Московский авиационный институт (национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №8 по курсу «Численные методы»

Студент: Я.А Борисов
Преподаватель: Д. Е. Пивоваров
Группа: М8О-408Б-20
Дата:
Оценка:
Подпись:

Москва, 2023

Лабораторная работа №8

Метод конечных разностей решения многомерных задач математической физики

Задача

Используя схемы переменных направлений и дробных шагов, решить двумерную начально-краевую задачу для дифференциального уравнения параболического типа. В различные моменты времени вычислить погрешность численного решения путем сравнения результатов с приведенным в задании аналитическим решением. Исследовать зависимость погрешности от сеточных параметров τ, h_x, h_y.

Описание метода

Рассматриваются два метода решения двумерной задачи параболического типа: метод переменных направлений и метод дробных шагов.

Общая поставка такой задачи выглядит следующим образом:

$$\frac{\partial u}{\partial t} = a\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + f(x,y,t), \quad x \in (0,\ell_1), \quad y \in (0,\ell_2), \quad t > 0 \; ;$$

$$u(x,0,t) = \varphi_1(x,t)\,,\; x \in [0,\ell_1],\;\; y = 0\,,\, t > 0 \; ;$$

$$u(x,\ell_2,t) = \varphi_2(x,t)\,,\; x \in [0,\ell_1],\;\; y = \ell_2\,,\, t > 0 \; ;$$

$$u(0,y,t) = \varphi_3(y,t)\,,\; x = 0\,, y \in [0,\ell_2], t > 0;$$

$$u(\ell_1,y,t) = \varphi_4(y,t),\, x = \ell_1, y \in [0,\ell_2], \;\; t > 0\,;$$

$$u(x,y,0) = \psi\,(x,y)\,,\; x \in [0,\ell_1]\,,\; y \in [0,\ell_2]\,,\;\; t = 0.$$

Вводится пространственно-временная сетка с шагами h1, h2, τ соответственно по переменным x, y, t:

$$\omega^{\tau}_{h_1 h_2} = \left\{x_i = ih_1, i = \overline{0,I};\; x_j = jh_2, j = \overline{0,J}\;;\; t^k = k\tau\;,\; k = 0,1,2,\ldots\right\}$$

Метод переменных направлений

Шаг по времени τ разбивается на два. На каждом временном полуслое первый из пространственных дифференциальных операторов аппроксимируется неявно, а второй -явно. На следующим дробном шаге соответственно первый – явно, второй – неявно.

$$\frac{u_{ij}^{k+1/2}-u_{ij}^{k}}{\tau/2}=\frac{a}{h_1^2}\left(u_{i+1j}^{k+1/2}-2u_{ij}^{k+1/2}+u_{i-1j}^{k+1/2}\right)+\frac{a}{h_2^2}\left(u_{ij+1}^{k}-2u_{ij}^{k}+u_{ij-1}^{k}\right)+f_{ij}^{k+1/2},$$

$$\frac{u_{ij}^{k+1}-u_{ij}^{k+1/2}}{\tau/2}=\frac{a}{h_1^2}\left(u_{i+1j}^{k+1/2}-2u_{ij}^{k+1/2}+u_{i-1j}^{k+1/2}\right)+\frac{a}{h_2^2}\left(u_{ij+1}^{k+1}-2u_{ij}^{k+1}+u_{ij-1}^{k+1}\right)+f_{ij}^{k+1/2}$$

Т.е. здесь оператор $a\frac{\partial^2}{\partial x^2}$ на первом временном полуслое аппроксимируется неявно, $a\frac{\partial^2}{\partial y^2}$ – явно.

На втором временном полуслое наоборот.

С помощью скалярных прогонок в количестве J – 1 в направлении переменной x получаем значение $u_{ij}^{k+1/2}$ на первом временном полуслое.

Уже на втором шаге с помощью скалярных прогонок в количестве I – 1 в направлении переменной y получаем значение u_{ij}^{k+1} на следующем временном слое k+1.

К достоинствам метода переменных направлений можно отнести высокую точность, т.к. метод имеет второй порядок точности по времени.

