,ie, grid size is decreased, the value becomes invalid thereby proving the fact that the grid size is restricted by the size of the time step and vice versa.
Variation of temperature at centre with time:

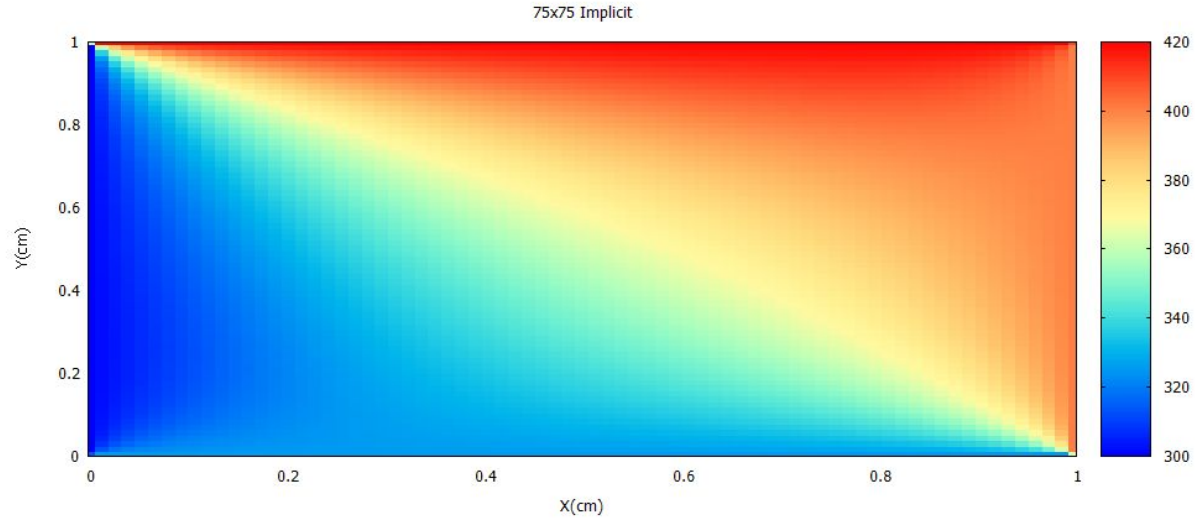


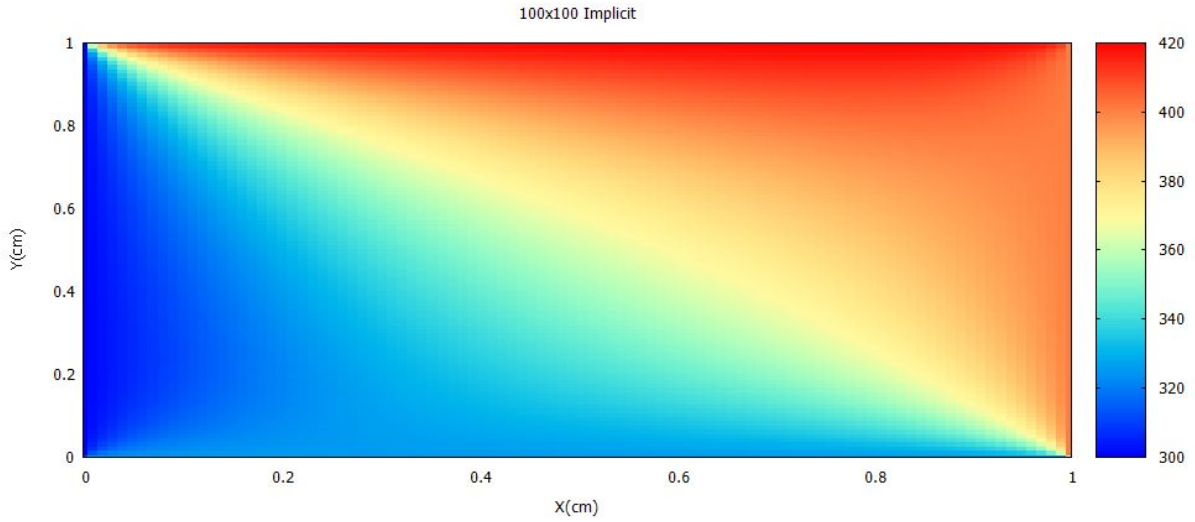
Implicit Scheme:



