Finite Element Method For Transient Conduction.

A Project Report
Submitted in partial fulfillment of the requirements of

ME F366: Laboratory Project

By

Karnati Venkata Kartheek

ID No.2014A4TS0260H

Under the supervision of

Dr.Rama Chandra Murthy K

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

HYDERABAD CAMPUS

April 18, 2018

Acknowledgements

I express my sincere gratitude and deep sense of respect to my mentor Dr.Rama Chandra Murthy K, for standing by my side all through the project and also for his constant motivation, guidance, encouragement and inspiration which helped me immensely, both professionally and personally.

I am also thankful to Mr.Srikanth, for his valueble suggestions, inspiring guidance and encouragement throughout the project.

Nevertheless, I express my gratitude toward my family for their constant support and encouragement in completing this venture successfully.

CERTIFICATE

This is to certify that the project report entitled, " Application of the modified Kinetic Flux Vector Splitting (m-KFVS) method to practical configurations " and submitted by Karnati Venkata Kartheek (ID No. 2014A4TS0260H) in the partial fulfillment of the requirements of ME F366, Laboratory Project Course, embodies the work done by him under my supervision and guidance.

Dr. Rama Chandra Murthy K

Assistant Professor
(Dept of Mech Engg)

Date:

Abstract

Finite Element Methods though initially developed to study the structural problems were later applied to wide range of problems of continuous mechanics. Finite element based commercial packages like Abaqus & Ansys were developed to solve industrial problems. Open source finite element packages like ParMooN (developed at IISC,Bangalore) are developed to solve Partial Differential Equations of Fluid Flow. To understand the complexity involved in these commercial and open source codes and be able to write own custom code for the solving problems, an effort is made through this project to develop object oriented c++ code for transient heat conduction problem.

Contents

Chapter 1

A brief on Finite Element Method

A general problem involves finding the field variables such as pressure, temperature, displacement, stress etc over a region when initial and boundary conditions are given. As there are infinite points in any given domain, the field vector should have infinitely many values and the problem will be the one having infinite number of unknowns.

In the finite element method we divide the domain into several sub domains. We define interpolation functions for the field variables over the sub domain. These interpolation functions of field variables are the defined based on field variable values at nodal points. Nodal points are usually those points which connect the element to the surrounding elements. By this exercise we reduced the infinite number problem to a finite number problem i.e here we have to find only the field variable values at nodal points and then we can find the variable value at any interior point of the element by using interpolation functions.

The interpolation functions should not be chosen arbitrarily but has to satisfy the compatibility and completeness conditions. If we have derivatives up to $(r+1)^{th}$ order under the integrals in weighted residual method, then compatibility condition is that interpolation function should have C^r continuity at the interface. And the completeness condition requires that interpolation function should have C^{r+1} continuity within the element.

The weighted residual approach is used in which the governing equations are multiplied by interpolation functions and are integrated over the domain to obtain the the element equations. These equations are assembled to obtain system of equations for entire domain. After this we impose the boundary conditions and then solve the system of equations. Additional calculations can be done once we find the nodal field vector for example we can find heat flux value once we know temperature

profile.

Listing all the above steps

- Divide domain into smaller elements
- Select appropriate interpolation functions for the elements.
- Find the matrix equations of element using weighted residual method
- Assemble element equations
- Impose boundary conditions
- Solve system of equations

In the next chapters we present the implementation and the code.

Chapter 2

Implementation

The governing equation is given by

$$-\left(\frac{\partial q_x}{\partial t}+\frac{\partial q_y}{\partial t}\right)+Q=\rho c\frac{\partial T}{\partial t} \tag{2.1}$$

Here q_x,q_y are heat flow rates per unit area in the respective coordinate directions,ρ is the density and c is the specific heat. From fourier's law we have

$$q_x=-\left(k\frac{\partial T}{\partial x}\right) \qquad q_y=-\left(k\frac{\partial T}{\partial y}\right) \tag{2.2}$$

here k is the thermal conductivity for isotropic medium. Now from the method of weighted residuals we have

$$\int_\Omega\left(\frac{\partial q_x}{\partial t}+\frac{\partial q_y}{\partial t}-Q+\rho c\frac{\partial T}{\partial t}\right)N_i d\Omega=0 \tag{2.3}$$

Here Ω is the volume of the element, N_i is the value of interpolating polynomial at Nodal point i and i = 1,2,3,...
By using Gauss's theorem we can introduce surface integrals and the equation becomes

$$\int_\Omega \rho c\frac{\partial T}{\partial t}N_i d\Omega-\int_\Omega\begin{bmatrix}\frac{\partial N_i}{\partial x} & \frac{\partial N_i}{\partial y}\end{bmatrix}\begin{bmatrix}q_x\\ q_y\end{bmatrix}d\Omega=\int_\Omega N_i d\Omega-\int_\Gamma\left(\begin{bmatrix}q_x\\ q_y\end{bmatrix}.\begin{bmatrix}n_x\\ n_y\end{bmatrix}\right)N_i d\Gamma \tag{2.4}$$

here Γ is the boundary surface, n_x, n_y are the components of normal vector to the surface. Boundary conditions are of three types namely constant temperature, convection boundary and specified heat flux lost q_s across the boundary. These are given below in the form of equations

$$\begin{aligned} q_x n_x+q_y n_y&=-q_s\\ q_x n_x+q_y n_y&=h\left(T_s-T_e\right)\end{aligned} \tag{2.5}$$

here h is the convective heat transfer coefficient, T_s is the surface temperature and T_e is the surrounding temperature. The last term in the above equation is relevant only for boundary elements as it will be mutually cancelled for all the interior elements with their adjacent elements.
For the Linear Transient Analysis the elemental equation becomes

$$[C]\left[\frac{\partial T}{\partial t}\right] + [K][T] = [R_Q] + [R_q] + [R_h] + [R_T] \tag{2.6}$$

Here the matrices $[C], [K]$ are called capacitance and conduction matrix respectively, $[R_Q], [R_q], [R_h], [R_T]$ are the heat vectors arising from internal heat generation, specified suface heat flux, surface convection and specified nodal temperatures respectively. The definitions for the matrices are given by

$$\begin{aligned}
[C] &= \int_{\Omega} \rho c \{N\} [N] \, d\Omega \\
[K] &= \int_{\Omega} k [B]^T [B] \, d\Omega \\
[R_T] &= \int_{S_1} (q_x n_x + q_y n_y) [N] \, d\Gamma \\
[R_Q] &= \int_{\Omega} (Q) [N] \, d\Gamma \\
[R_q] &= \int_{S_2} (q_s) [N] \, d\Gamma \\
[R_h] &= \int_{S_3} (hT_e) [N] \, d\Gamma
\end{aligned} \tag{2.7}$$

Here S_1 is the boundary where constant temperature is specified, S_2 is the boundary where heat flux is specified, S_3 is the surface where convection is present. The representations of some matrices above are given for the triangular element in the appendix.
We assemble the elemental equations to get global matrix equations. Solving these system of equations gives the solution or Temperature profile at a given time.

Chapter 3

Code

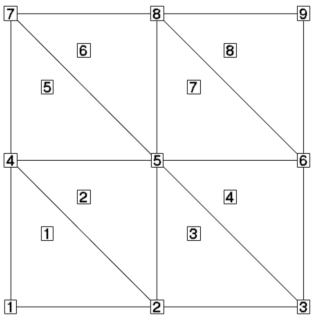

Figure 3.1: Triangular mesh

As stated in the previous chapters first step is the discretization of domain into smaller elements, in this project a rectangular geometry is discretized into triangular elements as shown in the figure. The mesh is characterized by the number of nodes, elements, element - node connectivity. The code takes a file called mesh1.dat as input which contains all the above stated characteristics.

The main function of the code is present in the file femcode.cpp. The header files like mesh.hpp, elements.hpp, nodes.hpp, boundary.hpp and solver.hpp perform several critial functions. the mesh.hpp defines a mesh class which defines several variables required for storing mesh and also defines readMesh() method which reads the mesh by taking mesh1.dat as the input. The elements.hpp, boundary.hpp files contain the classes for triangluar elements and boundaryline elements. These classes contains requisite variables and methods need for elemental matrix equations. The nodes.hpp contains a Node class which stores the index and x,y coordinates of the respective node object. The solver class attempts to assemble elemental equations to obtain global equations and solves them by Crank- Nicolson method. This class is not fully implemented and have to be developed in future. meshgenerator.cpp file generates mesh1.dat file. The implementations of all files are given in further pages

meshgenerator.cpp

```
#include <cstdlib> // for exit()
#include <iostream>
#include <fstream>
#include <iomanip>

int main()
{
    using namespace std;
    ofstream outf("mesh1.dat", ios::app);
    if (!outf)
    {
        cerr << "mesh.dat could not be opened for writing!" << endl;
        exit(1);
    }
    outf<<"density = 10"<<endl;
    outf<<"specificHeat = 5"<<endl;
    outf<<"thickness = 1"<<endl;
    outf<<"thermalConductivity = 4"<<endl;
    outf<<"time = 10"<<endl;
```

```
  int hori=6; int verti=6;
  outf <<"nElem = "<<(hori-1)*(verti-1)*2<< endl;
  int count = 0;
  int j = 1;
  for(int i=1;i<=((hori-1)*(verti-1));i++){
      //outf<<"j ="<<j<<endl;
      count = count + 1;
      outf <<"5e"<<" "<<count<<" "<< j<<" "<<j+1<<" "<<j+hori<<endl;
      count = count + 1;
      outf <<"5e"<<" "<<count<<" "<< j+1<<" "<<j+hori+1<<" "<<j+hori<<endl;
     if((i%(hori-1))==0) j=j+2;
     else j=j+1; }
  int nPoin = hori*verti;
  outf <<"nPoin = "<<nPoin<< endl;
 double deltax = 0.1,deltay = 0.1;
// outf<<setprecision(17);
  for(int i=0;i<nPoin;i++){
      outf <<"Poi"<<" "<< i+1<<" "<<(i%hori)*deltax<<" "<<(i/hori)*deltay<<endl;
  }
  outf<<"nMark = 4"<<endl;
  outf<<"Marker_Tag_ 1 = Bottom"<<endl;
```

```
    outf<<"nBottom = "<<hori-1<<endl;
   // outf<<"Bottom 3e"<<" "<<1<<" "<<2<<" ConsT "<<350<<" "<<600<<endl;
    for(int i=0;i<hori-1;i++) outf<<"Bottom 3e"<<" "<<i+1<<" "<<i+2<<" ConsT "<<600<<" "<<600<<endl;
   // outf<<"Bottom 3e"<<" "<<hori-2+1<<" "<<hori-2+2<<" ConsT "<<600<<" "<<350<<endl;

    outf<<"Marker_Tag_ 2 = Right"<<endl;
    outf<<"nRight = "<<verti-1<<endl;
    for(int i=0;i<verti-1;i++) outf<<"Right 3e"<<" "<<hori*(i+1)<<" "<<hori*(i+2)<<" ConsT "<<100<<" "<<100<<endl;

    outf<<"Marker_Tag_ 3 = Top"<<endl;
    outf<<"nTop = "<<hori-1<<endl;
    for(int i=0;i<hori-1;i++) outf<<"Top 3e"<<" "<<(verti*hori)-i<<" "<<(verti*hori)-i-1<<" ConsT "<<100<<" "<<100<<endl;

    outf<<"Marker_Tag_ 4 = Left"<<endl;
    outf<<"nLeft = "<<verti-1<<endl;
    for(int i=0;i<verti-1;i++) outf<<"Left 3e"<<" "<<(hori*(i+1))+1<<" "<<(hori*(i))+1<<" ConsT "<<100<<" "<<100<<endl;

    return 0;
}
```

mesh1.dat

```
density = 2800
specificHeat = 795
thickness = 1
thermalConductivity = 143
time = 100
cfl = 0.5
nElem = 50
5e 1 1 2 7
5e 2 2 8 7
5e 3 2 3 8
5e 4 3 9 8
5e 5 3 4 9
5e 6 4 10 9
5e 7 4 5 10
5e 8 5 11 10
5e 9 5 6 11
5e 10 6 12 11
5e 11 7 8 13
5e 12 8 14 13
```

