SARDAR VALLABHBHAI NATIONAL INSTITUTE OF TECHNOLOGY, SURAT

Computational Fluid Dynamics ASSIGNMENT   
(UNDER THE GUIDANCE OF DR. JYOTRIMAY BANERJEE)

Implementation and analysis of linear spatial variation in Material properties in 1D ( ) for an implicit time marched approach.

By

TANMAY SARAIYA (U15ME049)  
SAGAR ROHIT (U15ME069)  
ROHIT KRISHNAN V.(U15ME065)  
ARVIND REDDY SEELAM (U15ME063)  
KAMAL KRISHNA (U15ME046)



Question:

To model a generic implicit time marching method for 1D heat transfer in a pin fin under various boundary conditions.

Approach:

Since there is no advection term in this flow we assume diffusion to be the only driving term in the energy equation.

The governing equation:

The governing equation applicable here is that of heat transfer through a pin fin.





where,

The conduction term was modeled by integrated the whole equation over the cell volume which later transcended to integral over the surface area by Gauss Divergence Theorem).

The final governing equation is of the form 

Where,









Where

 are the cell center coordinates wrt to ‘x’. 

Consequently the boundary conditions are applied and the following results are obtained:

**CODE:**

The code consists of multiple files.

Main.cpp is the main file.

Flow.cpp is to manage the flow of the program and call the functions as required.(Manager file)

Grid\_gen.cpp is used to generate the initial grid as per the requirements (@T=0, Generator).

Res.cpp is used to run all the calculations and ask the user input values (The Brainy One).

The user is first asked to input the width of the fin followed by the number of divisions.

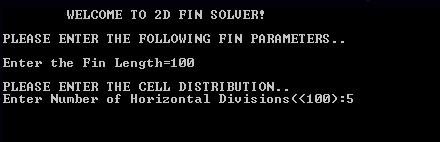
In the next step the user is supposed to select the boundary condition type from the menu given.

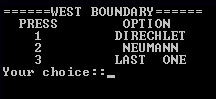
The grid is then initialized and the user is then asked to input parameters such as

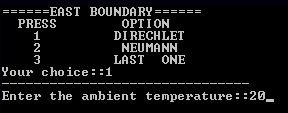


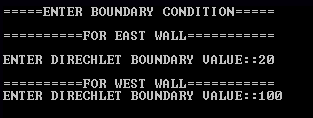
The results are then displayed stored in a text file with coordinate variation and are ready for further processing.

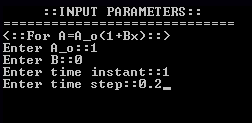
**SCREENSHOTS:**

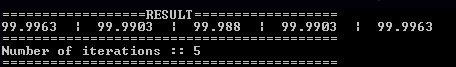
****

****

****

****





The code is as follows.

**::HEADER FILES::**

**FLOW.H**

#ifndef flow

#define flow

extern int width;

extern int M,e\_typ,w\_typ;

void Fin\_flow(void);

#endif

**GRID\_GEN.H**

#ifndef grid\_gen

#define grid\_gen

void grid\_write(void);

void init\_grid(void);

#endif

**RES.H**

#ifndef res

#define res

struct grid{

double data;

double del\_x;

double x\_e;

double x\_w;

double ap;

double ae;

double aw;

double lhs;

};

extern double T\_inf;

void iterations(void);

#endif

**::MAIN FILES::**

**MAIN.CPP**

#include "flow.h"

int main()

{

Fin\_flow();//calling the order of flow

}

**FLOW.CPP**

#include "res.h"

#include "grid\_gen.h"

#include "flow.h"

#include <iostream>

#include <stdlib.h>

using namespace std;

grid cell\_temp[100];

int width;

int M,e\_typ,w\_typ;

int ch\_disp()

{

cout<<" PRESS OPTION"<<endl;

cout<<" 1 DIRECHLET"<<endl;

cout<<" 2 NEUMANN"<<endl;

cout<<" 3 LAST ONE"<<endl;

int ch,flag=1;

while(flag)

{

cout<<"Your choice::";

cin>>ch;

if(ch>0 && ch<4)

flag=0;

else

cout<<"Invalid! Please enter again.."<<endl;

}

return ch;

}

void Fin\_flow()

{

system("cls");

cout<<"\tWELCOME TO 2D FIN SOLVER!"<<endl<<endl;

//DEFINE GRID

cout<<"PLEASE ENTER THE FOLLOWING FIN PARAMETERS.."<<endl<<endl;

cout<<"Enter the Fin Length=";

cin>>width;

//DEFINE DISTRIBUTION

cout<<"\nPLEASE ENTER THE CELL DISTRIBUTION.."<<endl;

cout<<"Enter Number of Horizontal Divisions(<100):";

cin>>M;

system("cls");

cout<<"======WEST BOUNDARY======"<<endl;

w\_typ=ch\_disp();

system("cls");

cout<<"======EAST BOUNDARY======"<<endl;

e\_typ=ch\_disp();

///////Generate and initialize grid with the following parameters# DEFAULT

init\_grid();

/////////////SOLVE//////////

iterations();