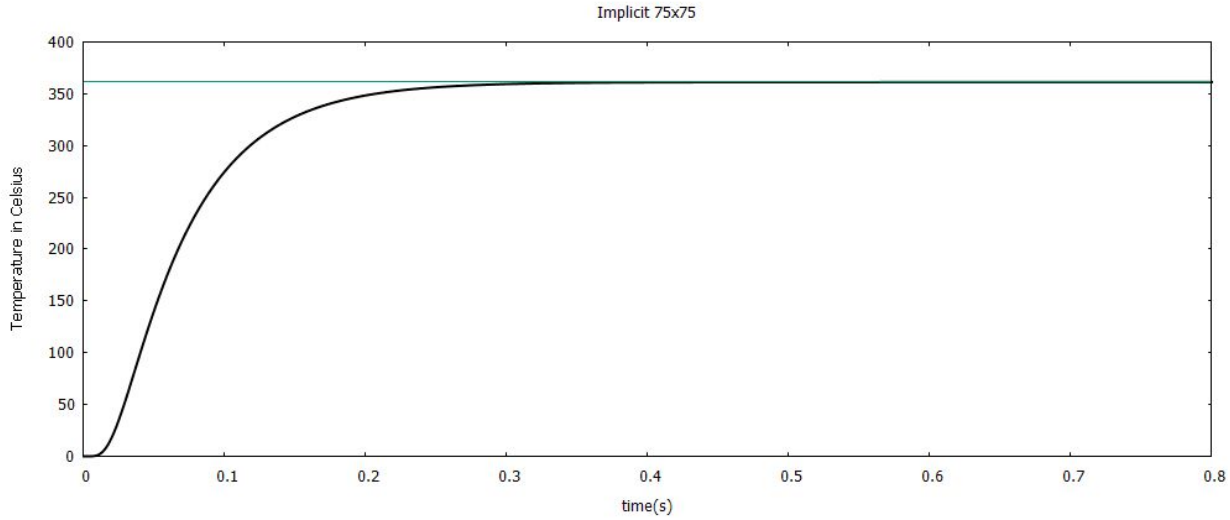

25x25 Implicit
Y(cm)
X(cm)
0
0.2
0.4
0.6
0.8
1
300
320
340
360
380
400
420

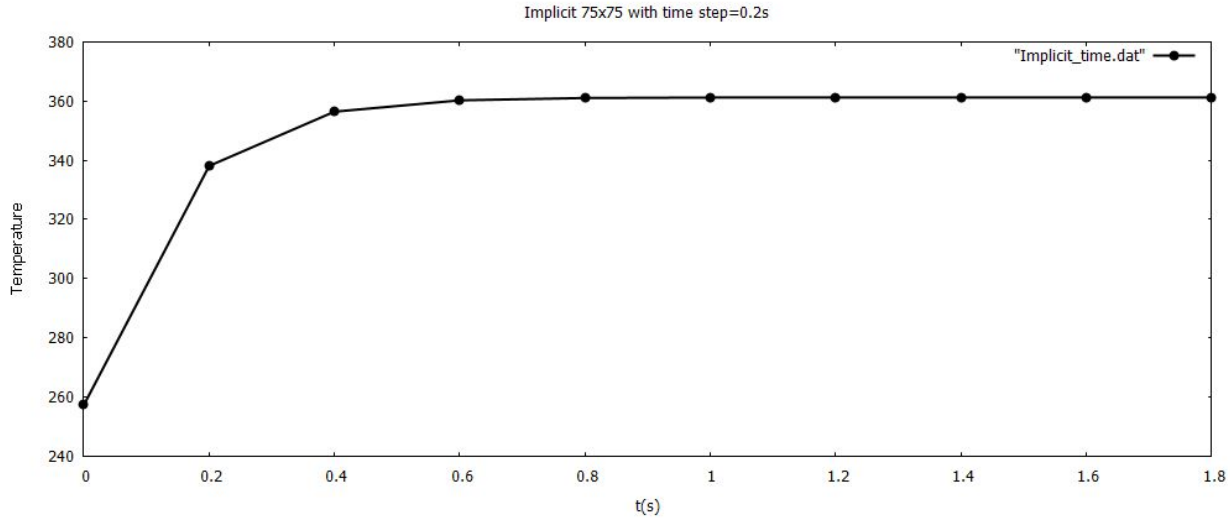

50x50 Implicit
Y(cm)
X(cm)
0
0.2
0.4
0.6
0.8
1
300
320
340
360
380
400
420


75x75 Implicit
Y(cm)
X(cm)
0
0.2
0.4
0.6
0.8
1
300
320
340
360
380
400
420



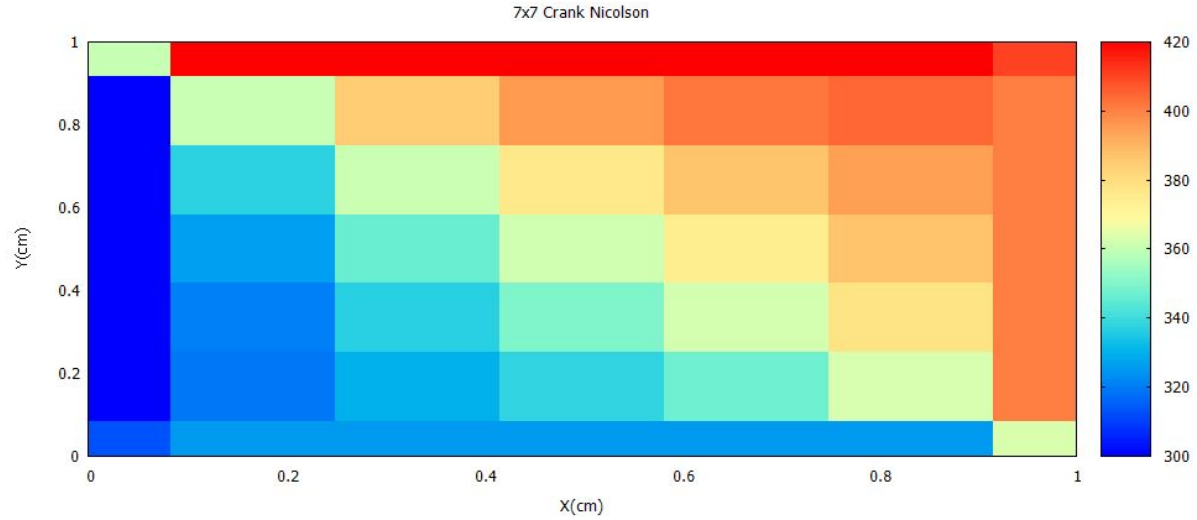
Variation of temperature at centre with time:

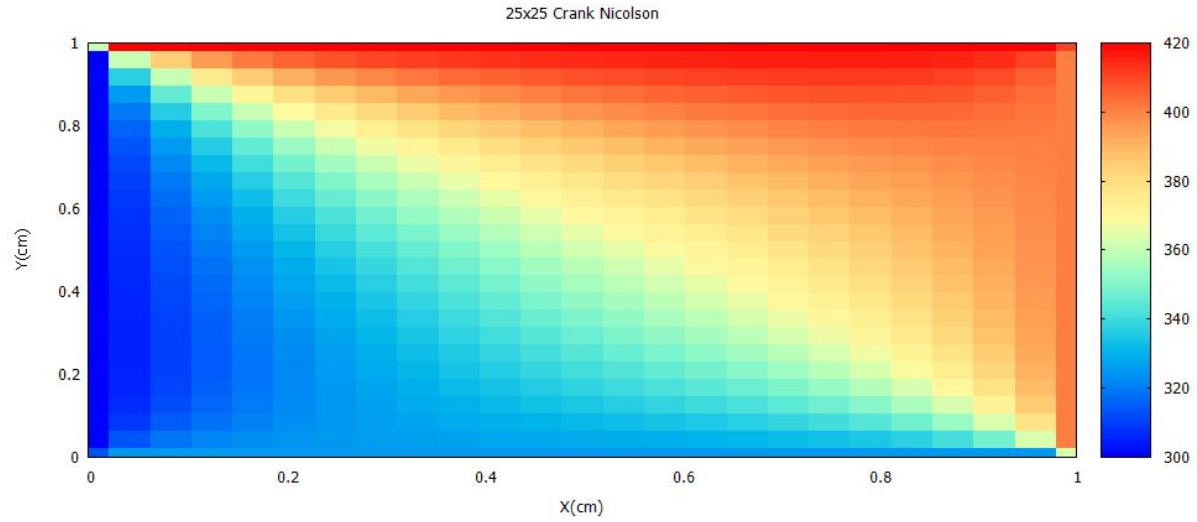


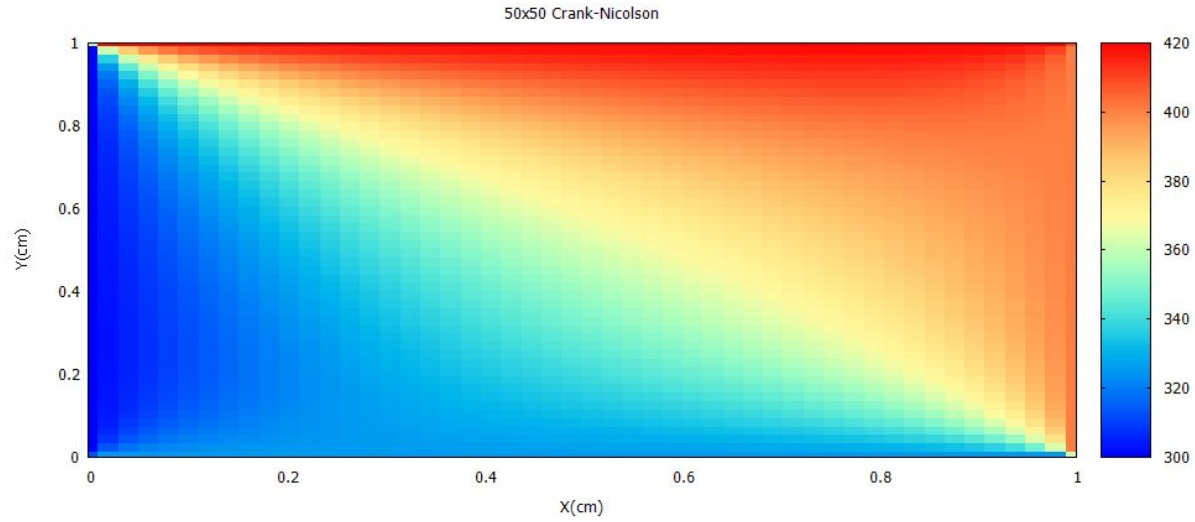


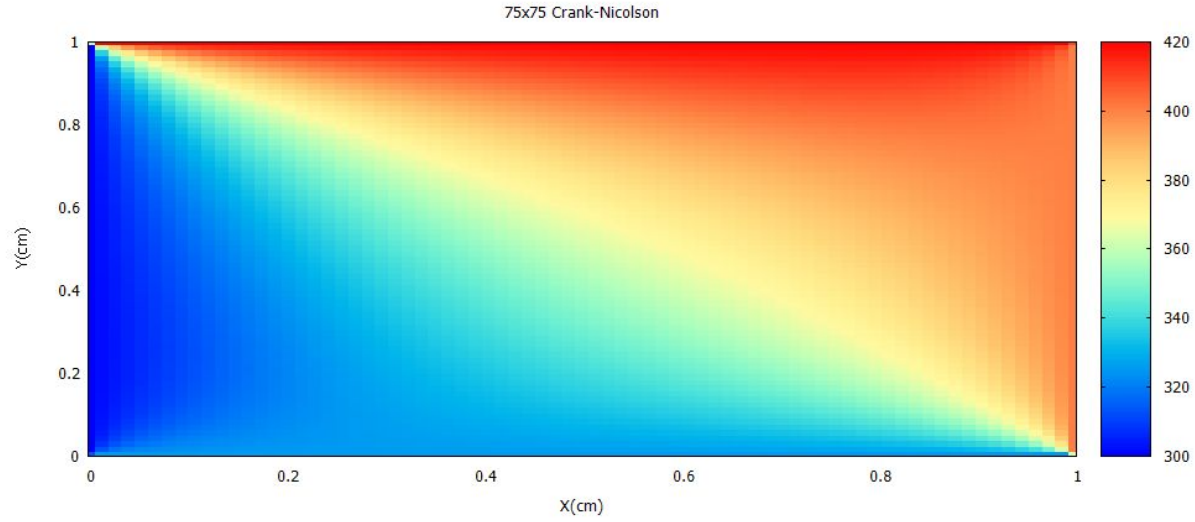
As seen in above figures, the resolution is improved