DEPARTMENT OF METALLURGICAL ENGINNERING, IIT(BHU)

MODELLING AND SIMULATION

Lab Manual

List of experiments

1. Introduction to Linux and C programming
2. Calculation of Gibbs Energy of mixing
3. Temperature distribution inside a heated plate (steady state)
4. Temperature distribution inside a long, thin rod using FDM
5. Temperature distribution inside a heated plate (unsteady state)
6. Temperature distribution for a long thin rod using finite element method
7. Temperature distribution inside a heated plate using finite element method on Ansys
8. Computation of Iso-G Curves
9. Computation of Phase diagram
10. Application of artificial neural network (ANN) in hot forging of steels
11. Modeling hardness of Nb-microalloyed steels using fuzzy logic

1. Introduction to Common Linux Commands and *vi* Editor

1.1.Statement of the Problem

To learn about common Linux commands and vi editor and to write a simple C program using vi editor, compile it and run its executable file.

1.2. Background

Common Linux Commands:

(i) The Manual

```
$man
```

This command brings up the online Unix manual. Use it on each of the commands below. For Example

```
$man pwd
```

You will see the manual entry for the pwd command.

(ii) Changing your password: Change your password from command line using *passwd*. This will prompt for the old password followed by the new password.

```
$ passwd
```

(iii) Accessing files in Folders (Directories) in terminal mode

```
$ pwd
```

"pwd" means present working directory. It shows what directory (folder) you are in. Let's suppose your home directory is "/home/particle" and directory "muondata" is an entry in your main home directory " /home/particle. Now, you have several data files (data1, data2 ... etc.) in a directory called "muondata". If you are in your home directory (where terminals start) and type pwd, you will see "/home/particle" and If you were in the "muondata" directory, pwd would give you "/home/particle/muondata" instead. As you can see, each slash (/) indicates another sub-directory.

```
$cd
```

Changes directories.
Examples of relative movement among directories:

```
$cd muondata
```

Moves down from your current directory into the “muondata” sub-directory

```
$cd ..
```

Moves up one directory (yes, include the two little dots). You can also move directly into directories

```
$ cd /home/particle/muondata
```

Moves from ANY directory into the muondata sub-directory of your home directory.

```
$cd ~
```

Takes you back to your home directory (/home/particle)

(iv) Making or Removing a Directory

```
$mkdir dirName
```

Creates a directory with name dirName.

```
$ rmdir dirName
```

Removes a directory dirName.

(v) Looking at or Finding your Files

```
$ ls
```

Lists files. If you add “–l” after “ls” it will give more details for each file. Such as, size, permissions, owners, dates etc.

```
$ ls -l
```

You'll see a huge list of files that you can't see with the 'ls' command alone and lots of details. If you see such a long list of files that they scroll off the terminal screen, one way to solve the problem is to use:

```
$ ls -l |more
```

Shows one screen of file names at a time.

```
$less data1
```

Dumps the contents of the data1 file to your screen with a pause at each line so you don't miss any contents as they scroll. You may move through the file using page up, page down, home and end keys. When done with less you use the q key to get back to the main terminal.

$whereis data1

Shows you the location of the data1 file.

(vi) Altering your Files

$ rm data1

Deletes the file data1 in the current directory.

$ rm -i muon*

Removes all of your muon data files (careful!! rm * will remove ALL your files) The "-i" makes the computer prompt before removing each file. If you really want to work without a net, omit the "-i".

$ cp data1 newdata/

will copy the file data1 to the directory newdata (assuming it has already been created)

$ mv data1 newdata/

moves the file data1 to the folder newdata and deletes the old one.

(vii) Creating your files

$ vi filename

This will create a file “filename” in the current directory using vi editor. If file, already exits, vi editor will open it for editing. Details of editing a file using vi editor are given in the next section.

(viii) Compiling your files

$ cc prog1.c

Command cc (or gcc) may be used to compile a file containing c program (notice .c extension in file name). Or one can use a detailed command as.

$ cc -o prog1.o prog1.c

This command generated the file prog1.o on successful compilation of code.

If you want to compile C++ program instead, then use

$ g++ prog1.cpp

Again notice file extension.

(vii) Running your executable file

Successful compilation of files leads to creation of executable file with ".out" or a file with "o" extension based on the command used for compilation, which can be run as follows

```
$ ./a.out
```

or

```
$ ./prog1.o
```

vi Editor Commands

To run vi and create a new file, simply run the command '*vi*' from any shell prompt:

```
$ vi
```

Alternatively, to load an existing text file into *vi*, run the command 'vi [filename]', from the shell prompt:

```
$ vi myfile.txt
```

Your file will be loaded into vi, and the cursor will be placed at the beginning of the first line. Note that an empty line is shown as a tilde (~).

Command Mode and Insert Mode

vi is different from many editors, in that it has two main modes of operation: command mode, and insert mode. This is the cause of much of the confusion when a new user is learning vi, but it is actually very simple to understand.

When you first load the editor, you will be placed into command mode. To switch into insert mode, simply press the 'i' key. Although nothing will change on the screen to indicate the new mode, anything that you type from now on will appear in the screen. This is what you are used to if you have ever used any other editor, or word processor. Try typing a few lines of text. When you press 'return' or 'enter' a new line will be created, and you may continue typing.

When you have finished typing, you may return to command mode. This is done by pressing your 'Esc' key. In command mode, key presses do not appear on the screen, but instead are used to indicate various commands to vi.

At first, you may often mistake command mode and insert mode. For example, you may think you are in insert mode, and start typing your text, when in fact you are in command mode, and

each key press you make will issue a command to vi. Be careful - you may accidentally modify or delete parts of your file.

If you are unsure which mode you are in, press 'Esc'. If you were in insert mode, you will be returned to command mode. If you were already in command mode, you will be left in command mode (possibly with a 'beep', to indicate that you were already in command mode).