Метод дробных шагов

В отличие от метода переменных направлений в методе дробных шагов используются только неявная схема аппроксимации.

$$\frac{u_{ij}^{k+1/2}-u_{ij}^{k}}{\tau}=\frac{a}{h_1^2}\left(u_{i+1j}^{k+1/2}-2u_{ij}^{k+1/2}+u_{i-1j}^{k+1/2}\right)+\frac{f_{ij}^{k}}{2},$$

$$\frac{u_{ij}^{k+1}-u_{ij}^{k+1/2}}{\tau}=\frac{a}{h_2^2}\left(u_{i\,j+1}^{k+1}-2u_{ij}^{k+1}+u_{i\,j-1}^{k+1}\right)+\frac{f_{ij}^{k+1}}{2}.$$

С помощью скалярных прогонок в количестве J – 1 в направлении переменной x получаем значение $u_{ij}^{k+1/2}$ на первом временном полуслое.

Уже на втором шаге с помощью скалярных прогонок в количестве I – 1 в направлении переменной y получаем значение u_{ij}^{k+1} на следующем временном слое k+1.

Схема метода дробных шагов имеет первый порядок точности по времени и второй порядок точности по пространству.

Вариант

4.

$$\frac{\partial u}{\partial t} = a\frac{\partial^2 u}{\partial x^2} + a\frac{\partial^2 u}{\partial y^2},\ a > 0,$$

$$u(0, y, t) = \cosh(y)\exp(-3at),$$

$$u(\frac{\pi}{4}, y, t) = 0,$$

$$u(x,0,t) = \cos(2x)\exp(-3at),$$

$$u_y(x, \ln 2, t) = \frac{3}{4}\cos(2x)\exp(-3at),$$

$$u(x, y, 0) = \cos(2x)\cosh(y).$$

Аналитическое решение: $U(x, y, t) = \cos(2x)\cosh(y)\exp(-3at)$.

Результаты работы программы

```
Метод переменных направлений
8.926431196019657E-4

Метод дробных шагов
0.025062081991189977
```

```
Метод переменных направлений
0.0011918315059199491

Метод дробных шагов
0.03380191590137083
```

