Московский авиационный институт

(национальный исследовательский университет)

Институт информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №8

«Метод конечных разностей решения многомерных задач математической физики»

Вариант №4

Студент: Железнов Д.А.

Группа: М8О-409Б-19

Руководитель: Пивоваров Д.Е.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 20.12.2022

Москва 2022

**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 .

**Описание метода**

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

Общая поставка такой задачи выглядит следующим образом:

Изображение выглядит как текст

Автоматически созданное описание

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



**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.

Изображение выглядит как текст

Автоматически созданное описание

Т.е. здесь оператор на первом временном полуслое аппроксимируется неявно, – явно.

На втором временном полуслое наоборот.

С помощью скалярных прогонок в количестве J – 1 в направлении переменной x получаем значение на первом временном полуслое.

Уже на втором шаге с помощью скалярных прогонок в количестве I – 1 в направлении переменной y получаем значение 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.

Изображение выглядит как текст

Автоматически созданное описание

С помощью скалярных прогонок в количестве J – 1 в направлении переменной x получаем значение на первом временном полуслое.

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

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

**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 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)) .

**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;  
  
 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;  
 }  
  
 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;  
 }  
}