An Exercise

Try this simple exercise to get used to the two modes:

Load vi (if you haven't already), by typing:
$ vi
(don't forget - the $ is the system's prompt, and may be different on your system. You only type the 'vi' part, shown in bold.)
The screen should show a blank file, with each blank line represented by a tilde (~), for example:

```
~
~
~
~
~
```

By default you are in command mode.
Switch to insert mode, by pressing the 'i' key. Then type some text, using ENTER or RETURN to start new lines. For example:

```
Hello.  This is my first session in the vi editor.
This is the second line of text.
~
~
~
```

When you have finished entering your sample text, press 'Esc' to return to command mode.

Now we will learn a useful command: to save the file. The command for this is ':w' (note the colon before the 'w'). After the 'w', put a space, and the name you want to store the file as. For example:

```
:w firstfile
```

Type it now. Notice how the text ':w' appears at the bottom on the screen. When you have finished the command, press ENTER or RETURN. You should see a confirmation that the file has been saved, which may include the number of lines in the file, and possibly the file size.

To finish this simple introduction to the editor, we will learn one final command: How to close the editor. The command for this is ':q' (again, with a colon before the 'q'). Don't forget you need to be in command mode. If you're not already in command mode, or you're not sure which mode you're in, press 'Esc' now. Then issue the command:

:q

To quit the editor discarding the modification carried, use the following command:
:q!

You should be returned to the Linux prompt. Saving and quitting operations can be combined in the command:
:wq

1.3.Methodology

(i) Open a file using vi editor with name *Prog0.c*
(ii) Change to insert mode
(iii) Write the code as given below.
(iv) Save and quit the editor
(v) Compile using cc command
(vi) Execute using /.a.out command

```
#include<stdio.h>
#include<math.h>

int main() {
   printf("Hello world!\n");
   return 0;
}
```

Prog0.c

2. Calculation of Gibbs Energy of mixing

2.1.Statement of the Problem

Write a program to compute Gibbs energy of mixing of phases for the given system at a given temperature using the Redlich-Kister (R-K) polynomial coefficients provided and write the output to a file.

2.2.Background

Gibbs energy of mixing consists of two terms: (i) ideal Gibbs energy of mixing (G_{id}^{ϕ}) and (ii) excess Gibbs energy (G_{xs}^{ϕ}).

$$G_{id}^{\phi} = RT\left[(1-x_B)\ln(1-x_B) + x_B \ln x_B\right] \tag{1}$$

The excess Gibbs energy for a binary system is modelled using R-K polynomial model as

$$G_{xs}^{\phi} = x_B(1-x_B)\sum_{i=0} L_i^{\phi}(1-2x_B)^i \tag{2}$$

where L_i are written as

$$L_i^{\phi} = a + b \cdot T \tag{3}$$

The Gibbs energy of mixing of a phase can be written as:

$$G_{mix}^{\phi} = G_{id}^{\phi} + G_{xs}^{\phi} \tag{3}$$

The R-K polynomial coefficients are given in the Table 1.

Table 1

Sc-Zr (T_A=1610K and T_B=1136K)[1]						
Phase	L_0		L_1		L_2	
	A	*B*	*A*	*B*	*A*	*B*
α	2550	6.7	0	0	0	0
β	1650	5.71	923	0	-1130	0

[1] Wang et al. *Thermodynamic assessment of Sc-M (M: r, Gd, Mo, W and Zr) systems* Journal of Phase Equilibria and Diffusion, Vol. 36, No-1, 2015.

2.3.Methodology

We start with a simple C program to calculate Gibbs energy of α and β phases of Sc-Zr system at a given temperature T=1500K at x_B=0.6. You have to pick R-K polynomial coefficients accordingly.

prog1.c

```
/*C Program to calculate Gibbs energy of alpha and beta phases for Sc-Zr system at xB=0.6 and T=1500K */
#include<stdio.h>
#include<math.h>

int main() {
        //define and initiate variables
        double R=8.3144;
        double Gid=0.0;
        double Gxs=0.0;
        double Gmix=0.0;
        double T = 0.0;
        double xB=0.6;
        //R-K polynomial coefficients for alpha phase
        double a_a0 = 2550.0;
        double a_b0 = 6.7;
        double a_a1 = 0.0;
        double a_b1 = 0.0;
        double a_a2 = 0.0;
        double a_b2 = 0.0;

        //Gibbs energy calculations
        T=1500;
        Gid= R*T*((1-xB)*log(1-xB)+xB*log(xB));
        Gxs = xB*(1-xB)*(a_a0+a_b0*T+ (a_a1+a_b1*T)*(1-2*xB) + (a_a2+a_b2*T)*pow((1-2*xB),2));
        Gmix = Gid + Gxs;

        //Print result
         printf("Gibbs energy of alpha phase at T=%lf and xB=%lf is %lf J/mol\n",T,xB,Gmix);
         return 0;
}

Output:
Gibbs energy of alpha phase at T=1500.000000 and xB=0.600000 is -5369.532306 J/mol
```

Data of the same type may be grouped using arrays.

```
/*C Program to calculate Gibbs energy of alpha and beta phases for Sc-Zr system at xB=0.6 and T=1500K */
#include<stdio.h>
#include<math.h>

int main() {
        //define and initiate variables
        double R=8.3144;
        double Gid=0.0;
        double Gxs=0.0;
        double Gmix=0.0;
        double T = 0.0;
        double xB=0.6;
        //R-K polynomial coefficients for alpha phase
        double a_set[6]={2550.0,6.7,0.0,0.0,0.0,0.0};

        // Gibbs energy calculations
        T=1500;
        Gid= R*T*((1-xB)*log(1-xB)+xB*log(xB));
        Gxs = xB*(1-xB)*( a_set [0]+ a_set [1]*T+ (a_set [2]+ a_set [3]*T)*(1-2*xB) + (a_set [4]+ a_set [5]*T)*pow((1-
2*xB),2));
        Gmix = Gid + Gxs;

         //Print result
         printf("Gibbs energy of alpha phase at T=%lf and xB=%lf is %lf J/mol\n",T,xB,Gmix);
         return 0;
}

Output:
Gibbs energy of alpha phase at T=1500.000000 and xB=0.600000 is -5369.532306 J/mol
```

