Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЁТ

По лабораторной работе № 4

По курсу: «Моделирование»

Тема: «Программно-алгоритмическая реализация моделей на основе ОДУ второго порядка с краевыми условиями II и III рода»

Студент: Керимов А. Ш.

Группа: ИУ7-64Б

Оценка (баллы): _____

Преподаватель: Градов В. М.

Москва

2020

Цель работы. Получение навыков разработки алгоритмов решения смешанной краевой задачи при реализации моделей, построенных на квазилинейном уравнении параболического типа.

Исходные данные

1. Задана математическая модель.

   Уравнение для функции $T(x,t)$

$$c(T)\frac{\partial T}{\partial t} = \frac{\partial}{\partial x}\left(k(T)\frac{\partial T}{\partial x}\right) - \frac{2}{R}\alpha(x)T + \frac{2T_0}{R}\alpha(x) \tag{1}$$

   Краевые условия

$$\begin{cases} t = 0, \quad T(x,0) = T_0, \\ x = 0, \quad -k(T(0))\dfrac{\partial T}{\partial x} = F0, \\ x = l, \quad -k(T(l))\dfrac{\partial T}{\partial x} = \alpha_N(T(l) - T_0). \end{cases} \tag{2}$$

   В обозначениях уравнения (14.1) лекции № 14.

$$p(x) = \frac{2}{R}\alpha(x), \quad f(u) \equiv f(x) = \frac{2T_0}{R}\alpha(x). \tag{3}$$

2. Разностная схема с разностным краевым условием при $x = 0$. Получено в Лекции № 14 (14.6), (14.7), и может быть использовано в данной работе. Самостоятельно надо получить интегро-интерполяционным методом разностный аналог краевого условия при $x = l$, точно так же, как это сделано при $x = 0$ (14.7). Для этого надо проинтегрировать на отрезке $[x_{N-\frac{1}{2}}, x_N]$ выписанное выше уравнение (1) и учесть, что поток

$$\widehat{F}_N = \alpha_N(\widehat{y}_N - T_0), \quad \widehat{F}_{N-\frac{1}{2}} = \widehat{\chi}_{N-\frac{1}{2}}\frac{\widehat{y}_{N-1} - \widehat{y}_N}{h}. \tag{4}$$

3. Значения параметров для отладки (все размерности согласованы)

$$\begin{aligned} k(T) &= a_1(b_1 + c_1T^{m_1}) \quad \text{Вт/см K}, \\ c(T) &= a_2 + b_2T^{m_2} - \frac{c_2}{T_2} \quad \text{Вт/см K}, \\ a_1 &= 0{,}0134, \quad b_1 = 1, \quad c_1 = 4{,}35 \cdot 10^{-4}, \quad m_1 = 1, \\ a_2 &= 2{,}049, \quad b_2 = 0{,}563 \cdot 10^{-3}, \quad c_2 = 0{,}528 \cdot 10^5, \quad m_2 = 1, \\ \alpha(x) &= \frac{c}{x - d}, \\ \alpha_0 &= 0{,}05 \quad \text{Вт/см}^2 \text{ K}, \\ \alpha_N &= 0{,}01 \quad \text{Вт/см}^2 \text{ K}, \\ l &= 10 \quad \text{см}, \\ T_0 &= 300 \quad \text{K}, \\ R &= 0{,}5 \quad \text{см}, \\ F_0 &= 50 \quad \text{Вт/см}^2 \text{ (для отладки принять постоянным)}. \end{aligned}$$

Физическое содержание задачи

Постановки задач в данной лабораторной работе и работе № 3 во многом совпадают. Отличия заключаются в следующем:

1. Сформулированная в данной работе математическая модель описывает **нестационарное** температурное поле $T(x,t)$, зависящее от координаты x и менающееся по времени.

2. Свойства материала стрежня привязаны к температуре, т. е. теплоёмкость и коэффициент теплопроводности $c(T)$, $k(T)$ зависят от T, тогда как в работе № 3 $k(x)$ зависит от координаты, а $c = 0$.

3. При $x = 0$ цилиндр нагружается тепловым потоком $F(t)$, в общем случае зависящем от времени, а в работе № 3 поток был постоянный.

Если в настоящей работе задать поток постоянным, т. е. $F(t) = \text{const}$, то будет происходить формирование температурного поля от начальной температуры T_0 до некоторого установившегося (стационарного) распределения $T(x,t)$. Это поле в дальнейшем с течением времени меняться не будет и должно совпасть с температурным распределением $T(x)$, получаемым в лаб. работе № 3, если все параметры задач совпадают, в частности, вместо $k(T)$ надо использовать $k(x)$ из лаб. работы № 3. Это полезный факт для тестирования программы.

Если после разогрева стержня положить поток $F(t) = 0$, то будет происходить остывание, пока температура не выровняется по всей длине и не станет равной T_0.

При произвольной зависимости потока $F(t)$ от времени температурное поле будет как-то сложным образом отслеживать поток.

Замечание. Варьируя параметры задачи, следует обращать внимание на то, что решения, в которых температура превышает примерно 2000K, физического смысла не имеют и практического интереса не представляют.

Ось x направлена вдоль оси цилиндра и начало координат совпадает с левым торчцем стержня. Слева при $x = 0$ цилиндр нагружается тепловым потоком F_0. Стержень обдувается воздухом, температура которого равна T_0. В результате происходит съем тепла с цилиндрической поверхности и поверхности правого торца при $x = l$. Функции $k(x)$, $\alpha(x)$ являются, соответственно, коэффициентами теплопроводности материала стержня и теплоотдачи при обдуве.