5e 13 8 9 14

5e 14 9 15 14

5e 15 9 10 15

5e 16 10 16 15

5e 17 10 11 16

5e 18 11 17 16

5e 19 11 12 17

5e 20 12 18 17

5e 21 13 14 19

5e 22 14 20 19

5e 23 14 15 20

5e 24 15 21 20

5e 25 15 16 21

5e 26 16 22 21

5e 27 16 17 22

5e 28 17 23 22

5e 29 17 18 23

5e 30 18 24 23

5e 31 19 20 25

5e 32 20 26 25

5e 33 20 21 26

```
5e 34 21 27 26
5e 35 21 22 27
5e 36 22 28 27
5e 37 22 23 28
5e 38 23 29 28
5e 39 23 24 29
5e 40 24 30 29
5e 41 25 26 31
5e 42 26 32 31
5e 43 26 27 32
5e 44 27 33 32
5e 45 27 28 33
5e 46 28 34 33
5e 47 28 29 34
5e 48 29 35 34
5e 49 29 30 35
5e 50 30 36 35
nPoin = 36
Poi 1 0 0
Poi 2 0.1 0
Poi 3 0.2 0
```

```
Poi 4 0.3 0
Poi 5 0.4 0
Poi 6 0.5 0
Poi 7 0 0.1
Poi 8 0.1 0.1
Poi 9 0.2 0.1
Poi 10 0.3 0.1
Poi 11 0.4 0.1
Poi 12 0.5 0.1
Poi 13 0 0.2
Poi 14 0.1 0.2
Poi 15 0.2 0.2
Poi 16 0.3 0.2
Poi 17 0.4 0.2
Poi 18 0.5 0.2
Poi 19 0 0.3
Poi 20 0.1 0.3
Poi 21 0.2 0.3
Poi 22 0.3 0.3
Poi 23 0.4 0.3
Poi 24 0.5 0.3
```

```
Poi 25 0 0.4
Poi 26 0.1 0.4
Poi 27 0.2 0.4
Poi 28 0.3 0.4
Poi 29 0.4 0.4
Poi 30 0.5 0.4
Poi 31 0 0.5
Poi 32 0.1 0.5
Poi 33 0.2 0.5
Poi 34 0.3 0.5
Poi 35 0.4 0.5
Poi 36 0.5 0.5
nMark = 4
Marker_Tag_ 1 = Bottom
nBottom = 5
Bottom 3e 1 2 ConsT 600 600
Bottom 3e 2 3 ConsT 600 600
Bottom 3e 3 4 ConsT 600 600
Bottom 3e 4 5 ConsT 600 600
Bottom 3e 5 6 ConsT 600 600
Marker_Tag_ 2 = Right
```

```
nRight = 5
Right 3e 6 12 ConsT 100 100
Right 3e 12 18 ConsT 100 100
Right 3e 18 24 ConsT 100 100
Right 3e 24 30 ConsT 100 100
Right 3e 30 36 ConsT 100 100
Marker_Tag_ 3 = Top
nTop = 5
Top 3e 36 35 ConsT 100 100
Top 3e 35 34 ConsT 100 100
Top 3e 34 33 ConsT 100 100
Top 3e 33 32 ConsT 100 100
Top 3e 32 31 ConsT 100 100
Marker_Tag_ 4 = Left
nLeft = 5
Left 3e 7 1 ConsT 100 100
Left 3e 13 7 ConsT 100 100
Left 3e 19 13 ConsT 100 100
Left 3e 25 19 ConsT 100 100
Left 3e 31 25 ConsT 100 100
```

femcode.cpp

```
//#include <cstdlib>
//#include <iostream>
//#include <fstream>
#include <bits/stdc++.h>
using namespace std;
#include "mesh.hpp"
int main()
{
    Mesh mesh1;
    mesh1.readMesh();
    mesh1.prepareElements();
    mesh1.solVe();



   return 0;
}
```

