Московский авиационный институт (национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №7 по курсу «Численные методы»

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

Москва, 2023

Лабораторная работа №7

Численное решение дифференциальных уравнений с частными производными

Задача

Решить краевую задачу для дифференциального уравнения эллиптического типа. Аппроксимацию уравнения произвести с использованием центрально-разностной схемы. Для решения дискретного аналога применить следующие методы: метод простых итераций (метод Либмана), метод Зейделя, метод простых итераций с верхней релаксацией. Вычислить погрешность численного решения путем сравнения результатов с приведенным в задании аналитическим решением $U(x, y)$.

Описание метода

Рассмотрим уравнение Пуассона для третьей краевой задачи в прямоугольнике:

$$\begin{cases} \dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2} = f(x,y), \quad x \in (0,l_1),\ y \in (0,l_2); \\ \alpha_1 \dfrac{\partial u}{\partial x}(0,y) + \beta_1 u(0,y) = \varphi_1(y), \\ \alpha_2 \dfrac{\partial u}{\partial x}(l_1,y) + \beta_2 u(l_1,y) = \varphi_2(y), \\ \alpha_3 \dfrac{\partial u}{\partial x}(x,0) + \beta_3 u(x,0) = \varphi_3(x), \\ \alpha_4 \dfrac{\partial u}{\partial x}(x,l_2) + \beta_4 u(x,l_2) = \varphi_4(x) \end{cases}$$

Частным случаем уравнение Пуассона является уравнение Лапласа (при f(x, y) = 0).

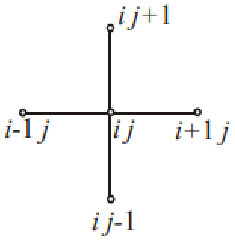
Для решения такой задачи применяют метод конечных разностей.

Аппроксимируя вторые частные производные, получим следующее выражение для внутренних узлов:

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h_1^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h_2^2} + O\left(h_1^2 + h_2^2\right) = f(x_i, y_j),$$
$$i = \overline{1, N_1 - 1}, \quad j = \overline{1, N_2 - 1}$$

СЛАУ, которая получается при решении данного уравнения, имеет пяти-диагональный вид (каждое уравнение содержит пять неизвестных и при соответствующей нумерации переменных матрица имеет ленточную структуру). Решать ее можно различными методами линейной алгебры, например, итерационными методами, методом матричной прогонки и т.п.

Шаблон данной схемы имеет следующий вид:



Рассмотрим метод простых итераций для решения данной СЛАУ. Для простоты положим h1 = h2 = h, тогда получим (k – номер итерации):

$$u_{i,j}^{(k+1)} = \frac{1}{4}[u_{i+1,j}^{(k)} + u_{i-1,j}^{(k)} + u_{i,j-1}^{(k)} + u_{i,j+1}^{(k)} - h^2 \cdot f_{i,j}], \quad f_{i,j} = f(x_i, y_j),$$
$$i = \overline{1, N_1 - 1}, \quad j = \overline{1, N_2 - 1}.$$

Перед решение методом простых итераций необходимо задать начальное приближение.

Условие выхода:

$$\left\| u^{(k+1)} - u^{(k)} \right\| \le \varepsilon,$$

где ε – наперёд заданная точность.

Аппроксимация граничных условий

Граничные условия аппроксимируются с первым порядком:

$$\alpha_1 \frac{u_{1j} - u_{0j}}{h_1} + \beta_1 u_{0j} = \varphi_1(y_j), \; j = \overline{1, N_2 - 1},$$

$$\alpha_2 \frac{u_{N_1 j} - u_{N_1 j}}{h_1} + \beta_2 u_{N_1 j} = \varphi_2(y_j), \;\; j = \overline{1, N_2 - 1},$$

$$\alpha_3 \frac{u_{i1} - u_{i0}}{h_2} + \beta_3 u_{i0} = \varphi_3(x_i), \; i = \overline{1, N_1 - 1}$$

$$\alpha_4 \frac{u_{iN_2} - u_{iN_2 - 1}}{h_2} + \beta_4 u_{iN_2} = \varphi_4(x_i), \; i = \overline{1, N_1 - 1}.$$

Вариант

4.

