Московский авиационный институт

(национальный исследовательский университет)

Институт информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №5

«Численное решение уравнений с частными производными параболического типа»

Вариант №4

Студент: Железнов Д.А.

Группа: М8О-409Б-19

Руководитель: Пивоваров Д.Е.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 28.10.2022

Москва 2022

**Лабораторная работа №5**

Численное решение уравнений с частными производными параболического типа. Понятие о методе конечных разностей. Конечно-разностные схемы.

**Задача**

Используя явную и неявную конечно-разностные схемы, а также схему Кранка - Николсона, решить начально-краевую задачу для дифференциального уравнения параболического типа. Осуществить реализацию трех вариантов аппроксимации граничных условий, содержащих производные: двухточечная аппроксимация с первым порядком, трехточечная аппроксимация со вторым порядком, двухточечная аппроксимация со вторым порядком. В различные моменты времени вычислить погрешность численного решения путем сравнения результатов с приведенным в задании аналитическим решением U(x, t) . Исследовать зависимость погрешности от сеточных параметров τ, h.

**Описание метода**

Рассматривается решение уравнения теплопроводности с граничными условиями второго рода на обоих концах интервала, т.е. рассматривается следующая задача:

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание



Для решения такой задачи применяют метод конченых разностей. Для этого вводят понятие разностной сетки



с пространственным шагом h и шагом по времени τ.

Введём понятие сеточной функции. Сеточной функцией называют следующее отображение целых аргументов j и k:

Затем происходит аппроксимация производной по времени и второй производной по пространству.

Если аппроксимировать вторую производную по пространству на нижнем временном слое, то получим **явную** конечно-разностную схему.

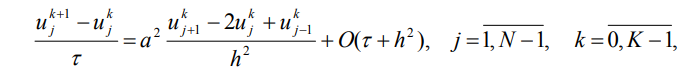
Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Явная конечно-разностная схема:



где для каждого j-го уравнения неизвестна только одна величина , которая может быть явно выражена из этого уравнения:



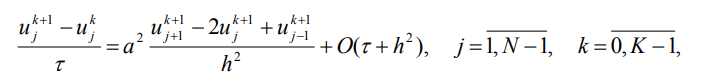
Для устойчивости данной схемы накладывается ограничение на σ: σ <= 1/2.

Если аппроксимировать производную по пространству на верхнем временном слое, то получим **неявную** конечно-разностную схему

Изображение выглядит как текст, антенна

Автоматически созданное описание

Неявная схема:



где для нахождения сеточной функции на верхнем временном слое необходимо решать СЛАУ с трёхдиагональной матрицей:

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

**Явно-неявная** конечно-разностная схема имеет следующий вид:

Изображение выглядит как текст, антенна

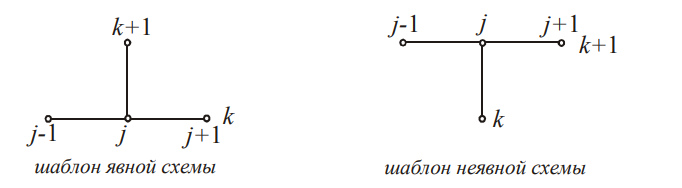
Автоматически созданное описание

где 0ϴ1.

При ϴ = 0 получается явная схема, при ϴ = 1 – неявная, при ϴ = 1/2 - схема Кранка-Николсона.

Здесь так же, как и в неявной схеме для нахождения на каждом шаге необходимо решать СЛАУ с трёхдиагональной матрицей.

Шаблоны трёх схем:



Шаблон схемы Кранка-Николсона:

Изображение выглядит как часы, датчик

Автоматически созданное описание

**Аппроксимация граничных условий**

Рассмотрим 3 способа аппроксимации граничных условий 2Т1П, 3Т2П, 2Т2П.

Двухточечная, с первым порядком:



Трёхточечная, со вторым порядком:

Изображение выглядит как текст

Автоматически созданное описание

При использовании неявной и явно-неявной схемы СЛАУ теряет трёхдиагональность, поэтому сначала необходимо привести СЛАУ к трёхдиагональному виду линейной комбинацией первой строки со второй, предпоследней с последней.