prog2.c

Let’s calculate the Gibbs energy using the function.

```
/*C Program to calculate Gibbs energy of alpha and beta phases for Sc-Zr system at xB=0.6 and T=1500K */
#include<stdio.h>
#include<math.h>
double r_k(double coeff[6], double t, double xb);
int main() {
        //define and initiate variables
        double R=8.3144;
        double Gid=0.0;
        double Gxs=0.0;
        double Gmix=0.0;
        double T = 0.0;
        double xB=0.6;
        //R-K polynomial coefficients for alpha phase
        double a_set[6]={2550.0,6.7,0.0,0.0,0.0,0.0};

        // Gibbs energy calculations
        T=1500;
        Gid= R*T*((1-xB)*log(1-xB)+xB*log(xB));
        Gxs =r_k(a_set,T,xB);
        Gmix = Gid + Gxs;

        //Print result
        printf("Gibbs energy of alpha phase at T=%lf and xB=%lf is %lf J/mol\n",T,xB,Gmix);
        return 0;
}

double r_k(double coeff[6], double t, double xb)
{
        int i=0;
        double gxs=0.0;

        for(i=0;i<=5;i+=2)
          gxs=gxs+(coeff[i]+coeff[i+1]*t)*pow((1-2*xb),(i/2));

        gxs= xb*(1-xb)*gxs;
        return(gxs);
}

Output:
Gibbs energy of alpha phase at T=1500.000000 and xB=0.600000 is -5369.532306 J/mol
```

prog3.c

Let’s use user defined header files, read and write data to the files.

```
2550.0 6.7 0.0 0.0 0.0 0.0
```

inputData.txt

```
void get_data(char file_name[], double coeff[6], int n);
void write_data(char file_name[], double data[50][2],int n);

//reads n no of data points from the file file_name and stores in the array coeff
void get_data(char file_name[], double coeff[6], int n)
{
	int i=0;
	FILE *fp;
	fp=fopen(file_name,"r");			//Openining the file for reading purpose
	for(i=0;i<=5;i=i+1)
	{
		fscanf(fp,"%lf",&coeff[i]);		//Reading the data from the file and storing
	}
	fclose(fp);					//Closing the file opened
}

//writes n no of data points to the file file_name from the two dimensional array data
void write_data(char file_name[], double data[50][2], int n)
{
	int i=0;
	FILE *fp;
	fp=fopen(file_name,"w");		//Openining the file for writing purpose
	for(i=0;i<n;i=i+1)
	{
		fprintf(fp,"%lf\t%lf\n",data[i][0],data[i][1]);	//Writing the data to the file
	}
	fclose(fp);				//Closing the file opened
}
```

io.h

```
double g_mix(double coeff[6], double t, double xb);
double g_id(double t, double xb);
double rk_evaluate(double coeff[6], double t, double xb);

double g_mix(double coeff[6], double t, double xb)
{
        double gid,gxs;
        gid=g_id(t,xb);
        gxs=rk_evaluate(coeff,t,xb);
        return(gid+gxs);
}

double g_id(double t, double xb)
{
        return(8.3144*t*((1-xb)*log(1-xb)+xb*log(xb)));
}

//evaluate the r-k polynomila with the coefficeints coeff at the given temp. t and composition xb
double rk_evaluate(double coeff[6], double t, double xb)
{
        int i=0;
        double gxs=0.0;

        for(i=0;i<=5;i+=2)
        {
                gxs=gxs+(coeff[i]+coeff[i+1]*t)*pow((1-2*xb),(i/2));
        }
        gxs= xb*(1-xb)*gxs;
        return(gxs);
}
```

calGmix.h

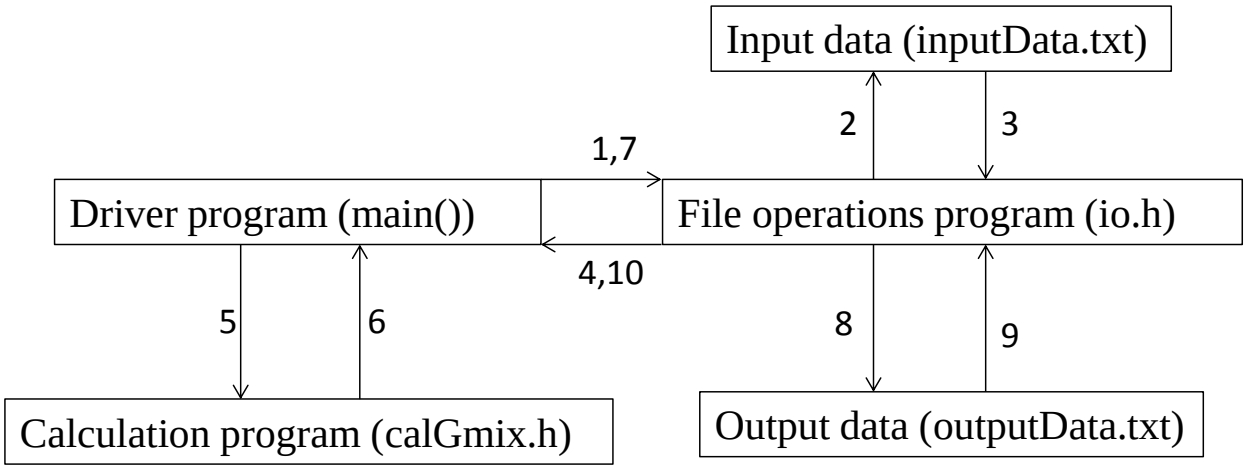
```c
/*C Program to calculate Gibbs energy of alpha and beta phases for Sc-Zr system at xB=0.6 and T=1500K */
#include<stdio.h>
#include<math.h>
#include "io.h"
#include "calGmix.h"                    //including user defined header file
#define R 8.3144                        //defining R as global variable


int main() {
        //define and initiate variables
        double Gmix=0.0;
        double T = 1500.0;
        double xB=0.6;
        char inputFileName[]="inputData.txt";           //file name to read input data
        char outputFileName[]="outputData.txt";         //file name to write output data
        double a_set[6];                                //to store the r-k polynomial data
        double output[50][2];                           //for storing output data

        //get data from the file
        get_data(inputFileName,a_set,6);                //ref to the header file for details
        //compute Gmix
        Gmix = g_mix(a_set,T,xB);
        //Write data to the file
        output[0][0]=xB;
        output[0][1]= Gmix;
        write_data(outputFileName, output,1);
        //Print result to the screen
        printf("Gibbs energy of alpha phase at T=%lf and xB=%lf is %lf J/mol\n",T,xB,Gmix);
        return 0;
}
```