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0,$$
$$u_x(0, y) = \exp(y),$$
$$u_x(\pi, y) = -\exp(y),$$
$$u(x, 0) = \sin x,$$
$$u(x, 1) = e \sin x.$$

Аналитическое решение: $U(x, y) = \sin x \exp(y)$.

Результаты работы программы

```
Метод простых итераций:
1.1243228619554564E-5
Число шагов: 344

Метод Зейделя:
1.0173642462200463E-5
Число шагов: 189
```

```
Метод простых итераций с верхней релаксацией:
9.746150347863631E-6
Число шагов: 438
```

Приложение. Листинг программы.

```
import java.util.ArrayList;

public class Lab7 {
    static int count = 0;
    public static void main(String[] args) {
        double h = 0.1;
        int flag = 0;
        double[][] u = SolveEqLaplas(h, flag);
        int s = 9;

        double max = 0;
        double delta = 0;
        double temp;
        for (int i = 0; i < u[0].length; i++) {
            temp = Math.abs(u[s][i] - U(h * i, h * s));
            delta += temp * temp;
            if(temp > max){
                max = temp;
            }
        }
        System.out.println("Метод простых итераций:");
        System.out.println(delta);
        System.out.println("Число шагов: " + count);

        flag = 1;
        u = SolveEqLaplas(h, flag);

        max = 0;
        delta = 0;
        for (int i = 0; i < u[0].length; i++) {
            temp = Math.abs(u[s][i] - U(h * i, h * s));
            delta += temp * temp;
            if(temp > max){
                max = temp;
            }
        }
        System.out.println("\nМетод Зейделя:");
        System.out.println(delta);
        System.out.println("Число шагов: " + count);

        flag = 2;
        u = SolveEqLaplas(h, flag);

        max = 0;
        delta = 0;
        for (int i = 0; i < u[0].length; i++) {
            temp = Math.abs(u[s][i] - U(h * i, h * s));
            delta += temp * temp;
```

```java
        if(temp > max){
            max = temp;
        }
    }
    System.out.println("\nМетод простых итераций с верхней релаксацией:");
    System.out.println(delta);
    System.out.println("Число шагов: " + count);
}

private static double[][] SolveEqLaplas(double h, int flag){
    int N = (int)(Math.PI / h);
    int M = (int)(1 / h);
    double[] column = new double[(N + 1) * (M + 1)];
    double[][] A = new double[(N + 1) * (M + 1)][(N + 1) * (M + 1)];

    int k = 1;
    int l = 1;
    int p = 0;
    for(int i = 0; i < A.length; i++){
        for(int j = 0; j < A[0].length; j++){
            if(i >= 0 && i < N + 1){
                if(j == i) {
                    A[i][j] = 1;
                    column[i] = Math.sin(i * h);
                }
            }
            else if(i == k * (N + 1) && k < M){
                if(j == i) {
                    A[i][j] = 1;
                    column[i] = -h * Math.exp(k * h);
                }
                else if(j - 1 == i) A[i][j] = -1;
                else if(j == A[0].length - 1) k++;
            }
            else if(i == l * (N + 1) + N && l < M){
                if(j == i) {
                    A[i][j] = 1;
                    column[i] = -h * Math.exp(l * h);
                }
                else if(j + 1 == i) A[i][j] = -1;
                else if(j == A[0].length - 1) l++;
            }
            else if(i >= k * (N + 1)){
                if(j == i) {
                    A[i][j] = 1;
                    column[i] = Math.E * Math.sin(p * h);
                    p++;
                }
            }
            else{
                if(j == i) {
                    A[i][j] = -4;
                    column[i] = 0;
                }
                else if(j - 1 == i) A[i][j] = 1;
                else if(j + 1 == i) A[i][j] = 1;
                else if(j + (N + 1) == i) A[i][j] = 1;
                else if(j - (N + 1) == i) A[i][j] = 1;
            }
        }
    }

    double[][] u_ = new double[M + 1][N + 1];
    if(flag == 0) {
        double[] u = Iteration(A, column);
        int n = 0;
        for (int i = 0; i < u_.length; i++) {
            for (int j = 0; j < u_[0].length; j++) {
