

Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа программной инженерии

Лабораторная работа № 2

по дисциплине
«Математические модели»

Выполнил:

Ферапонтов М.В.

Группа:

гр. 3530904/00104

Проверил:

Воскобойников С. П.

Санкт-Петербург
2023

Содержание

1	Вступление	2
1.1	Постановка задачи	2
1.2	Используемое ПО	2
2	Основная часть	3
2.1	Разностная схема	3
2.2	Решение системы ОДУ	5
2.2.1	Явный метод Эйлера	6
2.2.2	Неявный метод Эйлера	7
2.3	Тестирование	7
2.3.1	Стационарное решение	7
2.3.2	Решение стремящееся к стационарному	8
2.3.3	Нестационарное решение	9
3	Заключение	10
3.1	Вывод	10
3.2	Код	11

1 Вступление

1.1 Постановка задачи

Вариант СР. Используя интегро-интерполяционный метод (метод баланса), разработать программу для моделирования нестационарного распределения температуры в полном цилиндре, описываемого математической моделью вида:

$$\frac{\partial u}{\partial t} = \frac{1}{r} \frac{\partial}{\partial r} \left(r k(r, t) \frac{\partial u}{\partial r} \right) - q(r, t) u + f(r, t), \quad r \in [R_L, R_R], \quad t \in [0, T],$$
$$0 < c_1 \leq k(r, t) \leq c_2, \quad 0 \leq q(r, t)$$

Начальное условие:

$$u|_{t=0} = \varphi(r)$$

Граничные условия:

$$k \frac{\partial u}{\partial r} \Big|_{r=R_L} = -\nu_1(t) \qquad -k \frac{\partial u}{\partial r} \Big|_{r=R_R} = -\nu_2(t)$$

1.2 Используемое ПО

1. [Boost library](#) - библиотека для тестирования и других функций
2. [GSL](#) - GNU Scientific Library. Математическая библиотека для C и C++.

2 Основная часть

2.1 Разностная схема

Введем основную сетку, где N - число разбиений.

$$r_0 < r_1 < \dots < r_N, \quad r_i \in [R_L, R_R], \quad r_0 = R_L, \quad r_N = R_R$$

$$h_i = r_i - r_{i-1}, \quad i = 1, 2, \dots, N$$

$$r_{r-0.5} = \frac{r_i - r_{i-1}}{2}, \quad i = 1, 2, \dots, N$$

Введем дополнительную сетку:

$$\bar{h}_i = \begin{cases} \frac{h_i+1}{2}, & i = 0 \\ \frac{h_i+h_{i+1}}{2}, & i = 1, 2, \dots, N-1 \\ \frac{h_i}{2}, & i = N \end{cases}$$

Напишем наше уравнение:

$$\frac{\partial u}{\partial t} = \frac{1}{r} \frac{\partial}{\partial r} \left(rk(r, t) \frac{\partial u}{\partial r} \right) - q(r, t)u + f(r, t)$$

$$r \cdot \frac{\partial u}{\partial t} = \frac{\partial}{\partial r} \left(rk \frac{\partial u}{\partial r} \right) - r \cdot qu + r \cdot f$$

Проинтегрируем наше уравнение:

$$\int_{r_{i-\frac{1}{2}}}^{r_{i+\frac{1}{2}}} r \frac{\partial u}{\partial t} dt = \int_{r_{i-\frac{1}{2}}}^{r_{i+\frac{1}{2}}} \frac{\partial}{\partial r} \left(rk \frac{\partial u}{\partial r} \right) dr - \int_{r_{i-\frac{1}{2}}}^{r_{i+\frac{1}{2}}} rqu dr - \int_{r_{i-\frac{1}{2}}}^{r_{i+\frac{1}{2}}} r f dr$$

Используем формулу левых прямоугольников:

$$\int_{r_{i-\frac{1}{2}}}^{r_{i+\frac{1}{2}}} \varphi(x) dx \approx \bar{h}_i \varphi(x)$$

$$\bar{h}_i r_i \frac{dv_i}{dt} = rk \frac{\partial u}{\partial r} \Big|_{r=r_{i+\frac{1}{2}}} - rk \frac{\partial u}{\partial r} \Big|_{r=r_{i-\frac{1}{2}}} - \bar{h}_i r_i q v_i + \bar{h}_i r_i f_i$$

Используем формулы численного дифференцирования:

$$\begin{aligned} rk \frac{\partial u}{\partial r} \Big|_{r=r_{i+\frac{1}{2}}} &= r_{i+\frac{1}{2}} k_{i+\frac{1}{2}} \frac{v_{i+1} - v_i}{h_{i+1}} \\ rk \frac{\partial u}{\partial r} \Big|_{r=r_{i-\frac{1}{2}}} &= r_{i-\frac{1}{2}} k_{i-\frac{1}{2}} \frac{v_i - v_{i-1}}{h_i} \end{aligned}$$