mesh.hpp

```
#pragma once
#include <bits/stdc++.h>
#include <string>
#include "elements.hpp"
#include "nodes.hpp"
#include "boundary.hpp"
#include "solver.hpp"
using namespace std;
class Mesh
{    TriElement **Ele;Node **Poi;BoundLineElement ***Bou;Solver *MySolver;
   int nElem,nPoin,nMark;
   double density,specificHeat,thickness,thermalConductivity,time,cfl;
   public:
      Mesh()
      {
         Ele = NULL;
         Poi = NULL;
         Bou = NULL;
      }
```

```
// int nElem,nPoin,nMark,density,specificHeat,thickness;
   int *NoOfEachMarkers;
    vector <string> tokens;
    vector <string> tokens1;
    vector <string> tokens2;
    vector <string> markers;
   void readMesh()
   {
      ifstream inf("mesh1.dat");
        if (!inf)
            {
              cerr << "mesh.dat could not be opened for reading!" << endl;
              exit(1);
            }
int countElements = 0,countPoints = 0,countMarkers=0,eleNum=0,firstIndex=0,secondIndex=0,thirdIndex=0,pIndex=0;
double xCord=0,yCord=0;

       while (inf)
       {
           std::string line;
           getline(inf, line);
```

```
  //cout << line <<endl;

    stringstream check1(line);
    string intermediate;
    while(getline(check1, intermediate, ' '))
    {
        tokens.push_back(intermediate);
    }
 if(tokens[0]=="nElem") {
     nElem = 0;
    stringstream geek(tokens[tokens.size()-1]);
   geek >> nElem;
   Ele = new TriElement*[nElem+1];
    countElements=1;
}
if(tokens[0]=="5e"){
eleNum=0;firstIndex=0;secondIndex=0;thirdIndex=0;
stringstream geek1(tokens[1]);
geek1 >> eleNum;
stringstream geek2(tokens[2]);
geek2 >> firstIndex;
```

```
    stringstream geek3(tokens[3]);
    geek3 >> secondIndex;
    stringstream geek4(tokens[4]);
    geek4 >> thirdIndex;
    Ele[eleNum] = new TriElement(eleNum,firstIndex,secondIndex,thirdIndex);
    countElements+=1;
       }
       if(tokens[0]=="nPoin") {
          nPoin = 0;
         stringstream geek(tokens[tokens.size()-1]);
       geek >> nPoin;
       Poi = new Node*[nPoin+1];
    }

if(tokens[0]=="Poi"){
    pIndex=0;xCord=0;yCord=0;
    stringstream geek5(tokens[1]);
    geek5 >> pIndex;
    stringstream geek6(tokens[2]);
    geek6 >> xCord;
    stringstream geek7(tokens[3]);
```

```
   geek7 >> yCord;
   Poi[pIndex] = new Node(pIndex,xCord,yCord);
   countPoints+=1;
      }
if(tokens[0]=="density")
   {density=0;
      stringstream geek8(tokens[tokens.size()-1]);
      geek8 >> density;}
if(tokens[0]=="specificHeat")
   {specificHeat=0;
      stringstream geek9(tokens[tokens.size()-1]);
      geek9 >> specificHeat;}
if(tokens[0]=="thickness")
    {thickness=0;
      stringstream geek10(tokens[tokens.size()-1]);
      geek10 >> thickness;}

   if(tokens[0]=="thermalConductivity")
    {thermalConductivity=0;
      stringstream geek17(tokens[tokens.size()-1]);
      geek17 >> thermalConductivity;}
```

```
 if(tokens[0]=="time")
    {time=0;
      stringstream geek18(tokens[tokens.size()-1]);
      geek18 >> time;}

 if(tokens[0]=="cfl")
    {cfl=0;
      stringstream geek18(tokens[tokens.size()-1]);
      geek18 >> cfl;}
```



```
if(tokens[0]=="nMark") {
        nMark = 0;
       stringstream geek11(tokens[tokens.size()-1]);
      geek11 >> nMark;
     Bou = new BoundLineElement**[nMark];
     NoOfEachMarkers = new int[nMark];

   }
```

```
      if((tokens[0]=="Marker_Tag_"))

   {     markers.push_back(tokens[tokens.size()-1]);

   }

    tokens.clear();
}

//inf.close();
 cout<<"number of elements in the mesh is "<<nElem<<endl;
 cout<<"number of points in the mesh is "<<nPoin<<endl;
 cout<<"specificHeat is "<<specificHeat<<endl;
 cout<<"density is "<<density<<endl;
 cout<<"thickness is "<<thickness<<endl;
 cout<<"thermalConductivity is "<<thermalConductivity<<endl;
 cout<<"number of boundary markers are "<<nMark<<endl;
for(int i=0;i<nMark;i++) cout<<markers[i]<<endl;
   //for(int i=0;i<nMark;i++) *boundaryMarkers[countMarkers+1]<<endl;
   int *countEachMarker;
```

```
countEachMarker = new int[nMark];
   for(int i=0;i<nMark;i++) countEachMarker[i]=0;
   int tempCountMarker=0;

   ifstream inf1("mesh1.dat");
   while (inf1)
 {
     std::string line;
     getline(inf1, line);
     stringstream check2(line);
       string intermediate;
       while(getline(check2, intermediate, ' '))
       {
           tokens1.push_back(intermediate);
       }

       if((countMarkers<nMark)&&(tokens1[0]==("n"+markers[countMarkers]))) {
       cout<<countMarkers<<endl;
         cout<<"n"+markers[countMarkers]<<endl;
         NoOfEachMarkers[countMarkers] = 0;
        stringstream geek(tokens1[tokens1.size()-1]);
```

```
      geek >> NoOfEachMarkers[countMarkers];
     Bou[countMarkers] = new BoundLineElement*[NoOfEachMarkers[countMarkers]];
     tempCountMarker = countMarkers;
      countMarkers+=1;
   }
   tokens1.clear();
}


/*
    if((tokens1[0]==markers[tempCountMarker])&&(countEachMarker[tempCountMarker]<NoOfEachMarkers[tempCountMarker])){

      int tempfirstindex=0,tempsecondindex=0;
      stringstream geek13(tokens1[2]);
         geek13 >> tempfirstindex;
         stringstream geek14(tokens1[3]);
         geek14 >> tempsecondindex;
         Bou[countMarkers][countEachMarker[tempCountMarker]] = new
             BoundLineElement(tempfirstindex,tempsecondindex);
           countEachMarker[tempCountMarker]+=1;
```

```
  }      */
  countMarkers = 0;
  ifstream inf2("mesh1.dat");
  while (inf2)
{
    std::string line;
    getline(inf2, line);
    stringstream check3(line);
      string intermediate;
      while(getline(check3, intermediate, ' '))
      {
          tokens2.push_back(intermediate);
      }
    //
        if((tokens1[0]==markers[tempCountMarker])&&(countEachMarker[tempCountMarker]<NoOfEachMarkers[tempCountMarker]))

          if(tokens2[0]==markers[countMarkers])
      {

     int tempfirstindex=0,tempsecondindex=0; double
        tempTemperature1=0,tempTemperature2=0,Conveccoef=0,ConvecTemp=0,heatflux=0;
```

```
        if(tokens2[4]=="ConsT"){
              stringstream geek13(tokens2[2]);
                 geek13 >> tempfirstindex;
                 stringstream geek14(tokens2[3]);
                 geek14 >> tempsecondindex;
                 stringstream geek15(tokens2[5]);
                 geek15 >> tempTemperature1;
                 stringstream geek16(tokens2[6]);
                 geek16 >> tempTemperature2;
 Bou[countMarkers][countEachMarker[countMarkers]] = new
    BoundLineElement(tempfirstindex,tempsecondindex,tokens2[4],tempTemperature1,tempTemperature2);}
 if(tokens2[4]=="Convec"){
                  stringstream geek13(tokens2[2]);
                 geek13 >> tempfirstindex;
                 stringstream geek14(tokens2[3]);
                 geek14 >> tempsecondindex;
                 stringstream geek15(tokens2[5]);
                 geek15 >> Conveccoef;
                 stringstream geek16(tokens2[6]);
                 geek16 >> ConvecTemp;
Bou[countMarkers][countEachMarker[countMarkers]] = new
```

```
    BoundLineElement(tempfirstindex,tempsecondindex,Conveccoef,ConvecTemp,tokens2[4]);
 }
if(tokens2[4]=="HFlux"){
              stringstream geek13(tokens2[2]);
                 geek13 >> tempfirstindex;
                 stringstream geek14(tokens2[3]);
                 geek14 >> tempsecondindex;
                 stringstream geek15(tokens2[5]);
                 geek15 >> heatflux;
Bou[countMarkers][countEachMarker[countMarkers]] = new
    BoundLineElement(tempfirstindex,tempsecondindex,tokens2[4],heatflux);

}

                  countEachMarker[countMarkers]+=1;
                  if(countEachMarker[countMarkers]==NoOfEachMarkers[tempCountMarker]) countMarkers+=1;

           }


            tokens2.clear();
```

```
    }

 for(int i=0;i<nMark;i++) cout<<"number of elements in each boundary markers are "<<NoOfEachMarkers[i]<<endl;
  // int countEachMarker[nMark]={0};


}

 void prepareElements(){
   for(int i=1;i<=nElem;i++)
    {
      Ele[i]->setElementCordinates(Poi[Ele[i]->GetFirstIndex()]->GetXcord(),
                                   Poi[Ele[i]->GetFirstIndex()]->GetYcord(),
                                   Poi[Ele[i]->GetSecondIndex()]->GetXcord(),
                                   Poi[Ele[i]->GetSecondIndex()]->GetYcord(),
                                   Poi[Ele[i]->GetThirdIndex()]->GetXcord(),
                                   Poi[Ele[i]->GetThirdIndex()]->GetYcord()
                                   );
      Ele[i]->setArea();
      Ele[i]->setConstants(density,specificHeat,thickness,thermalConductivity);
      Ele[i]->setElementMatrices();
```

```
          // Ele[i]->roughwork();
        }

        for(int i=0;i<nMark;i++){
   for(int j=0;j<NoOfEachMarkers[i];j++){
      Bou[i][j]->setBouElemCoordinates(Poi[Bou[i][j]->GetFirstIndex()]->GetXcord(),
                                Poi[Bou[i][j]->GetFirstIndex()]->GetYcord(),
                                Poi[Bou[i][j]->GetSecondIndex()]->GetXcord(),
                                Poi[Bou[i][j]->GetSecondIndex()]->GetYcord());
      Bou[i][j]->setLengthAndProperties();

   }
}
        //cout<<Ele[i]->Getfirstindex( )<<endl;
        // Ele[1]->roughwork();
  // cout<<(Poi[Ele[1]->GetFirstIndex()]->GetXcord())<<" "<<(Poi[Ele[1]->GetFirstIndex()]->GetYcord())<<" "<<endl;
  // cout<<(Poi[Ele[1]->GetSecondIndex()]->GetXcord())<<" "<<(Poi[Ele[1]->GetSecondIndex()]->GetYcord())<<" "<<endl;
  // cout<<(Poi[Ele[1]->GetThirdIndex()]->GetXcord())<<" "<<(Poi[Ele[1]->GetThirdIndex()]->GetYcord())<<" "<<endl;

        // for(int i=0;i<3;i++){ for(int j=0;j<3;j++) {cout<<Ele[1]->GetConductionMatrix(i,j)<<" ";} cout<<endl;}
     }
```

```
        void solVe(){

         MySolver = new
             Solver(Ele,Poi,Bou,nPoin,nElem,nMark,density,specificHeat,thermalConductivity,thickness,NoOfEachMarkers,time,cfl);
         MySolver->assemble();
         MySolver->SetMinTimeStepValue();
         MySolver->Correction();
         MySolver->Initialize();
         MySolver->Iterate();


        }

};
```

elements.hpp

```
#pragma once
#include "nodes.hpp"
class TriElement
{
   int eleNumber,firstindex,secondindex,thirdindex;
   double Area,x1,x2,x3,y1,y2,y3,conductionMatrix[3][3],capacitanceMatrix[3][3],bMatrix[2][3],TransbMatrix[3][2];
   Node *node1,*node2,*node3;
   double density,specificHeat,thickness,thermalConductivity;
public:
   TriElement(int eleNum,int x, int y,int z)
   {
      eleNumber = eleNum; firstindex = x; secondindex = y; thirdindex = z;
   // cout<<" element number "<<eleNumber<<"is created with firstindex "<<firstindex<<" secondindex "<<secondindex<<"
       thirdindex "<<thirdindex<<endl;
   }
   int GetFirstIndex(){return firstindex;}
   int GetSecondIndex(){return secondindex;}
   int GetThirdIndex(){return thirdindex;}
```

```
 void setElementCordinates(double X1, double Y1, double X2, double Y2, double X3, double Y3){
   x1=X1;x2=X2;x3=X3;
   y1=Y1;y2=Y2;y3=Y3;
}
void setArea(){
   Area = 0.5*((x1*(y2-y3))-(y1*(x2-x3))+((x2*y3-x3*y2)));
  // cout<< "element number "<<eleNumber<<" with "<<x1<<" "<<y1<<" "<<x2<<" "<<y2<<" "<<x3<<" "<<y3<<" with area
      "<<Area<<endl;
}
void setConstants(double x,double y,double z,double p)
{
   density= x; specificHeat= y; thickness= z; thermalConductivity=p;
}
void setElementMatrices(){
  for(int i=0;i<3;i++) for(int j=0;j<3;j++) {conductionMatrix[i][j] = 0;capacitanceMatrix[i][j] = 0;}
  for(int i=0;i<2;i++) for(int j=0;j<3;j++) {bMatrix[i][j]=0;TransbMatrix[j][i]=0;}
  bMatrix[0][0] = y2-y3; bMatrix[0][1] = y3-y1; bMatrix[0][2] = y1-y2;
 bMatrix[1][0] = x3-x2; bMatrix[1][1] = x1-x3; bMatrix[1][2] = x2-x1;
// double bMatrix[2][3]= {{y2-y3,y3-y1,y1-y2},{x3-x2,x1-x3,x2-x1}};
// double TransbMatrix[3][2];
```

```
    for(int i=0;i<2;i++) for(int j=0;j<3;j++)
    {bMatrix[i][j]*=(0.5/Area); }
for(int i=0;i<3;i++) for(int j=0;j<2;j++)
      {TransbMatrix[i][j] = bMatrix[j][i]; }
    for(int i=0;i<3;i++) for(int j=0;j<3;j++) for(int k=0;k<2;k++)
        conductionMatrix[i][j]+=(TransbMatrix[i][k]*bMatrix[k][j]);
    for(int i=0;i<3;i++) for(int j=0;j<3;j++) conductionMatrix[i][j]*=(Area*thermalConductivity*thickness);
    for(int i=0;i<3;i++) for(int j=0;j<3;j++) {if(i==j) capacitanceMatrix[i][j]=2; else capacitanceMatrix[i][j]=1;}
    for(int i=0;i<3;i++) for(int j=0;j<3;j++) capacitanceMatrix[i][j]*= (density*specificHeat*thickness*Area/12);
    //  cout<<"conductionMatrix[0][0] of eleNumber "<<eleNumber<<" is "<<conductionMatrix[0][0]<<endl;
  }

   double GetConductionMatrix(int temp1,int temp2){return conductionMatrix[temp1][temp2];}
  double GetCapacitanceMatrix(int temp1,int temp2){return capacitanceMatrix[temp1][temp2];}
   void roughwork(){
   /*   cout<<"The bMatrix of "<<eleNumber<<" is "<<endl;
     for(int i=0;i<2;i++) {for(int j=0;j<3;j++) cout<<bMatrix[i][j]<<" "; cout<<endl;}
     cout<<"The TransbMatrix of "<<eleNumber<<" is "<<endl;
       for(int i=0;i<3;i++) {for(int j=0;j<2;j++) cout<<TransbMatrix[i][j]<<" "; cout<<endl;}*/
       cout<<"capacitanceMatrix of element "<<eleNumber<<endl;
       for(int i=0;i<3;i++) {for(int j=0;j<3;j++) cout<<conductionMatrix[i][j]<<" ";cout<<endl;}
```

```
    }


};
```

nodes.hpp

```
#pragma once
class Node
{
   int pIndex;
   double x,y;
   string type;
public:
   Node(int index, double xCord, double yCord){
      pIndex = index; x = xCord; y = yCord;
   //   cout<<"point number "<<pIndex<<"is created with xCord "<<xCord<<" and yCord "<<yCord<<endl;
      type = "interiorNode";
   }
   double GetXcord(){ return x;}
   double GetYcord(){ return y;}
   void setAsBouNode(){ type = "ConsT";}
   void setAsCovecNode(){type = "Convec";}
   void setAsHfluxNode(){type = "HFlux";}
   string getNodeType(){return type;}
   int GetIndex(){return pIndex;}
```

```
};
```