Двухточечная, со вторым порядком:

Для того, чтобы получить эти формулы для начала раскладывают в ряд Тейлора в окрестности x = 0 и в окрестности x = l:

Изображение выглядит как текст

Автоматически созданное описание

Далее берут информацию из исходного уравнения, выражая оттуда вторую производную и подставляя это выражение, получают:

Изображение выглядит как текст, стол

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 явная и неявная конечно-разностные схемы, а также схема Кранка – Николсона для решения начально-краевую задачу для дифференциального уравнения параболического типа. Осуществлена реализацию трех вариантов аппроксимации граничных условий, содержащих производные: двухточечная аппроксимация с первым порядком, трехточечная аппроксимация со вторым порядком, двухточечная аппроксимация со вторым порядком.

Для сравнения трёх схем вычисляется погрешность как сумма абсолютной разности между точным решением и полученным решением на каждом шаге j для одного момента времени (s = 25).

Погрешность для каждой схемы и каждой аппроксимации составила порядка 10^(-5).

**Приложение. Листинг программы.**

import java.util.ArrayList;

import java.util.Collections;

public class Lab5 {

public static void main(String[] args) {

double a = 1;

int N = 100;

double l = Math.PI;

double h = l / N;

double sigma = 0.5;

double tau = sigma \* h \* h / a;

int flag = 0;

double[][] u = explicitMethod(a, N, h, sigma, tau, flag);

int s = 25;

double max = 0;

double delta = 0;

double temp;

for (int i = 0; i < N; i++) {

temp = Math.abs(u[s][i] - resultFunc(a, h \* i, tau \* s));

delta += temp \* temp;

if(temp > max){

max = temp;

}

}

System.out.println("Явный метод 2Т1П");

//System.out.println(Math.sqrt(delta) + " " + max);

System.out.println(max);

flag = 1;

u = explicitMethod(a, N, h, sigma, tau, flag);

max = 0;

delta = 0;

for (int i = 0; i < N; i++) {

temp = Math.abs(u[s][i] - resultFunc(a, h \* i, tau \* s));

delta += temp \* temp;

if(temp > max){

max = temp;

}

}

System.out.println("\nЯвный метод 3Т2П");

//System.out.println(Math.sqrt(delta) + " " + max);

System.out.println(max);

flag = 2;

u = explicitMethod(a, N, h, sigma, tau, flag);

max = 0;

delta = 0;

for (int i = 0; i < N; i++) {

temp = Math.abs(u[s][i] - resultFunc(a, h \* i, tau \* s));

delta += temp \* temp;

if(temp > max){

max = temp;

}

}

System.out.println("\nЯвный метод 2Т2П");

//System.out.println(Math.sqrt(delta) + " " + max);

System.out.println(max);

flag = 0;

u = implicitMethod(a, N, h, sigma, tau, flag);

max = 0;

delta = 0;

for (int i = 0; i < N; i++) {

temp = Math.abs(u[s][i] - resultFunc(a, h \* i, tau \* s));

delta += temp \* temp;

if(temp > max){

max = temp;

}

}

System.out.println("\nНеявный метод 2Т1П");

//System.out.println(Math.sqrt(delta) + " " + max);

System.out.println(max);

flag = 1;

u = implicitMethod(a, N, h, sigma, tau, flag);

max = 0;

delta = 0;

for (int i = 0; i < N; i++) {

temp = Math.abs(u[s][i] - resultFunc(a, h \* i, tau \* s));

delta += temp \* temp;

if(temp > max){

max = temp;

}

}

System.out.println("\nНеявный метод 3Т2П");

//System.out.println(Math.sqrt(delta) + " " + max);

System.out.println(max);

flag = 2;

u = implicitMethod(a, N, h, sigma, tau, flag);

max = 0;

delta = 0;

for (int i = 0; i < N; i++) {

temp = Math.abs(u[s][i] - resultFunc(a, h \* i, tau \* s));

delta += temp \* temp;

if(temp > max){

max = temp;

}

}

System.out.println("\nНеявный метод 2Т2П");

//System.out.println(Math.sqrt(delta) + " " + max);

System.out.println(max);

flag = 0;

u = CrankNicolson(a, N, h, sigma, tau, flag);

max = 0;

delta = 0;

for (int i = 0; i < N; i++) {

temp = Math.abs(u[s][i] - resultFunc(a, h \* i, tau \* s));

delta += temp \* temp;

if(temp > max){

max = temp;

}

}

System.out.println("\nМетод Кранка-Николсона 2Т1П");

//System.out.println(Math.sqrt(delta) + " " + max);