Тем самым мы получили нашу разностную схему внутри промежутка:

$$\hbar_i r_i \frac{dv_i}{dt} = r_{i+\frac{1}{2}} k_{i+\frac{1}{2}} \frac{v_{i+1} - v_i}{h_{i+1}} - r_{i-\frac{1}{2}} k_{i-\frac{1}{2}} \frac{v_i - v_{i-1}}{h_i} - \hbar_i r_i q v_i + \hbar_i r_i f_i, \quad i = 1, 2, \dots, N-1$$

$$\frac{dv_i}{dt} = \frac{r_{i+\frac{1}{2}} k_{i+\frac{1}{2}}}{\hbar_i r_i h_{i+1}} (v_{i+1} - v_i) - \frac{r_{i-\frac{1}{2}} k_{i-\frac{1}{2}}}{\hbar_i r_i h_i} (v_i - v_{i-1}) - q v_i + f_i, \quad i = 1, 2, \dots, N-1$$

Проведем аппроксимацию граничного условия слева:

$$\int_{r_i}^{r_{i+\frac{1}{2}}} r \frac{\partial u}{\partial t} dt = \int_{r_i}^{r_{i+\frac{1}{2}}} \frac{\partial}{\partial r} \left(r k \frac{\partial u}{\partial r} \right) dr - \int_{r_i}^{r_{i+\frac{1}{2}}} r q u dr - \int_{r_i}^{r_{i+\frac{1}{2}}} r f dr, \quad i = 0$$

$$\hbar_i r_i \frac{dv_i}{dt} = r k \frac{\partial u}{\partial r} \Big|_{r=r_{i+\frac{1}{2}}} - r k \frac{\partial u}{\partial r} \Big|_{r=r_i} - \hbar_i r_i q v_i + \hbar_i r_i f_i, \quad i = 0$$

Граничное условие

$$k \frac{\partial u}{\partial r} \Big|_{r=R_L} = -\nu_1(t)$$

Получаем:

$$\hbar_i r_i \frac{dv_i}{dt} = r_{i+\frac{1}{2}} k_{i+\frac{1}{2}} \frac{v_{i+1} - v_i}{h_{i+1}} + r_i \cdot \nu_1 - \hbar_i r_i q v_i + \hbar_i r_i f_i, \quad i = 0$$

$$\frac{dv_i}{dt} = \frac{r_{i+\frac{1}{2}} k_{i+\frac{1}{2}}}{\hbar_i r_i h_{i+1}} (v_{i+1} - v_i) + \frac{\nu_1}{\hbar_i} - q v_i + f_i, \quad i = 0$$

Проведем аппроксимацию граничного условия справа:

$$\int_{r_{i-\frac{1}{2}}}^{r_i} r \frac{\partial u}{\partial t} dt = \int_{r_{i-\frac{1}{2}}}^{r_i} \frac{\partial}{\partial r} \left(r k \frac{\partial u}{\partial r} \right) dr - \int_{r_{i-\frac{1}{2}}}^{r_i} r q u dr - \int_{r_{i-\frac{1}{2}}}^{r_i} r f dr, \quad i = N$$

$$\hbar_i r_i \frac{dv_i}{dt} = r k \frac{\partial u}{\partial r} \Big|_{r=r_i} - r k \frac{\partial u}{\partial r} \Big|_{r=r_{i-\frac{1}{2}}} - \hbar_i r_i q v_i + \hbar_i r_i f_i, \quad i = N$$

Граничное условие:

$$-k \frac{\partial u}{\partial r} \Big|_{r=R_R} = -\nu_2(t)$$

Получаем:

$$\hbar_i r_i \frac{dv_i}{dt} = r_i \frac{\nu_2}{h_{i+1}} - r_{i-\frac{1}{2}} k_{i-\frac{1}{2}} \frac{v_i - v_{i-1}}{h_i} - \hbar_i r_i q v_i + \hbar_i r_i f_i, \quad i = N$$

$$\frac{dv_i}{dt} = \frac{\nu_2}{\hbar_i} - \frac{r_{i-\frac{1}{2}} k_{i-\frac{1}{2}}}{\hbar_i r_i h_i} (v_i - v_{i-1}) - q v_i + f_i, \quad i = N$$