boundary.hpp

```
class BoundLineElement{
   int firstindex,secondindex;
   double
      temperature1,temperature2,length,x1,y1,x2,y2,boundaryMatrix[2][1],ConvecCoeff,boundaryConvecMatrix[2][2],ConvTemp,heatFlux;
   string bctype;


public:
   BoundLineElement(int x, int y, string BCtype ,double Temperature1,double Temperature2)
   {
      firstindex = x; secondindex = y; temperature1 = Temperature1; temperature2 = Temperature2; bctype = BCtype;
 //cout<<"bou element is created with firstindex "<<firstindex<<" secondindex "<<secondindex<<" tem1 "<<temperature1<<"
     tem2 "<<temperature2<<" bctype "<<bctype<<endl;
   if(BCtype == "ConsT")
      {boundaryMatrix[0][0]=0;boundaryMatrix[1][0]=0;
          for(int i=0;i<2;i++) for(int j=0;j<2;j++) boundaryConvecMatrix[i][j]=0;
      }


   }
   BoundLineElement(int xX,int yY, double ConvCoeff, double ConTemp,string BCtypE)
```

```
   {
       firstindex = xX; secondindex = yY; ConvecCoeff=ConvCoeff; ConvTemp=ConTemp; bctype = BCtypE;

   }
   BoundLineElement(int xX,int yY, string BCtype, double heatFlu)
   {
firstindex = xX; secondindex = yY; bctype = BCtype; heatFlux=heatFlu;
   }

  void setBouElemCoordinates(double X1,double Y1,double X2,double Y2){
    x1=X1;y1=Y1;x2=X2;y2=Y2;
   }
   void setLengthAndProperties(){
length = sqrt(((x2-x1)*(x2-x1))+((y2-y1)*(y2-y1)));
if(bctype == "Convec")
   {  boundaryConvecMatrix[0][0]= (ConvecCoeff*length/3);
      boundaryConvecMatrix[0][1]= (ConvecCoeff*length/6);
      boundaryConvecMatrix[1][0]= (ConvecCoeff*length/6);
      boundaryConvecMatrix[1][1]= (ConvecCoeff*length/3);

      boundaryMatrix[0][0]=(ConvecCoeff*ConvTemp*length*0.5);boundaryMatrix[1][0]=(ConvecCoeff*ConvTemp*length*0.5);}
```

```
if(bctype == "HFlux"){
      boundaryMatrix[0][0]=(heatFlux*length*0.5);boundaryMatrix[1][0]=(heatFlux*length*0.5);
      for(int i=0;i<2;i++) for(int j=0;j<2;j++) boundaryConvecMatrix[i][j]=0;
}


  }
  int GetFirstIndex(){return firstindex;}
  int GetSecondIndex(){return secondindex;}
  double GetboundaryMatrix(int x,int y){return boundaryMatrix[x][y];}
  double GetboundaryConvecMatrix(int x,int y){return boundaryConvecMatrix[x][y];}
  double GetTemperature1(){return temperature1;}
  double GetTemperature2(){return temperature2;}
  string GetBoundaryType(){return bctype;}



};
```