prog4.c

Program Structure is being followed as follows:



Modify the above program to calculate G_{mix} for xB = 0 to 1 at the user defined interval and write the calculated data to the file. Sample output for T=1500K for ΔxB=0.02 is shown below.

```
0.020000   -975.744605
0.040000   -1610.692233
0.060000   -2120.008154
0.080000   -2549.340097
0.100000   -2920.304811
0.120000   -3245.581721
0.140000   -3533.502600
0.160000   -3789.946868
0.180000   -4019.271010
0.200000   -4224.818865
0.220000   -4409.225332
0.240000   -4574.608431
0.260000   -4722.696648
0.280000   -4854.916594
0.300000   -4972.455230
0.320000   -5076.305127
0.340000   -5167.298066
0.360000   -5246.130358
0.380000   -5313.382161
0.400000   -5369.532306
0.420000   -5414.969710
0.440000   -5450.002097
0.460000   -5474.862578
0.480000   -5489.714435
0.500000   -5494.654377
0.520000   -5489.714435
0.540000   -5474.862578
0.560000   -5450.002097
0.580000   -5414.969710
0.600000   -5369.532306
0.620000   -5313.382161
0.640000   -5246.130358
0.660000   -5167.298066
0.680000   -5076.305127
0.700000   -4972.455230
0.720000   -4854.916594
0.740000   -4722.696648
0.760000   -4574.608431
0.780000   -4409.225332
0.800000   -4224.818865
0.820000   -4019.271010
0.840000   -3789.946868
0.860000   -3533.502600
0.880000   -3245.581721
0.900000   -2920.304811
0.920000   -2549.340097
0.940000   -2120.008154
0.960000   -1610.692233
0.980000   -975.744605
```

outputData.txt

3. Temperature distribution inside a heated plate (steady state)

3.1.Statement of the Problem

Use Liebmann's method (Gauss-Seidel) to solve for the temperature inside a heated plate. Employ over relaxation with a value of 1.5 for the weighting factor and iterate to relative error ε_s = 1%. Write the temperature profile of the file to a file.

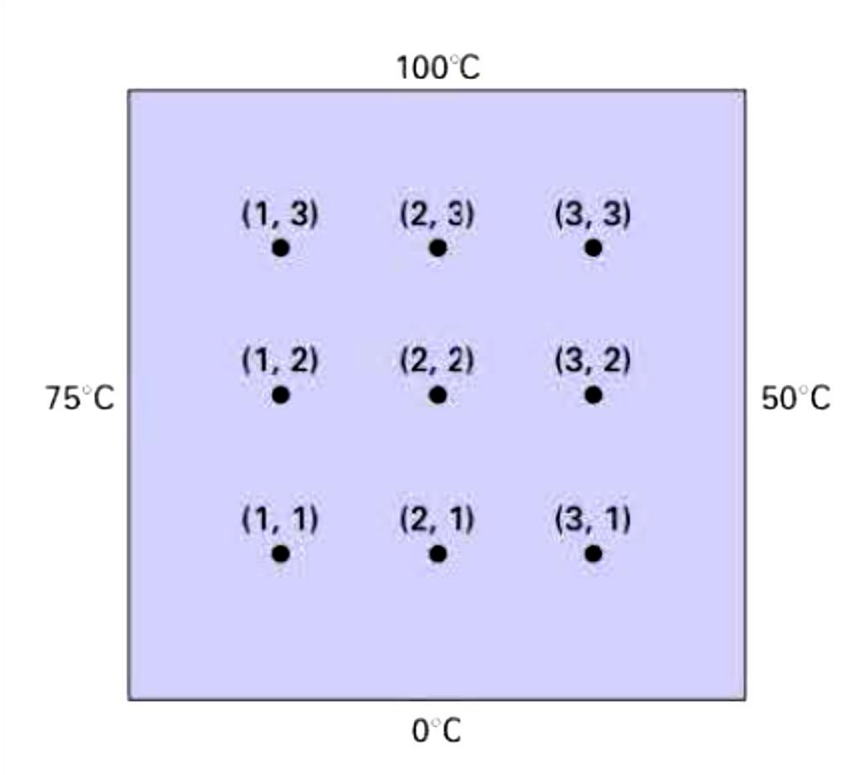


Figure 1 A heated plate where boundary temperatures are held at constant levels.

3.2.Background

Laplace equation:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0 \tag{1}$$

The Laplacian Difference Equation can be written as

$$\frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{\Delta x^2} + \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{\Delta y^2} = 0 \tag{2}$$

For the square grid as shown in Figure 1, $\Delta x = \Delta y$

$$T_{i+1,j} + T_{i-1,j} + T_{i,j+1} + T_{i,j-1} - 4T_{i,j} = 0 \tag{3}$$

This relationship, which holds for all interior points on the plate, is referred to as the *Laplacian difference equation*.

