Computational Fluid Dynamics ASSIGNMENT

By

Tanmay Saraiya  
Sagar Rohit  
Rohit Krishnan  
Arvind

Question:

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

The governing equation:

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

k.Ac(d2 T/dx2) – hp (T – To) = 0

The conduction term was modeled using (Alpha)(dT/dx) = dT/dt (integrated over the cell surface area).  
where Alpha = K / (rho \* Cp).

The final governing equation is of the form ap.Tpn+1 + aw.Twn+1 + ae.Ten+1 = Tpn - L

Where,

ap=1 + (( (A\_0 / (del\_x^2)) \* (2 + B \* (Xw+Xe)) ) + (Konstt \* A\_0 \* (1 + B\* (Xp) ) ) )\*del\_t

aw=( ((A\_0 \* B) / (2 \* del\_x)) - ( (A\_0/(del\_x^2)) \* (1 + (B \* Xw )) ) )\*del\_t;

ae=-1 \* (( ((A\_0 \* B)/(2 \* del\_x)) - ( (A\_0/(del\_x^2)) \* (1 + (B\*Xe)) ) ) \* del\_t)

L= (hP/kAc) \* A0 \* T\_inf \* (1 + B \* del\_x \* Xp)

Where

Xp, Xe, Xw are the cell center coordinates wrt to ‘X’. It is being used to calculate the Alpha variation.

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

**RESULTS:**

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 per order.

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

Res.cpp is used to run all the calculations and ask the user input values.

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

Surrounding temp (T\_inf)

hP/KA value (surrounding medium)

Alpha\_zero

Beta

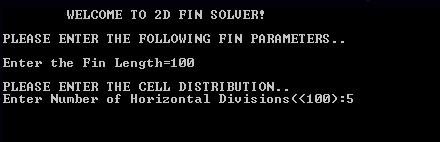
The time of result

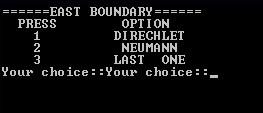
And the increment in time step.

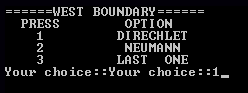
The results are then stored in a text file and ready for further processing.

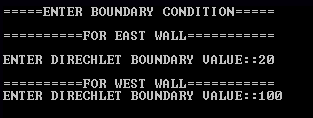
#the default input values are set for a long steel fin in air at 20C.

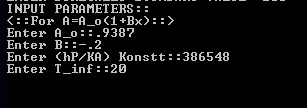
**SCREENSHOTS:**

****

****

****

****

****



The code is as follows.

**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;

};

void iterations(void);

#endif

**MAIN.CPP**

#include "flow.h"

int main()

{

Fin\_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;

cout<<"Your choice::";

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

e\_typ=ch\_disp();

system("cls");

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

w\_typ=ch\_disp();

///////Generate and initialize grid with the following params # 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()

{

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()

{

///DEFINE GEOMETRICAL PARAMS

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

{

cell[i].data=0;

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;

}

/// write the matrix

grid\_write();

}

**RES.CPP**

#include "res.h"

#include "grid\_gen.h"

#include "flow.h"

#include <iostream>

#include <fstream>

#include <stdlib.h>

using namespace std;

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

double matA[100][100];

double matB[100];

double EAST,WEST;

double psy,theta;

double alph\_0=.9387,B=-0.2,Konstt=386548,del\_t,T\_inf=20;

void gen\_TDMA(int t)

{

//MAT[A]\*MAT[X]=MAT[B]

///(MAT[X])value=sol\_cell[i].data

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

{

double P,E,W,L;

if(i==0)

{

P= 1 + (((double)alph\_0/(double)(sol\_cell[0][i].del\_x\*sol\_cell[0][i].del\_x)) \* (2 + (B \* sol\_cell[0][i].del\_x \* ((double)i + 0.5f))) + (Konstt \* alph\_0 \* (1 + B \* sol\_cell[0][i].del\_x \* ((double)i - 0.5f ))))\*del\_t;

//1 + (( (A\_0 / (del\_x^2)) \* (2 + B \* (Xe)) ) + (Konstt \* A\_0 \* (1 + B\* (Xp) ) ) )\*del\_t

W= ((double)(alph\_0 \* B) / (double)(2 \* sol\_cell[0][i].del\_x)) \* del\_t;

//( ((A\_0 \* B) / (2 \* del\_x)) - ( (A\_0/(del\_x^2)) \* (1 + (B \* Xw )) ) )\*del\_t;

E = -1 \* ((((double)(alph\_0 \* B)/(double)(2 \* sol\_cell[0][i].del\_x)) - (((double)alph\_0/(double)(sol\_cell[0][i].del\_x\*sol\_cell[0][i].del\_x))\*(1 + (B\* sol\_cell[0][i].del\_x \* ((double)i + 0.5f))))) \* del\_t);

// -1 \* (( ((A\_0 \* B)/(2 \* del\_x)) - ( (A\_0/(del\_x^2)) \* (1 + (B\*Xe)) ) ) \* del\_t);

}

else if(i==M-1)