Введем обозначения:

$$B_1 = \frac{r_{i+\frac{1}{2}}k_{i+\frac{1}{2}}}{\hbar_i r_i h_{i+1}} \quad B_2 = \frac{r_{i-\frac{1}{2}}k_{i-\frac{1}{2}}}{\hbar_i r_i h_i}$$

Запишем полученные уравнения:

$$\begin{cases} \frac{dv_i}{dt} = B_1(v_{i+1} - v_i) + \frac{\nu_1}{\hbar_i} - qv_i + f_i, & i = 0 \\ \frac{dv_i}{dt} = B_1(v_{i+1} - v_i) - B_2(v_i - v_{i-1}) - qv_i + f_i, & i = 1, 2, \dots, N-1 \\ \frac{dv_i}{dt} = \frac{\nu_2}{\hbar_i} - B_2(v_i - v_{i-1}) - qv_i + f_i, & i = N \end{cases}$$

Сгруппируем значения:

$$\begin{cases} \frac{dv_i}{dt} = B_1 v_{i+1} - (B_1 + q_i) v_i + \frac{\nu_1}{\hbar_i} + f_i, & i = 0 \\ \frac{dv_i}{dt} = B_1 v_{i+1} - (B_1 + B_2 + q_i) v_i - B_2 v_{i-1} + f_i, & i = 1, 2, \dots, N-1 \\ \frac{dv_i}{dt} = -(B_2 + q_i) v_i - B_2 v_{i-1} + \frac{\nu_2}{\hbar_i} + f_i, & i = N \end{cases}$$

Мы получили систему:

$$\begin{pmatrix} \frac{dv_0}{dt} \\ \frac{dv_1}{dt} \\ \vdots \\ \vdots \\ \vdots \\ \frac{dv_{n-1}}{dt} \\ \frac{dv_n}{dt} \end{pmatrix} = \begin{pmatrix} c_0 & b_0 & & & & & 0 \\ a_1 & c_1 & b_1 & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & \ddots & \ddots & \ddots \\ & & & & & a_{n-1} & c_{n-1} & b_{n-1} \\ 0 & & & & & & a_n & c_n \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ \vdots \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} + \begin{pmatrix} g_0 \\ g_1 \\ \vdots \\ \vdots \\ \vdots \\ g_{n-1} \\ g_n \end{pmatrix}$$

где:

$$\begin{array}{llll} c_0 = -(B_1 + q_0) & b_0 = B_1 & g_0 = \frac{\nu_1}{\hbar_0} + f_0 \\ a_i = B_2 & c_i = -(B_1 + B_2 + q_i) & b_i = B_1 & g_i = f_i \\ a_N = B_2 & c_N = -(B_2 + q_N) & g_N = \frac{\nu_2}{\hbar_N} + f_N \end{array}$$

2.2 Решение системы ОДУ

Система имеет вид:

$$\frac{dv_i}{dt} = Av + g$$

Введем дискретизацию по времени и проинтегрируем на одном из промежутков:

$$\int_{t_n}^{t_{n+1}} \frac{dv_i}{dt} dt = \int_{t_n}^{t_{n+1}} (Av + g) dt$$

$$v(t_{n+1}) - v(t_n) = \int_{t_n}^{t_{n+1}} (Av + g)dt$$

$$v(t_{n+1}) = v(t_n) + \int_{t_n}^{t_{n+1}} (Av + g)dt$$

Добавим к полученному уравнению начальное условие и тем самым получаем систему:

$$\begin{cases} v(t_{n+1}) = v(t_n) + \int_{t_n}^{t_{n+1}} (Av + g)dt \\ v(t_0) = \varphi(r) \end{cases}$$

Решение задачи сводится к поиску значения интеграла.

2.2.1 Явный метод Эйлера

Мы имеем уравнение:

$$v(t_{n+1}) = v(t_n) + \int_{t_n}^{t_{n+1}} (Av + g)dt$$

Интерполируем интеграл по формуле левых треугольников:

$$\int_a^b f(x)dx \approx f(a)(b - a)$$

Тем самым получаем:

$$v(t_{n+1}) = v(t_n) + (t_{n+1} - t_n)Av(t_n) + (t_{n+1} - t_n)g$$

Введем обозначение:

$$H = (t_{n+1} - t_n)$$

$$v(t_{n+1}) = v(t_n) + HAv(t_n) + Hg$$

$$v(t_{n+1}) = (E + HA)v(t_n) + Hg$$

Явный метод ломанных Эйлера:

$$\begin{cases} v(t_{n+1}) = (E + HA)v(t_n) + Hg \\ v(t_0) = \varphi(r) \end{cases}$$

