**Московский Авиационный Институт**

**(Национальный Исследовательский Университет)**

Институт информационных технологий и прикладной математики

**Отчет по лабораторной работе №5**

по курсу «Численные методы»

Студент: Марков А.Н.

Группа: М8О-408Б-18

Преподаватель: Пивоваров Д.Е.

Москва

2021

**Текст задания**

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 . Исследовать зависимость погрешности от сеточных параметров .

Вариант 3.

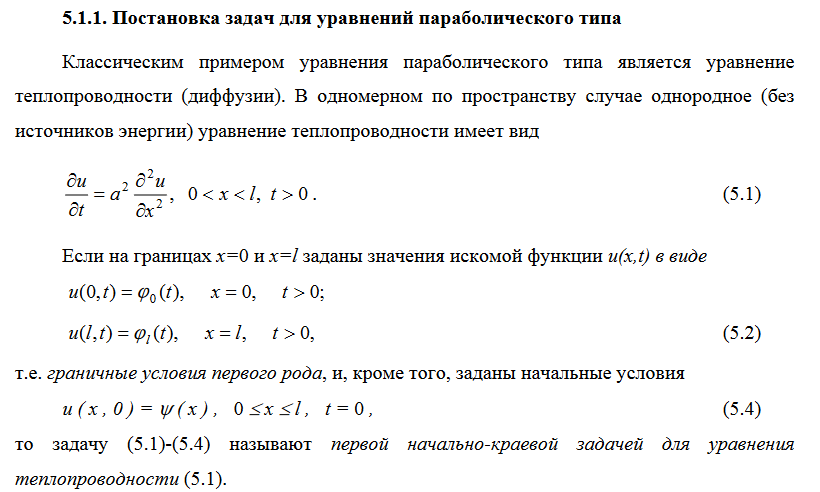
, ,

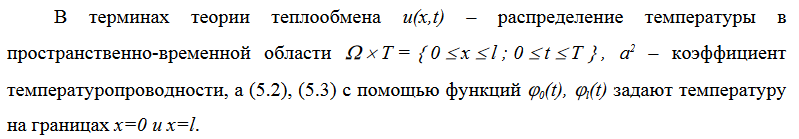


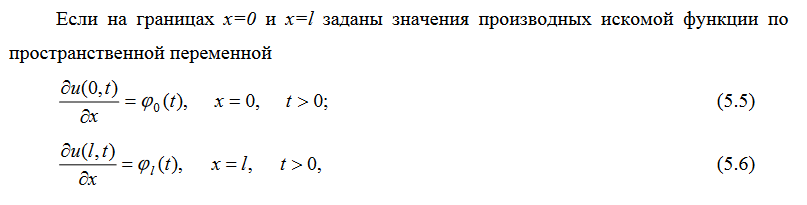
.

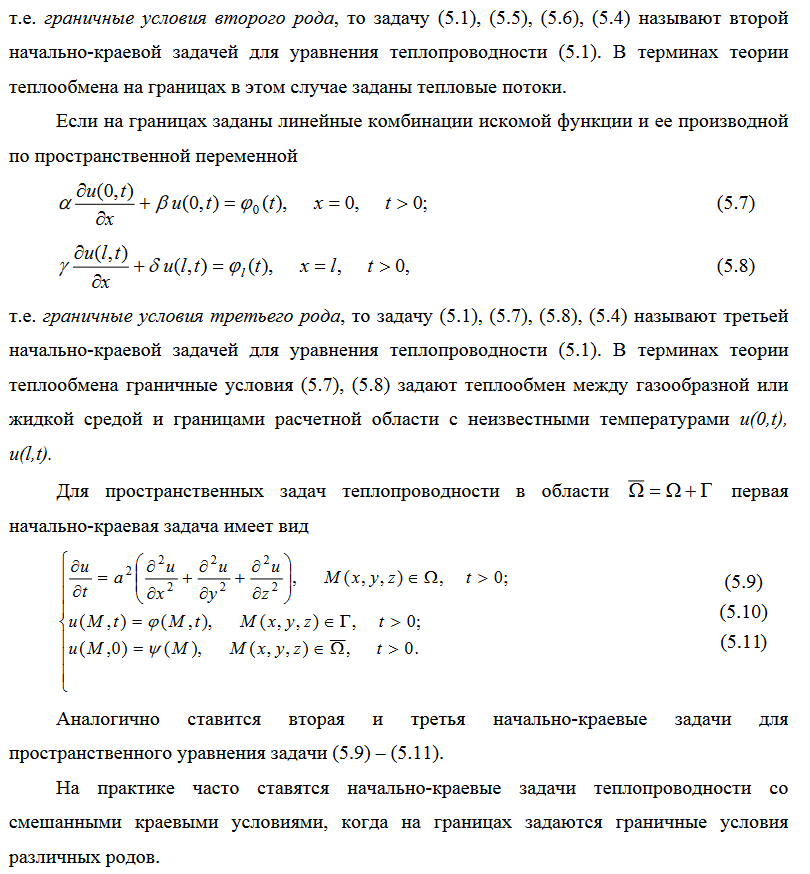
Аналитическое решение: .

