Project 2

Find a function $T(x,t)$ satisfying the following conditions:

$T(x,t)$ is defined in the domain $a < x < b, t_{init} \le t \le t_{end}$

$T_t = \alpha T_{xx}$ in the domain $a < x < b, t_{init} < t < t_{end}$

$T(x,t_{init}) = f(x)$ for $a < x < b$

$T(a,t) = g_1(x)$ for $t_{init} \le t \le t_{end}$

$T(b,t) = g_2(x)$ for $t_{init} \le t \le t_{end}$

$f(x) = 50x+3, g_1(t) = 20\cos(8t), g_2(t) = 20\sin(0.5t), a = -2, b = -0.5, t_{init} = 12000, t_{end} = 15000$

Material is Platinum and its $\alpha = 2.51 \cdot 10^{-5} \dfrac{m^2}{s}$.

Abstract

The purpose of this work is to become familiar with numerical techniques , finite difference schemes, available for solving parabolic partial difference equations (PDEs).

Introduction

Parabolic and hyperbolic equations

Problems involving time t as one independent variable lead usually to parabolic or hyperbolic equations. The simplest parabolic equation, $\dfrac{\partial U}{\partial t} = \alpha \dfrac{\partial^2 U}{\partial x^2}$, derives from the theory of heat conduction and its solution gives, fro example, the temperature U at a distance x from one end of a thermally insulated bar after t time of heat conduction. In such problem the temperatures at the ands of a bar of length l are often known for all time. In other words, the ***boundary conditions*** are known. It is also usual for the temperature distribution along the bar to be known at some particular instant. This instant is usually taken as zero time and the temperature distribution is called ***initial condition***.

Applications of finite-difference methods of solution to parabolic equations are integration of the differential equation over area S in $x-t$ plane. The integration is approximated by the solution of algebraic equations. The structure of the algebraic equations is different however in that it propagates the solution forward from one time row to the next in a step-by-step fashion.

Finite-difference approximations to derivatives

When a function U and its derivatives are single-valued, finite and continuous functions of x, the by Taylor's theorem

$$U(x+h) = U(x) + hU'(x) + \frac{1}{2}h^2U''(x) + \frac{1}{6}h^3U'''(x) + \ldots \qquad (1)$$

and

$$U(x-h) = U(x) - hU'(x) + \frac{1}{2}h^2U''(x) - \frac{1}{6}h^3U'''(x) + \ldots \qquad (3)$$

Addition of (1) and (2) gives

$$U(x+h) + U(x-h) = 2U(x) + h^2U''(x) + O(h^4) \qquad (3)$$

where $O(h^4)$ denotes terms containing fourth and higher power of h. Assuming these are negligible in comparison with lower powers of h it follows that,

$$U''(x) = \left(\frac{d^2U}{dx^2}\right)_{x=x} \cong \frac{1}{h^2}\left(U(x+h) - 2U(x) + U(x-h)\right) \qquad (4)$$

with an error on the right-hand side of order h^2.

Subtraction of (2) from (3) and neglect of terms of order h^3 leads to

$$U'(x)=\left(\frac{dU}{dx}\right)_{x=x}\cong\frac{1}{2h}\big(U(x+h)-U(x-h)\big) \qquad (5)$$

with an error of order h^2.

Equation (5) clearly approximates the slope of the tangent by the slope of the chord and is called a ***central-difference*** approximation. We can also approximate the slope by the ***forward-difference*** formula,

$$U'(x)=\left(\frac{dU}{dx}\right)_{x=x}\cong\frac{1}{h}\big(U(x+h)-U(x)\big) \qquad (6)$$

and by the ***backward-difference*** formula,

$$U'(x)=\left(\frac{dU}{dx}\right)_{x=x}\cong\frac{1}{h}\big(U(x)-U(x-h)\big) \qquad (7)$$

Notation of functions of several variables

Assume U is a function of the independent variables x and t. Subdivide the $x-t$ plane into sets of equal rectangles of sides $\delta x=h$, $\delta t=k$. Denote the value of U at the representative mesh point $P(x_i,t_j)$ by

$$U_P=U(x_i,t_j)=U_{i,j}.$$

The by eq. (4),

$$\left(\frac{\partial^2U}{\partial x^2}\right)_P=\left(\frac{\partial^2U}{\partial x^2}\right)_{i,j}\cong\frac{U_{i+1,j}-2U_{i,j}+U_{i-1,j}}{h^2} \qquad (8)$$

with a n error of order h^2.Similarly,

$$\left(\frac{\partial^2U}{\partial x^2}\right)_P=\left(\frac{\partial^2U}{\partial x^2}\right)_{i,j}\cong\frac{U_{i,j+1}-2U_{i,j}+U_{i,j-1}}{k^2} \qquad (9)$$

with the leading error pf order k^2.