2.2.2 Неявный метод Эйлера

Теперь проинтерполируем интеграл формулой правых треугольников:

$$\int_a^b f(x)dx \approx f(b)(b-a)$$

Получаем:

$$\begin{aligned}v(t_{n+1}) &= v(t_n) + HA v(t_{n+1}) + Hg \\(E - HA)v(t_{n+1}) &= v(t_n) + Hg\end{aligned}$$

Неявный метод ломанных Эйлера:

$$\begin{cases}(E - HA)v(t_{n+1}) = v(t_n) + Hg \\v(t_0) = \varphi(r)\end{cases}$$

2.3 Тестирование

Погрешность решения находится следующим образом:

$$\max_{t_0, \dots, t_n} = \|\varepsilon(t)\|$$

$$\|\varepsilon(t)\| = \|v(\tilde{t}) - v(t)\|$$

Для всех тестов возьмем одинаковое интервал для r и t .

$$r \in \left[\frac{\pi}{6}, \frac{\pi}{3}\right]$$

$$T \in [0, 1]$$

Для всех тестов возьмем одинаковые значения k и q :

$$k = \frac{\cos(r)}{2} + 3$$

$$q = \frac{\sin(r)}{2} + 2$$

Также для всех тестов возьмем одинаковое число разбиений:

$$N = 16$$

2.3.1 Стационарное решение

$$u(r, t) = r$$

Тогда получаем значение f , как:

$$f(r, t) = \frac{-\cos(r) + r \sin(r) - 6 + r^2 \sin(r) + 4r^2}{2r}$$

Значения ν_1 и ν_2 такие:

$$\nu_1 = \frac{\sqrt{3}}{4} + 3$$

$$\nu_2 = 3.25$$

После выполнения программы получаем следующие значения:

Н	$\ \varepsilon\ $, явный метод	$\ \varepsilon\ $, неявный метод
1e-1	2.25e+22	5.05e-5
1e-2	2.13e+140	5.05e-5
1e-3	1.09e+330	5.05e-5
1e-4	3.57e-10	5.05e-5
1e-5	3.88e-9	5.05e-5
1e-6	3.84e-8	5.05e-5

Таблица 1: Погрешность теста ДУ со стационарным решением

2.3.2 Решение стремящееся к стационарному

$$u(r, t) = 4 + e^{-10t}$$

Тогда получаем значение f , как:

$$f(r, t) = \frac{-16 + 4e^{10t} \sin(r) + \sin(r) + 16e^{10t}}{2e^{10t}}$$

Значения ν_1 и ν_2 такие:

$$\nu_1 = 0$$

$$\nu_2 = 0$$

После выполнения программы получаем следующие значения:

Н	$\ \varepsilon\ $, явный метод	$\ \varepsilon\ $, неявный метод
1e-1	2.96e+14	1.65e-01
1e-2	4.55e+140	1.36e-02
1e-3	4.52e+292	2.94e-03
1e-4	2.76e-05	4.54e-04
1e-5	2.76e-05	2.98e-04
1e-6	2.76e-03	2.44e-04

Таблица 2: Погрешность теста ДУ с решением стремящимся к стационарному

2.3.3 Нестационарное решение

$$u(r, t) = r + t$$

Тогда получаем значение f , как:

$$f(r, t) = 1 - \frac{\cos(r) - r \sin(r) + 6}{2r} + \frac{t \sin(r) + \sin(r)}{2} + 2t + 2r$$

Значения ν_1 и ν_2 такие:

$$\nu_1 = \frac{\sqrt{3}}{4} + 3$$
$$\nu_2 = 3.25$$

После выполнения программы получаем следующие значения:

Н	$\ \varepsilon\ $, явный метод	$\ \varepsilon\ $, неявный метод
1e-1	5.57e+20	3.18e-01
1e-2	1.18e+139	3.3e-02
1e-3	6.48e+300	3.31e-03
1e-4	1.69e-05	3.57e-04
1e-5	1.69e-05	1.29e-05
1e-6	1.69e-05	8.48e-05

Таблица 3: Погрешность теста ДУ с нестационарным решением

Полученные данные дают нам повод убедиться, что явный метод Эйлера неприемлем для решения жестких систем. При небольшом шаге можно заметить "взрыв погрешности".

В свою же очередь неявный метод Эйлера требует решения системы линейных алгебраических уравнений, что требует значительного увеличения числа вычислений.

3 Заключение

3.1 Вывод

Задание выполнено полностью. Были написаны: метод для приближенного решения дифференциального уравнения, явный и неявный метод Эйлера. Все методы были протестированы. Были выявлены недостатки и преимущества явного и неявного метода Эйлера.

