

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Российский химико-технологический университет имени Д.И.  
Менделеева»

## ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4

### Вариант 9

Выполнил студент группы КС-36: Золотухин Андрей Александрович  
Ссылка на репозиторий: [https://github.com/  
CorgiPuppy/  
num-methods-eq-math-phys-chem-labs](https://github.com/CorgiPuppy/num-methods-eq-math-phys-chem-labs)  
Принял: Лебедев Данила Александрович  
Дата сдачи: 16.05.2025

Москва  
2025

# Оглавление

Описание задачи . . . . .	1
Выполнение задачи . . . . .	2
Задание 1 . . . . .	2
Задание 2 . . . . .	2
Задание 3 . . . . .	2
Задание 4 . . . . .	2
Задание 5 . . . . .	3
Задание 6 . . . . .	3
Задание 7 . . . . .	3
Задание 8 . . . . .	5

## Описание задачи

Вариант	Уравнение	Метод	Граничные условия
9	$\frac{du}{dx} = \frac{d^2u}{dx^2} + 4$	Установление со схемой Кранка-Николсона	$\begin{cases} u(x=0) = 1 \\ u(x=1) = 6.7 \end{cases}$

Для заданного уравнения:

1. представить задачу в нестационарном виде;
2. записать разностную схему Кранка-Николсона;
3. привести схему к виду, удобному для использования метода прогонки;
4. проверить сходимость прогонки;
5. найти  $\alpha_1$ ,  $\beta_1$ ,  $u_N^{n+1}$ ;
6. записать рекуррентное прогоночное соотношение;
7. составить алгоритм (блок-схему) расчёта;
8. построить программу на любом удобном языке программирования;
9. построить численный расчёт с использованием различных значений  $\Delta t = \{0.1; 0.01; 0.001\}$ ,  $h = \{0.1; 0.01\}$ ;
10. Сравнить результаты вычислений между собой в точках:  $x = \{0; 0.1; 0.2; 0.3; 0.4; 0.5; 0.6; 0.7; 0.8; 0.9; 1\}$ .

# Выполнение задачи

## Задание 1

Представить задачу в нестационарном виде:

Представлю стационарную задачу в нестационарном виде. Для этого в уравнение необходимо добавить фиктивную производную по времени:

$$\frac{du}{dx} = \frac{d^2u}{dx^2} + 4 \rightarrow \frac{\partial \tilde{u}}{\partial \tau} + \frac{\partial \tilde{u}}{\partial x} = \frac{\partial^2 \tilde{u}}{\partial x^2} + 4. \quad (1)$$

При этом искомая функция станет уже функцией двух переменных:

$$u(x) \rightarrow \tilde{u}(x, \tau).$$

## Задание 2

Записать разностную схему Кранка-Николсона для уравнения (1):

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} + \frac{1}{2} \frac{u_j^{n+1} - u_{j-1}^{n+1}}{h} + \frac{1}{2} \frac{u_j^n - u_{j-1}^n}{h} = \frac{1}{2} \frac{u_{j+1}^{n+1} - u_j^{n+1} + u_j^{n+1} - u_{j-1}^{n+1}}{h^2} + \frac{1}{2} \frac{u_{j+1}^n - u_j^n + u_j^n - u_{j-1}^n}{h^2} + 4. \quad (2)$$

## Задание 3

Привести схему к виду, удобному для использования метода прогонки:

$$-\frac{1}{2} \frac{\Delta t}{h^2} u_{j+1}^{n+1} + (1 + \frac{1}{2} \frac{\Delta t}{h} + \frac{\Delta t}{h^2}) u_j^{n+1} - (\frac{1}{2} \frac{\Delta t}{h} + \frac{1}{2} \frac{\Delta t}{h^2}) u_{j-1}^{n+1} = u_j^n + \frac{1}{2} \frac{\Delta t}{h^2} (u_{j+1}^n - 2u_j^n + u_{j-1}^n) - \frac{1}{2} \frac{\Delta t}{h} (u_j^n - u_{j-1}^n) + \Delta t 4.$$

Введу следующие обозначения:

$$a_j = -\frac{1}{2} \frac{\Delta t}{h^2}, \quad b_j = 1 + \frac{1}{2} \frac{\Delta t}{h} + \frac{\Delta t}{h^2}, \quad c_j = -(\frac{1}{2} \frac{\Delta t}{h} + \frac{1}{2} \frac{\Delta t}{h^2}), \\ \xi_j^n = u_j^n + \frac{1}{2} \frac{\Delta t}{h^2} (u_{j+1}^n - 2u_j^n + u_{j-1}^n) - \frac{1}{2} \frac{\Delta t}{h} (u_j^n - u_{j-1}^n) + \Delta t 4.$$

С учётом обозначений равенство будет иметь вид:

$$\alpha_j u_{j+1}^{n+1} + b_j u_j^{n+1} + c_j u_{j-1}^{n+1} = \xi_j^n.$$

Данное преобразование называется *преобразованием схемы Кранка-Николсона к виду, удобному для использования метода прогонки*.

