Московский авиационный институт

(национальный исследовательский университет)

Институт информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №7

«Численное решение уравнений с частными производными эллиптического типа»

Вариант №4

Студент: Железнов Д.А.

Группа: М8О-409Б-19

Руководитель: Пивоваров Д.Е.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 09.12.2022

Москва 2022

**Лабораторная работа №7**

Метод конечных разностей для решения задачи эллиптического типа

**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 .

**Описание метода**

Рассмотрим уравнение Пуассона для третьей краевой задачи в прямоугольнике:

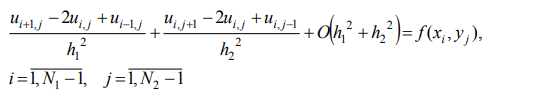
Изображение выглядит как текст

Автоматически созданное описание

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:



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:

Изображение выглядит как антенна

Автоматически созданное описание

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

Изображение выглядит как текст

Автоматически созданное описание

Перед решение методом простых итераций необходимо задать начальное приближение.

Условие выхода:



где ε – наперёд заданная точность.

**Аппроксимация граничных условий**

Граничные условия аппроксимируются с первым порядком:

Изображение выглядит как текст

Автоматически созданное описание

**Вариант**

**Изображение выглядит как текст

Автоматически созданное описание**

**Результаты работы программы**

**Изображение выглядит как текст

Автоматически созданное описание**

**Изображение выглядит как текст

Автоматически созданное описание**

**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).

**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;  
 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++;  
 }  
 }  
 }  
 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)  
 {  
 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;  
 }  
}