Результаты работы

1. **Разностный аналог краевого условия при $x = l$ и его краткий вывод интегро-интерполяционным методом**

   Обозначим $u \equiv T$, $F = -k(u)\dfrac{\partial u}{\partial x}$. Тогда (1), с учётом также (3), примет вид:

   $$c(u)\frac{\partial u}{\partial t} = -\frac{\partial F}{\partial x} - p(x)u + f(u). \tag{5}$$

   Проинтегрируем на отрезке $[x_{N-\frac{1}{2}}, x_N]$:

   $$\int_{x_{N-\frac{1}{2}}}^{x_N} \mathrm{d}x \int_{t_m}^{t_{m+1}} c(u)\frac{\partial u}{\partial t}\,\mathrm{d}t = -\int_{t_m}^{t_{m+1}} \mathrm{d}t \int_{x_{N-\frac{1}{2}}}^{x_N} \frac{\partial F}{\partial x}\,\mathrm{d}x - \int_{x_{N-\frac{1}{2}}}^{x_N} \mathrm{d}x \int_{t_m}^{t_{m+1}} p(x)u\,\mathrm{d}t + \int_{x_{N-\frac{1}{2}}}^{x_N} \mathrm{d}x \int_{t_m}^{t_{m+1}} f(u)\,\mathrm{d}t,$$

   или

   $$\int_{x_{N-\frac{1}{2}}}^{x_N} \widehat{c}(\widehat{u} - u)\,\mathrm{d}x = \int_{t_m}^{t_{m+1}} (F_{N-\frac{1}{2}} - F_N)\,\mathrm{d}t - \int_{x_{N-\frac{1}{2}}}^{x_N} p\widehat{u}\tau\,\mathrm{d}x + \int_{x_{N-\frac{1}{2}}}^{x_N} \widehat{f}\tau\,\mathrm{d}x.$$

Здесь при вычислении внутренних интегралов по t справа применен метод правых прямоугольников.

Вычисляем интегралы. Первый интеграл справа находим методом правых прямоугольников, а остальные — методом трапеций

$$\frac{h}{4}\left[\widehat{c}_N\left(\widehat{y}_N - y_N\right) + \widehat{c}_{N-\frac{1}{2}}\left(\widehat{y}_{N-\frac{1}{2}} - y_{N-\frac{1}{2}}\right)\right] = \\ = -\left(\widehat{F}_N - \widehat{F}_{N-\frac{1}{2}}\right)\tau - \left(p_N\widehat{y}_N + p_{N-\frac{1}{2}}\widehat{y}_{N-\frac{1}{2}}\right)\tau\frac{h}{4} + \left(\widehat{f}_N + \widehat{f}_{N-\frac{1}{2}}\right)\tau\frac{h}{4},$$

подставим (4), учитывая $\widehat{y}_{N-\frac{1}{2}} = \dfrac{\widehat{y}_N + \widehat{y}_{N-1}}{2}$ и $y_{N-\frac{1}{2}} = \dfrac{y_N + y_{N-1}}{2}$

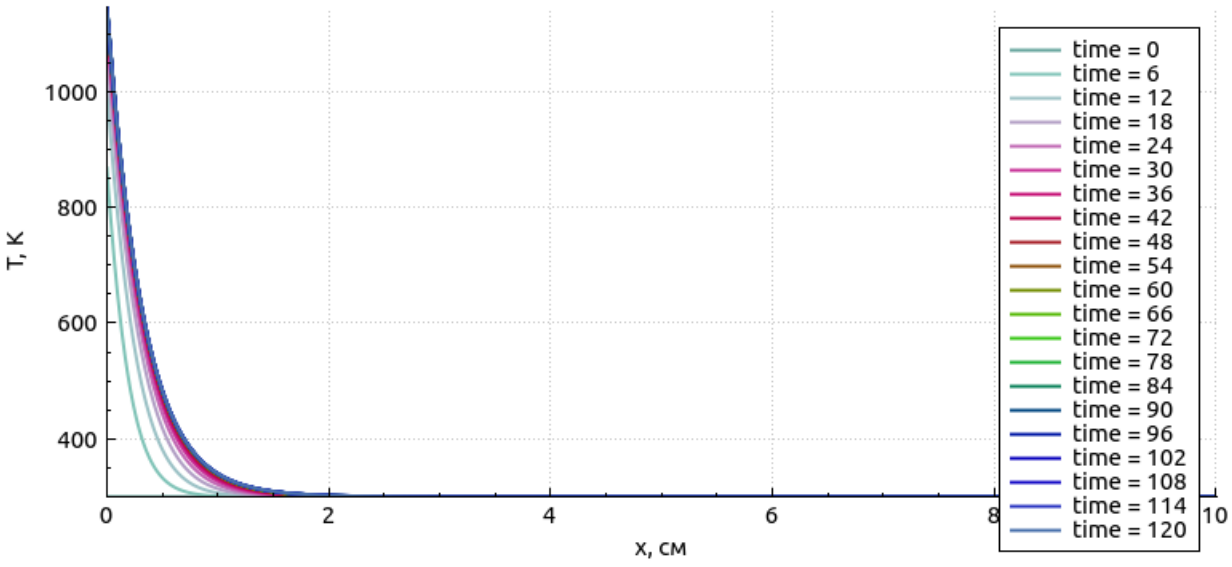
$$\frac{h}{4}\left[\widehat{c}_N\left(\widehat{y}_N - y_N\right) + \widehat{c}_{N-\frac{1}{2}}\left(\frac{\widehat{y}_N + \widehat{y}_{N-1}}{2} - \frac{y_N + y_{N-1}}{2}\right)\right] = \\ = -\left(\alpha_N(\widehat{y}_N - T_0) - \widehat{\chi}_{N-\frac{1}{2}}\frac{\widehat{y}_{N-1} - \widehat{y}_N}{h}\right)\tau - \left(p_N\widehat{y}_N + p_{N-\frac{1}{2}}\frac{\widehat{y}_N + \widehat{y}_{N-1}}{2}\right)\tau\frac{h}{4} + \left(\widehat{f}_N + \widehat{f}_{N-\frac{1}{2}}\right)\tau\frac{h}{4},$$