3.2 Код

```
1  #include <iostream>
2  #include <iomanip>
3  #include <vector>
4  #include <algorithm>
5  #include <initializer_list>
6
7  #include "utils/balance_utils.hpp"
8  #include "data_table.hpp"
9  #include "euler.hpp"
10
11 int main()
12 {
13     std::size_t N = 16;
14
15     auto data_table = get_data();
16
17     std::cout << std::scientific;
18     std::cout << "forward\n";
19     for(auto&& H : {1e-1, 1e-2, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6})
20     {
21         double t0 = 0;
22         std::cout << std::setprecision(0) << H << " ";
23         std::vector< double > first(N + 1);
24         std::vector< double > final(N + 1);
25
26         for (auto&& data : data_table)
27         {
28             init_solution(N, data, first, t0);
29             forward_euler(N, t0, H, data, first, final);
30
31             double x = 0;
32             for(double t = t0 + H; t < 1; t += H)
33             {
34                 first.swap(final);
35
36                 double eps = forward_euler(N, t, H, data, first, final);
37                 x = (eps > x ? eps : x);
38             }
39
40             std::cout << std::setw(10) << std::setprecision(2) << x << " ";
41         }
42         std::cout << "\n";
43     }
44 }
```

```

45
46     std::cout << "backward\n";
47     for(auto&& H : {1e-1, 1e-2, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6})
48     {
49         double t0 = 0;
50         std::cout << std::setprecision(0) << H << " ";
51
52         std::vector< double > first(N + 1);
53         std::vector< double > final(N + 1);
54
55         for (auto&& data : data_table)
56         {
57             init_solution(N, data, first, t0);
58             backward_euler(N, t0, H, data, first, final);
59
60             double x = 0;
61             for(double t = t0 + H; t < 1; t += H)
62             {
63                 first.swap(final);
64
65                 double eps = backward_euler(N, t, H, data, first, final);
66                 x = (eps > x ? eps : x);
67             }
68
69             std::cout << std::setw(10) << std::setprecision(2) << x << " ";
70         }
71         std::cout << "\n";
72     }
73     return 0;
74 }

```

```

1  #ifndef DATA_TABLE_HPP
2  #define DATA_TABLE_HPP
3
4  #include <cmath>
5  #include <vector>
6  #include "utils/data.hpp"
7
8  std::vector< Data< double > > get_data()
9  {
10     std::vector< Data< double > > data_table =
11     {
12         {
13             M_PI / 6, M_PI / 3,
14             [](double r, double t) -> double { return r; }, //u
15             [](double r, double t) -> double { return std::cos(r) / 2 + 3; }, //k

```

```

16     [](double r, double t) -> double { return std::sin(r) / 2 + 2; }, //q
17     [](double r, double t) -> double {
18         return (-std::cos(r) + r * std::sin(r) - 6 + r * r * std::sin(r) + 4 * r * r) / (2 * r);
19     },
20     [](double t) -> double { return std::sqrt(3) / 4 + 3; },
21     [](double t) -> double { return 3.25; },
22 },
23 };
24
25 return data_table;
26 }
27
28 #endif

```

```

1  #ifndef TMA_HPP
2  #define TMA_HPP
3
4  #include <vector>
5  #include <cstdint>
6  #include <type_traits>
7  #include <cassert>
8
9  template <typename T = double>
10 void tma(
11     const std::vector< T >& a,
12     const std::vector< T >& c,
13     const std::vector< T >& b,
14     const std::vector< T >& r,
15     std::vector< T >& x
16 ) {
17     static_assert(std::is_floating_point< T >::value);
18
19     std::size_t sz = r.size();
20
21     assert(sz == b.size());
22     assert(sz == c.size());
23     assert(sz == x.size());
24     assert(sz == r.size());
25     assert(a[0] == 0);
26     assert(b[sz - 1] == 0);
27
28     std::vector< T > y(sz);
29     std::vector< T > p(sz);
30
31     y[0] = b[0] / c[0];
32     p[0] = r[0] / c[0];

```

```

33
34     for( size_t i = 1; i < sz; i++)
35     {
36         y[i] = b[i] / (c[i] - a[i] * y[i - 1]);
37         p[i] = ( r[i] - a[i] * p[i - 1] ) / ( c[i] - a[i] * y[i - 1] );
38     }
39
40     for (int i = sz - 1; i >= 0; i--)
41     {
42         x[i] = p[i] - y[i] * x[i + 1];
43     }
44 }
45
46 #endif