Приложение. Листинг программы.

```
import java.util.ArrayList;
import java.util.Collections;

public class Lab8 {
    public static void main(String[] args) {
        double a = 1;
        int I = 10;
        double hx = Math.PI / (4 * I);
        int J = 20;
        double hy = Math.log(2) / J;
        int T = 20;
        double ht = 1. / T;

        double[][][] u = MPN(a, I, J, T, hx, hy, ht);
        int s = 10;
        int r = 12;
```

```java
    double max = 0;
    double delta = 0;
    double temp;
    for(int i = 0; i < I + 1; i++){
        temp = Math.abs(u[i][r][s] - U(i * hx, r * hy, s * ht, a));
        delta += temp * temp;
        if(temp > max){
            max = temp;
        }
    }
    System.out.println("Метод переменных направлений");
    //System.out.println(Math.sqrt(delta) + " " + max);
    System.out.println(max);

    u = MDS(a, I, J, T, hx, hy, ht);
    max = 0;
    delta = 0;
    for(int i = 0; i < I + 1; i++){
        temp = Math.abs(u[i][r][s] - U(i * hx, r * hy, s * ht, a));
        delta += temp * temp;
        if(temp > max){
            max = temp;
        }
    }
    System.out.println("\nМетод дробных шагов");
    //System.out.println(Math.sqrt(delta) + " " + max);
    System.out.println(max);
}

private static double[][][] MPN(double a, int I, int J, int T, double hx, double hy, double ht){
    double [][][] u = new double[I + 1][J + 1][T + 1];
    double [][] u_ = new double[I + 1][J + 1];
    double[] a_arr;
    double[] b_arr;
    double[] c_arr;
    double[] d_arr;

    //Граничные и начальные условия
    for (int i = 0; i < I + 1; i++) {
        for (int j = 0; j < J + 1; j++) {
            u[i][j][0] = Math.cos(2 * i * hx) * Math.cosh(j * hy);
        }
    }

    for (int k = 0; k < T + 1; k++) {
        for (int j = 0; j < J + 1; j++) {
            u[0][j][k] = Math.cosh(j * hy) * Math.exp(-3 * a * k * ht);
        }
    }

    for (int k = 0; k < T + 1; k++) {
        for (int j = 0; j < J + 1; j++) {
            u[I][j][k] = 0;
        }
    }

    for (int k = 0; k < T + 1; k++) {
        for (int i = 0; i < I + 1; i++) {
            u[i][0][k] = Math.cos(2 * i * hx) * Math.exp(-3 * a * k * ht);
        }
    }

    double sx = a * ht / (2 * hx * hx);
    double sy = a * ht / (2 * hy * hy);
    int m = 0;

    for(int k = 0; k < T; k++){
        a_arr = new double[I + 1];
        b_arr = new double[I + 1];
        c_arr = new double[I + 1];
        d_arr = new double[I + 1];
        u_ = new double[I + 1][J + 1];
```

```
    for(int j = 1; j < J; j++){
        m = 0;
        for(int i = 0; i < I; i++, m++){
            if(i == 0){
                a_arr[m] = 0;
                b_arr[m] = 1;
                c_arr[m] = 0;
                d_arr[m] = Math.cosh(j * hy) * Math.exp(-3 * a * (k + 0.5) * ht);
            }
            else{
                a_arr[m] = -sx;
                b_arr[m] = 1 + 2 * sx;
                c_arr[m] = -sx;
                d_arr[m] = u[i][j][k] + sy * (u[i][j + 1][k] - 2 * u[i][j][k] + u[i][j - 1][k]);
            }
        }

        a_arr[m] = 0;
        b_arr[m] = 1;
        c_arr[m] = 0;
        d_arr[m] = 0;

        ArrayList<Double> result = Progonka(a_arr, b_arr, c_arr, d_arr);
        for(int n = 0; n < u_.length; n++){
            u_[n][j] = result.get(n);
        }

        if(j == J - 1) {
            for (int i = 0; i < u_.length; i++) {
                u_[i][0] = Math.cos(2 * i * hx) * Math.exp(-3 * a * (k + 0.5) * ht);
            }

            for (int i = 0; i < u_.length; i++) {
                u_[i][J] = u_[i][j] + 0.75 * Math.cos(2 * i * hx) * Math.exp(-3 * a * (k + 0.5) * ht);
            }
        }
    }

    a_arr = new double[J + 1];
    b_arr = new double[J + 1];
    c_arr = new double[J + 1];
    d_arr = new double[J + 1];

    for(int i = 1; i < I; i++){
        m = 0;
        for(int j = 0; j < J; j++, m++){
            if(j == 0){
                a_arr[m] = 0;
                b_arr[m] = 1;
                c_arr[m] = 0;
                d_arr[m] = Math.cos(2 * i * hx) * Math.exp(-3 * a * (k + 1) * ht);
            }
            else{
                a_arr[m] = -sy;
                b_arr[m] = 1 + 2 * sy;
                c_arr[m] = -sy;
                d_arr[m] = u_[i][j] + sx * (u_[i + 1][j] - 2 * u_[i][j] + u_[i - 1][j]);
            }
        }

        a_arr[m] = -1 / hy;
        b_arr[m] = 1 / hy;
        c_arr[m] = 0;
        d_arr[m] = 0.75 * Math.cos(2 * i * hx) * Math.exp(-3 * a * (k + 1) * ht);

        ArrayList<Double> result = Progonka(a_arr, b_arr, c_arr, d_arr);
        for(int j = 0; j < J + 1; j++){
            u[i][j][k + 1] = result.get(j);
        }
    }
}
return u;
```

```java
}

private static double[][][] MDS(double a, int I, int J, int T, double hx, double hy, double ht){
    double [][][] u = new double[I + 1][J + 1][T + 1];
