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 $U(x,t)$. Исследовать зависимость погрешности от сеточных параметров τ, h.

Вариант 3.

$$\frac{\partial u}{\partial t} = a\frac{\partial^2 u}{\partial x^2},\ a>0,$$

$$u(0,t) = \exp(-at),$$
$$u(\pi,t) = -\exp(-at),$$

$$u(x,0) = \cos x.$$

Аналитическое решение: $U(x,t) = \exp(-at)\cos x$.

Теория

5.1.1. Постановка задач для уравнений параболического типа

Классическим примером уравнения параболического типа является уравнение теплопроводности (диффузии). В одномерном по пространству случае однородное (без источников энергии) уравнение теплопроводности имеет вид

$$\frac{\partial u}{\partial t} = a^2\frac{\partial^2 u}{\partial x^2},\quad 0<x<l,\ t>0. \qquad (5.1)$$

Если на границах $x=0$ и $x=l$ заданы значения искомой функции $u(x,t)$ *в виде*

$$u(0,t) = \varphi_0(t),\quad x=0,\quad t>0;$$
$$u(l,t) = \varphi_l(t),\quad x=l,\quad t>0, \qquad (5.2)$$

т.е. *граничные условия первого рода*, и, кроме того, заданы начальные условия

$$u(x,0) = \psi(x),\quad 0\le x\le l,\quad t=0, \qquad (5.4)$$

то задачу (5.1)-(5.4) называют *первой начально-краевой задачей для уравнения теплопроводности* (5.1).

В терминах теории теплообмена $u(x,t)$ – распределение температуры в пространственно-временной области $\Omega\times T = \{0\le x\le l;\ 0\le t\le T\}$, a^2 – коэффициент температуропроводности, а (5.2), (5.3) с помощью функций $\varphi_0(t)$, $\varphi_l(t)$ задают температуру на границах $x=0$ *и* $x=l$.

Если на границах $x=0$ и $x=l$ заданы значения производных искомой функции по пространственной переменной

$$\frac{\partial u(0,t)}{\partial x} = \varphi_0(t), \quad x = 0, \quad t > 0; \tag{5.5}$$

$$\frac{\partial u(l,t)}{\partial x} = \varphi_l(t), \quad x = l, \quad t > 0, \tag{5.6}$$

т.е. *граничные условия второго рода*, то задачу (5.1), (5.5), (5.6), (5.4) называют второй начально-краевой задачей для уравнения теплопроводности (5.1). В терминах теории теплообмена на границах в этом случае заданы тепловые потоки.

Если на границах заданы линейные комбинации искомой функции и ее производной по пространственной переменной

$$\alpha \frac{\partial u(0,t)}{\partial x} + \beta u(0,t) = \varphi_0(t), \quad x = 0, \quad t > 0; \tag{5.7}$$

$$\gamma \frac{\partial u(l,t)}{\partial x} + \delta u(l,t) = \varphi_l(t), \quad x = l, \quad t > 0, \tag{5.8}$$

т.е. *граничные условия третьего рода*, то задачу (5.1), (5.7), (5.8), (5.4) называют третьей начально-краевой задачей для уравнения теплопроводности (5.1). В терминах теории теплообмена граничные условия (5.7), (5.8) задают теплообмен между газообразной или жидкой средой и границами расчетной области с неизвестными температурами *u(0,t), u(l,t)*.

Для пространственных задач теплопроводности в области $\overline{\Omega} = \Omega + \Gamma$ первая начально-краевая задача имеет вид

$$\begin{cases} \dfrac{\partial u}{\partial t} = a^2\left(\dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2} + \dfrac{\partial^2 u}{\partial z^2}\right), \quad M(x,y,z) \in \Omega, \quad t > 0; & (5.9) \\ u(M,t) = \varphi(M,t), \quad M(x,y,z) \in \Gamma, \quad t > 0; & (5.10) \\ u(M,0) = \psi(M), \quad M(x,y,z) \in \overline{\Omega}, \quad t > 0. & (5.11) \end{cases}$$

Аналогично ставится вторая и третья начально-краевые задачи для пространственного уравнения задачи (5.9) – (5.11).