## Задание 4

Проверить сходимость прогонки:

Легко видеть, что для разностной схемы (2) достаточное условие сходимости прогонки выполняется:

$$|a_j| + |c_j| = \frac{1}{2} \frac{\Delta t}{h} + \frac{\Delta t}{h^2} < 1 + \frac{1}{2} \frac{\Delta t}{h} + \frac{\Delta t}{h^2} = |b_j|.$$

## Задание 5

Найти  $\alpha_1, \beta_1, u_N^{n+1}$ :

Для реализации разностной схемы Кранка-Николсона требуется ввести некоторое дополнительное условие, связывающее значения функции  $u(t, x)$  на  $(n+1)$ -м шаге по времени. Представлю это дополнительное условие в виде линейной зависимости

$$u_j^{n+1} = \alpha_j u_{j+1}^{n+1} + \beta_j, \quad (3)$$

справедливой для любого из значений  $j = 1..N - 1$ .

Соотношение (3) называют **рекуррентным прогоночным соотношением**, а коэффициенты  $\alpha_j, \beta_j$  - **прогоночными коэффициентами**.

Для определения прогоночных коэффициентов на 1-м шаге по координате  $x$ , используя рекуррентное прогоночное соотношение (3), записанное для  $j = 1$ :

$$u_1^{n+1} = \alpha_1 u_2^{n+1} + \beta_1$$

и левое граничное условие:

$$u_1^{n+1} = 1.$$

Сравнивая эти два соотношения, получаю:

$$\alpha_1 = 0, \beta_1 = 1.$$

Значение функции  $u(t, x)$  на  $(n+1)$ -м шаге по времени в крайней правой точке, которое можно определить из правого граничного условия:

$$u_N^{n+1} = 6.7.$$

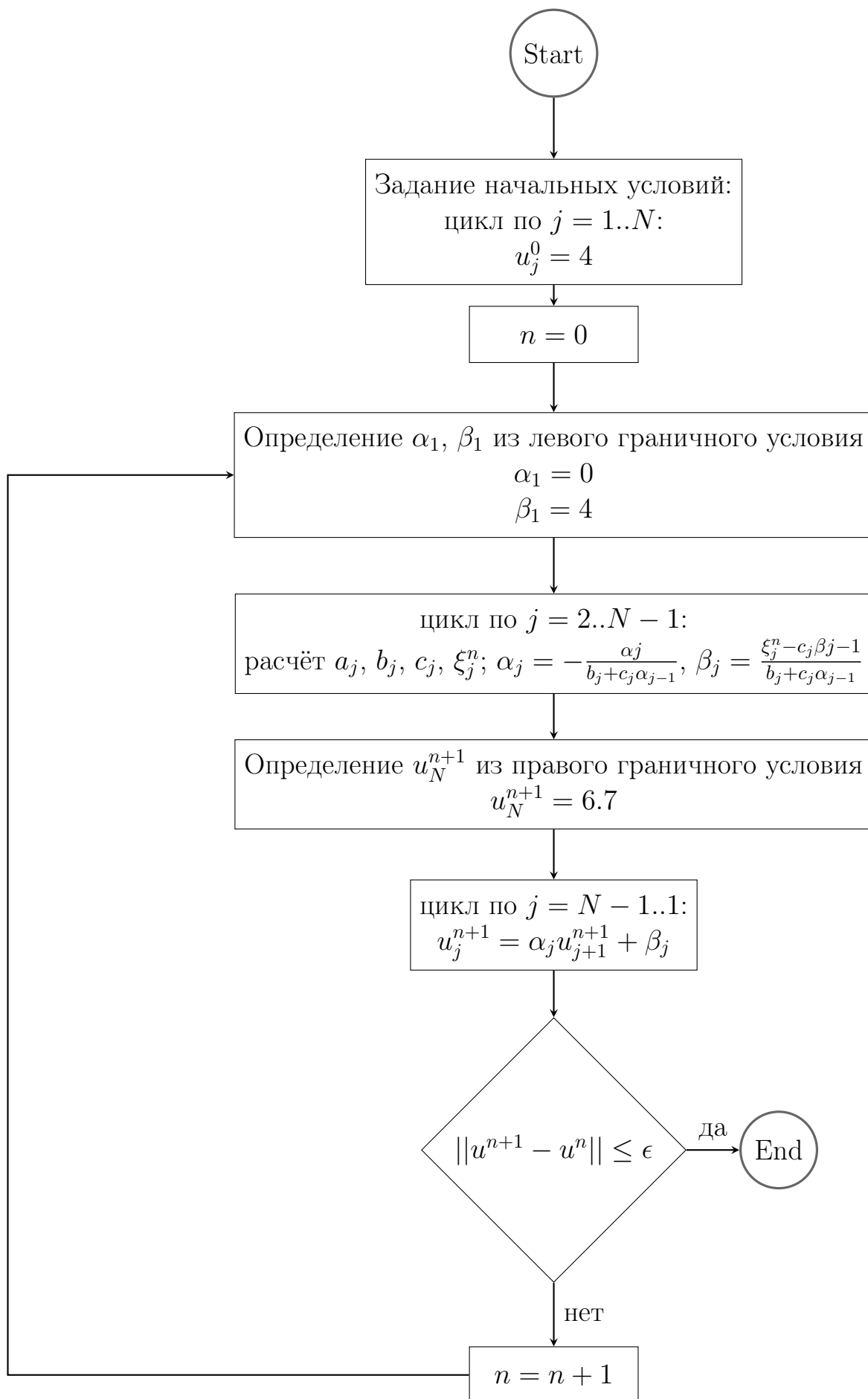
## Задание 6

Записать рекуррентное прогоночное соотношение:

Соотношение (3) является **рекуррентным прогоночным соотношением**.