With this notation the forward-difference approximation for U_t at P is

$$\frac{\partial U}{\partial t}\cong\frac{U_{i,j+1}-U_{i,j}}{k} \qquad (10)$$

with a leading error of $O(k)$.

Transformation to non-dimensional form

The computational stage of all numerical methods for solving problems of any complexity generally involves a great deal of arithmetic. It is usual therefore to arrange for one solution to suffice for a variety of different problems. This can be done by expressing all equations in terms of non-dimensional variables. Let

$$\frac{\partial U}{\partial T}=\alpha\frac{\partial^2U}{\partial X^2},\ \alpha \text{ constant,} \qquad (11)$$

the solution of which gives the temperature U at a distance X from one end of a thin uniform rod after time T. Let L represent the length of the rod and U_0 some particular temperature such as the maximum or minimum temperature at zero time. Put

$$x=\frac{X}{L} \text{ and } u=\frac{U}{U_0}.$$

Then

$$\frac{\partial U}{\partial X}=\frac{\partial U}{\partial x}\frac{dx}{dX}=\frac{\partial U}{\partial x}\frac{1}{L}$$

and

$$\frac{\partial^2 U}{\partial X^2} = \frac{\partial}{\partial x}\left(\frac{\partial U}{\partial x}\right) = \frac{\partial}{\partial x}\left(\frac{1}{L}\frac{\partial U}{\partial X}\right)\frac{dx}{dX} = \frac{1}{L^2}\frac{\partial^2 U}{\partial x^2},$$

so eq. (11) transforms to

$$\frac{L^2}{\alpha}\frac{\partial u}{\partial T} = \frac{\partial^2 u}{\partial x^2} \qquad (12)$$

Writing $t = \alpha \frac{T}{L^2}$ yields

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \qquad (13)$$

as the non-dimensional form of (11). It should be noted that the length of the rod is 1.

DuFort Frankel Method

By Taylor's Theorem

$$U(x+h,t) = U(x,t) + hU_x(x,t) + \frac{h^2}{2}U_{xx}(x,t) + \dots \qquad (14)$$

$$U(x-h,t) = U(x,t) - hU_x(x,t) + \frac{h^2}{2}U_{xx}(x,t) + \dots \qquad (15)$$

$$U(x,t+k) = U(x,t) + kU_t(x,t) + \dots \qquad (16)$$

$$U(x,t-k) = U(x,t) - kU_t(x,t) + \dots \qquad (17)$$

Addition of (14) and (15), (16) and (17) gives

$$2U(x,t) = U(x+h,t) + U(x-h,t) - h^2U_{xx}(x,t) + O(h^4) \qquad (18)$$

$$2U(x,t) = U(x,t+k) + U(x,t-k) + O(k^4) \qquad (19)$$

Subtraction of (18) and (19) forms

$$U_{xx}(x,t) = \frac{U(x+h,t) + U(x-h,t) - U(x,t+k) - U(x,t-k)}{h^2} \qquad (20)$$

From eq. (5) we have

$$U_t(x,t) = \frac{U(x,t+k) + U(x,t-k)}{2k} \qquad (21)$$

By eq. (20) and (21) finite-difference approximation to

$$\frac{\partial U}{\partial t} = \alpha\frac{\partial^2 U}{\partial x^2} \quad (22)$$

is

$$\frac{u_{i,j+1} - u_{i,j-1}}{2k} = \alpha\frac{u_{i+1,j} + u_{i-1,j} - u_{i,j+1} - u_{i,j+1}}{h^2} \quad (23)$$

If $\lambda = \alpha\frac{k}{h^2}$ then

$$u_{i,j+1} = \frac{(1-2\lambda)}{(1+2\lambda)}u_{i,j-1} + \frac{2\lambda}{(1+2\lambda)}(u_{i+1,j} + u_{i-1,j}) \qquad (24)$$

Local truncation error

Using Taylor expansions, it is easy to express ***local truncation error*** $T_{i,j}$ in terms of powers of h and k and partial derivatives of $U_{i,j}$.