приведём уравнение к виду $\widehat{K}_N\widehat{y}_N + \widehat{M}_N\widehat{y}_{N-1} = \widehat{P}_N$

$$\left(\widehat{c}_N\frac{h}{4} + \widehat{c}_{N-\frac{1}{2}}\frac{h}{8} + \alpha_N\tau + \widehat{\chi}_{N-\frac{1}{2}}\frac{\tau}{h} + p_N\frac{\tau h}{4} + p_{N-\frac{1}{2}}\frac{\tau h}{8}\right)\widehat{y}_N + \\ + \left(\widehat{c}_{N-\frac{1}{2}}\frac{h}{8} - \widehat{\chi}_{N-\frac{1}{2}}\frac{\tau}{h} + p_{N-\frac{1}{2}}\frac{\tau h}{8}\right)\widehat{y}_{N-1} = \\ = \frac{h}{4}\left(\widehat{c}_N y_N + \widehat{c}_{N-\frac{1}{2}}\frac{y_N + y_{N-1}}{2}\right) + \alpha_N T_0\tau + \left(\widehat{f}_N + \widehat{f}_{N-\frac{1}{2}}\right)\tau\frac{h}{4} \quad (6)$$

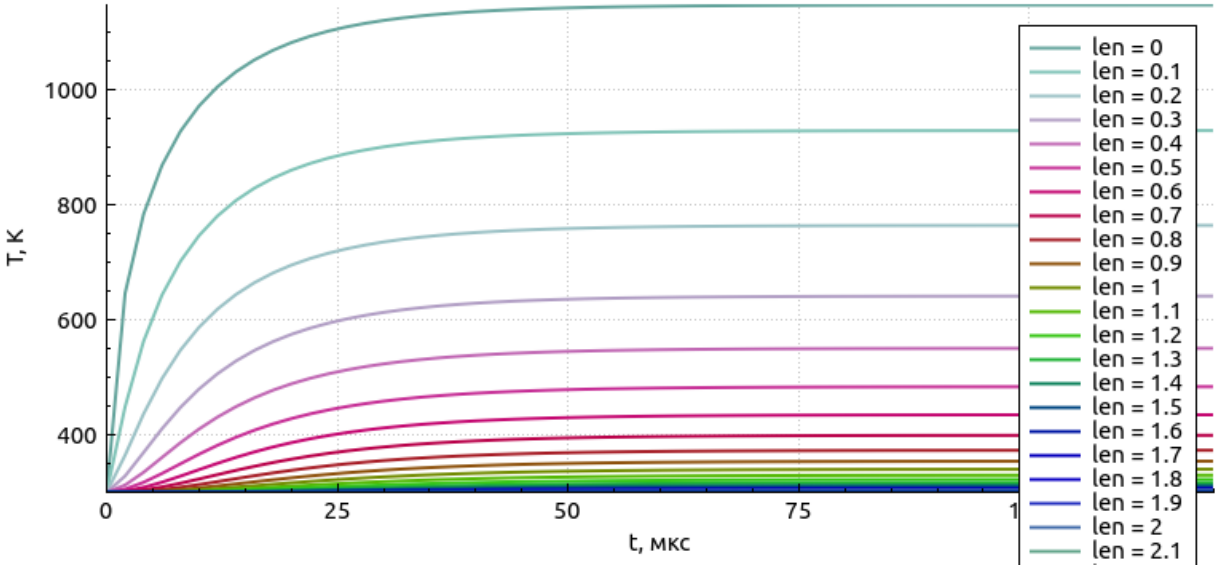
Таким образом, уравнение (6) — разностный аналог краевого условия при $x = l$.

2. **График зависимости $T(x, t_m)$ от координаты x при нескольких фиксированных значениях времени t_m при заданных выше параметрах**



На рисунке представлены графики зависимости температуры от координаты при фиксированных $t = 0,\ 6,\ 12,\ \ldots$

3. **График зависимости $T(x_n, t)$ при нескольких фиксированных значениях координаты x_n**