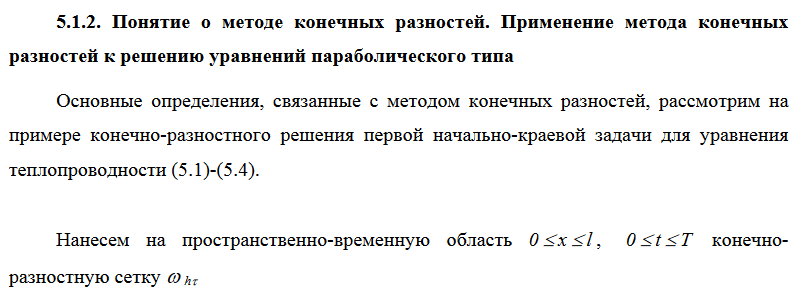
**Теория**

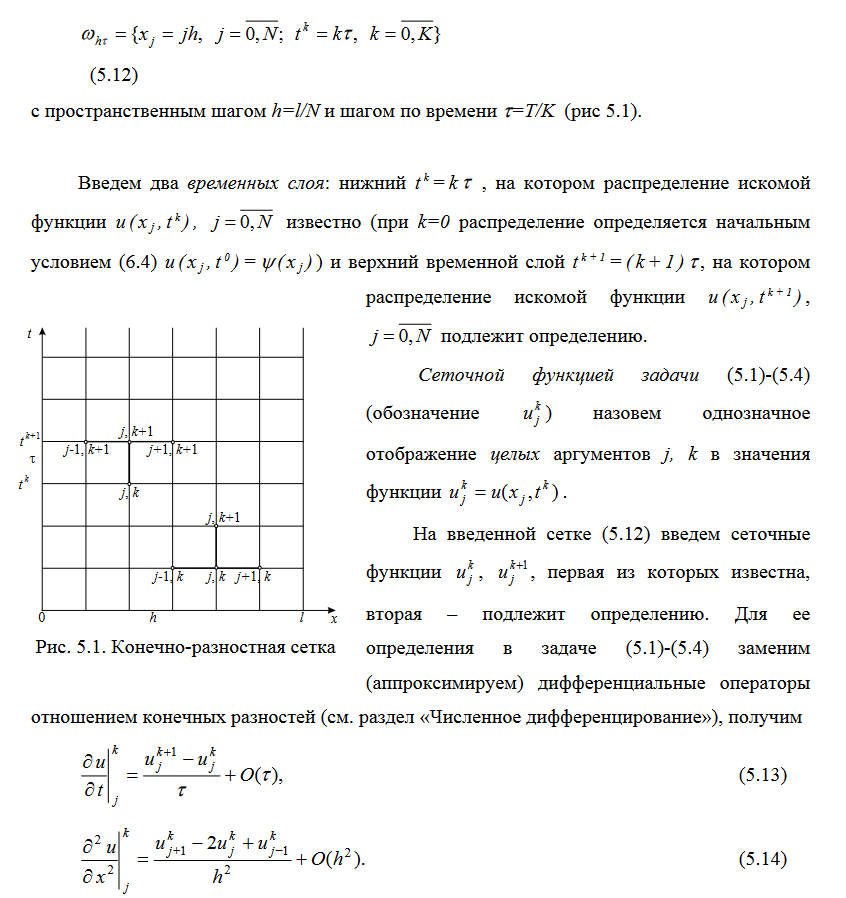


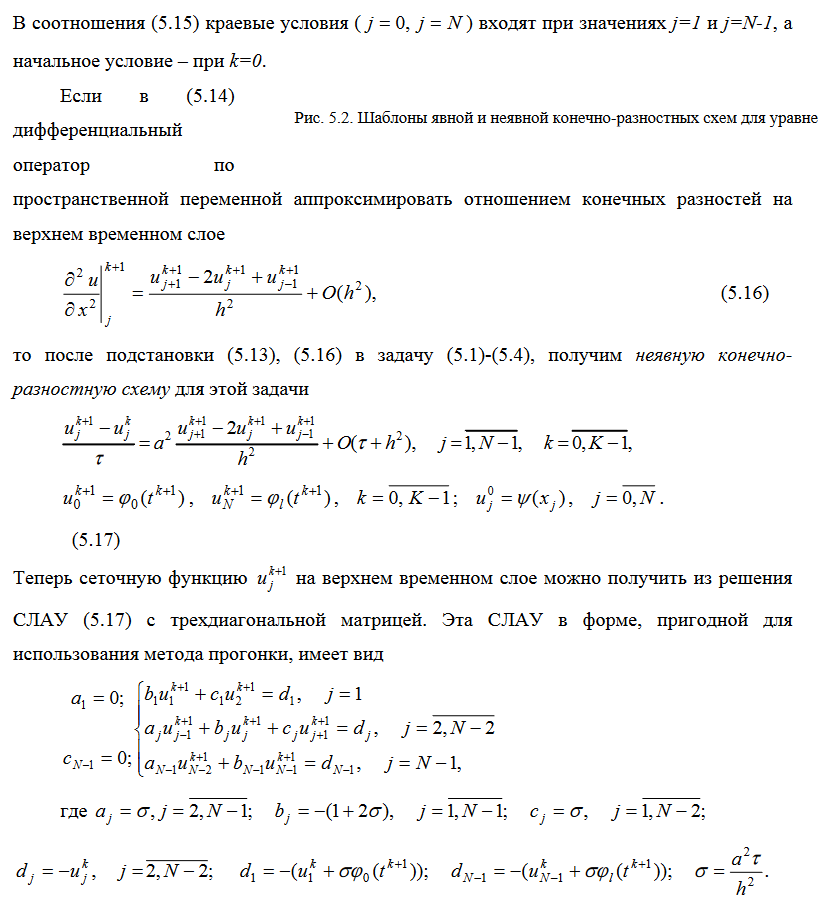
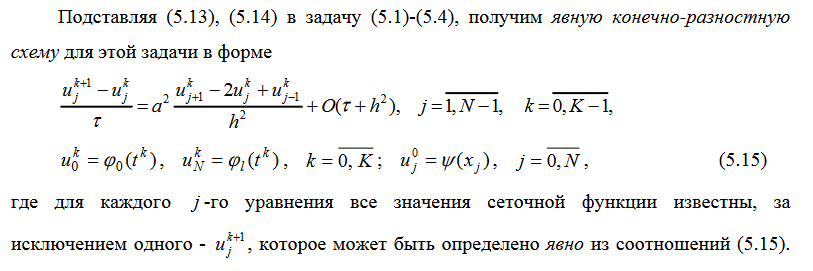