The Liebmann Method

In this technique, Equation (3) is expressed as

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

and solved iteratively for j = 1 to J and i = 1 to I (=J).

Successive Over Relaxation (SOR)

Overrelaxation is employed to accelerate the rate of convergence by applying the following formula after each iteration:

$$T_{i,j}^{n+1} \leftarrow \lambda T_{i,j}^{n+1} + (1-\lambda) T_{i,j}^{n} \tag{5}$$

where $T_{i,j}^{n+1}$ and $T_{i,j}^{n}$ are the values of $T_{i,j}$ from the present and the previous iteration, respectively, and λ . is a weighting factor that is set between 1 and 2.

The iterations are repeated until the absolute values of all the percent relative errors $(\varepsilon_a)_{i,j}$ fall below a pre specified stopping criterion ε_s . These percent relative errors are estimated by

$$\left|(\varepsilon_a)_{i,j}\right| = \left|\frac{T_{i,j}^{new} - T_{i,j}^{old}}{T_{i,j}^{new}}\right| 100\% \tag{6}$$

3.3.Methodology

1. Take user input for number of interior points, J^2 (= 9 in this case, J being the number of internal nodes in each direction assuming on a square grid), boundary conditions (4 boundary temperatures) and ε_s (percent relative error).
2. Set maximum number of iterations, $r_{\max}$ (say 50). Create a 2-dimensional array of size (J+2) × (J+2) for storing $T_{i,j}$ at interior points and boundary points. Create another 2-dimensional array of size (J) × (J) for storing $(\varepsilon_a)_{i,j}$.

3. Iteration (r = 0): Set initial values of all the interior points as zero; similarly set the value of points on the boundary as per boundary conditions. Also set all the $(\varepsilon_a)_{i,j}$ as zero.
4. Iteration ($r = r + 1$): for j = 1 to J and i = 1 to J, calculate value of $T_{i,j}$ using equation (4) at all the internal node and over relax it using (5). Also calculate percent relative errors $(\varepsilon_a)_{i,j}$. Store these values in the respective arrays. Always use the latest available values of $T_{i,j}$ while using equation (4). This completes first iteration.
5. If all $(\varepsilon_a)_{i,j}$ become lesser than ε_s or r reaches r_{max}, stop calculations.
6. Repeat steps number (4) to (5).
7. Display $T_{i,j}$ and also store in output text file.

3.4 Algorithm

A sample algorithm is given below:

```
//The sequence of data in inputfile should be (wintout commas and new lines):
//interior nodes along the x-axis,along the y-axis,temperature along the left edge, top edge,
right edge, bottom edge, interior nodes, permissible error, lambda

#include<stdio.h>
#include “io.h”
#include "laplace_fdm.h"
void main() {
        int i=0;
        int n[2];
        double T[20][20],t[5],other[2];
        char file_output[]="laplace_fdm_output.txt";
        char file_input[]="laplace_fdm_input.txt";

        get_data(file_input,n,t,other);         //read the data required for input
        initialize_temp(T,n,t);                 //initialize the array as per boundary conditions
        solve_laplace(T,n,other[0],other[1]);//Solve as per the formula provided
        write_data(file_output,T,n);            //write the final solution to the file

        return;
}
```

laplace_fdm.c

```
void solve_laplace(double T[20][20], int n[2], double error, double lambda);
void initialize_temp(double T[20][20], int n, double t[5]);


void solve_laplace(double T[20][20], int n[2], double error, double lambda)
{
	printf("solving the laplace equation\n");
	int i=0,j=0,counter=0,N=0;
	double tij,error_ij;
	while(counter<(n[0]*n[1]))
	{
		counter=0;		//number of nodes with error less than the permissible
		//code follows
	}
}

void initialize_temp(double T[20][20], int n, double t[5])
{
	//place the code here
}
```

laplace_fdm.h

4. Temperature distribution inside a long, thin rod using FDM

4.1.Statement of the Problem

Write a computer program to solve for the temperature distribution of a long, thin rod (as shown in Figure 1) the using the simple implicit finite-difference approximation.

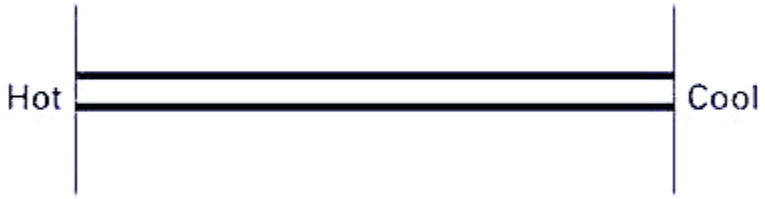


Figure 1 A thin rod, insulated at all points except at its ends

4.2.Solution

Heat-conduction equation is given by 1

$$k\frac{\partial^2 T}{\partial x^2} = \frac{\partial T}{\partial t} \tag{1}$$

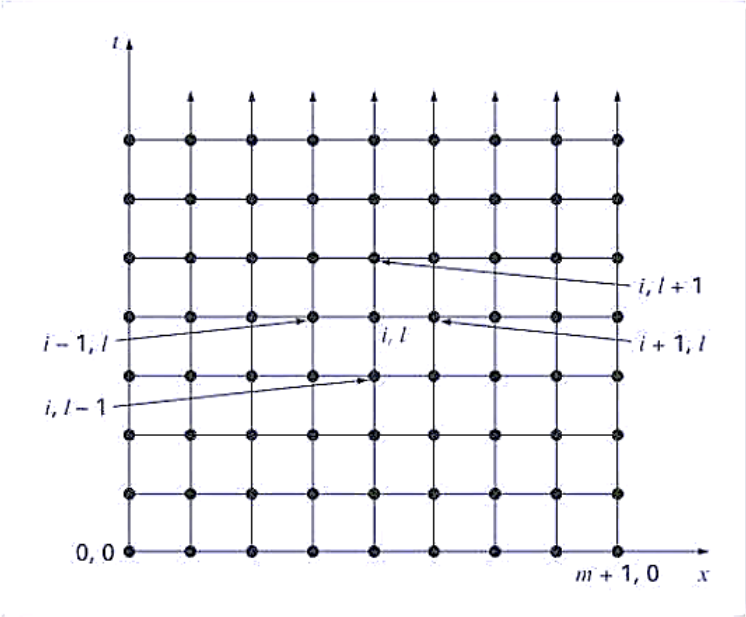


Figure 2 A grid used for the finite-difference solution of parabolic PDEs in two independent variables such as the heat-conduction equation.