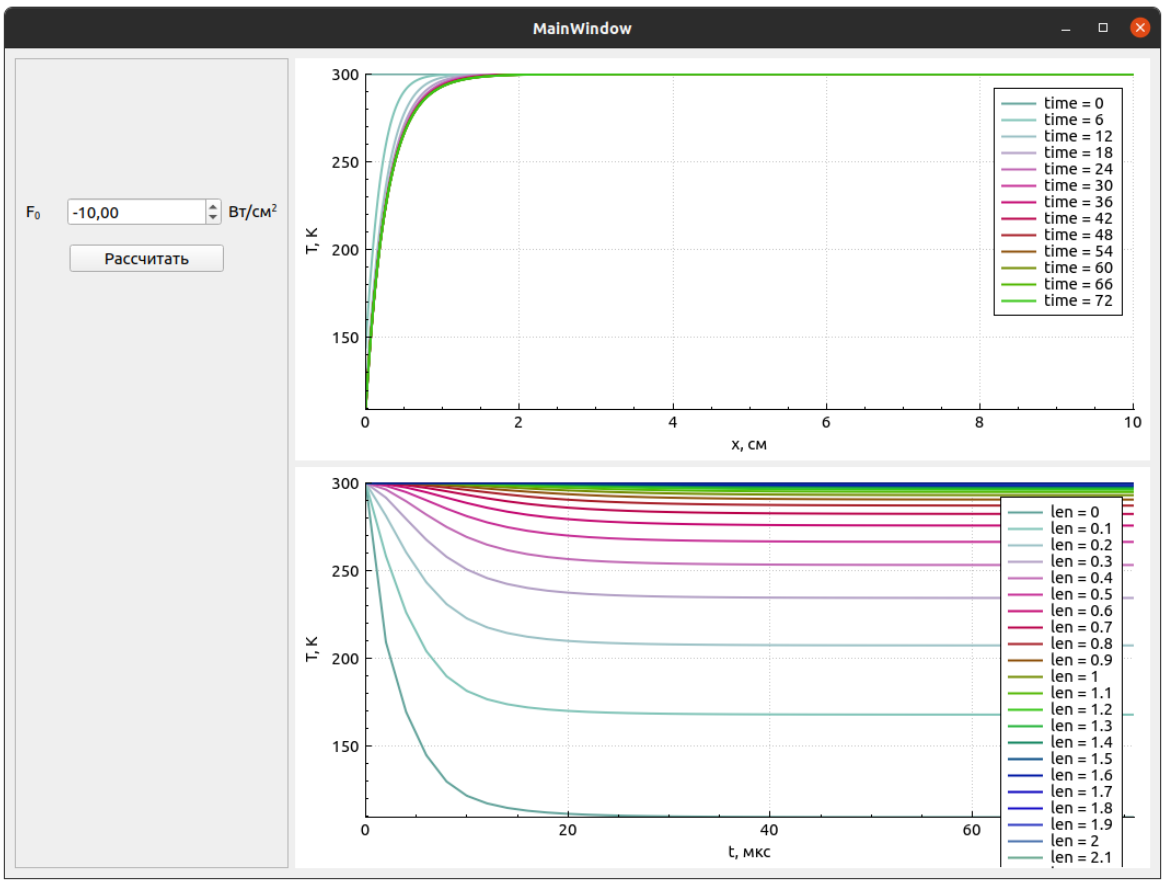


На рисунке представлены графики зависимости температуры от времени при фиксированных $x = 0, 0.1, 0.2, \ldots, 10$.

Вопросы

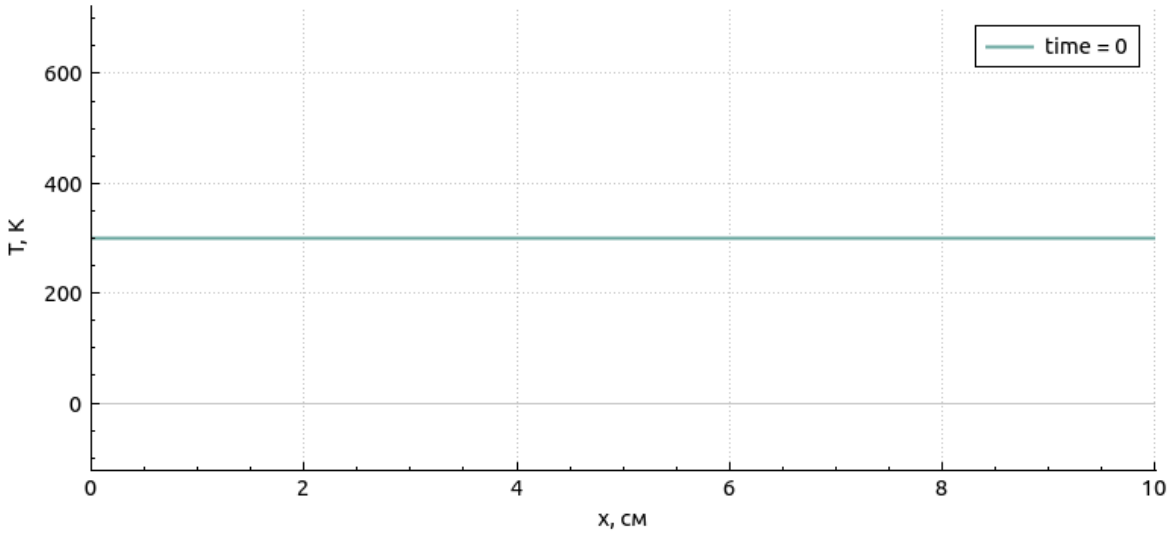
1. **Приведите результаты тестирования программы (графики, общие соображения, качественный анализ). Учесть опыт выполнения лабораторной работы № 3.**

   - Задать отрицательный тепловой поток. Стержень будет охлаждаться с левого торца, а значит $T(x)$ от 0 до l будет увеличиваться.