solver.hpp

```
#pragma once
#include <bits/stdc++.h>
class Solver{
   TriElement **EleSolver=NULL;Node **PoiSolver=NULL;BoundLineElement ***BouSolver=NULL;
   double **assembledCapacitanceMatrix, **assembledConductionMatrix;
   double* assembledRHSMatrix;
   int nPoin,nElem,nMark, *NoOfEachMarkers,numberOfSteps,numberOfConsT,*non_consTindex,*consTindex;
    double density,specificHeat,thermalConductivity,thickness,minTimeStepValue,lastTimeStep,time,cfl;
    double *solution,**equationRhsMatrix,**preConditionerMatrix,*solutiontemp,*consTvector,*solutiontemp1;
    double *reduced_solution,**reduced_preConditionerMatrix, **reduced_equationRhsMatrix, **Lmatrix,**Umatrix;
    double *forwardSubMatrix,*backwardSubMatrix;

   public:
   Solver(TriElement **EleSolver,
              Node **PoiSolver,
              BoundLineElement ***BouSolver,
              int nPoin,
              int nElem,
              int nMark,
```

```
            double density,
            double specificHeat,
            double thermalConductivity,
            double thickness,
            int *NoOfEachMarkers,
            double time,
            double cfl)

{   this->EleSolver=EleSolver;
   this->PoiSolver=PoiSolver;
   this->BouSolver=BouSolver;
     this->nPoin=nPoin;
     this->nElem=nElem;
     this->nMark=nMark;
     this->density=density;
     this->specificHeat=specificHeat;
     this->thermalConductivity=thermalConductivity;
     this->thickness=thickness;
     this->NoOfEachMarkers = NoOfEachMarkers;
     this->time = time;
     this->cfl = cfl;
```

```
        numberOfConsT = 0;
        assembledCapacitanceMatrix =new double* [nPoin];
        for(int i=0;i<nPoin;i++) assembledCapacitanceMatrix[i] = new double[nPoin];
        for(int i=0;i<nPoin;i++) for(int j=0;j<nPoin;j++) assembledCapacitanceMatrix[i][j]=0;
        assembledConductionMatrix = new double* [nPoin];
        for(int i=0;i<nPoin;i++) assembledConductionMatrix[i] = new double[nPoin];
        for(int i=0;i<nPoin;i++) for(int j=0;j<nPoin;j++) assembledConductionMatrix[i][j]=0;
        assembledRHSMatrix = new double[nPoin];
        for(int i=0;i<nPoin;i++) assembledRHSMatrix[i] = 0;

   // cout<<"first index of 18th element is "<<EleSolver[18]->GetFirstIndex()<<
   //" and its x,y coordinate is given by "<<PoiSolver[EleSolver[18]->GetFirstIndex()]->GetXcord()<<" ,
       "<<PoiSolver[EleSolver[18]->GetFirstIndex()]->GetYcord()<<endl;
      cout<<"hurray solver class is created"<<endl;
   }


////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

   void assemble(){
```

```
   // cout<<"EleSolver[1]->GetConductionMatrix(0,0) "<<EleSolver[1]->GetConductionMatrix(0,0)<<endl;

 for(int i=1;i<=nElem;i++) {
assembledConductionMatrix[(EleSolver[i]->GetFirstIndex())-1][(EleSolver[i]->GetFirstIndex())-1] +=
    EleSolver[i]->GetConductionMatrix(0,0);
assembledConductionMatrix[(EleSolver[i]->GetFirstIndex())-1][(EleSolver[i]->GetSecondIndex())-1] +=
    EleSolver[i]->GetConductionMatrix(0,1);
assembledConductionMatrix[(EleSolver[i]->GetFirstIndex())-1][(EleSolver[i]->GetThirdIndex())-1] +=
    EleSolver[i]->GetConductionMatrix(0,2);

assembledConductionMatrix[(EleSolver[i]->GetSecondIndex())-1][(EleSolver[i]->GetFirstIndex())-1] +=
    EleSolver[i]->GetConductionMatrix(1,0);
assembledConductionMatrix[(EleSolver[i]->GetSecondIndex())-1][(EleSolver[i]->GetSecondIndex())-1] +=
    EleSolver[i]->GetConductionMatrix(1,1);
assembledConductionMatrix[(EleSolver[i]->GetSecondIndex())-1][(EleSolver[i]->GetThirdIndex())-1] +=
    EleSolver[i]->GetConductionMatrix(1,2);

assembledConductionMatrix[(EleSolver[i]->GetThirdIndex())-1][(EleSolver[i]->GetFirstIndex())-1] +=
    EleSolver[i]->GetConductionMatrix(2,0);
assembledConductionMatrix[(EleSolver[i]->GetThirdIndex())-1][(EleSolver[i]->GetSecondIndex())-1] +=
    EleSolver[i]->GetConductionMatrix(2,1);
```

```
assembledConductionMatrix[(EleSolver[i]->GetThirdIndex())-1][(EleSolver[i]->GetThirdIndex())-1] +=
    EleSolver[i]->GetConductionMatrix(2,2);

    }

for(int i=1;i<=nElem;i++) {
assembledCapacitanceMatrix[(EleSolver[i]->GetFirstIndex())-1][(EleSolver[i]->GetFirstIndex())-1] +=
    EleSolver[i]->GetCapacitanceMatrix(0,0);
assembledCapacitanceMatrix[(EleSolver[i]->GetFirstIndex())-1][(EleSolver[i]->GetSecondIndex())-1] +=
    EleSolver[i]->GetCapacitanceMatrix(0,1);
assembledCapacitanceMatrix[(EleSolver[i]->GetFirstIndex())-1][(EleSolver[i]->GetThirdIndex())-1] +=
    EleSolver[i]->GetCapacitanceMatrix(0,2);

assembledCapacitanceMatrix[(EleSolver[i]->GetSecondIndex())-1][(EleSolver[i]->GetFirstIndex())-1] +=
    EleSolver[i]->GetCapacitanceMatrix(1,0);
assembledCapacitanceMatrix[(EleSolver[i]->GetSecondIndex())-1][(EleSolver[i]->GetSecondIndex())-1] +=
    EleSolver[i]->GetCapacitanceMatrix(1,1);
assembledCapacitanceMatrix[(EleSolver[i]->GetSecondIndex())-1][(EleSolver[i]->GetThirdIndex())-1] +=
    EleSolver[i]->GetCapacitanceMatrix(1,2);

assembledCapacitanceMatrix[(EleSolver[i]->GetThirdIndex())-1][(EleSolver[i]->GetFirstIndex())-1] +=
```

```
    EleSolver[i]->GetCapacitanceMatrix(2,0);
assembledCapacitanceMatrix[(EleSolver[i]->GetThirdIndex())-1][(EleSolver[i]->GetSecondIndex())-1] +=
    EleSolver[i]->GetCapacitanceMatrix(2,1);
assembledCapacitanceMatrix[(EleSolver[i]->GetThirdIndex())-1][(EleSolver[i]->GetThirdIndex())-1] +=
    EleSolver[i]->GetCapacitanceMatrix(2,2);


    }
for(int i=0;i<nMark;i++){
   for(int j=0;j<NoOfEachMarkers[i];j++){
      assembledRHSMatrix[(BouSolver[i][j]->GetFirstIndex())-1] += BouSolver[i][j]->GetboundaryMatrix(0,0);
      assembledRHSMatrix[(BouSolver[i][j]->GetSecondIndex())-1] += BouSolver[i][j]->GetboundaryMatrix(1,0);
      if((BouSolver[i][j]->GetBoundaryType())=="ConsT") numberOfConsT += 1;
assembledConductionMatrix[(BouSolver[i][j]->GetFirstIndex())-1][(BouSolver[i][j]->GetFirstIndex())-1] +=
    BouSolver[i][j]->GetboundaryConvecMatrix(0,0);
assembledConductionMatrix[(BouSolver[i][j]->GetFirstIndex())-1][(BouSolver[i][j]->GetSecondIndex())-1] +=
    BouSolver[i][j]->GetboundaryConvecMatrix(0,1);
assembledConductionMatrix[(BouSolver[i][j]->GetSecondIndex())-1][(BouSolver[i][j]->GetFirstIndex())-1] +=
    BouSolver[i][j]->GetboundaryConvecMatrix(1,0);
assembledConductionMatrix[(BouSolver[i][j]->GetSecondIndex())-1][(BouSolver[i][j]->GetSecondIndex())-1] +=
    BouSolver[i][j]->GetboundaryConvecMatrix(1,1);
   }
```

```
}
/*cout<<" assembledCapacitanceMatrix "<<endl;
for(int i=0;i<nPoin;i++) {for(int j=0;j<nPoin;j++) cout<<assembledCapacitanceMatrix[i][j]<<" "; cout<<endl;}
cout<<" assembledConductionMatrix "<<endl;
for(int i=0;i<nPoin;i++) {for(int j=0;j<nPoin;j++) cout<<assembledConductionMatrix[i][j]<<" "; cout<<endl;} */

   }

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

  void SetMinTimeStepValue(){
//time step for Crank-Nicolson Method
    minTimeStepValue = 0; double
        tempForSwap,firstElementTimeStep,x1,y1,x2,y2,x3,y3,Cx,Cy,distFromC1,distFromC2,distFromC3,characteristicLength;

   {
      int i =1;
      characteristicLength = 0;
               x1 = PoiSolver[EleSolver[i]->GetFirstIndex()]->GetXcord();
                y1 = PoiSolver[EleSolver[i]->GetFirstIndex()]->GetYcord();
```

```
                x2 = PoiSolver[EleSolver[i]->GetSecondIndex()]->GetXcord();
                y2 = PoiSolver[EleSolver[i]->GetSecondIndex()]->GetYcord();
                x3 = PoiSolver[EleSolver[i]->GetThirdIndex()]->GetXcord();
                y3 = PoiSolver[EleSolver[i]->GetThirdIndex()]->GetYcord();
                Cx=(x1+x2+x3)/3; Cy=(y1+y2+y3)/3;
                distFromC1 = (((Cx-x1)*(Cx-x1))+((Cy-y1)*(Cx-y1)));
                distFromC2 = (((Cx-x2)*(Cx-x2))+((Cy-y2)*(Cx-y2)));
                distFromC3 = (((Cx-x3)*(Cx-x3))+((Cy-y3)*(Cx-y3)));
    if(distFromC1 > distFromC2){if(distFromC1 > distFromC3) characteristicLength = distFromC1; else characteristicLength
        = distFromC3;}
    else{if(distFromC2 > distFromC3) characteristicLength = distFromC2; else characteristicLength = distFromC3;}
    minTimeStepValue = characteristicLength;
}

  for(int i=2;i<=nElem;i++){
            characteristicLength = 0;
             x1 = PoiSolver[EleSolver[i]->GetFirstIndex()]->GetXcord();
              y1 = PoiSolver[EleSolver[i]->GetFirstIndex()]->GetYcord();
              x2 = PoiSolver[EleSolver[i]->GetSecondIndex()]->GetXcord();
              y2 = PoiSolver[EleSolver[i]->GetSecondIndex()]->GetYcord();
              x3 = PoiSolver[EleSolver[i]->GetThirdIndex()]->GetXcord();
```

```
                y3 = PoiSolver[EleSolver[i]->GetThirdIndex()]->GetYcord();
                Cx=(x1+x2+x3)/3; Cy=(y1+y2+y3)/3;
                distFromC1 = (((Cx-x1)*(Cx-x1))+((Cy-y1)*(Cx-y1)));
                distFromC2 = (((Cx-x2)*(Cx-x2))+((Cy-y2)*(Cx-y2)));
                distFromC3 = (((Cx-x3)*(Cx-x3))+((Cy-y3)*(Cx-y3)));
   if(distFromC1 > distFromC2){if(distFromC1 > distFromC3) characteristicLength = distFromC1; else characteristicLength
      = distFromC3;}
   else{if(distFromC2 > distFromC3) characteristicLength = distFromC2; else characteristicLength = distFromC3;}

    if(minTimeStepValue > characteristicLength) minTimeStepValue = characteristicLength;
   }
//cout<<"minTimeStepValue before is "<<minTimeStepValue<<endl;

  minTimeStepValue /= (6*thermalConductivity/(density*specificHeat));

  minTimeStepValue *= cfl ;

  cout<<"minTimeStepValue after is "<<minTimeStepValue<<endl;
   }

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
void Correction(){

   for(int i=0;i<nMark;i++)
   for(int j=0;j<NoOfEachMarkers[i];j++)
   {
      if((BouSolver[i][j]->GetBoundaryType()) == "ConsT"){
            PoiSolver[(BouSolver[i][j]->GetFirstIndex())]->setAsBouNode();
            PoiSolver[(BouSolver[i][j]->GetSecondIndex())]->setAsBouNode();
      }}


   for(int i=0;i<nMark;i++)
   for(int j=0;j<NoOfEachMarkers[i];j++)
   {
if((BouSolver[i][j]->GetBoundaryType()) == "Convec"){
   if((PoiSolver[(BouSolver[i][j]->GetFirstIndex())]->getNodeType())!="ConsT")
       PoiSolver[(BouSolver[i][j]->GetFirstIndex())]->setAsCovecNode();
   if((PoiSolver[(BouSolver[i][j]->GetSecondIndex())]->getNodeType())!="ConsT")
       PoiSolver[(BouSolver[i][j]->GetSecondIndex())]->setAsCovecNode();
      }
      if((BouSolver[i][j]->GetBoundaryType()) == "HFlux"){
```

```
 if((PoiSolver[(BouSolver[i][j]->GetFirstIndex())]->getNodeType())!="ConsT")
     PoiSolver[(BouSolver[i][j]->GetFirstIndex())]->setAsHfluxNode();
 if((PoiSolver[(BouSolver[i][j]->GetSecondIndex())]->getNodeType())!="ConsT")
     PoiSolver[(BouSolver[i][j]->GetSecondIndex())]->setAsHfluxNode();
      }


   }
numberOfConsT = 0;
for(int i=0;i<nPoin;i++) {
     if((PoiSolver[i+1]->getNodeType())=="ConsT"){numberOfConsT+=1;}}
}

////////////////////////////////////////////////////////////////////////////////////////////////////

  void Initialize()
  {  solution = new double[nPoin];
    solutiontemp = new double[nPoin];
    solutiontemp1 = new double[nPoin];
   for(int i=0;i<nPoin;i++) {solution[i] = 0;solutiontemp[i] = 0;solutiontemp1[i] = 0;}


    equationRhsMatrix = new double*[nPoin];
```

```
for(int i=0;i<nPoin;i++) equationRhsMatrix[i] = new double[nPoin];
for(int i=0;i<nPoin;i++) for(int j=0;j<nPoin;j++) equationRhsMatrix[i][j] = 0;
   preConditionerMatrix = new double* [nPoin];
for(int i=0;i<nPoin;i++) preConditionerMatrix[i] = new double[nPoin];
for(int i=0;i<nPoin;i++) for(int j=0;j<nPoin;j++) preConditionerMatrix[i][j] = 0;

 reduced_solution = new double[(nPoin-numberOfConsT)];
for(int i=0;i<(nPoin-numberOfConsT);i++) reduced_solution[i] = 0;

 reduced_equationRhsMatrix = new double*[(nPoin-numberOfConsT)];
for(int i=0;i<(nPoin-numberOfConsT);i++) reduced_equationRhsMatrix[i] = new double[nPoin-numberOfConsT];
 for(int i=0;i<(nPoin-numberOfConsT);i++) for(int j=0;j<(nPoin-numberOfConsT);j++) reduced_equationRhsMatrix[i][j] = 0;

reduced_preConditionerMatrix = new double* [nPoin-numberOfConsT];
for(int i=0;i<(nPoin-numberOfConsT);i++) reduced_preConditionerMatrix[i] = new double[nPoin-numberOfConsT];
for(int i=0;i<(nPoin-numberOfConsT);i++) for(int j=0;j<(nPoin-numberOfConsT);j++) reduced_preConditionerMatrix[i][j] =
    0;

 forwardSubMatrix = new double[(nPoin-numberOfConsT)];
backwardSubMatrix = new double[(nPoin-numberOfConsT)];