incrementally and using the table mentioned before the best grid size for the implicit method has been chosen as 75x75 here.

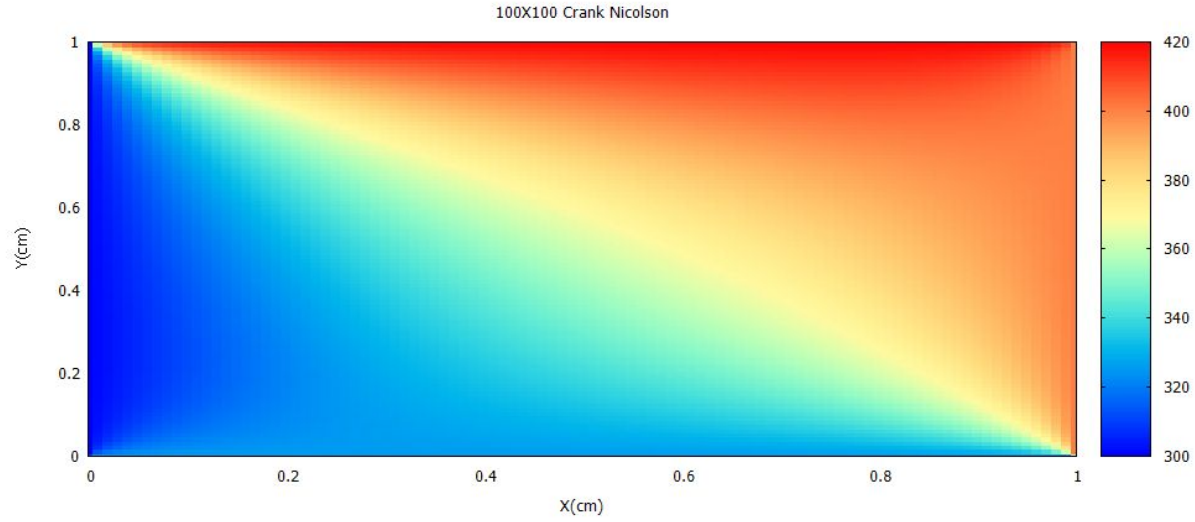
Crank Nicolson:

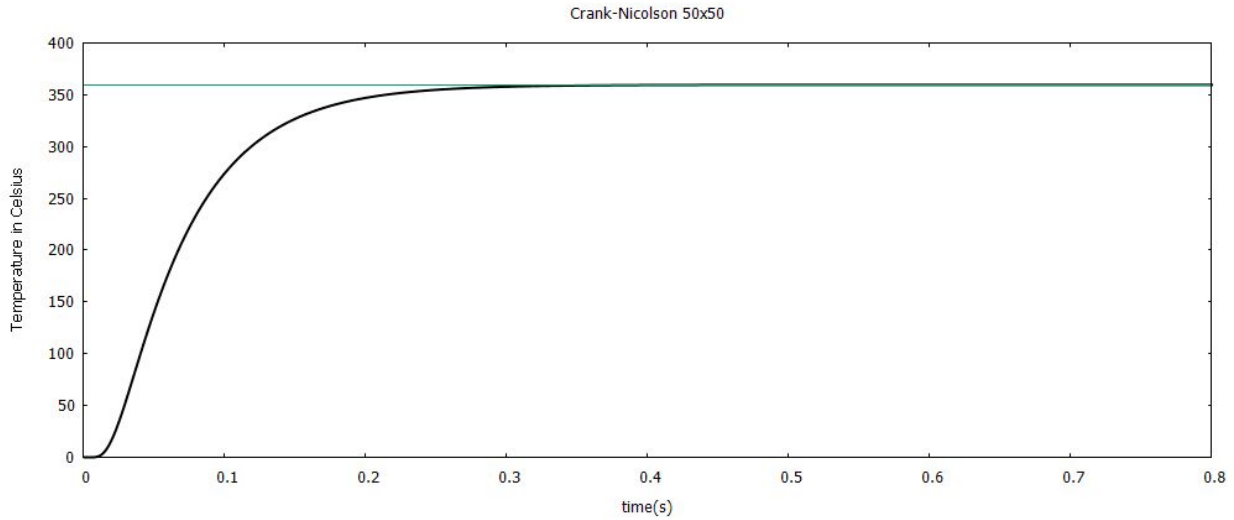





50x50 Crank-Nicolson
Y(cm)
X(cm)
0
0.2
0.4
0.6
0.8
1
300
320
340
360
380
400
420


75x75 Crank-Nicolson
Y(cm)
X(cm)
0
0.2
0.4
0.6
0.8
1
300
320
340
360
380
400
420


100X100 Crank Nicolson
Y(cm)
X(cm)
0
0.2
0.4
0.6
0.8
1
300
320
340
360
380
400
420



A grid size of 75x75 seems best suited for implementing this scheme after looking at the increase in time needed for computing the values at a grid size of 100x100 and the minimal difference in resolution involved.

Looking at the variation of temperature at the centre of the plate for all the three schemes, the explicit method seems to reach a steady state slightly earlier than the other schemes at a time of approximately 0.55s compared to 0.6 in the other 2 schemes. If we are to get any results using the explicit scheme we will have to decrease the time step thereby unnecessarily increasing the computation needed.

Validation:

The value at the centre was compared to the value obtained by Shyla et al. The values obtained through the code developed are compared with the paper as:

Explicit	Implicit	Crank-Nicolson	Shya et al
361.25	361.25	361.249	360.75

Conclusion:

To conclude,this project has implemented the 3 numerical schemes for unsteady heat conduction in a solid and satisfactory results have been obtained. Simulations have been carried out for different grid sizes and a grid size of 75x75 was found to be most favourable for the current system in use. The steady state graphs have been plotted and it was seen that the explicit method attains steady state slightly faster than the other two methods. The limitation of

grid size and time steps are not present for the implicit or Crank-Nicolson methods giving us complete control on modelling the simulation according to our objective.

References:

Shyla, D.M., Bai, D.H., & Jeyanthi, M.P. (2018). Finite Difference Solutions of Heat Conduction Problem with Dirichlet and Neumann Boundary Conditions.

http://people.eecs.berkeley.edu/~demmel/cs267/lecture17/lecture17.html#link_1.4

Appendix:

3 different codes were written for each one of the time schemes.