```

```

1  #ifndef EULER_HPP
2  #define EULER_HPP
3
4  #include <cstdlib>
5  #include <vector>
6  #include <iostream>
7  #include "utils/data.hpp"
8  #include "utils/balance_utils.hpp"
9  #include "utils.hpp"
10 #include "tma.hpp"
11
12 template <typename T = double>
13 double forward_euler(
14     std::size_t N, T t, T H,
15     const Data< T >& data,
16     const std::vector< T >& prev,
17     std::vector< T >& next
18 )
19 {
20     std::vector< T > a(N + 1);
21     std::vector< T > c(N + 1);
22     std::vector< T > b(N + 1);
23     std::vector< T > r(N + 1);
24     std::vector< T > x(N + 1);
25
26     std::vector< T > rez(N + 1);
27
28     init_balance(N, data, a, c, b, r, x, t);
29     init_solution(N, data, x, t + H);
30
31     for(int i = 0; i <= N; i++)

```

```

32     {
33         a[i] *= H;
34         c[i] = 1 + H * c[i];
35         b[i] *= H;
36         r[i] *= H;
37     }
38
39     multiply(a, c, b, prev, next);
40     add(next, r, next);
41     return eps(x, next);
42 }
43
44 template <typename T = double>
45 double backward_euler(
46     std::size_t N, T t, T H,
47     const Data< T >& data,
48     const std::vector< T >& prev,
49     std::vector< T >& next
50 )
51 {
52     std::vector< T > a(N + 1);
53     std::vector< T > c(N + 1);
54     std::vector< T > b(N + 1);
55     std::vector< T > r(N + 1);
56     std::vector< T > x(N + 1);
57
58     std::vector< T > rez(N + 1);
59
60     init_balance(N, data, a, c, b, r, x, t);
61     init_solution(N, data, x, t + H);
62
63     for (int i = 0; i <= N; i++)
64     {
65         a[i] *= -H;
66         c[i] = 1 - H * c[i];
67         b[i] *= -H;
68         r[i] *= H;
69     }
70
71     add(r, prev, r);
72     tma(a, c, b, r, next);
73     return eps(x, next);
74 }
75
76 #endif

```

```

1  #ifndef UTILS_HPP
2  #define UTILS_HPP
3
4  #include <vector>
5  #include <iomanip>
6  #include <cmath>
7
8  #include <boost/range/combine.hpp>
9
10 #include <gsl/gsl_matrix.h>
11 #include <gsl/gsl_linalg.h>
12
13 template< typename T >
14 T eps(const std::vector< T > &v1,
15       const std::vector< T > &v2)
16 {
17     T eps_ = 0;
18
19     for (auto &&tuple: boost::combine(v1, v2))
20     {
21         T x1, x2;
22         boost::tie(x1, x2) = tuple;
23
24         T del = std::fabs(x1 - x2);
25
26         if (del > eps_)
27         {
28             eps_ = del;
29         }
30     }
31
32     return eps_;
33 }
34
35 template< typename T >
36 void add( const std::vector< T >& v1 ,
37          const std::vector< T >& v2 ,
38          std::vector< T >& rez
39 )
40 {
41     std::size_t size = rez.size ();
42     assert(v1.size() == size );
43     assert(v2.size() == size );
44
45     std::vector< double > tmp(size);
46

```

```

47     for( std::size_t i = 0; i < size; i++)
48     {
49         tmp [i] = v1[i] + v2[i];
50     }
51
52     eps(v1, v2);
53     rez.swap(tmp);
54 }
55
56 template< typename T >
57 void multiply(
58     const std::vector< T >& a,
59     const std::vector< T >& c,
60     const std::vector< T >& b,
61     const std::vector< T >& x,
62     std::vector< T >& rez
63 )
64 {
65     std::size_t size = rez.size();
66
67     assert(a.size() == size);
68     assert(c.size() == size);
69     assert(b.size() == size);
70     assert(x.size() == size);
71
72     std::vector< T > tmp(size);
73
74     tmp[0] = c[0] * x[0] + b[0] * x[1];
75
76     for (std::size_t i = 1; i < size - 1; i++)
77     {
78         tmp[i] = a[i] * x[i - 1] + c[i] * x[i] + b[i] * x[i + 1];
79     }
80
81     tmp[size - 1] = a[size - 1] * x[size-2] + b[size - 1] * x[size - 1];
82
83     rez.swap(tmp);
84 }
85
86 template <typename T>
87 T cond(
88     const std::vector< T >& a,
89     const std::vector< T >& c,
90     const std::vector< T >& b
91 )
92 {