Calculate the order of the local truncation error of the DuFort Frankel difference approximation to

$$\frac{\partial U}{\partial t} - \alpha \frac{\partial^2 U}{\partial x^2} = 0$$

If

$$F_{i,j}(u) = \frac{u_{i,j+1} - u_{i,j-1}}{2k} - \alpha \frac{u_{i+1,j} + u_{i-1,j} - u_{i,j+1} - u_{i,j+1}}{h^2}$$

then

$$T_{i,j} = F_{i,j}(U) = \frac{U_{i,j+1} - U_{i,j-1}}{2k} - \alpha \frac{U_{i+1,j} + U_{i-1,j} - U_{i,j+1} - U_{i,j+1}}{h^2}$$

By Taylor's expansion, substituting eq. (14),(15),(16) and (17) into the expression for $T_{i,j}$ the gives

$$T_{i,j} = \left(\frac{\partial U}{\partial t} - \alpha \frac{\partial^2 U}{\partial x^2} \right) + O(k^2) + O(h^2) + O\left(\left(\frac{k}{h} \right)^2 \right)$$

Bu U is the solution of the differential equation, therefore the principal part of the local truncation error is $O(k^2) + O(h^2) + O\left(\left(\frac{k}{h} \right)^2 \right)$.

$$T_{i,j} = O\left(k^2, h^2, \frac{k^2}{h^2} \right)$$

User requirements

Solution of the differential equation (11) requires initial condition – function $f(x)$ that is defined in the interval (a,b), boundary conditions – functions $g_1(t), g_2(t)$ defined in the interval $[t_{init}, t_{end}]$. Values of ***specific heat*** c, ***thermal conductivity*** κ and ***density*** ρ or $\alpha = \frac{\kappa}{c\rho}$ are also required. Output of the program will be matrix of values $U(x,t)$ or specific point of this matrix. Program will use ***double*** data type for calculation, thus according to Microsoft Specific of ***double*** data type, the output values must be bounded by $\pm 1.7 \cdot 10^{307}$ with at least 15 digits of precision.

Algorithm

According to the DuFort Frankel method it is unconditionally stable but we need to generate first row of the matrix using another method , explicit , for example, that is stable only for $\lambda \leq \frac{1}{2}$. Thus we can use next algorithm.

Step 1.
Define boundary and initial conditions $g_1(t), g_2(t), f(x)$.
Step 2.
Define step h and λ.
Step 3.
Define time step $k = \lambda \frac{h^2}{\alpha}$.
Step 4.
Generate first row of the matrix using explicit method $u_{i,j+1} = \lambda u_{i-1,j} + (1-2\lambda) u_{i,j} + \lambda u_{i+1,j}$.
Step 5.
Fill matrix using DuFort Frankel method $u_{i,j+1} = \frac{(1-2\lambda)}{(1+2\lambda)} u_{i,j-1} + \frac{2\lambda}{(1+2\lambda)} (u_{i+1,j} + u_{i-1,j})$.
Step 6.

If error is unacceptable, then $h \leftarrow \frac{h}{2}$ and return to **Step 2**.

User information for code in C++ implementing the algorithm

Full code listing can be found in the Appendix. Program uses functions that are implemented in standard C++ libraries. For better presicion and comparability included functions for streamed input and output. For algorithm implementation we don't need to save full matrix of the values into the memory. DuFort Frankel method has three-level scheme, thus we need to memorize only three rows of the matrix at each iteration. Values of the full matrix can be found in the output file.

Testing

$f(x)=10, g_1(x)=10, g_2(x)=10.$ Exact solution is $U(x,t)=10$.

Point (x,t)	Calculated Value $U(x,t)$	Real Value $U(x,t)$
(-1.5,12500)	10	10
(-1.0,13000)	10	10
(-0.6,14500)	10	10

$f(x)=x, g_1(t)=0, g_2(t)=0.4$. Exact solution $U(x,t)=x$.

Point (x,t)	Calculated Value $U(x,t)$	Real Value $U(x,t)$
(-1.5,12500)	-1.4923	-1.5
(-1.0,13000)	-1.01213	-1.0
(-0.6,14500)	-0.61126	-0.6

$f(x)=2\cos(0.1x)+5\sin(0.1x), g_1(t)=2e^{-0.01\alpha t}, g_2(t)=e^{-0.01\alpha t}(2\cos(0.04)+5\sin(0.04))$. Exact solution $U(x,t)=e^{-0.01\alpha t}(2\cos(0.1x)+5\sin(0.1x))$.