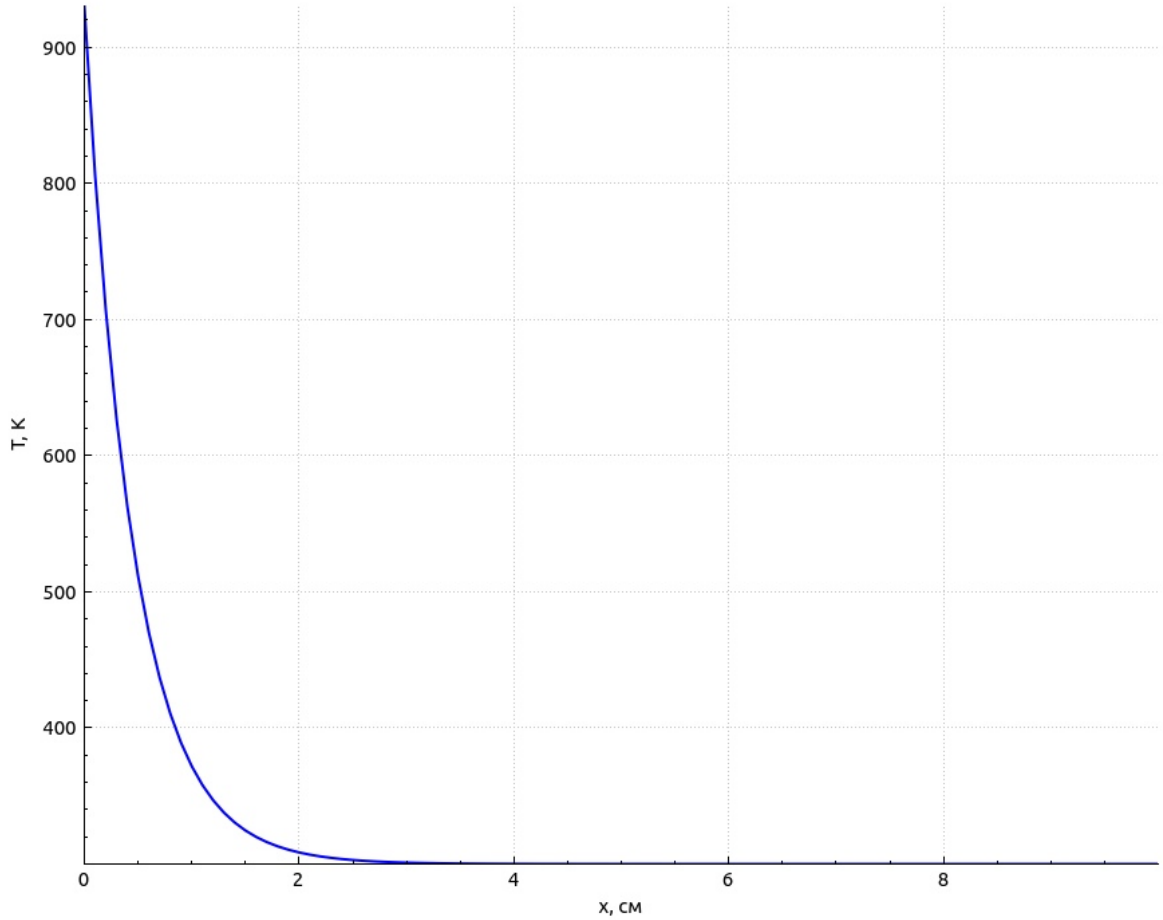


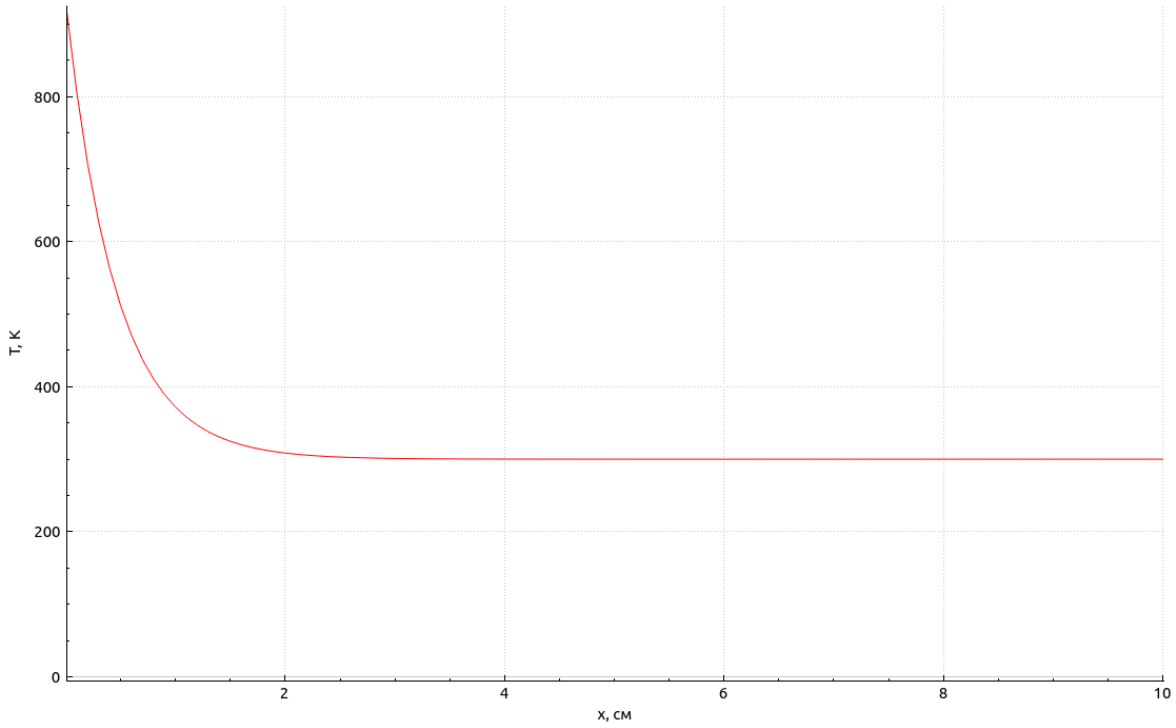
   - При большей теплоотдачи стержня его температура должна снизиться.

- При нулевом тепловом потоке температура стержня должна быть неизменной и равняться температуре окружающей среды.



- Свойства материала стержня привязаны к температуре, теплоемкость и коэффициент теплопроводности зависят от T, поменяем зависимости: установим зависимости T от , а $= 0$. Тогда температурное поле $T(x,t)$ должно совпасть с температурным распределением $T(x)$ из лабораторной 3.





На первом рисунке представлен график из 4 работы с изменёнными параметрами, а на втором — из 3. Видно, что графики совпадают, пиковые температуры на левых торцах одинаковые, на правых — равны температуре окружающей среды. Может показаться, что графики различаются, но это не так. Визуальные различия связаны с разным масштабом координатной плоскости.

2. **Выполните линеаризацию уравнения (14.8) по Ньютону, полагая для простоты, что все коэффициенты зависят только от одной переменной $\widehat{y}_N$. Приведите линеаризованный вариант уравнения и опишите алгоритм его решения. Воспользуйтесь процедурой вывода, описанной в лекции № 8.**

Система квазилинейных разностных уравнений имеет канонический вид

$$\begin{cases} \widehat{K}_0\widehat{y}_0 + \widehat{M}_0\widehat{y}_1 = \widehat{P}_0, \\ \widehat{A}_n\widehat{y}_{n-1} - \widehat{B}_n\widehat{y}_n + \widehat{D}_n\widehat{y}_{n+1} = -\widehat{F}_n, \quad 1 \leqslant n \leqslant N-1, \\ \widehat{K}_N\widehat{y}_N + \widehat{M}_N\widehat{y}_{N-1} = \widehat{P}_N. \end{cases}$$

Учитывая зависимость только от одной переменной $\widehat{y}_N$

$$\widehat{A}_n = \widehat{A}_n(\widehat{y}_n), \quad \widehat{B}_n = \widehat{B}_n(\widehat{y}_n), \quad \widehat{D}_n = \widehat{D}_n(\widehat{y}_n), \quad \widehat{F}_n = \widehat{F}_n(\widehat{y}_n),$$