The finite difference equations are given by

$$k\frac{T_{i+1}^{l+1}-2T_i^{l+1}+T_{i-1}^{l+1}}{\left(\Delta x\right)^2}=\frac{T_i^{l+1}-T_i^{l}}{\Delta t} \tag{2}$$

which can be expressed as

$$-\lambda T_{i-1}^{l+1}+(1+2\lambda)T_i^{l+1}-\lambda T_{i+1}^{l+1}=T_i^{l} \tag{3}$$

where $\lambda = k\Delta t/\left(\Delta x\right)^2$. This equation applies to all but the first and the last interior nodes

$$(1+2\lambda)T_1^{l+1}-\lambda T_2^{l+1}=T_1^{l}+\lambda T_0^{l+1} \tag{4}$$

Similarly, for the last interior node ($i = m$),

$$-\lambda T_{m-1}^{l+1}+\left(1+2\lambda\right)T_m^{l+1}=T_m^{l}+\lambda T_{m+1}^{l+1} \tag{5}$$

When Equations (3), (4), and (5) are written for all the interior nodes, the resulting set of m linear algebraic equations has *m* unknowns. The program then does the following:

1. Take user input of length of rod x (= 10cm), time (= 2 s), Δx (= 2 cm), Δt (= 0.1 s), k (= 0.835 cm^2s^{-1}), boundary conditions T_0 (= 100°C) and T_{m+1} (= 50°C) and initial condition T^0 (= 0°C). Calculate λ and number of interior points (m).
2. Generate stiffness matrix and force vectors using equation (3), (4) or (5).
3. Solve the system of equations using any method of solving system of linear equations for the first time step t (= Δt = 0.1 s).
4. Repeat step number (2) and (3) for desired number of time-steps (=20).

A variant of the code is given below

```
//the input file should contain: length of the rod, no of internal nodes, right end temperature, internal node temperature, left end temperature, time step, no of time steps, k
#include<stdio.h>
#include “io.h”
#include "parabolic.h"
#define max 20
void main()
{
	int i,XN,TN;
	double L,dx,t0,tn,ti,dt,k,lambda;
	double stiffness_matrix[max][max],T[max][max],data[8];
	char file_input[]="parabolic_input.txt";
	char file_output[]="parabolic_output.txt";

	getdata(file_input,data,8);
	L=data[0];
	XN=(int)data[1];
	t0=data[2];
	ti=data[3];
	tn=data[4];
	dt=data[5];
	TN=(int)data[6];
	k=data[7];

	dx=L/(XN+1);
	lambda=k*dt/(dx*dx);

	get_stiffness_matrix(stiffness_matrix,lambda,XN);
	initialize(T,t0,ti,tn,XN);
	solve_parabolic(stiffness_matrix,T,lambda,XN,TN);
	write_data(file_output,T,XN,dt,TN);
	}
```

parabolic.c

```
#include "gauss_elimination.h"
#define max 20
void get_stiffness_matrix(double a[max][max],double lambda, int n);
void initialize(double T[max][max],double t0,double ti,double tn,int XN);
void solve_parabolic(double a[max][max],double T[max][max],double lambda,int XN,int TN);

void get_stiffness_matrix(double a[max][max],double lambda, int n)
{
        int i=0,j=0;
        //code goes here
}

void initialize(double T[max][max],double t0,double ti,double tn,int XN)
{
        int i=0;
        //code goes here
}

void solve_parabolic(double a[max][max],double T[max][max],double lambda,int XN,int TN)
{
        int i=1,j=0;
        double b[max],t[max];
        double t0=T[0][0],tn=T[0][XN+1];
        for(i=1;i<=TN;i++)
        {
                //frame the rhs matrix, "b"
                ……
                //solving the equations
                gauss_elimination(a,b,t,XN);
                //updating ith row of T array
                ……
        }
}
```

parabolic.h

4.3.Flow Diagram

4.4.Computer Output

4.5.Discussion of Results

References

S.C. Chapra and R. P. Canale, Numerical Methods for Engineers, Seventh Edition, McGraw Hill, New Delhi, 2016.

5. Temperature distribution inside a heated plate (unsteady state)

5.1.Statement of the Problem

Write a computer program for solving temperature distribution inside a heated plate at various time intervals. At t = 0, assume that the temperature of the plate is zero and the boundary temperatures are instantaneously brought to the levels shown in Figure 1. Employ a time step of 10 s. the coefficient of thermal diffusivity for aluminum is k = 0.835 cm^2/s. Size of the plate is 40 × 40 - cm.

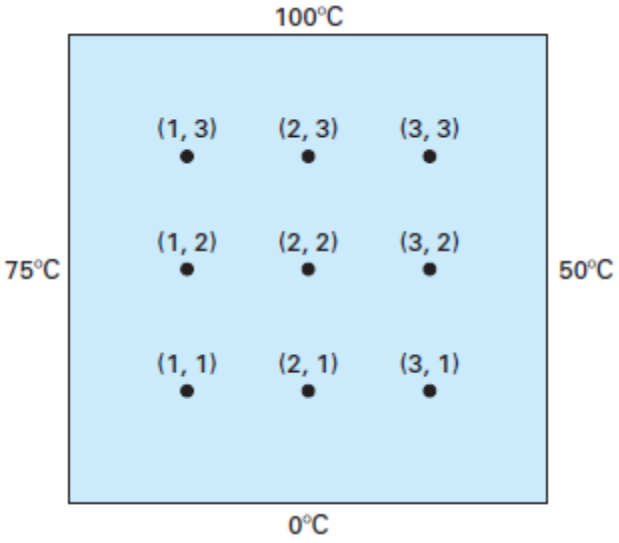


Figure 1 A heated plate where boundary temperatures are held at constant level.