```

```

93     std::size_t N = b.size() - 1;
94     double rez = 1;
95
96     gsl_matrix* m = gsl_matrix_alloc(N + 1, N + 1);
97     gsl_matrix* n = gsl_matrix_alloc(N + 1, N + 1);
98
99     gsl_matrix_set_zero(m);
100
101     gsl_matrix_set(m, 0, 0, c[0]);
102     gsl_matrix_set(m, 0, 1, b[0]);
103
104     for (std::size_t i = 1; i < N; i++)
105     {
106         gsl_matrix_set(m, i, i - 1, a[i]);
107         gsl_matrix_set(m, i, i, c[i]);
108         gsl_matrix_set(m, i, i + 1, b[i]);
109     }
110
111     gsl_matrix_set(m, N, N - 1, a[N]);
112     gsl_matrix_set(m, N, N, c[N]);
113
114     rez *= gsl_matrix_norm1(m);
115
116     int s;
117     gsl_permutation* p = gsl_permutation_alloc(c.size());
118
119     gsl_linalg_LU_decomp(m, p, &s);
120     gsl_linalg_LU_invert(m, p, n);
121
122     rez *= gsl_matrix_norm1(n);
123
124     gsl_permutation_free(p);
125     gsl_matrix_free(m);
126     gsl_matrix_free(n);
127
128     return rez;
129 }
130
131
132 #endif

```

```

1  #ifndef BALANCE_UTILS_HPP
2  #define BALANCE_UTILS_HPP
3
4  #include <vector>
5  #include <iostream>

```

```

6  #include "data.hpp"
7  #include "grid.hpp"
8
9  template <typename T = double>
10 void init_balance(
11     std::size_t N,
12     const Data< T >& data,
13     std::vector< T >& a,
14     std::vector< T >& c,
15     std::vector< T >& b,
16     std::vector< T >& r,
17     std::vector< T >& x,
18     T t
19 )
20 {
21     static_assert(std::is_floating_point< T >::value);
22
23     std::size_t size = N + 1;
24     assert(a.size() == size);
25     assert(c.size() == size);
26     assert(b.size() == size);
27     assert(r.size() == size);
28
29     auto nu_1 = data.nu_1;
30     auto nu_2 = data.nu_2;
31
32     auto h_1 = h1< T >(N, data);
33     auto h_2 = h2< T >(N, data);
34
35     auto r_1 = r1< T >(N, data);
36     auto r_2 = r2< T >(N, data);
37
38     auto k_1 = k1< T >(N, data);
39     auto k_2 = k2< T >(N, data);
40
41     auto q_1 = q< T >(N, data);
42     auto f_1 = f< T >(N, data);
43     double B1 = (r_2(0) * k_2(0, t)) / (h_1(1) * r_1(0) * h_2(0));
44     double B2 = 0;
45
46     a[0] = 0;
47     c[0] = -(B1 + q_1(0, t));
48     b[0] = B1;
49     r[0] = nu_1(t) / h_2(0) + f_1(0, t);
50     x[0] = data.u(r_1(0), t);
51

```

```

52  for (int i = 1; i < N; i++)
53  {
54      B1 = (r_2(i) * k_2(i, t)) / (h_1(i + 1) * r_1(i) * h_2(i));
55      B2 = (r_2(i - 1) * k_2(i - 1, t)) / (h_1(i) * r_1(i) * h_2(i));
56
57      a[i] = B2;
58      c[i] = -(B1 + B2 + q_1(i, t));
59      b[i] = B1;
60      r[i] = f_1(i, t);
61      x[i] = data.u(r_1(i), t);
62  }
63
64  B2 = (r_2(N - 1) * k_2(N - 1, t)) / (h_1(N) * r_1(N) * h_2(N));
65
66  a[N] = B2;
67  c[N] = -(B2 + q_1(N, t));
68  b[N] = 0;
69  r[N] = nu_2(t) / h_2(N) + f_1(N, t);
70  x[N] = data.u(r_1(N), t);
71 }
72
73
74 template< typename T >
75 void init_solution(
76     std::size_t N,
77     const Data< T >& data,
78     std::vector< T >& x,
79     T t
80 )
81 {
82     static_assert(std::is_floating_point< T >::value);
83
84     auto r_1 = r1< T >(N, data);
85     for (std::size_t i = 1; i <= N; i++)
86     {
87         x[i - 1] = data.u(r_1(i), t);
88     }
89 }
90 #endif

```

```

1  #ifndef DATA_HPP
2  #define DATA_HPP
3
4  template< typename T = double >
5  using func_t = T (*)(T r, T t);
6