На практике часто ставятся начально-краевые задачи теплопроводности со смешанными краевыми условиями, когда на границах задаются граничные условия различных родов.

5.1.2. Понятие о методе конечных разностей. Применение метода конечных разностей к решению уравнений параболического типа

Основные определения, связанные с методом конечных разностей, рассмотрим на примере конечно-разностного решения первой начально-краевой задачи для уравнения теплопроводности (5.1)-(5.4).

Нанесем на пространственно-временную область $0 \le x \le l$, $0 \le t \le T$ конечно-разностную сетку $\omega_{h\tau}$

$$\omega_{h\tau} = \{x_j = jh,\ \ j = \overline{0,N};\ \ t^k = k\tau,\ \ k = \overline{0,K}\} \tag{5.12}$$

с пространственным шагом $h=l/N$ и шагом по времени $\tau=T/K$ (рис 5.1).

Введем два *временных слоя*: нижний $t^k = k\,\tau$, на котором распределение искомой функции $u(x_j, t^k)$, $j = \overline{0,N}$ известно (при $k=0$ распределение определяется начальным условием (6.4) $u(x_j, t^0) = \psi(x_j)$) и верхний временной слой $t^{k+1} = (k+1)\,\tau$, на котором распределение искомой функции $u(x_j, t^{k+1})$, $j = \overline{0,N}$ подлежит определению.

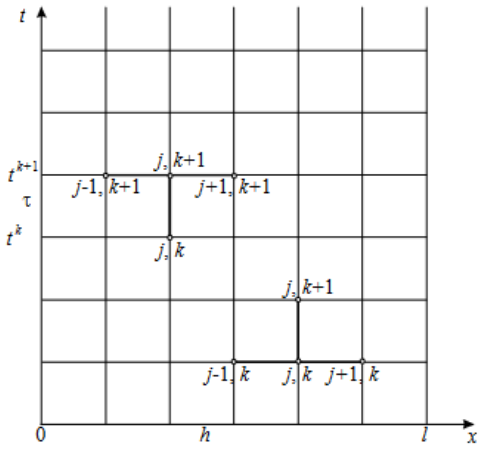


Рис. 5.1. Конечно-разностная сетка

Сеточной функцией задачи (5.1)-(5.4) (обозначение u_j^k) назовем однозначное отображение *целых* аргументов $j,\ k$ в значения функции $u_j^k = u(x_j, t^k)$.

На введенной сетке (5.12) введем сеточные функции u_j^k, u_j^{k+1}, первая из которых известна, вторая – подлежит определению. Для ее определения в задаче (5.1)-(5.4) заменим (аппроксимируем) дифференциальные операторы отношением конечных разностей (см. раздел «Численное дифференцирование»), получим

$$\left.\frac{\partial u}{\partial t}\right|_j^k = \frac{u_j^{k+1} - u_j^k}{\tau} + O(\tau), \tag{5.13}$$

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_j^k = \frac{u_{j+1}^k - 2u_j^k + u_{j-1}^k}{h^2} + O(h^2). \tag{5.14}$$

Подставляя (5.13), (5.14) в задачу (5.1)-(5.4), получим *явную конечно-разностную схему* для этой задачи в форме

$$\frac{u_j^{k+1}-u_j^k}{\tau}=a^2\frac{u_{j+1}^k-2u_j^k+u_{j-1}^k}{h^2}+O(\tau+h^2),\quad j=\overline{1,N-1},\quad k=\overline{0,K-1},$$

$$u_0^k=\varphi_0(t^k),\quad u_N^k=\varphi_l(t^k),\quad k=\overline{0,K};\quad u_j^0=\psi(x_j),\quad j=\overline{0,N},\qquad (5.15)$$

где для каждого j-го уравнения все значения сеточной функции известны, за исключением одного - u_j^{k+1}, которое может быть определено *явно* из соотношений (5.15).

В соотношения (5.15) краевые условия ($j=0$, $j=N$) входят при значениях *j=1* и *j=N-1*, а начальное условие – при *k=0*.