5.2.Solution

Temperature distribution inside the heated plate under unsteady conditions is obtained by solving following equation:

$$\frac{\partial T}{\partial t} = k\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right) \quad (1)$$

At t = 0 T = 0 at all the points inside the plate.

At x = 0: T = 75 °C, x = 40 cm: T = 50 °C, y = 0: T = 0 °C and y = 40: T = 100 °C.

Equation (1) can be solved using the alternating-direction implicit, or ADI, scheme. ADI provides a means for solving parabolic equations in two spatial dimensions using tridiagonal matrices. To do this, each time increment is executed in two steps (Figure 2). For the first step, Eq. (1) is approximated by

$$\frac{T_{i,j}^{l+1/2} - T_{i,j}^{l}}{\Delta t/2} = k\left[\frac{T_{i+1,j}^{l} - 2T_{i,j}^{l} + T_{i-1,j}^{l}}{(\Delta x)^2} + \frac{T_{i+1,j}^{l+1/2} - 2T_{i,j}^{l+1/2} + T_{i-1,j}^{l+1/2}}{(\Delta x)^2}\right] \tag{2}$$

For the case of a square grid (Δy = Δx), this equation can be expressed as

$$-\lambda\, T_{i,j-1}^{l+1/2} + 2(1+\lambda)T_{i,j}^{l+1/2} - \lambda T_{i,j}^{l+1/2} = \lambda T_{i-1,j}^{l} + 2(1-\lambda)T_{i,j}^{l} + \lambda T_{i+1,j}^{l} \tag{3}$$

For the second step from $t_{l+1/2}$ to t_{l+1}, Eq. (1) is approximated by

$$\frac{T_{i,j}^{l+1} - T_{i,j}^{l+1/2}}{\Delta t/2} = k\left[\frac{T_{i+1,j}^{l+1} - 2T_{i,j}^{l+1} + T_{i-1,j}^{l+1}}{(\Delta x)^2} + \frac{T_{i,j+1}^{l+1/2} - 2T_{i,j}^{l+1/2} + T_{i,j-1}^{l+1/2}}{(\Delta x)^2}\right] \tag{4}$$

For a square grid, Eq. (4) can be written as

$$-\lambda\, T_{i-1,j}^{l+1} + 2(1+\lambda)T_{i,j}^{l+1} - \lambda T_{i+1,j}^{l+1} = \lambda T_{i,j-1}^{l+1/2} + 2(1-\lambda)T_{i,j}^{l+1/2} + \lambda T_{i,j+1}^{l+1/2} \tag{5}$$

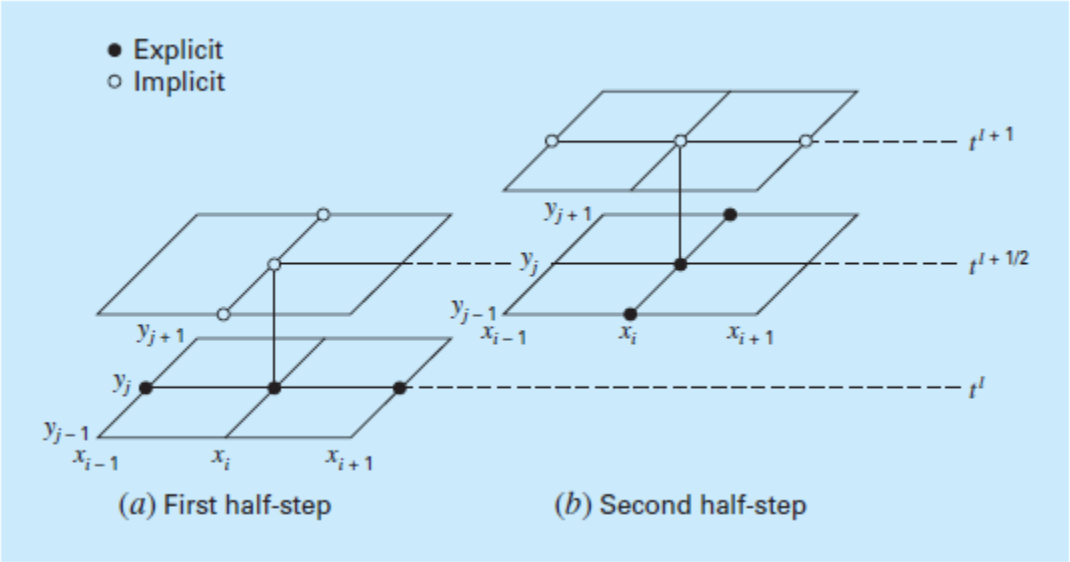


Figure 2 alternating direction implicit scheme for solving parabolic equations in two spatial dimensions.