Point (x,t)	Calculated Value $U(x,t)$	Real Value $U(x,t)$
(-1.5,12500)	1.2145	1.22649
(-1.0,13000)	1.4923	1.48598
(-0.6,14500)	1.6924	1.6 9191

We can see that program gives results that are clear to the real values of $U(x,t)$.

Simulation of heat transfer

In Figure 1 we can see plotted program MathCad values of $U(x,t)$ generated by the program in the domain $12000 \le t \le 12500, -2 \le x \le -1.5$. Next table presents values of $U(x,t)$ in some points with different precision.

Point (x,t)	Calculated Value $U(x,t)$	Precision
(-1.5,12500)	-71.8554	0.01
(-1.5,12500)	-71.8453	0.001
(-1.0,13000)	-46.4019	0.01
(-1.0,13000)	-46.3945	0.001
(-0.6,14500)	-9.8696	0.01
(-0.6,14500)	-9.8634	0.001

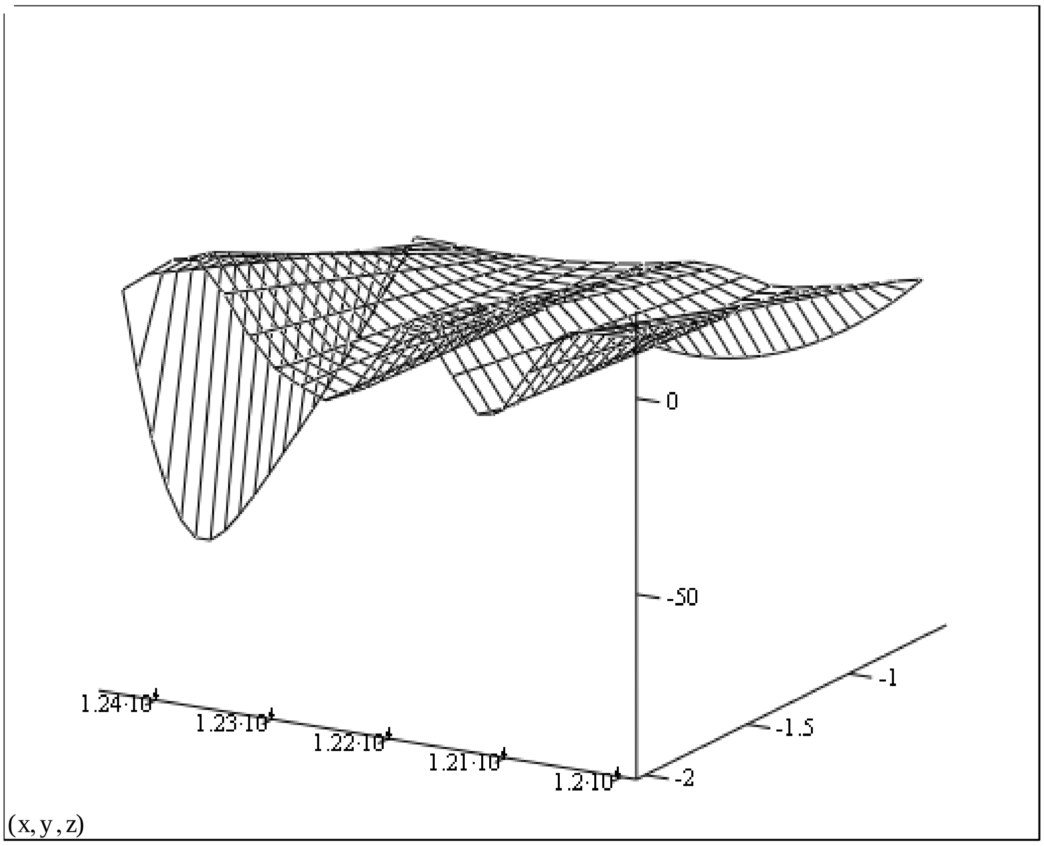

Figure 1

Appendix

```
#include <iostream.h>
#include <fstream.h>
#include <math.h>
#include <assert.h>
#include <iomanip.h>

const double a_value=-2.0; /* value of  $a$ */
const double b_value=-0.5; /* value of  $b$ */

const double ci=133.0; /* value of $c$ */
const double ro=21450.0; /* value of  $\rho$ */
const double kei=71.6; /* value of  $\kappa$ */
const double alpha=kei/(ci*ro); /*value of $\alpha$ */

const double t_init=12000.0; /* $t_{init}$ */
const double t_end=15000.0; /* $t_{end}$ */
double f(const double x_value) /* calculates  $f(x)$ */
{
        return 50*x_value+3;
}

double g1(const double t_value) /*calculates  $g_1(t)$ */
{
	return 20*cos(8*t_value);
}

double g2(const double t_value) /*calculates $g_2(t)$ */
{
	return 20*sin(0.5*t_value);
}
double *matrix[3]={NULL,NULL,NULL}; /*dynamic matrix with 3 rows*/
void init_matrix(const int size) /*function for dynamic initialization */
{
	for (int i=0; i<3; i++)
	{
		if (matrix[i]!=NULL)
			delete []matrix[i];
		matrix[i]=new double[size];
		assert(matrix[i]!=NULL);
	}
}

void main()
{
	int vector_size=8;
	int matrix_height=0;
	int matrix_size;
	int k=0;
	double dx;
	double dt;
	double lambda;
	double previous_value;
	double next_value;
	bool flag=false;
	double x;
	double t;
	double error=0.001;

	cout<<"Enter value of x "<<flush;
	cin>>x;
	cout<<"Enter value of t "<<flush;
	cin>>t;

	do
	{
		ofstream file("data.txt",ios::out); /*output file*/
		dx=(x-a_value)/vector_size;
		matrix_size=(b_value-a_value)/dx;
		init_matrix(matrix_size);
		dt=0.4*dx*dx/alpha;
		matrix_height=(t-t_init)/dt;