Если в (5.14) дифференциальный оператор по пространственной переменной аппроксимировать отношением конечных разностей на верхнем временном слое

Рис. 5.2. Шаблоны явной и неявной конечно-разностных схем для уравне

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_j^{k+1}=\frac{u_{j+1}^{k+1}-2u_j^{k+1}+u_{j-1}^{k+1}}{h^2}+O(h^2),\qquad (5.16)$$

то после подстановки (5.13), (5.16) в задачу (5.1)-(5.4), получим *неявную конечно-разностную схему* для этой задачи

$$\frac{u_j^{k+1}-u_j^k}{\tau}=a^2\frac{u_{j+1}^{k+1}-2u_j^{k+1}+u_{j-1}^{k+1}}{h^2}+O(\tau+h^2),\quad j=\overline{1,N-1},\quad k=\overline{0,K-1},$$

$$u_0^{k+1}=\varphi_0(t^{k+1}),\quad u_N^{k+1}=\varphi_l(t^{k+1}),\quad k=\overline{0,K-1};\quad u_j^0=\psi(x_j),\quad j=\overline{0,N}.$$

(5.17)

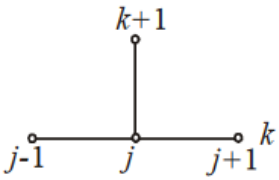
Теперь сеточную функцию u_j^{k+1} на верхнем временном слое можно получить из решения СЛАУ (5.17) с трехдиагональной матрицей. Эта СЛАУ в форме, пригодной для использования метода прогонки, имеет вид

$$a_1=0;\quad c_{N-1}=0;\quad \begin{cases} b_1u_1^{k+1}+c_1u_2^{k+1}=d_1, & j=1\\ a_ju_{j-1}^{k+1}+b_ju_j^{k+1}+c_ju_{j+1}^{k+1}=d_j, & j=\overline{2,N-2}\\ a_{N-1}u_{N-2}^{k+1}+b_{N-1}u_{N-1}^{k+1}=d_{N-1}, & j=N-1, \end{cases}$$

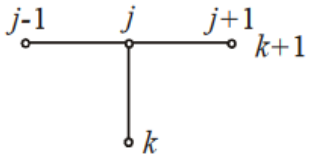
где $a_j=\sigma, j=\overline{2,N-1};\quad b_j=-(1+2\sigma),\quad j=\overline{1,N-1};\quad c_j=\sigma,\quad j=\overline{1,N-2};$

$$d_j=-u_j^k,\quad j=\overline{2,N-2};\quad d_1=-(u_1^k+\sigma\varphi_0(t^{k+1}));\quad d_{N-1}=-(u_{N-1}^k+\sigma\varphi_l(t^{k+1}));\quad \sigma=\frac{a^2\tau}{h^2}.$$

Шаблоном конечно-разностной схемы называют ее геометрическую интерпретацию



шаблон явной схемы



шаблон неявной схемы

на конечно-разностной сетке.

На рисунке 5.2 приведены шаблоны для явной (5.15) и неявной (5.17) конечно-разностных схем при аппроксимации задачи (5.1)-(5.4).

Явная конечно-разностная схема (5.15), записанная в форме

$$u_j^{k+1} = \sigma \cdot u_{j+1}^k + (1-2\sigma)u_j^k + \sigma \cdot u_{j-1}^k,\ \sigma = \frac{a^2\tau}{h^2},\quad j = \overline{1, N-1},\quad k = 0, 1, 2 \ldots, \qquad (5.18)$$

обладает тем *достоинством*, что решение на верхнем временном слое t^{k+1} получается сразу (без решения СЛАУ) по значениям сеточных функций на нижнем временном слое t^k, где решение известно (при k=0 значения сеточной функции формируются из начального условия (5.4.)). Но эта же схема обладает существенным *недостатком*, поскольку она является условно устойчивой с условием $\sigma = \frac{a^2\tau}{h^2} \le \frac{1}{2}$, накладываем на сеточные характеристики τ и h.