}

**GRID\_GEN.CPP**

#include "res.h"

#include "flow.h"

#include <fstream>

#include <iostream>

using namespace std;

/////////GLOBAL VARS/////////

grid cell[100];//no of divisions

/////////////////////////////////////INITIAL CELL SETUP/////////////////////////////////////

void assign\_dim(grid &temp)

{

temp.del\_x=(double)width/(double)M;

}

void assign\_params(grid &temp)//set distance of cell edges from CG; ###for rectangular cells ONLY

{

temp.x\_e = temp.del\_x/((double)2);

temp.x\_w = temp.del\_x/((double)2);

}

void grid\_write()

{

remove("grid0.dat");

ofstream fout("grid0.dat",ios::binary);

for(int i=0;i<M;i++)

fout.write((char\*)&cell[i],sizeof(grid));

fout.close();

}

////////////////////////////////////////////USER FUNCTIONS////////////////////////////////////////////

void init\_grid()

{

cout<<"-------------------------------"<<endl;

cout<<"Enter the ambient temperature::";

cin>>T\_inf;

///DEFINE GEOMETRICAL PARAMS

for(int i=0;i<M;i++)

{

cell[i].data=T\_inf;// eequals T\_inf

assign\_dim(cell[i]);//assign del\_x;

assign\_params(cell[i]);//assign values of x\_w,x\_e;

cell[i].ap=0;

cell[i].ae=0;

cell[i].aw=0;

cell[i].lhs=0;

}

/// save the intitial matrix

grid\_write();

}

**RES.CPP**

#include "res.h"

#include "grid\_gen.h"

#include "flow.h"

#include <iostream>

#include <fstream>

#include <stdlib.h>

#include <math.h>

using namespace std;

grid sol\_cell[70][100];//steps \* grid

double matA[100][100];

double matB[100];

double EAST,WEST;

double alph\_0=1,B=0,del\_t,T\_inf=0;//Default conditions set to isotropic Alpha =1;

//DATA MANIPULATION FUNCTIONS

void assign\_coeffs(int i,int t)//called while solving for each cell

{

double P,E,W,L;

if(i==0)

{

W= -1 \* alph\_0;

// (A\_0)

E =-1 \* alph\_0 \* (1 + B \* sol\_cell[0][i].del\_x \* 0.5f);

// (A\_0 (1 + B \* del\_x (p +0.5))

}

else if(i==M-1)

{

W=-1 \* alph\_0 \* (1 + B \* sol\_cell[0][i].del\_x \* (M - 2.5f));

// (A\_0\* (1 + B \* del\_t (p -1.5))

E = -1 \* alph\_0;

// (A\_0)

}

else

{

W= -1 \* alph\_0 \* (1 + B \* sol\_cell[0][i].del\_x \* ((double)i - 1.5f));

// A\_0 \* (1 + B \* del\_t (p -1.5))

E =-1 \* alph\_0 \* (1 + B \* sol\_cell[0][i].del\_x \* ((double)i + 0.5f));

// A\_0 \* (1 + B \* del\_t (p +0.5))

}

L= sol\_cell[t-1][i].data;

double theta=0;

theta = del\_t / (double)pow(sol\_cell[0][i].del\_x,2) ;

E \*= theta;

W \*= theta;

P = (double)1 - (E + W);

if(i==0)

{

if(w\_typ==1)//direchlet

{

sol\_cell[t][i].aw = 0;

sol\_cell[t][i].ap = P-W;

sol\_cell[t][i].ae = E;

sol\_cell[t][i].lhs = sol\_cell[t-1][i].data -(double)2\*W\*WEST;

}

else if(w\_typ==2)//neumann

{

sol\_cell[t][i].aw = 0;

sol\_cell[t][i].ap = W + P;

sol\_cell[t][i].ae = E;

sol\_cell[t][i].lhs = sol\_cell[t-1][i].data - W\*WEST\*sol\_cell[0][i].del\_x;

}

else if(w\_typ==3)//robins

{

theta = 2 / sol\_cell[0][i].del\_x \* WEST;

sol\_cell[t][i].aw = 0;

sol\_cell[t][i].ap = (W \* (theta - 1 / theta + 1)) + P;

sol\_cell[t][i].ae = E;

sol\_cell[t][i].lhs = sol\_cell[t-1][i].data - (((double)2 / theta + 1) \* T\_inf \* W);

}

}

else if(i==M-1) ///last box

{

if(e\_typ==1)//direchlet

{

sol\_cell[t][i].aw = W;

sol\_cell[t][i].ap = P-E;

sol\_cell[t][i].ae = 0;

sol\_cell[t][i].lhs = sol\_cell[t-1][i].data -(double)2\*E\*EAST;

}

else if(e\_typ==2)//neumann

{

sol\_cell[t][i].aw = W;

sol\_cell[t][i].ap = P+E;

sol\_cell[t][i].ae = 0;

sol\_cell[t][i].lhs = sol\_cell[t-1][i].data + E\*EAST\*sol\_cell[0][i].del\_x;