```

```

7  template< typename T = double >
8  using time_func_t = T (*)(T);
9
10 template <typename T = double>
11 struct Data
12 {
13     const T R_L;
14     const T R_R;
15     const func_t< T > u;
16     const func_t< T > k;
17     const func_t< T > q;
18     const func_t< T > f;
19     const time_func_t< T > nu_1;
20     const time_func_t< T > nu_2;
21 };
22
23 #endif

```

```

1  #ifndef GRID_HPP
2  #define GRID_HPP
3
4  #include <cstddef>
5  #include <iostream>
6  #include <cassert>
7  #include "data.hpp"
8
9  template <typename T = double>
10 class h1
11 {
12     public:
13         h1(std::size_t N, const Data< T >& data):
14             N_(N),
15             data_(data)
16         {}
17
18         T operator() (std::size_t i)
19         {
20             assert(i > 0);
21             assert(i <= N_);
22             auto R_L = data_.R_L;
23             auto R_R = data_.R_R;
24
25             return (R_R - R_L) / N_;
26         }
27
28     private:

```

```

29     std::size_t N_;
30     const Data< T >& data_;
31 };
32
33 template <typename T = double>
34 class h2
35 {
36     public:
37     h2(std::size_t N, const Data< T >& data):
38         N_(N),
39         data_(data)
40     {}
41
42     T operator() (std::size_t i)
43     {
44         assert(i >= 0);
45         assert(i <= N_);
46
47         auto h_1 = h1(N_, data_);
48
49         if (i == 0)
50         {
51             return h_1(i + 1) / 2;
52         }
53
54         if (i == N_)
55         {
56             return h_1(i) / 2;
57         }
58
59         return (h_1(i) + h_1(i + 1)) / 2;
60     }
61
62     private:
63     std::size_t N_;
64     const Data< T >& data_;
65 };
66
67 template <typename T = double>
68 class r1
69 {
70     public:
71     r1(std::size_t N, const Data<T>& data):
72         N_(N),
73         data_(data)
74     {}

```

```

75
76     T operator() (std::size_t i)
77     {
78         auto h_1 = h1(N_, data_);
79         auto R_L = data_.R_L;
80
81         auto r = R_L;
82
83         for (int j = 1; j <= i; j++)
84         {
85             r += h_1(j);
86         }
87
88         return r;
89     }
90 private:
91     std::size_t N_;
92     const Data< T >& data_;
93 };
94
95
96 template <typename T = double>
97 class r2
98 {
99     public:
100     r2(std::size_t N, const Data< T >& data):
101         N_(N),
102         data_(data)
103     {}
104
105     T operator() (std::size_t i)
106     {
107         auto h_2 = h2(N_, data_);
108         auto R_L = data_.R_L;
109
110         auto r = R_L;
111
112         for (int j = 0; j <= i; j++)
113         {
114             r += h_2(j);
115         }
116
117         return r;
118     }
119 private:
120     std::size_t N_;

```



```

121     const Data< T >& data_;
122 };
123
124 template <typename T = double>
125 class k1
126 {
127     public:
128     k1(std::size_t N, const Data< T > data):
129         N_(N),
130         data_(data)
131     {}
132
133     T operator() (std::size_t i, T t)
134     {
135         auto r_1 = r1(N_, data_);
136
137         auto r = r_1(i);
138         return data_.k(r, t);
139     }
140
141     private:
142     std::size_t N_;
143     const Data< T >& data_;
144 };
145
146 template <typename T = double>
147 class k2
148 {
149     public:
150     k2(std::size_t N, const Data< T >& data):
151         N_(N),
152         data_(data)
153     {}
154
155     T operator() (std::size_t i, T t)
156     {
157         auto r_2 = r2(N_, data_);
158         auto r = r_2(i);
159         return data_.k(r, t);
160     }
161
162     private:
163     size_t N_;
164     const Data< T >& data_;
165 };
166

```

```

167 template <typename T = double>
168 class q
169 {
170     public:
171         q(std::size_t N, const Data< T >& data):
172             N_(N),
173             data_(data)
174         {}
175
176         T operator() (std::size_t i, T t)
177         {
178             auto r_1 = r1(N_, data_);
179             auto r = r_1(i);
180
181             return data_.q(r, t);
182         }
183
184     private:
185         std::size_t N_;
186         const Data< T >& data_;
187 };
188
189 template <typename T = double>
190 class f
191 {
192     public:
193         f(std::size_t N, const Data< T >& data):
194             N_(N),
195             data_(data)
196         {}
197
198         T operator() (std::size_t i, T t)
199         {
200             auto r_1 = r1(N_, data_);
201             auto r = r_1(i);
202
203             return data_.f(r, t);
204         }
205
206     private:
207         std::size_t N_;
208         const Data< T >& data_;
209 };
210
211 #endif

```