С другой стороны, неявная конечно-разностная схема (5.17), записанная форме

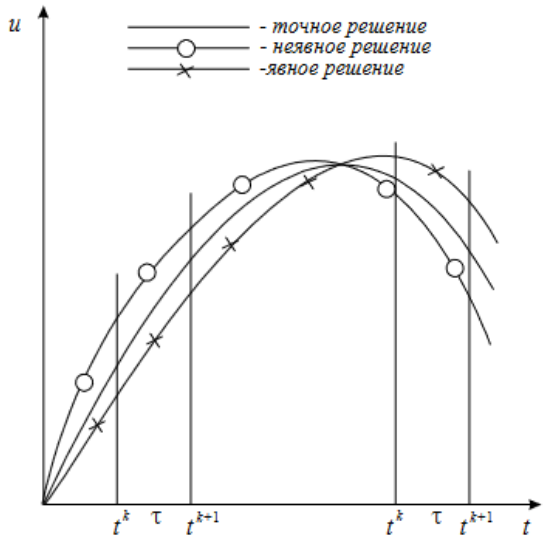
$$a_j u_{j-1}^{k+1} + b_j u_j^{k+1} + c_j u_{j+1}^{k+1} = d_j,\qquad j = \overline{1, N-1},\quad \mathrm{k} = 0, 1, 2, \ldots, \qquad (5.19)$$

приводит к необходимости решать СЛАУ, но зато эта схема абсолютно устойчива.

Проанализируем схемы (5.18), (5.19). Пусть точное решение, которое не известно, возрастает по времени, т.е. $u_j^{k+1} > u_j^k$. Тогда, в соответствии с явной схемой (5.18) разностное решение будет заниженным по сравнению с точным, т.к. u_j^{k+1} определяется по меньшим значениям сеточной функции на предыдущем временном слое, поскольку решение является возрастающим по времени.

Для неявной схемы (5.19) на возрастающем решении, наоборот, решение завышено по сравнению с точным, поскольку оно определяется по значениям сеточной функции на верхнем временном слое.

На убывающем решении картина изменяется противоположным образом: явная конечно-разностная схема завышает решения, а неявная - занижает (см. рис. 5.3)



На основе этого анализа возникла идея о построении более точной неявно-явной конечно-разностной схемы с весами при пространственных конечно-разностных операторах, причем при измельчении шагов τ и h точное (неизвестное) решение может быть взято в "вилку" сколь угодно узкую, т.к. если явная и неявная схемы аппроксимируют дифференциальную задачу и эти схемы устойчивы, то при стремлении сеточных характеристик τ и h к нулю, решения по явной и неявной схемам стремятся к точному решению с разных сторон.

Рассмотрим неявно-явную схему с весами для простейшего уравнения теплопроводности

$$\frac{u_j^{k+1}-u_j^k}{\tau}=\theta a^2\frac{u_{j+1}^{k+1}-2u_j^{k+1}+u_{j-1}^{k+1}}{h^2}+(1-\theta)a^2\frac{u_{j+1}^{k}-2u_j^{k}+u_{j-1}^{k}}{h^2}, \qquad (5.20)$$

где θ - вес неявной части конечно-разностной схемы, 1–θ - вес для явной части, причем 0≤θ≤1. При θ=1 имеем полностью неявную схему, при θ=0 - полностью явную схему, и при θ=1/2 - схему *Кранка-Николсона.*

Для схемы Кранка-Николсона (θ=1/2) порядок аппроксимации составляет $O\left(\tau^2+h^2\right)$, т.е. на один порядок по времени выше, чем обычные явная или неявная схемы.

Неявно-явная схема с весами (5.20) абсолютно устойчива при 1/2≤θ≤1 и условно устойчива с условием $\sigma\le\frac{1}{2}$ при 0≤θ<1/2.