                u_[i][j] = u[n];
                n++;
            }
        }
```

```java
    }
    else if(flag == 1){
        double[] u = Zeidel(A, column);
        int n = 0;
        for (int i = 0; i < u_.length; i++) {
            for (int j = 0; j < u_[0].length; j++) {
                u_[i][j] = u[n];
                n++;
            }
        }
    }
    else if(flag == 2){
        double[] u = IterationWithRelax(A, column);
        int n = 0;
        for (int i = 0; i < u_.length; i++) {
            for (int j = 0; j < u_[0].length; j++) {
                u_[i][j] = u[n];
                n++;
            }
        }
    }
    return u_;
}


private static double U(double x, double y){
    return Math.sin(x) * Math.exp(y);
}
private static double[] Iteration(double[][] matrix, double[] column)
{
    double[] result = new double[column.length];

    double[][] alpha = new double[matrix.length][matrix[0].length];
    double[] betta = new double[column.length];
    double[] x_cur = new double[column.length];
    double[] x_prev = new double[column.length];
    boolean norm = true;
    double sum = 0;
    double max_sum = 0;
    int count_iteration = 0;

    for (int i = 0; i < matrix.length; i++)
    {
        sum = 0;
        for (int j = 0; j < matrix[0].length + 1; j++)
        {
            if (j != i && j != matrix[0].length)
            {
                alpha[i][j] = -matrix[i][j] / matrix[i][i];
                sum += Math.abs(alpha[i][j]);
            }
            else if (j == i) alpha[i][j] = 0;

            if (j == matrix[0].length)
            {
                betta[i] = column[i] / matrix[i][i];
                x_prev[i] = betta[i];
            }
        }

        if (sum > max_sum) max_sum = sum;
    }

    if (max_sum > 1) norm = false;

    if(norm)
    {
        double epsilon = 0.0001;
        double epsilon_i = 1;

        while (epsilon_i > epsilon)
        {
```

```java
            epsilon_i = 0;
            x_cur = SumVectors(betta, MultyMatrVector(alpha, x_prev));

            for (int i = 0; i < column.length; i++)
                epsilon_i += Math.pow(x_prev[i] - x_cur[i], 2);
            epsilon_i = Math.sqrt(epsilon_i);

            x_prev = x_cur;
            count_iteration++;
        }
    }
    count = count_iteration;
    result = x_cur;
    return result;
}

static double[] MultyMatrVector(double[][] matrix, double[] column)
{
    double[] result = new double[column.length];
    for(int i = 0; i < matrix.length; i++)
    {
        for(int j = 0; j < matrix[0].length; j++)
        {
            result[i] += matrix[i][j] * column[j];
        }
    }
    return result;
}

static double[] SumVectors(double[] a, double[] b)
{
    double[] result = new double[a.length];

    for (int i = 0; i < a.length; i++)
    {
        result[i] = a[i] + b[i];
    }
    return result;
}

static double[] MultyNumberVector(double[] a, double lambda){
    double[] res = new double[a.length];
    for (int i = 0; i < a.length; i++) {
        res[i] = lambda * a[i];
    }
    return res;
}

static double[] Zeidel(double[][] matrix, double[] column)
{
    double[] result = new double[column.length];

    double[][] alpha = new double[matrix.length][matrix[0].length];
    double[] betta = new double[column.length];
    double[] x_prev = new double[column.length];
    double[] x_cur = new double[column.length];
    boolean norm = true;
    double sum = 0;
    double max_sum = 0;
    int count_iteration = 0;

    for (int i = 0; i < matrix.length; i++)
    {
        sum = 0;
        for (int j = 0; j < matrix[0].length + 1; j++)
        {
            if (j != i && j != matrix[0].length)
            {
                alpha[i][j] = -matrix[i][j] / matrix[i][i];
                sum += Math.abs(alpha[i][j]);
            }
            else if (j == i) alpha[i][j] = 0;
```

```
            if (j == matrix[0].length)
            {
                betta[i] = column[i] / matrix[i][i];
                x_prev[i] = betta[i];
            }
        }

        if (sum > max_sum) max_sum = sum;