1) Explicit Scheme

```
#include <iostream>
#include<vector>
#include<fstream>
#include<math.h>
#include<time.h>
#include<iomanip>

using namespace std;

int main()
{
      clock_t ct;

      int nx, ny;
      double dx, dy, dt;
      dt = 0.0005333;
      double ts = 0.777 / dt;

      nx = ny = 7;
      dx = 1 / (double(nx) - 1);
      dy = 1 / (double(ny) - 1);

      double alpha = 1.0;
      double lambda = alpha * dt / pow(dx, 2.0);
```

```
	cout << lambda << endl;
	double w = 1.9;


	vector<vector<double>> Tg(ny, vector<double>(nx)), Tn(ny,
vector<double>(nx)), To(ny, vector<double>(nx));

	//initializing
	for (int i = 0;i < ny;i++)
	{
		for (int j = 0;j < nx;j++)
		{
			To[i][j] = Tn[i][j] = Tg[i][j] = 0.0;
		}
	}


	//boundary values
	//left
	for (int i = 0;i < ny;i++)
	{
		Tg[i][0] = Tn[i][0] = 300.0;
	}
	//top
	for (int j = 0;j < nx;j++)
	{
		Tg[ny - 1][j] = Tn[ny - 1][j] = 420.0;
	}
	//right
	for (int i = 0;i < ny;i++)
	{
		Tg[i][nx - 1] = Tn[i][nx - 1] = 400.0;
	}
	//bottom
	for (int j = 0;j < nx;j++)
	{
		Tg[0][j] = Tn[0][j] = 325.0;
	}
	//corners
	Tn[0][0] = Tg[0][0] = double(625.0 / 2);
	Tn[ny - 1][0] = Tg[ny - 1][0] = double(720 / 2);
	Tn[ny - 1][nx - 1] = Tg[ny - 1][nx - 1] = 410.0;
	Tn[0][nx - 1] = Tg[0][nx - 1] = double(725.0 / 2);

	double errm = 0.0, err;
```

```
	fstream tfile;

tfile.open("C:/Users/ACHINTYA/Desktop/Mini_Project/Explicit_time.dat", ios::out);
	ct = clock();
	int g = 0;
	//time loop here
	for (int t = 0;t < ts;t++)
	{
		for (int i = 0;i < ny;i++)
		{
			for (int j = 0;j < nx;j++)
			{
				To[i][j] = Tn[i][j];
			}
		}
		//interior nodes GS SOR
		do
		{
			err = 0.0;
			errm = 0.000001;
			for (int i = 1;i < ny - 1;i++)
			{
				for (int j = 1;j < nx - 1;j++)
				{
					Tn[i][j] = (To[i][j] * (1 - (4 * lambda))) + (lambda * (To[i][j + 1] + To[i + 1][j] + To[i - 1][j] + To[i][j - 1]));
					Tn[i][j] = (Tn[i][j] * w) + ((1 - w) * Tg[i][j]);
					err = abs(Tn[i][j] - Tg[i][j]);
					if (err > errm)
						errm = err;
					Tg[i][j] = Tn[i][j];
				}
			}

			g++;
		} while (errm >= 0.00001);


		tfile << t * dt << " " << Tn[(ny - 1) / 2][(nx - 1) / 2]<<endl;
		//end of time loop
	}
```

```
	tfile.close();
	fstream ofile;

ofile.open("C:/Users/ACHINTYA/Desktop/Mini_Project/TrialE.csv",
ios::out);

	for (int i = 0;i < ny;i++)
	{
		for (int j = 0;j < nx;j++)
		{
			ofile << Tn[i][j] << ",";
		}
		ofile << endl;
	}
	ct = clock() - ct;
	cout << setprecision(5)<<Tn[(ny - 1) / 2][(nx - 1) / 2] <<
endl;

	cout << "Time is " << double(ct / CLOCKS_PER_SEC) << " and
iterations= " << g << endl;
	ofile.close();
	//putting in a dat file as well
	fstream op;

op.open("C:/Users/ACHINTYA/Desktop/Mini_Project/TrialE.dat",
ios::out);
	for (int j = 0;j < nx;j++)
	{
		for (int i = 0;i < ny;i++)
		{
			double x = j * dx;
			double y = i * dy;
			op << x << " " << y << " " << Tn[i][j];
			op << endl;
		}
		op << endl;
	}
	op.close();
}
```