Таким образом, схема Кранка-Николсона (5.20) при θ=1/2 *абсолютно устойчива* и *имеет второй порядок аппроксимации по времени и пространственной переменной x.*

5.1.3. Аппроксимация граничных условий, содержащих производные

В задачах математической физики вообще, и в задачах теплопроводности в частности, граничные условия 1-го рода аппроксимируются точно в узлах на границе расчетной области. Граничные условия 2-го и 3-го рода отличаются тем, что в них присутствует производная первого порядка искомой функции по пространственной переменной. Поэтому для замыкания конечно-разностной схемы необходима их аппроксимация. Простейшим вариантом является аппроксимация производных направленными разностями первого порядка:

$$\left.\frac{\partial u}{\partial x}\right|_{j=0}^{k+1} = \frac{u_1^{k+1} - u_0^{k+1}}{h} + O(h); \qquad \left.\frac{\partial u}{\partial x}\right|_{j=N}^{k+1} = \frac{u_N^{k+1} - u_{N-1}^{k+1}}{h} + O(h),$$

Тогда в общем случае граничных условий 3-го рода (5.7), (5.8) уравнения, связывающие значения искомой функции в двух крайних узлах разностной сетки, выглядят следующим образом:

$$\alpha \frac{u_1^{k+1} - u_0^{k+1}}{h} + \beta u_0^{k+1} = \varphi_0(t^{k+1}) + O(h),$$

$$\gamma \frac{u_N^{k+1} - u_{N-1}^{k+1}}{h} + \delta u_N^{k+1} = \varphi_l(t^{k+1}) + O(h).$$

Дополняя полученными уравнениями явную конечно-разностную аппроксимацию во внутренних узлах, получим явную разностную схему для третьей начально-краевой задачи (5.1), (5.4), (5.7), (5.8).

$$\alpha \frac{u_1^{k+1} - u_0^{k+1}}{h} + \beta u_0^{k+1} = \varphi_0(t^{k+1}),$$

$$\frac{u_j^{k+1} - u_j^k}{\tau} = a^2 \frac{u_{j+1}^k - 2u_j^k + u_{j-1}^k}{h^2}, \quad j = \overline{1, N-1}, \quad k = \overline{0, K-1},$$

$$\gamma \frac{u_N^{k+1} - u_{N-1}^{k+1}}{h} + \delta u_N^{k+1} = \varphi_l(t^{k+1}).$$

В результате алгоритм перехода на новый временной слой t^{k+1} с использованием явной схемы можно представить в следующем виде:

$$u_j^{k+1} = \sigma \cdot u_{j+1}^k + (1 - 2\sigma) u_j^k + \sigma \cdot u_{j-1}^k, \ \sigma = \frac{a^2 \tau}{h^2}, \quad j = \overline{1, N-1},$$

$$u_0^{k+1} = -\frac{\alpha / h}{\beta - \alpha / h} u_1^{k+1} + \frac{\varphi_0(t^{k+1})}{\beta - \alpha / h},$$

$$u_N^{k+1} = \frac{\gamma / h}{\delta + \gamma / h} u_{N-1}^{k+1} + \frac{\varphi_l(t^{k+1})}{\delta + \gamma / h}.$$

Т.е. сначала рассчитываются значения искомой функции во всех внутренних узлах на новом временном слое, а затем определяются значения на границах.

При использовании неявной конечно-разностной схемы получаем следующий разностный аналог дифференциальной задачи:

$$b_0 u_0^{k+1} + c_0 u_1^{k+1} = d_0,$$

$$a_j u_{j-1}^{k+1} + b_j u_j^{k+1} + c_j u_{j+1}^{k+1} = d_j, \qquad j = \overline{1, N-1},$$

$$a_N u_{N-1}^{k+1} + b_N u_N^{k+1} = d_N,$$

где $b_0 = \beta - \alpha / h,\ c_0 = \alpha / h,\ d_0 = \dfrac{\varphi_0(t^{k+1})}{\beta - \alpha / h},$

$$a_N = -\gamma / h,\ b_N = \delta + \gamma / h,\ d_N = \frac{\varphi_l(t^{k+1})}{\delta + \gamma / h},$$

$$a_j = \sigma, \quad b_j = -(1+2\sigma), \quad c_j = \sigma, \quad d_j = -u_j^k, \quad j = \overline{1, N-1}, \quad \sigma = \frac{a^2 \tau}{h^2}.$$

В результате для получения решения на новом временном слое t^{k+1} решается система линейных алгебраических уравнений с трехдиагональной матрицей. Аналогичная картина имеет место и при использовании неявно-явной схемы с весами.