non_consTindex = new int[(nPoin-numberOfConsT)];
```

```
consTindex = new int[numberOfConsT];
consTvector = new double[numberOfConsT];
 for(int i=0;i<(nPoin-numberOfConsT);i++){forwardSubMatrix[i]=0;backwardSubMatrix[i]=0;}

Lmatrix = new double* [nPoin-numberOfConsT];
 Umatrix = new double* [nPoin-numberOfConsT];
for(int i=0;i<(nPoin-numberOfConsT);i++)
   {Lmatrix[i] = new double[nPoin-numberOfConsT];
    Umatrix[i] = new double[nPoin-numberOfConsT];
   }
for(int i=0;i<(nPoin-numberOfConsT);i++) for(int j=0;j<(nPoin-numberOfConsT);j++)
   { Lmatrix[i][j] = 0; Umatrix[i][j] = 0;}




for(int i=0;i<nMark;i++)
for(int j=0;j<NoOfEachMarkers[i];j++)
{
   if((BouSolver[i][j]->GetBoundaryType()) == "ConsT"){
         solution[(BouSolver[i][j]->GetFirstIndex())-1] = BouSolver[i][j]->GetTemperature1();
```

```
            solution[(BouSolver[i][j]->GetSecondIndex())-1] = BouSolver[i][j]->GetTemperature2();
          // PoiSolver[(BouSolver[i][j]->GetFirstIndex())]->setAsBouNode();
          // PoiSolver[(BouSolver[i][j]->GetSecondIndex())]->setAsBouNode();
        }



    }

/*for(int i=0;i<nMark;i++)
    for(int j=0;j<NoOfEachMarkers[i];j++)
    {
if((BouSolver[i][j]->GetBoundaryType()) == "Convec"){
    if((PoiSolver[(BouSolver[i][j]->GetFirstIndex())]->getNodeType())!="ConsT")
        PoiSolver[(BouSolver[i][j]->GetFirstIndex())]->setAsCovecNode();
    if((PoiSolver[(BouSolver[i][j]->GetSecondIndex())]->getNodeType())!="ConsT")
        PoiSolver[(BouSolver[i][j]->GetSecondIndex())]->setAsCovecNode();
       }
       if((BouSolver[i][j]->GetBoundaryType()) == "HFlux"){
 if((PoiSolver[(BouSolver[i][j]->GetFirstIndex())]->getNodeType())!="ConsT")
     PoiSolver[(BouSolver[i][j]->GetFirstIndex())]->setAsHfluxNode();
 if((PoiSolver[(BouSolver[i][j]->GetSecondIndex())]->getNodeType())!="ConsT")
```

```
    PoiSolver[(BouSolver[i][j]->GetSecondIndex())]->setAsHfluxNode();
     }

   }   */



/* cout<< "initial solution "<<endl;
   for(int i=0;i<nPoin;i++) cout<<"initial solution at index "<<i+1<<" is "<<solution[i]<<endl;
   cout<< "end of initial solution "<<endl; */
  }


 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

 void Iterate()
 { double temp1 = time/minTimeStepValue;
  double temp2 = floor(time/minTimeStepValue);
  double theta = 0.5;
  lastTimeStep = 0;
  if((temp1-temp2)>0) { numberOfSteps = temp2 + 1; lastTimeStep = time - (temp2*minTimeStepValue); }
```

```
   else { numberOfSteps = temp2 ;}
   cout<<temp1<<endl;
   cout<<temp2<<endl;
   cout<<" minTimeStepValue "<<minTimeStepValue<<endl;
   cout << "numberOfSteps" << numberOfSteps<<endl;
   cout << "lastTimeStep " << lastTimeStep <<endl;

 //  cout<<" solution before multiplying "<<endl;
//for(int i=0;i<(nPoin);i++) cout<< solution[i]<<endl;

  int tempCount = 0;
  for(int i=0;i<nPoin;i++) {
     if((PoiSolver[i+1]->getNodeType())=="ConsT"){
   consTvector[tempCount]=solution[i];
tempCount +=1;
}

}

 //cout<<" consTvector "<<endl;
//for(int i=0;i<tempCount;i++) cout<< consTvector[i]<<endl;
```

```
   for(int i=0;i<nPoin;i++) for(int j=0;j<nPoin;j++){
  preConditionerMatrix[i][j] =
      (((theta)*assembledConductionMatrix[i][j])+((1/minTimeStepValue)*assembledCapacitanceMatrix[i][j]));}

   for(int i=0;i<nPoin;i++) for(int j=0;j<nPoin;j++){
 equationRhsMatrix[i][j] =
     (((-1)*(1-theta)*assembledConductionMatrix[i][j])+((1/minTimeStepValue)*assembledCapacitanceMatrix[i][j]));}
double sum1 = 0;
  for(int i=0;i<nPoin;i++){ sum1 = 0; for(int k=0;k<nPoin;k++) {sum1 += (equationRhsMatrix[i][k]*solution[k]);}
      solutiontemp[i]=sum1;}
  for(int i=0;i<nPoin;i++) solution[i]=solutiontemp[i];
 //   cout<<"i was here"<<endl;
  for(int i=0;i<nPoin;i++) solution[i]+=assembledRHSMatrix[i];
 // cout<<"i was here"<<endl;
/*cout<<" after multiplying solution "<<endl;
for(int i=0;i<(nPoin);i++) cout<< solution[i]<<endl;

   cout<<"preConditionerMatrix"<<endl;
for(int i=0;i<(nPoin);i++)
{for(int j=0;j<(nPoin);j++) cout<<preConditionerMatrix[i][j]<<" "; cout<<endl;}
```

```
cout<<" equationRhsMatrix"<<endl;
for(int i=0;i<(nPoin);i++)
{for(int j=0;j<(nPoin);j++) cout<<equationRhsMatrix[i][j]<<" "; cout<<endl;} */

  int rowCount = 0;
  for(int i=0;i<nPoin;i++) {
     int columnCount = 0;
     if((PoiSolver[i+1]->getNodeType())!="ConsT"){
   for(int j=0;j<nPoin;j++){
      if((PoiSolver[j+1]->getNodeType())!="ConsT"){
    reduced_preConditionerMatrix[rowCount][columnCount] = preConditionerMatrix[i][j];
    reduced_equationRhsMatrix[rowCount][columnCount] = equationRhsMatrix[i][j];
    columnCount +=1;
  } }
rowCount +=1;
}

}
//cout<<"i was here"<<endl;
  rowCount = 0;
  for(int i=0;i<nPoin;i++) {
```

```
     if((PoiSolver[i+1]->getNodeType())!="ConsT"){
   reduced_solution[rowCount]=solution[i];
rowCount +=1;
}

}

//cout<<" reduced_solution "<<endl;
//for(int i=0;i<(nPoin-numberOfConsT);i++) cout<<reduced_solution[i]<<endl;


int count1=0,count2=0;
for(int i=0;i<nPoin;i++) {
     if((PoiSolver[i+1]->getNodeType())=="ConsT") {consTindex[count1] = i;count1+=1;}
     else {non_consTindex[count2]=i;count2+=1;}
      }
for(int i=0;i<(numberOfConsT);i++){
   for(int j=0;j<(nPoin-numberOfConsT);j++){
      reduced_solution[j] -= preConditionerMatrix[non_consTindex[j]][consTindex[i]]*consTvector[i];
   }
```

```
}


//cout<<" reduced_solution after subtraction"<<endl;
//for(int i=0;i<(nPoin-numberOfConsT);i++) cout<< reduced_solution[i]<<endl;


  //LU DECOMPOSITION OF REDUCED PRECONDITIONER MATRIX

for(int i=0;i<(nPoin-numberOfConsT);i++) for(int j=0;j<(nPoin-numberOfConsT);j++)
   {if(i==j) Lmatrix[i][j]=1; Umatrix[i][j]=reduced_preConditionerMatrix[i][j];}
/*cout<<"reduced_preConditionerMatrix"<<endl;
for(int i=0;i<(nPoin-numberOfConsT);i++)
{for(int j=0;j<(nPoin-numberOfConsT);j++) cout<<reduced_preConditionerMatrix[i][j]<<" "; cout<<endl;} */

//cout<<"reduced_equationRhsMatrix"<<endl;
//for(int i=0;i<(nPoin-numberOfConsT);i++)
   //{for(int j=0;j<(nPoin-numberOfConsT);j++) cout<<reduced_equationRhsMatrix[i][j]<<" "; cout<<endl;}

//for(int i=0;i<(nPoin-numberOfConsT);i++)
double tempHere = 0;
```

```
for(int k=0;k<(nPoin-numberOfConsT);k++)
   for(int i=k+1;i<(nPoin-numberOfConsT);i++){

                        tempHere = (Umatrix[i][k]/Umatrix[k][k]);
      for(int j=k;j<(nPoin-numberOfConsT);j++){

            Umatrix[i][j] = Umatrix[i][j] - (tempHere)*Umatrix[k][j];

      }
       Lmatrix[i][k] = tempHere;
   }
/*cout<<"upper triangular matrix"<<endl;
for(int i=0;i<(nPoin-numberOfConsT);i++)
{for(int j=0;j<(nPoin-numberOfConsT);j++) cout<<Umatrix[i][j]<<" "; cout<<endl;}
cout<<"lower triangular matrix"<<endl;
for(int i=0;i<(nPoin-numberOfConsT);i++)
{for(int j=0;j<(nPoin-numberOfConsT);j++) cout<<Lmatrix[i][j]<<" "; cout<<endl;}
cout<<"multiplication of lower and upper triangular matrix"<<endl; */
 /*for(int i=0;i<(nPoin-numberOfConsT);i++) {

   for(int j=0;j<(nPoin-numberOfConsT);j++){
```

```
      double tempodele = 0;
      for(int k=0;k<(nPoin-numberOfConsT);k++){
            tempodele += Lmatrix[i][k]*Umatrix[k][j];
      }
    //  cout<<tempodele<<" ";

   }
//cout<<endl;
   }  */

 //cout<<" reduced_solution before the looping "<<endl;
//for(int i=0;i<(nPoin-numberOfConsT);i++) cout<< reduced_solution[i]<<endl;


  //foward substitution for every timestep in loop
 //backward substitution for every timestep in loop
//cout<<"reduced_solution "<<endl;
//for(int i=0;i<(nPoin-numberOfConsT);i++) cout<< reduced_solution[i]<<endl;
ofstream outf1("solution.dat", ios::app);
//ofstream outf2("paroutput.vtk", ios::app);
outf1.precision(6);
```

```
string prefix = "solution_";
string ext(".vtk");
int countOutput = 0;
for (int i1=0;i1<numberOfSteps;i1++)
   {
      //cout<<"reduced_solution at beginning of step "<<i1+1<<endl;
//for(int j=0;j<(nPoin-numberOfConsT);j++) cout<<reduced_solution[j]<<endl;

      for(int i=0;i<(nPoin-numberOfConsT);i++){forwardSubMatrix[i] = 0;backwardSubMatrix[i]=0;}
       //forward substitution
      forwardSubMatrix[0]=reduced_solution[0];
      double sum2=0;
       for(int i=1;i<(nPoin-numberOfConsT);i++){
         sum2=0;
         for(int j=0;j<i;j++){
            sum2 += Lmatrix[i][j]*forwardSubMatrix[j];
         }
         forwardSubMatrix[i]=reduced_solution[i]-sum2;
       }
     //  cout<<"forwardSubMatrix in step "<<i1+1<<endl;
     // for(int i=0;i<(nPoin-numberOfConsT);i++) cout<< forwardSubMatrix[i]<<endl;
```

```
      //backward substitution
backwardSubMatrix[(nPoin-numberOfConsT)-1] =
    forwardSubMatrix[(nPoin-numberOfConsT)-1]/Umatrix[(nPoin-numberOfConsT)-1][(nPoin-numberOfConsT)-1];

if((nPoin-numberOfConsT)>1)
for(int i=((nPoin-numberOfConsT)-2);i>=0;i--){
      double sum = 0;
      for(int j=i+1;j<(nPoin-numberOfConsT);j++){
         sum += Umatrix[i][j]*backwardSubMatrix[j];
      }
      backwardSubMatrix[i]= ((forwardSubMatrix[i]-sum)/Umatrix[i][i]);
}
//for(int i=0;i<nPoin;i++) solution[i]=0;
//cout<<"backwardSubMatrix in step "<<i1+1<<endl;
//for(int i=0;i<(nPoin-numberOfConsT);i++) cout<<backwardSubMatrix[i]<<endl;
count1=0;count2=0;
for(int i=0;i<nPoin;i++) {
     if((PoiSolver[i+1]->getNodeType())=="ConsT") {solution[i]=consTvector[count1];count1+=1;}
     else {solution[i]=backwardSubMatrix[count2];count2+=1;}
      }
```

```
//outf1<<"Solution at time step "<<i1+1<<endl;
//for(int i=0;i<(nPoin);i++){
//outf1<<PoiSolver[i+1]->GetIndex()<<" "<<PoiSolver[i+1]->GetXcord()<<" "<<PoiSolver[i+1]->GetYcord()<<"
    "<<solution[i]<<endl;
//}

if((i1%10)==0){
stringstream ss;
       ss << prefix << std::setfill('0') << std::setw(4) << countOutput << ext;
string temp;
ss >> temp;

ofstream outf2(ss.str().c_str());
       if ( !outf2 )
       {
           std::cerr << "Error: failed to create file " << ss.str() << '\n';
           break;
       }
```

```
 outf2<<"# vtk DataFile Version 3.0"<<endl;
 outf2<<"vtk output"<<endl;
 outf2<<"ASCII"<<endl;
 outf2<<"DATASET UNSTRUCTURED_GRID"<<endl;
 outf2<<"POINTS "<<nPoin<<" double"<<endl;
 for(int i=0;i<(nPoin);i++){
outf2<<PoiSolver[i+1]->GetXcord()<<" "<<PoiSolver[i+1]->GetYcord()<<" "<<"0 ";
}
outf2<<endl;
outf2<<"CELLS "<<nElem<<" "<<nElem*4<<endl;
for(int i=1;i<=nElem;i++) outf2<<"3 "<<((EleSolver[i]->GetFirstIndex())-1)<<"
    "<<((EleSolver[i]->GetSecondIndex())-1)<<" "<<((EleSolver[i]->GetThirdIndex())-1)<<" ";
outf2<<endl;
outf2<<"CELL_TYPES "<<nElem<<endl;
for(int i=1;i<=nElem;i++) outf2<<"5 ";
outf2<<"POINT_DATA "<<nPoin<<endl;
outf2<<endl;
outf2<<"SCALARS Temperature double 1"<<endl;
outf2<<"LOOKUP_TABLE default"<<endl;
for(int i=0;i<(nPoin);i++) outf2<<solution[i]<<" ";
```

```
 countOutput+=1;
}

 //cout<<"solution at step "<<i1+1<<endl;
//for(int i=0;i<(nPoin);i++) cout<<solution[i]<<endl;
for(int i=0;i<(nPoin);i++) solutiontemp1[i]=0;
for(int i=0;i<(nPoin);i++)
{ double sum =0;
for(int k=0;k<(nPoin);k++) sum += equationRhsMatrix[i][k]*solution[k];
   solutiontemp1[i]=sum;}

for(int i=0;i<(nPoin);i++) solution[i]=solutiontemp1[i];

for(int i=0;i<(nPoin);i++) solution[i]+=assembledRHSMatrix[i];
//cout<<"solution after multiplying with equationRhsMatrix at step "<<i1+1<<endl;
//for(int i=0;i<(nPoin);i++) cout<<solution[i]<<endl;
for(int i=0;i<(nPoin-numberOfConsT);i++) reduced_solution[i] = 0;
   int rowCount = 0;
  for(int i=0;i<nPoin;i++) {
     if((PoiSolver[i+1]->getNodeType())!="ConsT"){
```

```
   reduced_solution[rowCount]=solution[i];
rowCount +=1;
}

}

for(int i=0;i<(numberOfConsT);i++){
   for(int j=0;j<(nPoin-numberOfConsT);j++){
      reduced_solution[j] -= (preConditionerMatrix[non_consTindex[j]][consTindex[i]]*consTvector[i]);
   }

}
//cout<<"reduced_solution at end of step "<<i1+1<<endl;
//for(int j=0;j<(nPoin-numberOfConsT);j++) cout<<reduced_solution[j]<<endl;
cout<<"S.No "<< i1+1 <<" "<<numberOfSteps<<endl;
//for(int i=1;i<(nPoin-numberOfConsT);i++) cout<< backwardSubMatrix[i]<<endl;


   }
//cout<<"final solution "<<endl;
//for(int i=0;i<(nPoin-numberOfConsT);i++) cout<< backwardSubMatrix[i]<<endl;
```

```
   cout<<"numberOfConsT is "<<numberOfConsT<<endl;
    cout<<"nPoin-numberOfConsT is "<<nPoin-numberOfConsT<<endl;
 //cout<<"consTvector "<<endl;
 //for(int i=0;i<(numberOfConsT);i++) cout<< consTvector[i]<<endl;
 //cout<<"consTindex "<<endl;
 //for(int i=0;i<(numberOfConsT);i++) cout<< consTindex[i]+1<<endl;
  }
};
```