2) Implicit Scheme

```
#include<iostream>
#include<vector>
#include<math.h>
#include<time.h>
#include<fstream>

using namespace std;

int main()
{
	clock_t ct;
	fstream tfile;

tfile.open("C:/Users/ACHINTYA/Desktop/Mini_Project/Implicit_time.dat"
, ios::out);
	int nx, ny;
	double dx, dy, dt;
	dt = 0.2;
	double ts = 2.0 / dt;
	double alpha = 1.0;
	nx = ny = 7;
	dx = 1 / (double(nx) - 1);
	dy = 1 / (double(ny) - 1);


	double lambda = alpha*dt / pow(dx, 2.0);
	cout << lambda << endl;
	double w = 1.9;


	vector<vector<double>> Tg(ny, vector<double>(nx)), Tn(ny,
vector<double>(nx)), To(ny, vector<double>(nx));
	vector<vector<double>> b(nx, vector<double>(nx));

	//initializing
	for (int i = 0;i < ny;i++)
	{
		for (int j = 0;j < nx;j++)
		{
			To[i][j] = Tn[i][j] = Tg[i][j] = 0.0;
		}
```

```
}


//boundary values
//left
for (int i = 0;i < ny;i++)
{
     Tg[i][0] = Tn[i][0] = 300.0;
}
//top
for (int j = 0;j < nx;j++)
{
     Tg[ny - 1][j] = Tn[ny - 1][j] = 420.0;
}
//right
for (int i = 0;i < ny;i++)
{
     Tg[i][nx - 1] = Tn[i][nx - 1] = 400.0;
}
//bottom
for (int j = 0;j < nx;j++)
{
     Tg[0][j] = Tn[0][j] = 325.0;
}
//corners
Tn[0][0] = Tg[0][0] = double(625.0 / 2);
Tn[ny - 1][0] = Tg[ny - 1][0] = double(720 / 2);
Tn[ny - 1][nx - 1] = Tg[ny - 1][nx - 1] = 410.0;
Tn[0][nx - 1] = Tg[0][nx - 1] = double(725.0 / 2);

double errm = 0.0, err;

for (int i = 0;i < ny;i++)
{
     for (int j = 0;j < nx;j++)
     {
          b[i][j] = Tn[i][j];
     }
}

ct = clock();

int g = 0;//iteration counter
//time loop starts
     ct = clock() - ct;for (int t = 0;t < ts;t++)
```

```
	{
		for (int i = 0;i < ny;i++)
		{
			for (int j = 0;j < nx;j++)
			{
				To[i][j] = Tn[i][j];//updating for next time
			}
		}


		//GS SOR iteration for each time step
		do
		{



			err = 0.0;
			errm = 0.000001;
			for (int i = 1;i < ny - 1;i++)
			{

				for (int j = 1;j < nx - 1;j++)
				{
					b[i][j] = Tn[i + 1][j] + Tn[i - 1][j] +
Tn[i][j + 1] + Tn[i][j - 1];//updating b after each iteration
					Tn[i][j] = (To[i][j] + (lambda * b[i][j]))
/ (1 + (4 * lambda));
					Tn[i][j] = (Tn[i][j] * w) + ((1 - w) *
Tg[i][j]);
					err = abs(Tn[i][j] - Tg[i][j]);
					if (err > errm)
						errm = err;
					Tg[i][j] = Tn[i][j];//for GS previous
values for next iteration
				}
			}

			g++;
		} while (errm > 0.000001);

		tfile << t * dt << " " << Tn[(ny - 1) / 2][(nx - 1) /
2]<<endl;

	}
	tfile.close();
```

```
	fstream ofile;

ofile.open("C:/Users/ACHINTYA/Desktop/Mini_Project/Trial_implicit.csv
", ios::out);
	//writing to a csv file
	for (int i = 0;i < ny;i++)
	{
		for (int j = 0;j < nx;j++)
		{
			ofile << Tn[i][j] << ",";
		}
		ofile << endl;
	}
	ofile.close();
	//putting in a dat file as well
	fstream op;
	op.open("C:/Users/ACHINTYA/Desktop/Mini_Project/TrialI.dat",
ios::out);
	for (int j = 0;j < nx;j++)
	{
		for (int i = 0;i < ny;i++)
		{
			double x = j * dx;
			double y = i * dy;
			op << x << " " << y << " " << Tn[i][j];
			op << endl;
		}
		op << endl;
	}
	op.close();
	cout << endl << Tn[(ny - 1) / 2][(nx - 1) / 2];
	//showing some results
	cout << "Iteration:" << g << " Time: " << double(ct /
CLOCKS_PER_SEC);
	return 0;
}
```