выполним линеаризацию по Ньютону

$$\left(\widehat{A}_n\widehat{y}_{n-1} - \widehat{B}_n\widehat{y}_n + \widehat{D}_n\widehat{y}_{n+1} + \widehat{F}_n\right)\Big|_{(s-1)} + \widehat{A}_n^{(s-1)}\Delta\widehat{y}_{n-1}^{(s)} +$$
$$+ \left(\frac{\partial\widehat{A}_n}{\partial\widehat{y}_n}\widehat{y}_{n-1} - \frac{\partial\widehat{B}_n}{\partial\widehat{y}_n}\widehat{y}_n - \widehat{B}_n + \frac{\partial\widehat{D}_n}{\partial\widehat{y}_n}\widehat{y}_{n+1} + \frac{\partial\widehat{F}_n}{\partial\widehat{y}_n}\right)\Big|_{(s-1)} \cdot \Delta\widehat{y}_n^{(s)} + \widehat{D}_n^{(s-1)}\Delta\widehat{y}_{n+1}^{(s)} = 0.$$

Тогда

$$A_n\Delta\widehat{y}_{n-1}^{(s)} - B_n\Delta\widehat{y}_n^{(s)} + D_n\Delta\widehat{y}_{n+1}^{(s)} = -F_n$$

$$\begin{cases} A_n = \widehat{A}_n^{(s-1)}, \\ B_n = \left(-\dfrac{\partial \widehat{A}_n}{\partial \widehat{y}_n}\widehat{y}_{n-1} + \dfrac{\partial \widehat{B}_n}{\partial \widehat{y}_n}\widehat{y}_n + \widehat{B}_n - \dfrac{\partial \widehat{D}_n}{\partial \widehat{y}_n}\widehat{y}_{n+1} + \dfrac{\partial \widehat{F}_n}{\partial \widehat{y}_n} \right) \Bigg|_{(s-1)}, \\ D_n = \widehat{D}_n^{(s-1)}, \\ F_n = \left(\widehat{A}_n\widehat{y}_{n-1} - \widehat{B}_n\widehat{y}_n + \widehat{D}_n\widehat{y}_{n+1} + \widehat{F}_n \right) \Big|_{(s-1)}. \end{cases}$$

Краевые условия

$$\begin{cases} \left(\widehat{K}_0\widehat{y}_0 + \widehat{M}_0\widehat{y}_1 - \widehat{P}_0 \right) \Big|_{(s-1)} + \widehat{K}_0^{(s-1)}\Delta\widehat{y}_0^{(s)} + \widehat{M}_0^{(s-1)}\Delta\widehat{y}_1^{(s)} = 0, \\ \left(\widehat{K}_N\widehat{y}_N + \widehat{M}_N\widehat{y}_{N-1} - \widehat{P}_N \right) \Big|_{(s-1)} + \widehat{K}_N^{(s-1)}\Delta\widehat{y}_N^{(s)} + \widehat{M}_N^{(s-1)}\Delta\widehat{y}_{N-1}^{(s)} = 0. \end{cases}$$

Тогда

$$\begin{cases} K_0\Delta\widehat{y}_0^{(s)} + M_0\Delta\widehat{y}_1^{(s)} = P_0, \\ K_N\Delta\widehat{y}_N^{(s)} + M_N\Delta\widehat{y}_{N-1}^{(s)} = P_N, \end{cases}$$

$$\begin{cases} K_0 = \widehat{K}_0^{(s-1)}, \\ M_0 = \widehat{M}_0^{(s-1)}, \\ P_0 = -\left(\widehat{K}_0\widehat{y}_0 + \widehat{M}_0\widehat{y}_1 - \widehat{P}_0 \right) \Big|_{(s-1)}, \\ K_N = \widehat{K}_N^{(s-1)}, \\ M_N = \widehat{M}_N^{(s-1)}, \\ P_N = -\left(\widehat{K}_N\widehat{y}_N + \widehat{M}_N\widehat{y}_{N-1} - \widehat{P}_N \right) \Big|_{(s-1)}. \end{cases}$$

Получаем систему

$$\begin{cases} A_n\Delta\widehat{y}_{n-1}^{(s)} - B_n\Delta\widehat{y}_n^{(s)} + D_n\Delta\widehat{y}_{n+1}^{(s)} = -F_n, \\ K_0\Delta\widehat{y}_0^{(s)} + M_0\Delta\widehat{y}_1^{(s)} = P_0, \\ K_N\Delta\widehat{y}_N^{(s)} + M_N\Delta\widehat{y}_{N-1}^{(s)} = P_N. \end{cases} \tag{7}$$

Решается методом прогонки, в результате находятся все $\Delta\widehat{y}_n^s$, после чего определяются значения искомой функции в узлах $\widehat{y}_n^s = \widehat{y}_n^{(s-1)} + \Delta\widehat{y}_n^s$. Итерационный процесс заканчивается при выполнении условия $\max \left| \dfrac{\Delta\widehat{y}_n^s}{\widehat{y}_n^s} \right| \leqslant \varepsilon$.

Листинг

Листинг 1: `solve.hpp`

```hpp
#ifndef SOLVE_HPP_
#define SOLVE_HPP_

#include <QVector>

class Parameters {
public:
        double F0;
};

class Dependency {
public:
        QVector<double> x;
        QVector<QVector<double>> Tx;
        QVector<double> t;
        QVector<QVector<double>> Tt;
};

Dependency solve(const Parameters& parameters);

#endif  // SOLVE_HPP_
```