		lambda=alpha*dt/(dx*dx);

		for (int i=0; i<matrix_size; i++) /*initial conditions*/
			matrix[0][i]=f(a_value+i*dx);
```

```
        for (i=0; i<3; i++) /*boundary conditions*/
        {
                matrix[i][0]=g1(t_init+dt*i);
                matrix[i][matrix_size-1]=g2(t_init+dt*i);
        }
        for (i=1; i<matrix_size-1; i++) /*generation of the firs row with explicit method*/
                matrix[1][i]=lambda*matrix[0][i-1]+(1-
        2*lambda)*matrix[0][i]+lambda*matrix[0][i+1];

        for (i=0; i<2; i++) /*output to the file*/
        {
                for(int j=0; j<matrix_size; j++)
                        file<<setiosflags(ios::left)<<setw(15)<<matrix[i][j]<<" "<<flush;
                                        file<<endl;
        }

        for (int j=1; (j+1)<matrix_height; j++)
        {
                for (i=1; i<matrix_size-1; i++) /*calculation with DuFort Frankel Method*/
                        matrix[2][i]=(1-
                        2*lambda)/(1+2*lambda)*matrix[0][i]+2*lambda/(1+2*lambda)
                        *(matrix[1][i+1]+matrix[1][i-1]);
                for (i=0; i<matrix_size; i++)
                { /*output to the file*/
                        matrix[0][i]=matrix[1][i];
                        matrix[1][i]=matrix[2][i];
                        file<<setiosflags(ios::left)<<setw(15)<<matrix[2][i]<<" "<<flush;
                }
                matrix[2][0]=g1(t_init+dt*j);
                matrix[2][matrix_size-1]=g2(t_init+dt*j);
                file<<endl;
        }
        if (!flag)
        {
                next_value=matrix[1][int((x-a_value)/(b_value-a_value)*matrix_size)];
                previous_value=-next_value;
                flag=true;
        }
        else
        {
                previous_value=next_value;
                next_value=matrix[1][int((x-a_value)/(b_value-a_value)*matrix_size)];
        }
        cout<<"X= "<<x<<" T= "<<t<<" U(X,T)= "<<next_value<<" size=
        "<<vector_size<<endl;
        vector_size*=2;
        file.close();
    }
    while (fabs(next_value-previous_value)>error); /*stopping condition*/
}
```

Refereces

1. G.D. Smith “Numerical solution of Partial Differential Equations”,1985.
2. Stanley J.Farlow “Partial Differential Equations for Scientists and Engineers”,1993.