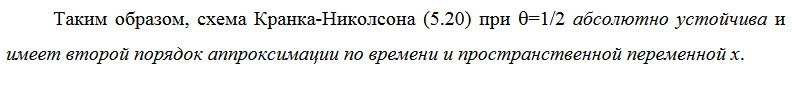
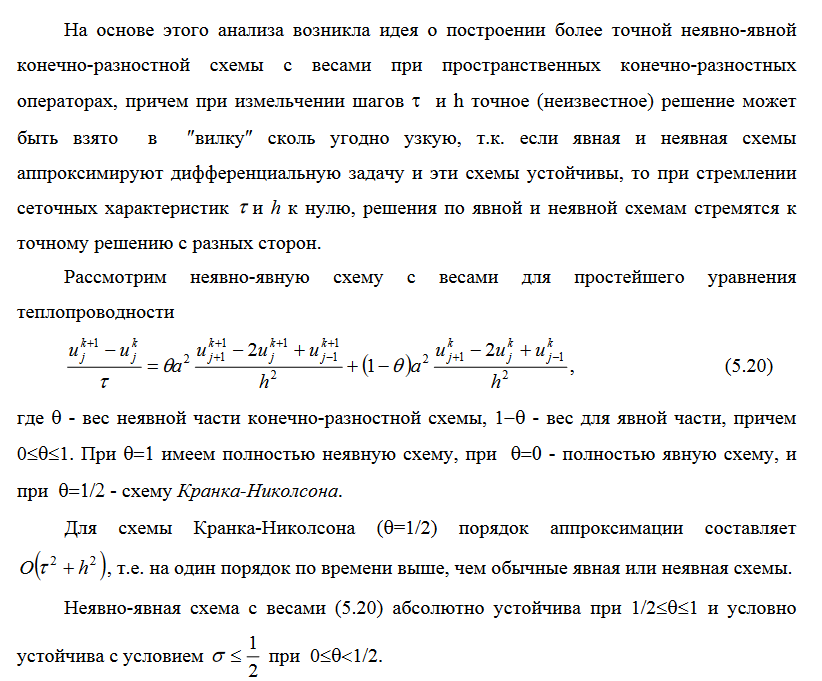
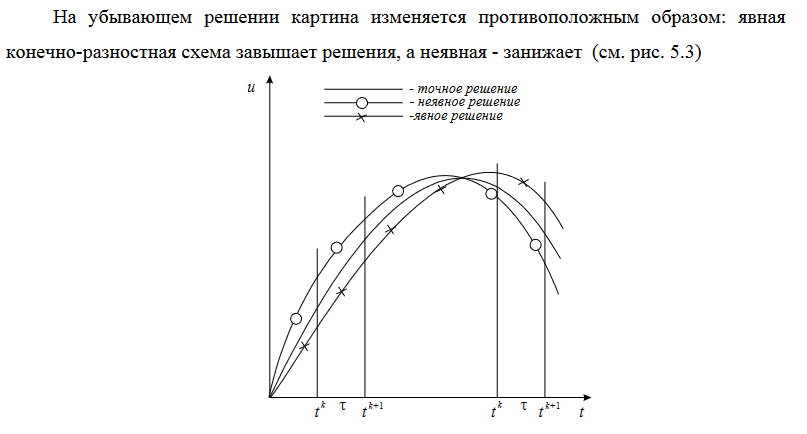
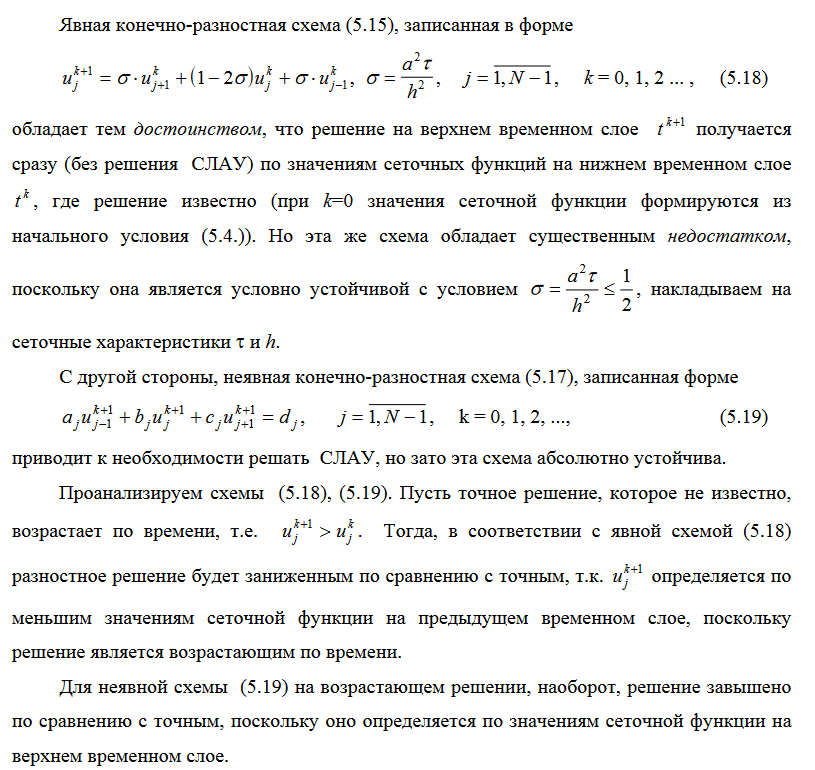
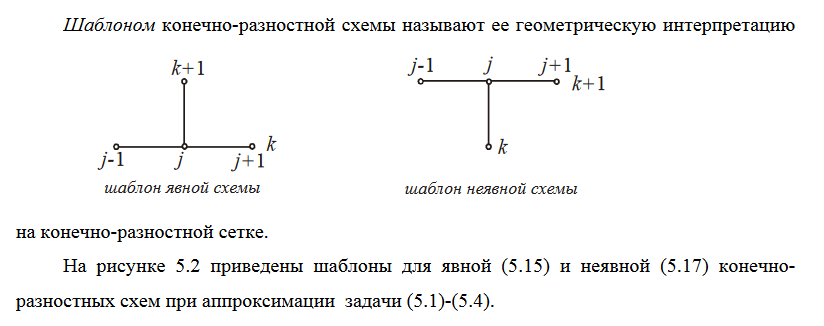