Chapter 4

Results & Conclusion

4.1 Results

The developed code is applied to a Square geometry meshed with triangular elements as shown in the figure. The mesh has 1800 triangular elements with 961 points.

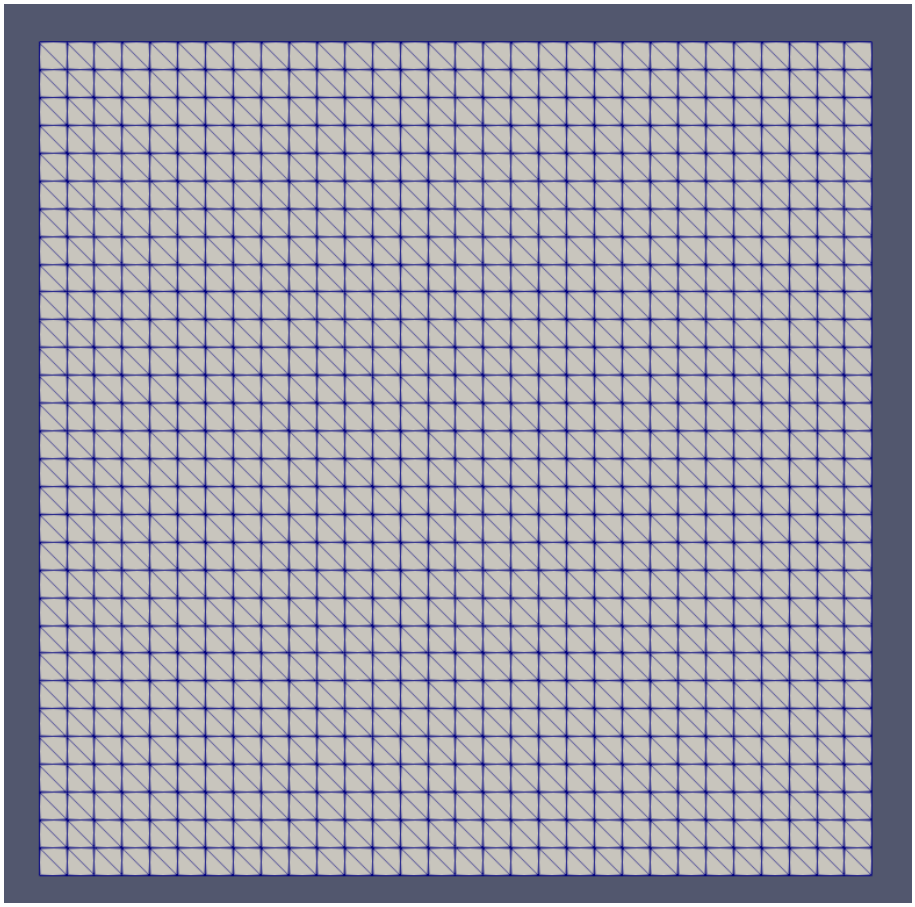

Figure 4.1: Square geometry with triangular mesh

Different boundary conditions are applied to test the code.

- Case 1: Bottom Const Temp 60, Right-Top-Left Const Temp 10
- Case 2: Bottom Const Temp 60, Right Convection Exter Temp 100, Top Insulation, Left Const Temp 10
- Case 3: Bottom Const Temp 60, Right Covection Exter Temp 100, Top Insulation, Left Insulation
- Case 4: Bottom Insulation, Right Covection Exter Temp 100, Top Insulation, Left Const Temp 60

The steady state temperature contour plots for all the four cases are shown below.