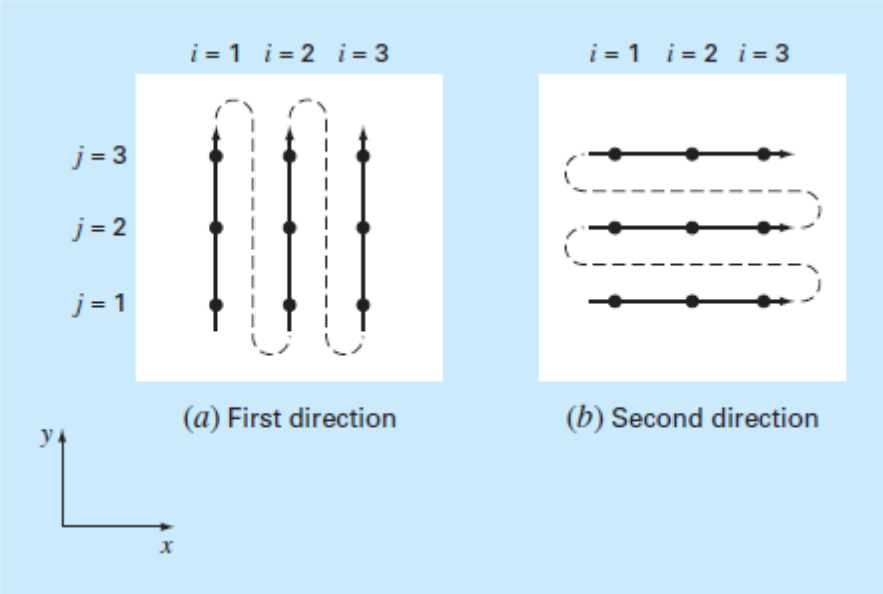


Figure 3 Alternating direction implicit scheme

The program then does the following:

(i) Take user input of length of dimensions of square plate (= 40 cm), time (= 100 s), Δx (= 10 cm), Δt (= 10 s), k (= 0.835 cm^2s^{-1}). Calculate λ and number of interior points (m).

(ii) At t = $\Delta t/2$, Generate stiffness matrix and force vectors using equation (3) for the nodes along y-axis (1, 1), (1, 2) and (1, 3) as shown in figure 3a. Solve the system of equations

(iii) Generate force vectors using equation (3) for the nodes on the second column along y-axis. Solve the system of equations.

(iv) Generate force vectors using equation (3) for the nodes on the second column along y-axis. Solve the system of equations

(v) At t = Δt, Generate stiffness matrix and force vectors using equation (5) for the nodes along x-axis (1, 1), (2, 1) and (3, 1) as shown in figure 3b. Solve the system of equations

(vi) Repeat step (v) for the remaining two rows. This will complete one time-step.

(vii) Repeat step number (ii) to (vi) for desired number of time-steps (=10).

One of the variants of the code is given below

```
#include<stdio.h>
#include "ADI.h"
#include “io.h”
#define maxNodes 20
#define maxdt 40

void main()
{
	double L1,dt,lambda,dx,k;
	int n1,TN;
	double T[maxdt][maxNodes][maxNodes],t[5];
	double stiffness_matrix[maxNodes][maxNodes], stiffness_Inv[maxNodes][maxNodes]={0};

	//get data
	//calculate lambda and others

	initialize_temp(T,n1,t);
	get_stiffness_matrix(stiffness_matrix,lambda,n1);
	get_mat_inverse(stiffness_matrix,stiffness_Inv,n1);
	solve_ADI(stiffness_Inv,T,n1,TN,lambda);
	//write data
}
```

ADI.c

```
//include header files required
//other definitions if any
//prototype declarations

void get_stiffness_matrix(double a[maxNodes][maxNodes],double lambda, int n)
{
        int i=0,j=0;
        ……
}

void initialize_temp(double T[maxdt][maxNodes][maxNodes],int n, double t[5])
{
        int i=0,j=0;
        …….
}

void initialize_T0(double T0[maxNodes][maxNodes],double T[maxdt][maxNodes][maxNodes],int n, int
xn)
{       //update T0 with the nth layer of T
        int i=0,j=0;
        ……..
}

void mat_multi_square_column(double a_inv[maxNodes][maxNodes],double b[maxNodes], int n)
{       //multiply a and b having length n
        …….
}

void update_T0_1(double T0[maxNodes][maxNodes],double b[maxNodes], int column,int n)
{       //update the column of T0 with b of length n in the position 'column'
        ……..
}

void update_T0_2(double T0[maxNodes][maxNodes],double b[maxNodes], int row,int n)
{       //update the row of T0 with b of length n in the position 'row'
        ……..
}

double updateT(double T[maxdt][maxNodes][maxNodes],double T0[maxNodes][maxNodes],int n,int xn)
{       //update nth layer of T with T0 matrix of length n
        ……
}
```

```
void solve_ADI(double a_inv[maxNodes][maxNodes], double T[maxdt][maxNodes][maxNodes], int xn, int tn,double lambda)
{
	int i,j,k,t;
	double T0[maxNodes][maxNodes]={0};
	double b[maxNodes]={0};
	for(t=1;t<tn;t++)
	{
		//initializing middle time layer temp profile
		initialize_T0(T0,T,t-1,xn);
		//frame the b matrix for first direction
		for(i=1;i<=xn;i++)
		{
			//frame b matrix
			//Lets solve the equations
			mat_multi_square_column(a_inv,b,xn);
			//update b values to the T0 matrix
			update_T0_1(T0,b,i,xn);
		}

		//frame the b matrix for second direction
		for(i=1;i<=xn;i++)
		{
			//frame b matrix
			//Lets solve the equations
			mat_multi_square_column(a_inv,b,xn);
			//update b values to the T0 matrix
			update_T0_2(T0,b,i,xn);
		}
		updateT(T,T0,t,xn);
	}
}
```

ADI.h

4.6.Flow Diagram

4.7.Computer Output

75.000000	100.000000	100.000000	100.000000	50.000000
75.000000	58.199878	49.531289	50.749669	50.000000
75.000000	38.906747	24.730294	30.068302	50.000000
75.000000	27.471223	13.399242	19.826793	50.000000
75.000000	0.000000	0.000000	0.000000	50.000000

Temperature profile of the plate at t = 100 s

4.8.Discussion of Results

References

S.C. Chapra and R. P. Canale, Numerical Methods for Engineers, Seventh Edition, McGraw Hill,