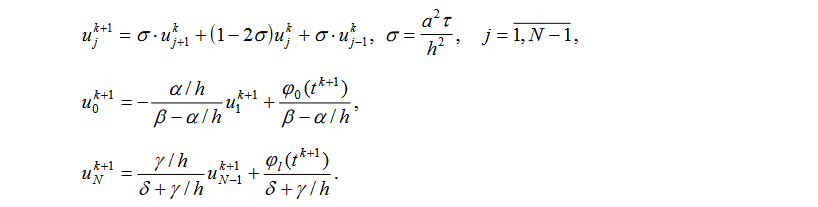
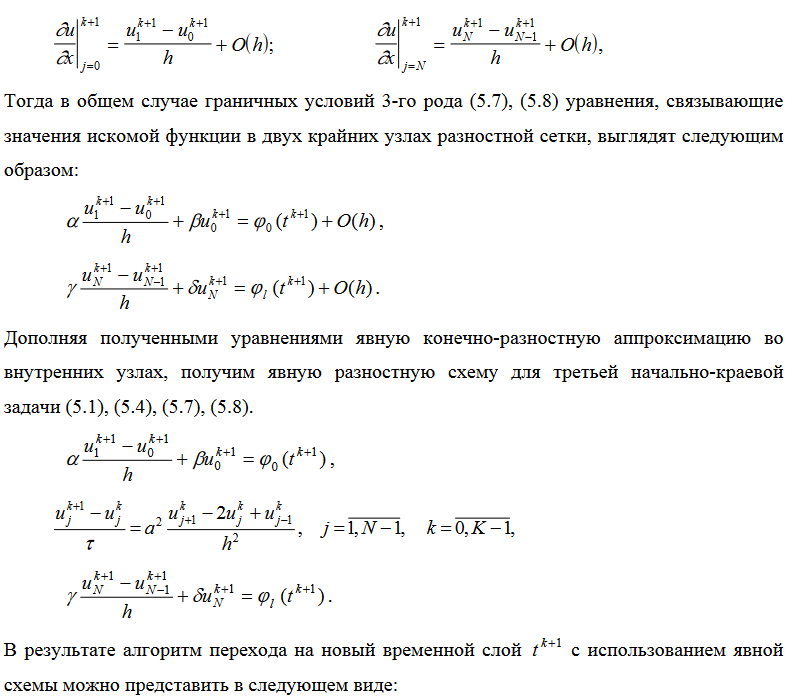
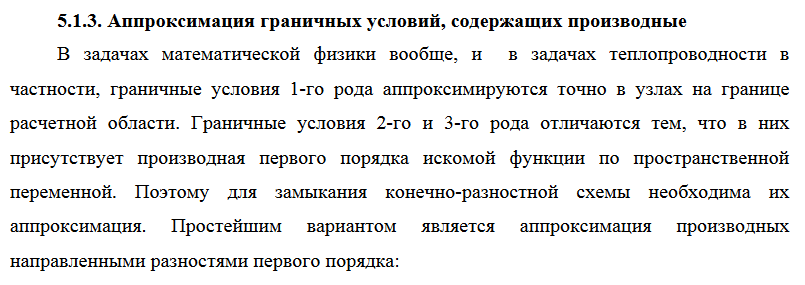