System.out.println(max);

flag = 1;

u = CrankNicolson(a, N, h, sigma, tau, flag);

max = 0;

delta = 0;

for (int i = 0; i < N; i++) {

temp = Math.abs(u[s][i] - resultFunc(a, h \* i, tau \* s));

delta += temp \* temp;

if(temp > max){

max = temp;

}

}

System.out.println("\nМетод Кранка-Николсона 3Т2П");

//System.out.println(Math.sqrt(delta) + " " + max);

System.out.println(max);

flag = 2;

u = CrankNicolson(a, N, h, sigma, tau, flag);

max = 0;

delta = 0;

for (int i = 0; i < N; i++) {

temp = Math.abs(u[s][i] - resultFunc(a, h \* i, tau \* s));

delta += temp \* temp;

if(temp > max){

max = temp;

}

}

System.out.println("\nМетод Кранка-Николсона 2Т2П");

//System.out.println(Math.sqrt(delta) + " " + max);

System.out.println(max);

}

static double resultFunc(double a, double x, double t){

return Math.exp(-a \* t) \* Math.sin(x);

}

static double[][] explicitMethod(double a, int N, double h, double sigma, double tau, int flag) {

int K = N \* 2;

double[][] u = new double[K + 1][N + 1];

for (int i = 0; i < u[1].length; i++) {

u[0][i] = Math.sin(i \* h);

}

for (int k = 0; k < K; k++) {

for(int j = 1; j < N; j++){

u[k + 1][j] = sigma \* u[k][j + 1] + (1 - 2 \* sigma) \* u[k][j] + sigma \* u[k][j - 1];

}

if(flag == 0) {

u[k + 1][0] = u[k + 1][1] - h \* Math.exp(-a \* (k + 1) \* tau);

u[k + 1][N] = u[k + 1][N - 1] - h \* Math.exp(-a \* (k + 1) \* tau);

}

else if(flag == 1){

u[k + 1][0] = (-2 \* h \* Math.exp(-a \* (k + 1) \* tau) + 4 \* u[k + 1][1] - u[k + 1][2]) / 3;

u[k + 1][N] = (-2 \* h \* Math.exp(-a \* (k + 1) \* tau) + 4 \* u[k + 1][N - 1] - u[k + 1][N - 2]) / 3;

}

else if(flag == 2){

u[k + 1][0] = (u[k + 1][1] + h \* h / (2 \* a \* tau) \* u[k][0] - h \* Math.exp(-a \* (k + 1) \* tau)) / (1 + h \* h / (2 \* a \* tau));

u[k + 1][N] = (u[k + 1][N - 1] + h \* h / (2 \* a \* tau) \* u[k][N] - h \* Math.exp(-a \* (k + 1) \* tau)) / (1 + h \* h / (2 \* a \* tau));

}

}

return u;

}

static double[][] implicitMethod(double a, int N, double h, double sigma, double tau, int flag){

int K = N \* 2;

double[][] u = new double[K + 1][N + 1];

double[] a\_arr = new double[N + 1];

double[] b\_arr = new double[N + 1];

double[] c\_arr = new double[N + 1];

double[] d\_arr = new double[N + 1];

for (int i = 0; i < u[1].length; i++) {

u[0][i] = Math.sin(i \* h);

}

double koef = 1 / (2 \* h \* sigma);

for (int i = 0; i < N + 1; i++) {

if(i == 0){

if(flag == 0){

a\_arr[0] = 0;

b\_arr[0] = -1 / h;

c\_arr[0] = 1 / h;

}

else if(flag == 1) {

a\_arr[0] = 0;

b\_arr[0] = -1 / h;

c\_arr[0] = 1 / h - koef;

}

else if(flag == 2){

a\_arr[0] = 0;

b\_arr[0] = 2 \* a / h + h / tau;

c\_arr[0] = -2 \* a / h;

}

}

else if(i < N) {

a\_arr[i] = sigma;

b\_arr[i] = -(1 + 2 \* sigma);

c\_arr[i] = sigma;

}

else{

if(flag == 0){

a\_arr[i] = -1 / h;

b\_arr[i] = 1 / h;

c\_arr[i] = 0;

}

else if(flag == 1){

a\_arr[i] = -1 / h + koef;

b\_arr[i] = 1 / h;

c\_arr[i] = 0;

}

else if(flag == 2){

a\_arr[i] = -2 \* a / h;

b\_arr[i] = 2 \* a / h + h / tau;

c\_arr[i] = 0;