Принципиальной особенностью рассмотренного выше подхода является первый порядок аппроксимации граничных условий. Т.е. порядок аппроксимации в граничных узлах ниже порядка аппроксимации во внутренних узлах расчетной области. При этом глобальный порядок аппроксимации (во всей расчетной области) равен *наименьшему* относительно всех узлов сетки порядку аппроксимации.

Одним из способов повышения порядка аппроксимации граничных условий является использование формул численного дифференцирования второго порядка:

$$\frac{\partial u}{\partial x}(0, t^{k+1}) = \frac{-3u_0^{k+1} + 4u_1^{k+1} - u_2^{k+1}}{2h} + O(h^2),$$

$$\frac{\partial u}{\partial x}(l, t^{k+1}) = \frac{u_{N-2}^{k+1} - 4u_{N-1}^{k+1} + 3u_N^{k+1}}{2h} + O(h^2).$$

В случае явной схемы алгоритм вычисления решения на новом временном слое при такой аппроксимации граничных условий не приобретает принципиальных изменений. Если же используется неявная схема, то получающаяся при этом СЛАУ теряет трехдиагональный вид (первое и последнее уравнение содержат три неизвестных). Этот недостаток легко устраним, т.к. путем линейной комбинации первого уравнения со вторым (последнего с предпоследним) можно добиться исключения третьего неизвестного из соответствующего уравнения. Однако при этом возможно нарушение диагонального преобладания матрицы и, следовательно, нарушение условий применимости метода прогонки.

Более эффективным является подход, позволяющий повысить порядок аппроксимации граничных условий без увеличения числа узлов в аппроксимационных соотношениях. Для иллюстрации этого подхода рассмотрим следующий пример.

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 τ = 0,1
- Шаг по x $h = \frac{\pi}{10}$
- T = 10
- a = 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

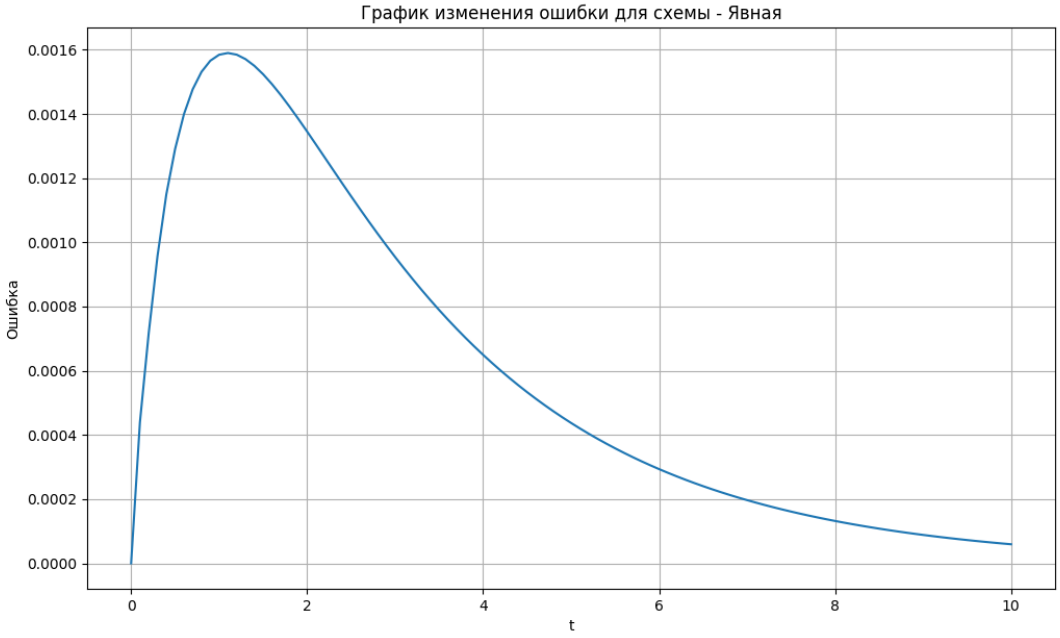


График изменения ошибки для схемы - Явная
Ошибка
0.0016
0.0014
0.0012
0.0010
0.0008
0.0006
0.0004
0.0002
0.0000
0
2
4
6
8
10
t