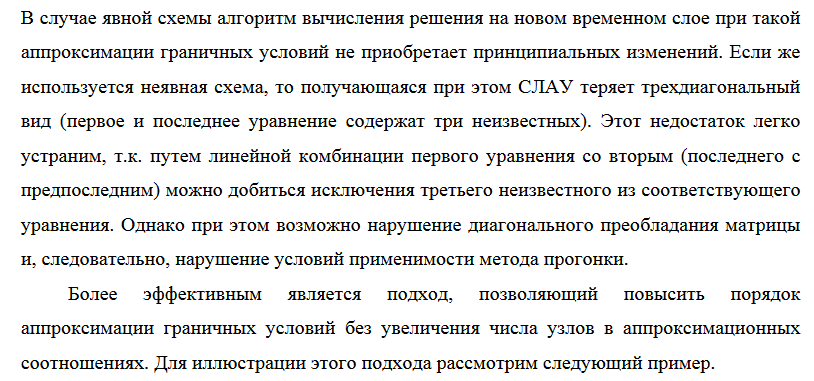
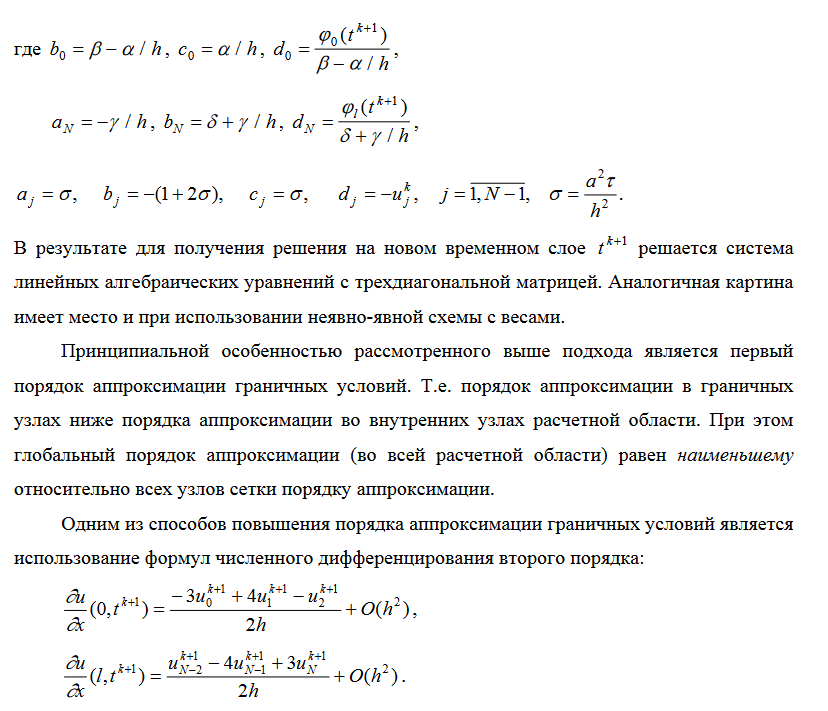
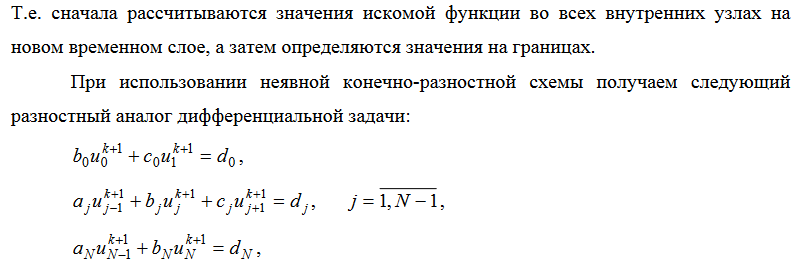












**Код программы**

package ru.mai.chm.lab1;

import com.google.gson.Gson;

import com.google.gson.GsonBuilder;

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

import java.util.function.BiFunction;

import java.util.function.Function;

public class Solver {

private static final Gson gson = new GsonBuilder()

.setPrettyPrinting()

.create();

private final double leftBound;

private final double rightBound;

private final double a;

private final int N;

private final int K;

private final double T;

private final double b;

private final double c;

private final double alpha;

private final double beta;

private final double gamma;

private final double delta;

private final BiFunction<Double, Double, Double> f;

private final ThreeArgFunction<Double, Double, Double, Double> analyticalSolution;

private final BiFunction<Double, Double, Double> boundaryCondition1;

private final BiFunction<Double, Double, Double> boundaryCondition2;

private final Function<Double, Double> initialCondition;

public Solver(double leftBound, double rightBound, double a, int N, int K, double T, double b,

double c, double alpha, double beta, double gamma, double delta,

BiFunction<Double, Double, Double> f,

ThreeArgFunction<Double, Double, Double, Double> analyticalSolution,

BiFunction<Double, Double, Double> boundaryCondition1,

BiFunction<Double, Double, Double> boundaryCondition2,

Function<Double, Double> initialCondition) {

this.leftBound = leftBound;

this.rightBound = rightBound;

this.a = a;

this.N = N;

this.K = K;

this.T = T;

this.b = b;

this.c = c;

this.alpha = alpha;

this.beta = beta;

this.gamma = gamma;

this.delta = delta;

this.f = f;

this.analyticalSolution = analyticalSolution;

this.boundaryCondition1 = boundaryCondition1;

this.boundaryCondition2 = boundaryCondition2;

this.initialCondition = initialCondition;

}

public String answerAnalyticalSolution() {

double tau = T / K;

double h = (rightBound - leftBound) / N;

List<List<Double>> u = new ArrayList<>(K + 1);

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

u.add(new ArrayList<>(Collections.nCopies(N + 1, 0.)));

}

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

double t = k \* tau;

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

double x = leftBound + j \* h;

u.get(k).set(j, analyticalSolution.apply(x, t, a));

}

}

Answer answer = new Answer(leftBound, rightBound, tau, h, T, u);

return gson.toJson(answer);

}

public String explicitScheme(ApproximationType approximationType) {

double tau = T / K;

double h = (rightBound - leftBound) / N;

List<List<Double>> u = new ArrayList<>(K + 1);