Листинг 2: `solve.cpp`

```cpp
#include "solve.hpp"
#include <algorithm>
#include <cmath>

class Solver {
public:
        explicit Solver(const Parameters& parameters) : F0_ (parameters.F0) {}

        Dependency solve() const {
                Dependency result;
                for (double x = 0.0; x <= l_ + h_; x += h_) {
                        result.x.push_back(x);
                }
                QVector<double> T_prev(result.x.size(), T0_);
                QVector<double> T_curr(T_prev);
                result.t.push_back(0.0);
                result.Tx.push_back(T_prev);

                for (double t = tau_; t <= 700; t += tau_) {
                        QVector<double> previous(T_prev);
                        for (;;) {
                                const auto [K0, M0, P0] = left_border(T_curr, T_prev);
                                const auto [KN, MN, PN] = right_border(T_curr, T_prev);

                                QVector<double> A(1), B(1), D(1), F(1);
                                for (auto [x, i] = std::tuple{0.0, 1}; x <= l_; x += h_, ++i) {
                                        auto ip1 = i + 1 < T_curr.size() ? i + 1 : i;
                                        A.push_back(x_nmh(T_curr[i], T_curr[i - 1]) * tau_ / h_);
                                        D.push_back(x_nph(T_curr[i], T_curr[ip1]) * tau_ / h_);
                                        B.push_back(-A.back() - D.back() - c(T_curr[i]) * h_ - p(x) *
                                            h_ * tau_);
                                        F.push_back(-f(x) * h_ * tau_ - c(T_curr[i]) * T_prev[i] * h_);
                                }

                                // forward sweep
                                QVector<double> xi(A.size() + 1);
                                QVector<double> eta(A.size() + 1);
                                xi[1]  = -M0 / K0;
                                eta[1] =  P0 / K0;
                                for (int i = 1; i < A.size(); ++i) {
                                        const double det = B[i] + A[i] * xi[i];
                                        xi[i + 1]  = -D[i] / det;
                                        eta[i + 1] = (F[i] - A[i] * eta[i]) / det;
                                }

                                // backward substitution
                                T_curr.back() = (PN - KN * eta.back()) / (MN + KN * xi.back());
                                for (int i = T_curr.size() - 2; i >= 0; --i) {
```

```
                                        T_curr[i] = xi[i + 1] * T_curr[i + 1] + eta[i + 1];
                                }

                                if (max_diff(T_curr, previous) < eps_) {
                                        result.t.push_back(t);
                                        result.Tx.push_back(T_curr);
                                        break;
                                }

                                qCopy(T_curr.begin(), T_curr.end(), previous.begin());
                        }

                        if (max_diff(T_curr, T_prev) < eps_) {
                                break;
                        }

                        qCopy(T_curr.begin(), T_curr.end(), T_prev.begin());
                }

                const auto step = static_cast<int>(0.1 / (result.x[1] - result.x[0]));

                for (int i = 0; i <= 100; ++i) {
                        result.Tt.push_back({});
                        for (auto &Tx: result.Tx) {
                                result.Tt.back().push_back(Tx[i * step]);
                        }
                }

                return result;
        }

private:
        const double F0_;

        static constexpr double l_   = 10;
        static constexpr double T0_  = 300;
        static constexpr double R_   = 0.5;
        static constexpr double tau_ = 2;

        static constexpr double a1_ = 0.0134;
        static constexpr double b1_ = 1;
        static constexpr double c1_ = 4.35e-4;
        static constexpr double m1_ = 1;

        static constexpr double a2_ = 2.049;
        static constexpr double b2_ = 0.563e-3;
        static constexpr double c2_ = 0.528e+5;
        static constexpr double m2_ = 1;

        static constexpr double alpha0_ = 0.05;
        static constexpr double alphaN_ = 0.01;

        static constexpr double h_ = 1e-2;
        static constexpr double eps_ = 1e-5;

        static double k(double T) {
                return a1_ * (b1_ + c1_ * std::pow(T, m1_));
        }

        static double c(double T) {
                return a2_ + b2_ * std::pow(T, m2_) - c2_ / (T * T);
        }

        static constexpr double alpha(double x) {
                constexpr double delta_alpha = alphaN_ - alpha0_;
                auto __c = -(alphaN_ * alpha0_ * l_) / delta_alpha;
                auto __d = (alphaN_ * l_) / delta_alpha;
                return __c / (x - __d);
        }

        static constexpr double p(double x) {
                return 2 * alpha(x) / R_;
        }

        static constexpr double f(double x) {
                return  2 * alpha(x) * T0_ / R_;
        }

        static double x_nph(double T_n, double T_np1) {
```

```
                return (k(T_n) + k(T_np1)) / 2;
        }

        static double x_nmh(double T_n, double T_nm1) {
                return (k(T_n) + k(T_nm1)) / 2;
        }

        std::tuple<double, double, double> left_border(const QVector<double> &T_curr, const
            QVector<double> &T_prev) const {
                constexpr double p_0 = p(0);
                constexpr double p_h = (p_0 + p(h_)) / 2;
                constexpr double f_0 = f(0);
                constexpr double f_h = (f_0 + f(h_)) / 2;

                const double c_0   = c(T_curr[0]);
                const double c_h   = c((T_curr[0] + T_curr[1]) / 2);
                const double chi_h = (k(T_curr[0]) + k(T_curr[1])) / 2;

                const double K0 = h_ * c_h / 8 + h_ * c_0 / 4 + tau_ * chi_h / h_ + tau_ * h_ * p_h /
                    8 + tau_ * h_ * p_0 / 4;
                const double M0 = h_ * c_h / 8 - tau_ * chi_h / h_ + tau_ * h_ * p_h / 8;
                const double P0 = h_ * c_h / 8 * (T_prev[0] + T_prev[1]) + h_ * c_0 / 4 * T_prev[0] +
                    F0_ * tau_ + tau_ * h_ / 4 * (f_h + f_0);

                return {K0, M0, P0};
        }

        std::tuple<double, double, double> right_border(const QVector<double> &T_curr, const
            QVector<double> &T_prev) const {
                constexpr double p_N   = p(l_);
                constexpr double p_Nmh = (p_N + p(l_ - h_)) / 2;
                constexpr double f_N   = f(l_);
                constexpr double f_Nmh = (f_N + f(l_ - h_)) / 2;

                const int N = T_curr.size() - 1;

                const double c_N     = c(T_curr[N]);
                const double c_Nmh   = c((T_curr[N] + T_curr[N - 1]) / 2);
                const double chi_Nmh = (k(T_curr[N]) + k(T_curr[N - 1])) / 2;

                const double KN = h_ * c_N / 4 + h_ * c_Nmh / 8 + chi_Nmh * tau_ / h_ + alphaN_ * tau_
                    + p_N * tau_ * h_ / 4 + p_Nmh * tau_ * h_ / 8;
                const double MN = h_ * c_Nmh / 8 - chi_Nmh * tau_ / h_ + p_Nmh * tau_ * h_ / 8;
                const double PN = h_ * c_N * T_prev[N] / 4 + h_ * c_Nmh * (T_prev[N] + T_prev[N - 1])
                    / 8 + alphaN_ * T0_ * tau_ + (f_N + f_Nmh) * tau_ * h_ / 4;

                return {KN, MN, PN};
        }

        // flexin'
        template <typename Container>
        static double max_diff(const Container& a, const Container& b) {
                return std::transform_reduce(
                        a.begin(),
                        a.end(),
                        b.begin(),
                        0.0,
                        [](double acc, double cur) { return std::max(acc, cur); },
                        [](double ai, double bi) { return std::abs((ai - bi) / ai); }
                );
        }
};

Dependency solve(const Parameters& parameters) {
        return Solver(parameters).solve();
}
```