{

P= 1 + (((double)alph\_0/(double)(sol\_cell[0][i].del\_x\*sol\_cell[0][i].del\_x)) \* (2 + (B \* sol\_cell[0][i].del\_x \* ((double)i-1.5f))) + (Konstt \* alph\_0 \* (1 + B \* sol\_cell[0][i].del\_x \* ((double)i - 0.5f ))))\*del\_t;

//1 + (( (A\_0 / (del\_x^2)) \* (2 + B \* (Xw)) ) + (Konstt \* A\_0 \* (1 + B\* (Xp) ) ) )\*del\_t

W= (((double)(alph\_0 \* B) / (double)(2 \* sol\_cell[0][i].del\_x)) - (((double)alph\_0/(double)(sol\_cell[0][i].del\_x \* sol\_cell[0][i].del\_x)) \* (1 + (B \* sol\_cell[0][i].del\_x \* ((double)i-1.5f))))) \* del\_t;

//( ((A\_0 \* B) / (2 \* del\_x)) - ( (A\_0/(del\_x^2)) \* (1 + (B \* Xw )) ) )\*del\_t;

E = -1 \* (((double)(alph\_0 \* B)/(double)(2 \* sol\_cell[0][i].del\_x))\* del\_t);

// -1 \* (( ((A\_0 \* B)/(2 \* del\_x)) - ( (A\_0/(del\_x^2)) \* (1 + (B\*Xe)) ) ) \* del\_t);

}

else

{

P= 1 + (((double)alph\_0/(double)(sol\_cell[0][i].del\_x\*sol\_cell[0][i].del\_x)) \* (2 + (B \* sol\_cell[0][i].del\_x \* (2\*i) - 1 )) + (Konstt \* alph\_0 \* (1 + B \* sol\_cell[0][i].del\_x \* ((double)i - 0.5f ))))\*del\_t;

//1 + (( (A\_0 / (del\_x^2)) \* (2 + B \* (Xw+Xe)) ) + (Konstt \* A\_0 \* (1 + B\* (Xp) ) ) )\*del\_t

W= (((double)(alph\_0 \* B) / (double)(2 \* sol\_cell[0][i].del\_x)) - (((double)alph\_0/(double)(sol\_cell[0][i].del\_x \* sol\_cell[0][i].del\_x)) \* (1 + (B \* sol\_cell[0][i].del\_x \* ((double)i-1.5f))))) \* del\_t;

//( ((A\_0 \* B) / (2 \* del\_x)) - ( (A\_0/(del\_x^2)) \* (1 + (B \* Xw )) ) )\*del\_t;

E = -1 \* ((((double)(alph\_0 \* B)/(double)(2 \* sol\_cell[0][i].del\_x)) - (((double)alph\_0/(double)(sol\_cell[0][i].del\_x\*sol\_cell[0][i].del\_x))\*(1 + (B\* sol\_cell[0][i].del\_x \* ((double)i + 0.5f))))) \* del\_t);

// -1 \* (( ((A\_0 \* B)/(2 \* del\_x)) - ( (A\_0/(del\_x^2)) \* (1 + (B\*Xe)) ) ) \* del\_t);

}

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

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 - L -2\*W\*WEST;

}

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

{

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 - L - W\*WEST\*sol\_cell[0][i].del\_x;

}

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

{

theta = (double)(2 + WEST \*sol\_cell[0][i].del\_x)/(double)(2 - WEST \*sol\_cell[0][i].del\_x);

psy = (double)(2 \* T\_inf)/(double)(2 - WEST \*sol\_cell[0][i].del\_x);

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

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

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

sol\_cell[t][i].lhs = sol\_cell[t-1][i].data - L + W \* psy;

}

}

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 - L -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 - L - E\*EAST\*sol\_cell[0][i].del\_x;

}

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

{

theta = (double)(2 + EAST \*sol\_cell[0][i].del\_x)/(double)(2 - EAST \*sol\_cell[0][i].del\_x);

psy = (double)(2 \* T\_inf)/(double)(2 - EAST \*sol\_cell[0][i].del\_x);

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

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

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

sol\_cell[t][i].lhs = sol\_cell[t-1][i].data - L + E \* psy;

}

}

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 - L;

}

}

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

{

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

{

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

matA[i][i] = (frac \* matA[i][i])- matA[i][i+1];//diagonal terms

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

matA[i][i+1]\*= frac;//east to diagonal term

}

//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] ;

}

}

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 EAST WALL==========="<<endl<<endl;

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

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

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

}

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<<"\t";

}

cout<<endl;

}

void notepad(int x)

{ //writes final grid RESULT

ofstream fout("RESULT.txt");

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

{

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

}

fout.close();

}

void iterations()

{

grid\_read();

solve\_setup();

int ttl\_count,x=1;

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

ttl\_count=Time/D\_t+1;

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

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

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

cin>>alph\_0;

cout<<"Enter B::";

cin>>B;

cout<<"Enter (hP/KA) Konstt::";

cin>>Konstt;

cout<<"Enter T\_inf::";

cin>>T\_inf;

cout<<"Enter time instant::";

cin>>Time;

cout<<"Enter time step::";

cin>>D\_t;

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);

x++;

}

notepad(--x);

remove("grid0.dat");

system("cls");

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

display(x);

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

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

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

}

THE END

THANK YOU!