    double [][] u_ = new double[I + 1][J + 1];
    double[] a_arr;
    double[] b_arr;
    double[] c_arr;
    double[] d_arr;

    //Граничные и начальные условия
    for (int i = 0; i < I + 1; i++) {
        for (int j = 0; j < J + 1; j++) {
            u[i][j][0] = Math.cos(2 * i * hx) * Math.cosh(j * hy);
        }
    }

    for (int k = 0; k < T + 1; k++) {
        for (int j = 0; j < J + 1; j++) {
            u[0][j][k] = Math.cosh(j * hy) * Math.exp(-3 * a * k * ht);
        }
    }

    for (int k = 0; k < T + 1; k++) {
        for (int j = 0; j < J + 1; j++) {
            u[I][j][k] = 0;
        }
    }

    for (int k = 0; k < T + 1; k++) {
        for (int i = 0; i < I + 1; i++) {
            u[i][0][k] = Math.cos(2 * i * hx) * Math.exp(-3 * a * k * ht);
        }
    }

    double sx = a * ht / (hx * hx);
    double sy = a * ht / (hy * hy);
    int m = 0;

    for(int k = 0; k < T; k++){
        a_arr = new double[I + 1];
        b_arr = new double[I + 1];
        c_arr = new double[I + 1];
        d_arr = new double[I + 1];
        u_ = new double[I + 1][J + 1];
        for(int j = 1; j < J; j++){
            m = 0;
            for(int i = 0; i < I; i++, m++){
                if(i == 0){
                    a_arr[m] = 0;
                    b_arr[m] = 1;
                    c_arr[m] = 0;
                    d_arr[m] = Math.cosh(j * hy) * Math.exp(-3 * a * (k + 0.5) * ht);
                }
                else{
                    a_arr[m] = -sx;
                    b_arr[m] = 1 + 2 * sx;
                    c_arr[m] = -sx;
                    d_arr[m] = u[i][j][k];
                }
            }

            a_arr[m] = 0;
            b_arr[m] = 1;
            c_arr[m] = 0;
            d_arr[m] = 0;

            ArrayList<Double> result = Progonka(a_arr, b_arr, c_arr, d_arr);
            for(int n = 0; n < u_.length; n++){
                u_[n][j] = result.get(n);
            }
```

```
            if(j == J - 1) {
                for (int i = 0; i < u_.length; i++) {
                    u_[i][0] = Math.cos(2 * i * hx) * Math.exp(-3 * a * (k + 0.5) * ht);
                }

                for (int i = 0; i < u_.length; i++) {
                    u_[i][J] = u_[i][j] + 0.75 * Math.cos(2 * i * hx) * Math.exp(-3 * a * (k + 0.5) * ht);
                }
            }
        }

        a_arr = new double[J + 1];
        b_arr = new double[J + 1];
        c_arr = new double[J + 1];
        d_arr = new double[J + 1];

        for(int i = 1; i < I; i++){
            m = 0;
            for(int j = 0; j < J; j++, m++){
                if(j == 0){
                    a_arr[m] = 0;
                    b_arr[m] = 1;
                    c_arr[m] = 0;
                    d_arr[m] = Math.cos(2 * i * hx) * Math.exp(-3 * a * (k + 1) * ht);
                }
                else{
                    a_arr[m] = -sy;
                    b_arr[m] = 1 + 2 * sy;
                    c_arr[m] = -sy;
                    d_arr[m] = u_[i][j];
                }
            }

            a_arr[m] = -1 / hy;
            b_arr[m] = 1 / hy;
            c_arr[m] = 0;
            d_arr[m] = 0.75 * Math.cos(2 * i * hx) * Math.exp(-3 * a * (k + 1) * ht);

            ArrayList<Double> result = Progonka(a_arr, b_arr, c_arr, d_arr);
            for(int j = 0; j < J + 1; j++){
                u[i][j][k + 1] = result.get(j);
            }
        }
    }
    return u;
}

private static double U(double x, double y, double t, double a){
    return Math.cos(2 * x) * Math.cosh(y) * Math.exp(-3 * a * t);
}

static ArrayList<Double> Progonka(double[] a, double[] b, double[] c, double[] d)
{
    ArrayList<Double> roots = new ArrayList<>();
    ArrayList<Double> P = new ArrayList<>();
    ArrayList<Double> Q = new ArrayList<>();

    P.add(-c[0] / b[0]);
    Q.add(d[0] / b[0]);

    //Прямой ход
    for(int i = 1; i < a.length; i++)
    {
        P.add(-c[i] / (b[i] + a[i] * P.get(i - 1)));
        Q.add((d[i] - a[i] * Q.get(i - 1)) / (b[i] + a[i] * P.get(i - 1)));
    }

    Collections.reverse(P);
    Collections.reverse(Q);

    //Обратный ход
    roots.add(Q.get(0));
```

```
        for (int i = 1; i < a.length; i++)
        {
            roots.add(P.get(i) * roots.get(i - 1) + Q.get(i));
        }

        Collections.reverse(roots);
        return roots;
    }
}
```

Выводы

В данной работе используются схемы переменных направлений и дробных шагов для решения двумерной начально-краевую задачу для дифференциального уравнения параболического типа. В различные моменты времени вычисляется погрешность численного решения путем сравнения результатов с приведенным в задании аналитическим решением.

Погрешность метода переменных направлений (в среднем порядка 10^(-3)-10^(-4)) меньше по сравнению с погрешностью метода дробных шагов (в среднем порядка 10^(-2)-10^(-3)) .