Листинг 3: `mainwindow.hpp`

```
#ifndef MAINWINDOW_HPP_
#define MAINWINDOW_HPP_

#include <array>
#include <QMainWindow>
#include "qcustomplot.h"

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE
```

```

class MainWindow : public QMainWindow
{
        Q_OBJECT

public:
        MainWindow(QWidget *parent = nullptr);
        ~MainWindow();

private slots:
        void on_calculatePushButton_clicked();

private:
        Ui::MainWindow *ui;
        std::array<QCustomPlot*, 2> customPlots;
};

#endif  // MAINWINDOW_HPP_
```

Листинг 4: `mainwindow.cpp`

```
#include "mainwindow.hpp"
#include "./ui_mainwindow.h"
#include "solve.hpp"

MainWindow::MainWindow(QWidget *parent):
                QMainWindow(parent),
                ui(new Ui::MainWindow) {
        ui->setupUi(this);

        customPlots = {
                ui->Tx_customPlot,
                ui->Tt_customPlot,
        };

        for (auto&& customPlot: customPlots) {
                customPlot->yAxis->setLabel("T, K");
                customPlot->setInteractions(QCP::iRangeDrag | QCP::iRangeZoom |
                    QCP::iSelectPlottables);
                customPlot->legend->setVisible(true);
                customPlot->legend->setRowSpacing(-3);
        }

        ui->Tx_customPlot->xAxis->setLabel("x, см");
        ui->Tt_customPlot->xAxis->setLabel("t, мкс");
}

MainWindow::~MainWindow() {
        delete ui;
}

static QPen getPen(int i) {
        const auto r = static_cast<int>(qSin(i * 0.3)       * 100) + 100;
        const auto g = static_cast<int>(qSin(i * 0.6 + 0.7) * 100) + 100;
        const auto b = static_cast<int>(qSin(i * 0.4 + 0.6) * 100) + 100;
        QPen pen(QColor(r, g, b));
        pen.setWidth(2);
        return pen;
}

void MainWindow::on_calculatePushButton_clicked() {
        const Parameters parameters = {
                ui->F0_doubleSpinBox->value(),
        };

        auto result = solve(parameters);

        for (auto&& customPlot: customPlots) {
                customPlot->clearGraphs();
        }

        for (int i = 0, j = 0; i < result.Tx.size(); i += 3, ++j) {
                ui->Tx_customPlot->addGraph();
                ui->Tx_customPlot->graph()->setPen(getPen(j));
                ui->Tx_customPlot->graph()->setName(QString("time = ") + QString::number(result.t[i]));
                ui->Tx_customPlot->graph()->setData(result.x, result.Tx[i]);
        }

        for (int i = 0; i < result.Tt.size(); ++i) {
```

```
                ui->Tt_customPlot->addGraph();
                ui->Tt_customPlot->graph()->setPen(getPen(i));
                ui->Tt_customPlot->graph()->setName(QString("len = ") + QString::number(i * 0.1));
                ui->Tt_customPlot->graph()->setData(result.t, result.Tt[i]);
        }

        for (auto&& customPlot: customPlots) {
                customPlot->rescaleAxes();
                customPlot->replot();
        }
}
```

Листинг 5: `main.cpp`

```
#include "mainwindow.hpp"

#include <QApplication>

int main(int argc, char *argv[]) {
        QApplication a(argc, argv);
        MainWindow w;
        w.show();
        return a.exec();
}
```

Листинг 6: `CMakeLists.txt`

```
cmake_minimum_required(VERSION 3.15)
project(lab04 CXX)

set(CMAKE_INCLUDE_CURRENT_DIR ON)

set(CMAKE_AUTOUIC ON)
set(CMAKE_AUTOMOC ON)
set(CMAKE_AUTORCC ON)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

find_package(Qt5 COMPONENTS Widgets REQUIRED)
find_package(Qt5 COMPONENTS PrintSupport REQUIRED)

add_compile_options(
        -Werror

        -Wall
        -Wextra
        -Wpedantic

        -Wcast-align
        -Wcast-qual
        -Wconversion
        -Wctor-dtor-privacy
        -Wenum-compare
        -Wfloat-equal
        -Wnon-virtual-dtor
        -Wold-style-cast
        -Woverloaded-virtual
        -Wredundant-decls
        -Wsign-conversion
        -Wsign-promo
)

include_directories(../qcustomplot)

set(
        HEADERS
        ../qcustomplot/qcustomplot.h
        mainwindow.hpp
        solve.hpp
)

set(
        SOURCES
        ../qcustomplot/qcustomplot.cpp
        mainwindow.cpp
        solve.cpp
)
```

```
52
53 set(
54         FORMS
55         mainwindow.ui
56 )
57
58 add_executable(${PROJECT_NAME} main.cpp ${HEADERS} ${SOURCES} ${FORMS})
59
60 target_link_libraries(${PROJECT_NAME} PRIVATE Qt5::Widgets)
61 target_link_libraries(${PROJECT_NAME} PRIVATE Qt5::PrintSupport)
```