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

u.add(new ArrayList<>(Collections.nCopies(N + 1, 0.)));

}

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

double t = k \* tau;

if (k == 0) {

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

double x = leftBound + j \* h;

u.get(k).set(j, initialCondition.apply(x));

}

} else {

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

double sigma = a \* tau / Math.pow(h, 2);

double x = leftBound + j \* h;

double curU = u.get(k - 1).get(j + 1) \* (sigma + b \* tau / 2. / h)

+ u.get(k - 1).get(j) \* (c \* tau + 1 - 2 \* a \* tau / Math.pow(h, 2))

+ u.get(k - 1).get(j - 1) \* (sigma - b \* tau / 2. / h)

+ tau \* f.apply(x, t);

u.get(k).set(j, curU);

}

if (approximationType == ApproximationType.TWO\_POINT\_FIRST\_DEGREE) {

double curU = (boundaryCondition1.apply(t, a) - alpha / h \* u.get(k).get(1))

/ (beta - alpha / h);

u.get(k).set(0, curU);

curU = (boundaryCondition2.apply(t, a) + gamma / h \* u.get(k).get(N - 1))

/ (gamma / h + delta);

u.get(k).set(N, curU);

} else if (approximationType == ApproximationType.TWO\_POINT\_SECOND\_DEGREE) {

double curU = (boundaryCondition1.apply(t, a) - u.get(k).get(1)

\* 2 \* a \* alpha / h / (2 \* a - h \* b) + u.get(k - 1).get(0)

\* h \* alpha / tau / (2 \* a - h \* b) - f.apply(leftBound, t)

\* h \* alpha / (2 \* a - h \* b)) / (beta - 2 \* a \* alpha / h / (2 \* a - h \* b)

- h \* alpha / (2 \* a - h \* b) / tau + c \* h \* alpha / (2 \* a - h \* b));

u.get(k).set(0, curU);

curU = (boundaryCondition2.apply(t, a) + u.get(k).get(N - 1)

\* 2 \* a \* gamma / h / (2 \* a + h \* b) + u.get(k - 1).get(N)

\* h \* gamma / (2 \* a + h \* b) + f.apply(rightBound, t)

\* h \* gamma / (2 \* a + h \* b)) / (delta + 2 \* a \* gamma / h / (2 \* a + h \* b)

+ h \* gamma / tau / (2 \* a + h \* b) - c \* h \* gamma / (2 \* a + h \* b));

u.get(k).set(N, curU);

} else if (approximationType == ApproximationType.THREE\_POINT\_SECOND\_DEGREE) {

double curU = (boundaryCondition1.apply(t, a) + alpha / 2. / h \* u.get(k).get(2)

- 2 \* alpha / h \* u.get(k).get(1)) / (beta - 3 \* alpha / 2. / h);

u.get(k).set(0, curU);

curU = (boundaryCondition2.apply(t, a) + 2 \* gamma / h \* u.get(k).get(N - 1)

- gamma / 2. / h \* u.get(k).get(N - 2)) / (delta + 3 \* gamma / 2. / h);

u.get(k).set(N, curU);

}

}

}

Answer answer = new Answer(leftBound, rightBound, tau, h, T, u);

return gson.toJson(answer);

}

public List<Double> tridiagonalAlgo(List<Double> a, List<Double> b, List<Double> c, List<Double> d) {

int n = a.size();

List<Double> P = new ArrayList<>(Collections.nCopies(n + 1, 0.));

List<Double> Q = new ArrayList<>(Collections.nCopies(n + 1, 0.));

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

P.set(i + 1, -1 \* c.get(i) / (b.get(i) + a.get(i) \* P.get(i)));

Q.set(i + 1, (d.get(i) - a.get(i) \* Q.get(i)) / (b.get(i) + a.get(i) \* P.get(i)));

}

List<Double> x = new ArrayList<>(Collections.nCopies(n, 0.));

for (int i = n - 1; i >= 0; --i) {

if (i == n - 1) {

x.set(i, Q.get(n));

} else {

x.set(i, Q.get(i + 1) + P.get(i + 1) \* x.get(i + 1));

}

}

return x;

}

public String implicitScheme(ApproximationType approximationType) {

double tau = T / K;

double h = (rightBound - leftBound) / N;

double sigma = a \* tau / Math.pow(h, 2);

List<List<Double>> u = new ArrayList<>(K + 1);

// u(j, 0)

u.add(new ArrayList<>(Collections.nCopies(N + 1, 0.)));

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

double x = leftBound + h \* j;

u.get(0).set(j, initialCondition.apply(x));

}

for (int k = 1; k <= K; ++k) {

double t = k \* tau;

List<Double> aCoefficients = new ArrayList<>(Collections.nCopies(N + 1, 0.));

List<Double> bCoefficients = new ArrayList<>(Collections.nCopies(N + 1, 0.));

List<Double> cCoefficients = new ArrayList<>(Collections.nCopies(N + 1, 0.));

List<Double> dCoefficients = new ArrayList<>(Collections.nCopies(N + 1, 0.));

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

aCoefficients.set(j, sigma - b \* tau / 2 / h);

bCoefficients.set(j, c \* tau - (1 + 2 \* sigma));

cCoefficients.set(j, sigma + b \* tau / 2 / h);

dCoefficients.set(j, -1 \* u.get(k - 1).get(j) - tau \* f.apply(leftBound + j \* h, t));