## Задание 7

Составить алгоритм (блок-схему) расчёта:



## Задание 8

Построить программу на любом удобном языке программирования:

```
1 package com.stateEquation;
2
3 import java.lang.Math;
4
5 public class Main {
6     private static final int x_start = 0;
7     private static final int x_end = 1;
8
9     // private static final int t_start = 0;
10    // private static final int t_end = 1;
11
12    private static final double[] delta_t = { 0.1, 0.01, 0.001 };
13    private static final double[] h = { 0.1, 0.01 };
14
15    private static final double[] xPoints = {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7,
16        0.8, 0.9, 1.0};
17    private static final double epsilon = 1e-6;
18    private static final int maxIterations = 10000;
19
20    public static void main( String[] args ) {
21        for (int index_h = 0; index_h < h.length; index_h++) {
22            int N = (int) (1 + (x_end - x_start) / h[index_h]);
23
24            for (int index_delta_t = 0; index_delta_t < delta_t.length; index_delta_t++) {
25                // int N_t = (int) (1 + (t_end - t_start) / delta_t[index_delta_t]);
26
27                double[][] u = new double[2][(int) N];
28                for (int j = 0; j <= N - 1; j++)
29                    u[0][j] = 4.0;
30
31                // int n = 0;
32                // boolean flag = false;
33                // while (!flag && n < maxIterations) {
34                //     for (int j = 1; j <= N - 2; j++)
35                //         u[1][j] = (u[0][j + 1] + (1 + 1 * h[index_h]) * u[0][j - 1] + Math.pow(h[
36                index_h], 2) * 4) / (1 * h[index_h] + 2 * 1);
37                //     u[1][0] = 1.0;
38                //     u[1][N - 1] = 6.7;
39
40                //     double sum = 0.0;
41                //     for (int j = 0; j < N; j++)
42                //         sum += Math.pow(u[1][j] - u[0][j], 2);
43                //     double norma = Math.sqrt(h[index_h] * sum);
44                //
45                //     if (norma <= epsilon)
46                //         flag = true;
47                //     else {
48                //         double[] temp = u[0];
49                //         u[0] = u[1];
50                //         u[1] = temp;
51                //         n++;
52                //     }
53                // }
54
55                double a = - 0.5 * delta_t[index_delta_t] / (h[index_h] * h[index_h]);
```

```

54     double b = 1.0 + 0.5 * delta_t[index_delta_t] / h[index_h] + delta_t[
index_delta_t] / (h[index_h] * h[index_h]);
55     double c = - (0.5 * delta_t[index_delta_t] / h[index_h] + 0.5 * delta_t[
index_delta_t] / (h[index_h] * h[index_h]));
56
57     int n = 0;
58     boolean flag = false;
59     while (!flag && n < maxIterations) {
60         // for (int n = 0; n < N_t - 1; n++) {
61             double[] alpha = new double[N];
62             double[] beta = new double[N];
63
64             alpha[0] = 0.0;
65             beta[0] = 1.0;
66
67             for (int j = 1; j <= N - 2; j++) {
68                 double xi = u[0][j] + 0.5 * delta_t[index_delta_t] / (h[index_h] * h[
index_h]) * (u[0][j + 1] - 2.0 * u[0][j] + u[0][j - 1]) - 0.5 * delta_t[
index_delta_t] / h[index_h] * (u[0][j] - u[0][j - 1]) + delta_t[index_delta_t] *
4;
69                 alpha[j] = - a / (b + c * alpha[j - 1]);
70                 beta[j] = (xi - c * beta[j - 1]) / (b + c * alpha[j - 1]);
71             }
72
73             u[1][N - 1] = 6.7;
74
75             for (int j = N - 2; j >= 0; j--)
76                 u[1][j] = alpha[j] * u[1][j + 1] + beta[j];
77
78             double sum = 0.0;
79             for (int j = 0; j < N; j++)
80                 sum += Math.pow(u[1][j] - u[0][j], 2);
81             double norma = Math.sqrt(h[index_h] * sum);
82
83             if (norma <= epsilon)
84                 flag = true;
85             else {
86                 double[] temp = u[0];
87                 u[0] = u[1];
88                 u[1] = temp;
89                 n++;
90             }
91         }
92
93         System.out.println("x\tu(x)");
94         for (double x : xPoints) {
95             int j = (int) Math.round(x / h[index_h]);
96             if (j >= N) j = N-1;
97             System.out.printf("%f\t%f\t%.1f\t%.6f\n", delta_t[index_delta_t], h[index_h
], x, u[1][j]);
98         }
99     }
100 }
101 }
102 }

```