    }
    if (max_sum > 1) norm = false;

    double[] vctr = new double[column.length];

    if(norm)
    {
        double epsilon = 0.0001;
        double epsilon_i = 1;

        ArrayList<double[]> str = StrOfMatr(alpha);
        while (epsilon_i > epsilon)
        {
            epsilon_i = 0;

            for (int i = 0; i < vctr.length; i++)
            {
                vctr[i] = x_prev[i];
            }

            for (int i = 0; i < x_cur.length; i++)
            {
                x_cur[i] = betta[i] + MultyStrVector(str.get(i), x_prev);
                x_prev[i] = x_cur[i];
            }

            for (int i = 0; i < column.length; i++)
                epsilon_i += Math.pow(vctr[i] - x_cur[i], 2);
            epsilon_i = Math.sqrt(epsilon_i);
            count_iteration++;
        }

    }
    count = count_iteration;
    result = x_cur;
    return result;
}

static ArrayList<double[]> StrOfMatr(double[][] matrix)
{
    ArrayList<double[]> str = new ArrayList<>();
    //double[] mas = new double[matrix.GetLength(0)];

    for(int i = 0; i < matrix.length; i++)
    {
        double[] mas = new double[matrix.length];
        for (int j = 0; j < matrix[0].length; j++)
        {
            mas[j] = matrix[i][j];
        }

        str.add(mas);
    }
    return str;
}

static double MultyStrVector(double[] str, double[] vctr)
{
    double result = 0;

    for(int i = 0; i < str.length; i++)
    {
        result += str[i] * vctr[i];
```

```
        }
        return result;
    }

    static double[] IterationWithRelax(double[][] matrix, double[] column){
        double[] result = new double[column.length];

        double[][] alpha = new double[matrix.length][matrix[0].length];
        double[] betta = new double[column.length];
        double[] x_cur = new double[column.length];
        double[] x_prev = new double[column.length];
        double[] x_predict = new double[column.length];
        boolean norm = true;
        double sum = 0;
        double max_sum = 0;
        int count_iteration = 0;
        double w = 1.01;

        for (int i = 0; i < matrix.length; i++)
        {
            sum = 0;
            for (int j = 0; j < matrix[0].length + 1; j++)
            {
                if (j != i && j != matrix[0].length)
                {
                    alpha[i][j] = -matrix[i][j] / matrix[i][i];
                    sum += Math.abs(alpha[i][j]);
                }
                else if (j == i) alpha[i][j] = 0;

                if (j == matrix[0].length)
                {
                    betta[i] = column[i] / matrix[i][i];
                    x_prev[i] = betta[i];
                }
            }

            if (sum > max_sum) max_sum = sum;
        }

        if (max_sum > 1) norm = false;

        if(norm)
        {
            double epsilon = 0.0001;
            double epsilon_i = 1;

            while (epsilon_i > epsilon)
            {
                epsilon_i = 0;
                x_predict = SumVectors(betta, MultyMatrVector(alpha, x_prev));
                x_cur = SumVectors(MultyNumberVector(x_predict, w), MultyNumberVector(x_prev, 1 - w));

                for (int i = 0; i < column.length; i++)
                    epsilon_i += Math.pow(x_prev[i] - x_cur[i], 2);
                epsilon_i = Math.sqrt(epsilon_i);

                x_prev = x_cur;
                count_iteration++;
            }
        }
        count = count_iteration;
        result = x_cur;
        return result;
    }
}
```

Выводы

В данной работе реализована конечно-разностная схемы для решения краевой задачи для дифференциального уравнения эллиптического типа. Для решения дискретного аналога применить следующие методы: метод простых итераций (метод Либмана), метод Зейделя, метод простых итераций с верхней релаксацией. Для сравнения с точным решением вычисляется погрешность как корень квадратный из суммы квадратов погрешностей между точным решением и полученным решением на каждом шаге j для сечения по y (s = 9).

Погрешность составила порядка 10^(-5).