}

if (approximationType == ApproximationType.TWO\_POINT\_FIRST\_DEGREE) {

bCoefficients.set(0, beta - alpha / h);

cCoefficients.set(0, alpha / h);

dCoefficients.set(0, boundaryCondition1.apply(t, a));

aCoefficients.set(N, -1 \* gamma / h);

bCoefficients.set(N, gamma / h + delta);

dCoefficients.set(N, boundaryCondition2.apply(t, a));

} else if (approximationType == ApproximationType.TWO\_POINT\_SECOND\_DEGREE) {

double tmp = beta + alpha \* (Math.pow(h, 2) \* c / 2 / a - Math.pow(h, 2) / 2 / a / tau - 1)

/ h / (1 - h \* b / 2 / a);

bCoefficients.set(0, tmp);

tmp = alpha / h / (1 - h \* b / 2 / a);

cCoefficients.set(0, tmp);

tmp = boundaryCondition1.apply(t, a) + f.apply(leftBound, t) \* h \* alpha / (h \* b - 2 \* a)

+ u.get(k - 1).get(0) \* h \* alpha / tau / (h \* b - 2 \* a);

dCoefficients.set(0, tmp);

tmp = -1 \* gamma / (h + Math.pow(h, 2) \* b / 2 / a);

aCoefficients.set(N, tmp);

tmp = gamma \* (1 + Math.pow(h, 2) / 2 / a / tau - Math.pow(h, 2) \* c / 2 / a)

/ (h + Math.pow(h, 2) \* b / 2 / a) + delta;

bCoefficients.set(N, tmp);

tmp = boundaryCondition2.apply(t, a) + u.get(k - 1).get(N) \* h \* gamma / tau / (2 \* a + h \* b)

+ f.apply(rightBound, t) \* h \* gamma / (2 \* a + h \* b);

dCoefficients.set(N, tmp);

} else if (approximationType == ApproximationType.THREE\_POINT\_SECOND\_DEGREE) {

double tmp = beta - 3 \* alpha / 2 / h + (sigma - b \* tau / 2 / h) \* alpha / (sigma + b \* tau / 2 / h)

/ 2 / h;

bCoefficients.set(0, tmp);

tmp = 2 \* alpha / h + (c \* tau - (1 + 2 \* sigma)) \* alpha / (sigma + b \* tau / 2 / h) / 2 / h;

cCoefficients.set(0, tmp);

tmp = boundaryCondition1.apply(t, a) - (u.get(k - 1).get(1) + tau \* f.apply(leftBound + h, t)) \* alpha

/ (sigma + b \* tau / 2 / h) / 2 / h;

dCoefficients.set(0, tmp);

tmp = -2 \* gamma / h - (c \* tau - (1 + 2 \* sigma)) \* gamma / (sigma - b \* tau / 2 / h) / 2 / h;

aCoefficients.set(N, tmp);

tmp = 3 \* gamma / 2 / h + delta - (sigma + b \* tau / 2 / h) \* gamma / (sigma - b \* tau / 2 / h) / 2 / h;

bCoefficients.set(N, tmp);

tmp = boundaryCondition2.apply(t, a) + gamma \* (u.get(k - 1).get(N - 1) + tau

\* f.apply(rightBound - h, t)) / (sigma - b \* tau / 2 / h) / 2 / h;

dCoefficients.set(N, tmp);

}

u.add(tridiagonalAlgo(aCoefficients, bCoefficients, cCoefficients, dCoefficients));

}

Answer answer = new Answer(leftBound, rightBound, tau, h, T, u);

return gson.toJson(answer);

}

public String crankNicolson(ApproximationType approximationType, double theta) {

double tau = T / K;

double h = (rightBound - leftBound) / N;

List<List<Double>> u = new ArrayList<>(K + 1);

u.add(new ArrayList<>(Collections.nCopies(N + 1, 0.)));

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

double x = h \* j;

u.get(0).set(j, initialCondition.apply(x));

}

for (int k = 1; k <= K; ++k) {

double t = k \* tau;

List<Double> aCoefficients = new ArrayList<>(Collections.nCopies(N + 1, 0.));

List<Double> bCoefficients = new ArrayList<>(Collections.nCopies(N + 1, 0.));

List<Double> cCoefficients = new ArrayList<>(Collections.nCopies(N + 1, 0.));

List<Double> dCoefficients = new ArrayList<>(Collections.nCopies(N + 1, 0.));

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

aCoefficients.set(j, a \* theta \* tau / Math.pow(h, 2) - b \* theta \* tau / 2 / h);

bCoefficients.set(j, c \* theta \* tau - 2 \* a \* theta \* tau / Math.pow(h, 2) - 1);

cCoefficients.set(j, a \* theta \* tau / Math.pow(h, 2) + b \* theta \* tau / 2 / h);

dCoefficients.set(j, u.get(k - 1).get(j + 1) \* (-1 \* a \* (1 - theta) \* tau / Math.pow(h, 2)

- b \* (1 - theta) \* tau / 2 / h) + u.get(k - 1).get(j)

\* (2 \* a \* (1 - theta) \* tau / Math.pow(h, 2) - c \* (1 - theta) \* tau - 1)

+ u.get(k - 1).get(j - 1) \* (b \* (1 - theta) \* tau / 2 / h

- a \* (1 - theta) \* tau / Math.pow(h, 2)) - tau \* f.apply(j \* h, t));

}

if (approximationType == ApproximationType.TWO\_POINT\_FIRST\_DEGREE) {

bCoefficients.set(0, beta - alpha / h);

cCoefficients.set(0, alpha / h);

dCoefficients.set(0, boundaryCondition1.apply(t, a));

aCoefficients.set(N, -1 \* gamma / h);

bCoefficients.set(N, gamma / h + delta);

dCoefficients.set(N, boundaryCondition2.apply(t, a));