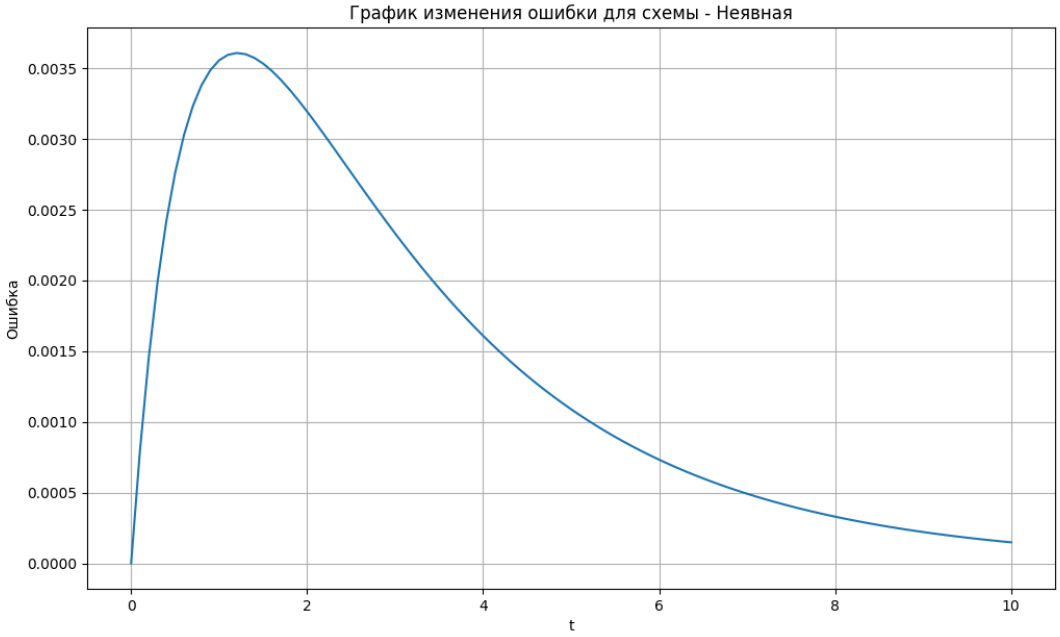


График изменения ошибки для схемы - Неявная
Ошибка
0.0035
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
0
2
4
6
8
10
t