}

else if(e\_typ==3)//robins

{

theta = 2 / sol\_cell[0][i].del\_x \* EAST;

sol\_cell[t][i].aw = W;

sol\_cell[t][i].ap = ((theta - 1 / theta + 1) \* E) + P;

sol\_cell[t][i].ae = 0;

sol\_cell[t][i].lhs = sol\_cell[t-1][i].data - (((double)2 / theta + 1) \* T\_inf \* E);

}

}

else

{

sol\_cell[t][i].aw = W;

sol\_cell[t][i].ap = P;

sol\_cell[t][i].ae = E;

sol\_cell[t][i].lhs = sol\_cell[t-1][i].data;

}

}

void gen\_TDMA(int t)

{

for(int i=0;i<M;i++)

{

if(i!=0)

matA[i][i-1]=sol\_cell[t][i].aw;

matA[i][i]=sol\_cell[t][i].ap;

if(i!=M-1)

matA[i][i+1]=sol\_cell[t][i].ae;

//setting up matB too!

matB[i]=sol\_cell[t][i].lhs;

}

}

void solve\_TDMA(int t)//for time step t

{

system("cls");

for(int i=1;i<M;i++)//leave 0th row

{

double frac = (double)matA[i][i-1] / (double)matA[i-1][i-1];

matA[i][i-1] = matA[i][i-1] - frac \* matA[i-1][i-1];

matA[i][i] = matA[i][i] - frac \* matA[i-1][i];

matB[i] = matB[i] - frac \* matB[i-1];

}

//calc values

sol\_cell[t][M-1].data = (double)matB[M-1] / (double)matA[M-1][M-1];

for(int i=M-2;i>=0;i--)

sol\_cell[t][i].data = (double)((double)matB[i] - (double)matA[i][i+1] \* sol\_cell[t][i+1].data ) / (double)matA[i][i] ;

}

////DATA OUTPUT FUNCTIONS

void grid\_read()//read grid and init matA and matB

{

grid temp;

int i=0;

ifstream fin("grid0.dat",ios::binary);

while(fin.read((char\*)&temp,sizeof(grid)))

{

sol\_cell[0][i]=temp;

i++;

}

fin.close();

for(int j=0;j<i;j++)

{

for(int k=0;k<i;k++)

matA[j][k] = 0;

matB[j]=0;

}

}

void display(int x)

{

for(int i=0;i<M;i++)

{

cout<<sol\_cell[x][i].data<<" ";

if(i!=M-1)

cout<<"| ";

}

cout<<endl;

}

void notepad(int x)//writes final grid RESULT

{

ofstream fout("RESULT.txt");

for(int i=0;i<M;i++)

{

fout<<(i\*sol\_cell[0][0].del\_x)+(sol\_cell[0][0].del\_x/2)<<"\t"<<sol\_cell[x][i].data<<endl;

}

fout.close();

}

////SET UP THE USER PREFERENCES

double ret\_bound\_val(int x)

{

double inp;

if(x==1)

cout<<"ENTER DIRECHLET BOUNDARY VALUE::";

else if(x==2)

cout<<"ENTER NEUMANN BOUNDARY VALUE::";

else

cout<<"ENTER ROBIN'S BOUNDARY VALUE::k(T-To)/nEnter k::";

cin>>inp;

return inp;

}

void solve\_setup()

{

//ask input values for all boundaries as per conditions

system("cls");

cout<<"=====ENTER BOUNDARY CONDITION====="<<endl;

cout<<"\n==========FOR WEST WALL==========="<<endl;

WEST=ret\_bound\_val(w\_typ);

cout<<"\n==========FOR EAST WALL==========="<<endl<<endl;

EAST=ret\_bound\_val(e\_typ);

}

/// ITERATE THE GRID

void iterations()

{

grid\_read();//read the grid from the saved file.

solve\_setup();//ask for solving input values

int ttl\_count,x=1;//ttl\_count keeps track of the total number of iterations to be performed

double Time=0.2f,D\_t=0.2f;//user input #time= time wanted #D\_t =time step required

system("cls");

cout<<" ::INPUT PARAMETERS::"<<endl;

cout<<"============================="<<endl;

cout<<"<::For A=A\_o(1+Bx)::>"<<endl;

cout<<"Enter A\_o::";

cin>>alph\_0;

cout<<"Enter B::";

cin>>B;

cout<<"Enter time instant::";

cin>>Time;

cout<<"Enter time step::";

cin>>D\_t;

ttl\_count=(double)Time/D\_t+2;

while(x<ttl\_count)

{

del\_t=D\_t\*x;

cout<<"del\_t::"<<del\_t<<endl;//comment for final

for(int i=0;i<M;i++)

assign\_coeffs(i,x);

gen\_TDMA(x);

solve\_TDMA(x++);

}

notepad(--x);

remove("grid0.dat");

system("cls");

cout<<"==================RESULT=================="<<endl;

display(x);

cout<<"=========================================="<<endl;

cout<<"Number of iterations :: "<<x<<endl;

cout<<"=========================================="<<endl;

}

For any queries or suggestions  
feel free to email:: tanmaysaraiya@gmail.com

THE END

THANK YOU!