} else if (approximationType == ApproximationType.TWO\_POINT\_SECOND\_DEGREE) {

double tmp = beta + alpha \* (Math.pow(h, 2) \* c / 2 / a - Math.pow(h, 2) / 2 / a / tau - 1)

/ h / (1 - h \* b / 2 / a);

bCoefficients.set(0, tmp);

tmp = alpha / h / (1 - h \* b / 2 / a);

cCoefficients.set(0, tmp);

tmp = boundaryCondition1.apply(t, a) + f.apply(leftBound, t) \* h \* alpha / (h \* b - 2 \* a)

+ u.get(k - 1).get(0) \* h \* alpha / tau / (h \* b - 2 \* a);

dCoefficients.set(0, tmp);

tmp = -1 \* gamma / (h + Math.pow(h, 2) \* b / 2 / a);

aCoefficients.set(N, tmp);

tmp = gamma \* (1 + Math.pow(h, 2) / 2 / a / tau - Math.pow(h, 2) \* c / 2 / a)

/ (h + Math.pow(h, 2) \* b / 2 / a) + delta;

bCoefficients.set(N, tmp);

tmp = boundaryCondition2.apply(t, a) + u.get(k - 1).get(N) \* h \* gamma / tau / (2 \* a + h \* b)

+ f.apply(rightBound, t) \* h \* gamma / (2 \* a + h \* b);

dCoefficients.set(N, tmp);

} else if (approximationType == ApproximationType.THREE\_POINT\_SECOND\_DEGREE) {

double sigma = a \* tau / Math.pow(h, 2);

double tmp = beta - 3 \* alpha / 2 / h + (sigma - b \* tau / 2 / h) \* alpha / (sigma + b \* tau / 2 / h)

/ 2 / h;

bCoefficients.set(0, tmp);

tmp = 2 \* alpha / h + (c \* tau - (1 + 2 \* sigma)) \* alpha / (sigma + b \* tau / 2 / h) / 2 / h;

cCoefficients.set(0, tmp);

tmp = boundaryCondition1.apply(t, a) - (u.get(k - 1).get(1) + tau \* f.apply(leftBound + h, t)) \* alpha

/ (sigma + b \* tau / 2 / h) / 2 / h;

dCoefficients.set(0, tmp);

tmp = -2 \* gamma / h - (c \* tau - (1 + 2 \* sigma)) \* gamma / (sigma - b \* tau / 2 / h) / 2 / h;

aCoefficients.set(N, tmp);

tmp = 3 \* gamma / 2 / h + delta - (sigma + b \* tau / 2 / h) \* gamma / (sigma - b \* tau / 2 / h) / 2 / h;

bCoefficients.set(N, tmp);

tmp = boundaryCondition2.apply(t, a) + gamma \* (u.get(k - 1).get(N - 1) + tau

\* f.apply(rightBound - h, t)) / (sigma - b \* tau / 2 / h) / 2 / h;

dCoefficients.set(N, tmp);

}

u.add(tridiagonalAlgo(aCoefficients, bCoefficients, cCoefficients, dCoefficients));

}

Answer answer = new Answer(leftBound, rightBound, tau, h, T, u);

return gson.toJson(answer);

}

class Answer {

private final double leftBound;

private final double rightBound;

private final double tau;

private final double h;

private final double T;

private final List<List<Double>> u;

public Answer(double leftBound, double rightBound, double tau, double h, double t,

List<List<Double>> u) {

this.leftBound = leftBound;

this.rightBound = rightBound;

this.tau = tau;

this.h = h;

T = t;

this.u = u;

}

}

enum ApproximationType {

TWO\_POINT\_FIRST\_DEGREE,

TWO\_POINT\_SECOND\_DEGREE,

THREE\_POINT\_SECOND\_DEGREE

}

}

**Результаты**

Результаты приведены при следующих параметрах:

* Шаг по времени = 0,1
* Шаг по x =
* T = 10
* = 0,4

Результат на последнем слое для аналитического решения:

[

0.01831563888873418,

0.017419207715239565,

0.014817663123820627,

0.010765662425112451,

0.005659843679453539,

1.121509426971355E-18,

-0.005659843679453536,

-0.01076566242511245,

-0.014817663123820626,

-0.017419207715239565,

-0.01831563888873418

]

Результат на последнем слое для явной схемы:

[

0.01831563888873418,

0.01737763988632814,

0.014758385807740533,

0.010711284703603071,

0.0056278862439745145,

3.903127820947816E-18,

-0.005627886243974508,

-0.010711284703603064,

-0.014758385807740528,

-0.017377639886328136,

-0.01831563888873418

]

Результат на последнем слое для неявной схемы:

[

0.01831563888873418,

0.017523032288886315,

0.014965917574055711,

0.010901787845869197,

0.005739886943233832,

2.0166160408230382E-17,

-0.005739886943233794,

-0.01090178784586916,

-0.014965917574055684,

-0.017523032288886298,

-0.01831563888873418

]

Результат на последнем слое для схемы Кранка-Николсона:

[

0.01831563888873418,

0.017448435424950576,

0.014859370057035678,

0.010803939273094492,

0.005682344689469386,

-6.505213034913027E-19,

-0.005682344689469388,

-0.010803939273094495,

-0.014859370057035678,

-0.01744843542495058,

-0.01831563888873418

]

Графики