3) Crank-Nicolson Scheme

```
#include<iostream>
#include<vector>
#include<math.h>
#include<time.h>
```

```
#include<fstream>

using namespace std;

int main()
{
	clock_t ct;

	int nx, ny;
	double dx, dy, dt;
	dt = 0.000533;
	double ts = 0.777 / dt;
	double alpha = 1.0;
	nx = ny = 7;
	dx = 1 / (double(nx)-1);
	dy = 1 / (double(ny)-1);


	double lambda = alpha*dt / pow(dx, 2.0);

	cout << lambda << endl;
	double w = 1.9;


	vector<vector<double>> Tg(ny, vector<double>(nx)), Tn(ny,
vector<double>(nx)), To(ny, vector<double>(nx));
	vector<vector<double>> b(nx, vector<double>(nx));

	//initializing
	for (int i = 0;i < ny;i++)
	{
		for (int j = 0;j < nx;j++)
		{
			To[i][j] = Tn[i][j] = Tg[i][j] = 0.0;
		}
	}


	//boundary values
	//left
	for (int i = 0;i < ny;i++)
	{
		Tg[i][0] = Tn[i][0] = 300.0;
	}
	//top
```

```
	for (int j = 0;j < nx;j++)
	{
		Tg[ny -1][j] = Tn[ny - 1][j] = 420.0;
	}
	//right
	for (int i = 0;i < ny;i++)
	{
		Tg[i][nx - 1] = Tn[i][nx - 1] = 400.0;
	}
	//bottom
	for (int j = 0;j < nx;j++)
	{
		Tg[0][j] = Tn[0][j] = 325.0;
	}
	//corners
	//Tn[0][0] = Tg[0][0] = double(625.0 / 2);
	//Tn[ny - 1][0] = Tg[ny - 1][0] = double(720 / 2);
	//Tn[ny - 1][nx - 1] = Tg[ny - 1][nx - 1] = 410.0;
	//Tn[0][nx - 1] = Tg[0][nx - 1] = double(725.0 / 2);

	double errm = 0.0, err;

	for (int i = 0;i < ny;i++)
	{
		for (int j = 0;j < nx;j++)
		{
			b[i][j] = Tn[i][j];
		}
	}

	ct = clock();

	fstream tfile;

tfile.open("C:/Users/ACHINTYA/Desktop/Mini_Project/Crank_nicolson_tim
e.dat", ios::out);

	int g = 0;//iteration counter
	//time loop starts
	for (int t = 0;t < ts;t++)
	{
		for (int i = 0;i < ny;i++)
		{
			for (int j = 0;j < nx;j++)
			{
```

```
                        To[i][j] = Tn[i][j];//updating for next time
                  }
            }


            //GS SOR iteration for each time step
            do
            {



                  err = 0.0;
                  errm = 0.000001;
                  for (int i = 1;i < ny - 1;i++)
                  {

                        for (int j = 1;j < nx - 1;j++)
                        {
                              b[i][j] = Tn[i + 1][j] + Tn[i - 1][j] +
Tn[i][j + 1] + Tn[i][j -
1]+To[i][j+1]+To[i][j-1]+To[i-1][j]+To[i+1][j];//updating b after
each iteration
                              Tn[i][j] = (((1 - 2 * lambda) * To[i][j])
+ (lambda * b[i][j] / 2)) / (1 + (2 * lambda));
                              Tn[i][j] = (Tn[i][j] * w) + ((1 - w) *
Tg[i][j]);
                              err = abs(Tn[i][j] - Tg[i][j]);
                              if (err > errm)
                                    errm = err;
                              Tg[i][j] = Tn[i][j];//for GS previous
values for next iteration
                        }
                  }
                  g++;

            } while (errm > 0.00001);

            tfile << t * dt << " " << Tn[(ny - 1) / 2][(nx - 1) / 2]
<< endl;
      }

      tfile.close();

      fstream ofile;
```

```
ofile.open("C:/Users/ACHINTYA/Desktop/Mini_Project/Trial_crank_nicols
on.csv", ios::out);
	//end of time loop
	ct = clock() - ct;

	//writing to a csv file
	for (int i = 0;i < ny;i++)
	{
		for (int j = 0;j < nx;j++)
		{
			ofile << Tn[i][j] << ",";
		}
		ofile << endl;
	}
	ofile.close();
	cout << "\n" << Tn[(ny-1) / 2][(nx-1) / 2];

	cout << "\nIterations= " << g << " time= " << double(ct /
CLOCKS_PER_SEC);
	//putting in a dat file as well
	fstream op;
	op.open("C:/Users/ACHINTYA/Desktop/Mini_Project/TrialCN.dat",
ios::out);
	for (int j = 0;j < nx;j++)
	{
		for (int i = 0;i < ny;i++)
		{
			double x = j * dx;
			double y = i * dy;
			op << x << " " << y << " " << Tn[i][j];
			op << endl;
		}
		op << endl;
	}
	op.close();
}
```