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
Принял:	Лебедев Данила Александрович
Дата сдачи:	16.05.2025

Москва
2025

Оглавление

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 α_1, β_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 виду. Для этого в уравнение необходимо добавить фиктивную производную по времени:

$$\frac{du}{dx} = \frac{d^2u}{dx^2} + 4 \to \frac{\partial \tilde{u}}{\partial \tau} + \frac{\partial \tilde{u}}{\partial x} = \frac{\partial^2 \tilde{u}}{\partial x^2} + 4. \tag{1}$$

При этом искомая функция станет уже функцией двух переменных:

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

Задание 2

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

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

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:

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

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 α_1, β_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

$$u_j^{n+1} = \alpha_j u_{j+1}^{n+1} + \beta j, \tag{3}$$

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

Соотношение (3) называют **рекуррентным прогоночным соотношением**, а коэффициенты α_j, β_j - **прогоночными коэффициентами**.

Для определения прогоночных коэффициентов на *1*-м шаге по координате x, использую 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:

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

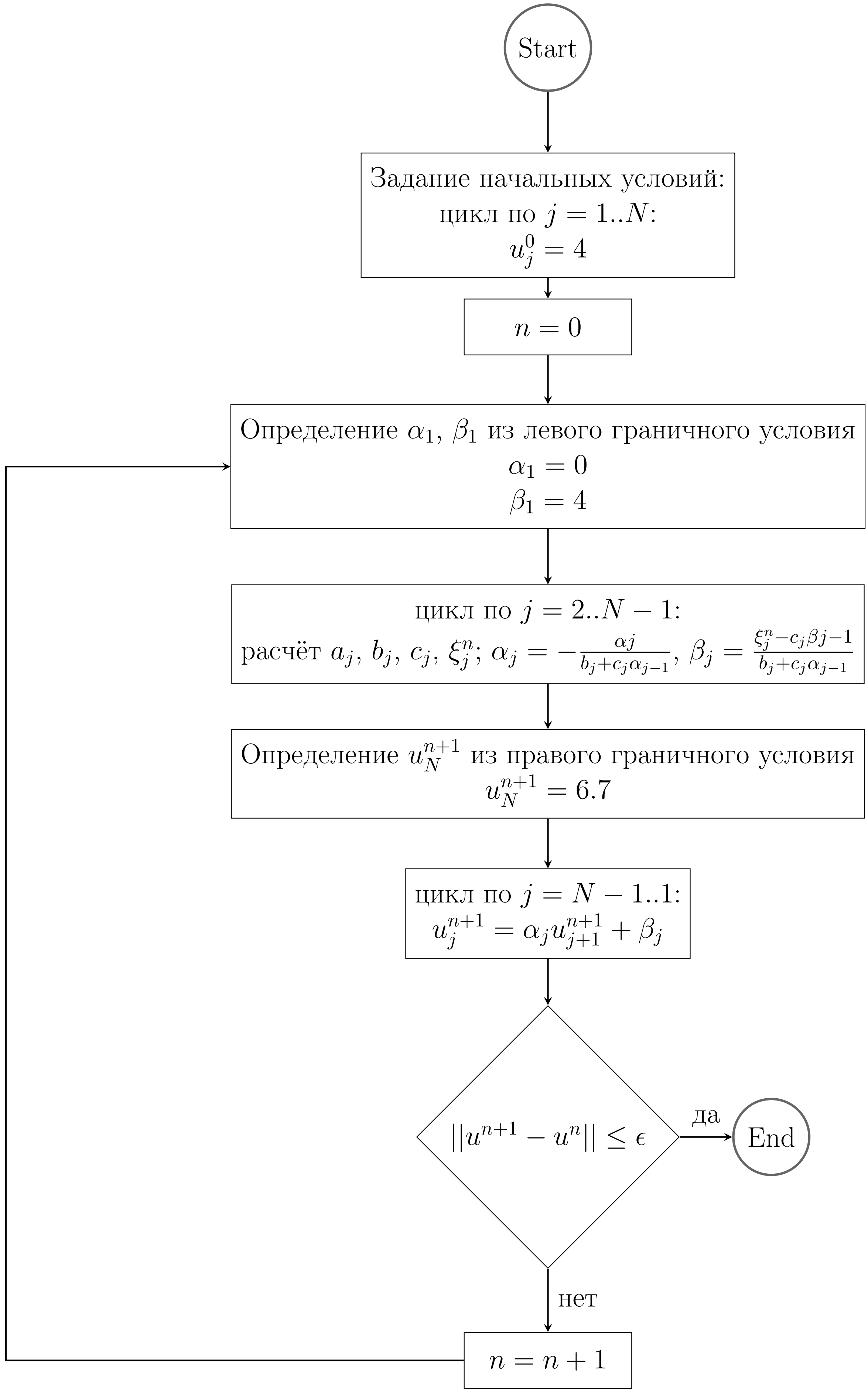
Задание 6

Записать рекуррентное прогоночное соотношение:

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

Задание 7

Составить алгоритм (блок-схему) расчёта:


Start
Задание начальных условий:
цикл по $j = 1..N$:
$u_j^0 = 4$
$n = 0$
Определение α_1, β_1 из левого граничного условия
$\alpha_1 = 0$
$\beta_1 = 4$
цикл по $j = 2..N-1$:
расчёт a_j, b_j, c_j, ξ_j^n; $\alpha_j = -\frac{\alpha j}{b_j + c_j \alpha_{j-1}}$, $\beta_j = \frac{\xi_j^n - c_j \beta j - 1}{b_j + c_j \alpha_{j-1}}$
Определение u_N^{n+1} из правого граничного условия
$u_N^{n+1} = 6.7$
цикл по $j = N-1..1$:
$u_j^{n+1} = \alpha_j u_{j+1}^{n+1} + \beta_j$
$||u^{n+1} - u^n|| \leq \epsilon$
да
End
нет
$n = n + 1$

Задание 8

Построить программу на любом удобном языке программирования:

```java
package com.stateEquation;

import java.lang.Math;

public class Main {
  private static final int x_start = 0;
  private static final int x_end = 1;

  // private static final int t_start = 0;
  // private static final int t_end = 1;

  private static final double[] delta_t = { 0.1, 0.01, 0.001 };
  private static final double[] h = { 0.1, 0.01 };

  private static final double[] xPoints = {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0};
  private static final double epsilon = 1e-6;
  private static final int maxIterations = 10000;

    public static void main( String[] args ) {
    for (int index_h = 0; index_h < h.length; index_h++) {
      int N = (int) (1 + (x_end - x_start) / h[index_h]);

      for (int index_delta_t = 0; index_delta_t < delta_t.length; index_delta_t++) {
        // int N_t = (int) (1 + (t_end - t_start) / delta_t[index_delta_t]);

        double[][] u = new double[2][(int) N];
        for (int j = 0; j <= N - 1; j++)
          u[0][j] = 4.0;

      // int n = 0;
      // boolean flag = false;
      // while (!flag && n < maxIterations) {
      //  for (int j = 1; j <= N - 2; j++)
      //    u[1][j] = (u[0][j + 1] + (1 + 1 * h[index_h]) * u[0][j - 1] + Math.pow(h[index_h], 2) * 4) / (1 * h[index_h] + 2 * 1);
      //  u[1][0] = 1.0;
      //  u[1][N - 1] = 6.7;

      //  double sum = 0.0;
      //  for (int j = 0; j < N; j++)
      //    sum += Math.pow(u[1][j] - u[0][j], 2);
      //  double norma = Math.sqrt(h[index_h] * sum);
      //
      //  if (norma <= epsilon)
      //    flag = true;
      //  else {
      //    double[] temp = u[0];
      //    u[0] = u[1];
      //    u[1] = temp;
      //    n++;
      //  }
      // }

        double a = - 0.5 * delta_t[index_delta_t] / (h[index_h] * h[index_h]);
```

```
        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]);
        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]));

        int n = 0;
        boolean flag = false;
        while (!flag && n < maxIterations) {
        // for (int n = 0; n < N_t - 1; n++) {
          double[] alpha = new double[N];
          double[] beta = new double[N];

          alpha[0] = 0.0;
          beta[0] = 1.0;

          for (int j = 1; j <= N - 2; j++) {
            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]);
            alpha[j] = - a / (b + c * alpha[j - 1]);
            beta[j] = (xi - c * beta[j - 1]) / (b + c * alpha[j - 1]);
          }

          u[1][N - 1] = 6.7;

          for (int j = N - 2; j >= 0; j--)
            u[1][j] = alpha[j] * u[1][j + 1] + beta[j];

          double sum = 0.0;
          for (int j = 0; j < N; j++)
            sum += Math.pow(u[1][j] - u[0][j], 2);
          double norma = Math.sqrt(h[index_h] * sum);

          if (norma <= epsilon)
            flag = true;
          else {
            double[] temp = u[0];
            u[0] = u[1];
            u[1] = temp;
            n++;
          }
        }

        System.out.println("x\tu(x)");
        for (double x : xPoints) {
          int j = (int)(x / h[index_h]);
          if (j >= N) j = N-1;
          System.out.printf("%f\t%f\t%.1f\t%.6f\n", delta_t[index_delta_t], h[index_h], x, u[1][j]);
        }
      }
    }
    }
}
```