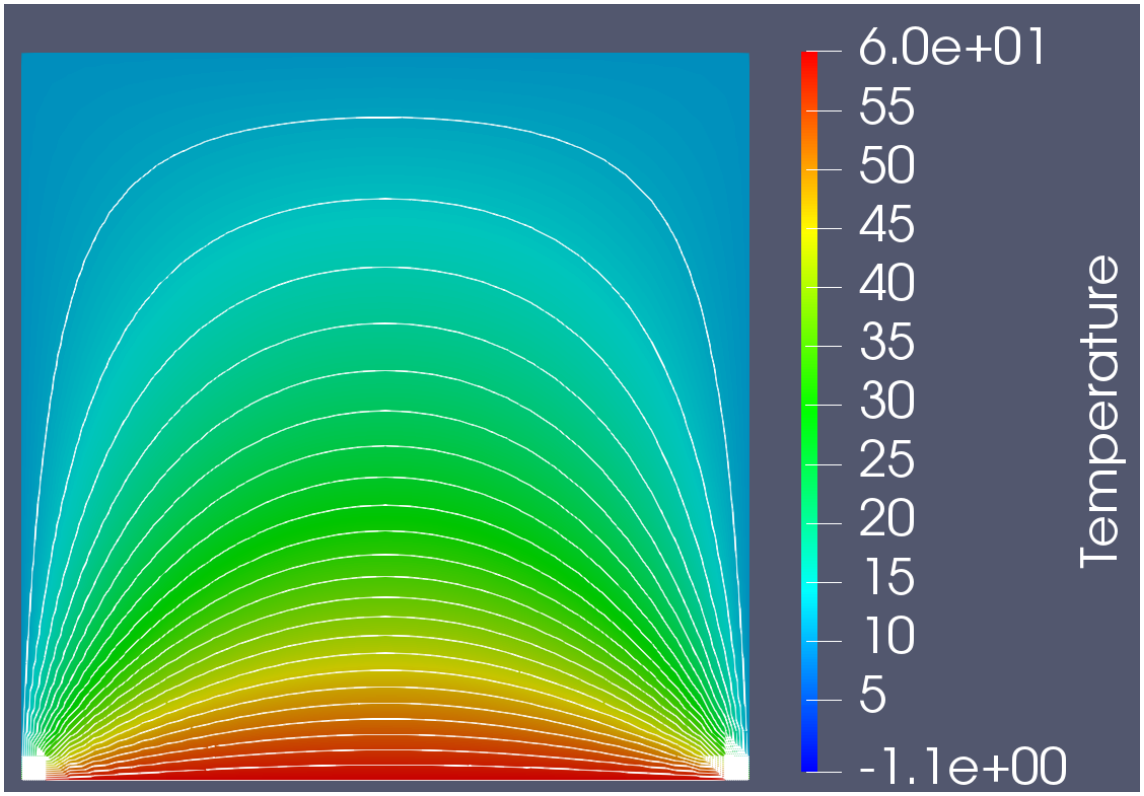

Figure 4.2: Contour plot for Case 1

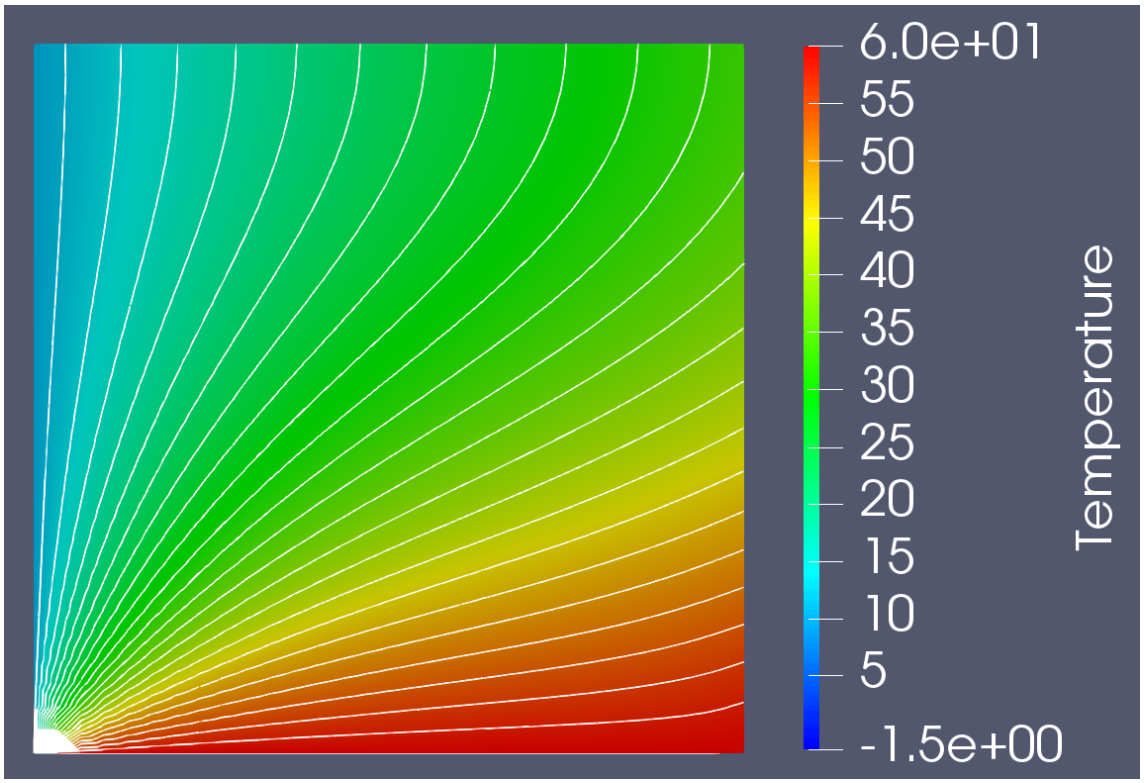

Figure 4.3: Contour plot for Case 2

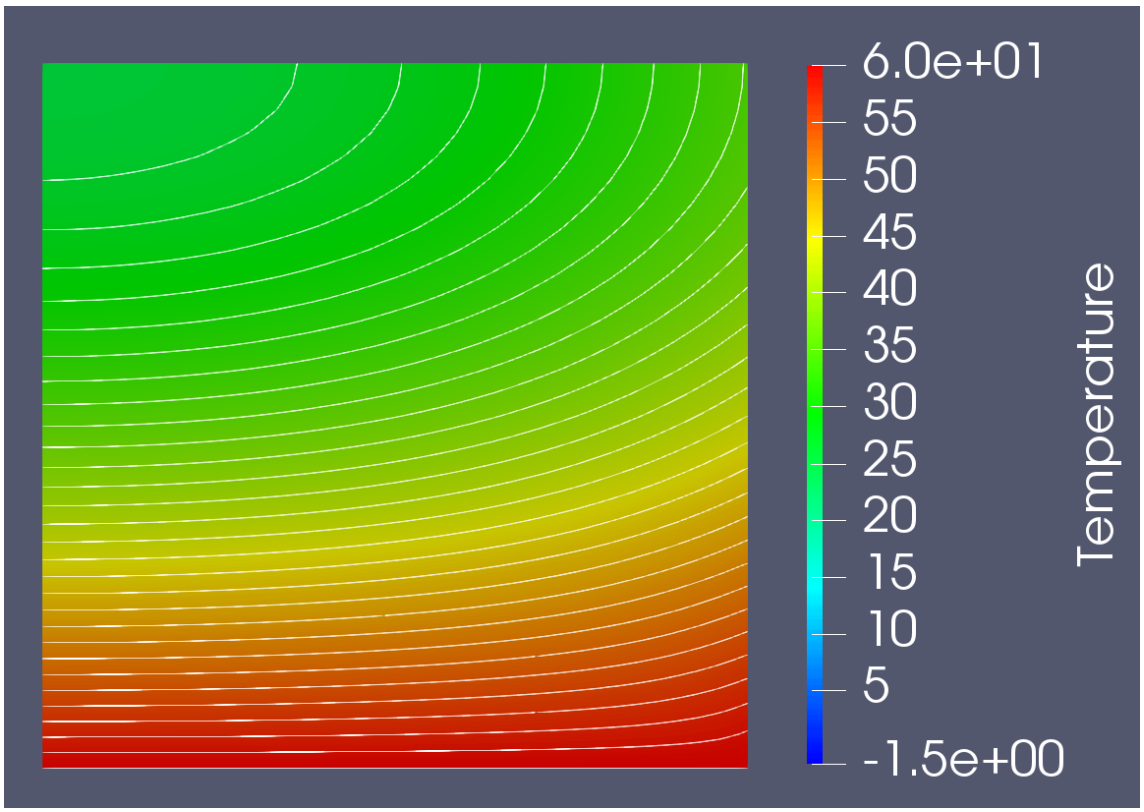

Figure 4.4: Contour plot for Case 3

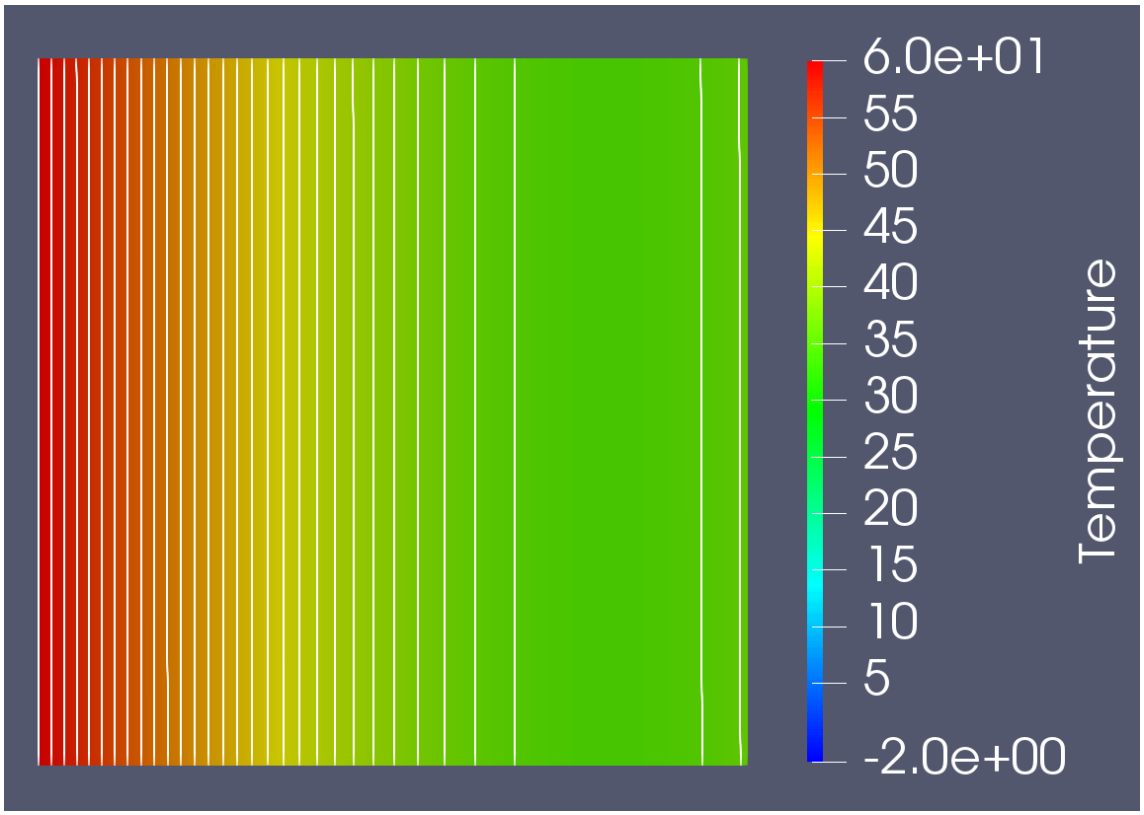

Figure 4.5: Contour plot for Case 4

4.2 Conclusion

Here we can see that the code works well for different cases. We can also observe where ever the insulation boundary is present the temperature contours end normally into that boundary which further verifies the correctness of code. Thus a object oriented c++ code for transient heat conduction is successfully developed. This code can handle triangular meshes of any arbitrary 2-d geometry with different boundary conditions like Convection, Constant Temperature, Heat flux. It outputs solution in .vtk format at different time steps which can viewed as an animation in paraview.

Appendix A

Appendix

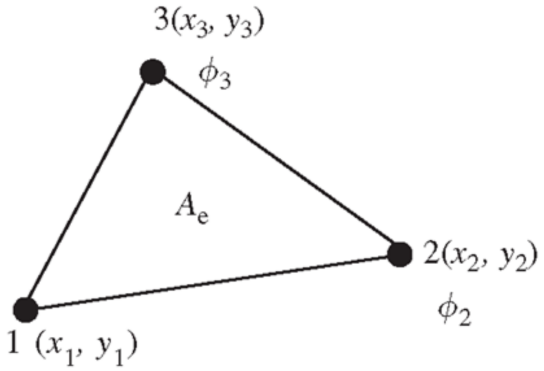

Figure A.1: Triangular Element

A.1 B matrix

The B matrix for triangular element is given by

$$B = \frac{1}{2\Delta}\begin{bmatrix} y_2 - y_3 & y_3 - y_1 & y_1 - y_2 \\ x3 - x2 & x_1 - x_3 & x_2 - x_1 \end{bmatrix} \tag{A.1}$$

Here Δ is the area of the triangular element.

A.2 C matrix

The capacitance matrix for triangular element is given by

$$C = \frac{\rho ct\Delta}{12}\begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} \tag{A.2}$$

Here ρ is the density, c is the specific heat, t is the thickness of the element, Δ is the area of the element.

A.3 Load vectors due to heat flux and convection

$$
\begin{aligned}
K_h &= \frac{htl_{12}}{6}\begin{bmatrix}2 & 1\\ 1 & 2\end{bmatrix}\\
R_h &= \frac{hT_e tl_{12}}{2}\begin{bmatrix}1\\ 1\end{bmatrix}\\
R_h &= \frac{q_s tl_{12}}{2}\begin{bmatrix}1\\ 1\end{bmatrix}
\end{aligned}
\tag{A.3}
$$

Here h is the convection heat transfer coefficient, t is the thickness, T_e is the surrounding temperature or exposed temperature, l_{12} is the length of the edge element, q_s is the given heat flux lost.

Bibliography

[1] Finite Element Method for Engineers *by Kenneth H.Huebner, Donald L, E. Smith, Ted G.*

[2] Computational Heat Transfer, Second Edition *by Yogesh Jaluria, Kenneth E. Torrance*