}

}

}

for(int k = 0; k < K; k++){

for(int j = 0; j < N; j++){

if(j == 0){

if(flag == 0) {

d\_arr[0] = Math.exp(-a \* (k + 1) \* tau);

}

else if(flag == 1){

d\_arr[0] = Math.exp(-a \* (k + 1) \* tau) - koef \* u[k][1];

}

else if(flag == 2){

d\_arr[0] = h / tau \* u[k][0] - 2 \* a \* Math.exp(-a \* (k + 1) \* tau);

}

}

else{

d\_arr[j] = -u[k][j];

}

}

if(flag == 0) {

d\_arr[N] = -Math.exp(-a \* (k + 1) \* tau);

}

else if(flag == 1){

d\_arr[N] = -Math.exp(-a \* (k + 1) \* tau) + koef \* u[k][N - 1];

}

else if(flag == 2){

d\_arr[N] = h / tau \* u[k][N] - 2 \* a \* Math.exp(-a \* (k + 1) \* tau);

}

ArrayList<Double> res\_progonka = Progonka(a\_arr, b\_arr, c\_arr, d\_arr);

for (int i = 0; i < N + 1; i++) {

u[k + 1][i] = res\_progonka.get(i);

}

}

return u;

}

static double[][] CrankNicolson(double a, int N, double h, double sigma, double tau, int flag){

int K = N \* 2;

double[][] u = new double[K + 1][N + 1];

//sigma = 0.5;

double[] a\_arr = new double[N + 1];

double[] b\_arr = new double[N + 1];

double[] c\_arr = new double[N + 1];

double[] d\_arr = new double[N + 1];

for (int i = 0; i < u[1].length; i++) {

u[0][i] = Math.sin(i \* h);

}

double koef = 1 / (2 \* h \* sigma);

double r = a \* tau / (h \* h);

for (int i = 0; i < N + 1; i++) {

if(i == 0){

if(flag == 0){

a\_arr[0] = 0;

b\_arr[0] = -1 / h;

c\_arr[0] = 1 / h;

}

else if(flag == 1) {

a\_arr[0] = 0;

b\_arr[0] = -1 / h;

c\_arr[0] = 1 / h - koef;

}

else if(flag == 2){

a\_arr[0] = 0;

b\_arr[0] = 2 \* a / h + h / tau;

c\_arr[0] = -2 \* a / h;

}

}

else if(i < N) {

a\_arr[i] = -r / 2;

b\_arr[i] = r + 1;

c\_arr[i] = -r / 2;

}

else{

if(flag == 0){

a\_arr[i] = -1 / h;

b\_arr[i] = 1 / h;

c\_arr[i] = 0;

}

else if(flag == 1){

a\_arr[i] = -1 / h + koef;

b\_arr[i] = 1 / h;

c\_arr[i] = 0;

}

else if(flag == 2){

a\_arr[i] = -2 \* a / h;

b\_arr[i] = 2 \* a / h + h / tau;

c\_arr[i] = 0;

}

}

}

for(int k = 0; k < K; k++){

for(int j = 0; j < N; j++){

if(j == 0){

if(flag == 0) {

d\_arr[0] = Math.exp(-a \* (k + 1) \* tau);

}

else if(flag == 1){

d\_arr[0] = Math.exp(-a \* (k + 1) \* tau) - koef \* u[k][1];

}

else if(flag == 2){

d\_arr[0] = h / tau \* u[k][0] - 2 \* a \* Math.exp(-a \* (k + 1) \* tau);

}

}

else{

d\_arr[j] = r / 2 \* (u[k][j - 1] + u[k][j + 1]) + u[k][j] \* (1 - r);

}

}

if(flag == 0) {

d\_arr[N] = -Math.exp(-a \* (k + 1) \* tau);

}

else if(flag == 1){

d\_arr[N] = -Math.exp(-a \* (k + 1) \* tau) + koef \* u[k][N - 1];

}

else if(flag == 2){

d\_arr[N] = h / tau \* u[k][N] - 2 \* a \* Math.exp(-a \* (k + 1) \* tau);

}

ArrayList<Double> res\_progonka = Progonka(a\_arr, b\_arr, c\_arr, d\_arr);

for (int i = 0; i < N + 1; i++) {

u[k + 1][i] = res\_progonka.get(i);

}

}

return u;

}

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;

}

}