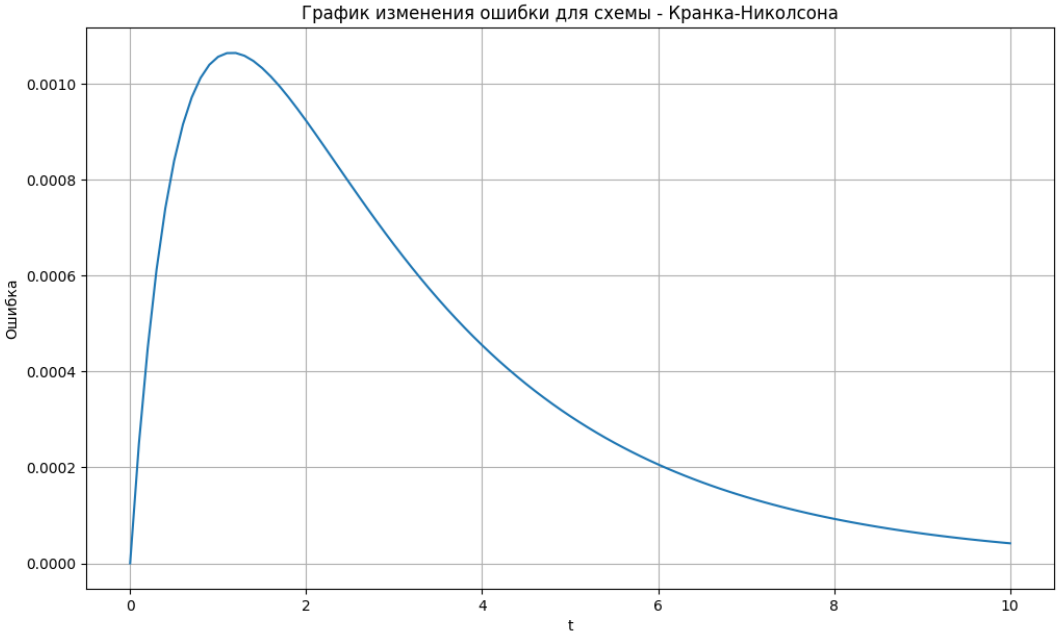
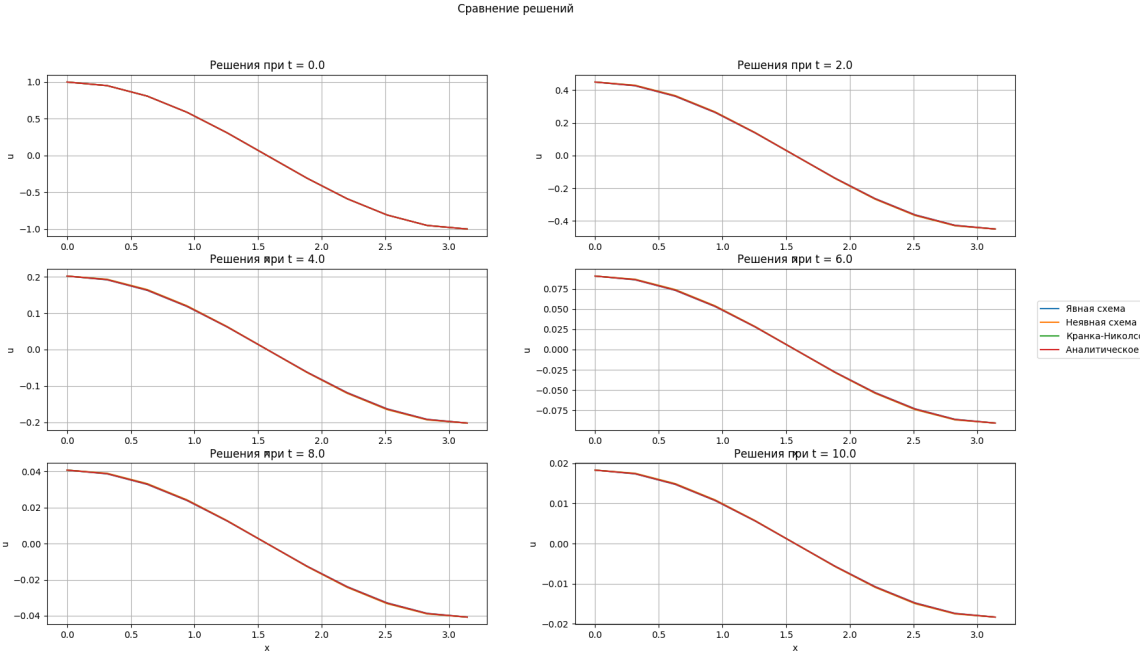
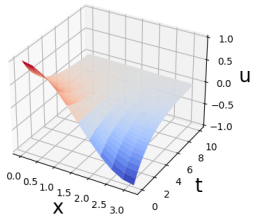


График изменения ошибки для схемы - Кранка-Николсона
Ошибка
t
0.0000
0.0002
0.0004
0.0006
0.0008
0.0010
0
2
4
6
8
10


Сравнение решений
Решения при t = 0.0
Решения при t = 2.0
Решения при t = 4.0
Решения при t = 6.0
Решения при t = 8.0
Решения при t = 10.0
u
x
Явная схема
Неявная схема
Кранка-Николсо
Аналитическое

Явная схема



Неявная схема

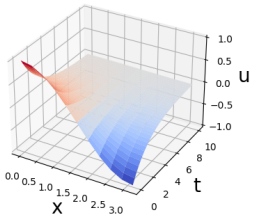
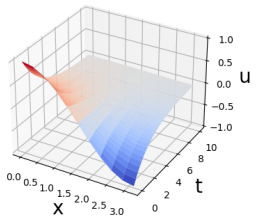


Схема Кранка-Николсона



Аналитическое решение